Concurrenty

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  

\[
\begin{array}{c|c|c|c|c|c}
0 & 2 & 4 & 0 & 1 & 3 \\
\end{array}
\]
Resource Allocation Graph

Vertex = Thread or Resource (lock)

T → O

T is waiting for R

O → R

R has been given to T

Wait

Eating

R2
1. Let each philosopher pick up the 1st chopstick
2. Let each philosopher continue running— they all reach for the other chopstick
   and wait
3. DEADLOCK!
One solution: Give up locks if waiting a while

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

⇒ LIVELOCK
Another idea: Token Passing or Arbiter (Different options)

Fairness
Requirement for deadlocks: Be holding a resource waiting for another.

If P0 or P4 can't both be waiting, it's a deadlock.

If, there is a deadlock, then all philosophers are involved.

Resource allocation graph needs a cycle for deadlock to occur.

Resource allocation avoids deadlock.

Breaking symmetry avoids deadlock.
Generalize:

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

WAIT: 0 determine which locks we want
② get them — options: arbitrator — slow order the locks & acquire in order