Process 1

read page1

read page2

add

write page1

Process 2

read page1

read page2

add

write page2

P1 → P2

P2 → P1

Serial execution

~~First P1 → First P2 → rest of P1 → rest of P2~~

↖ Not a serial execution

Object A, B, C

↑ Buffer page
file
record

P1 a+=b

read A
read B
write A

P2 b+=a

read B
read A
write B

P1 R(A) R(B) W(A)
P2 R(B) R(A) W(B)
✓ Serial

P1 R(A) R(B) W(A)
P2 R(B) R(A) W(B)
✗

P1 R(A) R(B) W(A)
P2 R(B) R(A) W(B)
✓ Serial

chal P1 R(A) W(B)

4 P2 R(B) R(A) W(B)

✓ Serializable

chal P1 R(A) R(B) W(A)

5 P2 R(B) R(A) W(B)

"conflict equivalent"

P1 W(B)

P2 W(B)

P3

Not conflict equiv

W(B) W(B)

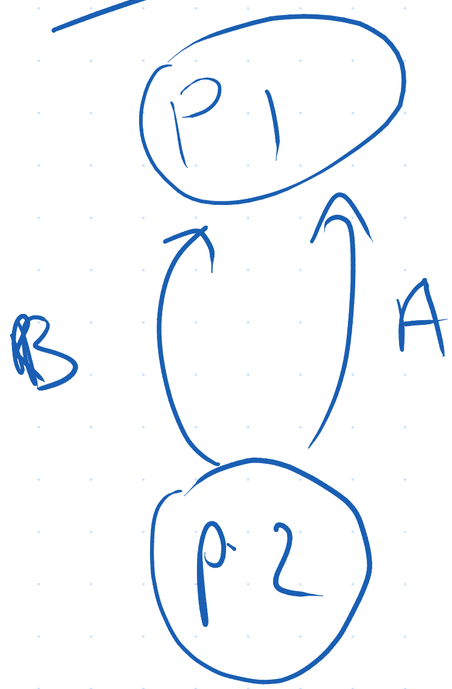
R(B) R(B)

conflict serializability

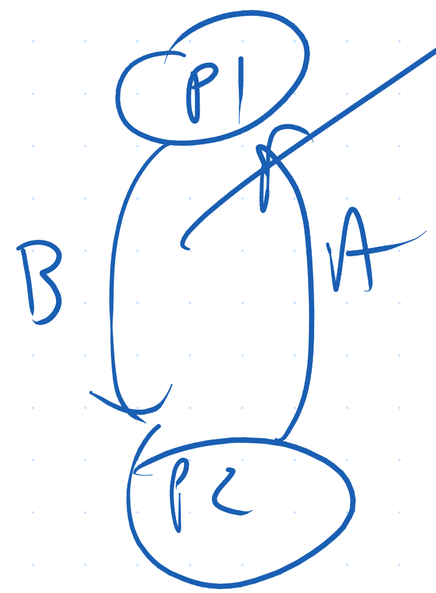
- serial sched: One process at a time
- Conflict: 2 processes Read/write an object + at least one writes
- conflictequiv: Both sched exec same ops in same order

  Conflicts happen in the same direction
- conflict serializable: A schedule is conflict equivalent to some serial schedule

Sched 1



Sched 2



Cycle

↑

[for each conflict, put an edge from earlier op's process to later op's process

Process 1

[R(A)
[R(B)
[W(A)

Process 2

[R(B)
[R(A)
[W(B)

P1

R(A)

R(B)

W(A)

P2

R(B)

R(A)

W(B)

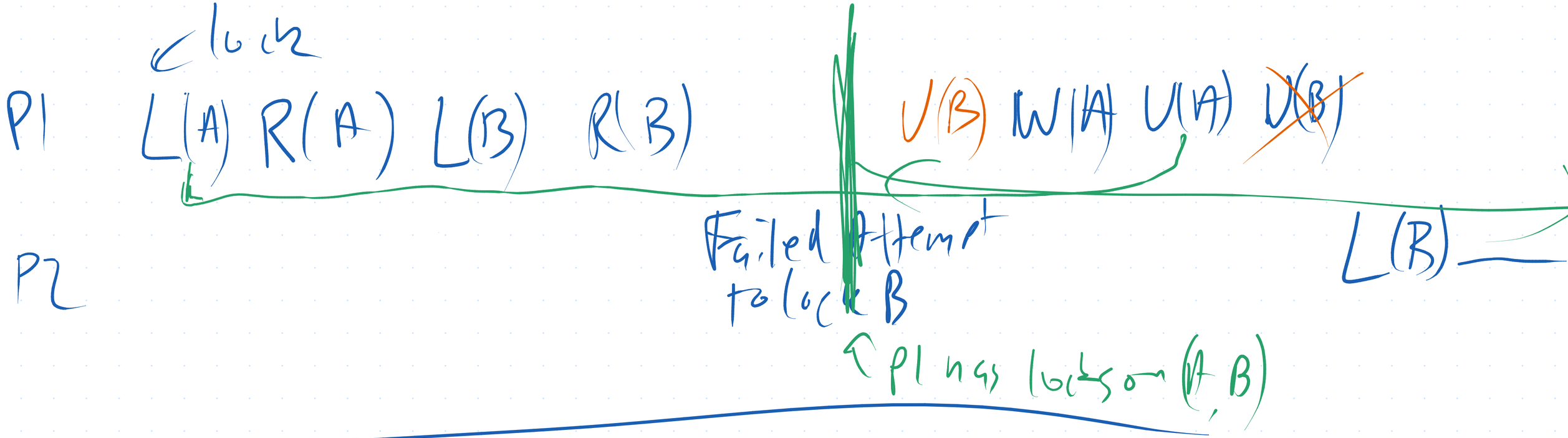
P1 W(A) → P2 crash

P1. W(A)

P1 R(B) → P2 R(B) → P2. R(A) → P1. W(A) → P2. W(B)

P1 R(A) → P2 R(B)

P1 R(B)

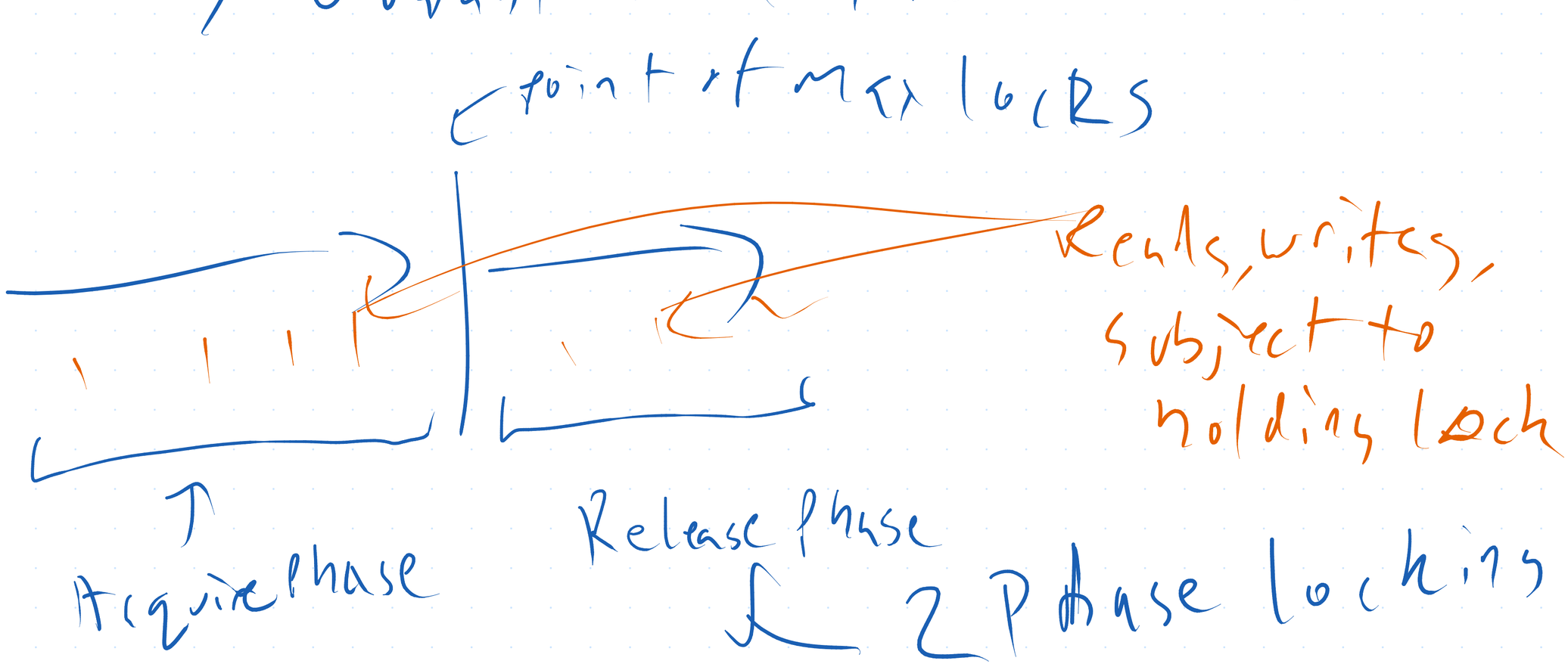


Each process
- Acquire locks
- compute
- Release locks

→ Guaranteed to be conflict serializable

- Every object is locked before access
- There is a point where I hold all the locks
- Never re-acquire a lock

↳ Guaranteed conflict Serializable

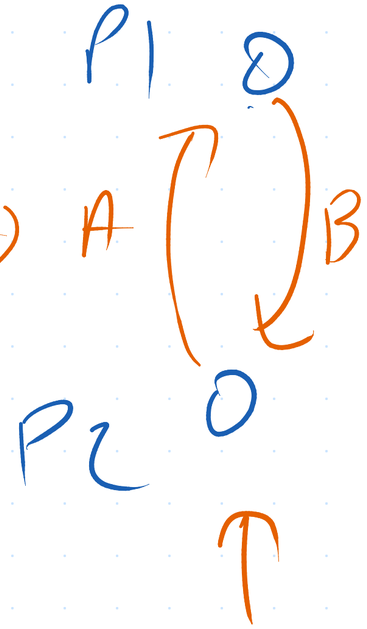


P1 L(A) R(A) L(B)

P2 L(B) R(B) L(A)

Cycle → No processin Cyclerun continue

↑ Deadlock ↑



// Waits for graph

P1 waits for P2 to release

P2 waits for P1 to release

Prevention

- Define an order over objects
- Only aquire in that order
  - ↳ No deadlock

  e.g. B+ Tree
  - ↳ Left-to right
    top-down

Recovery

- Periodically check for cycles
- kill & reset state of a process in the cycle

B+Tree

A

B

D

C

E

P1.R(C)

P1 locks: A, B, C

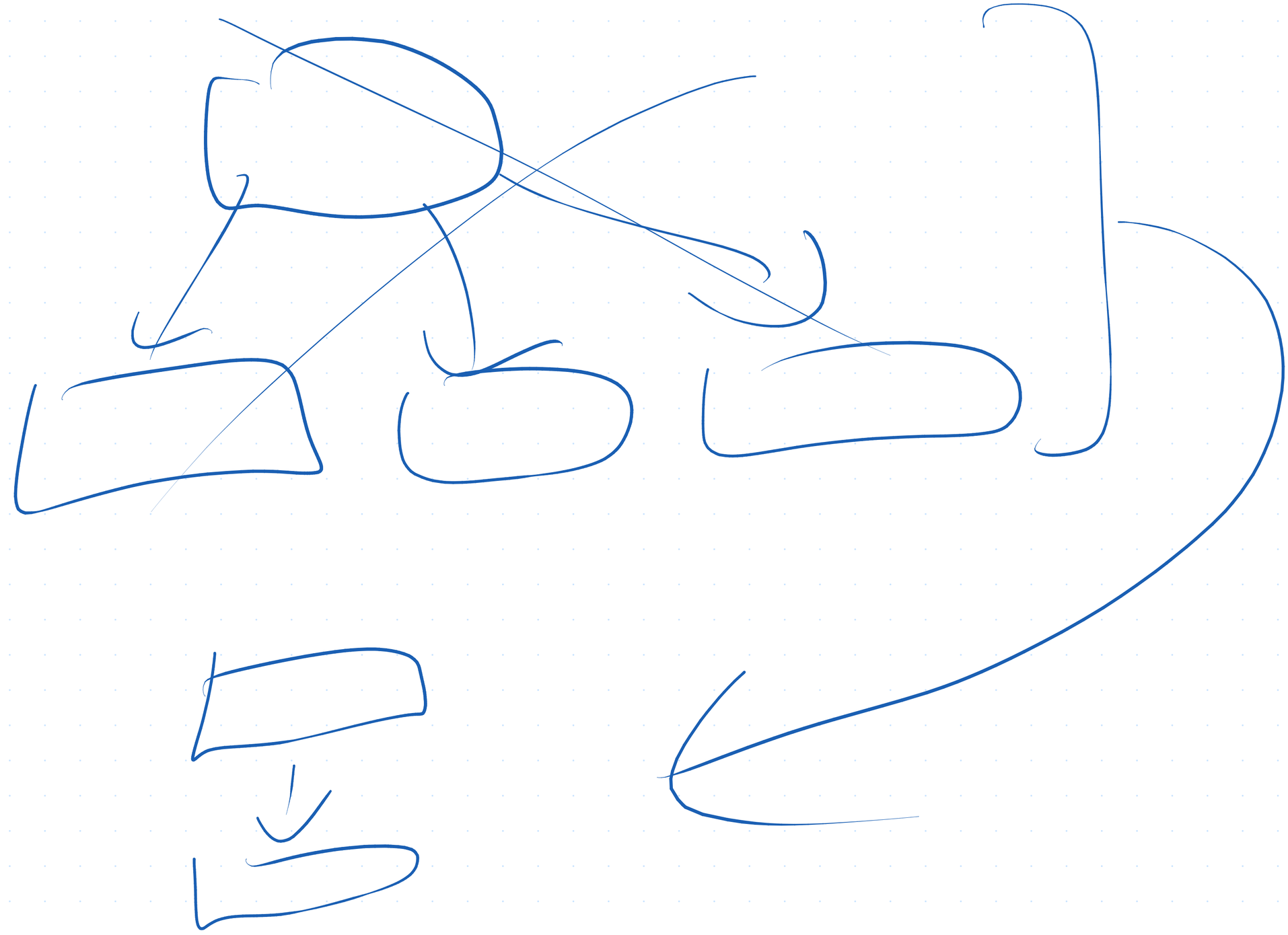
Conflicts

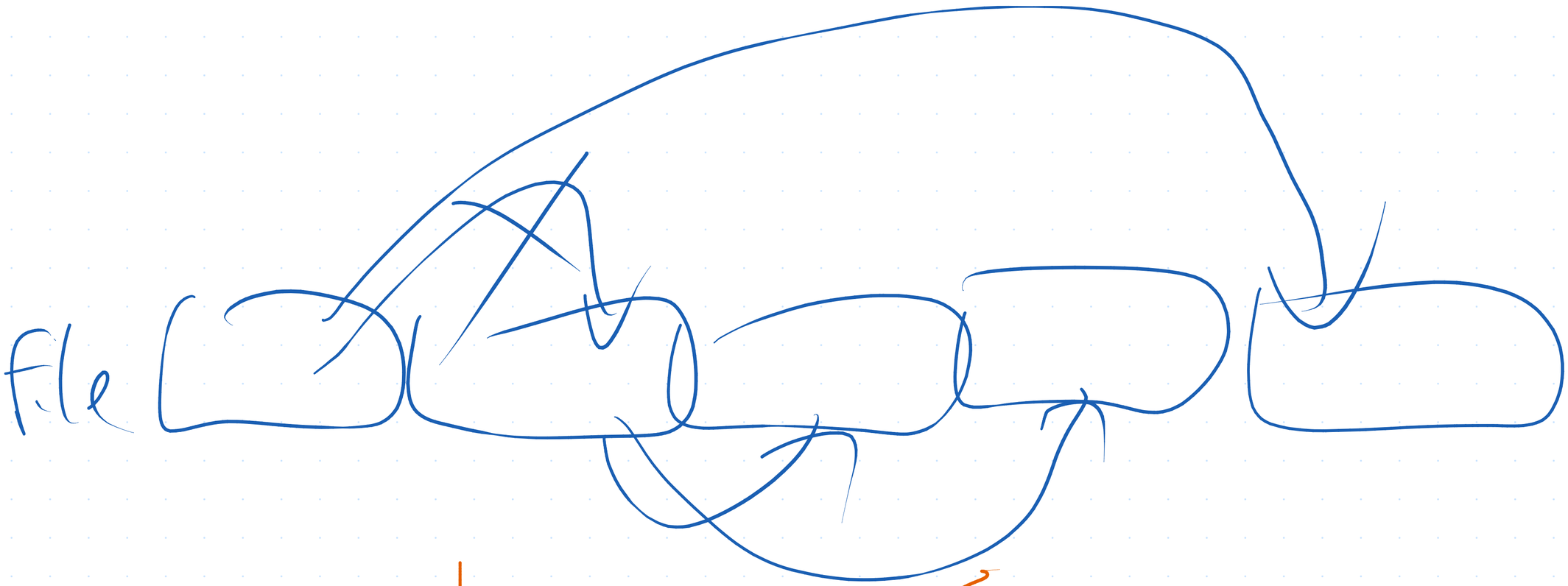
W

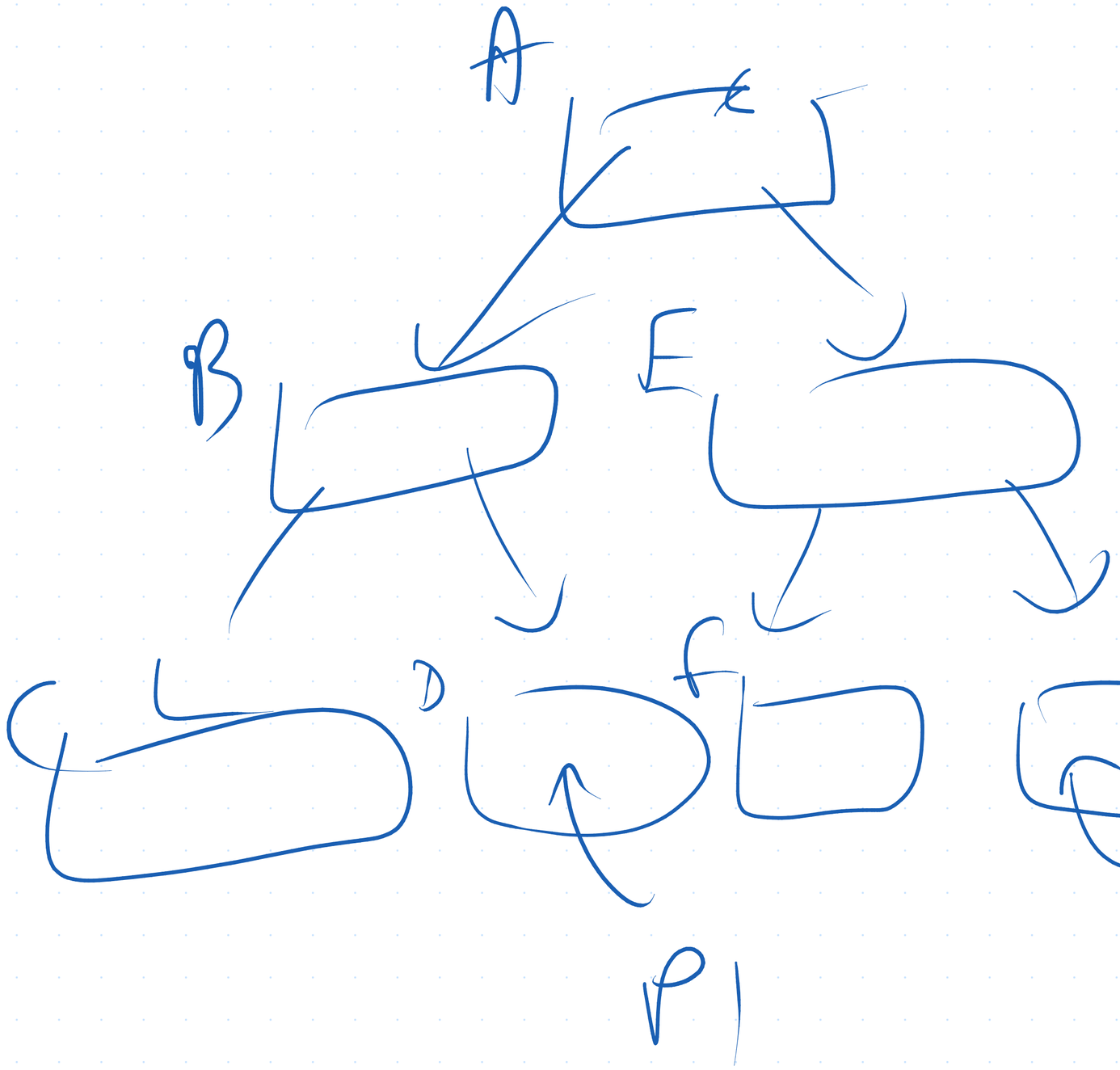
P2.R(E)

P2: locks(A, D, E

T2 locks on A, B




file
Need to free



P1 -
↳ L A (Read
↳ L B (Read
↳ L D

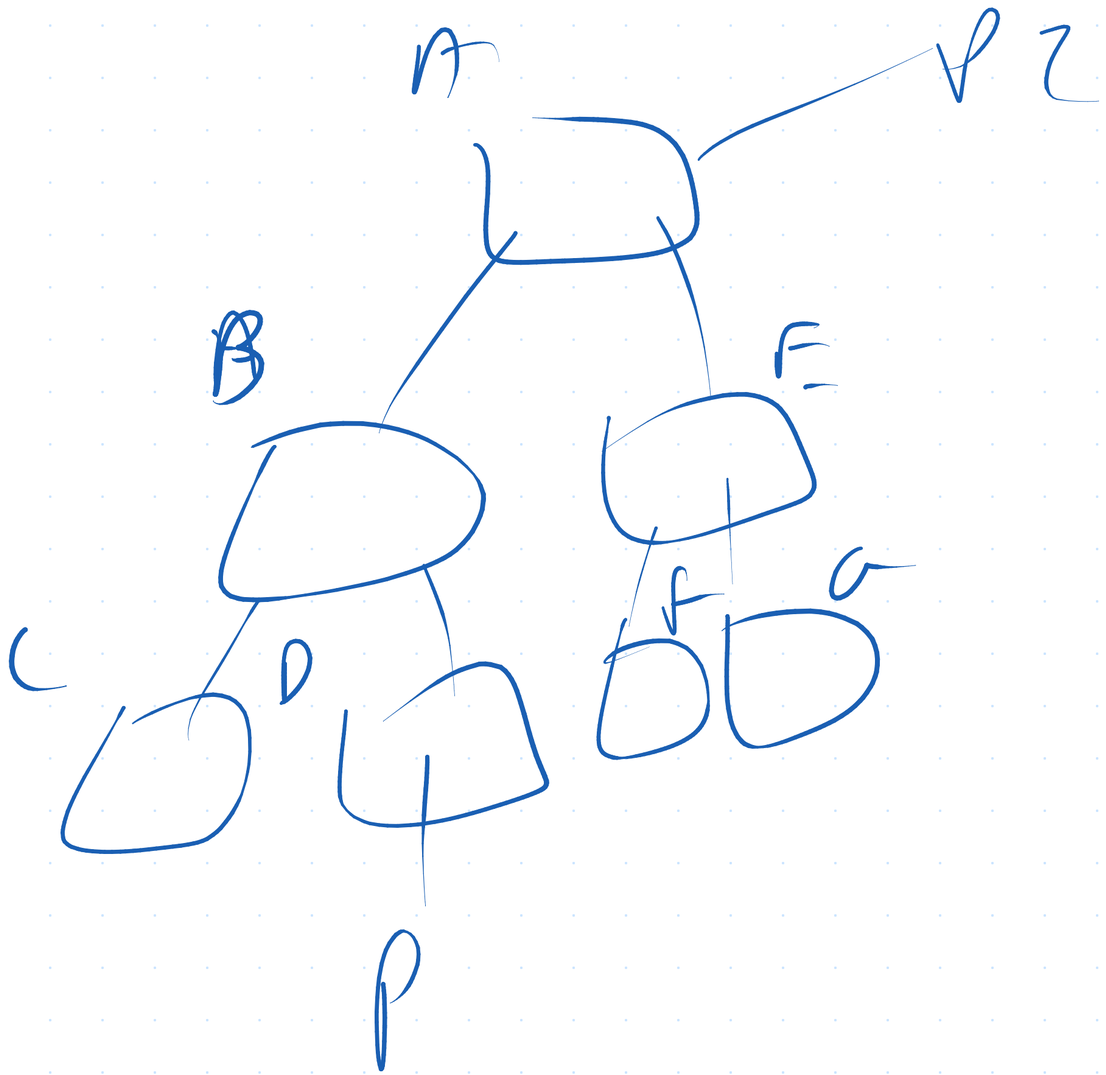
P2
↳ LA (Read
↳ LE (Read
↳ L C

P2

P1	Read	write
Read	✓	X
write	X	X

RWLock

- acquireReader
- upgrade
- acquireWriter
- release



P1

L(A) Reader

L(B) Reader

L(P) writer

P2

L(A) writer

P3

R(E) → L(A) read
L(E) read

P1 R(A) R(B) R(D)

L(A) read

W(A)

L(A)

P2

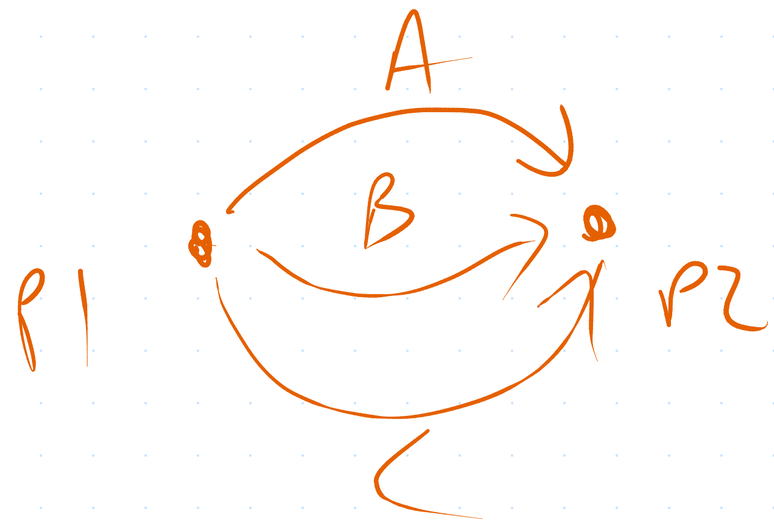
P3 R(A) R(C)

P1 R(A) W(B) W(C)

orig

P2 W(A) R(B) W(C)

P1



A orig P2

B orig P1

C orig P1 P2

P_1 $R(A)$ $W(B)$ $W(C)$

orig

P_2

$R(A)$ $R(B)$ $W(C)$

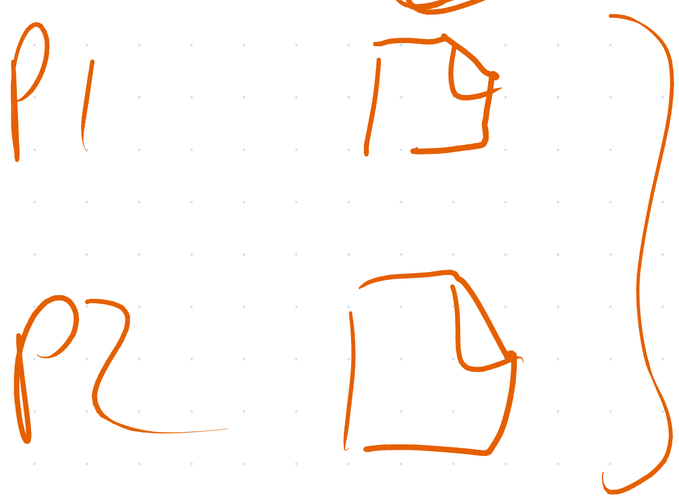
T_1

A orig P_2

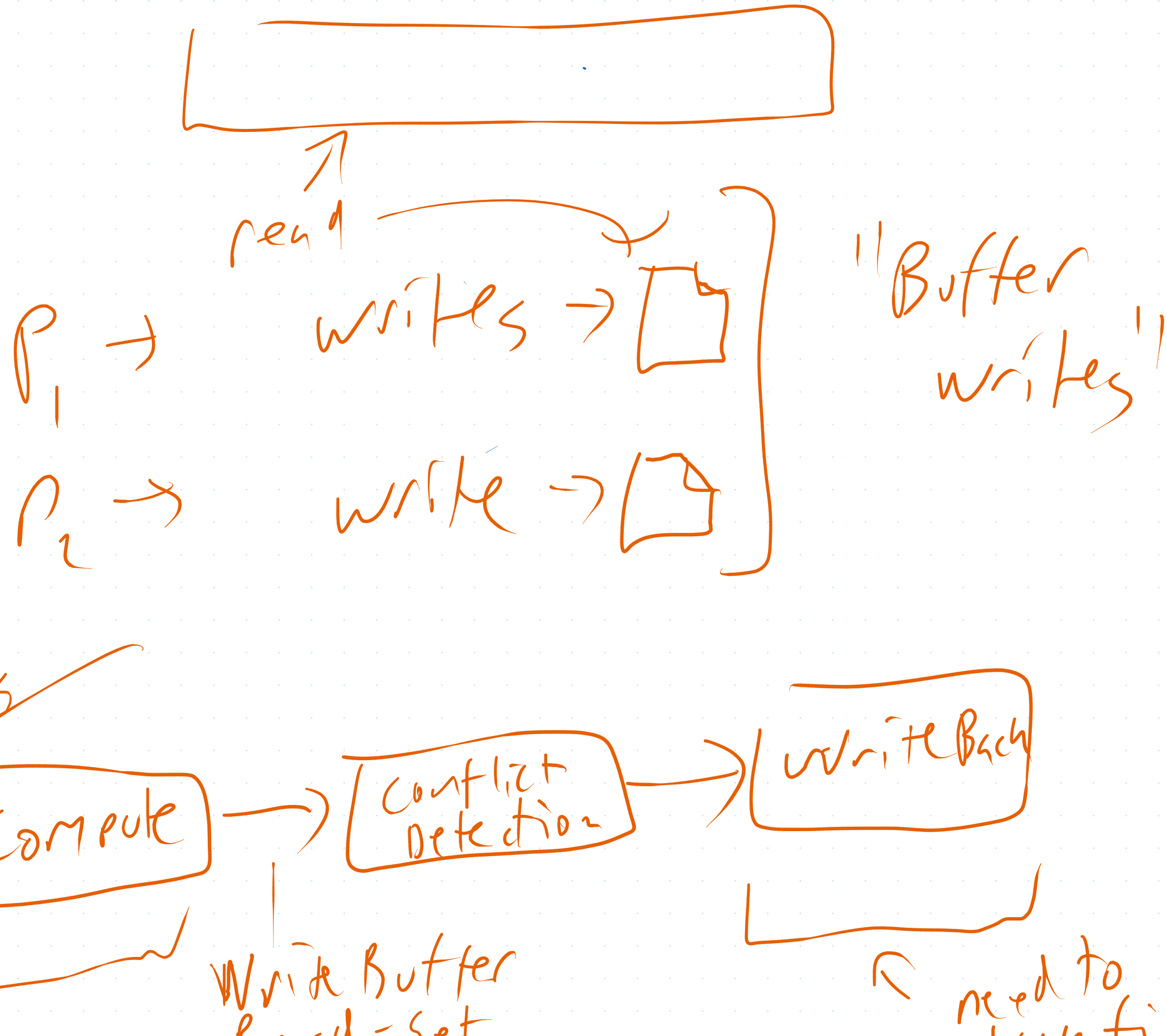
B orig P_1

C orig P_2

optimistic concurrency control



git: each process gets a full copy

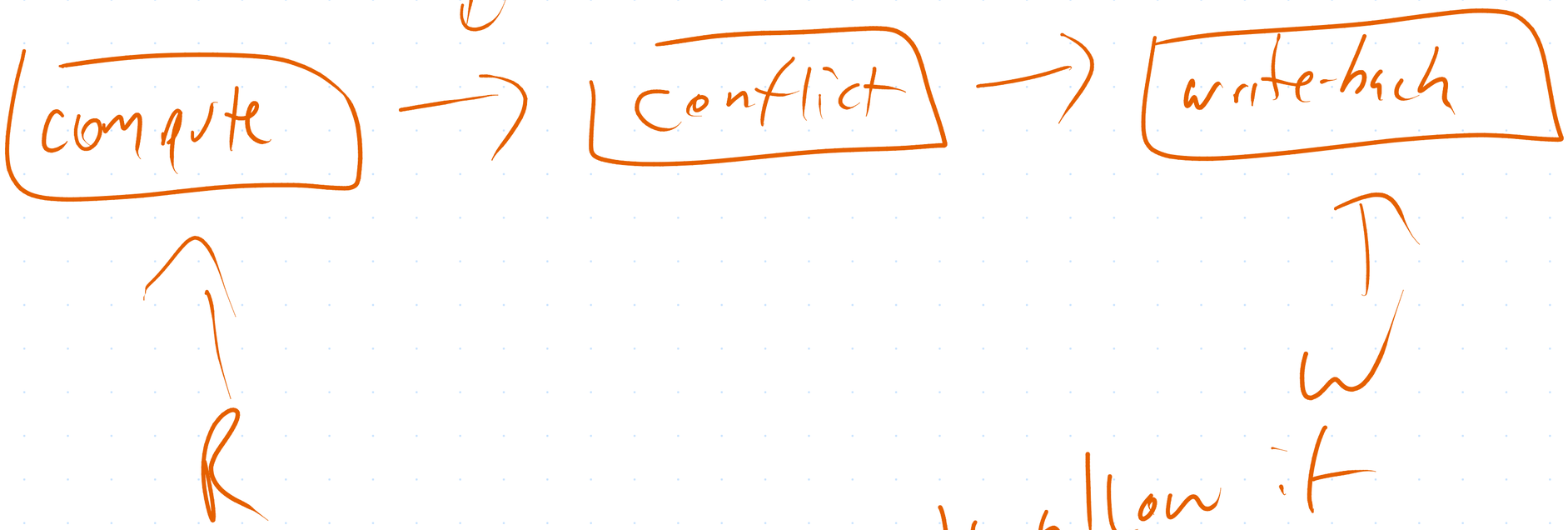

read
P_1 → writes →
"Buffer writes"
P_2 → write →
Process
Compute
Conflict Detection
WriteBack
Write Buffer
Read-Set
need to lock file

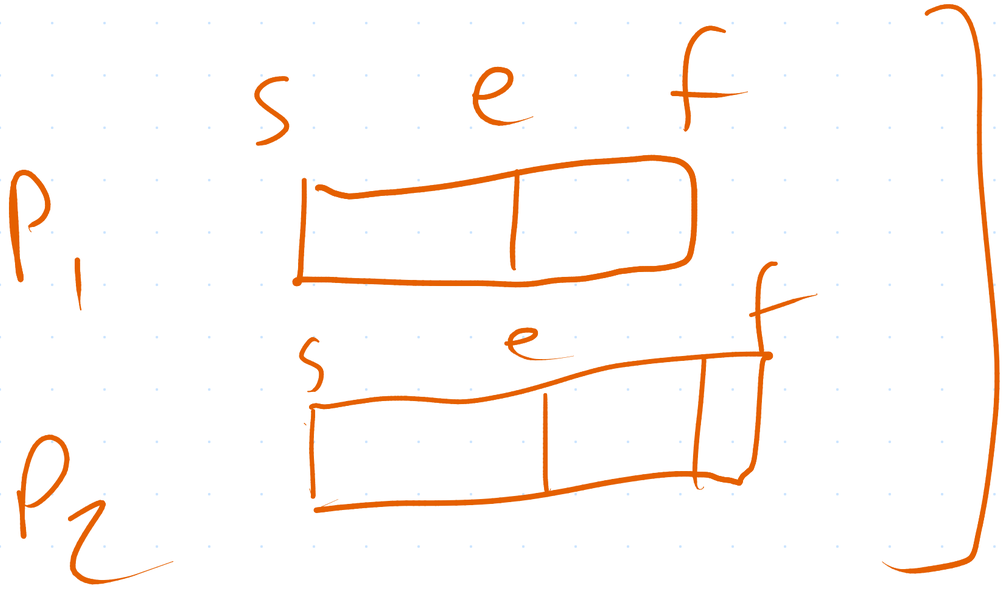
usually most of the cost

P: Read Set } — all objects that we may
Write Set } — " " we will

t start

t end

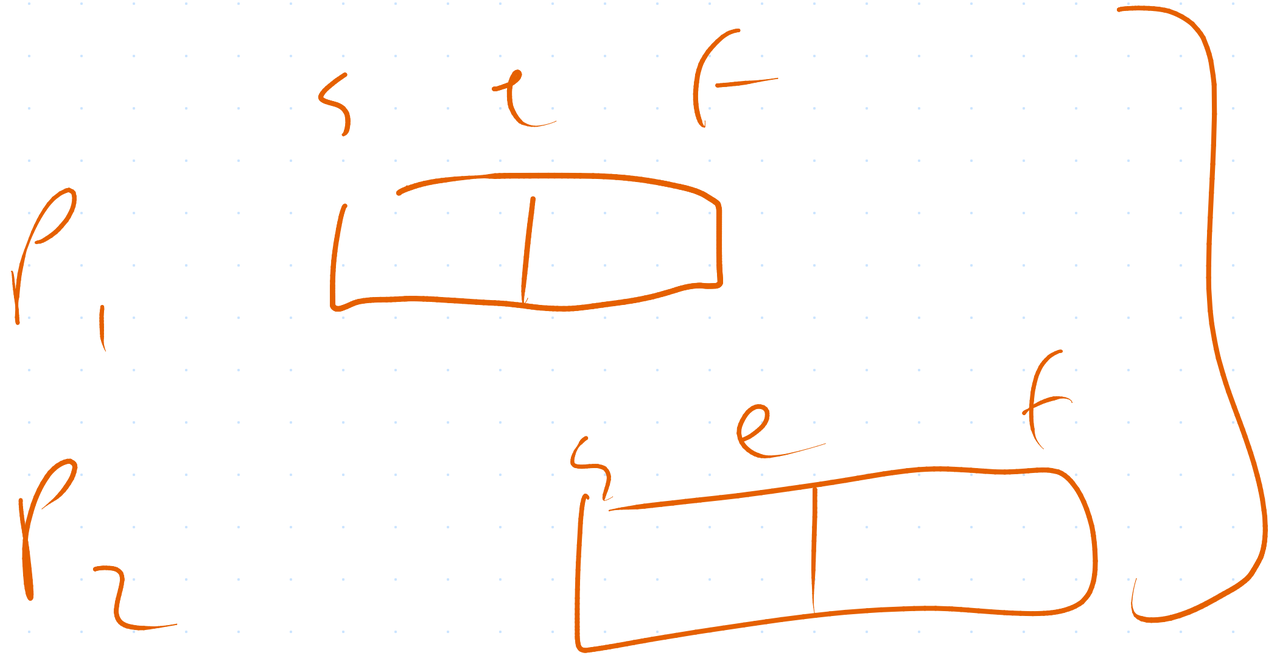


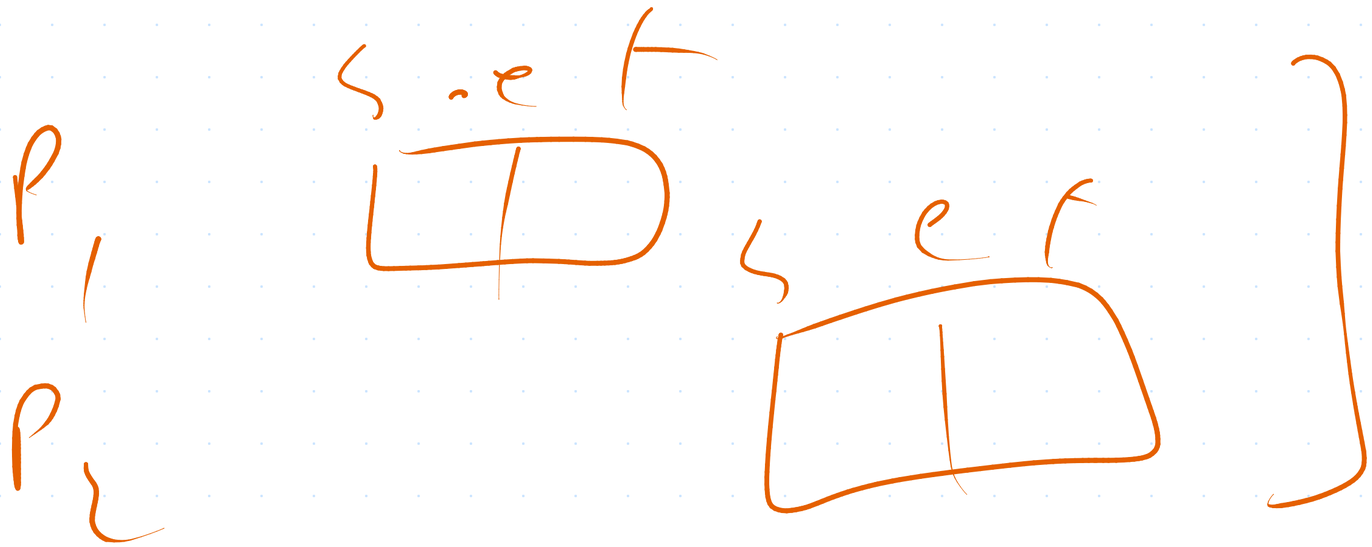


only allow :t

P_1 R ✗ P_2 W

P_2 R ✗ P_1 W


P1
P2
only allow: f
P2 ⊄ P1


P1
P2
always ok