

Concurrency

Note Title

3/22/2007

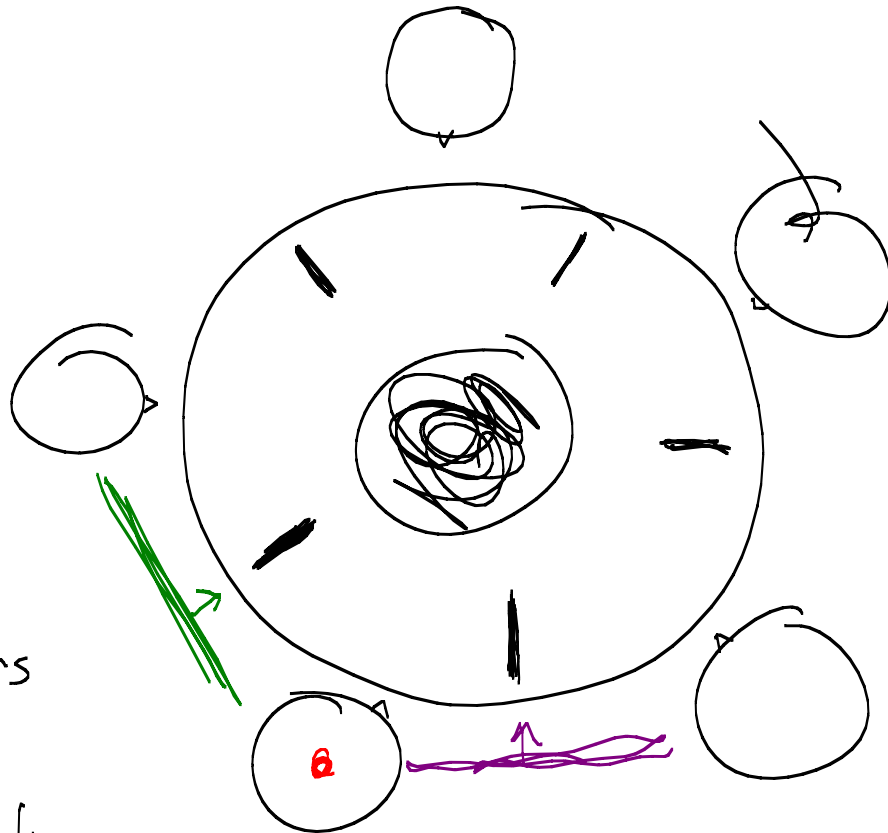
Performance vs. Simplicity

Conservative — clearly safe

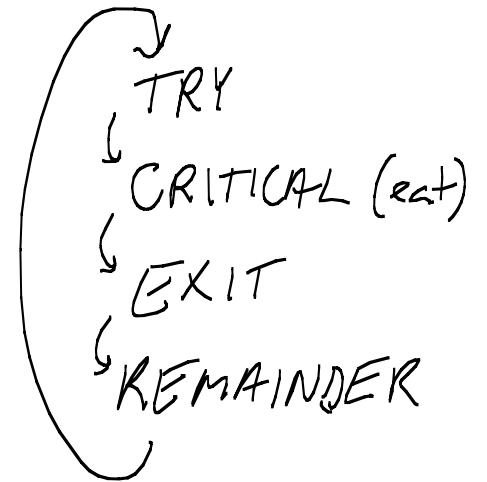
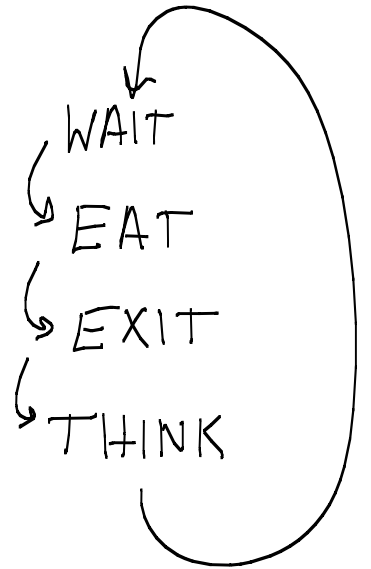
Aggressive — requires careful reasoning to show safety

Optimistic — assume things will be ok, but
detect & fix problems during execution

Dining Philosophers



Restriction:
No 2 neighbors
can eat
simultaneously



Idea 1: eat one at a time

use a lock —

WAIT: lock.lock();

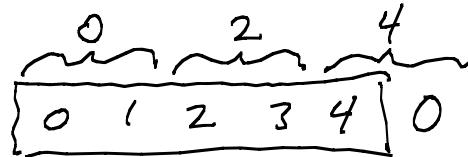
DONE: lock.unlock();

Idea 2:

Use 2 locks for each philosopher
(chopsticks)

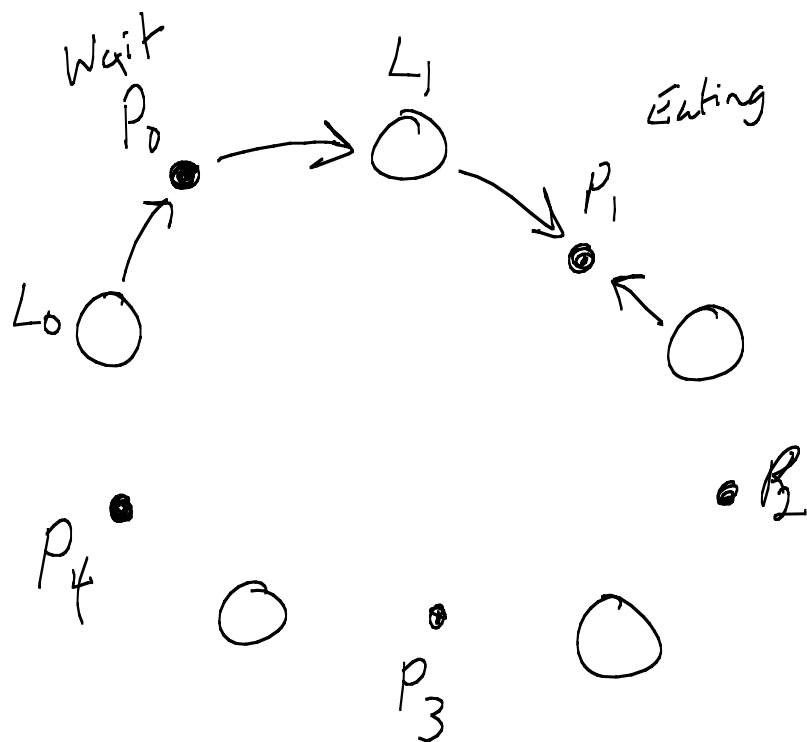
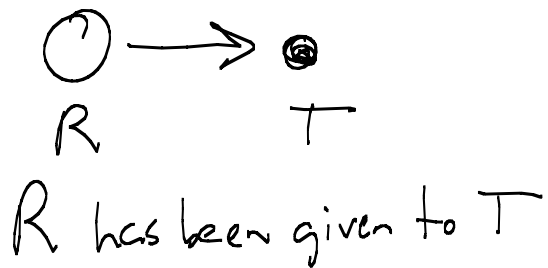
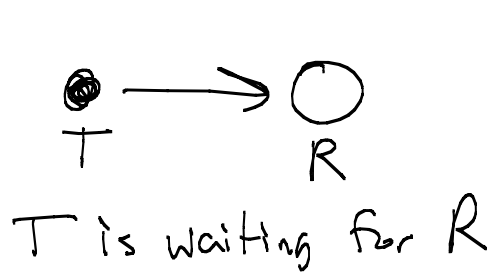
WAIT: get both chopsticks

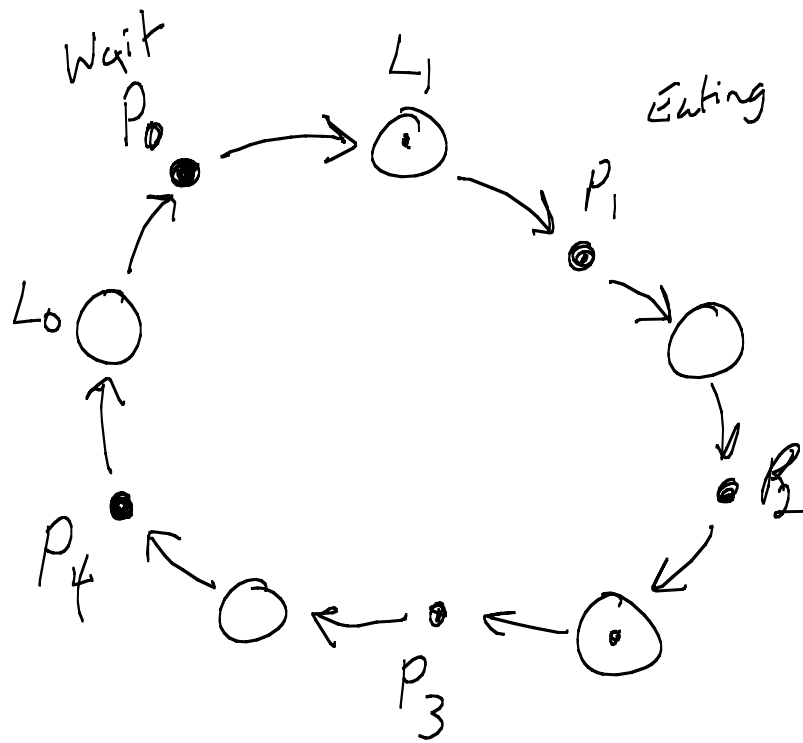
DONE: release them



Resource Allocation Graph

Vertex = Thread OR Resource (lock)



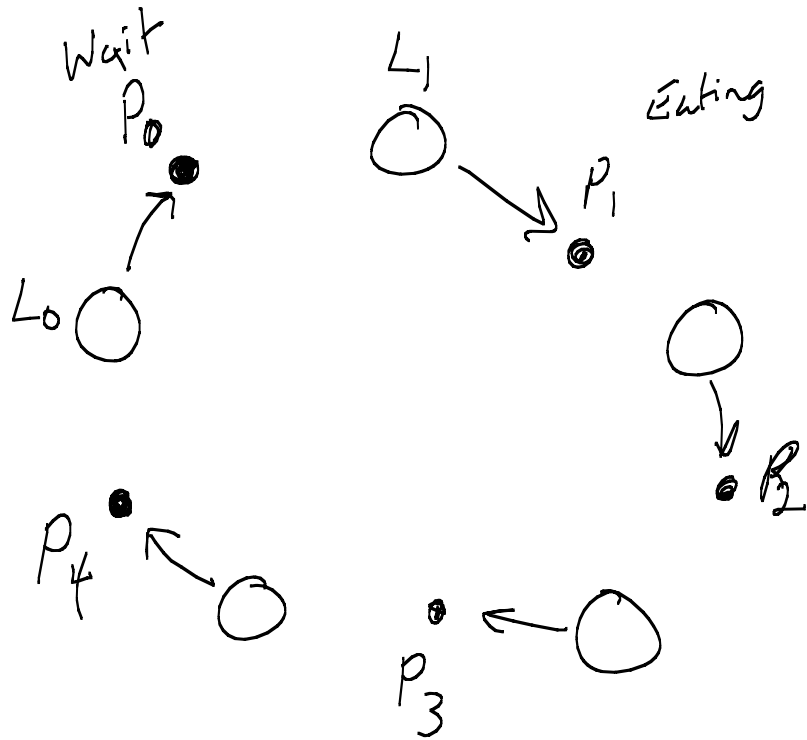


Cycle in resource allocation graph
 \Rightarrow deadlock

- ① Let each philosopher pick up the 1st chopstick
- ② Let each philosopher continue running — they all reach for the other chopstick & wait
- ③ DEADLOCK!

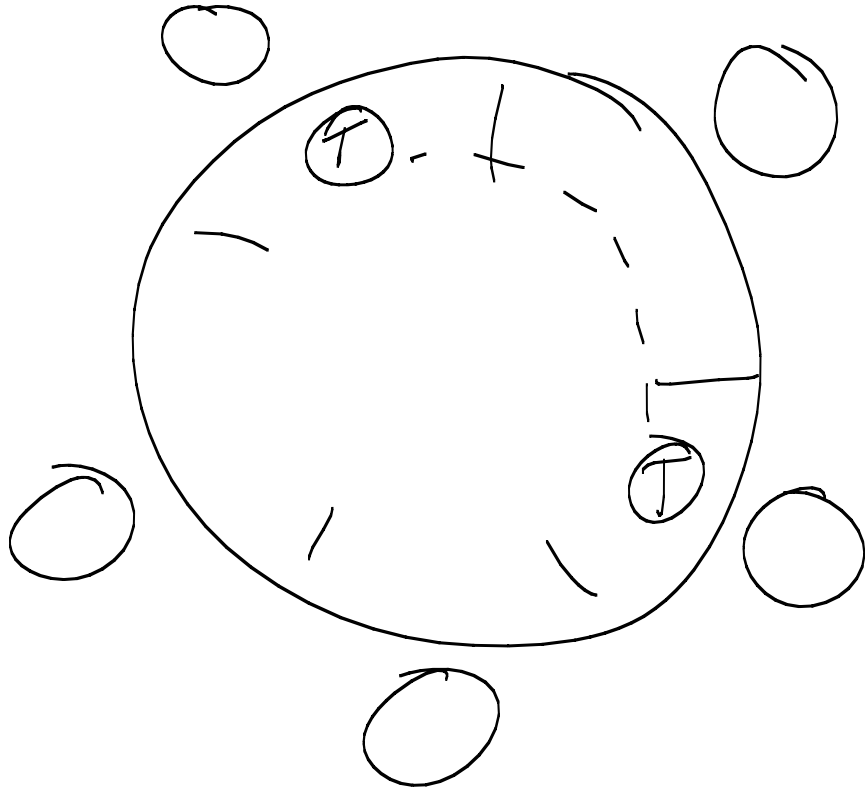
One solution: Give up locks if waiting a while

⇒ could be slow? — giving up locks = negative progress

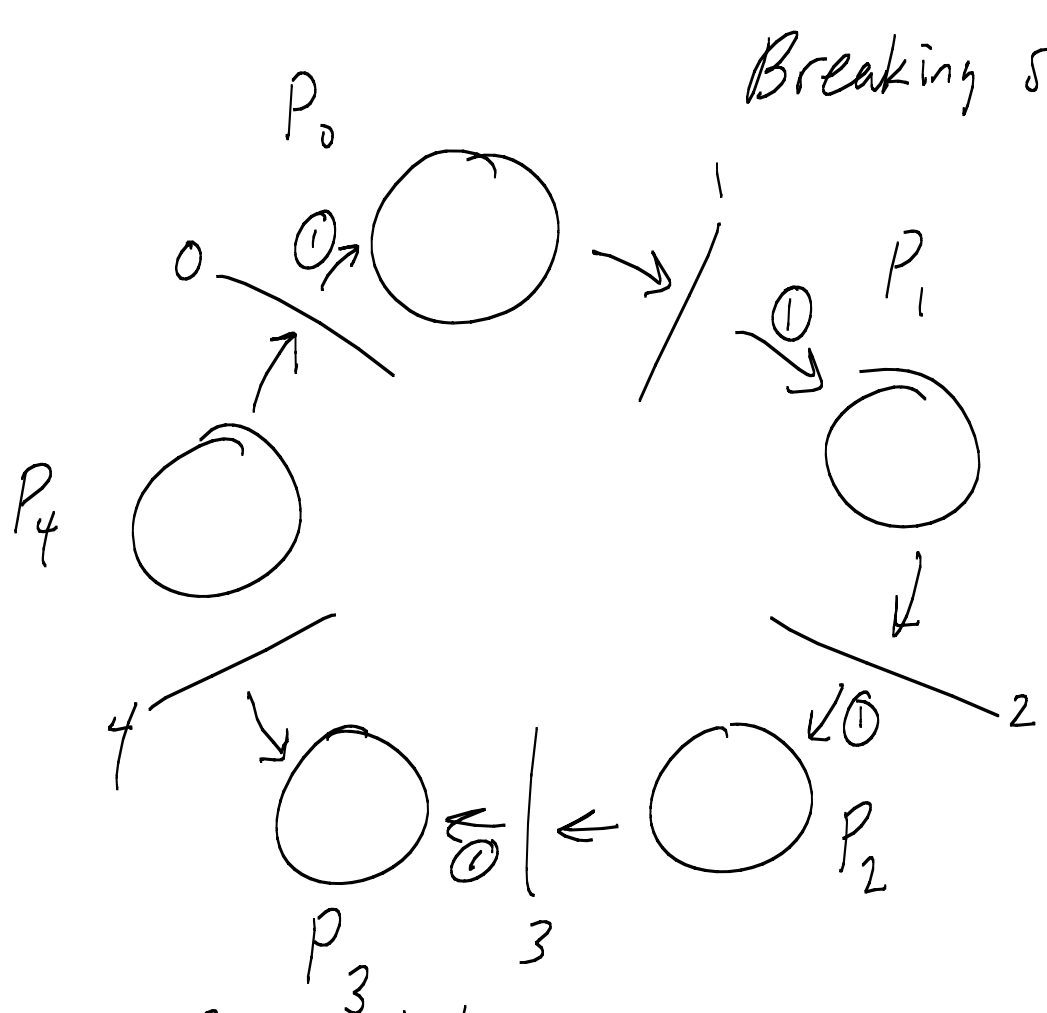


⇒ LIVEDLOCK

Another idea: Token Passing or Arbiter
(Different options)



Fairness



Breaking symmetry avoids deadlock

take own first
unless you're
the P4

Resource allocation
graph needs a
cycle for deadlock
to occur.

If there is a deadlock
HERE, then all
philosophers are involved
[P0 or P4 can't both be waiting &
holding a fork

Requirement for deadlock:
Be holding a resource &
waiting for another

Generalize:

Many locks. Don't know in advance what set of locks we need.

WAIT: ① determine which locks we want
② get them — options: arbiter — slow
order the locks & acquire in order

