Redpanda 21.10.1

- A Jepsen Report -

Kyle Kingsbury

Distsys Safety Tester

In a nutshell

3 liveness (crash, delayed messages)

7 safety (dups, last write, split-brain, aborted read, G1c)

2 ambiguous (G0, G1c, indefinite error code)

Redpanda

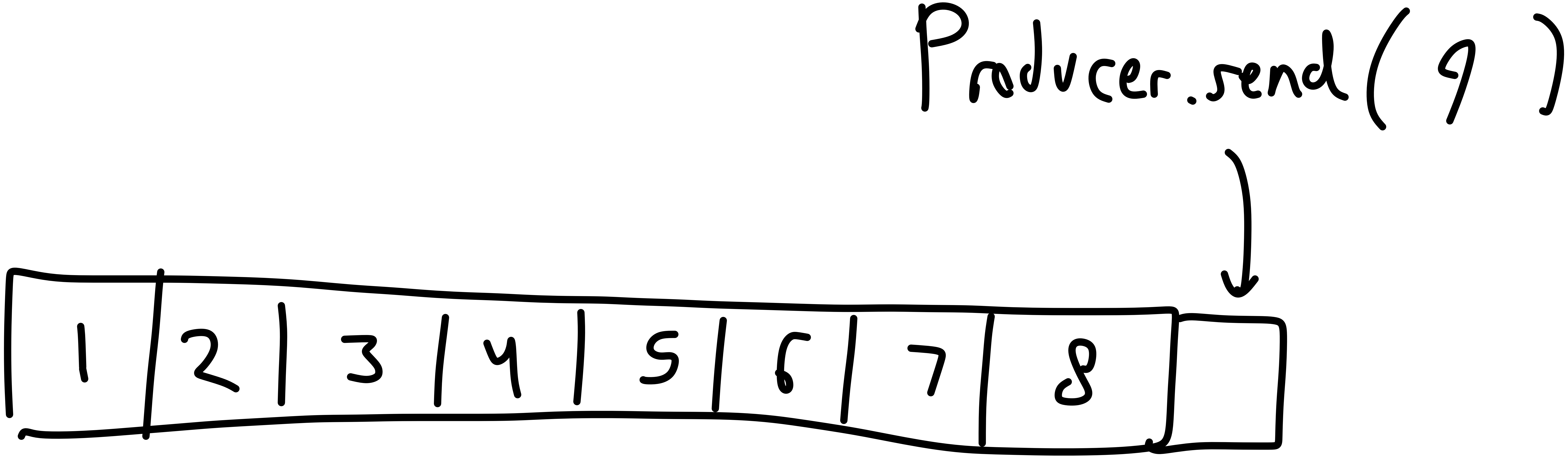
Fundamentals

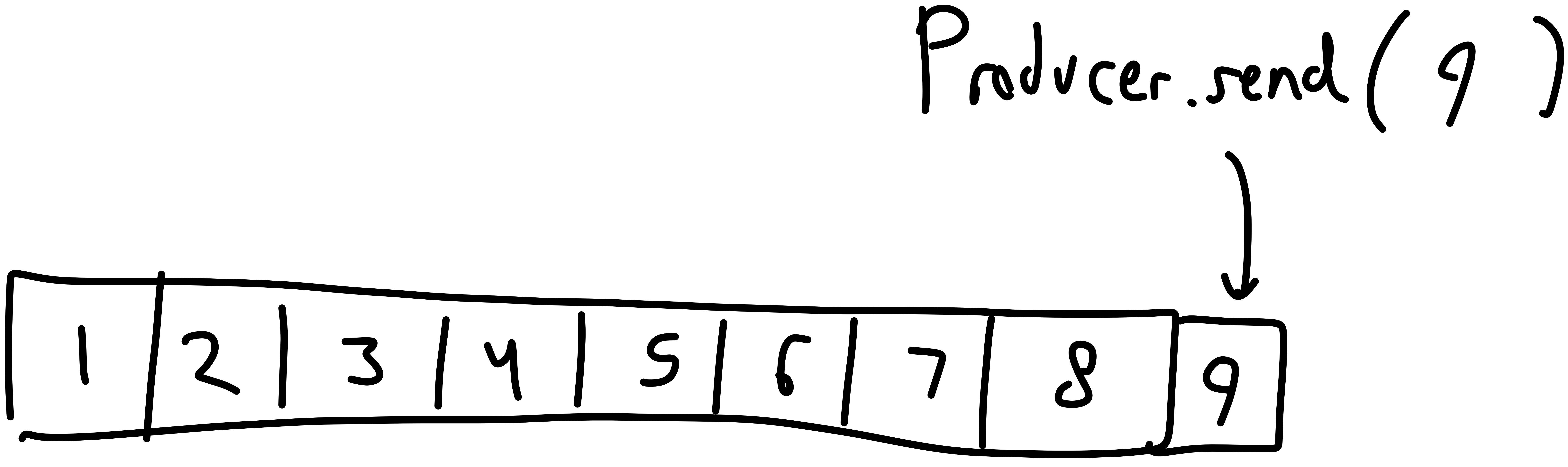
Distributed Log, ala Kafka

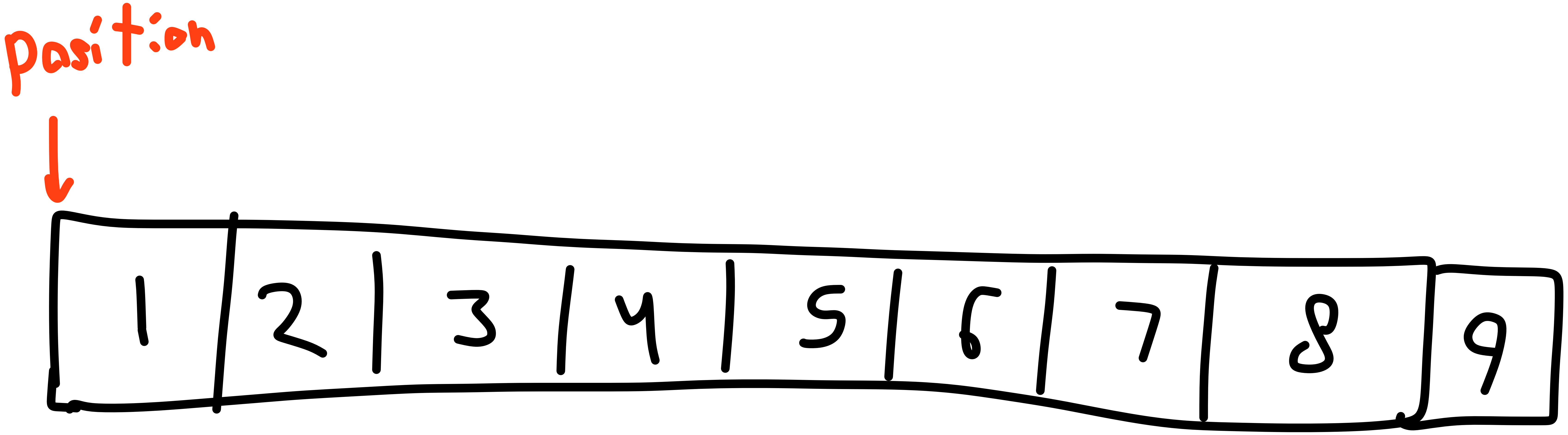
- API compatible
- Aiming for better performance
- No Zookeeper

A Partition

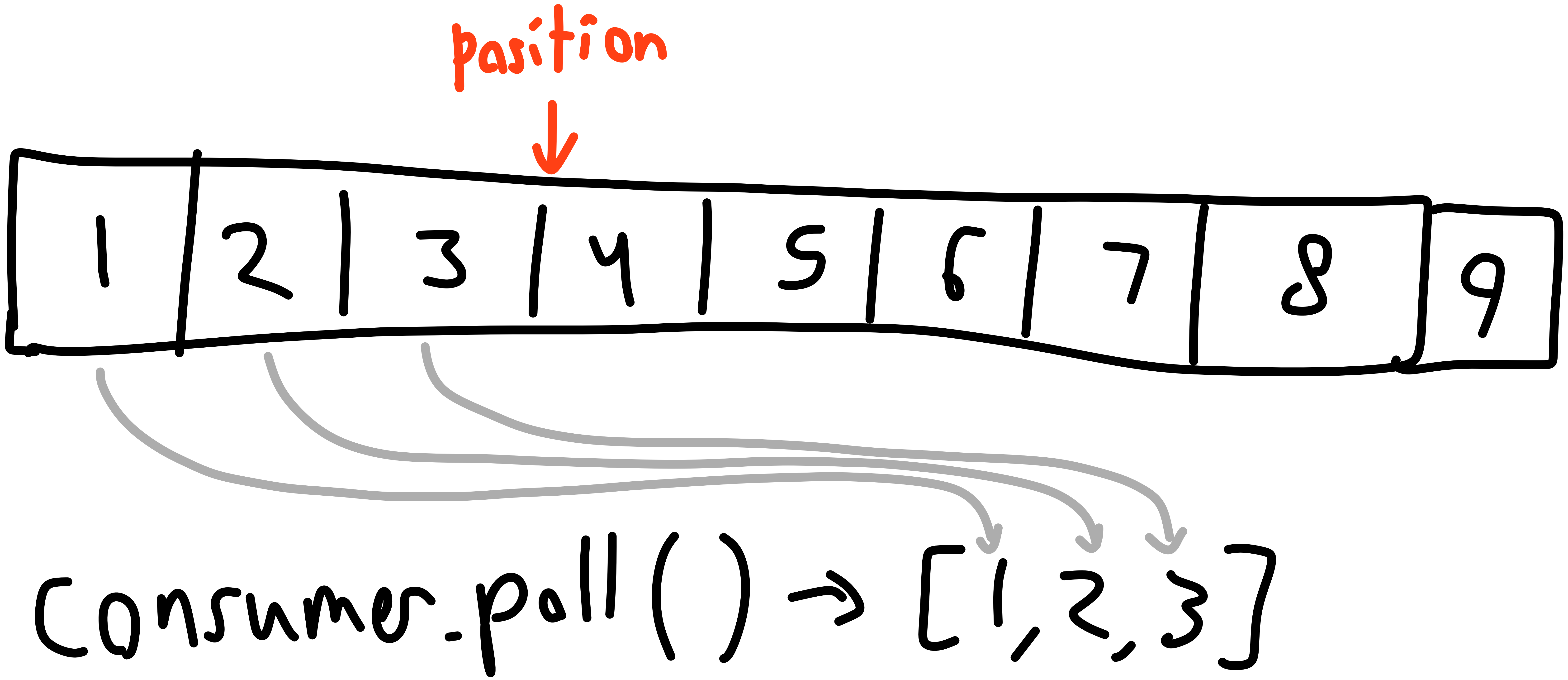
1	2	3	4	5	6	7	8	

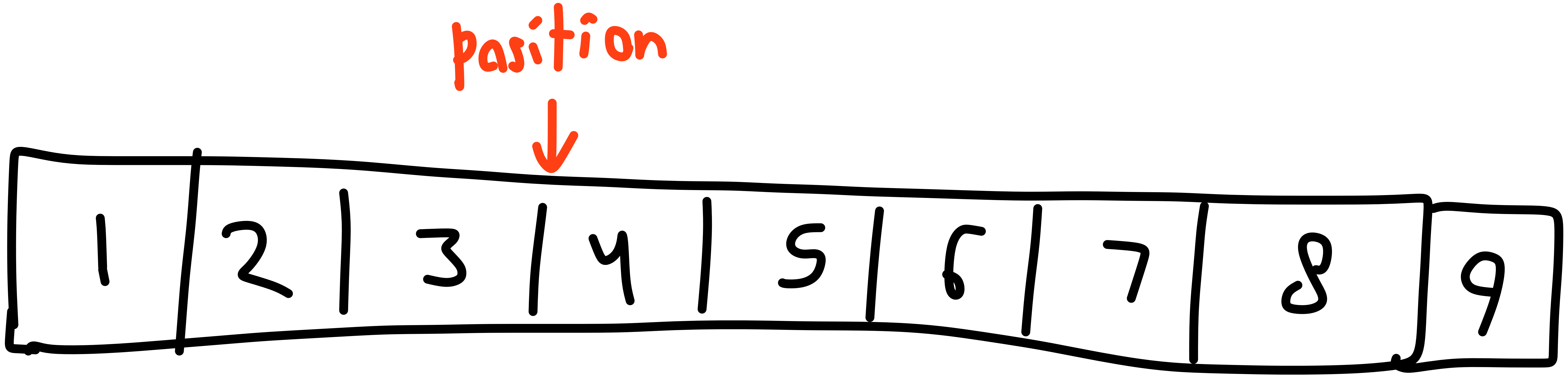

Producer.send(9)
1
2
3
4
5
6
7
8


Producer.send(9)
1
2
3
4
5
6
7
8
9

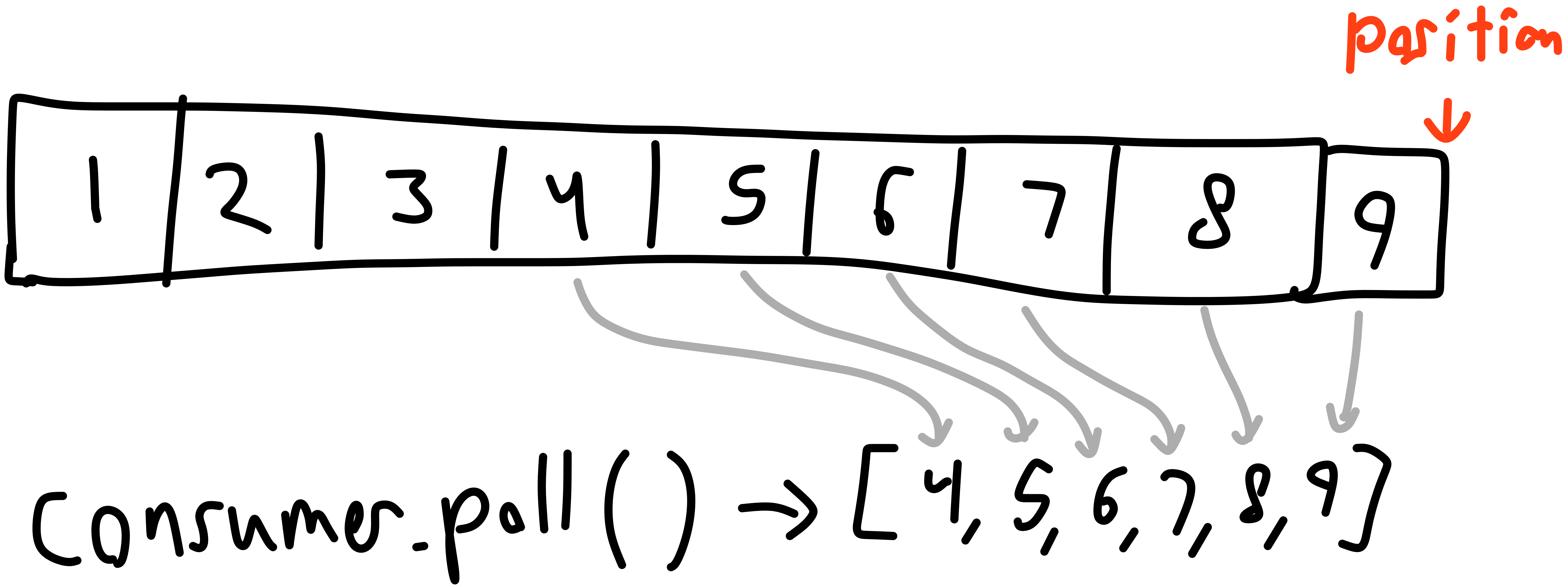


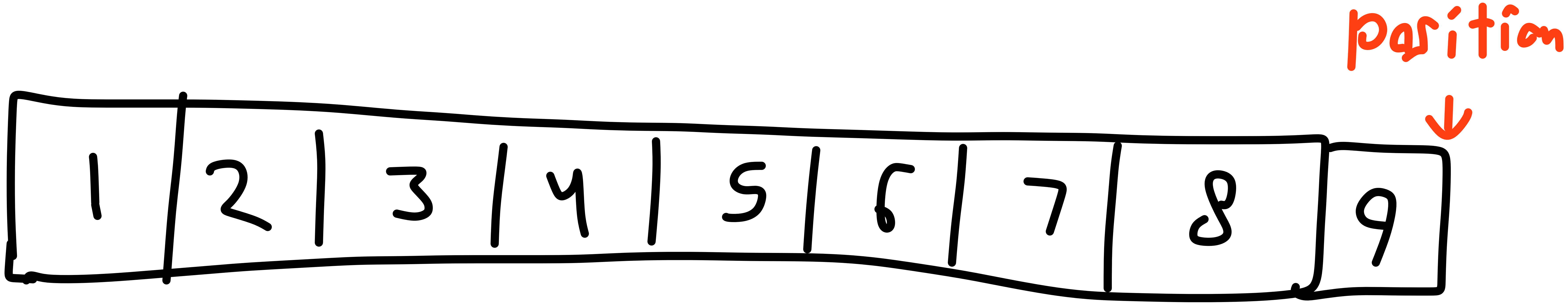
consumer.poll()


position
1
2
3
4
5
6
7
8
9
consumer.poll() -> [1,2,3]



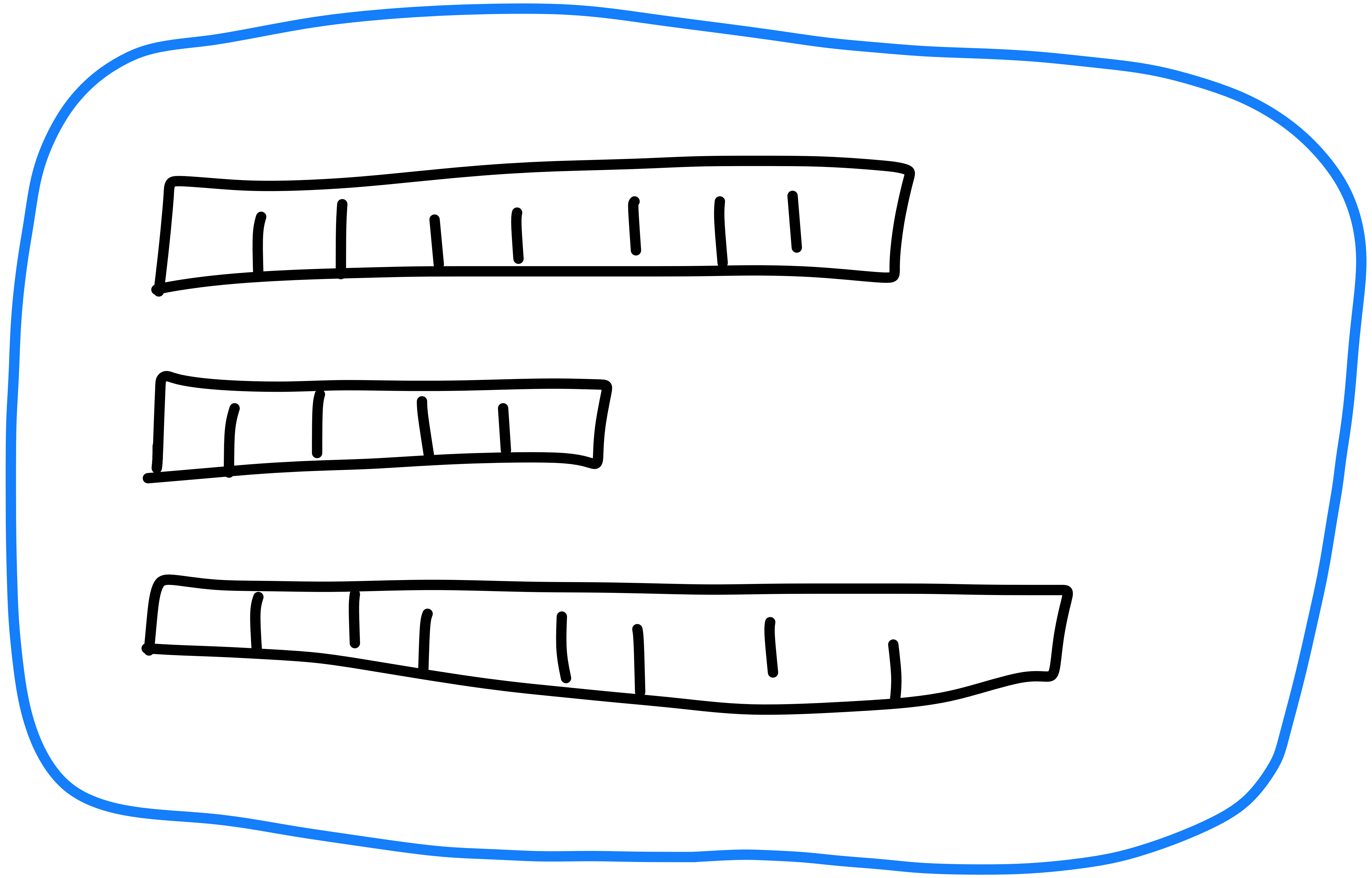
consumer.poll()


position
1
2
3
4
5
6
7
8
9
consumer.poll() -> [4, 5, 6, 7, 8, 9]

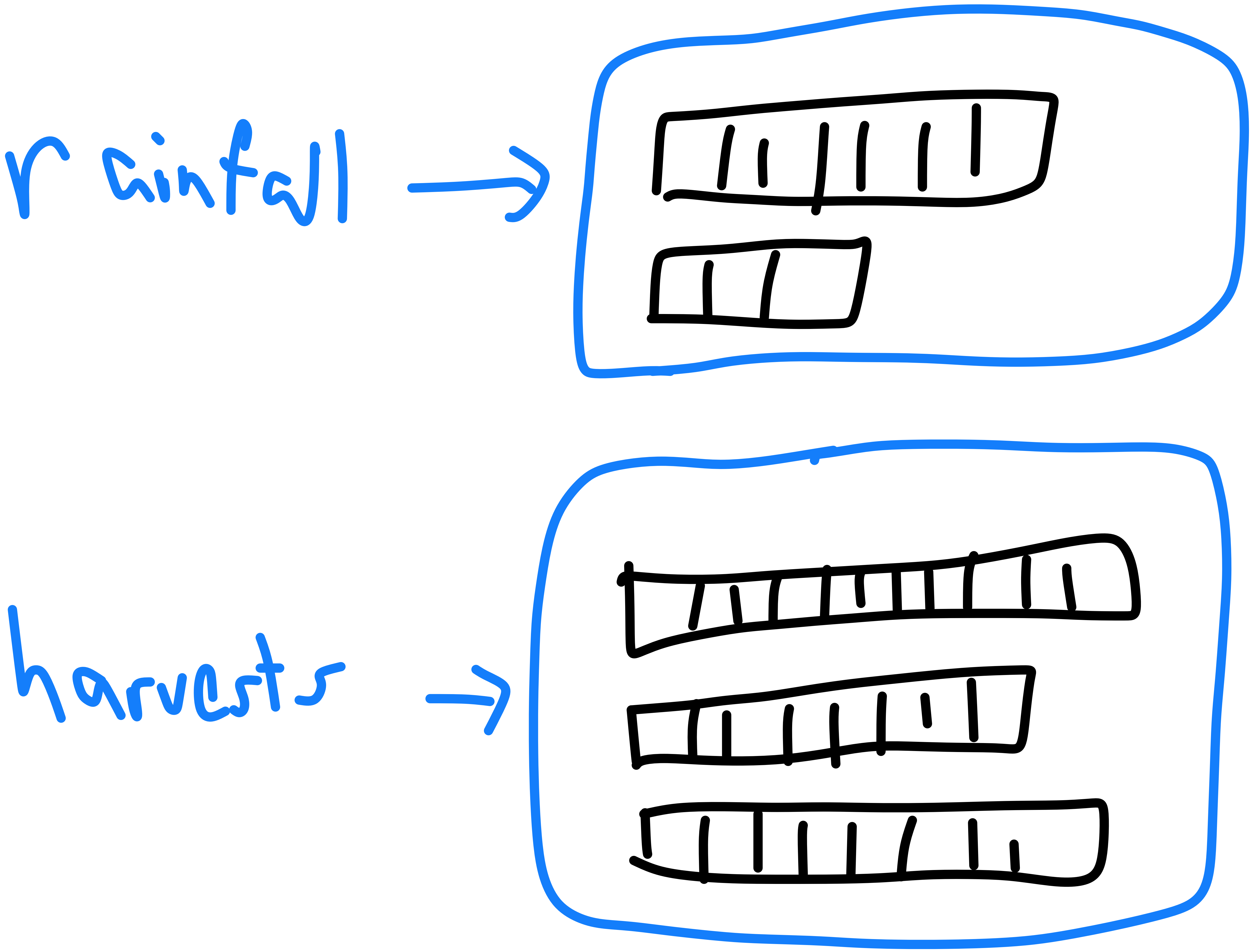


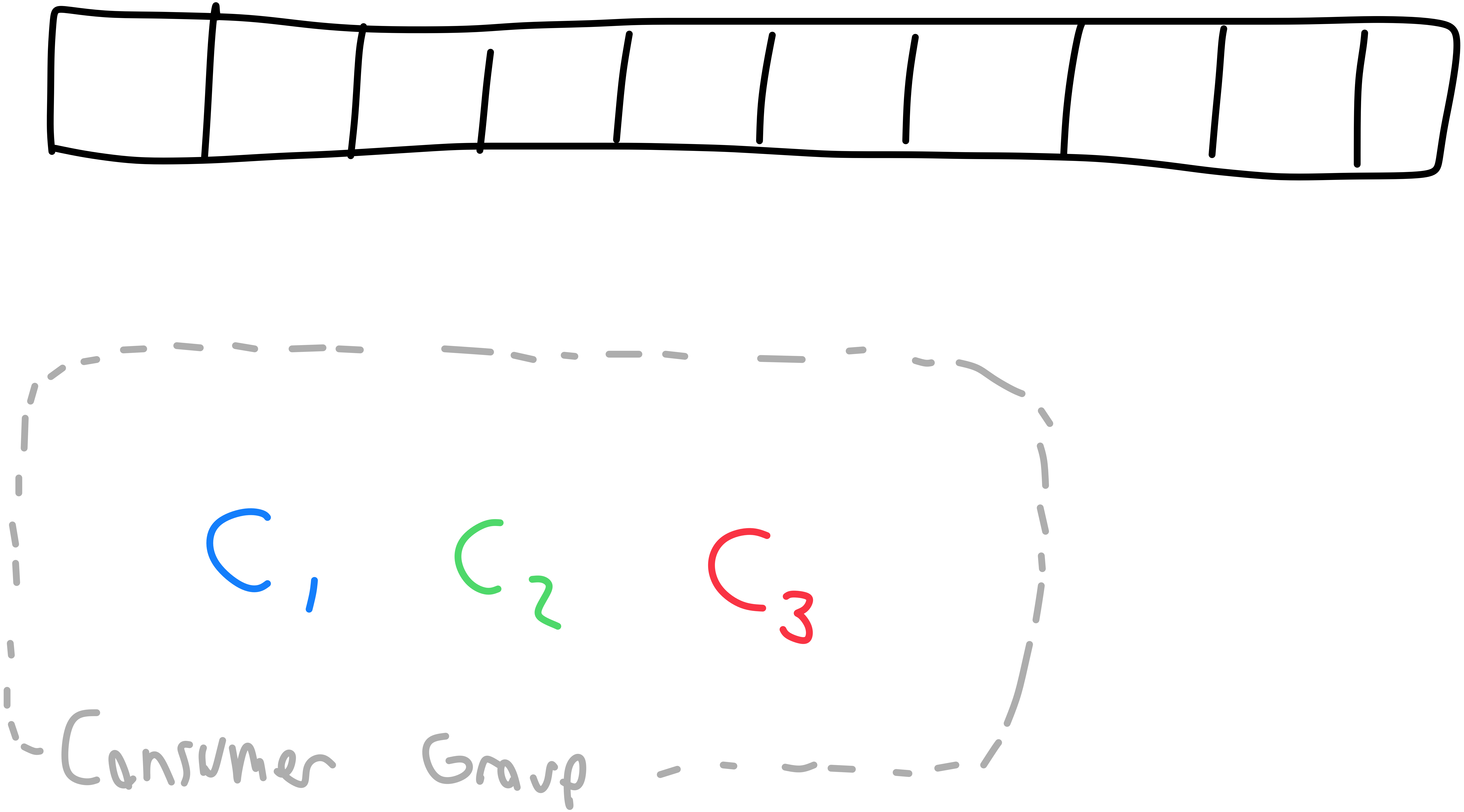
consumer.poll() → []

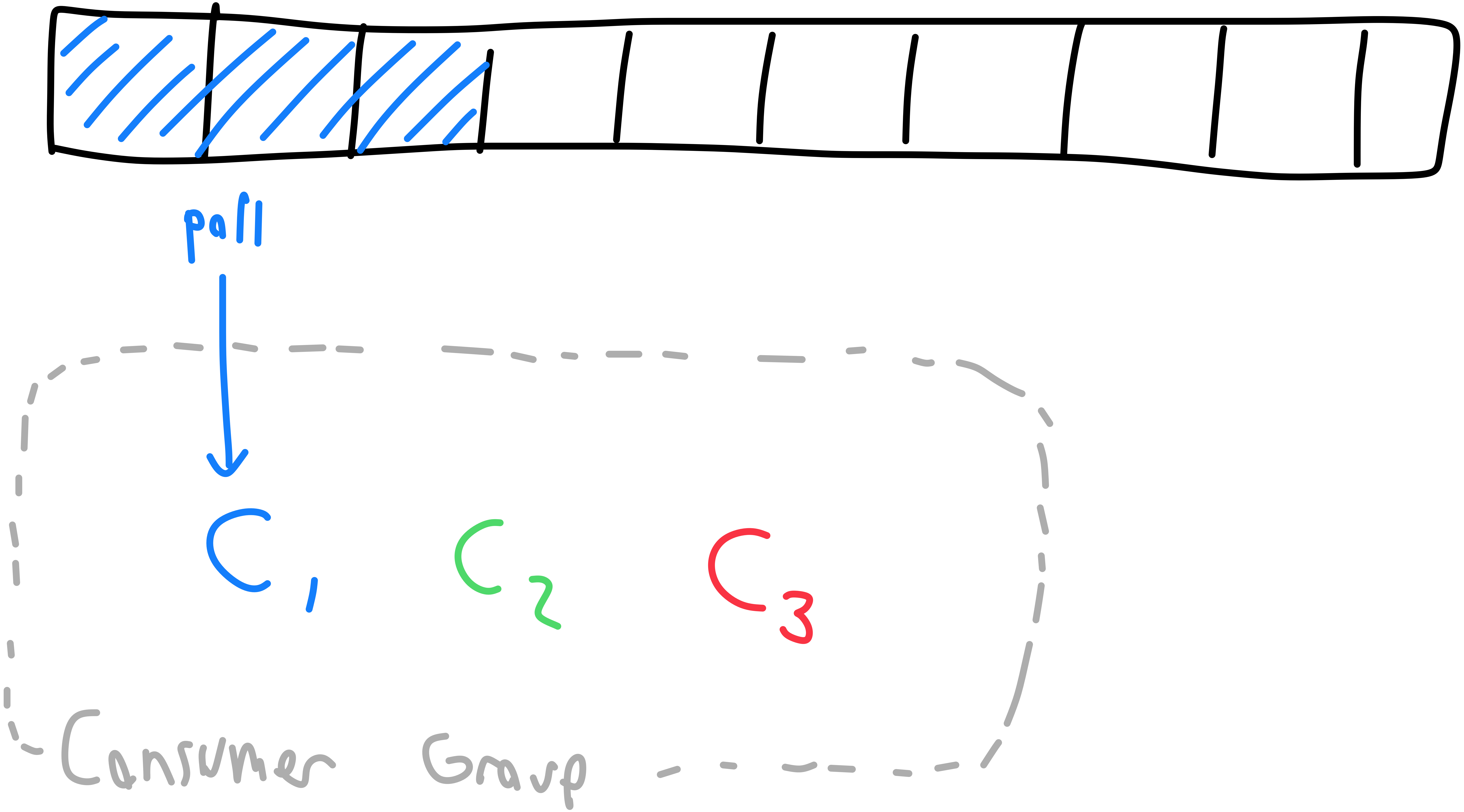
A Topic

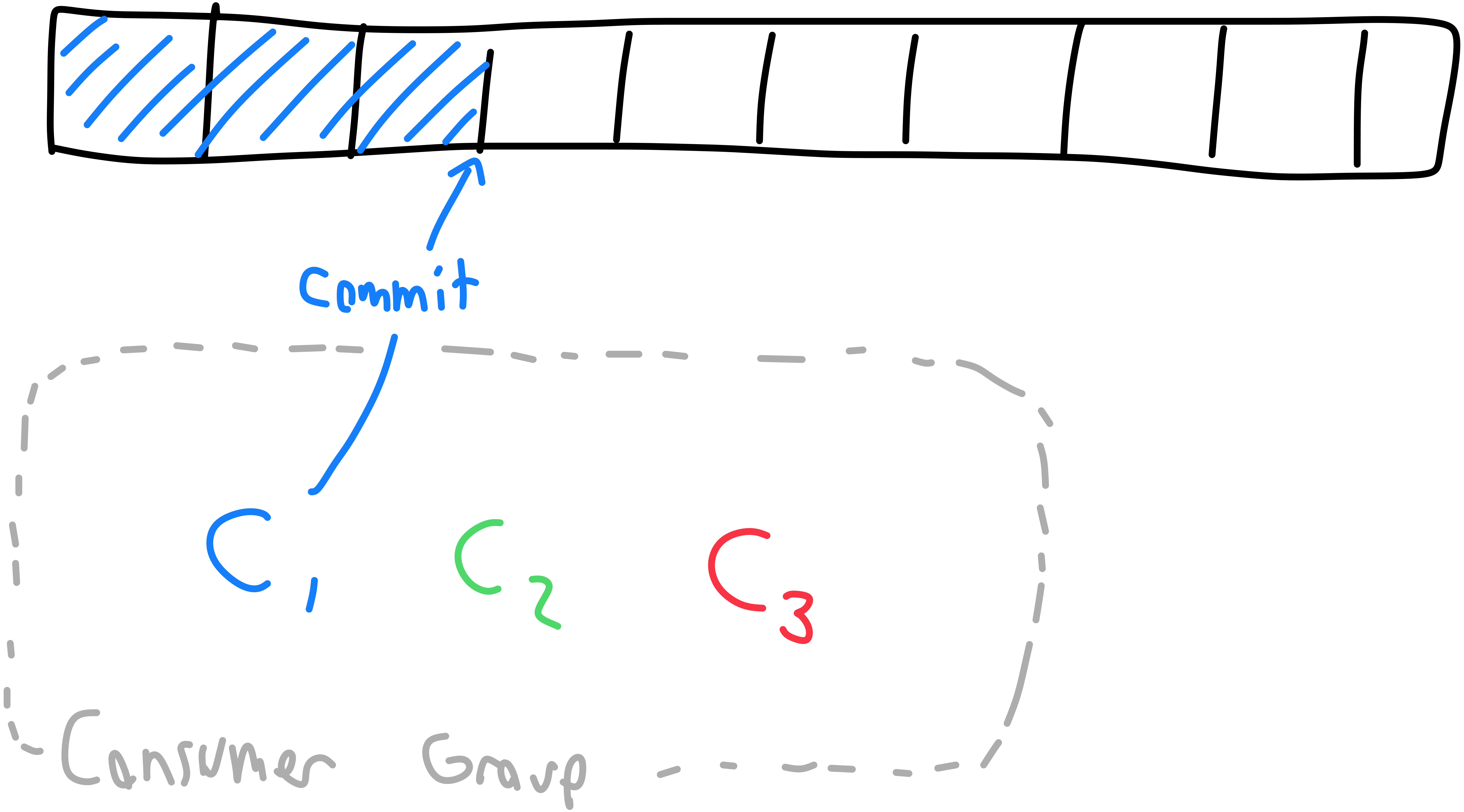


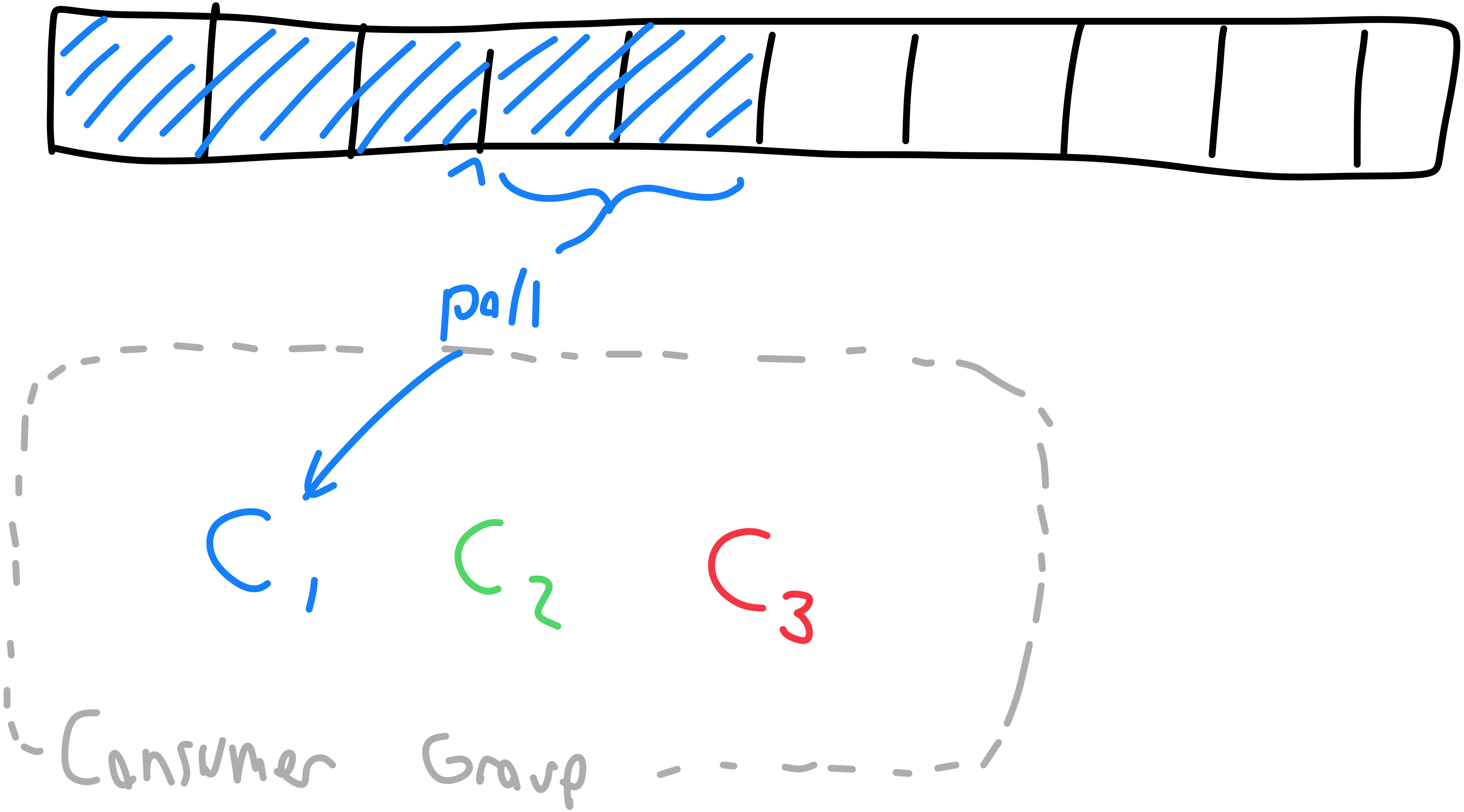
Redpanda

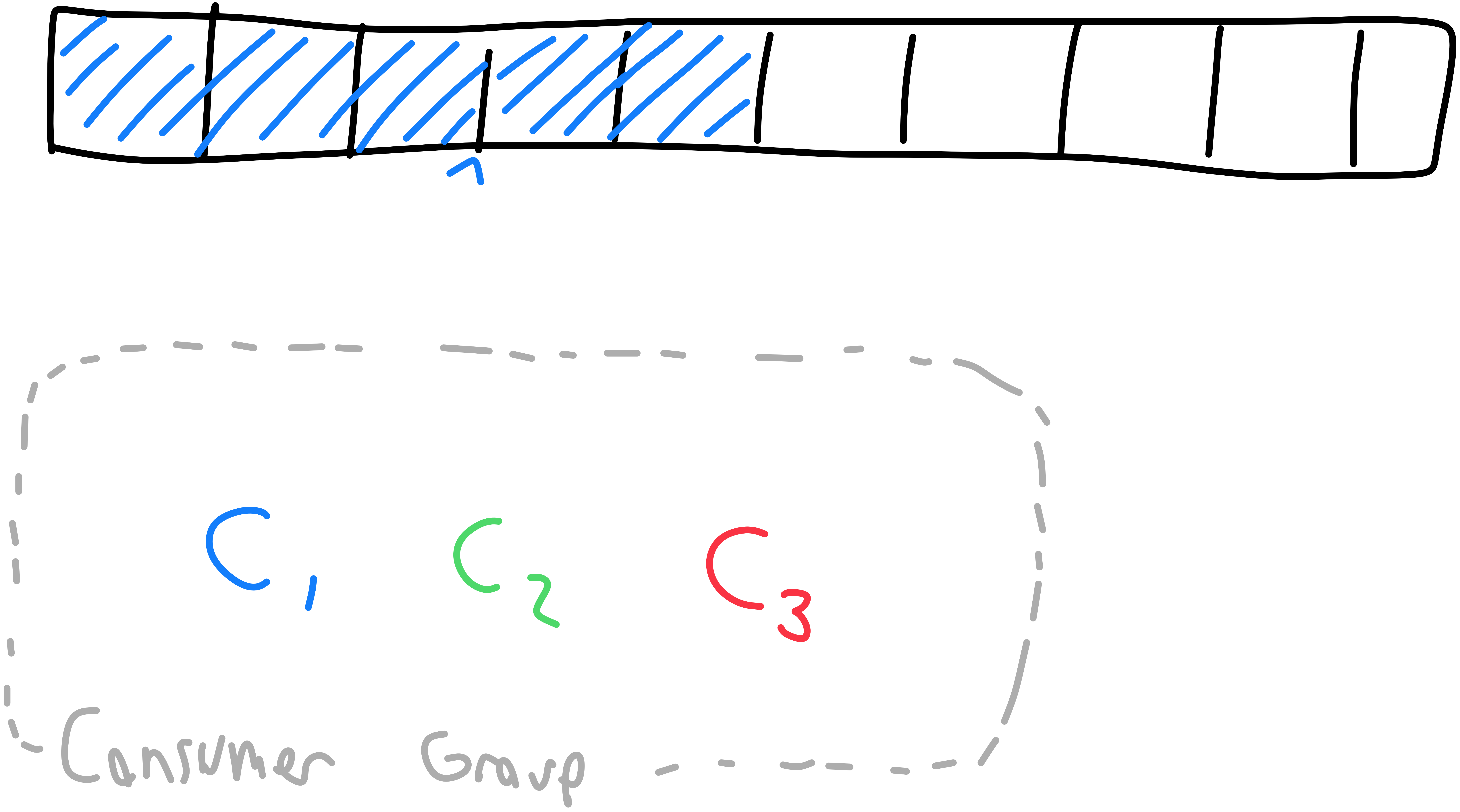


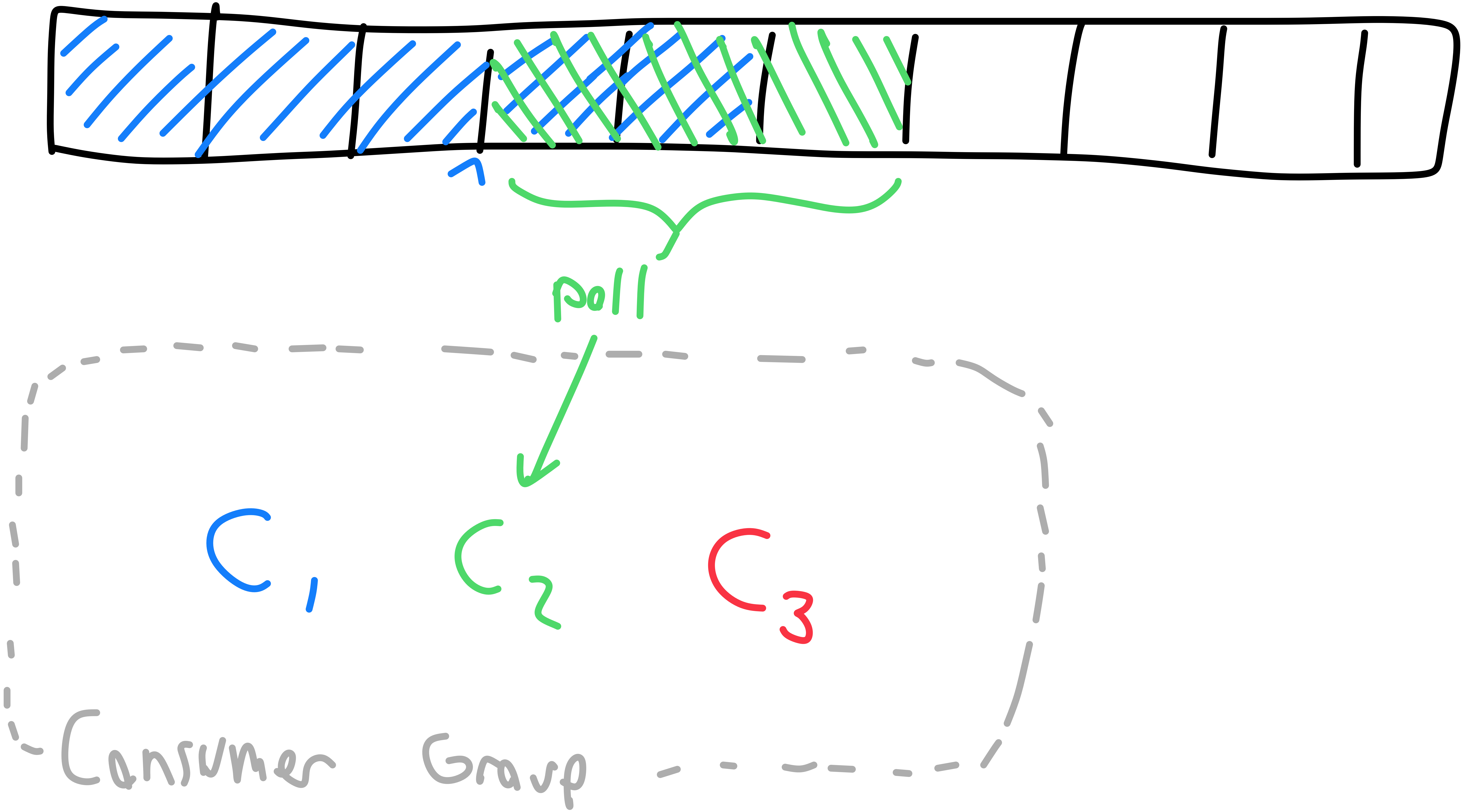

C_1
C_2
C_3
Consumer Group

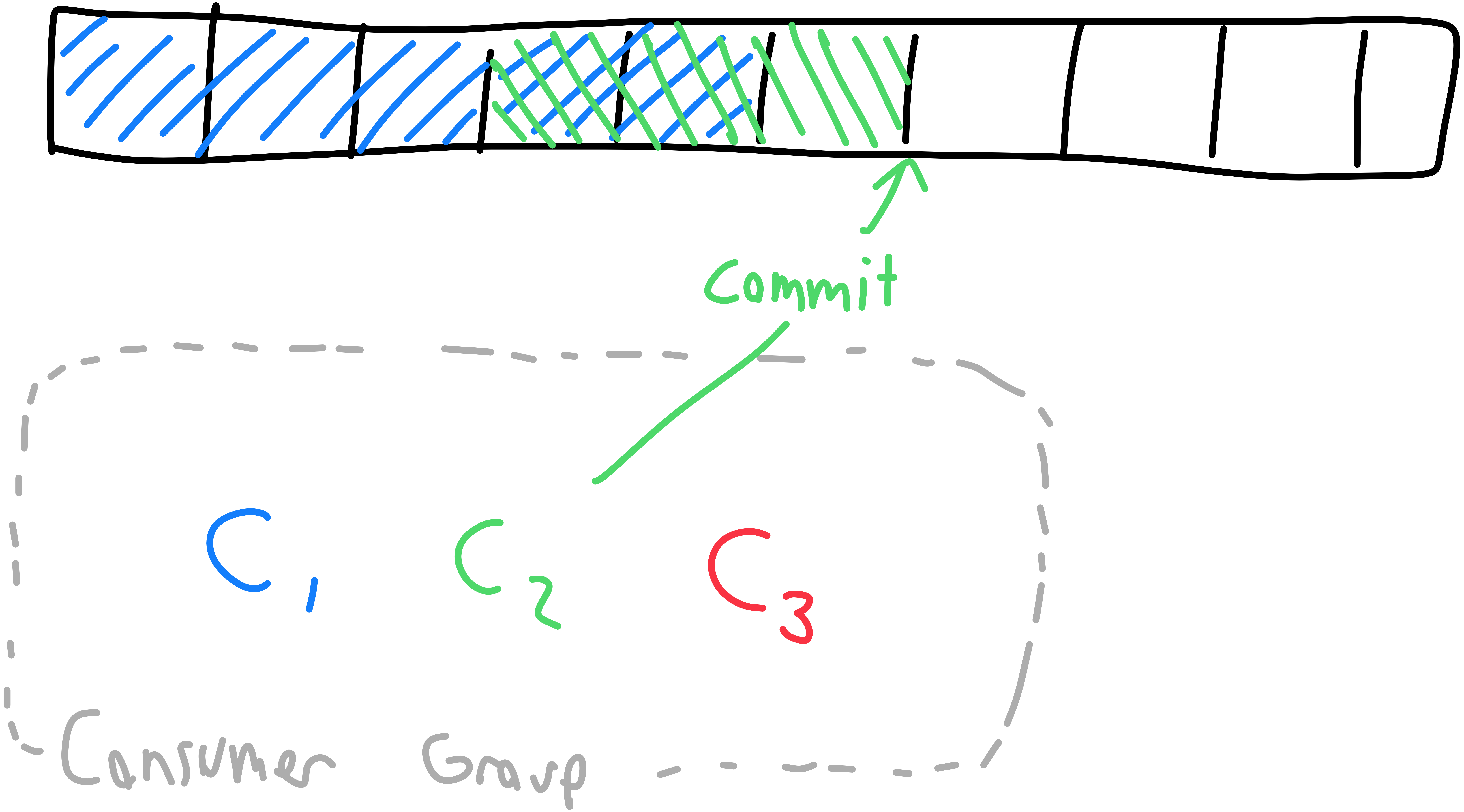

poll
C1
C2
C3
Consumer Group

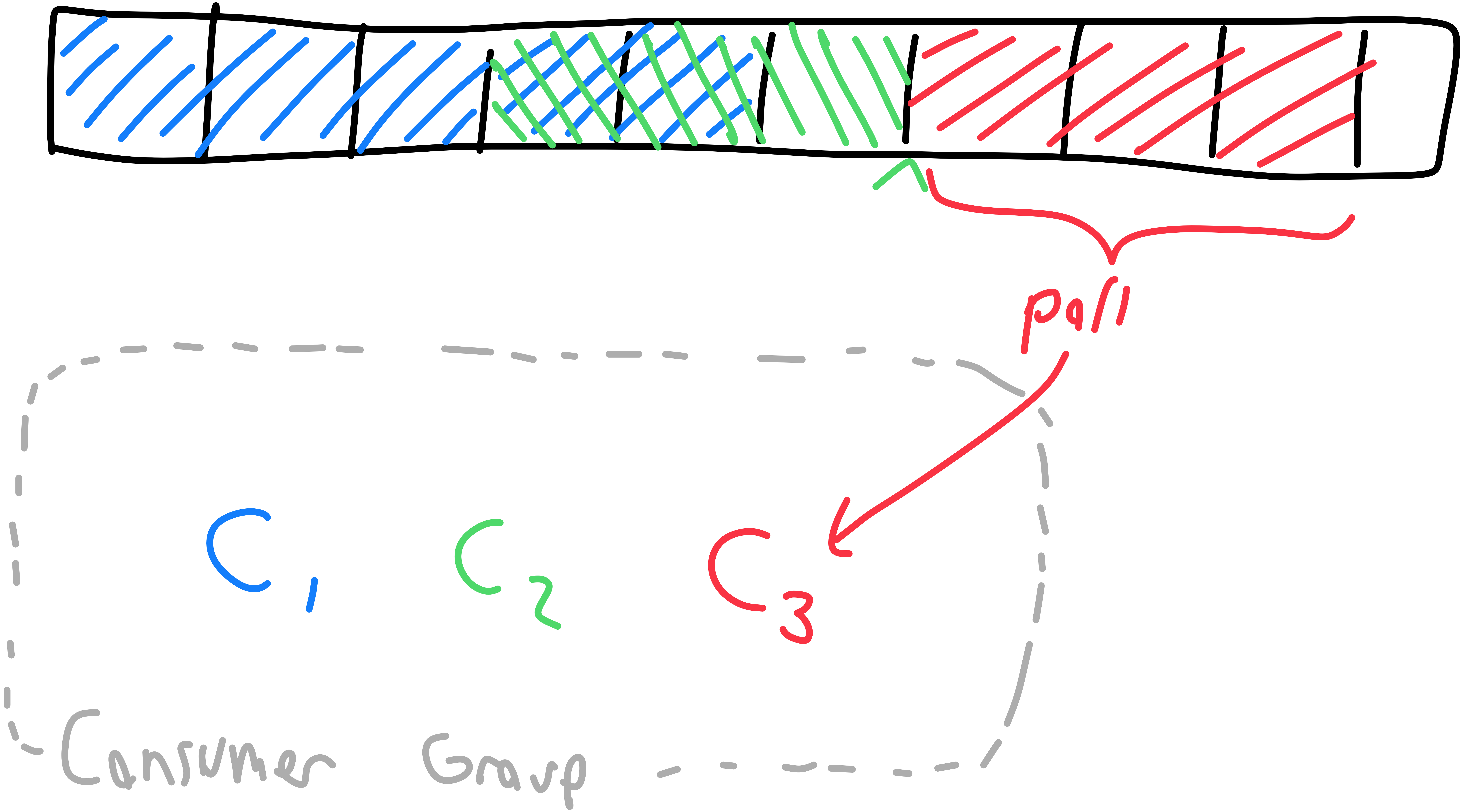

Commit
C1
C2
C3
Consumer Group


poll
C1
C2
C3
Consumer Group

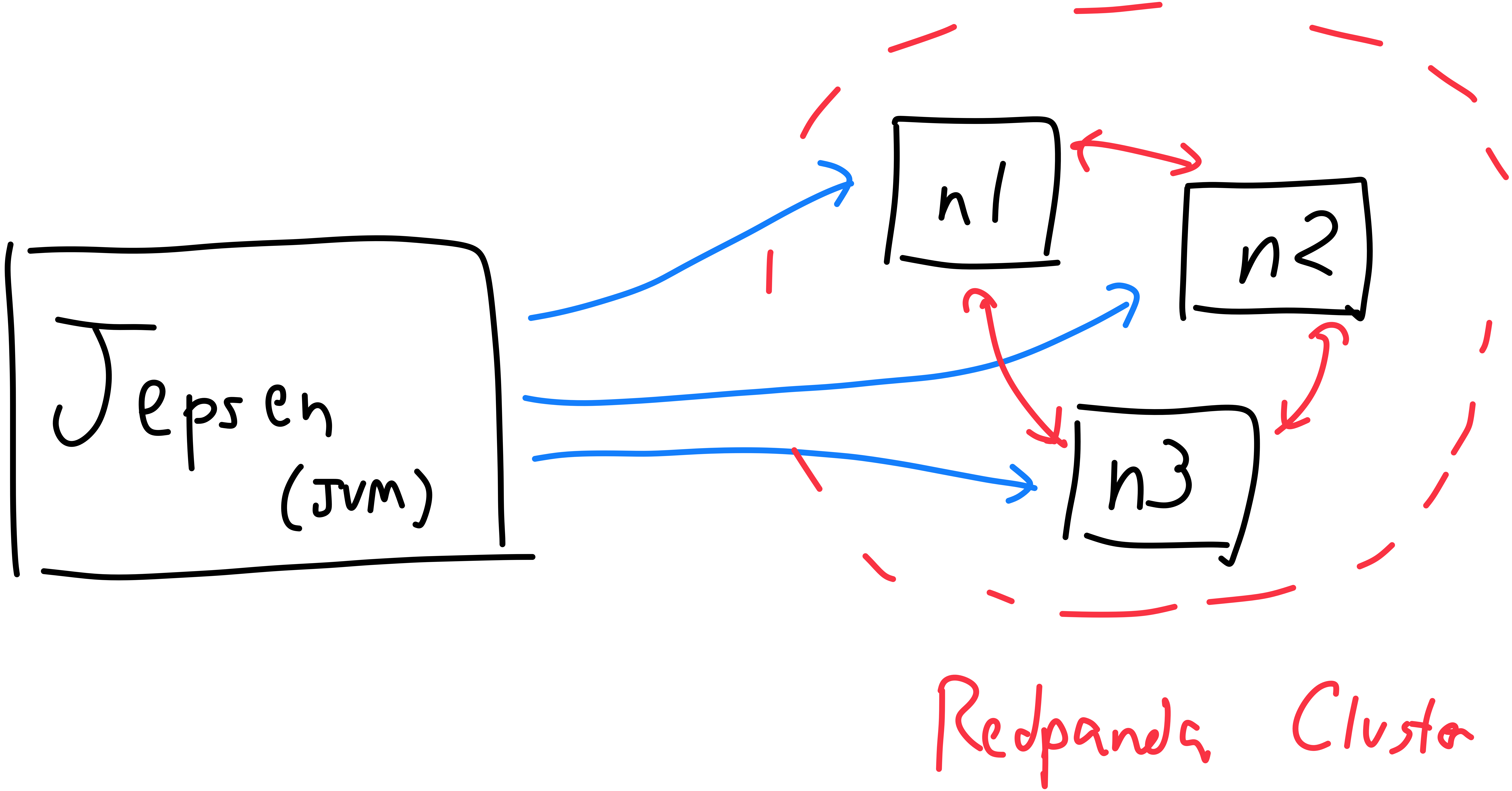

C1
C2
C3
Consumer Group

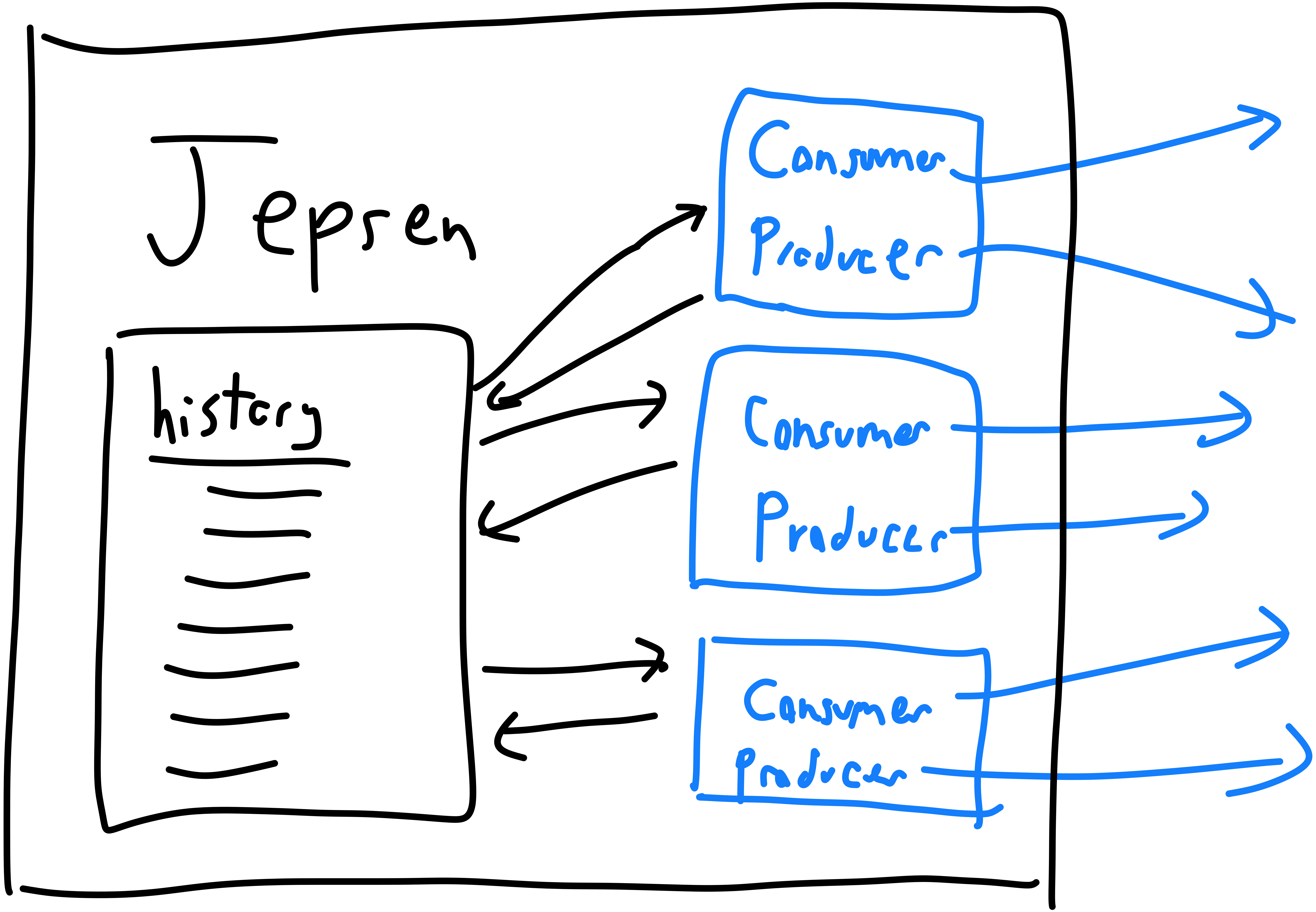

poll
C1
C2
C3
Consumer Group


Commit
C1
C2
C3
Consumer Group


poll
C1
C2
C3
Consumer Group

Test Design


Jepsen
(JVM)
n1
n2
n3
Redpanda Cluster


Jepsen
history
Consumer
Producer
Consumer
Producer
Consumer
Producer

Quene

History

invoke	subscribe	[k]
ok	subscribe	(k]

History

invoke	subscribe	[k]
ok	subscribe	(k)
invoke	poll	[poll]
ok	poll	[poll {k, [[0 :a] [1 :b]]}]

History

invoke	subscribe	[k]
ok	subscribe	[k]
invoke	poll	[poll]
ok	poll	[poll {k_1 [[0:a] [1:b]]}]
invoke	send	[send k_2 :c]
ok	send	[send k_2 [3 :c]]

History

ok txn [[poll {k_1 [[0 :a] [1 :b]]}],
[send k_2 [3 :c]]]

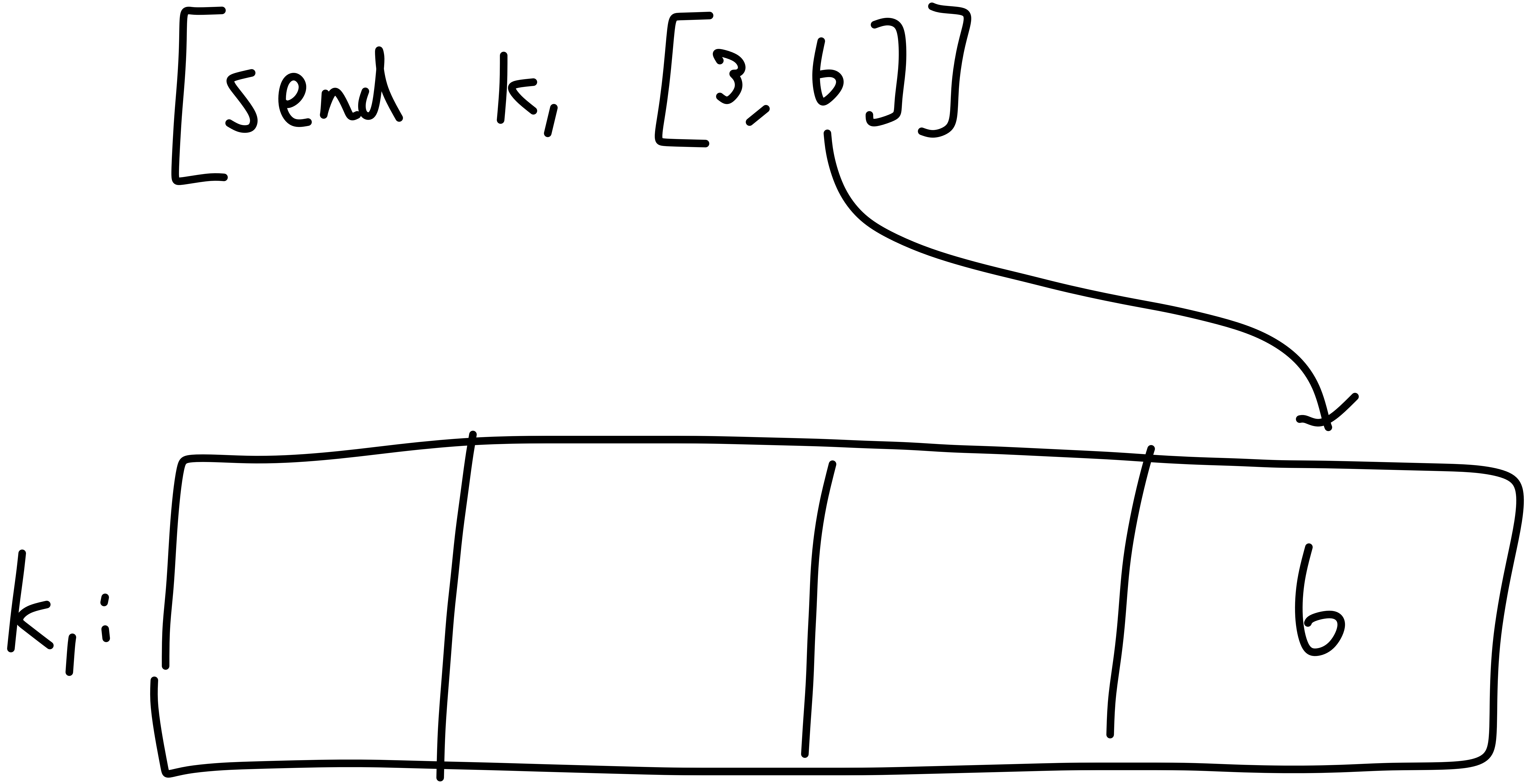
Checker

fail [[send x a]]   ok ([poll {x ([3, a])}])

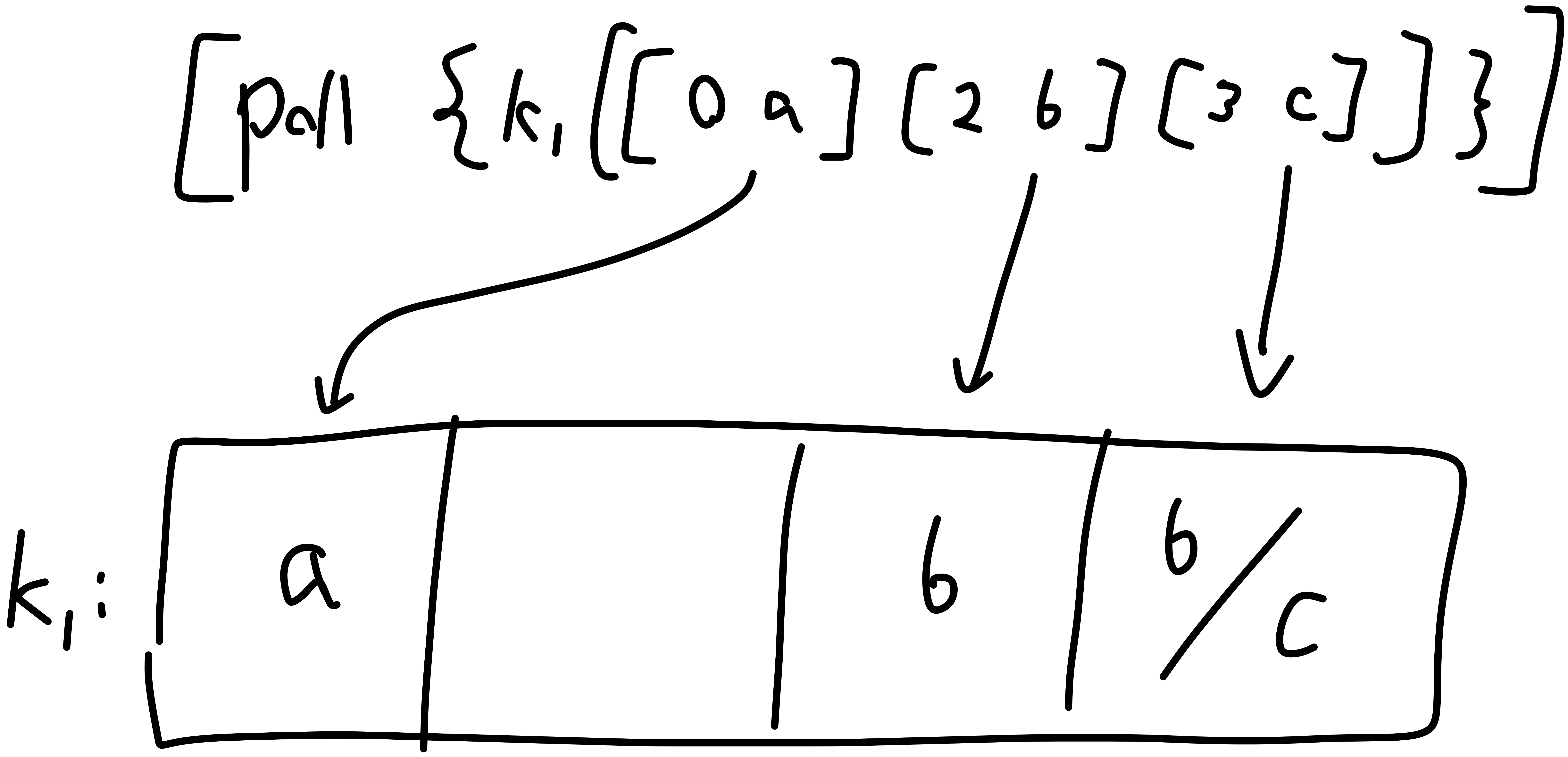
fail [[send x a]]   ok [[poll {x ([3, a])}]]

Aborted Read!

(G1a)

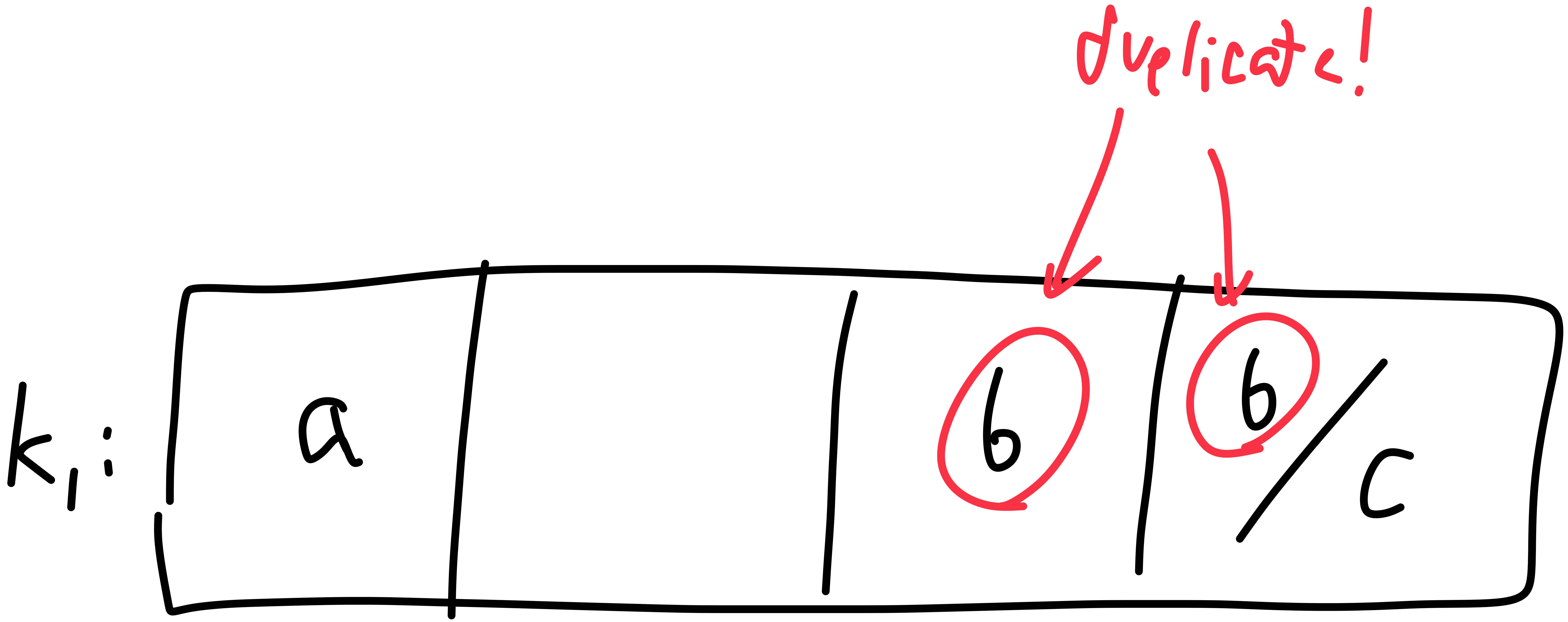


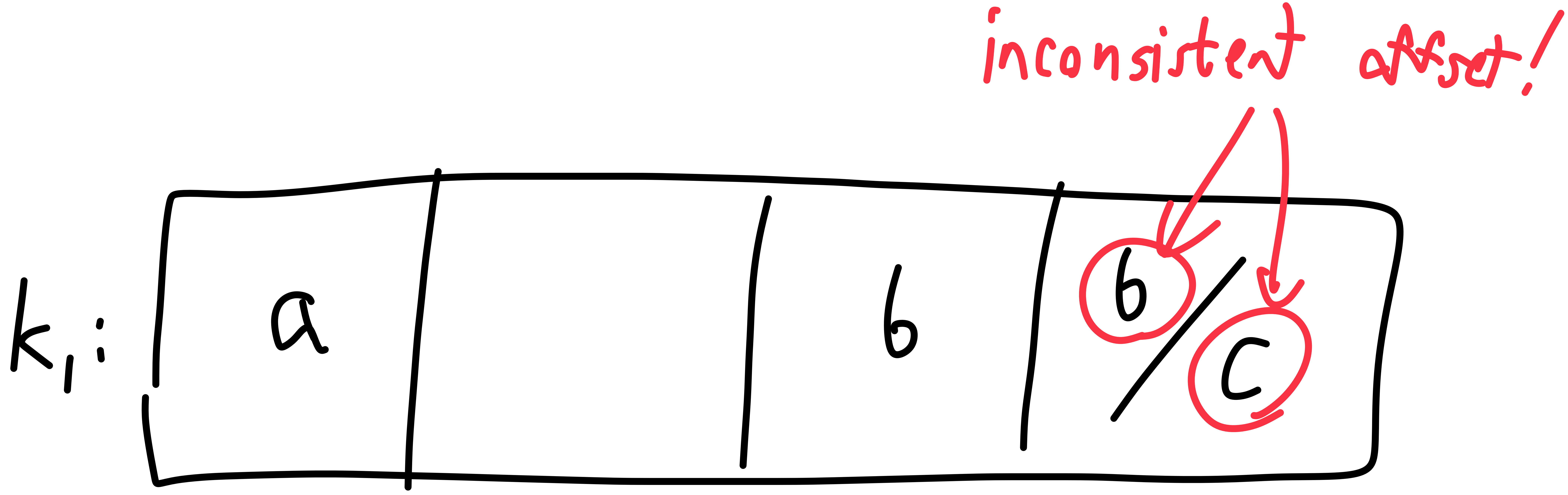
"version order"


[pall {k1 ([0 a] [2 b] [3 c])}]
k1:
a
b
b
c

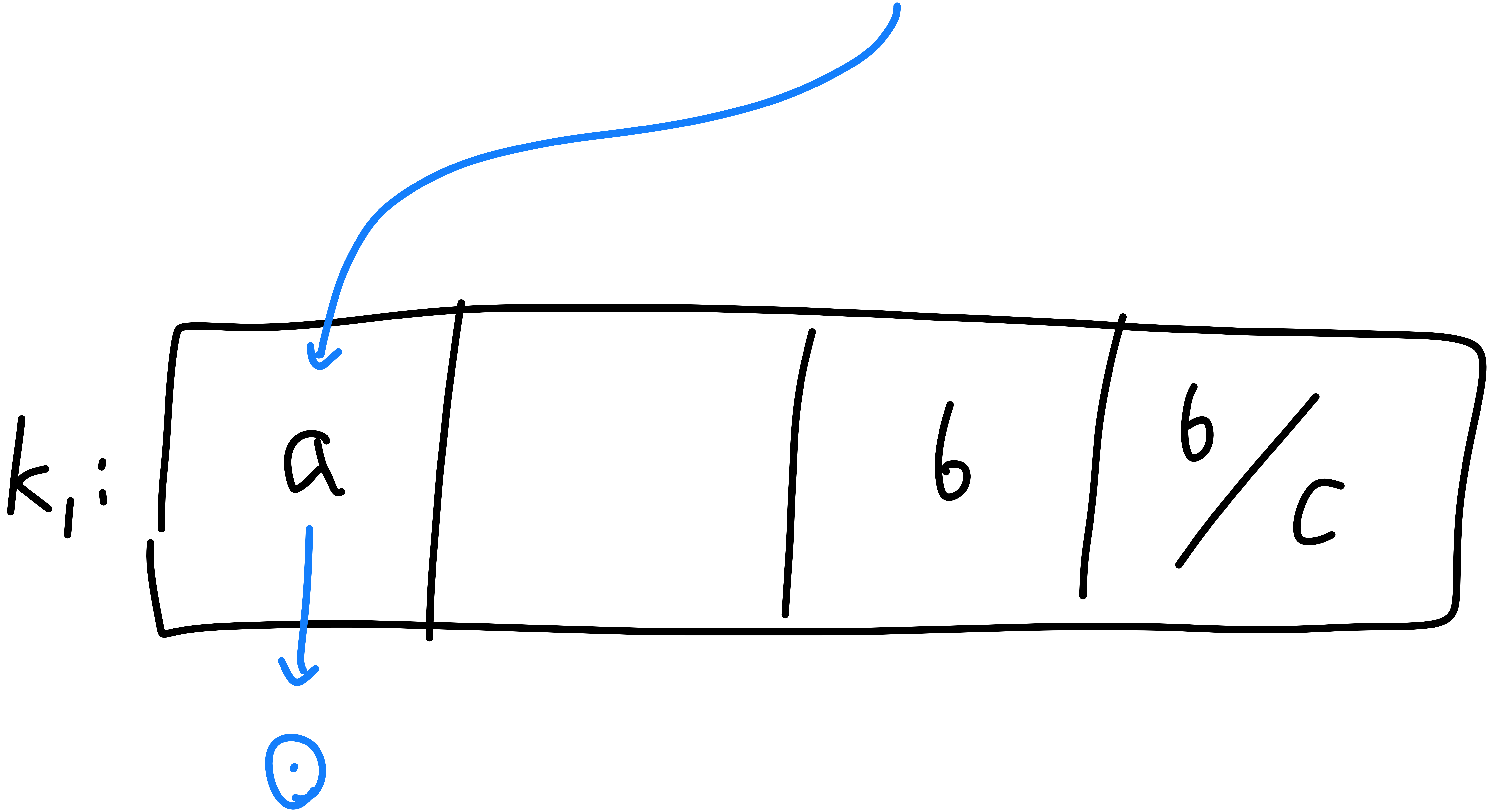
Ok! We might not observe all

k_1:	a		b	b/c

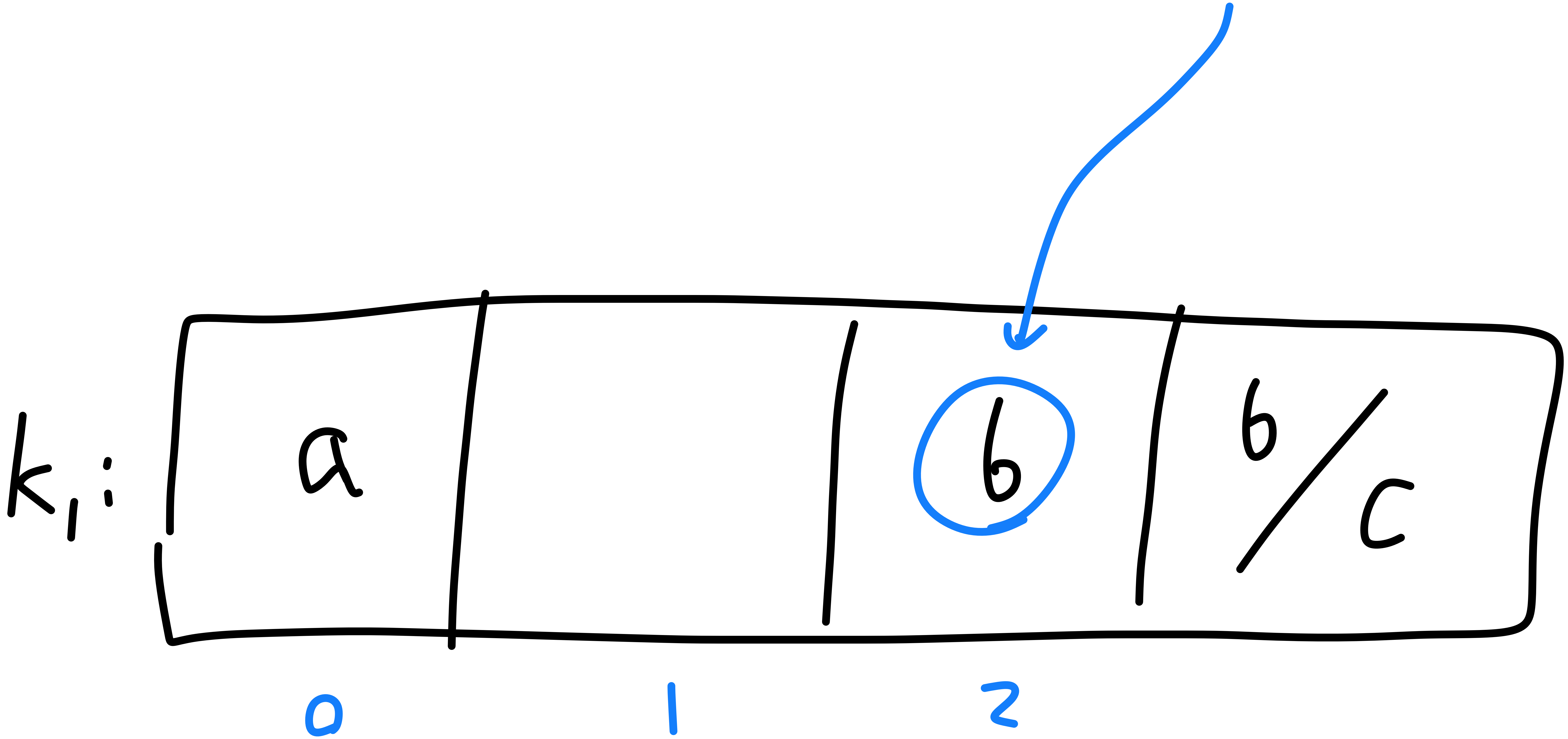

duplicate!
k_1:
a
b
b
c


inconsistent offset!
k_1:
a
b
b
c

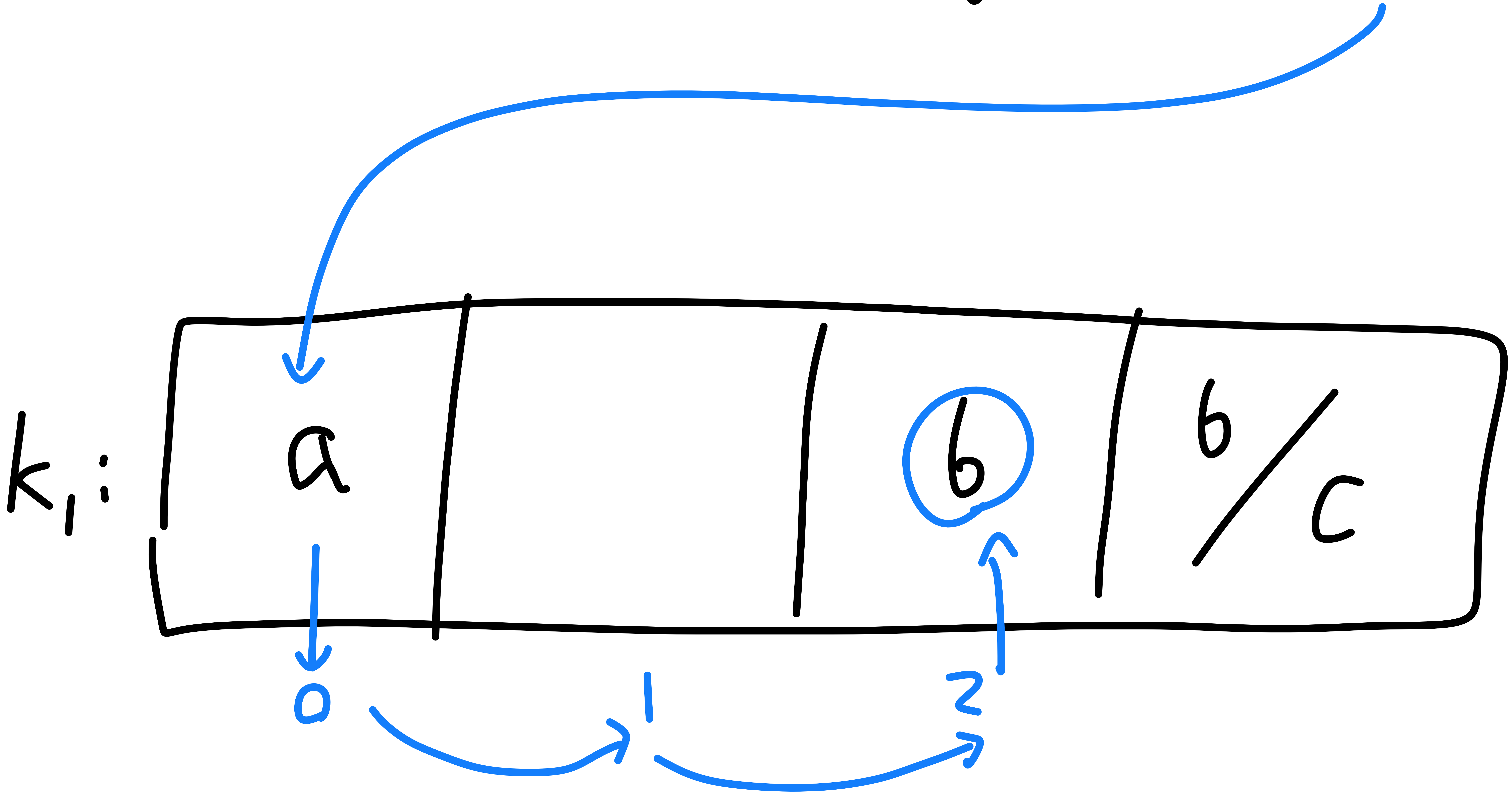
"what affect is a?"



"what value at offset 2?"



"Which value came just after a?"

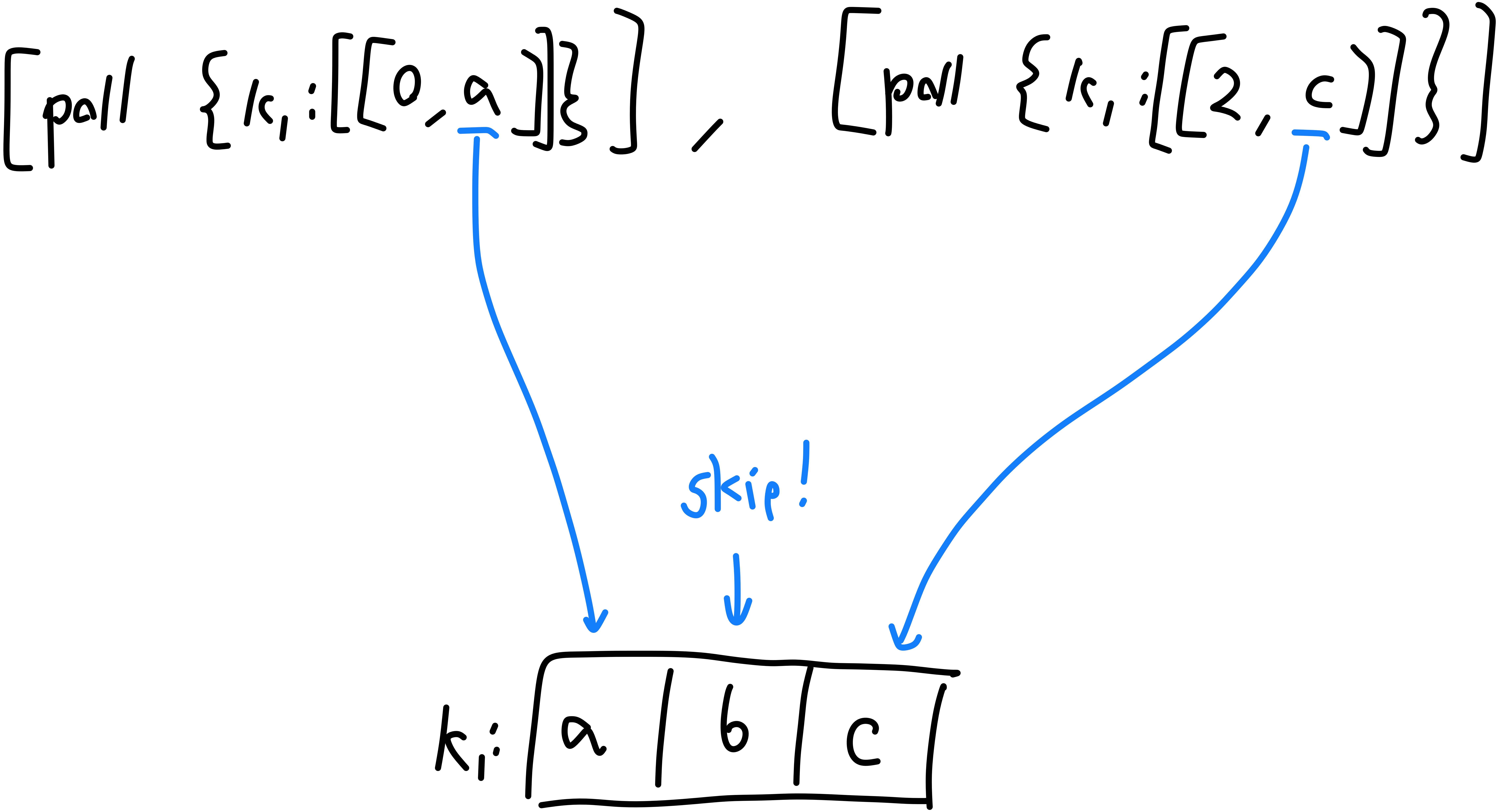


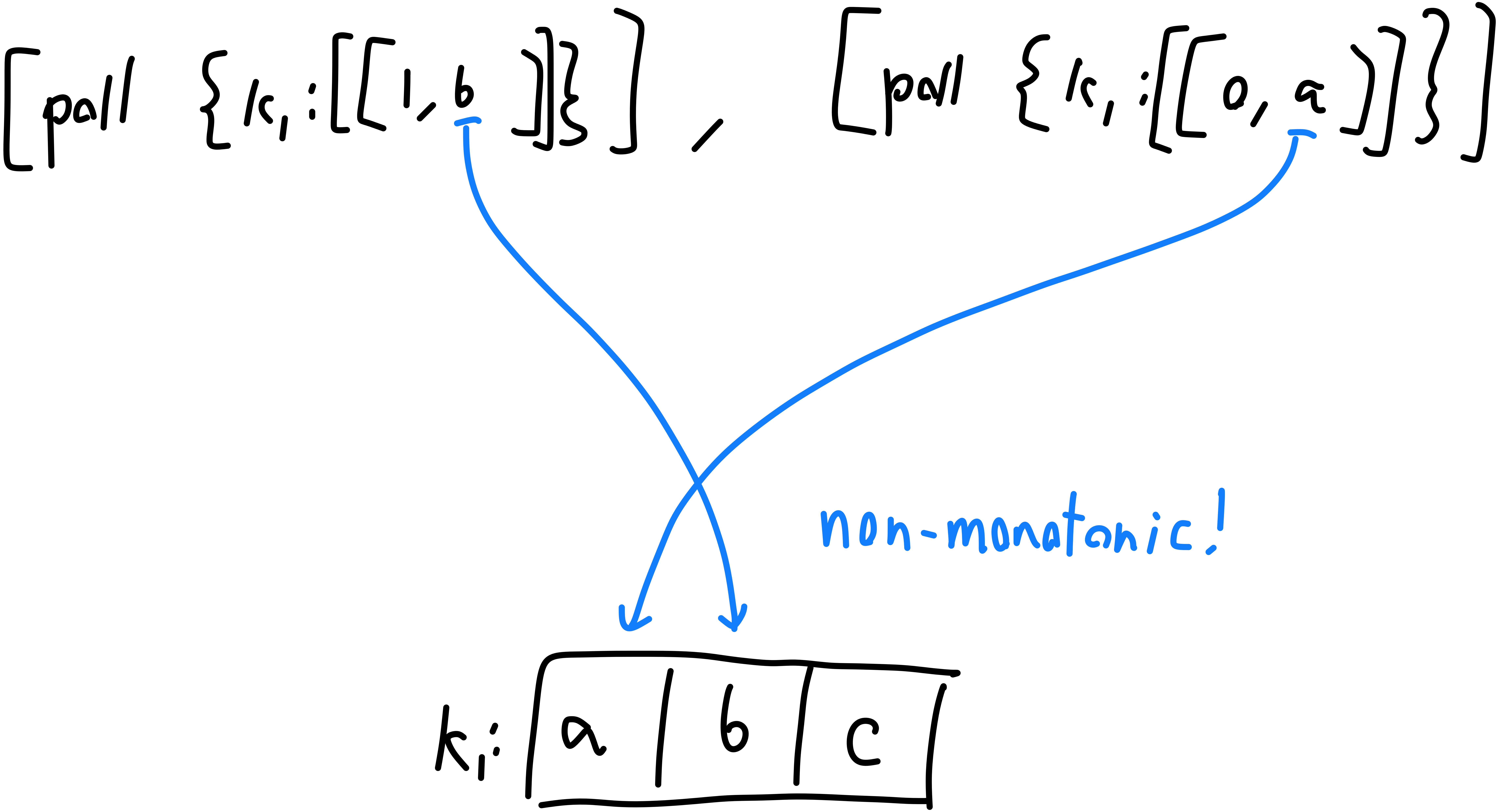
k_1:

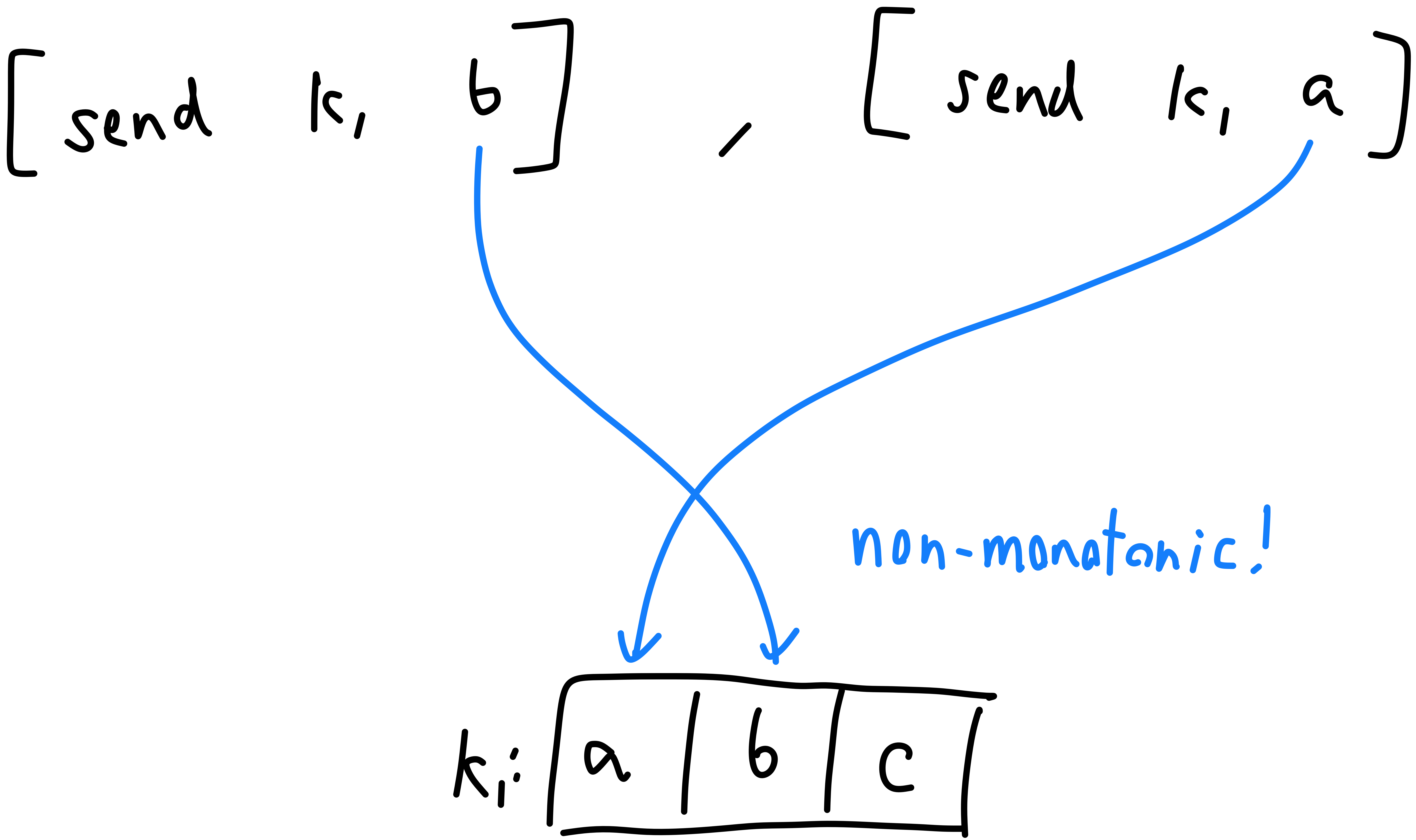
a	b	c

[pall {k_1:[[0, a]]}] , [pall {k_1:[[2, c]]}]

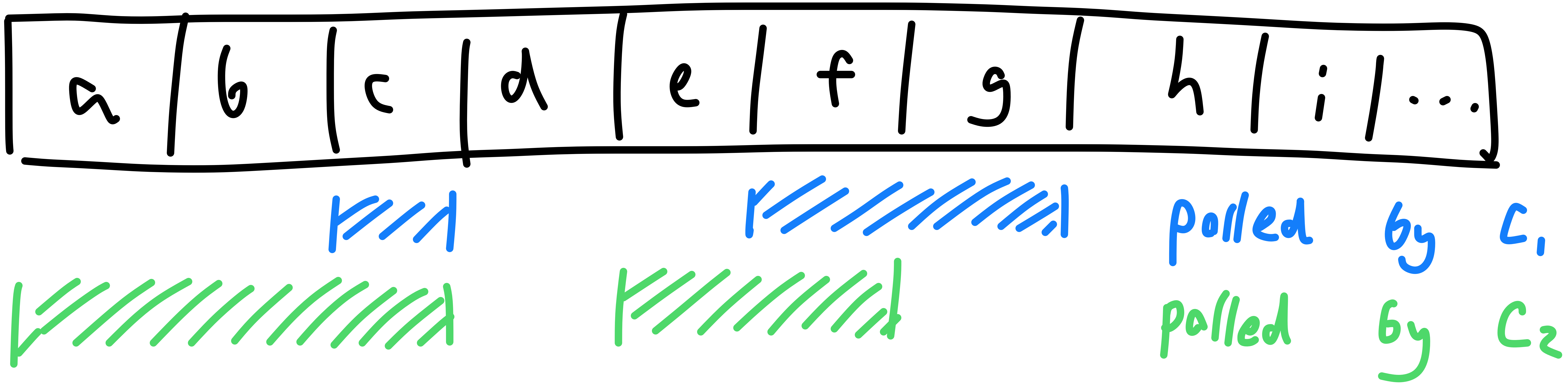
k_1:	a	b	c

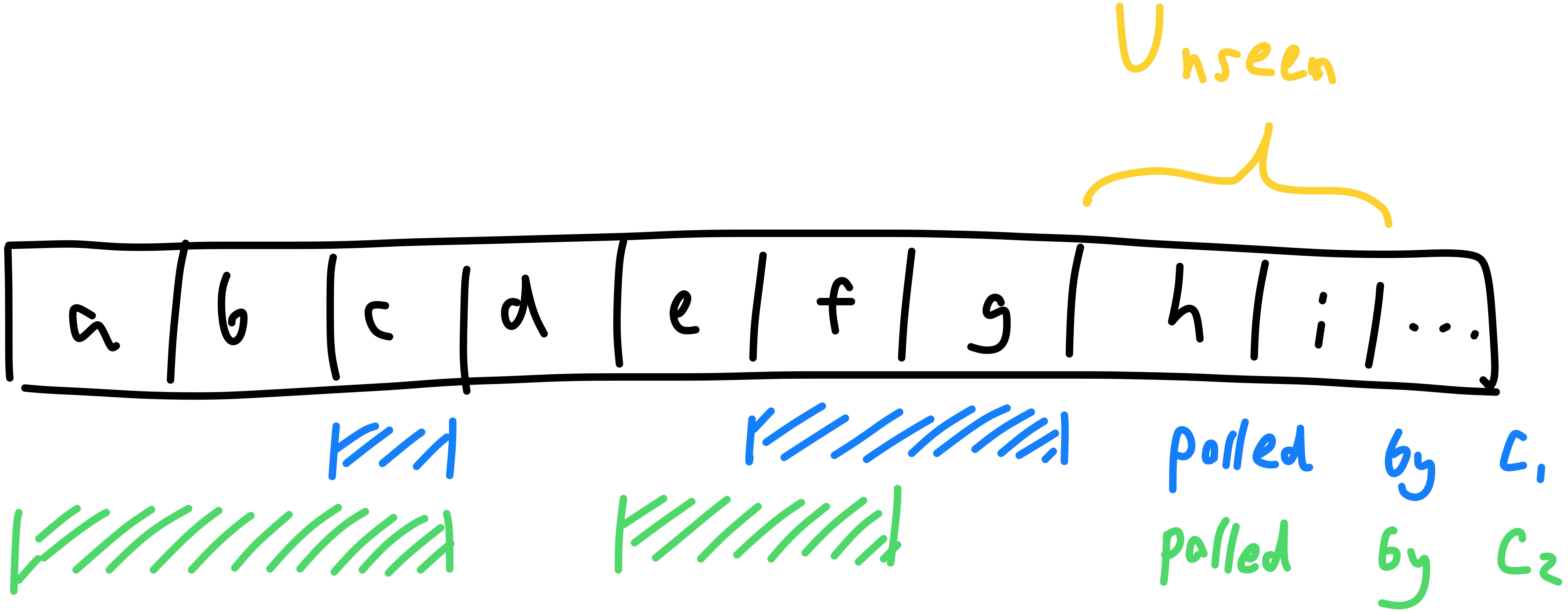

[pall {k1:[[0, a]]}]
,
[pall {k1:[[2, c]]}]
skip!
k1:
a
b
c

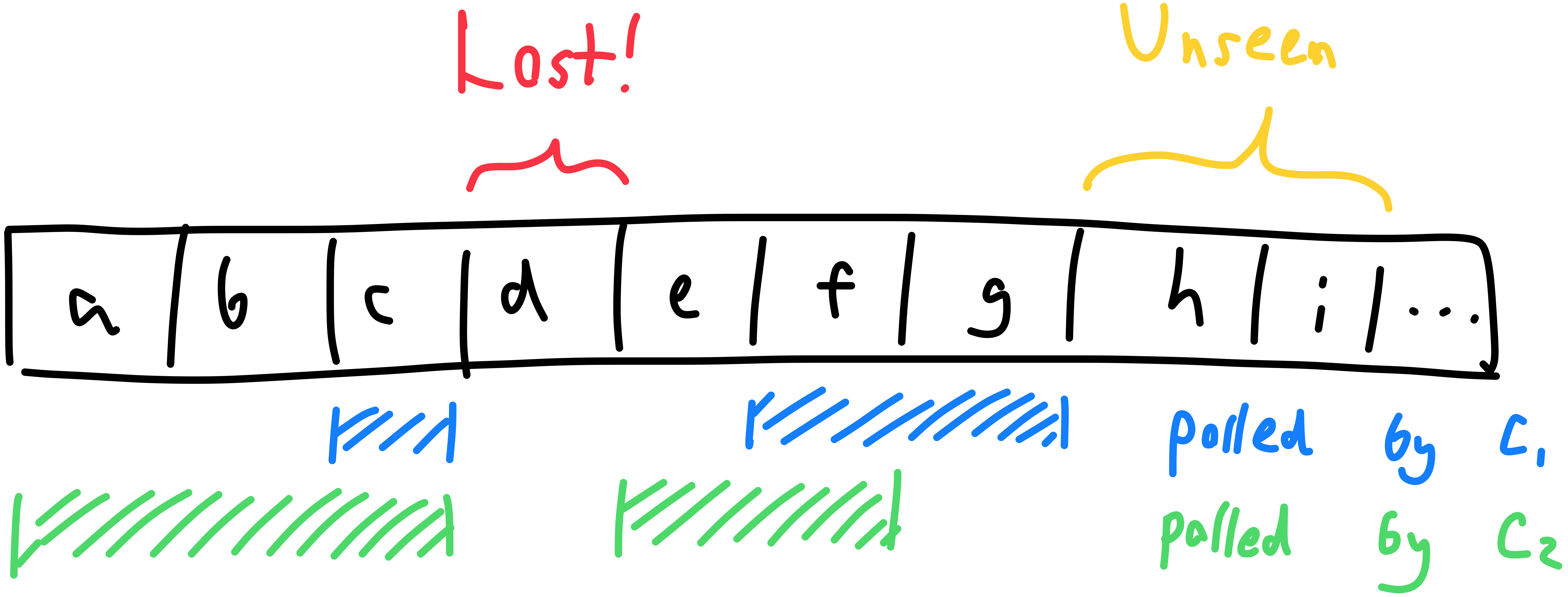

[pall {k1 :[[1, b]]}] , [pall {k1 :[[0, a]]}]
non-monatonic!
k1: a b c


[send k, b] , [send k, a]
non-monatonic!
k_1: a | b | c

a	b	c	d	e	f	g	h	i	...


a
b
c
d
e
f
g
h
i
...
polled by c_1
polled by c_2


Unseen
a
b
c
d
e
f
g
h
i
...
polled by c_1
polled by c_2


Lost!
Unseen
a
b
c
d
e
f
g
h
i
...
polled by C1
polled by C2

T_1

$[\![\,\text{send } k_1 \quad b\,]\!]$

T_2

$[\![\,\text{poll } \{k_1 \; [\ldots [1, b] \ldots]\}\,]\!]$

K_1:

a	b	c

$$T_1 \xrightarrow{\text{write-read}} T_2$$

$$[\![\text{send } k_1, \textcircled{b}]\!] \qquad [\![\text{poll } \{k_1, [\ldots [1, \textcircled{b}] \ldots]\}]\!]$$

K_1:

a	b	c

T_1

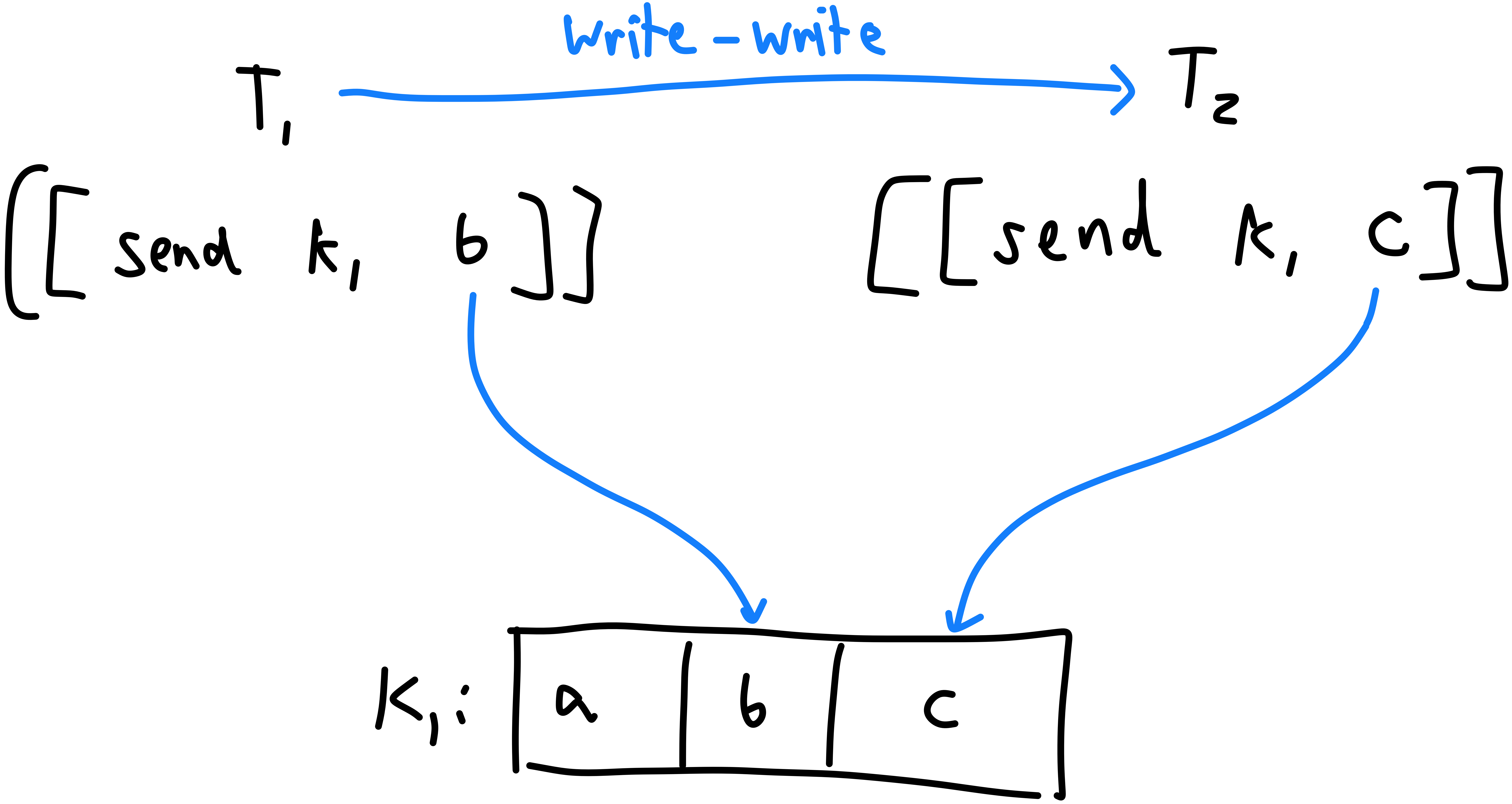
$[\![\text{send } k_1 \ b]\!]$

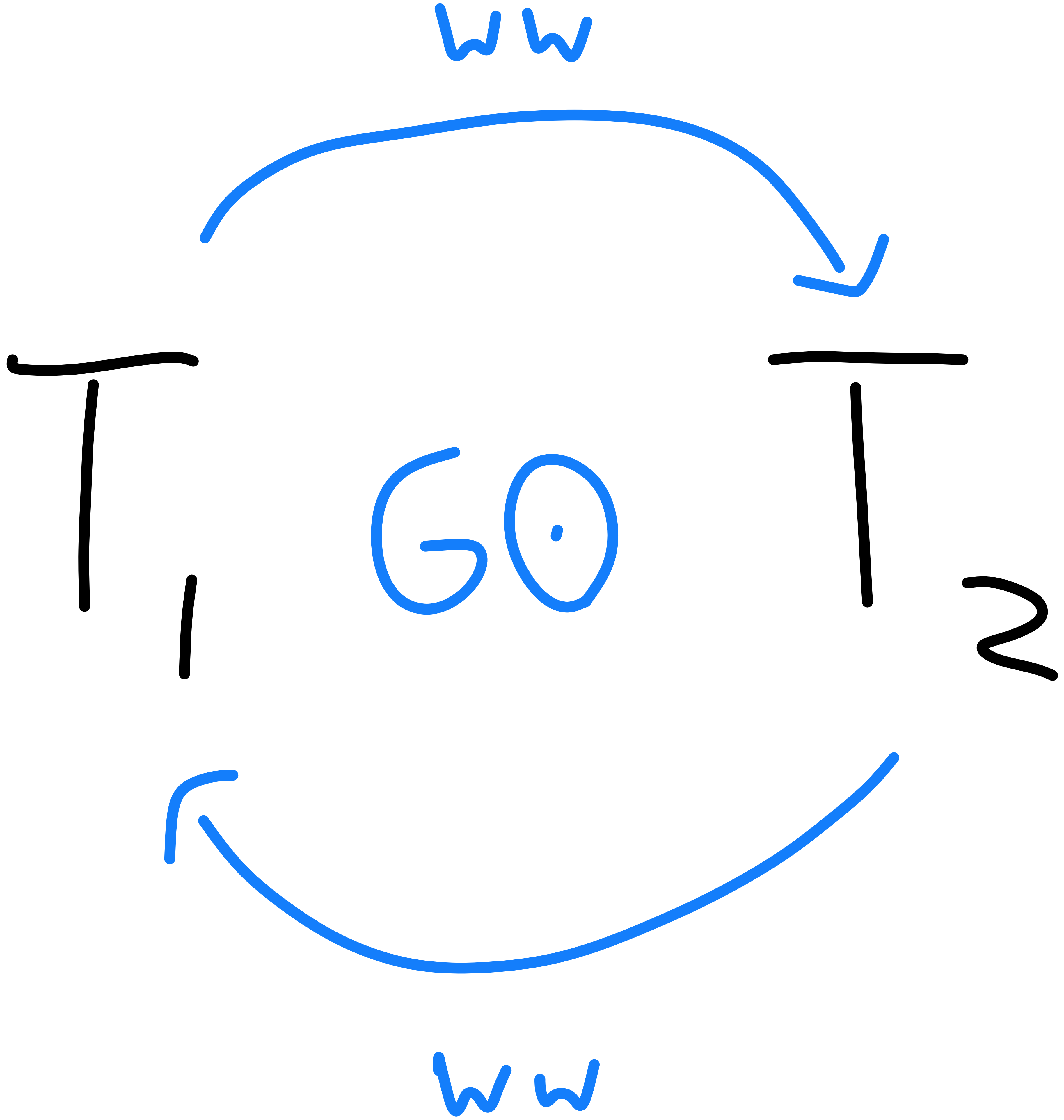
T_2

$[\![\text{send } k_1 \ c]\!]$

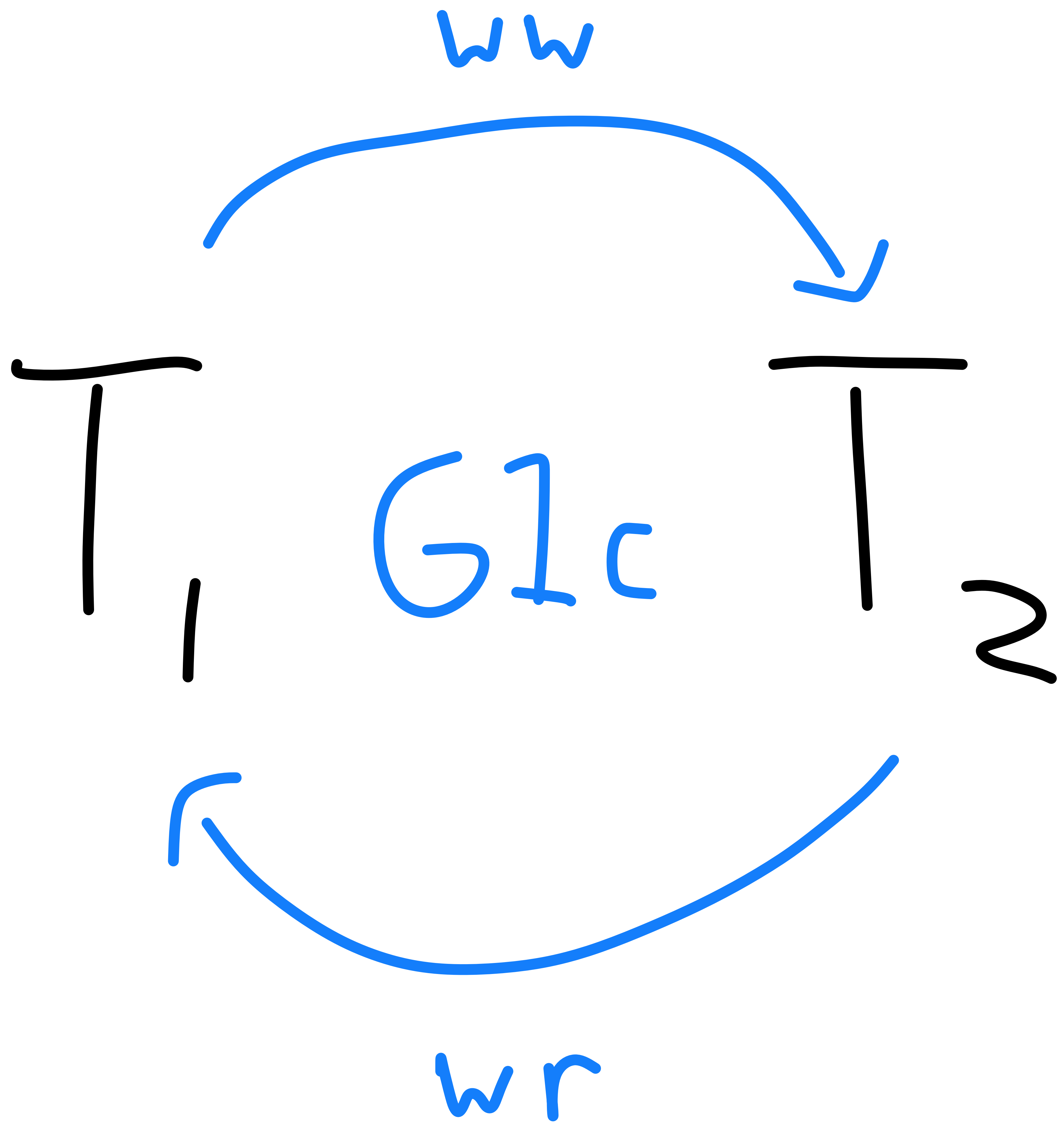
K_1:

a	b	c


write-write
T_1
T_2
[[send k, b]]
[[send k, c]]
K_1:
a
b
c



Write Cycle



Circular Information Flow

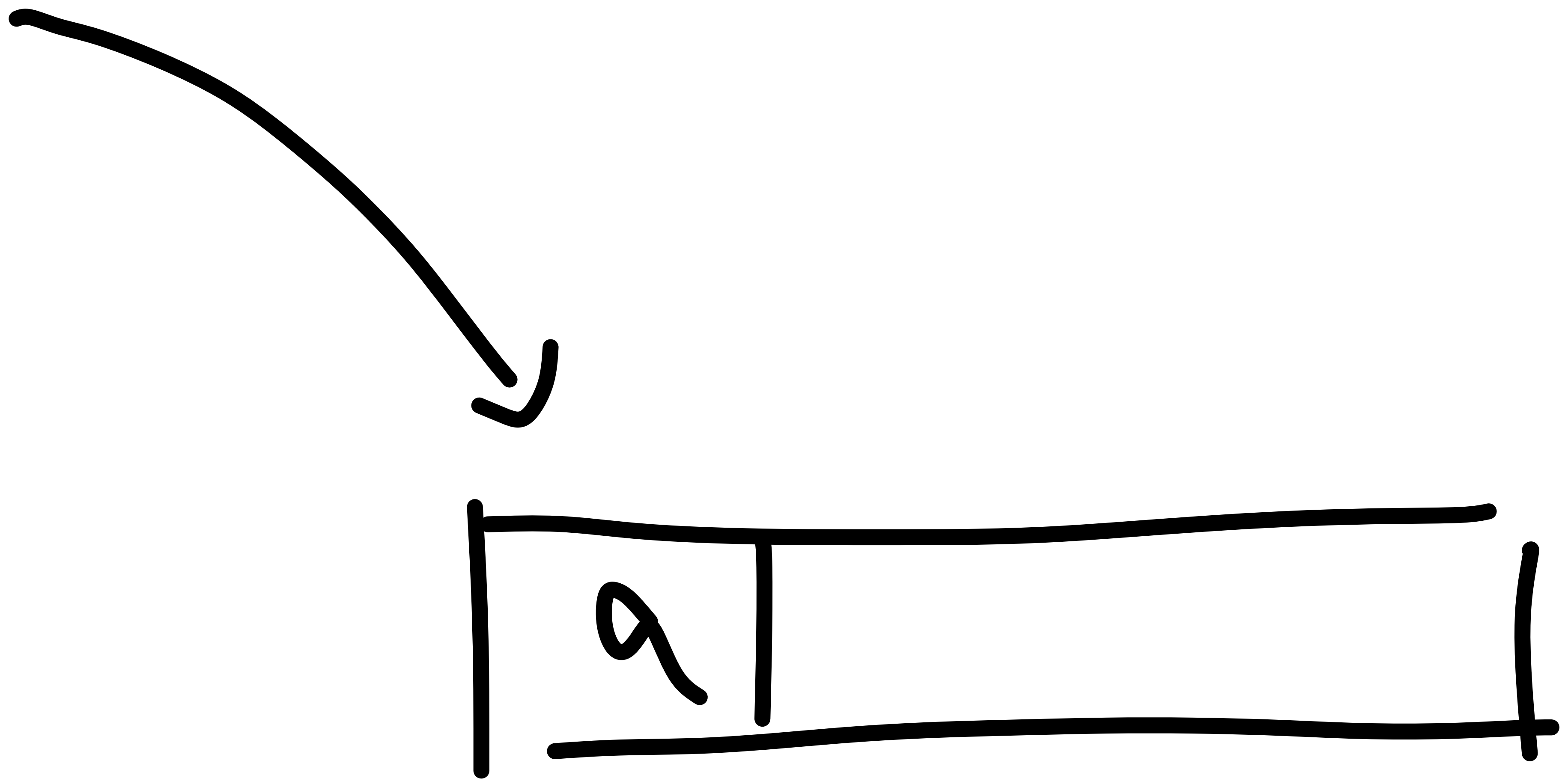
Faults

- Crash
- Pause
- Clock skew
- Network Partition
- Membership change

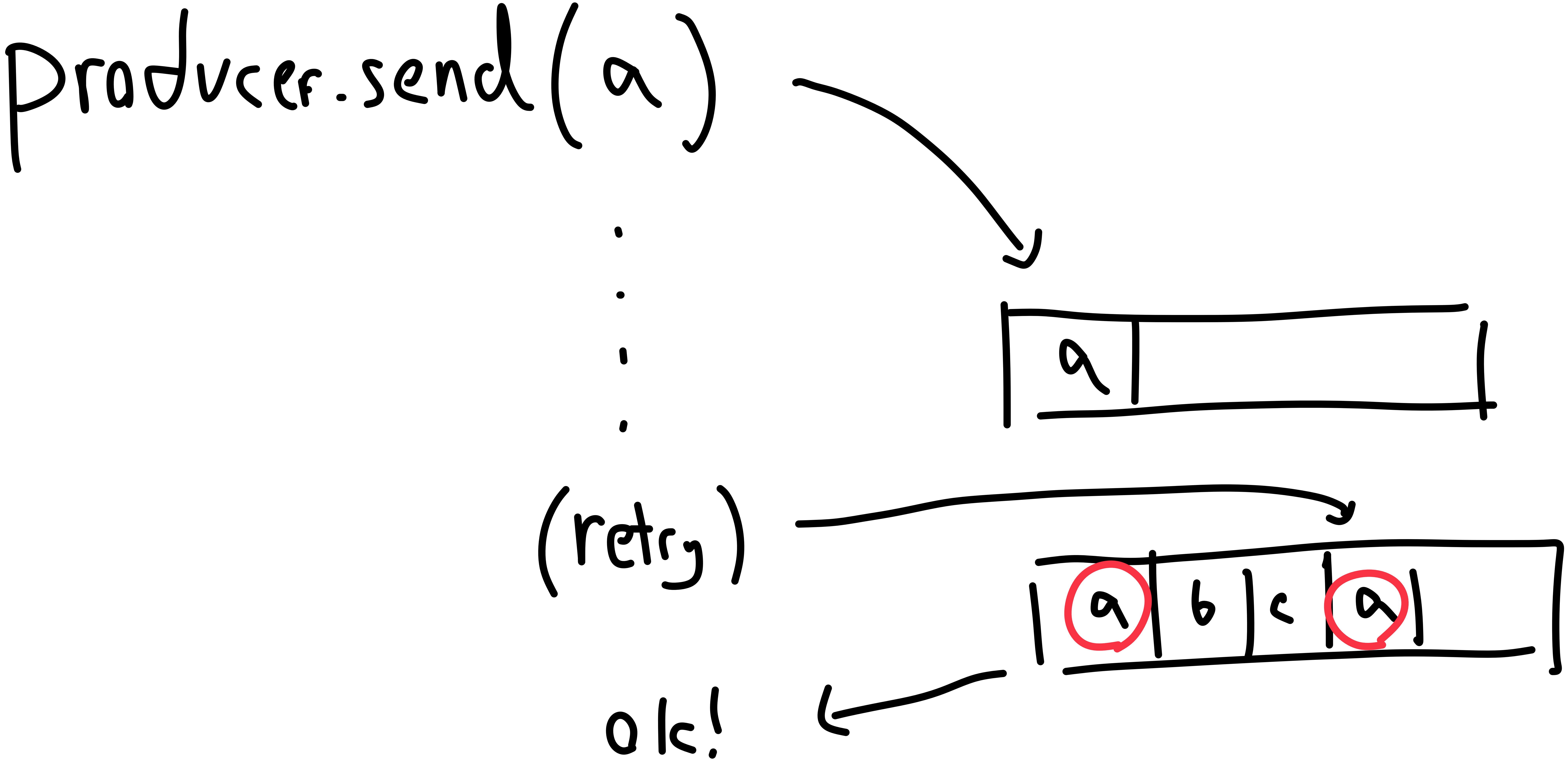
Results

#1 Duplicate Writes by Default

`producer.send(a)`



#1 Duplicate Writes by Default

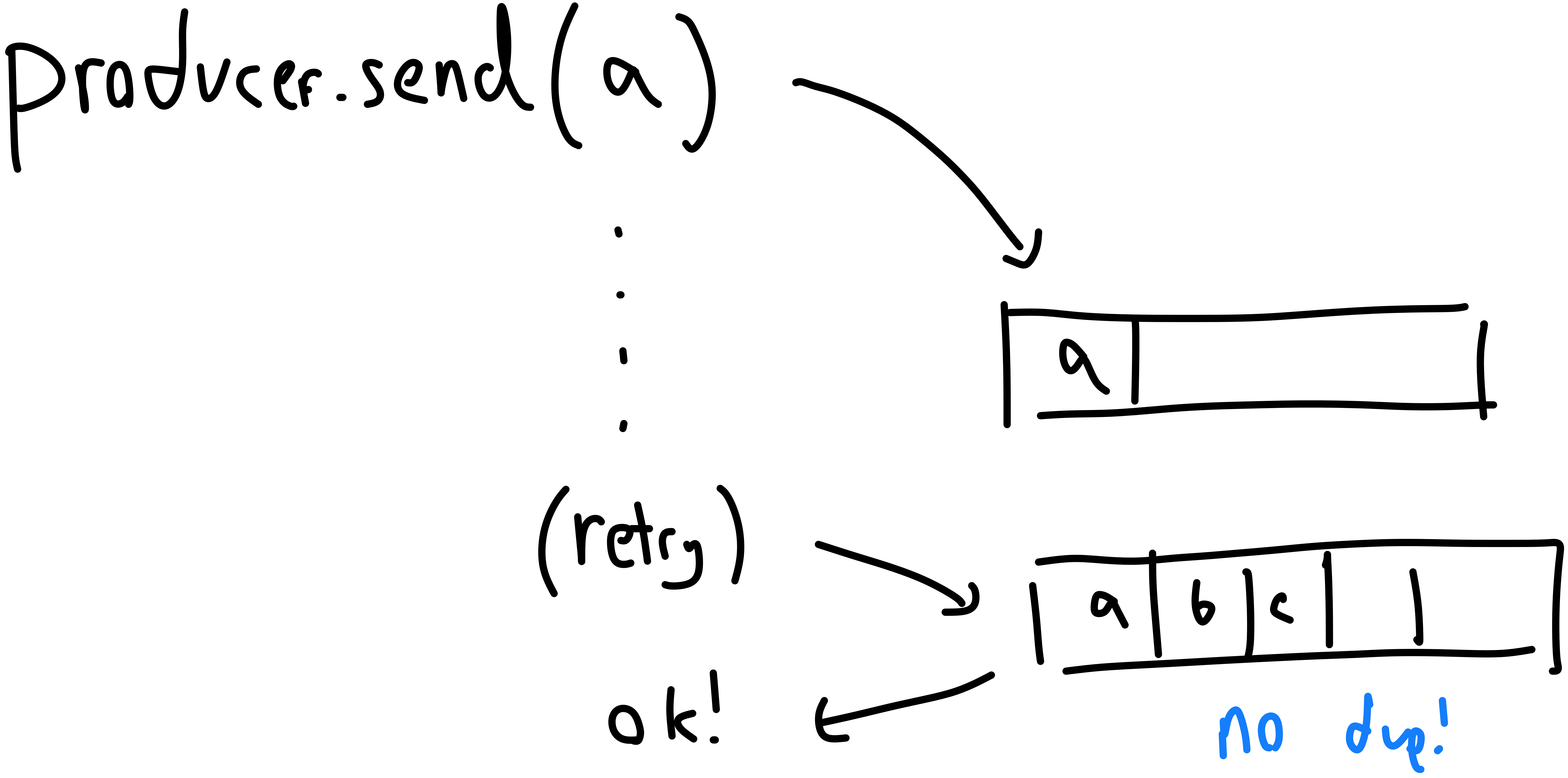


#1 Duplicate Writes by Default

enable.idempotence = true

(default in 3.0.0 client)

#1 Duplicate Writes by Default



#1 Duplicate Writes by Default

Except... we still saw dups
w/ any crash, pause, partition...

#1 Duplicate Writes by Default

Partly a client bug!

Redpanda server needs:

- enable_idempotence true
- id_allocator_replication 3

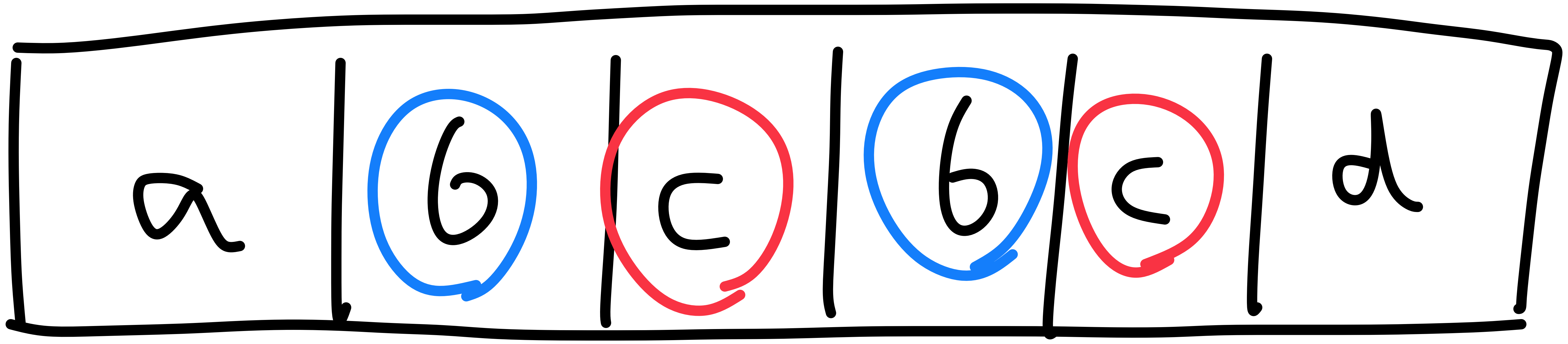
#1 Duplicate Writes by Default

Should be fixed by enabling idempotence by default

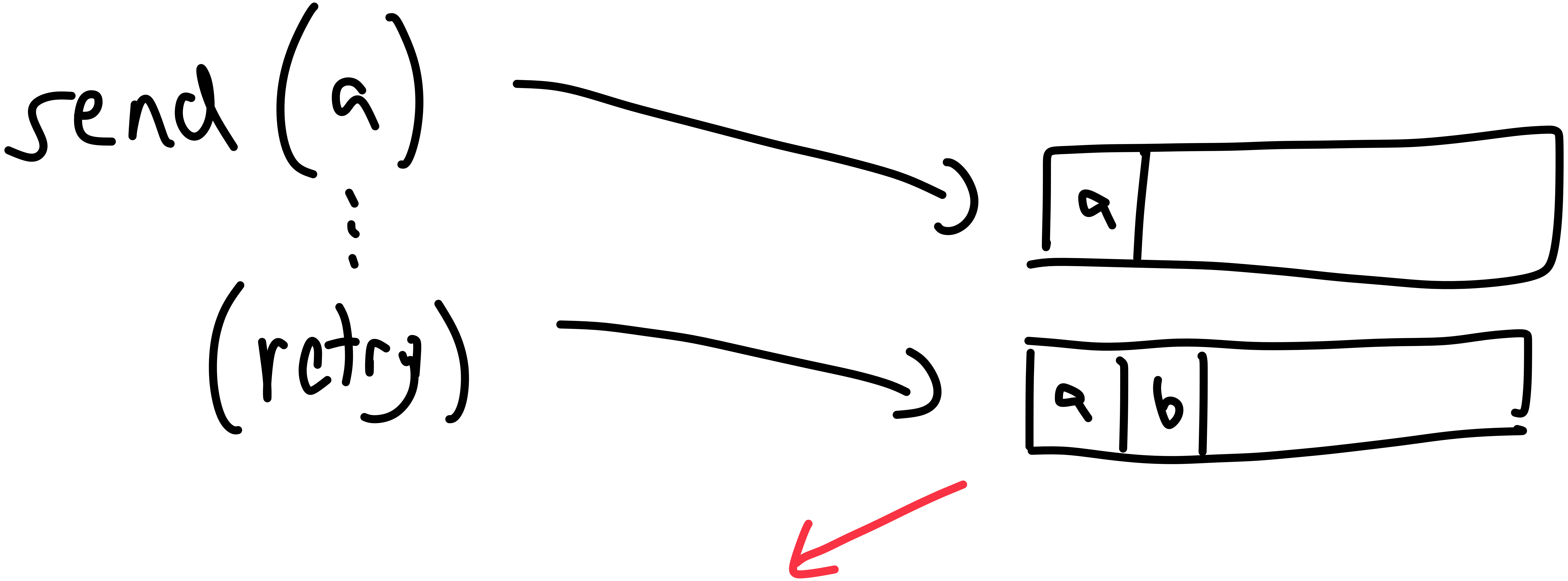
Planned for 22.1.1

#3039 Duplicate Writes Not by Default

We *still* saw duplicate messages

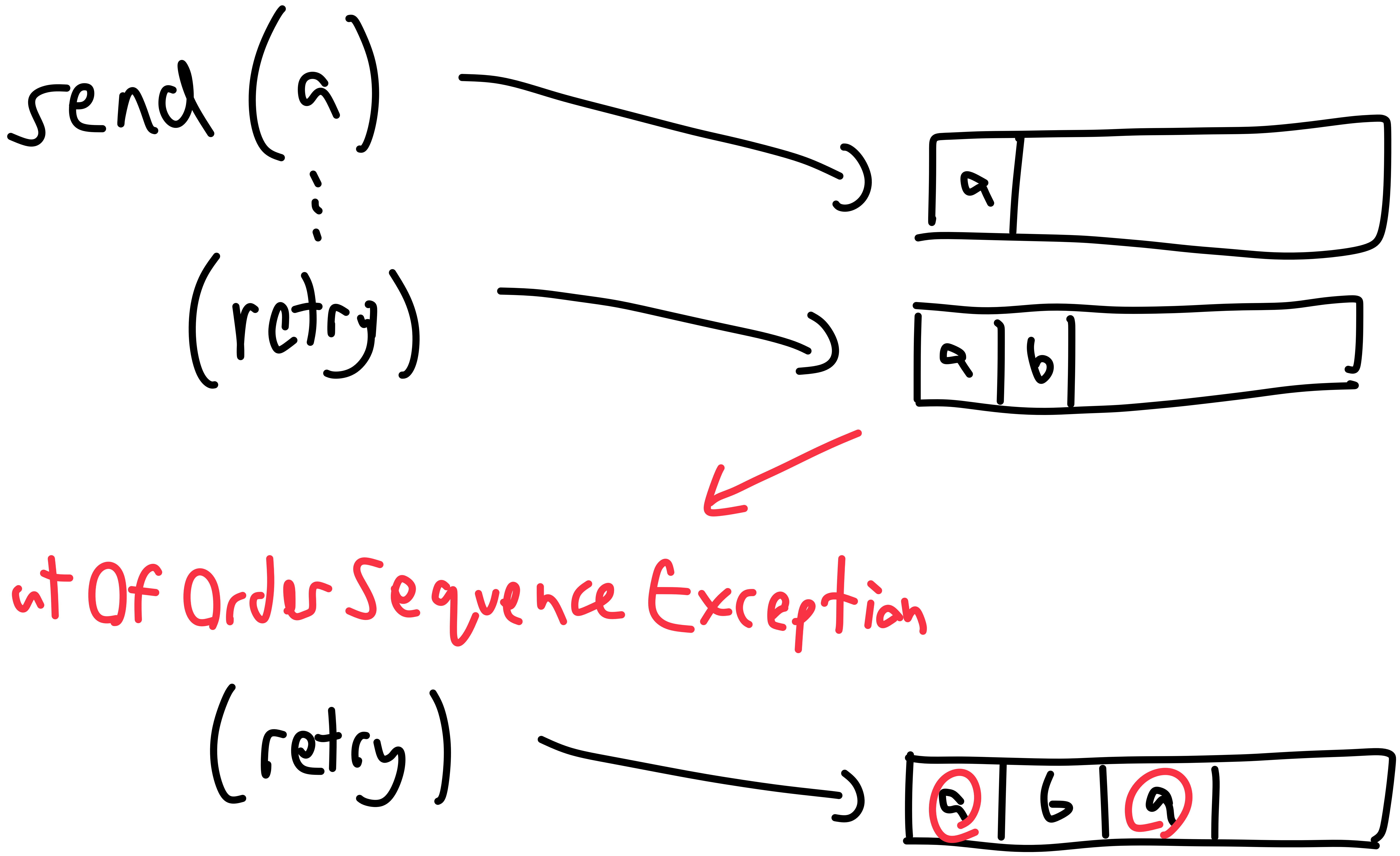


#3039 Duplicate Writes Not by Default



OutOfOrderSequenceException

#3039 Duplicate Writes Not by Default



#3039 Duplicate Writes Not by Default

- 21.10.2: Fixed by adding deduplication
- 21.10.3: Performance improvements

#3335 Assert Failure Deallocating Partitions

With membership changes...

Assert failure... allocated partitions > allocation capacity

Still investigating

#3336 Assert failure in response.partition_index

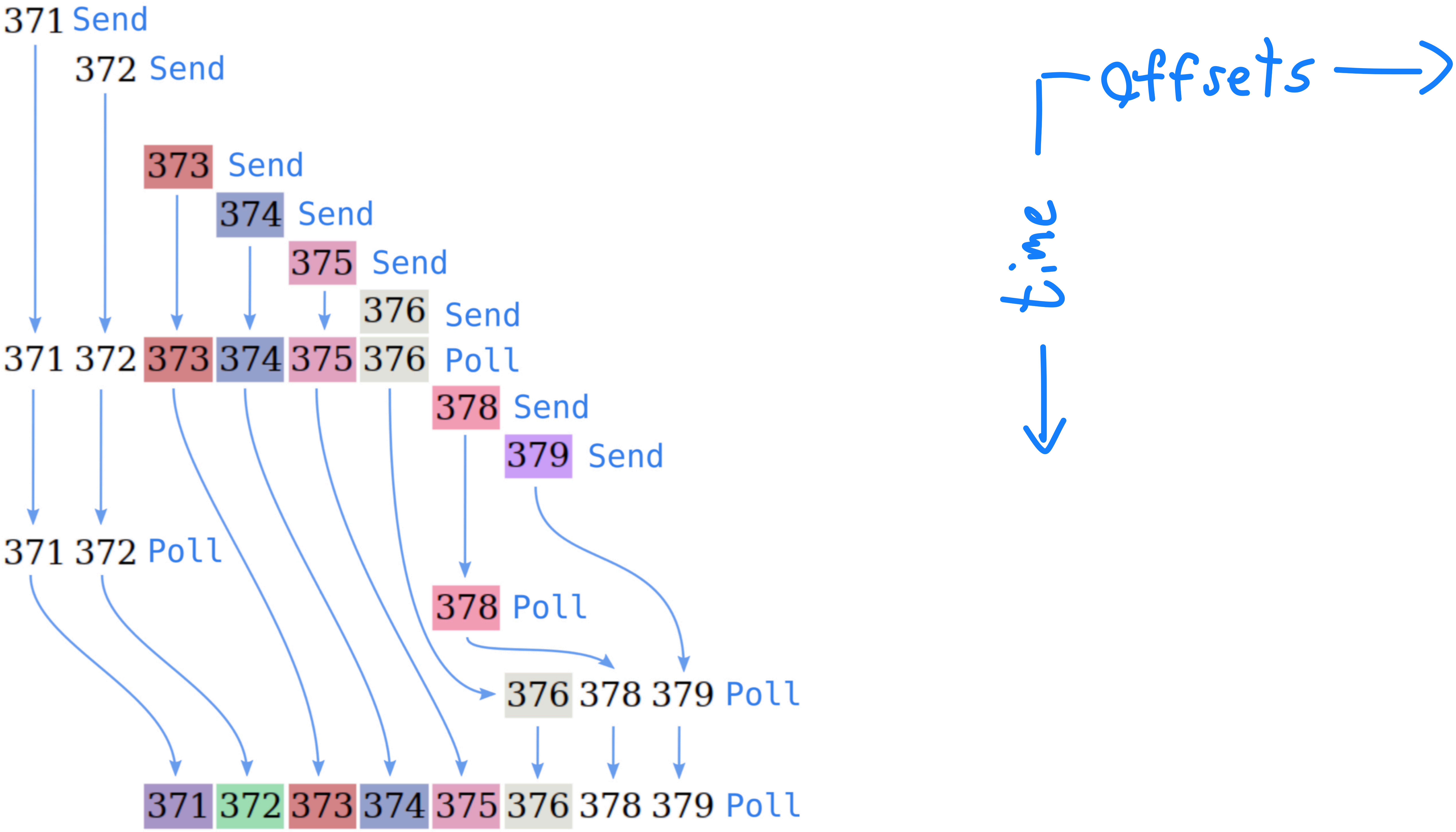
w/process crashes...

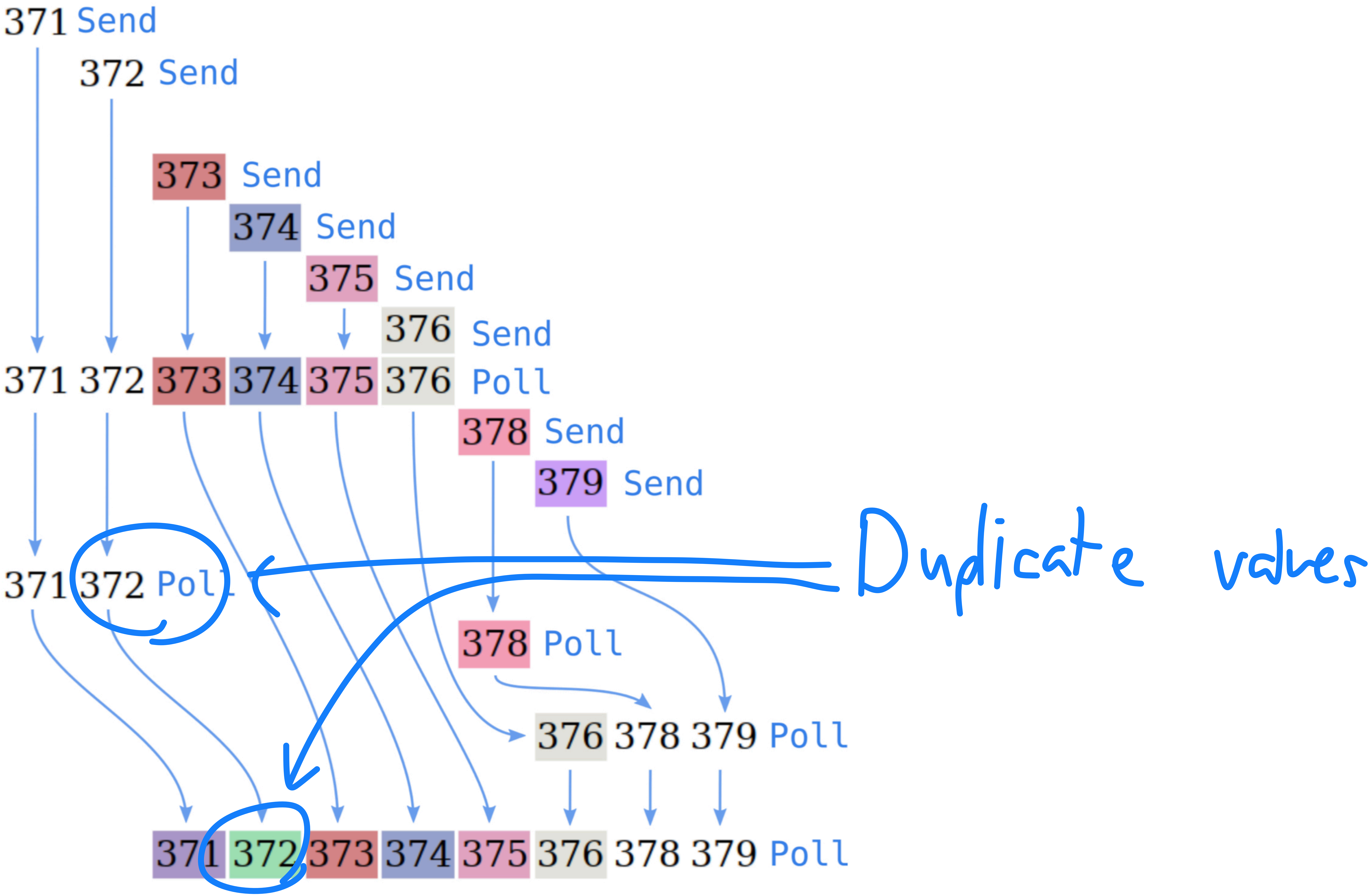
Assert failure: ... Response & current partition ids have to be the same. Current response 0, update 1

Patched → 22.1.1

#3003 Inconsistent Offsets

w/ process crashes or network partitions...


371 Send
372 Send
373 Send
374 Send
375 Send
376 Send
371 372 373 374 375 376 Poll
378 Send
379 Send
371 372 Poll
378 Poll
376 378 379 Poll
371 372 373 374 375 376 378 379 Poll
offsets
time



371 Send

372 Send

373 Send

374 Send

375 Send

376 Send

371 372 373 374 375 376 Poll

378 Send

379 Send

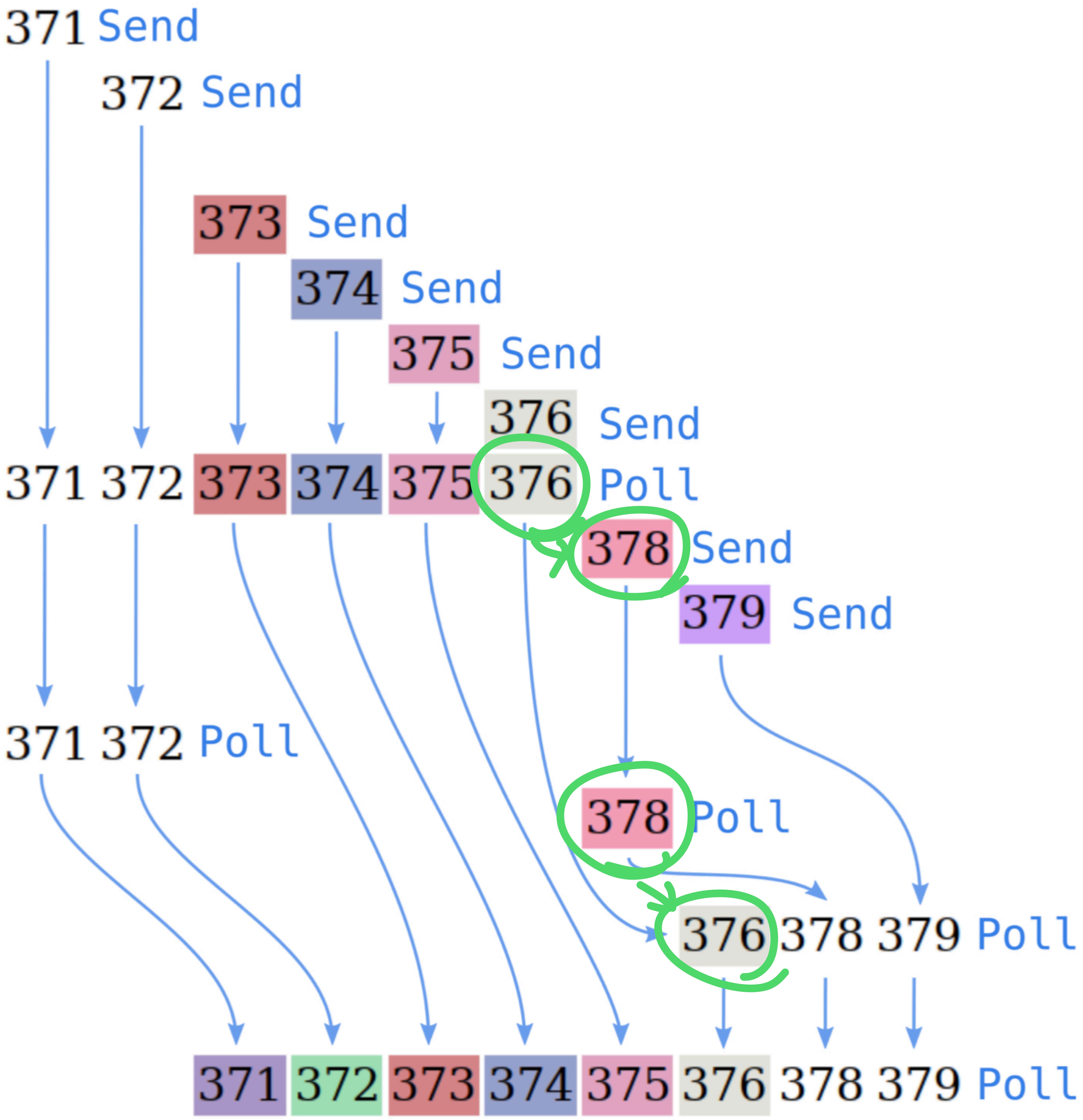
371 372 Poll

378 Poll

376 378 379 Poll

371 372 373 374 375 376 378 379 Poll

Duplicate values



371 Send

372 Send

373 Send

374 Send

375 Send

376 Send

371 372 373 374 375 376 Poll

378 Send

379 Send

371 372 Poll

378 Poll

376 378 379 Poll

371 372 373 374 375 376 378 379 Poll

Order inversion

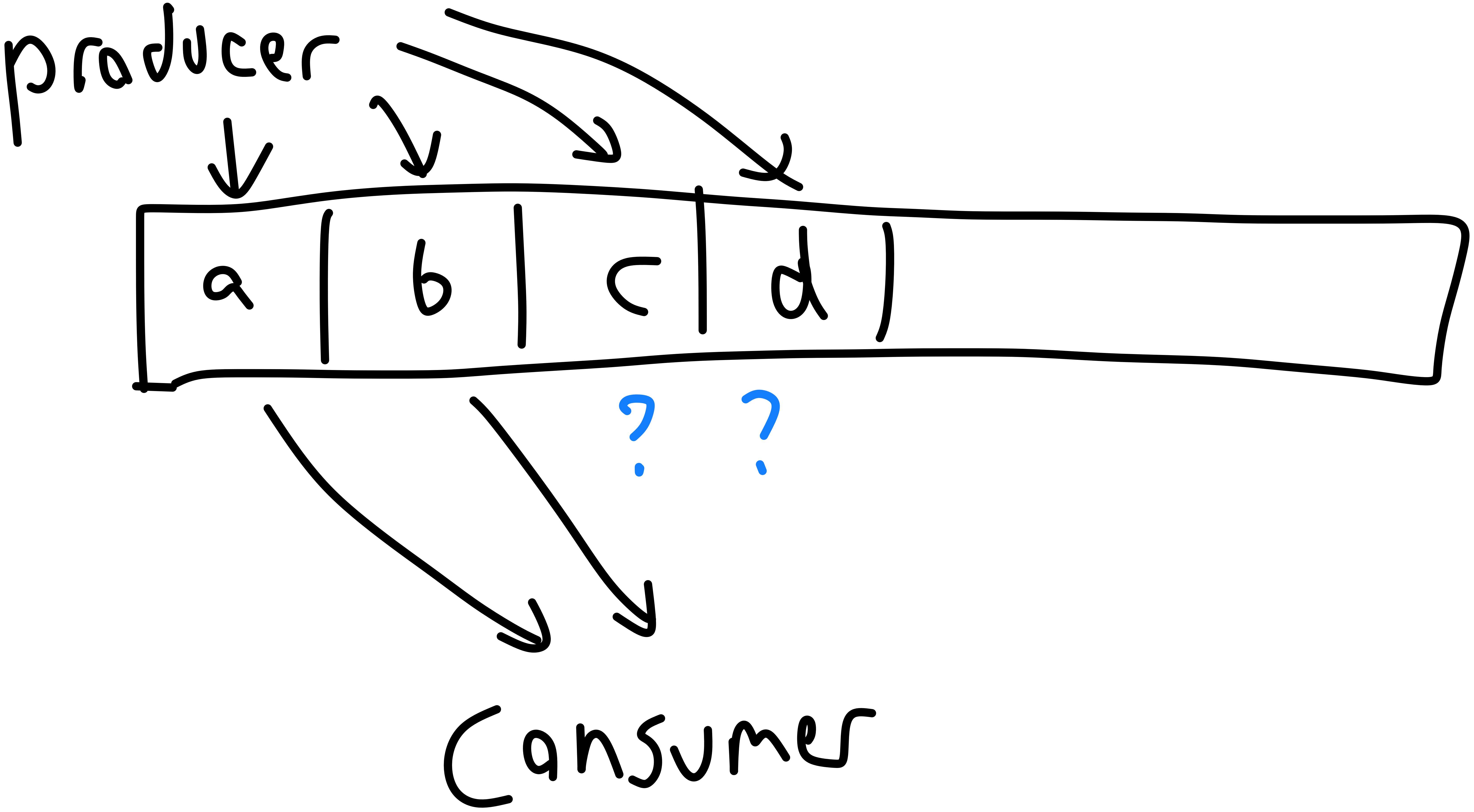
#3003 Inconsistent Offsets

Caused by Raft state machine applying uncommitted log entries

→ fixed in 21.10.3 by waiting for commit pointer to advance first

#6 Last/Stale Messages

w/process pause



#6 Lost/Stale Messages

Consumer.poll()

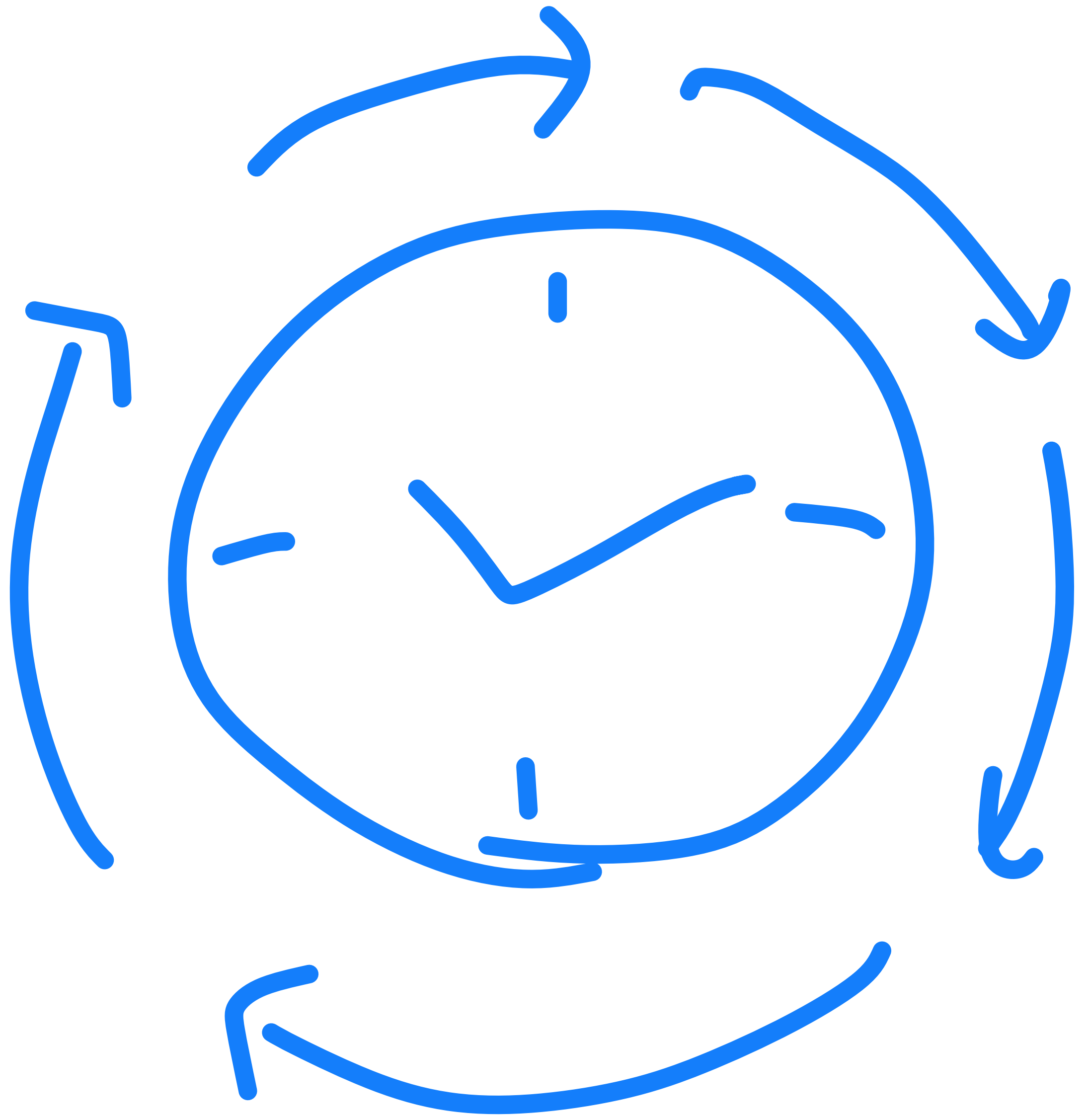
→ OK! Messages: [ ]

Consumer.poll()

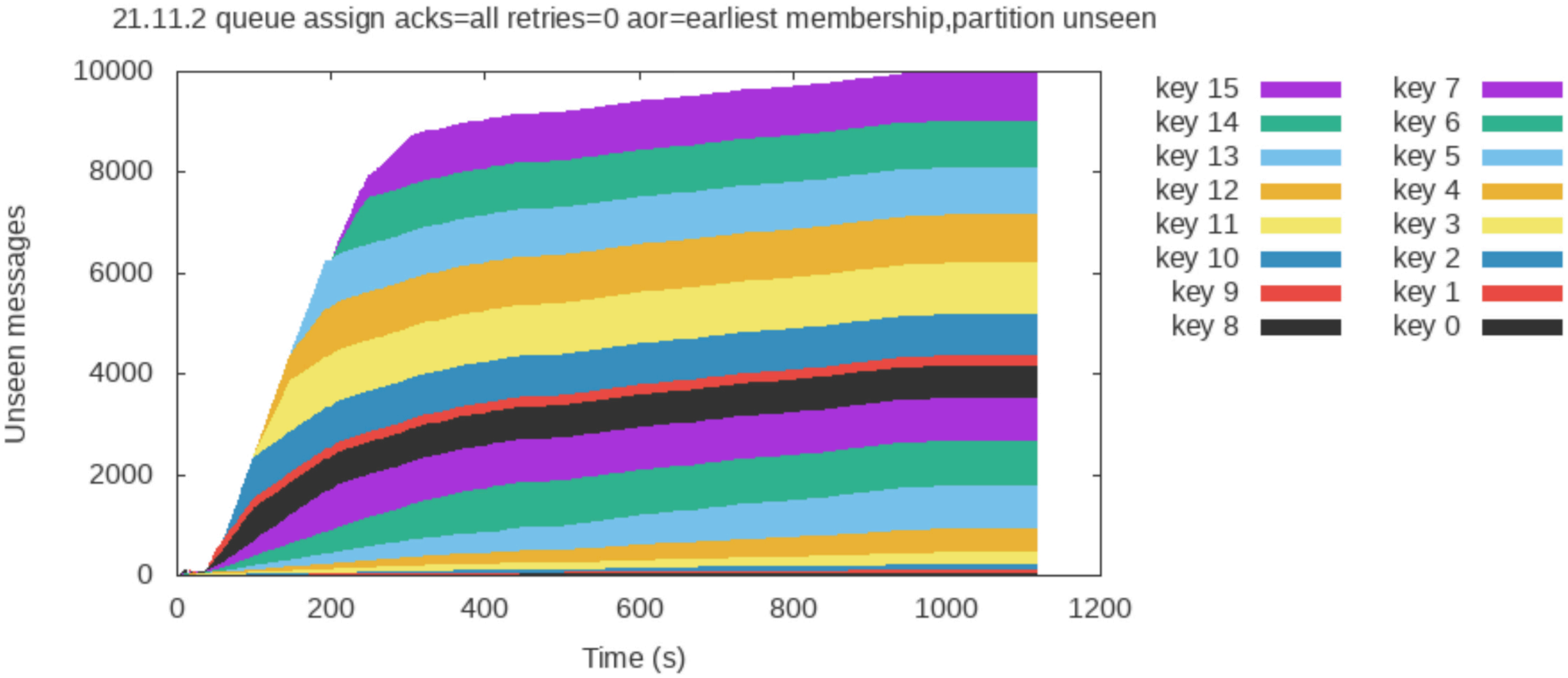
→ OK! Messages: [ ]

Consumer.poll()

→ OK! Messages: [ ]



#6 Lost/Stale Messages

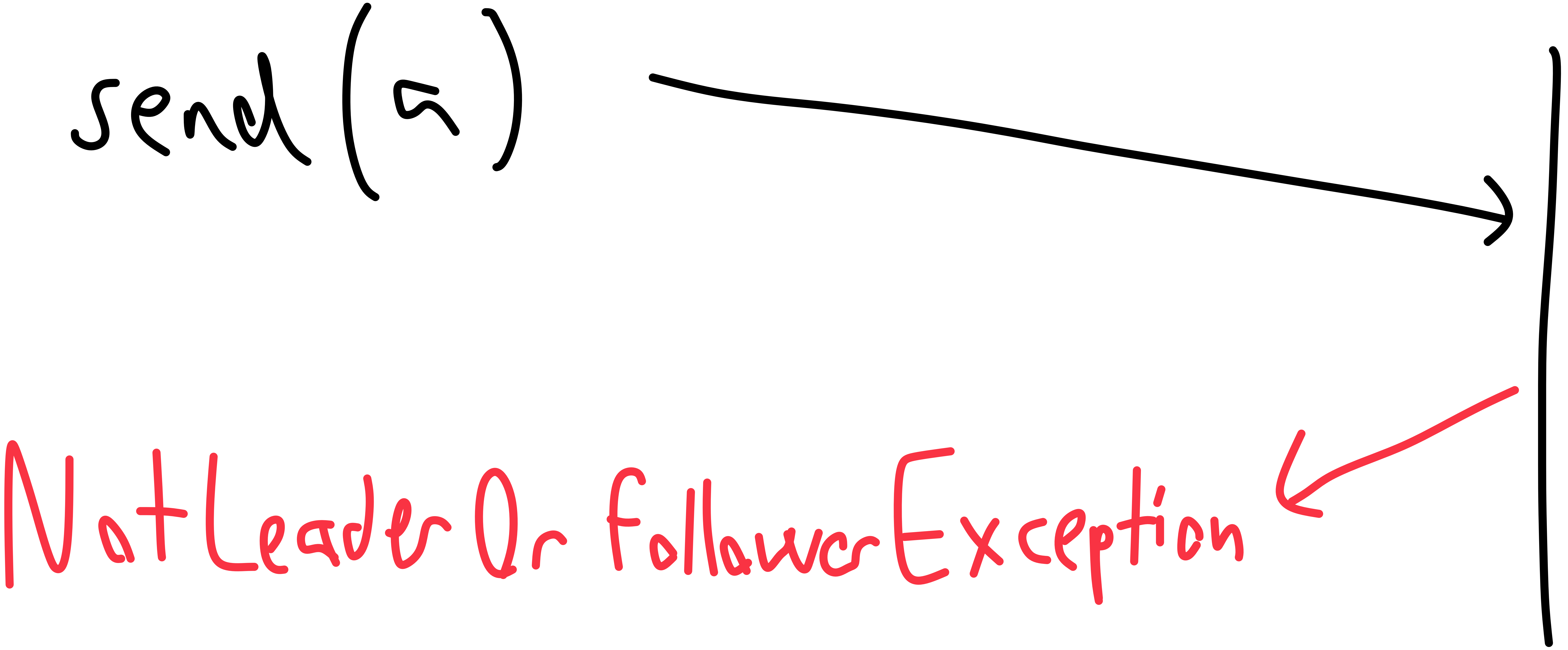


#6 Lost/Stale Messages

Still on disk... just not being delivered...

Redpanda still investigating!

#7 Aborted Read — NotLeaderOrFollowerException



#7 Aborted Read — Not Leader Or Follower Exception

"This failed, right?"

"I think so..."

#7 Aborted Read — NotLeaderOrFollowerException

Broker returns this error if a request could not be processed because the broker is not the leader or follower for a topic partition. This could be a transient exception during leader elections and reassignments. For Produce and other requests which are intended only for the leader, this exception indicates that the broker is not the current leader.

#7 Aborted Read — NotLeaderOrFollowerException

Kafka team:

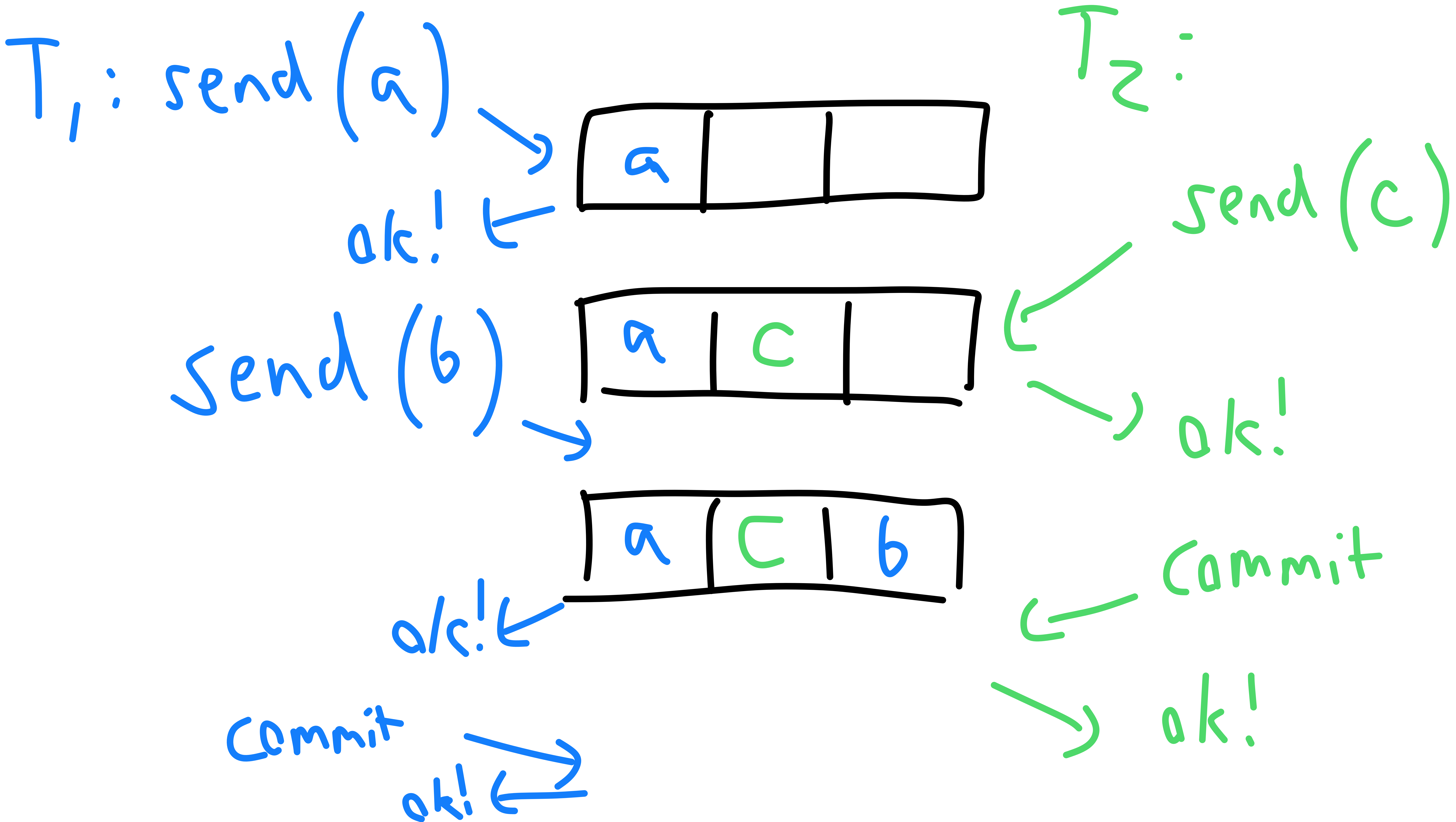
"Actually, it's indeterminate!

Could have succeeded!"

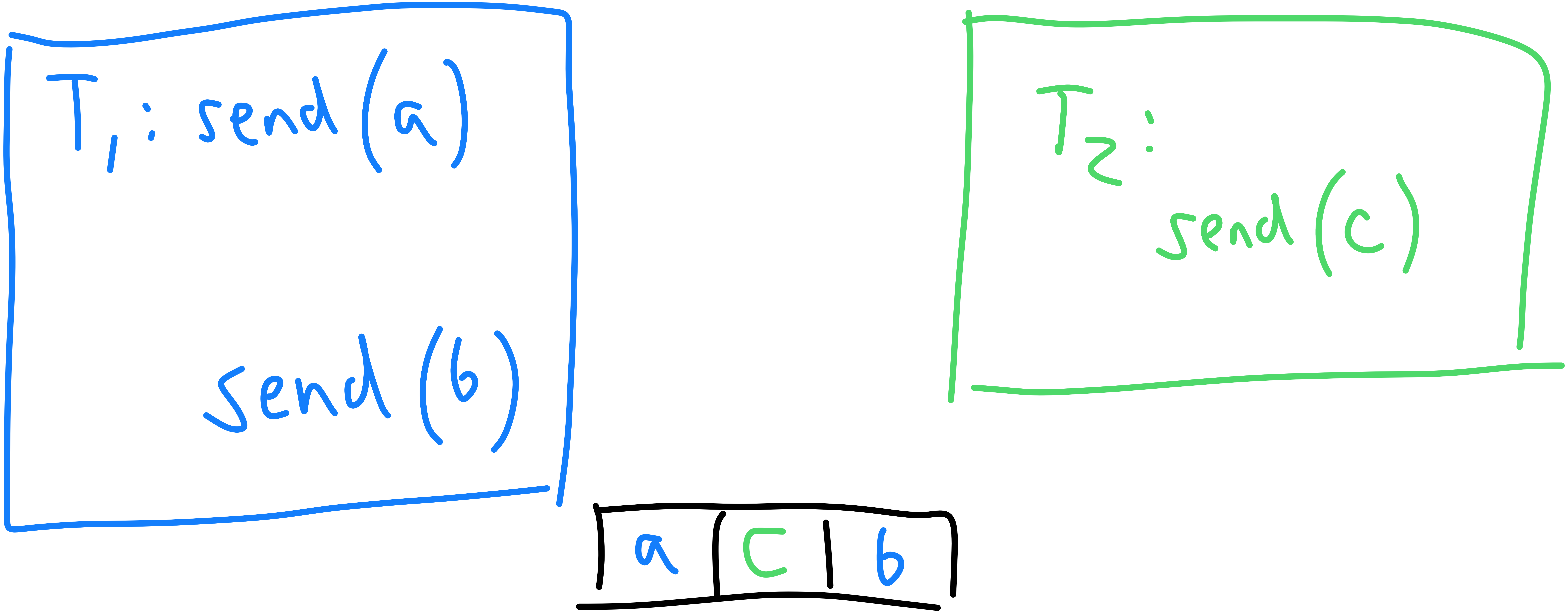
#7 Aborted Read — Not Leader Or FollowerException

Just needs docs!

#8 Write Cycles



#8 Write Cycles



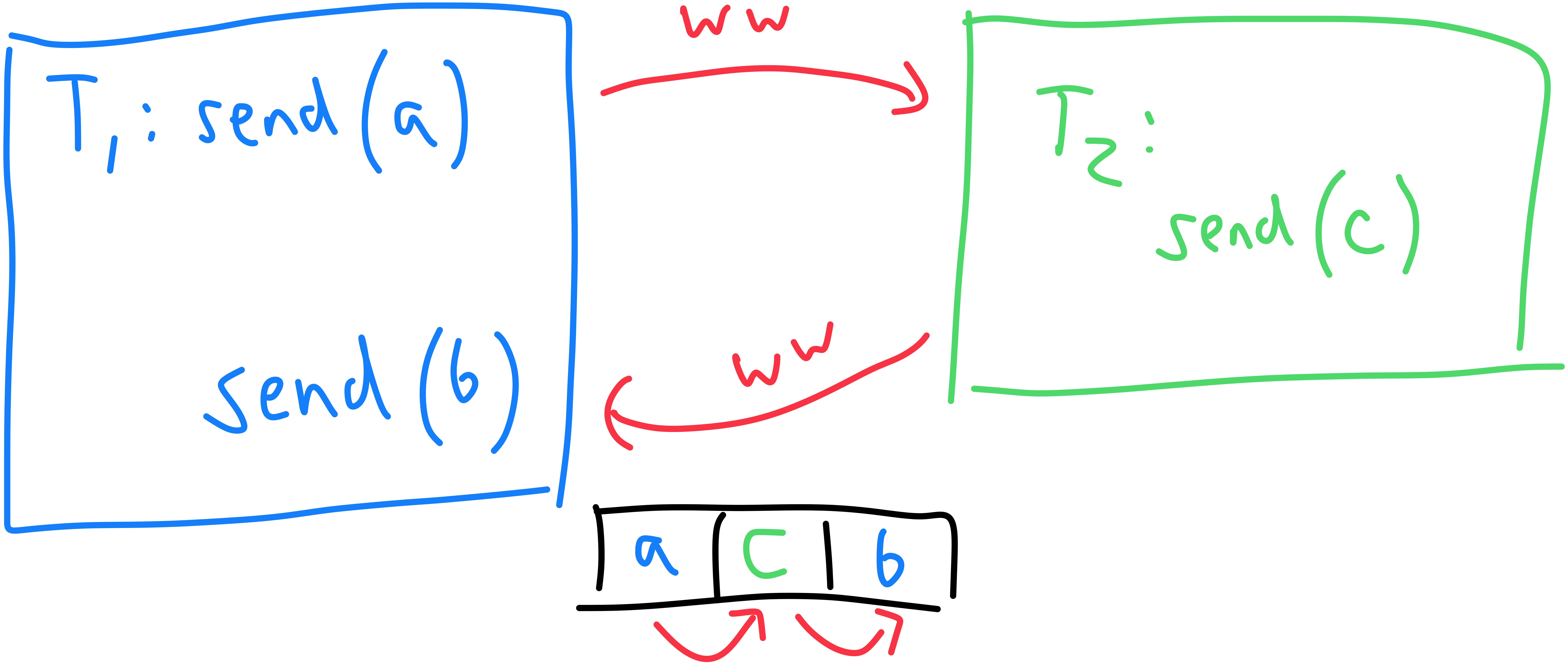
T₁: send(a)

send(b)

T₂: send(c)

a	c	b

#8 Write Cycles



#8 Write Cycles

G0 happens constantly in healthy Redpanda & Kafka clusters...

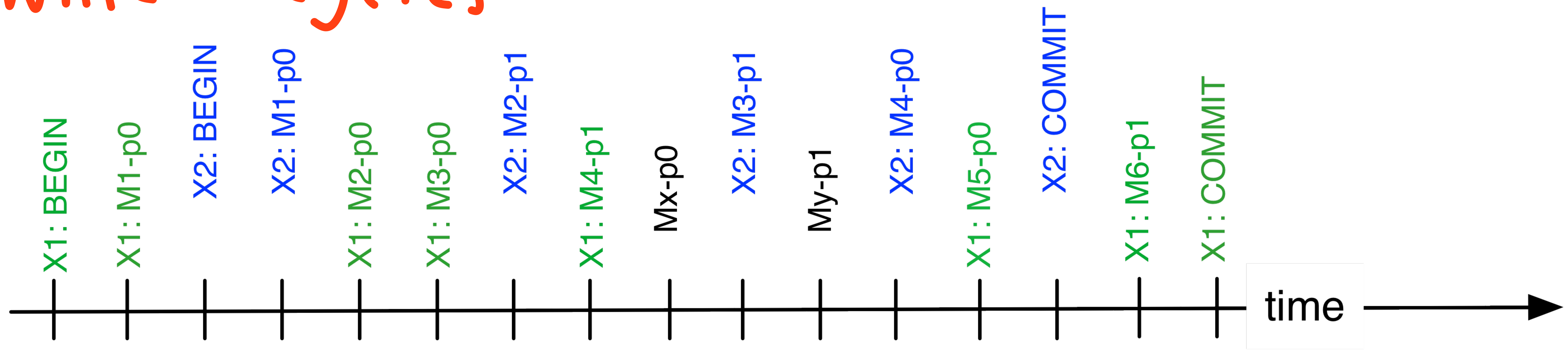
Forbidden by Adya read-uncommitted!

#8 Write Cycles

G0 happens constantly in healthy Redpanda & Kafka clusters...

Forbidden by Adya read-uncommitted!

#8 Write Cycles



Commit order: X2 < X0

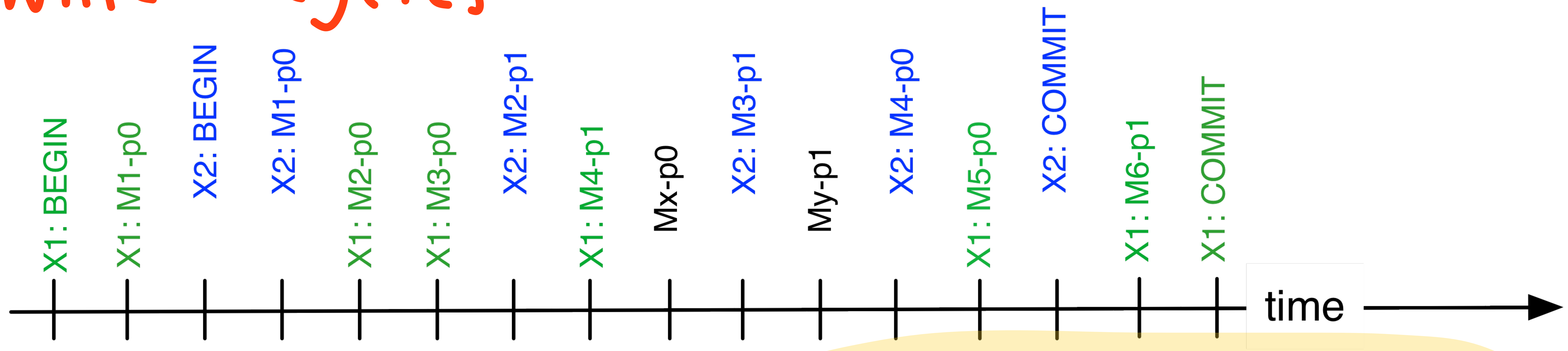
Since X2 is committed first, each partition will expose messages from X2 before X1.

Consumer processing order

p0: Mx-p0, X2: M1-p0, X2: M4-p0, X1: M1-p0, X1: M2-p0, X1: M3-p0, X1: M5-p0

p1: My-p1, X2: M2-p1, X2: M3-p1, X1: M4-p1, X1: M6-p1

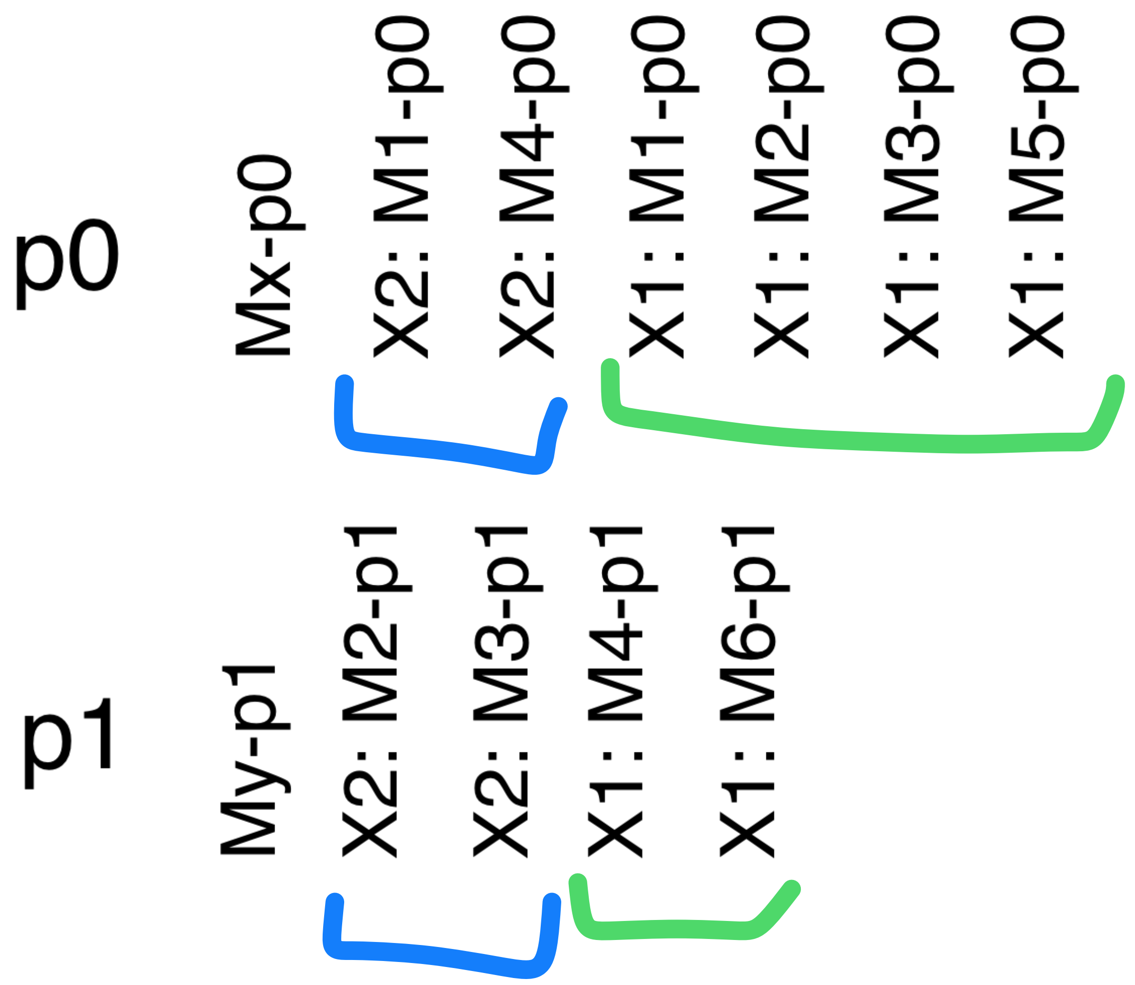
#8 Write Cycles



Commit order: X2 < X0

Since X2 is committed first, each partition will expose messages from X2 before X1.

Consumer processing order



#8 Write Cycles

Liiiiieeees

Since X2 is committed first each partition will expose messages from X2 before X1

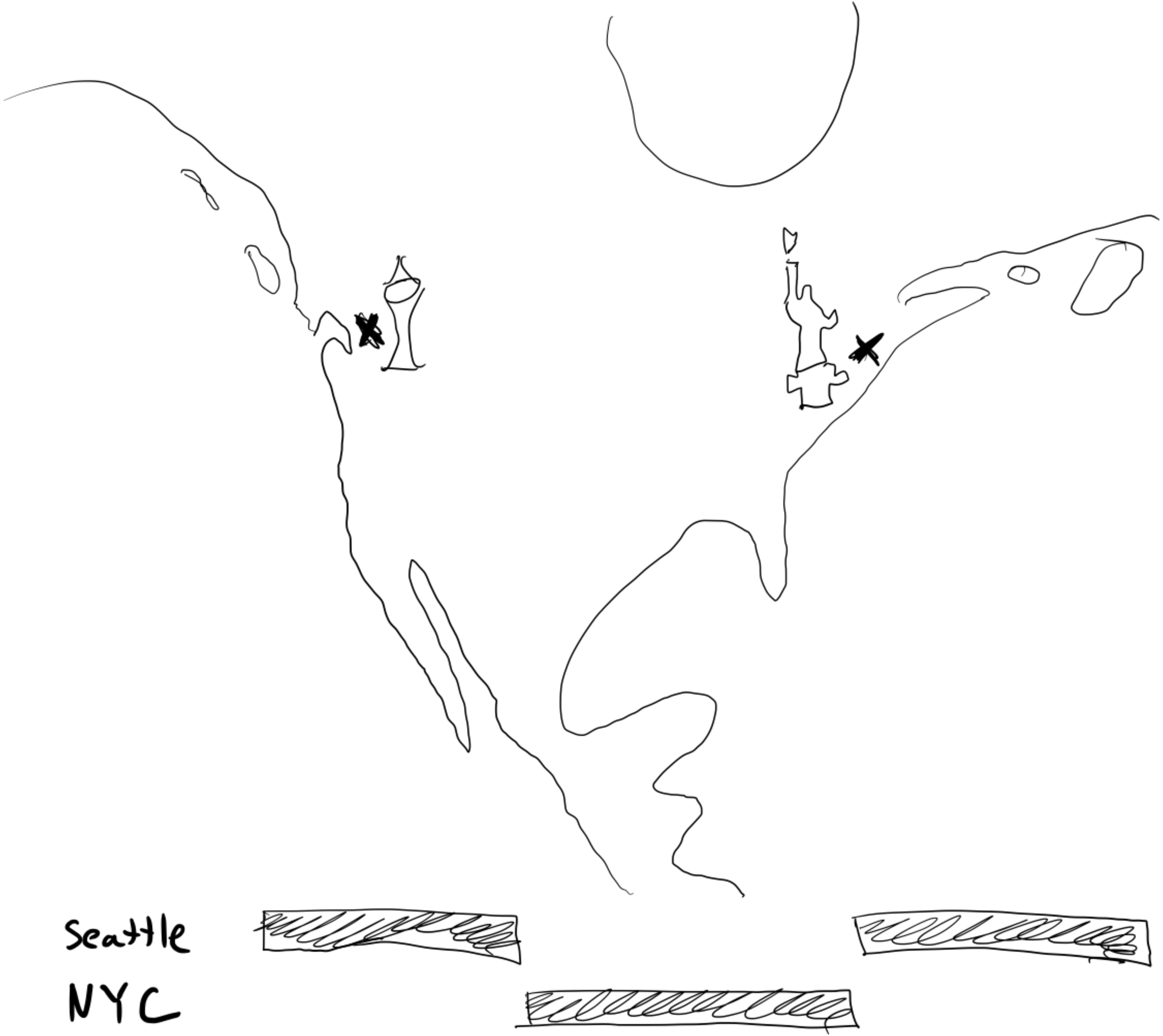
"Transactional Messaging in Kafka"

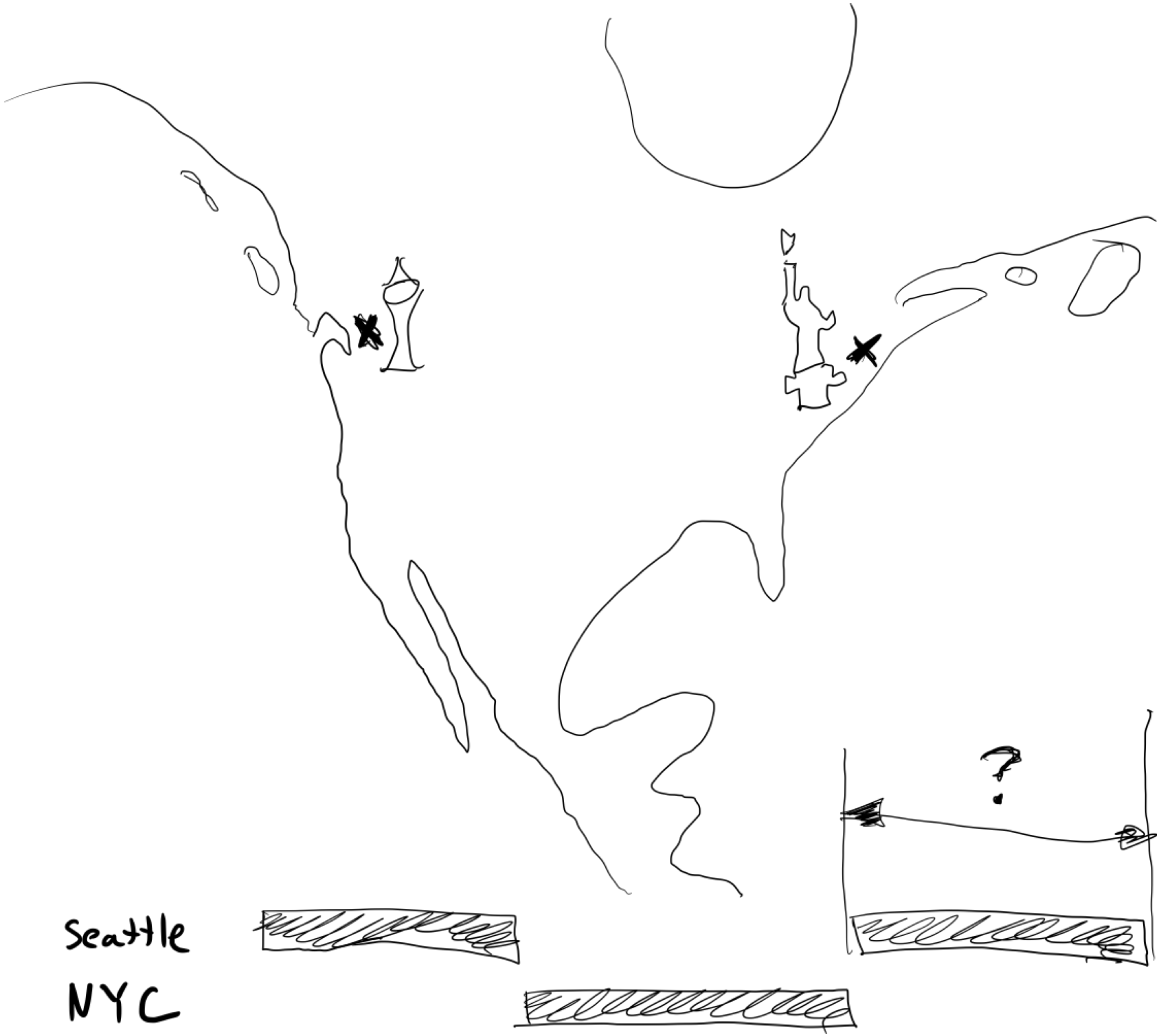
we have no strong evidence that applications can benefit from the commit order option, we opted for not implementing it

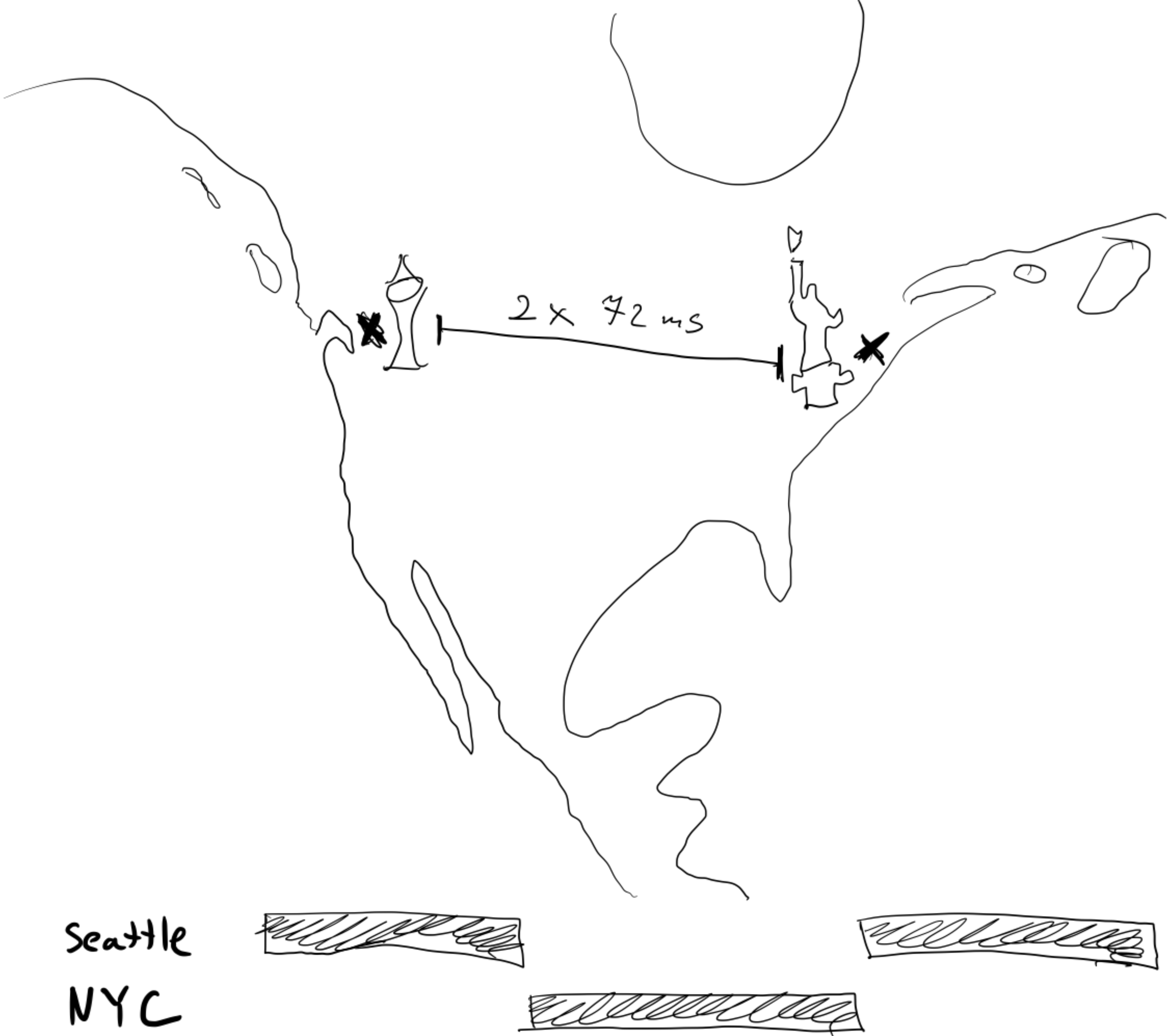
"Exactly Once Delivery and Transactional Messaging in Kafka"

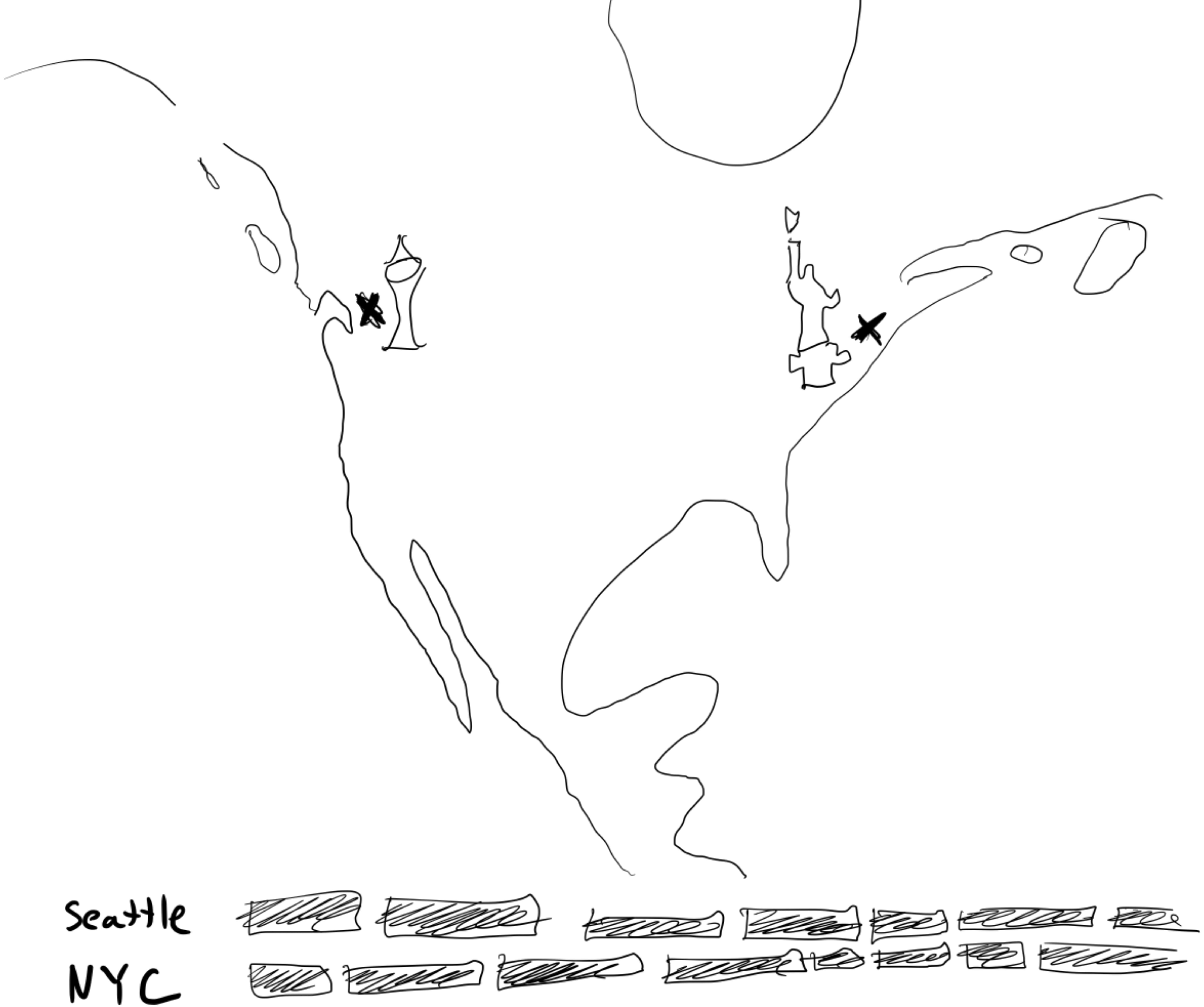
In theory, theory and practice are the same. In practice, they are not

Kafka protocol
permits G0
(tx interleaving)


Seattle
NYC


?
Seattle
NYC


2 x 72 ms
Seattle
NYC


Seattle
NYC

2013 IEEE 32nd International Symposium on Reliable Distributed Systems

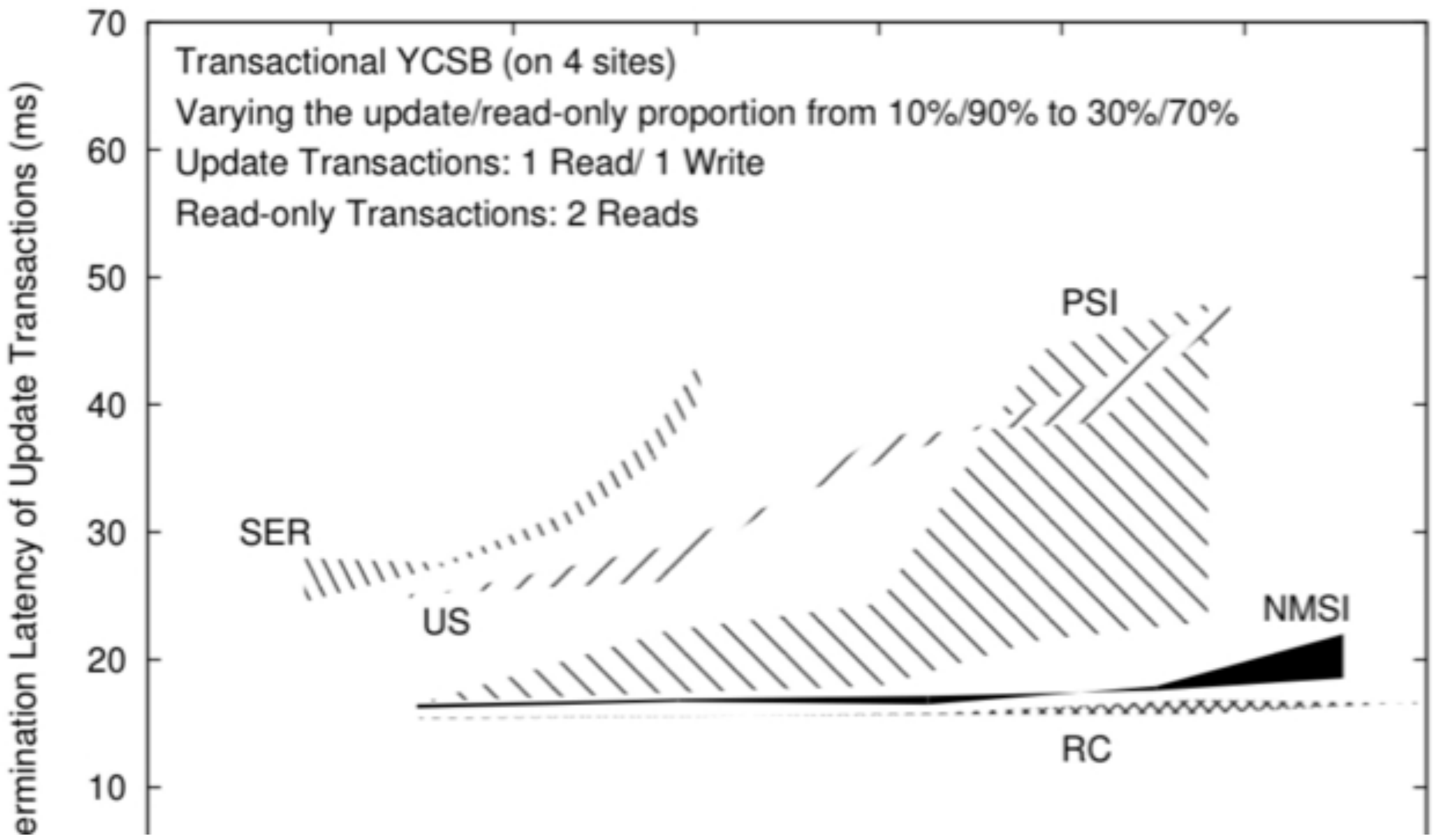
Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems

Masoud Saeida Ardekani
Université Pierre-et-Marie Curie (UPMC-LIP6)
Paris, France
masoud.saeida-ardekani@lip6.fr

Pierre Sutra
University of Neuchâtel
Neuchâtel, Switzerland
pierre.sutra@unine.ch

Marc Shapiro
INRIA & UPMC-LIP6
Paris, France
http://lip6.fr/Marc.Shapiro/

***Abstract*—Modern cloud systems are geo-replicated to improve application latency and availability. Transactional consistency is essential for application developers; however, the corresponding concurrency control and commitment protocols are costly in a geo-replicated setting. To minimize this cost, we identify the following essential scalability properties: *(i)* only replicas updated by a transaction T make steps to execute T; *(ii)* a read-only transaction never waits for concurrent transactions and always commits; *(iii)* a transaction may read object versions committed after it started; and *(iv)* two transactions synchronize with each other only if their writes conflict. We present Non-Monotonic Snapshot Isolation (NMSI), the first strong consistency criterion to allow implementations with all four properties. We also present a practical implementation of NMSI called Jessy, which we**



1) all-or-nothing for multi-partitions writes

2) atomic consume-transform-produce loop

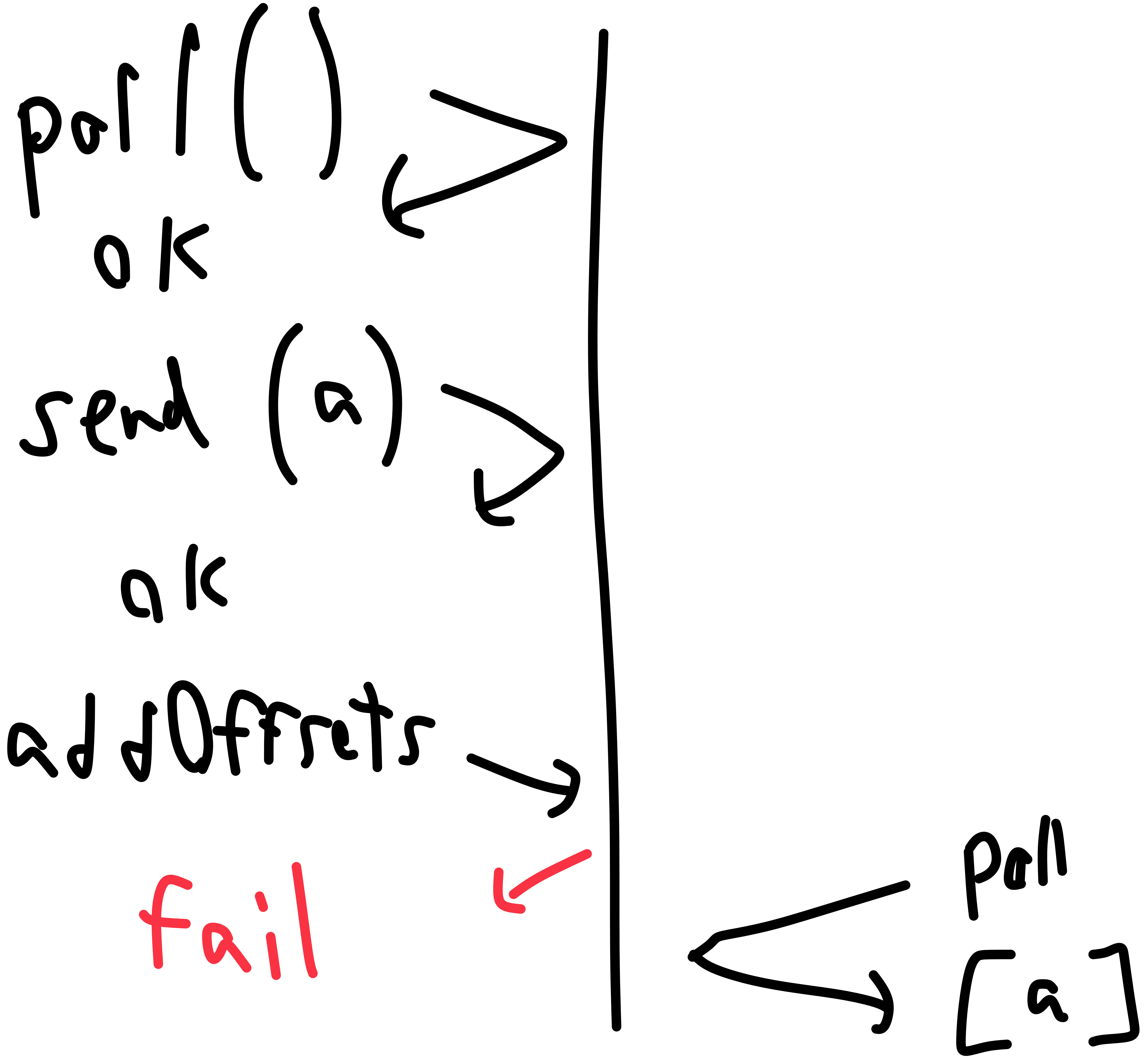
3) ~~consistent snapshots~~

#8 Write Cycles

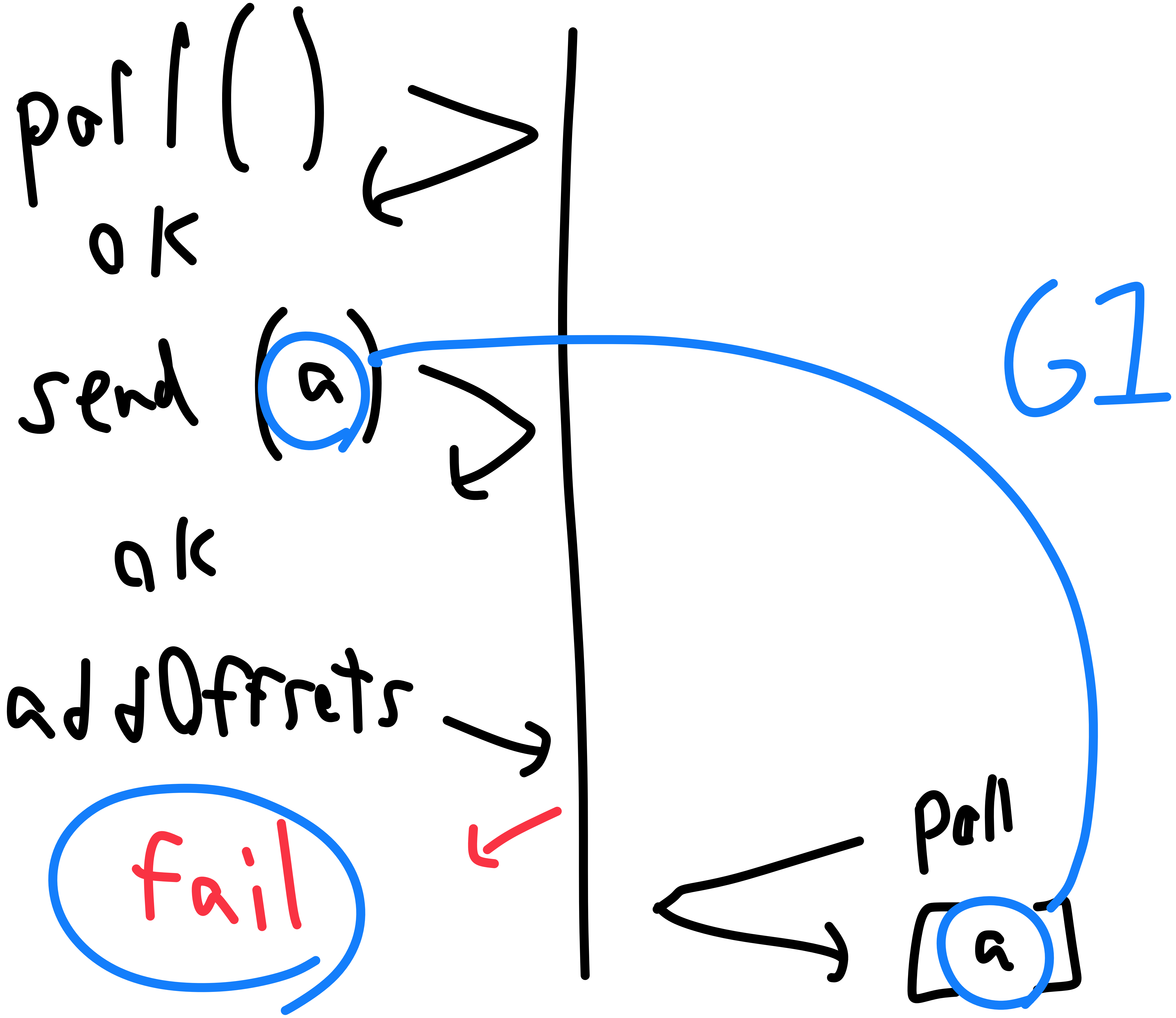
This is just how Kafka/ Redpanda txns work

→ Needs documentation

#3036 Aborted Reads & Circular Information Flow



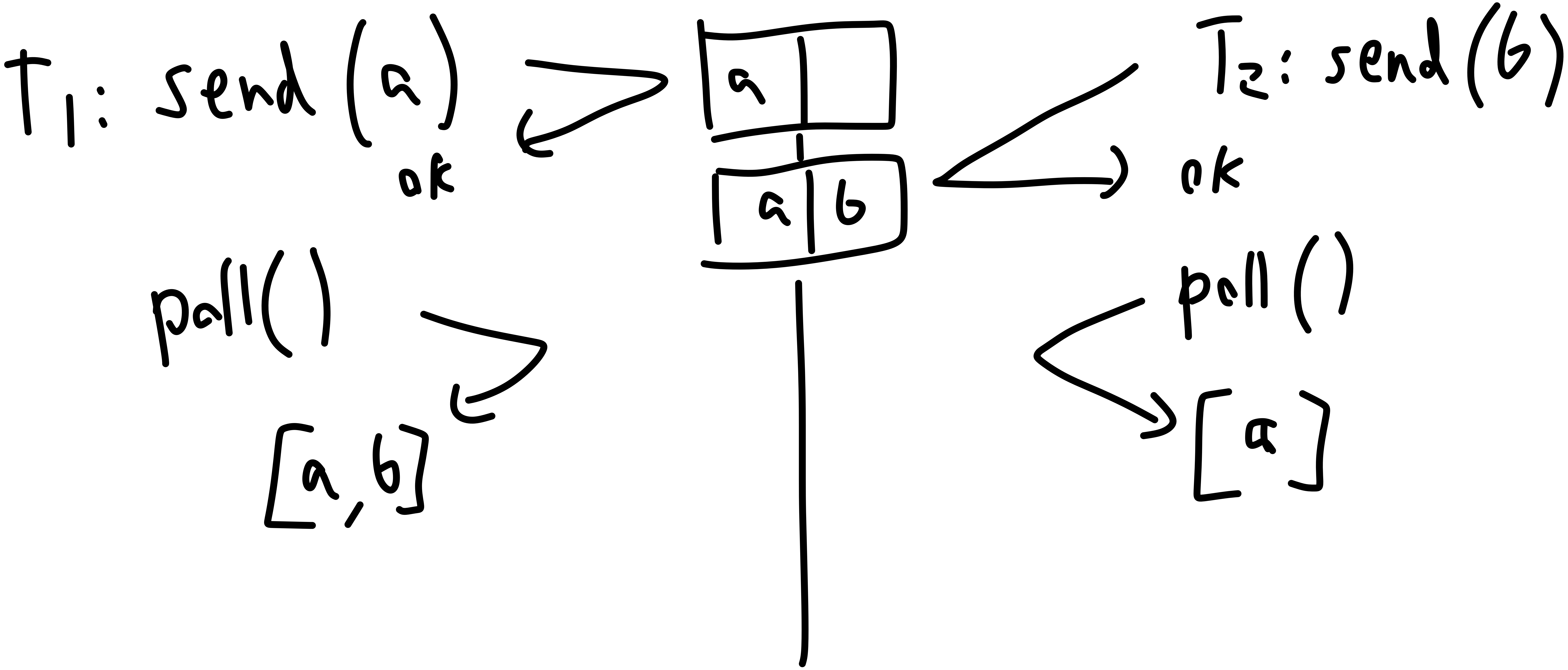
#3036 Aborted Reads & Circular Information Flow



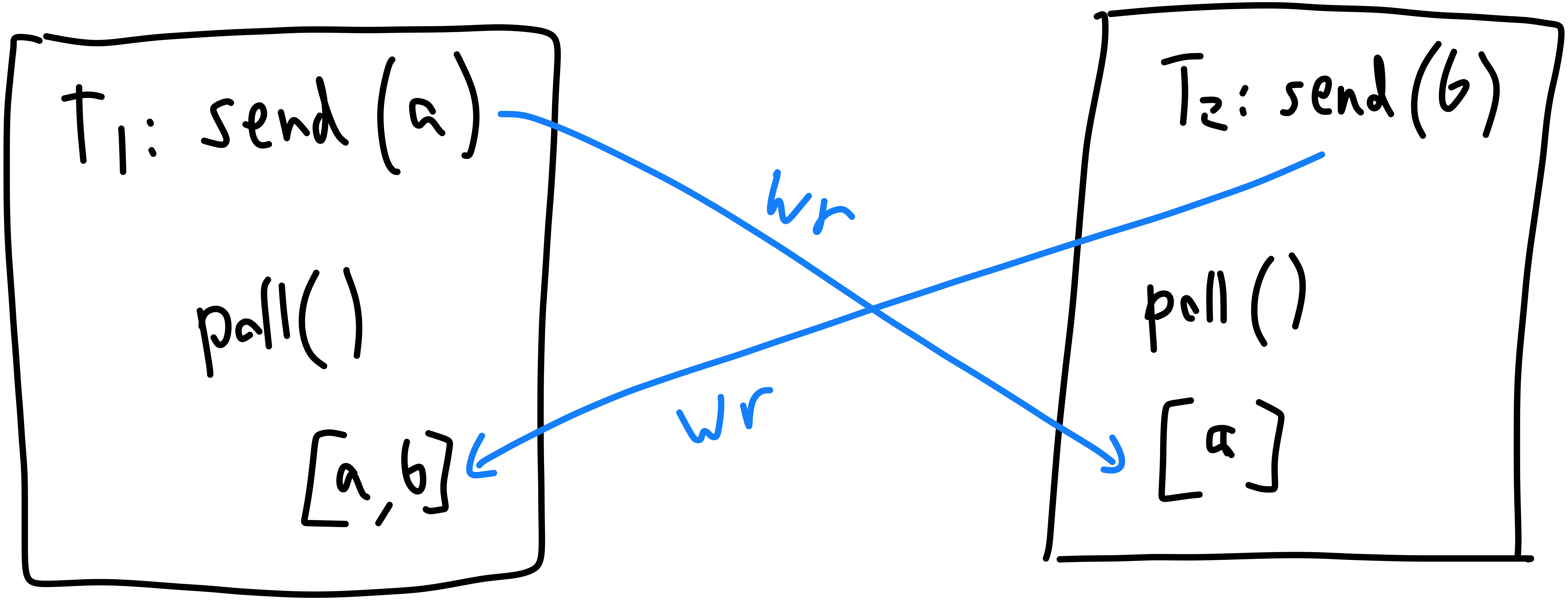
G1a: Aborted Read

Happened often in healthy clusters!

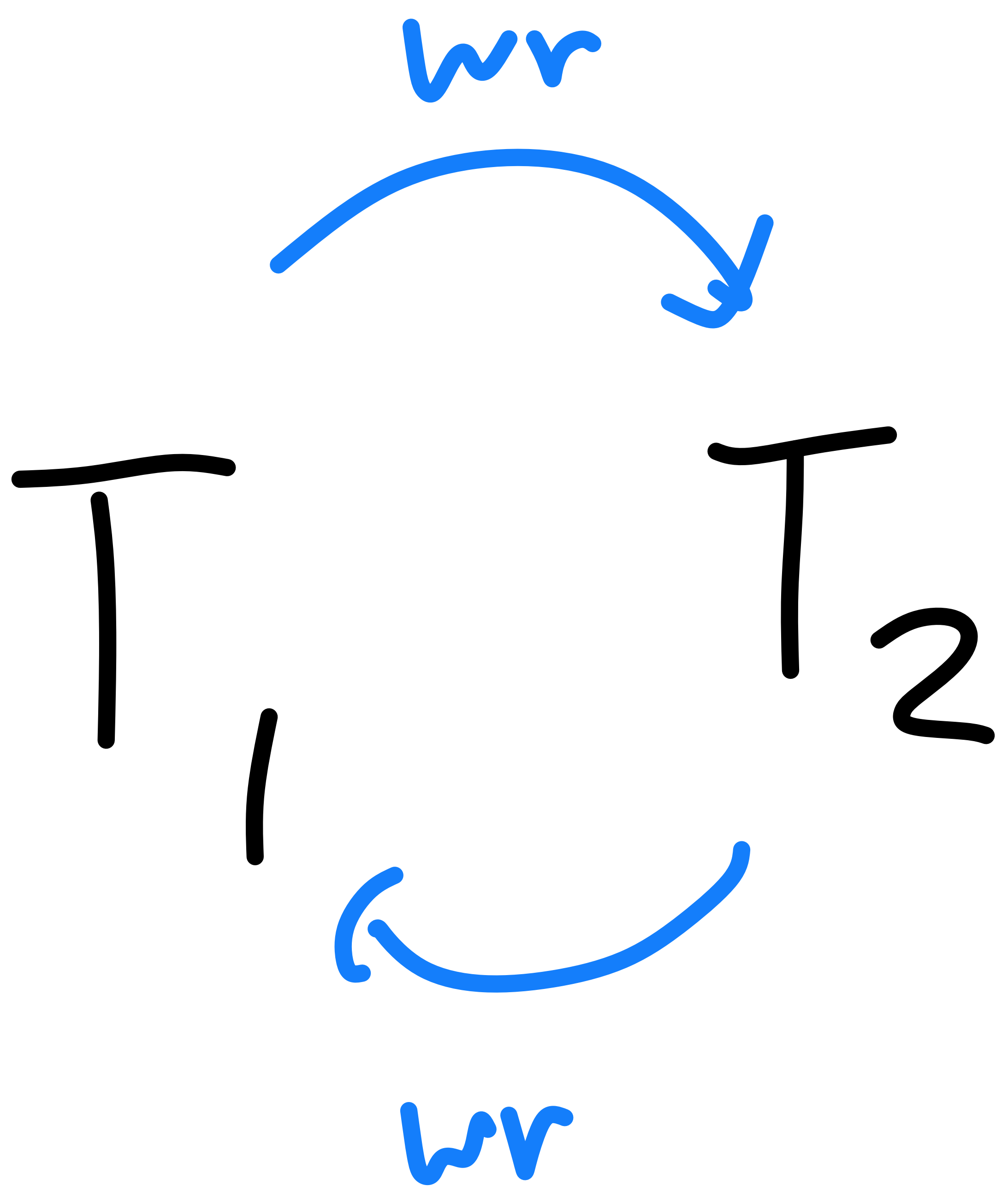
#3036 Aborted Reads & Circular Information Flow



#3036 Aborted Reads & Circular Information flow



#3036 Aborted Reads & Circular Information Flow



G1c: Circular
Information
Flow

Illegal in Adya Read Committed!

#3036 Aborted Reads & Circular Information Flow

- Off-by-one error allowed LSO to advance beyond committed offsets
- #3232 accidentally aborted more txns than necessary

Fixed in 21.10.2

#10 Internal Non-monotonic Polls

```
beginTxn()
poll()
poll()
addOffsetsToTxn(...)
commitTxn()
```

#10 Internal Non-monotonic Polls

```
beginTxn()
poll() → [a, b, c]
poll() → [   b, c, d]
addOffsetsToTxn(...)
commitTxn()
```

#10 Internal Non-monotonic Polls

- Caused by consumer rebalance
- Clients must register rebalance listeners to detect potential non-monotonicity in transactions

#3616-a Aborted Read w/ InvalidTxnStateException

```
beginTxn()
send(a)
commitTxn() -> InvalidTxnStateException
```

#3616-a Aborted Read w/ InvalidTxnStateException

beginTxn()

send(a)

commitTxn() → InvalidTxnStateException

⋮

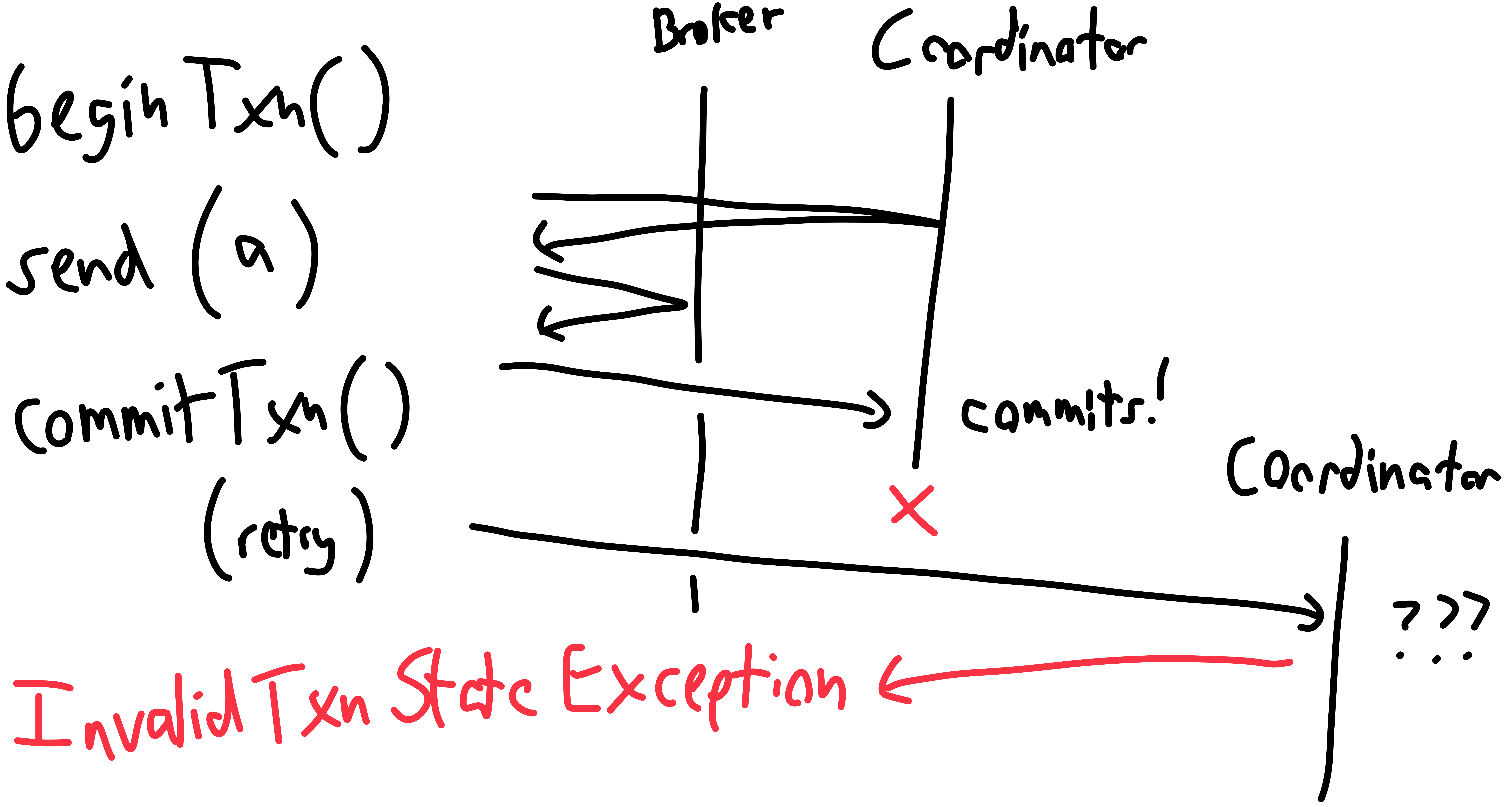
poll() → [a]

#3616-a Aborted Read w/ InvalidTxnStateException

Pause, crash, or partition

led to G1a

#3616-a Aborted Read w/ InvalidTxnStateException



#3616-a Aborted Read w/ InvalidTxnStateException

Fixed by returning UnknownServerError

January 21, 2022

↳ Released in 21.11.15

#3616-6 Lost Transactional Writes

```
beginTxn()
send(a)          -> offset 2
commitTxn()      -> ok
```

#3616-6 Last Transactional Writes

```
beginTxn()
send(a)      -> offset 2
commitTxn() -> ok
```

poll() ->

b	c		d
0	1	2	3

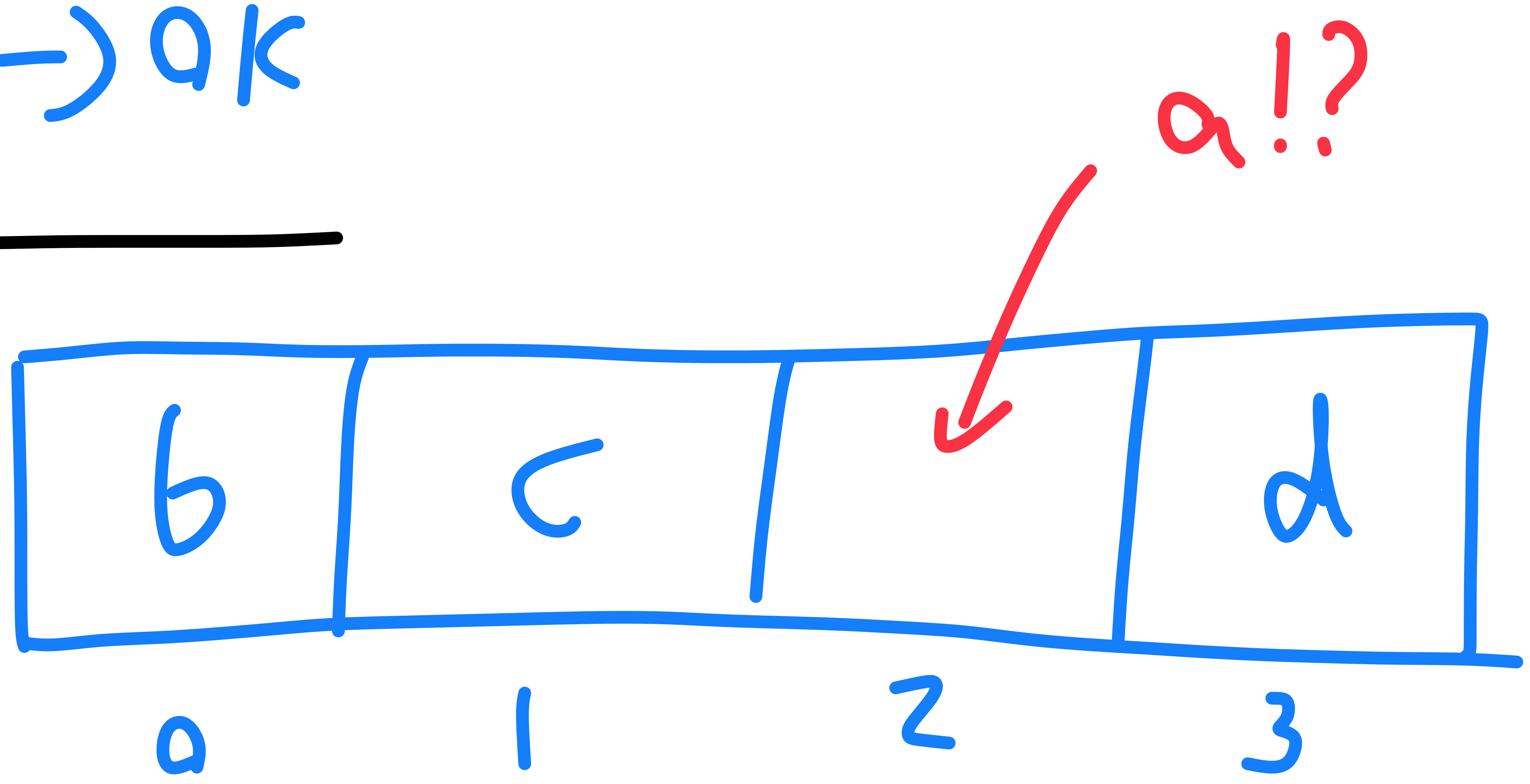
#3616-6 Last Transactional Writes

beginTxn()

send(a) -> offset 2

commitTxn() -> OK

poll() ->



#3616-6 Lost Transactional Writes

Happened in healthy clusters

on 21.11.2

#3616-6 Last Transactional Writes

```
lock.acquire()
if (! log.sync()) {
    return;
}
do_x();
do_y();
lock.release()
```

Critical section

#3616-6 Last Transactional Writes

```
lock.acquire()
if (! log.sync()) {
    return;
}
do_x();
do_y();
lock.release()
```

Critical section

Node loses, then regains leadership — other leader did things in our critical section!

#3616-6 Last Transactional Writes

```
lock.acquire()
if (! log.sync()) {
    return;
}
term = _insync_term
do_x(term);
do_y(term);
lock.release()
```

Critical section

#3616-6 Lost Transactional Writes

Fixed January 21, 2022

↳ Released in 21.11.15

Discussion

#	Issue	Fault	Resolved
1	Duplicate writes by default	$P \lor C \lor \text{Part}$	22.1.1
3039	Duplicate writes w/ idempotence	$P \lor C \lor \text{Part}$	21.10.3
3335	Assert crash deallocating partitions	M	
3336	Assert crash w/ partition IDs	C	22.1.1
3003	Inconsistent offsets	$C \lor \text{Part}$	21.10.3
6	Lost/stale messages	$P \lor C \lor \text{Part}$	
7	Aborted read – NotLeaderOrFollower	$M \land P$	
8	Write cycles	$\emptyset$	
3036	Aborted read/Circular information flow	$\emptyset$	21.10.2
10	Internal non-monotonic polls	$\emptyset$	
3616-a	Aborted read – Invalid Txn State	$P \lor C$	21.11.15
3616-b	Lost txn writes	$\emptyset$	21.11.15

#	Issue	Fault	Resolved
1	Duplicate writes by default	P ∨ C ∨ Part	22.1.1
3039	Duplicate writes w/ ' ' '	P.. C ∨ Part	21.10.3
3335	Assert crash d		
3336	Assert crash		22.1.1
3003	Inconsistent ot.	, ant	21.10.3
6	Lost/stale messages	P ∨ C ∨ Part	
7	Aborted read — NotLeaderOrFollower	M ∧ P	
8	Write cycles	∅	
3036	Aborted read/Circular information flow	∅	21.10.2
10	Internal non-monotonic polls	∅	
3616-a	Aborted read — Invalid Txn State	P ∨ C	21.11.15
3616-b	Lost txn writes	∅	21.11.15

Need Docs

#	Issue	Fault	Resolved
1	Duplicate writes by default	P ∨ C ∨ Part	22.1.1
3039	Duplicate write	C ∨ Part	21.10.3
3335	Assert crash		
3336	Assert crash		22.1.1
3003	Inconsistent offsets	C ∨ Part	21.10.3
6	Lost/stale messages	P ∨ C ∨ Part	
7	Aborted read – NotLeaderOrFollower	M ∧ P	
8	Write cycles	∅	
3036	Aborted read/Circular information flow	∅	21.10.2
10	Internal non-monotonic polls	∅	
3616-a	Aborted read – Invalid Txn state	P ∨ C	21.11.15
3616-b	Lost txn writes	∅	21.11.15

Effective data loss

Redpanda found many of these bugs independently

#3039 #3003 #3036

Most frequent issues resolved in 21.10.3 — but some serious problems in 21.11.2

No clock skew issues

There may be more....

Ease of operation

Not fault tolerant by default

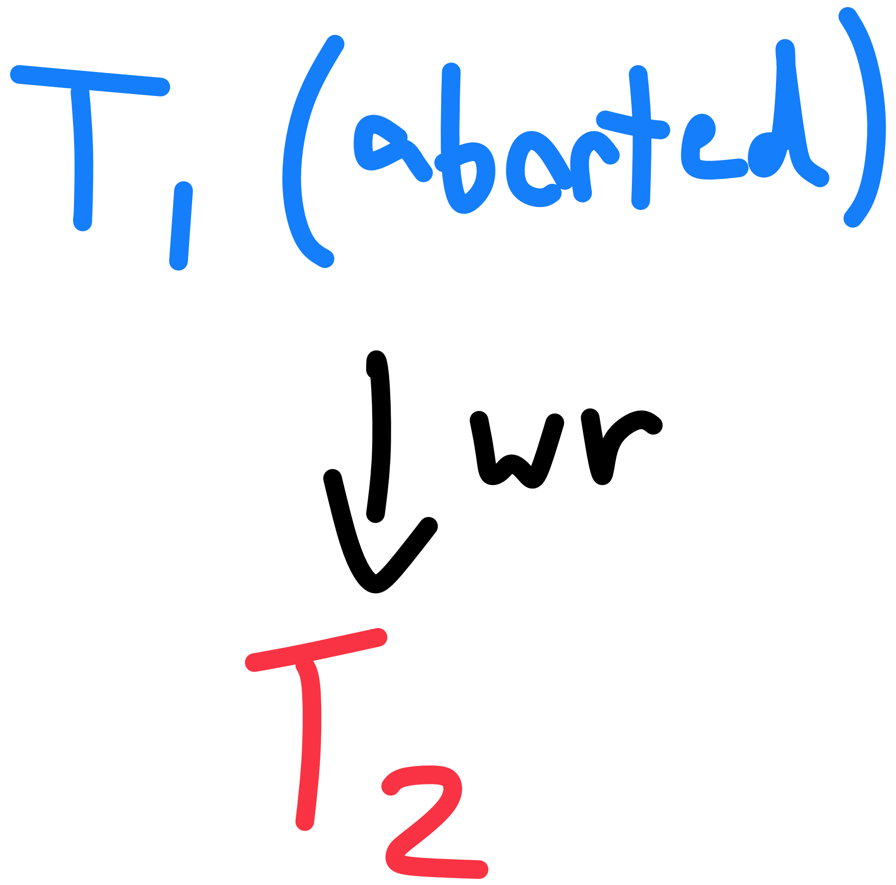
Idempotence

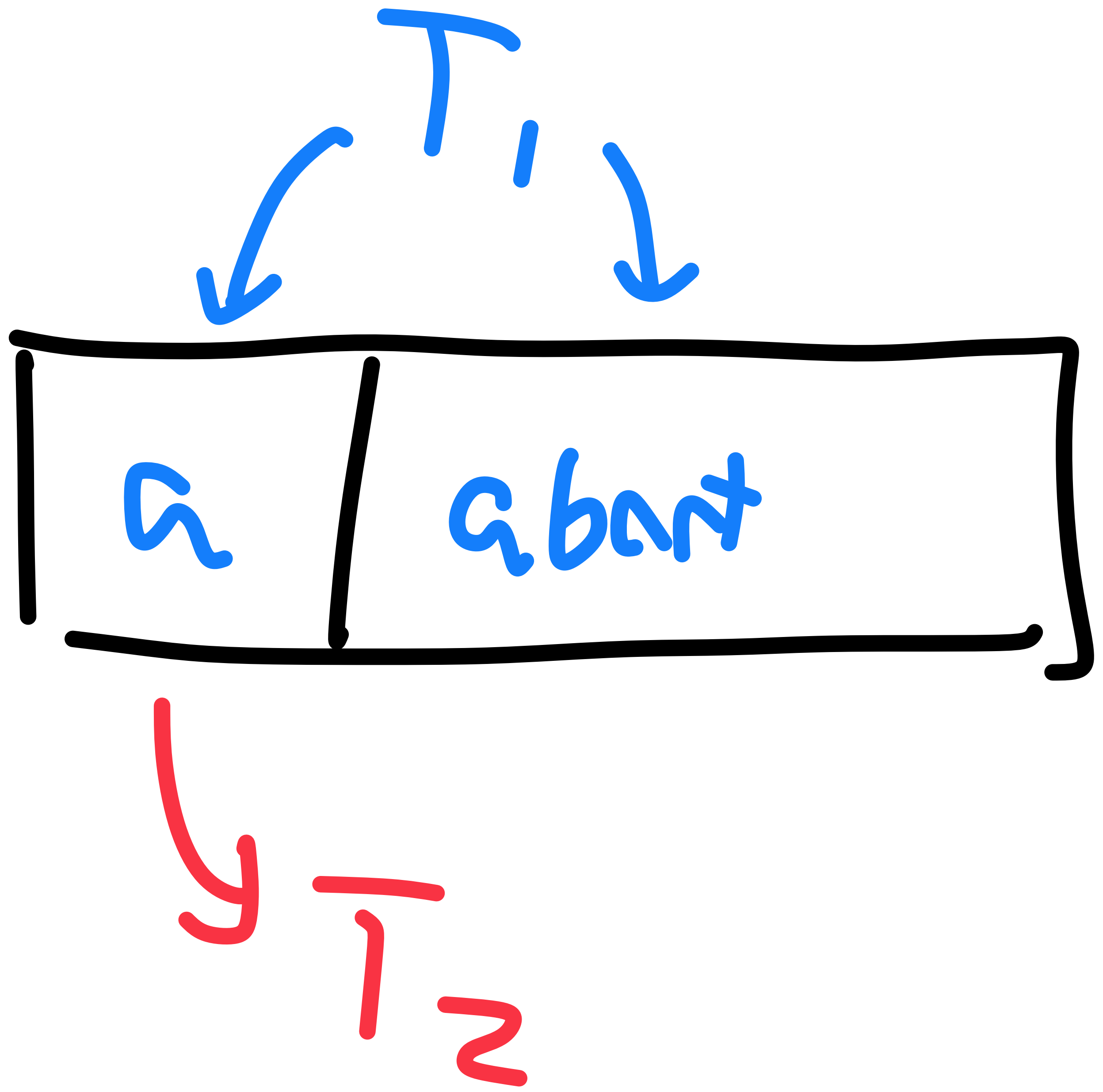
Txns

Default topics

Transactions

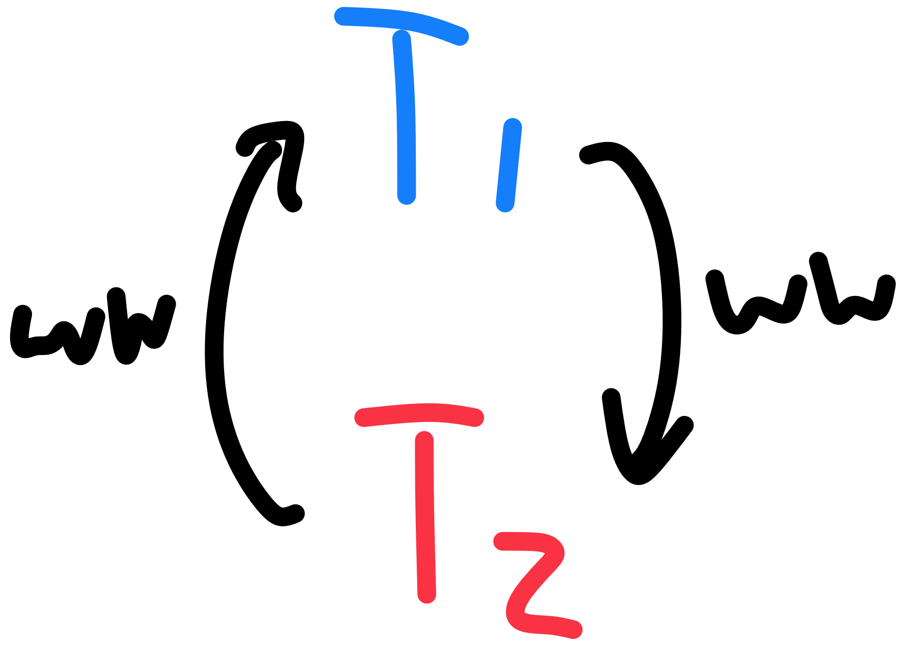
G1a

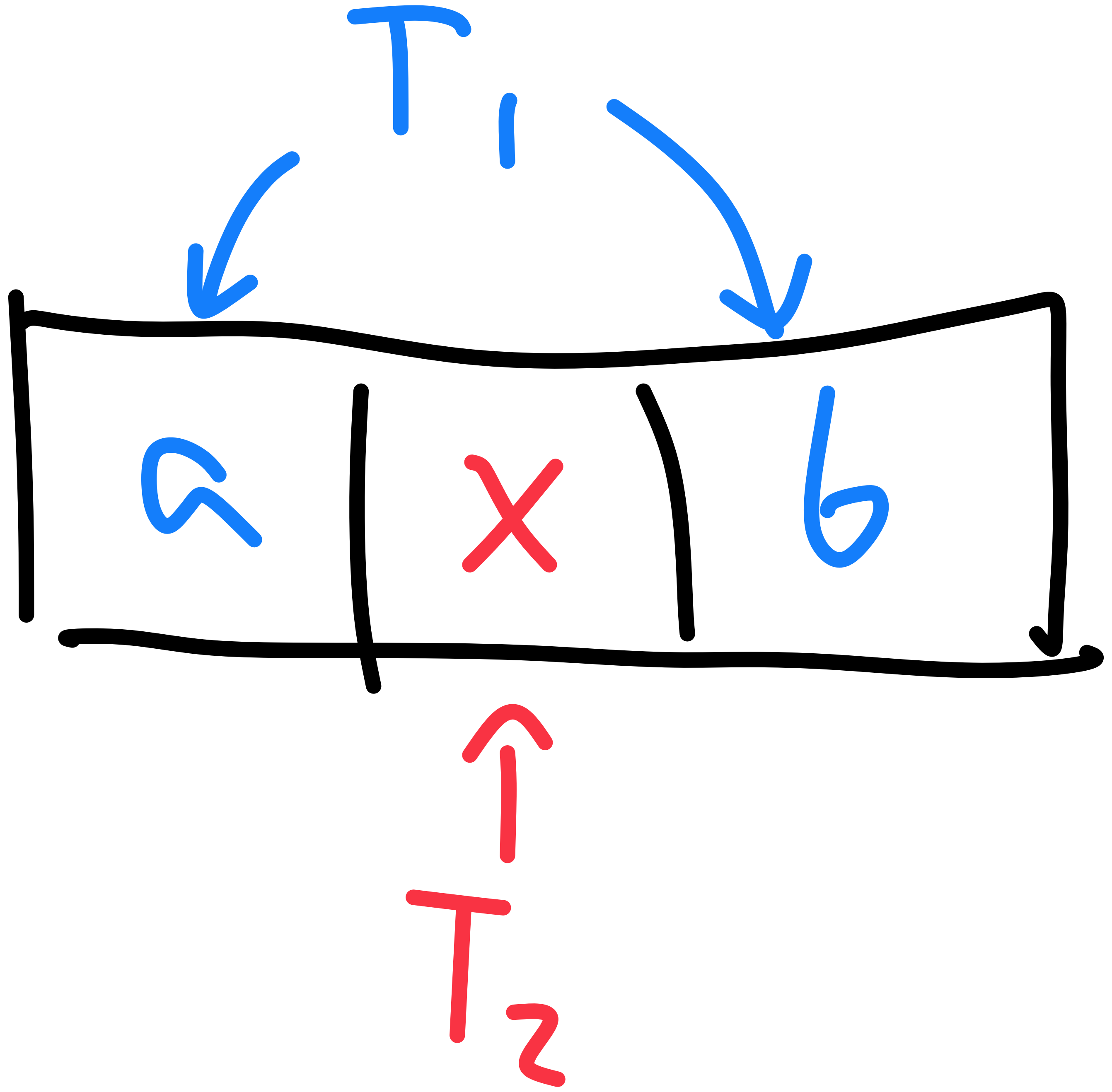




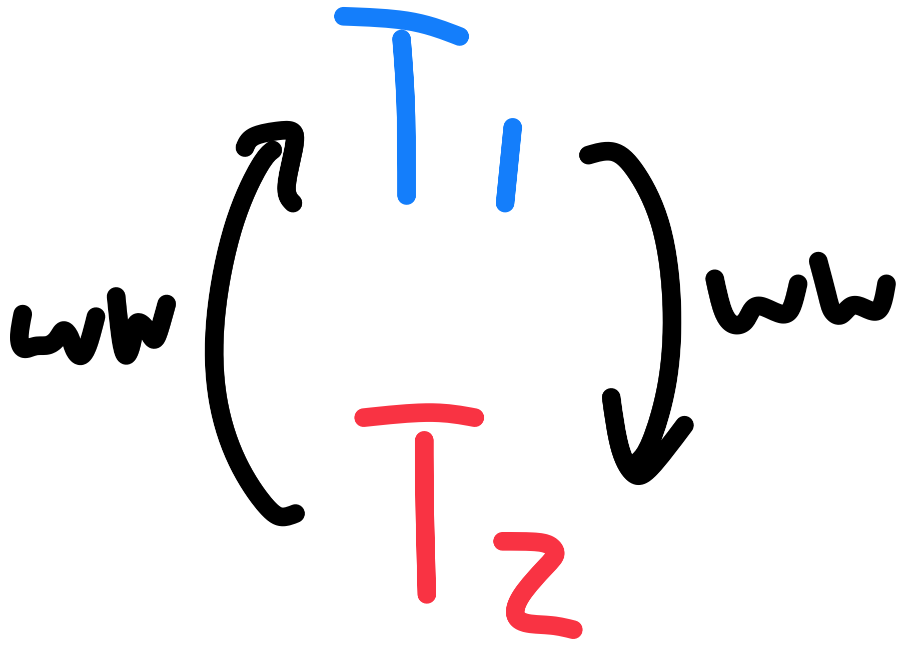
As of 21.11.15, forbidden by read_committed

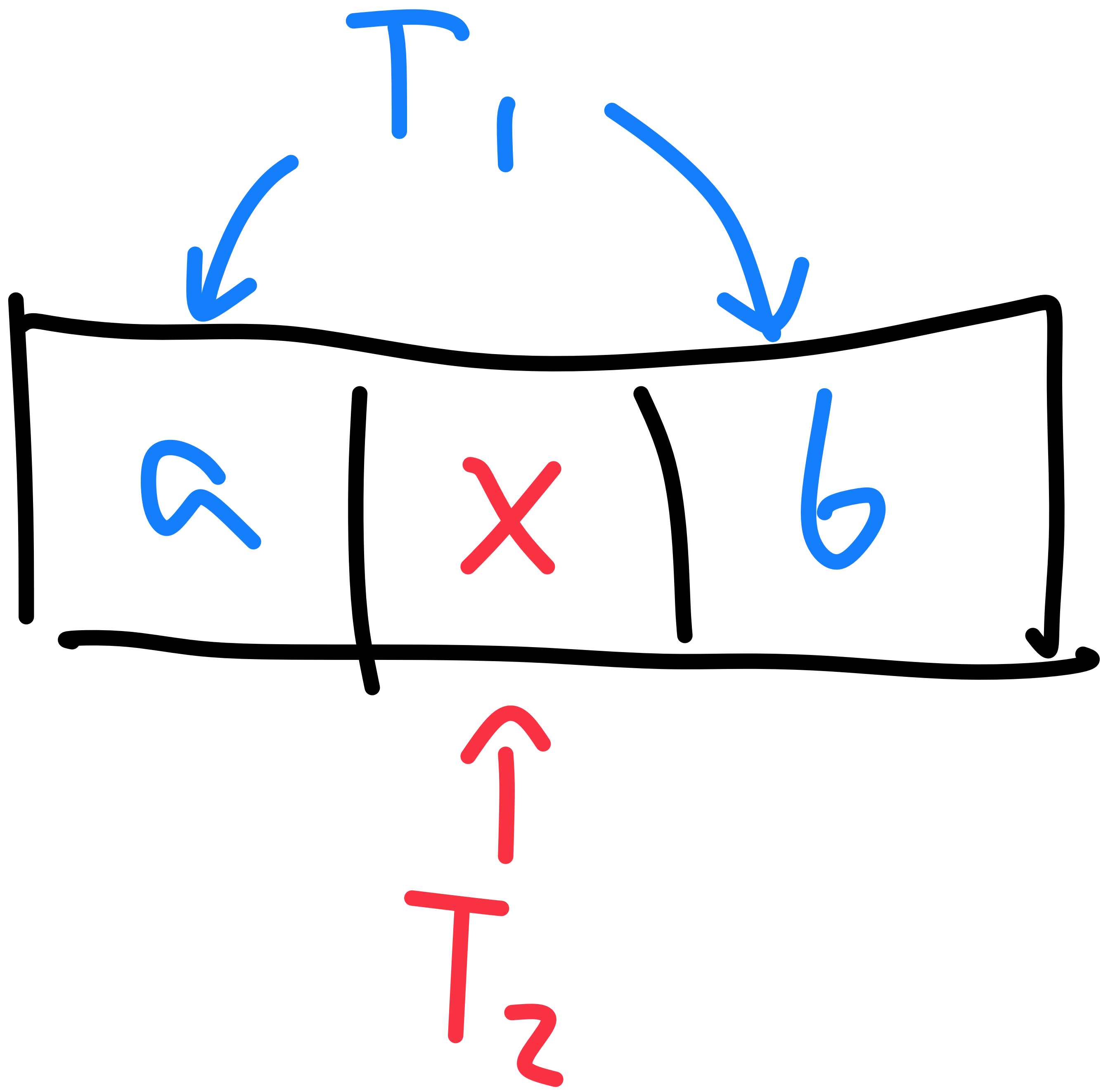
GO



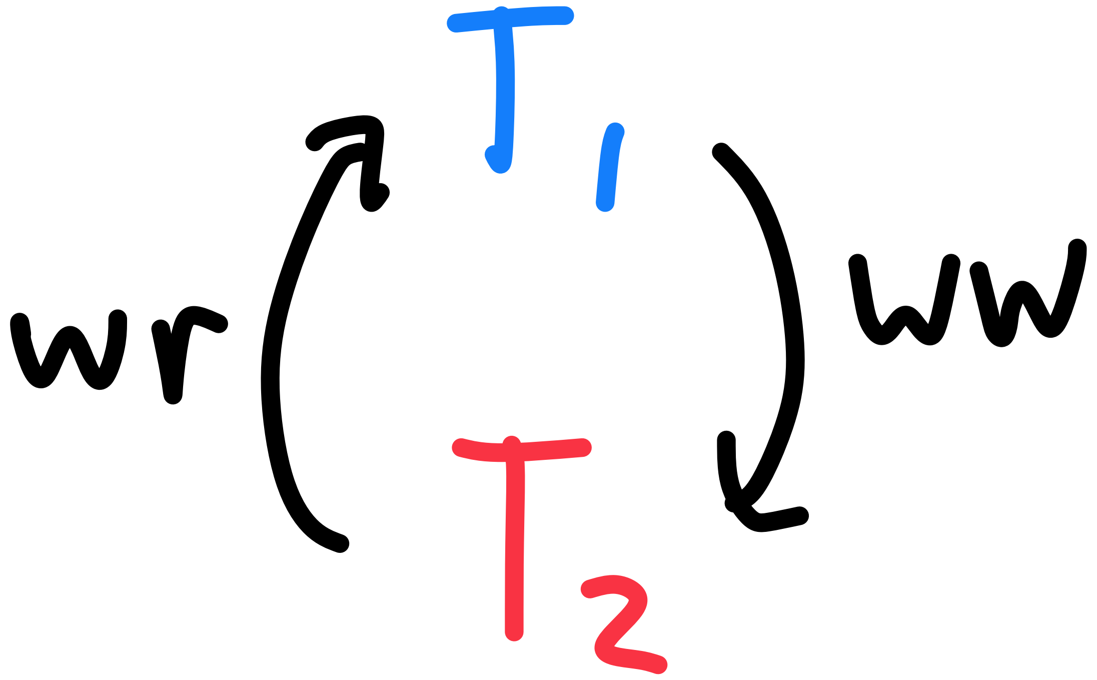


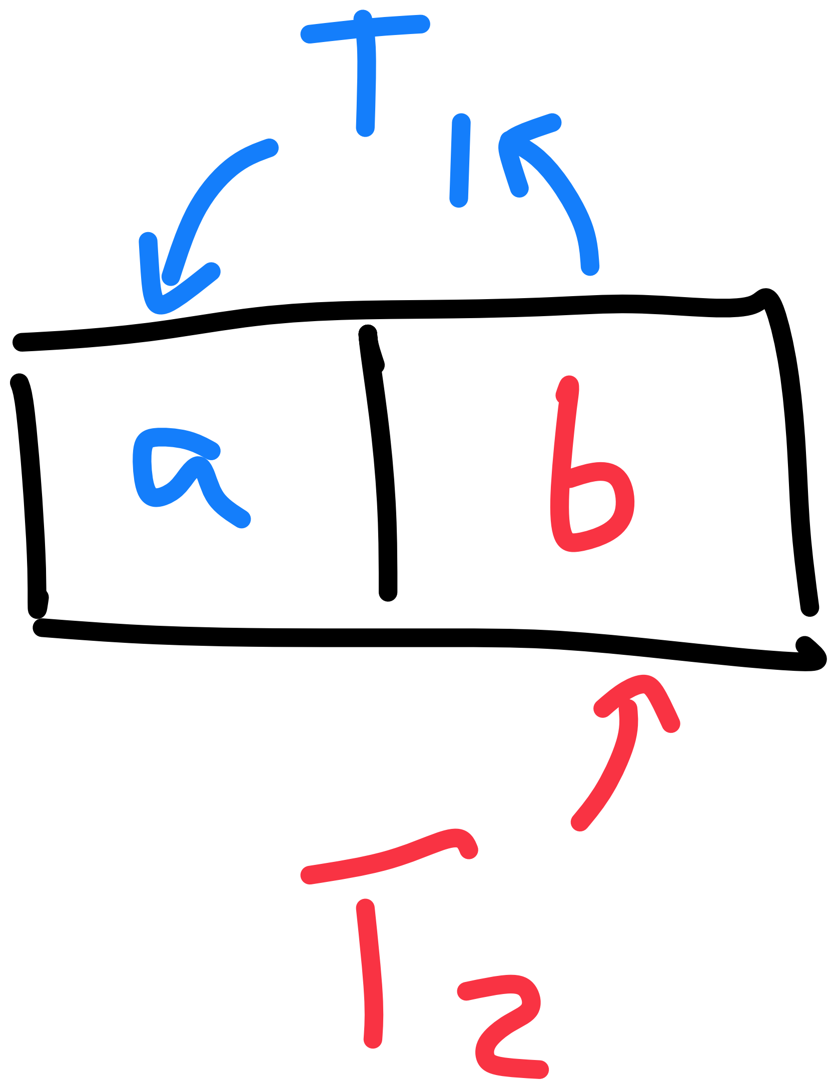
G0



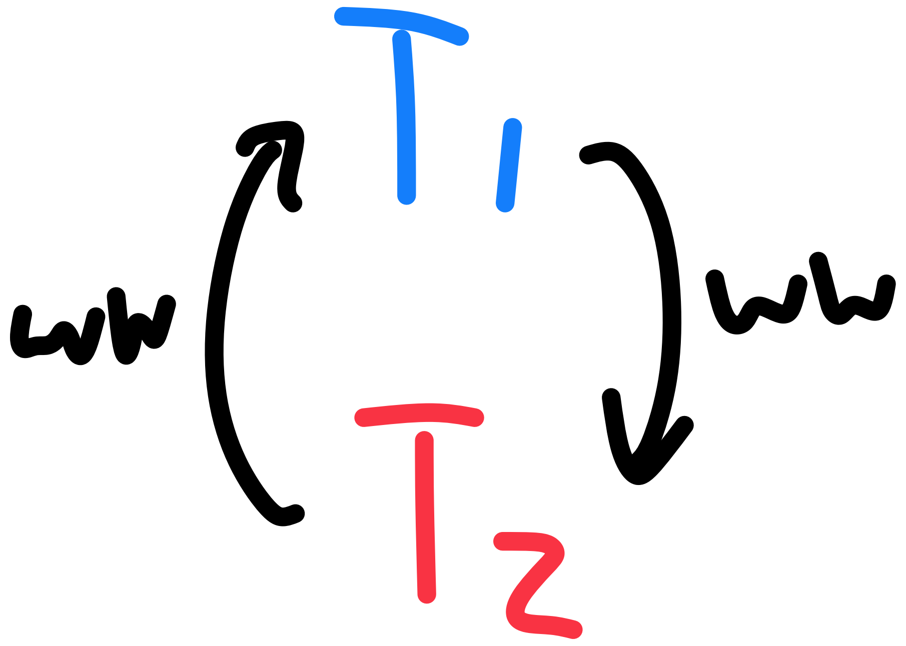


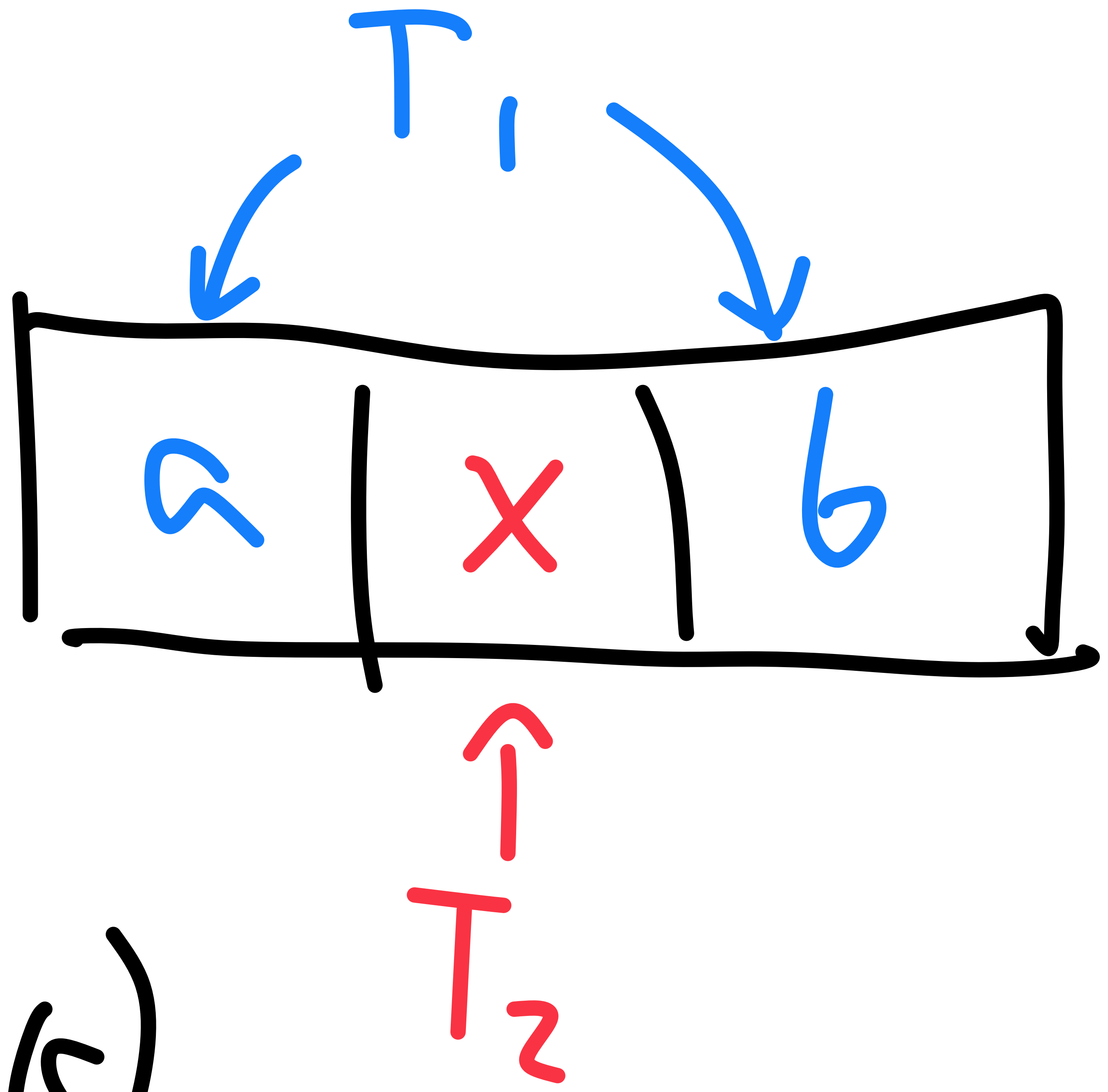
G1c





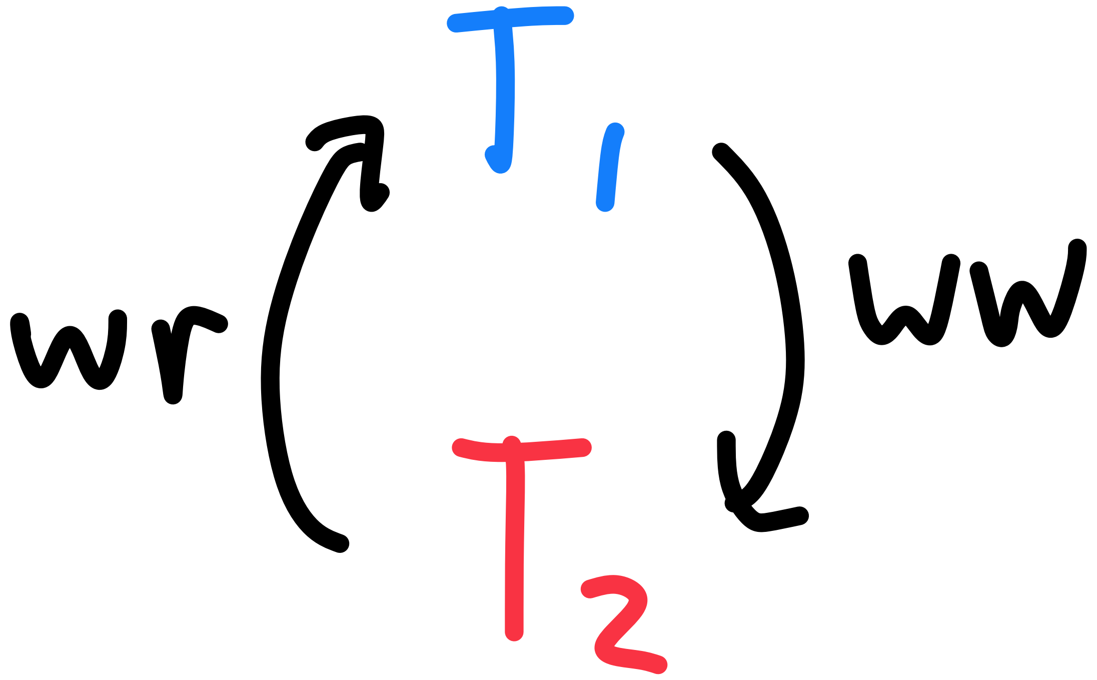
G0

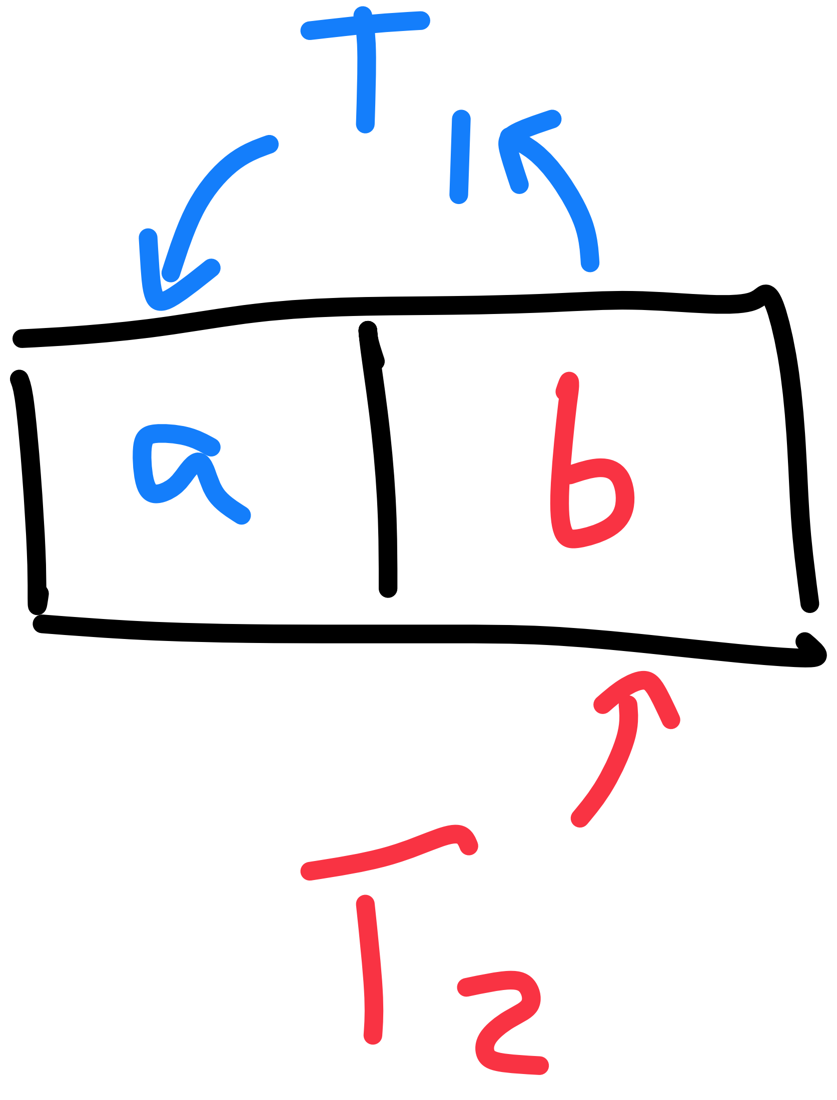




(Ambiguous if this is OK)

G1c

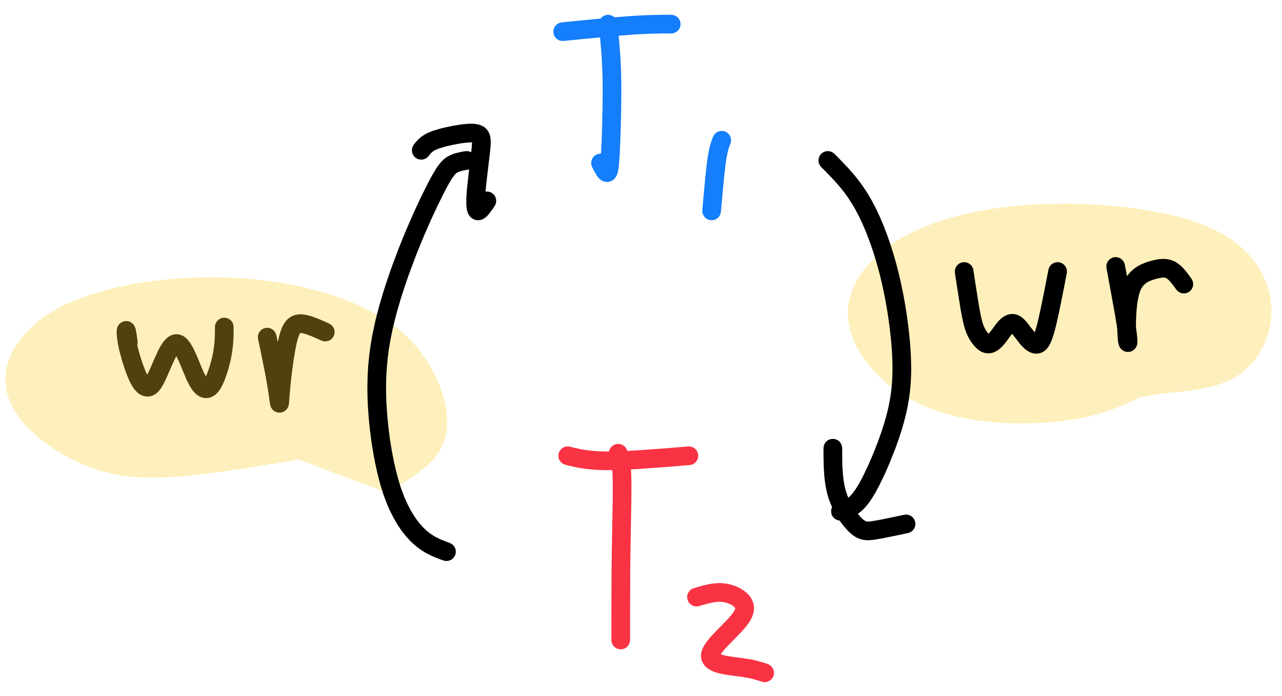


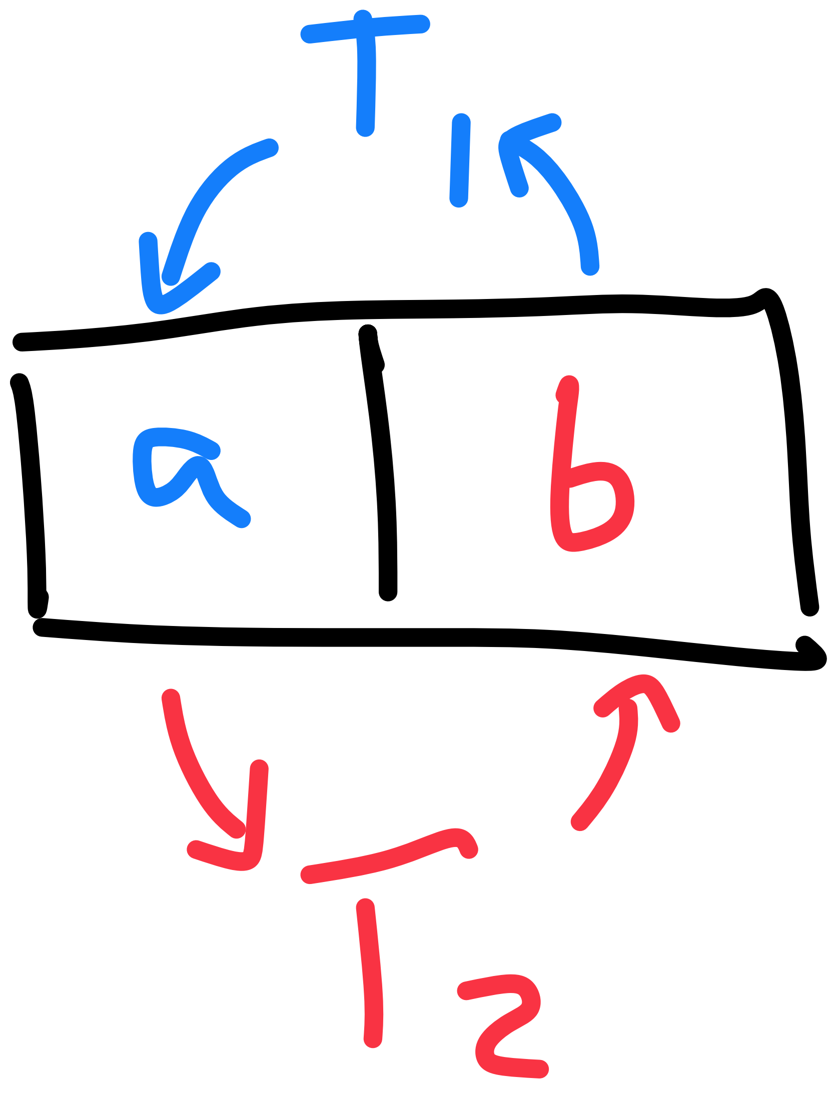


Should be fixed under read_committed in 21.11.15!

G1c

w/all wr edges





Didn't check exactly-once semantics

1. Multiple polls per txn
2. Allowed more than one transactional.id to consume a partition

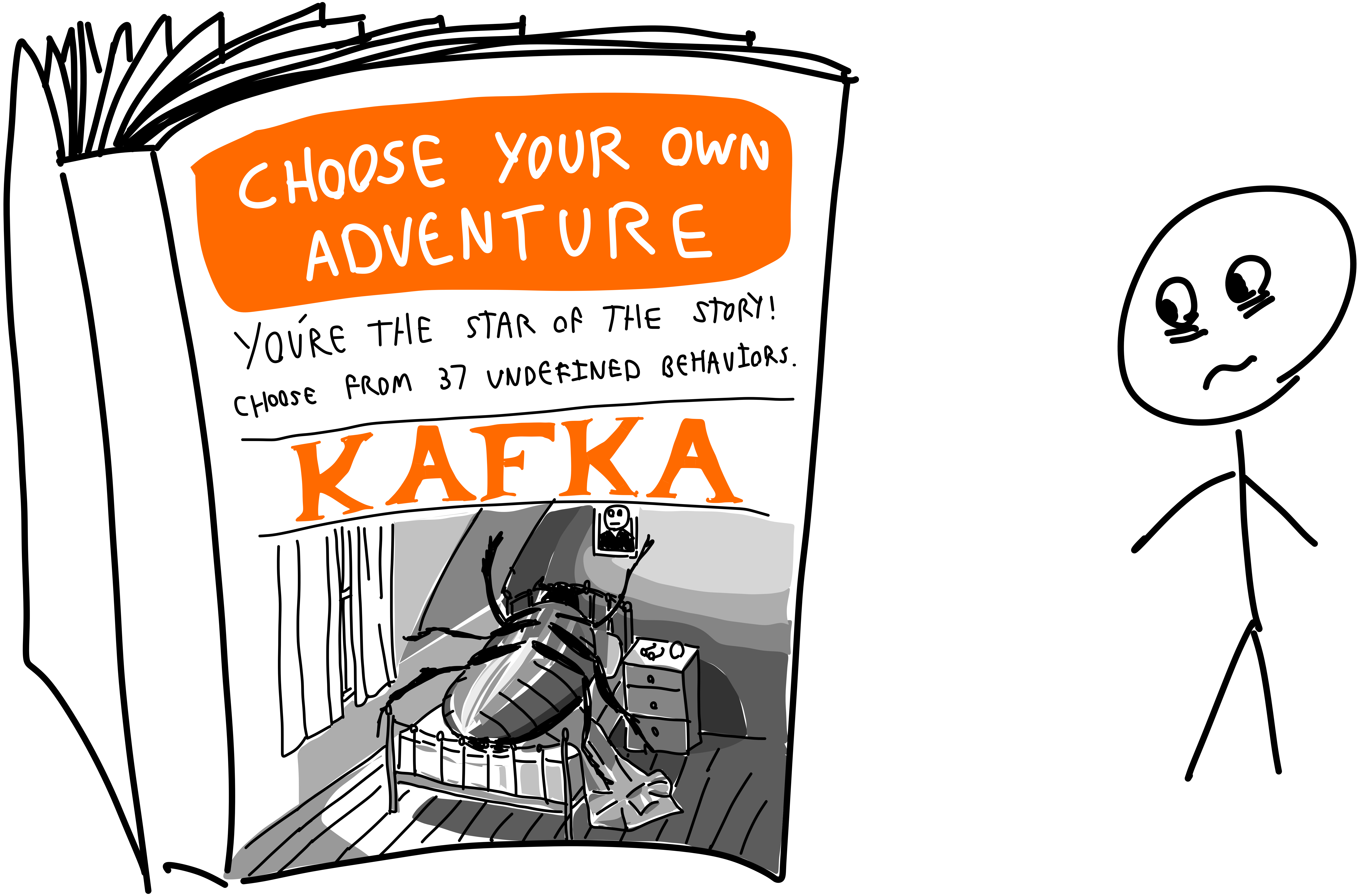
All bets are off!?

1. Multiple polls per txn
2. Allowed more than one transactional.id to consume a partition

Txn Documentation

- Absent
- Incomplete
- Vague
- Misleading
- Wrong

- Largely explained in terms of implementation
- Go read 67 pages of design doc?


CHOOSE YOUR OWN ADVENTURE
YOU'RE THE STAR OF THE STORY!
CHOOSE FROM 37 UNDEFINED BEHAVIORS.
KAFKA

Membership Changes

- No docs, but customers were learning how in Slack
- Improved APIs & some docs now
- Don't re-use node IDs!

FUTURE WORK!

- Binding clients to nodes
- Adding partitions
- Exactly-once semantics
- Streams

Thanks!

Funded by

Redpanda Data

- Camilo Aguilar
- Travis Bischel
- Bob Dever
- Juan Castilla
- Alexander Gallego
- Dhruv Gupta
- Michal Maslanka
- Denis Rystsov
- John Spray
- Coral Waters
- David Wang
- Noah Watkins
- Allison Daly

https://jepsen.io

https://redpanda.com