Guardians and Actions: Linguistic Support for Robust, Distributed Programs
Liskov and Scheifler, ToPLS 1983

big idea:
  language support for distributed programs
  only useful if it solves significant problems
  and if solutions are general
  RPC/RMI, transactions, sub-actions, locking, persistence, crash recovery

want to illustrate:
  guardian == RMI object
  RMI-like transparent guardian references, args, &c
  has better story than RMI for failure
  RPC always has prepare/commit
  sub-actions: why?
  versions: why?
  implicit locking

walk through send mail to many users example
  what if one user doesn't exist?
    but mail has already been delivered to some other users
    how to un-do?
  why do nested transactions make sense?
    are they just a feature? or neccessary?
    i.e. xactions + RPC => nested transactions?
    required by modularity?
      you don't know who is calling, but you want to be atomic
  what if a guardian crashes after RPC return, before final commit?
    or while commiting?
  what if concurrently one reader reads his/her mail?
    how does user not see tentative new mail?
    does reading user block? where?
  read_mail also deletes it
    what if new mail arrives just before delete?
    will it be deleted but not returned?
    why not? what lock protects? where is the lock acquired? released?
  what if a user is on the list twice?
    locks are held until end of top-level xaction
    deadlock?

stable variables are like DB data
  versions for abort, logged (?) to disk

crash recovery
  stable variables, per-guardian recovery