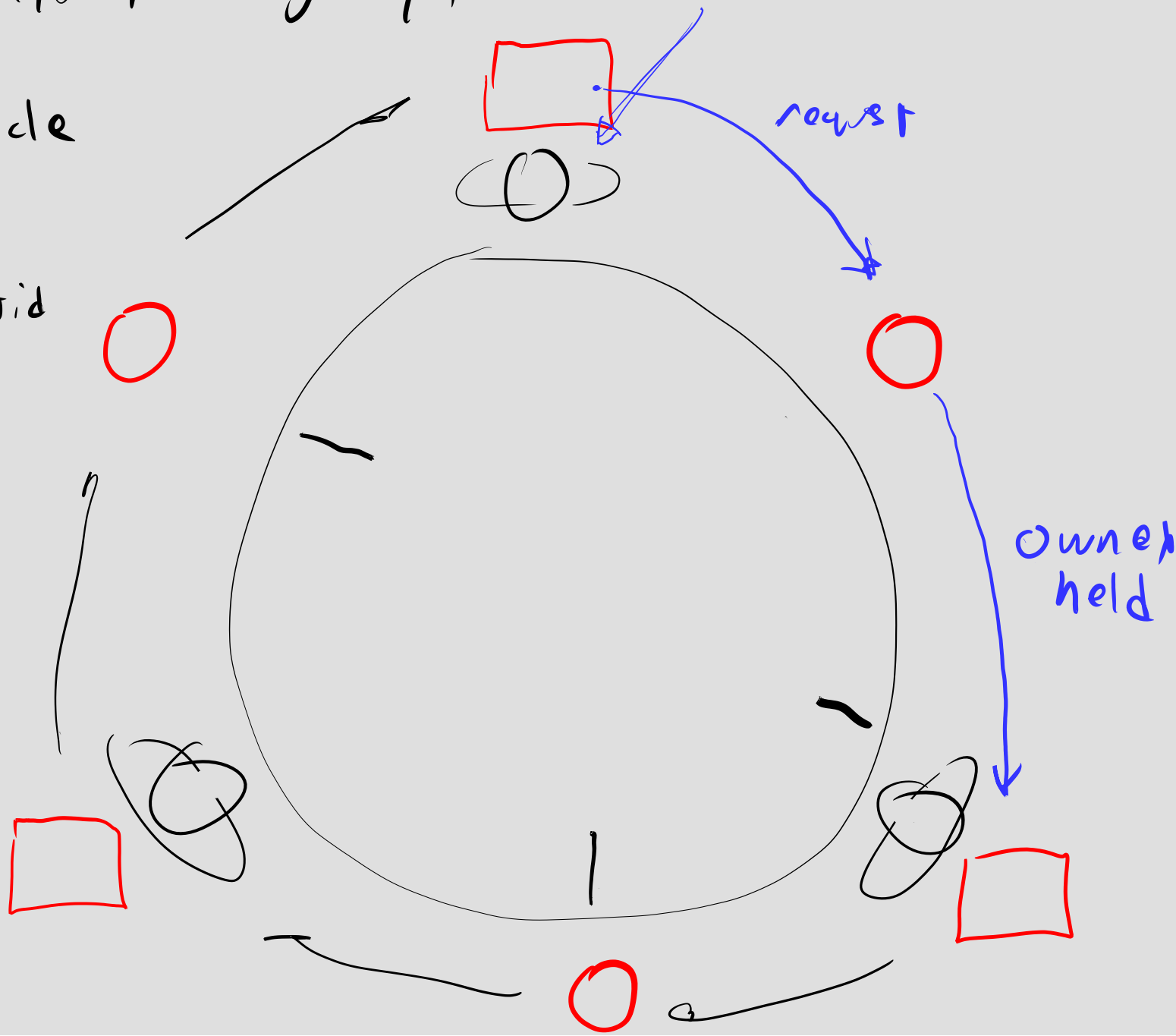
Resource Allocation graph

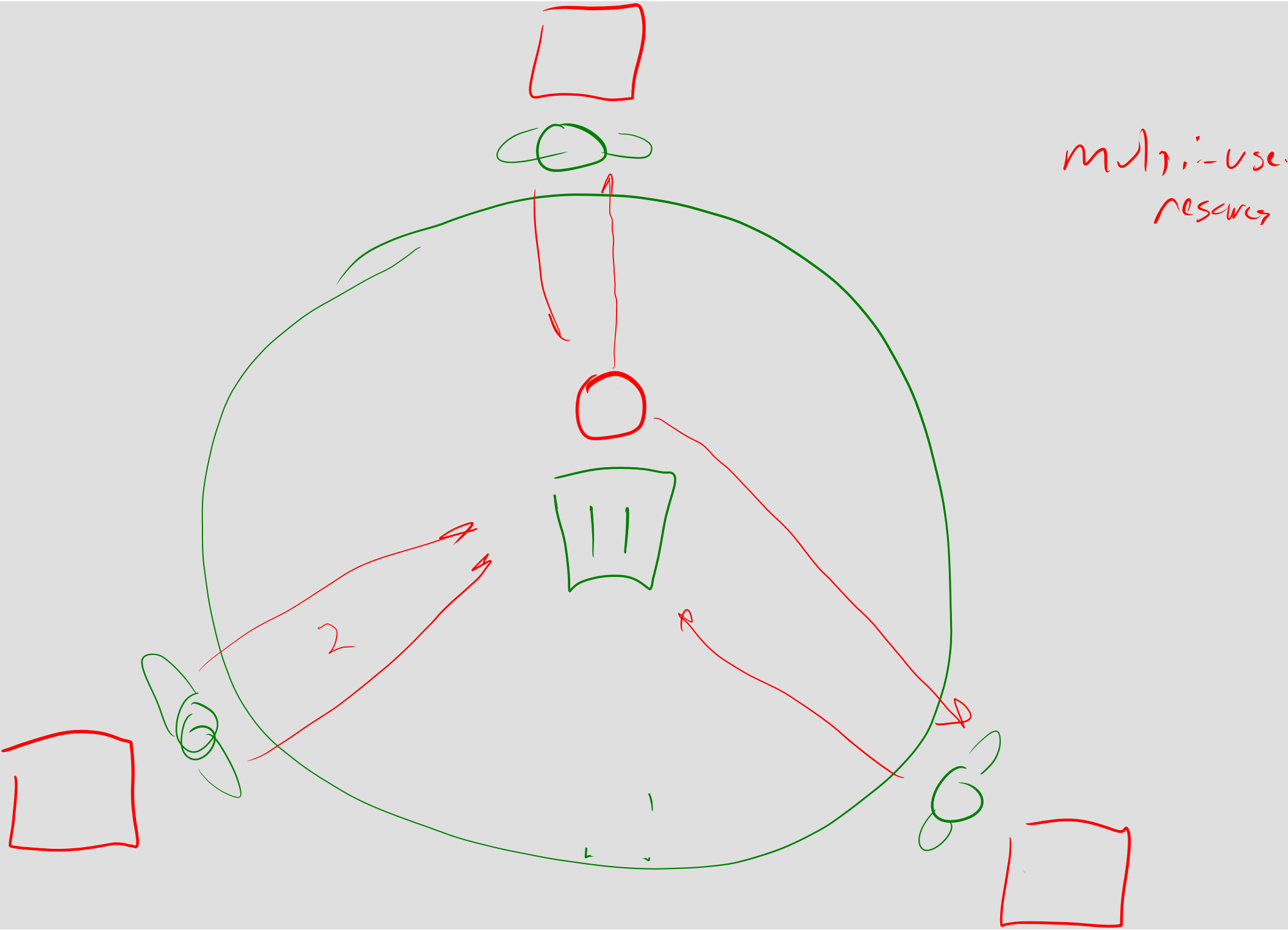
deadlock = cycle

change graph

lock – res, tid

unlock




multi-use
reserv
2
1

dead lock resolution

steal a resource — lock to prevent R.C

↳ say "respawn" → transaction rollback

kill a process/thread

↳ exception

Yell at Programmer

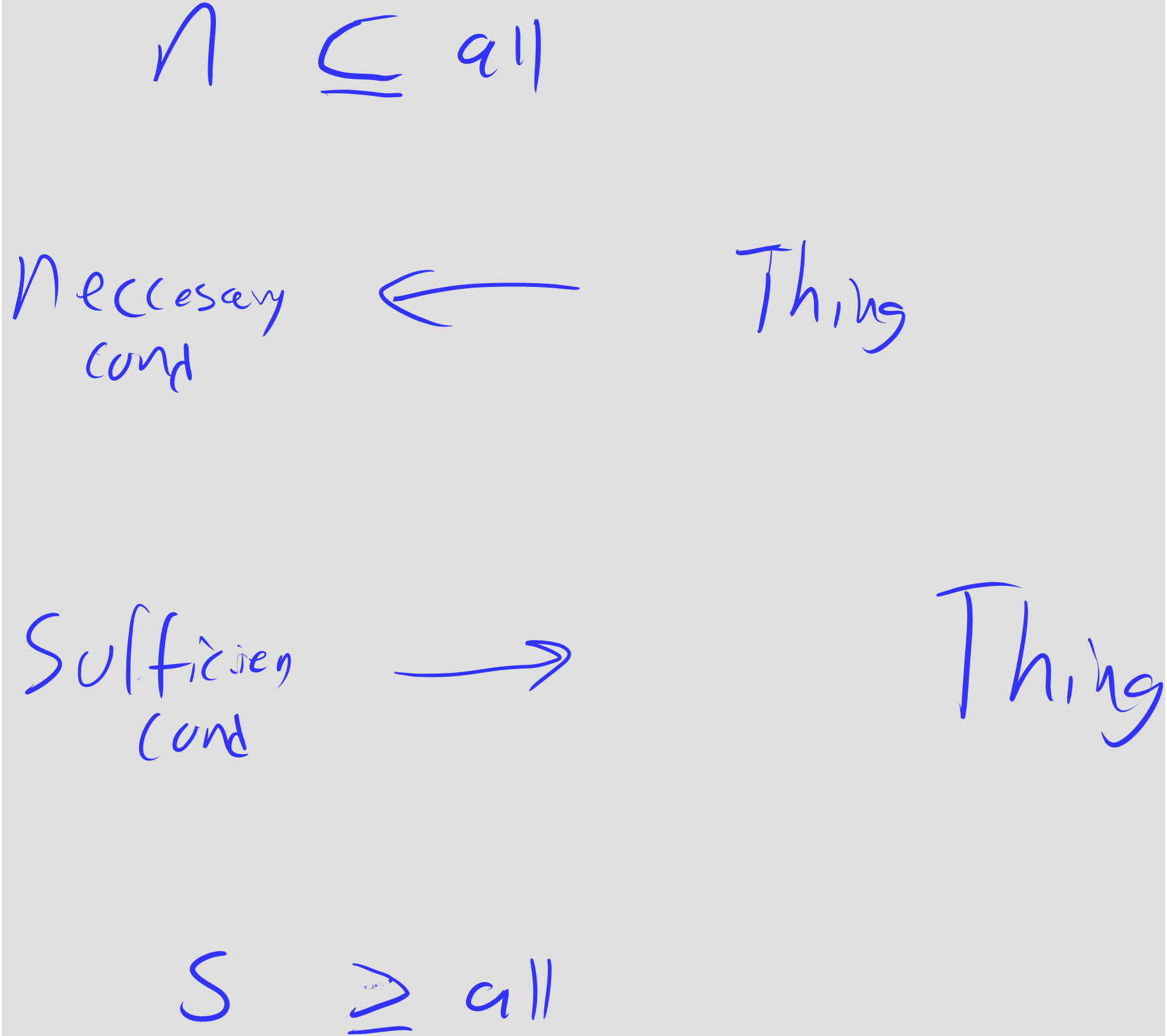
↳ detect before run

```
try {
    :
    lock( )
    :
} except (LostLock ex) {

}
```

Necessary Cond

1. Mutual Exclusion — some resours that can only be used by finite number of agents at a time — unit
2. Hold-and-wait — some process — has 1 resa try to get another — 2 locks
3. Cycles — graph must be able to have cycles
4. no preemption — can't steal — unit ∞


N ⊆ all
Neccesary cond
Thing
Sufficien cond
Thing
S ≥ all

Static - over-approx

before we run

-O3

4 necc cond

---

message-passing approach

API - exposed set of methods

#pragma

Dynamic

during a run

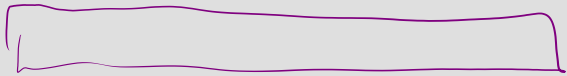
JIT

cycle-based

decrem

Send (Thread id, bytes)

(Tid , bytes) recv ( ) , .



lock-free queue