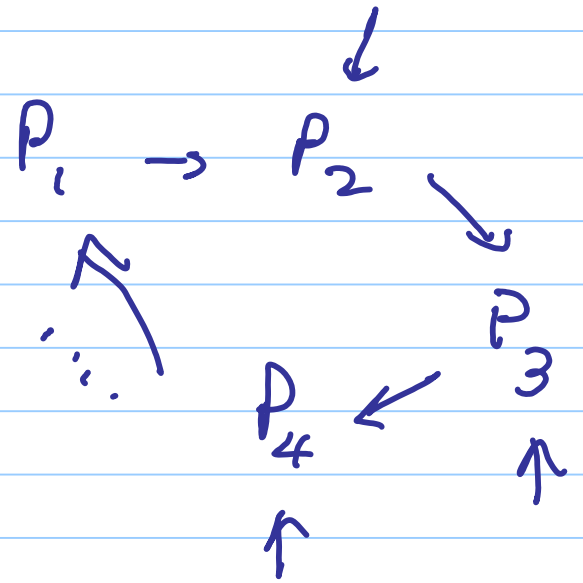
Lock server
- get & release locks (exclusive locks)
- deadlocks
  - detection
  - resolution

↳ terminate one process
↳ Send a 'not OK' to a lock request



'NOT OK' for resource R_i To P_j

↳ release <u>all</u> resources from P_j

↳ terminate P_j & restart

↳ will it get into a deadlock again?

→ maybe! –

→ forward progress

Exclusive locking + shared locks

↳ lock-x, lock-sh

unlock,

wait till
no process holds
a lock (x or sh)

wait till
no process
holds an x lock

↳ may starve
↳ do not grant
if x-lock req pendu
pending

lock_x → wait for all locks to clear

sh lock preference

↓

unlock → grant all sh locks pending
(if nothing then grant x lock
if nothing unlock

lock_sh – wait for x_lock to clear
also wait if x_lock req pendig

↓

unlock → last sh lock ? → NO → (nothing)
→ yes → grant an xlock if any
→ if none → unlock it

process P_i has a sh lock on R_i

↳ can P_i request an x lock on R_i

↳ P_i releases sh lock & gets x lock?

(NO!)

CASE 1: release & re-request x lock

CASE 2: [get x lock without releasing (upgrade)]

upgrade sh to x

↳ • ignore waiters

• Conversion has to wait for existing sh locks to clear

• P_i has sl ock on R_i

• P_i wants to upgrade

• But P_j has shared lock on R_i

• P_j wants to upgrade

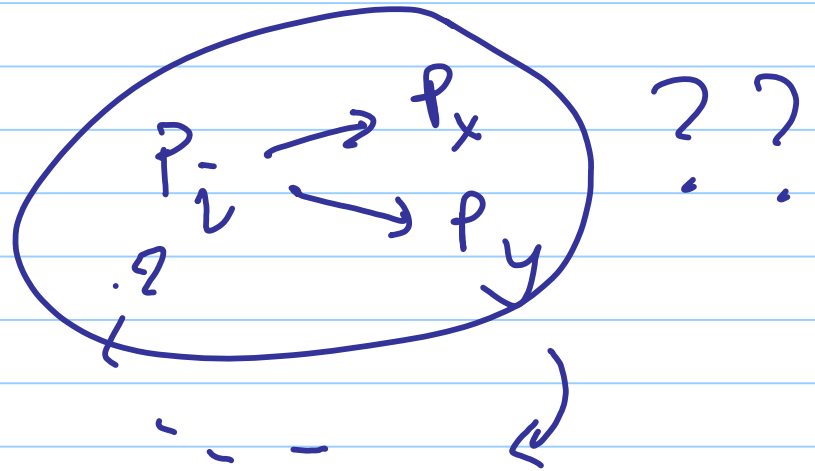
??? - deadlock

Sh, x locks no upgrades

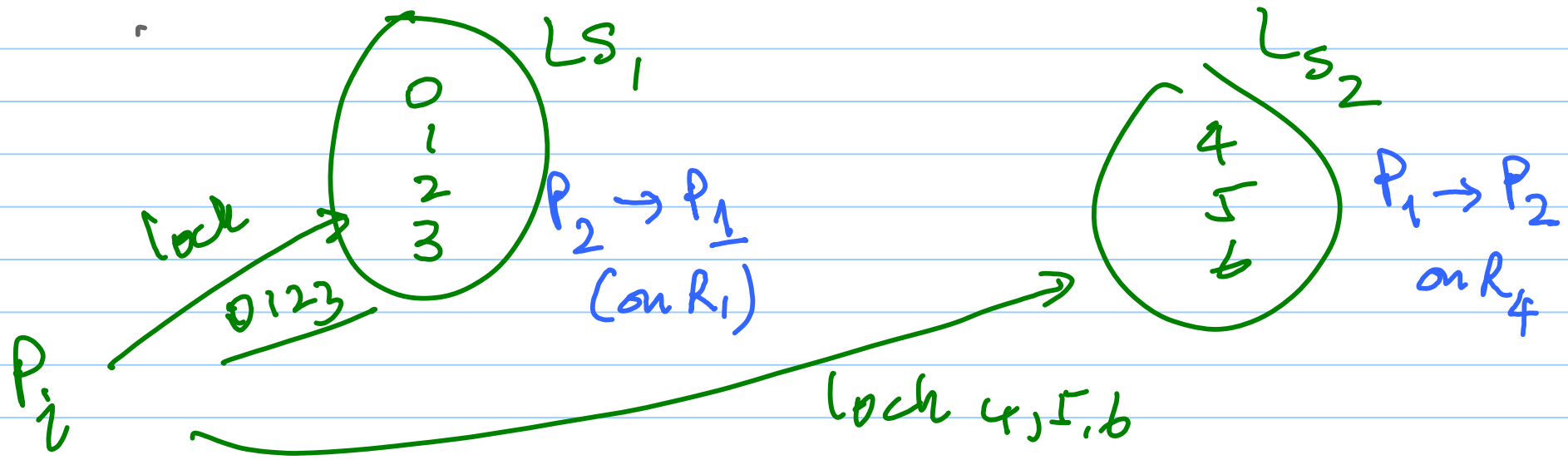
→ deadlocks

P_i → wants x lock on R_i

P_x, P_y have shared locks



More than 1 lock server



P_1 locks R_1 & wants R_4

P_2 locks R_4 and wants R_1

multiple lock servers

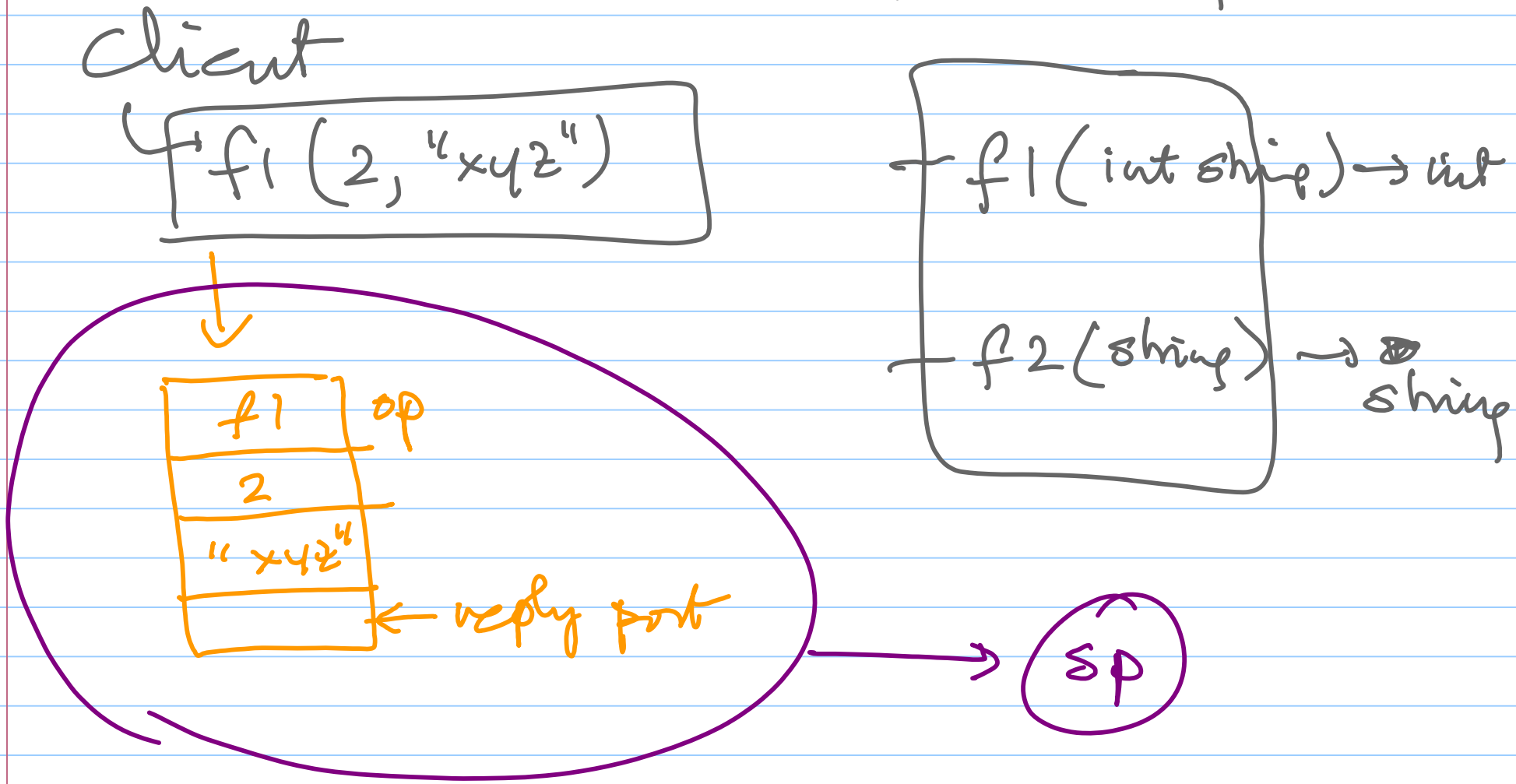
→ distributed deadlock detection

→ how

① try to coalesce graphs (get false deadlocks)

② use probes.

RPC / RMI → equivalent to client server msg based programming

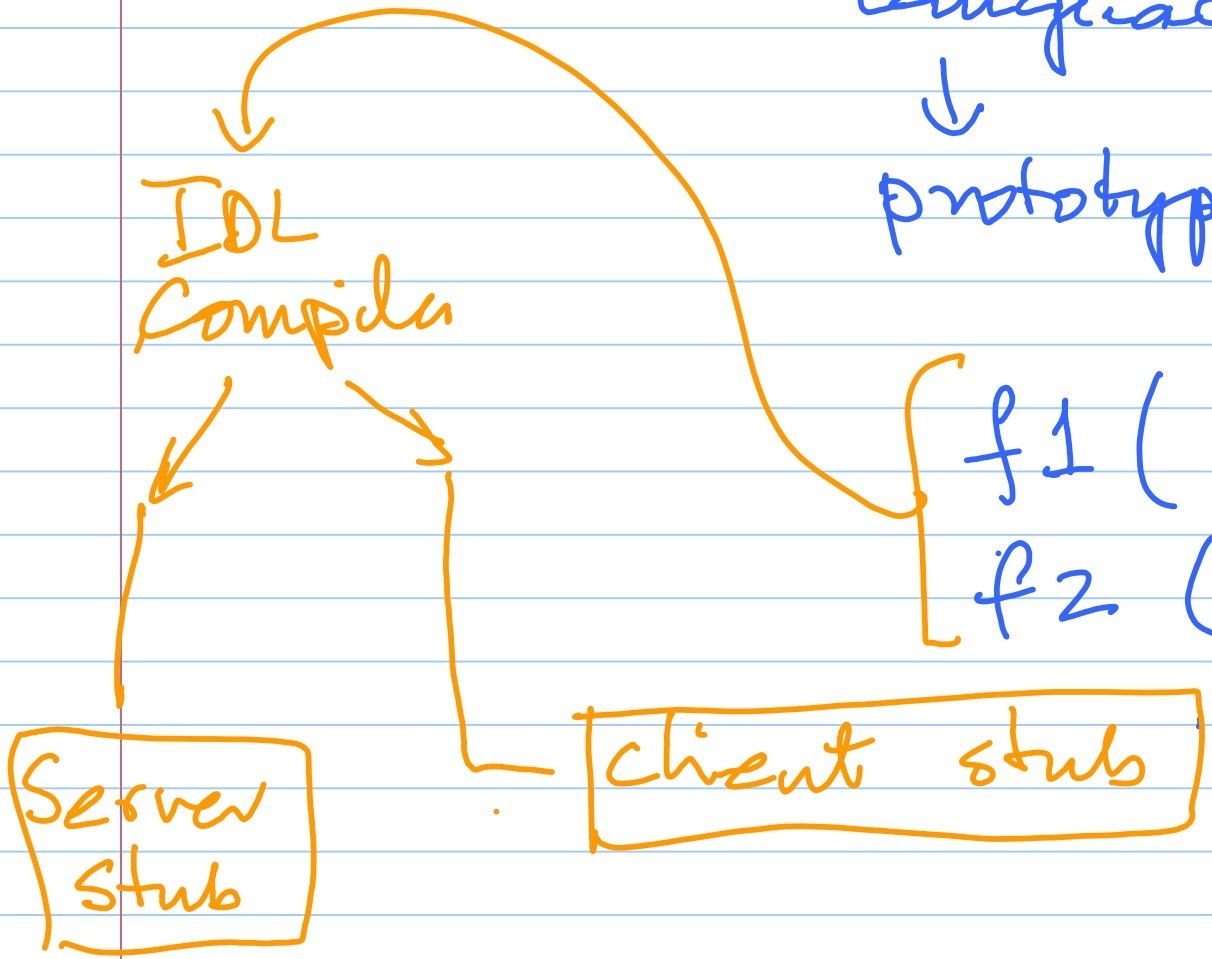


IDL

→ interface definition language

↓

prototypes of server fn's



client

include

f1( )

{ package args( )

Send (sp,

rcv (rp,

unpack ( )

client main

{ f1 (2, "xyz"

Server

f1( )

{

{

server loop

{ register sp

loop

include rcv( )

f1

Send