CSE 451: Operating Systems
Spring 2010

Module 9
Deadlock

John Zahorjan
zahorjan@cs.washington.edu
Allen Center 534

(Is Google the greatest, or what?)

Definition

- A thread is deadlocked when it's waiting for an event that can never occur
  - I'm waiting for you to clear the intersection, so I can proceed
    - but you can't move until he moves, and he can't move until she moves, and she can't move until I move
  - thread A is in critical section 1, waiting for access to critical section 2; thread B is in critical section 2, waiting for access to critical section 1
  - I'm trying to book a vacation package to Tahiti – air transportation, ground transportation, hotel, side-trips. It's all-or-nothing – one high-level transaction – with the four databases locked in that order. You're trying to do the same thing in the opposite order.

Requirements (to have deadlock)

1. Mutual Exclusion

2. Hold and Wait

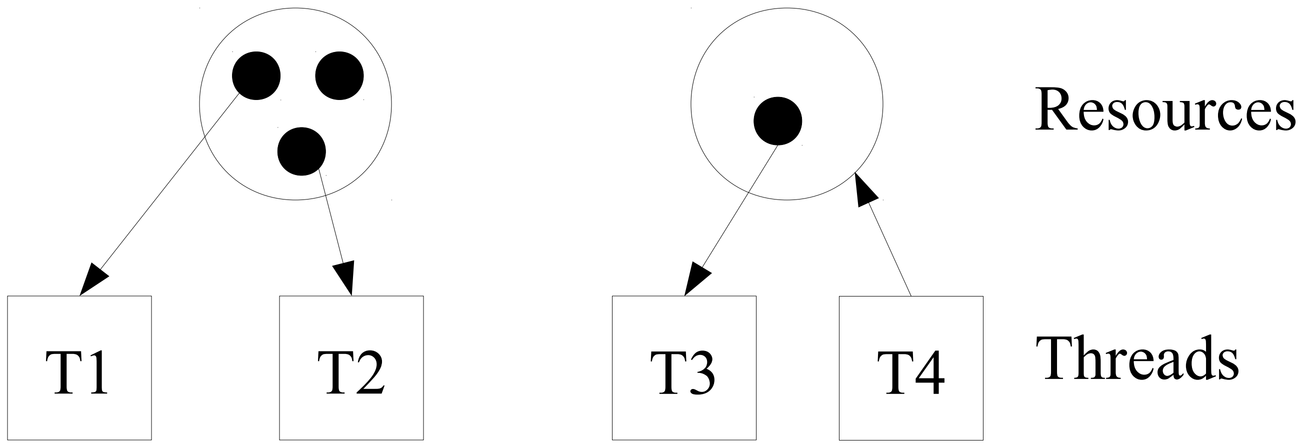
3. No Preemption

4. Circular Wait

We'll see that deadlocks can be addressed by attacking any of these four conditions.

Resource graphs

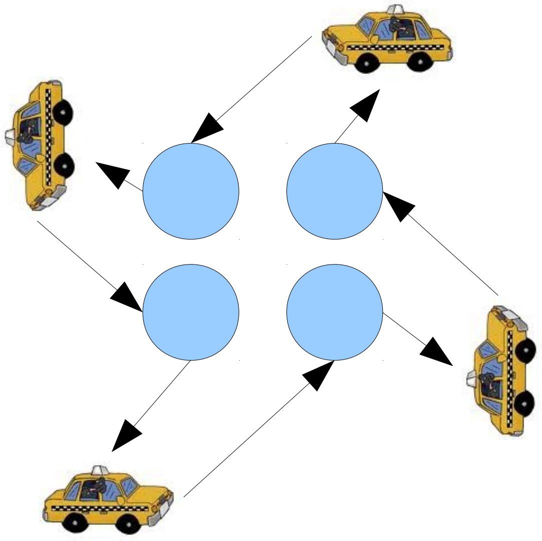
- Resource graphs are a way to visualize the (deadlock-related) state of the threads, and to reason about deadlock



- 1 or more identical units of a resource are available
- A thread may hold resources (arrows to threads)
- A thread may request resources (arrows from threads)

Cycles and deadlock

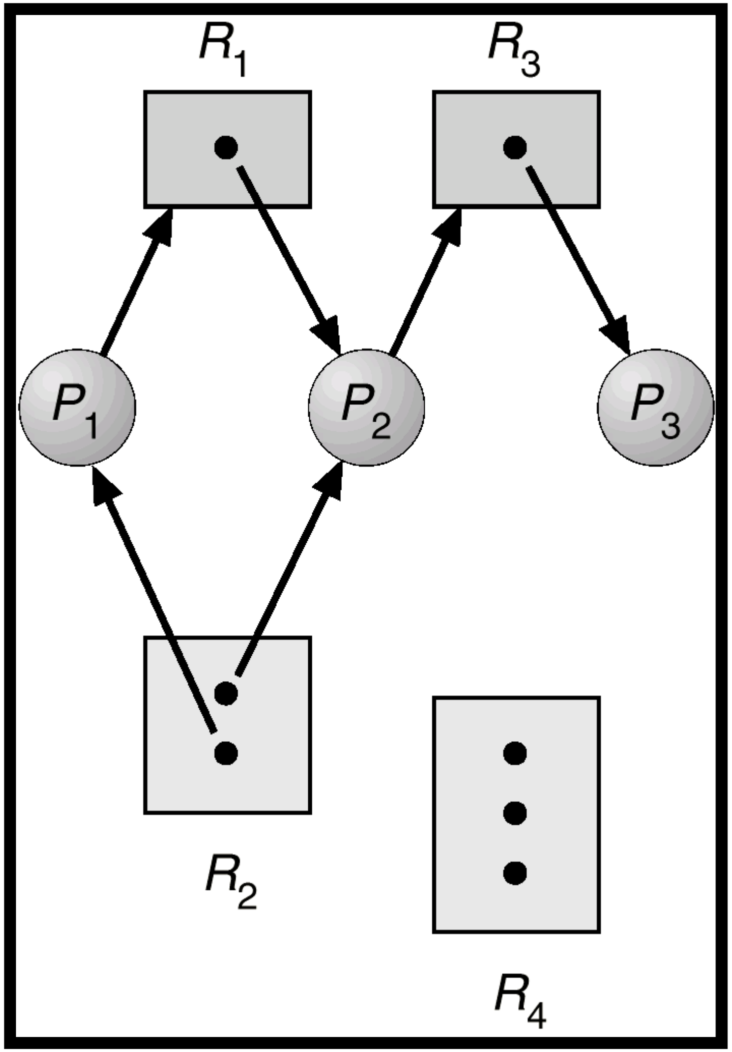
- Cycles in the graph are related to deadlock
  - There is no deadlock unless there is a cycle
- The city intersection example:



Graph reduction

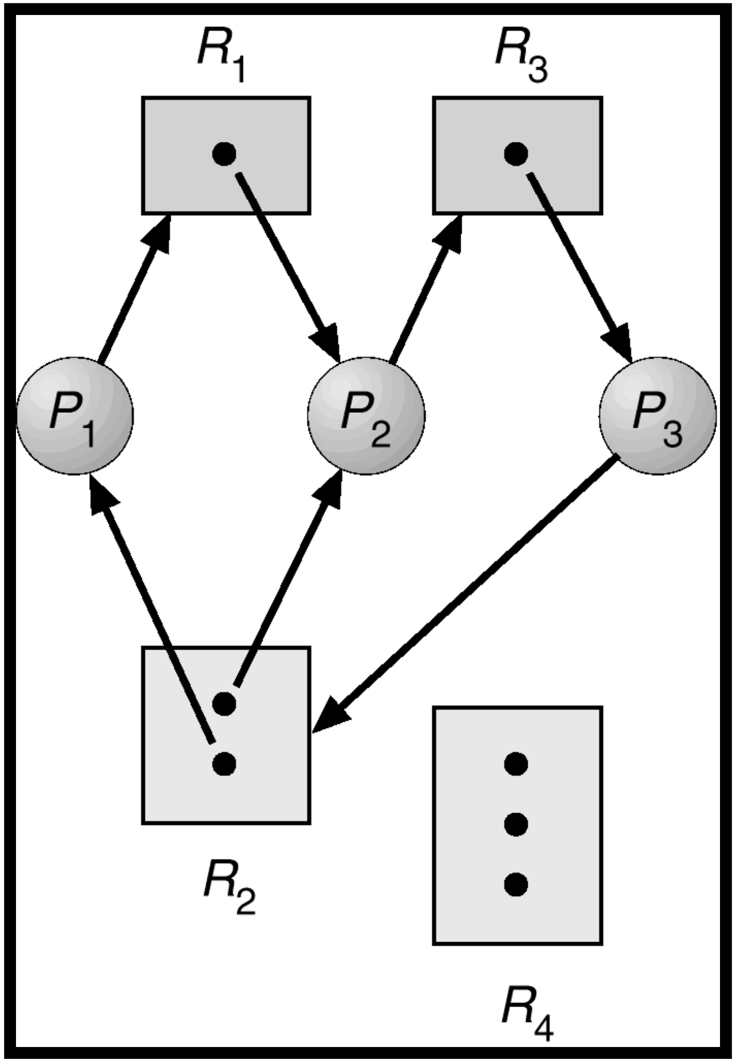
- A graph can be *reduced* by a thread if all of that thread's requests can be granted
  - in this case, the thread eventually will terminate – all resources are freed – all arcs (allocations) to it in the graph are deleted
- Miscellaneous theorems (Holt, Havender):
  - There are no deadlocked threads iff the graph is completely reducible
  - The order of reductions is irrelevant

Resource allocation graph with no cycle

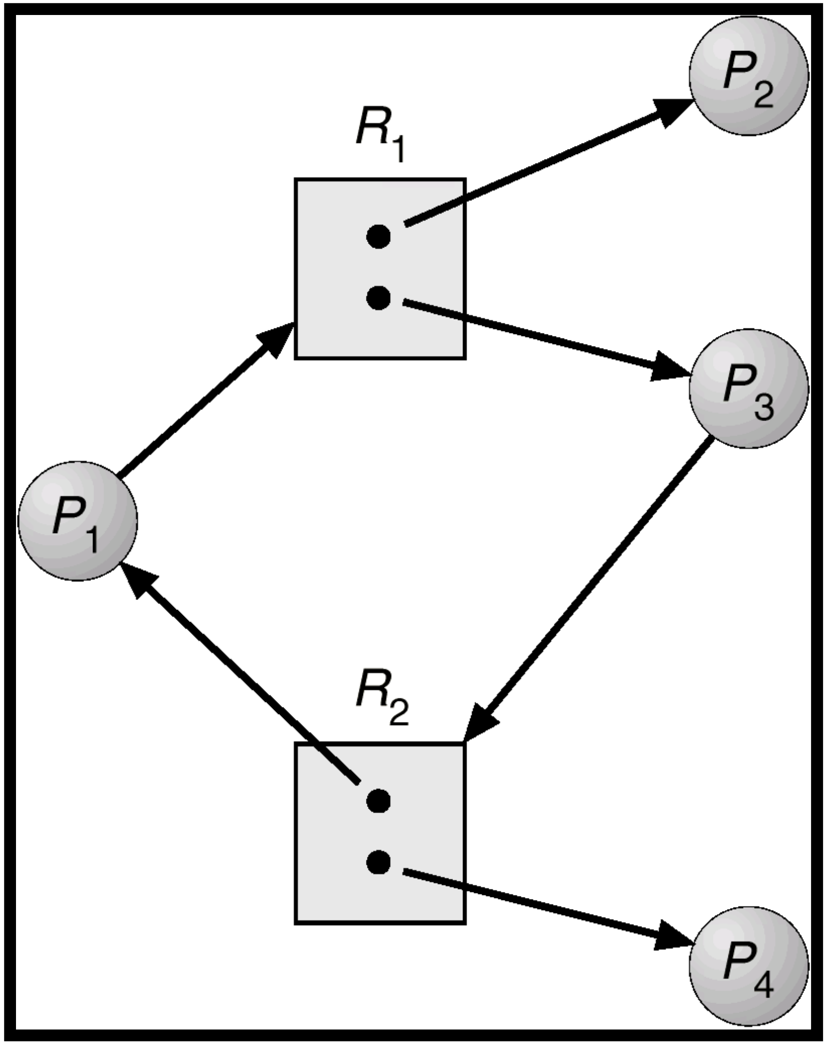


What would cause a deadlock?

Resource allocation graph with a deadlock



Resource allocation graph with a cycle but no deadlock



Approaches to Deadlock

- Break one of the four required conditions
  - Mutual Exclusion?
  - Hold and Wait?
  - No Preemption?
  - Circular Wait?

- Broadly classified as:
  - prevention, or
  - avoidance, or
  - detection (and recovery)

Prevention

Applications must conform to behaviors guaranteed never to deadlock.

- Hold and Wait
  - each thread obtains all resources "atomically" at the beginning
  - blocks until all are available
    - drawback?

- Circular Wait
  - resources are numbered
  - each thread obtains them in sequence (which could require acquiring some before they are actually needed)
    - why does this work?
    - pros and cons?

- Mutual Exclusion
  No Preemption
  – Application limited

Avoidance

Less severe restrictions on program behavior + system support

- Circular Wait
  - each thread states its maximum claim for every resource type
  - system runs the Banker's algorithm at each allocation request
    - Banker $\Rightarrow$ incredibly conservative
    - *if I were to allocate you that resource, and then everyone were to request their maximum claim for every resource, could I find a way to allocate remaining resources so that everyone finished?*
  - More on this in a moment…

Detection and Recover

- Every once in a while, check to see if there's a deadlock
  - how?

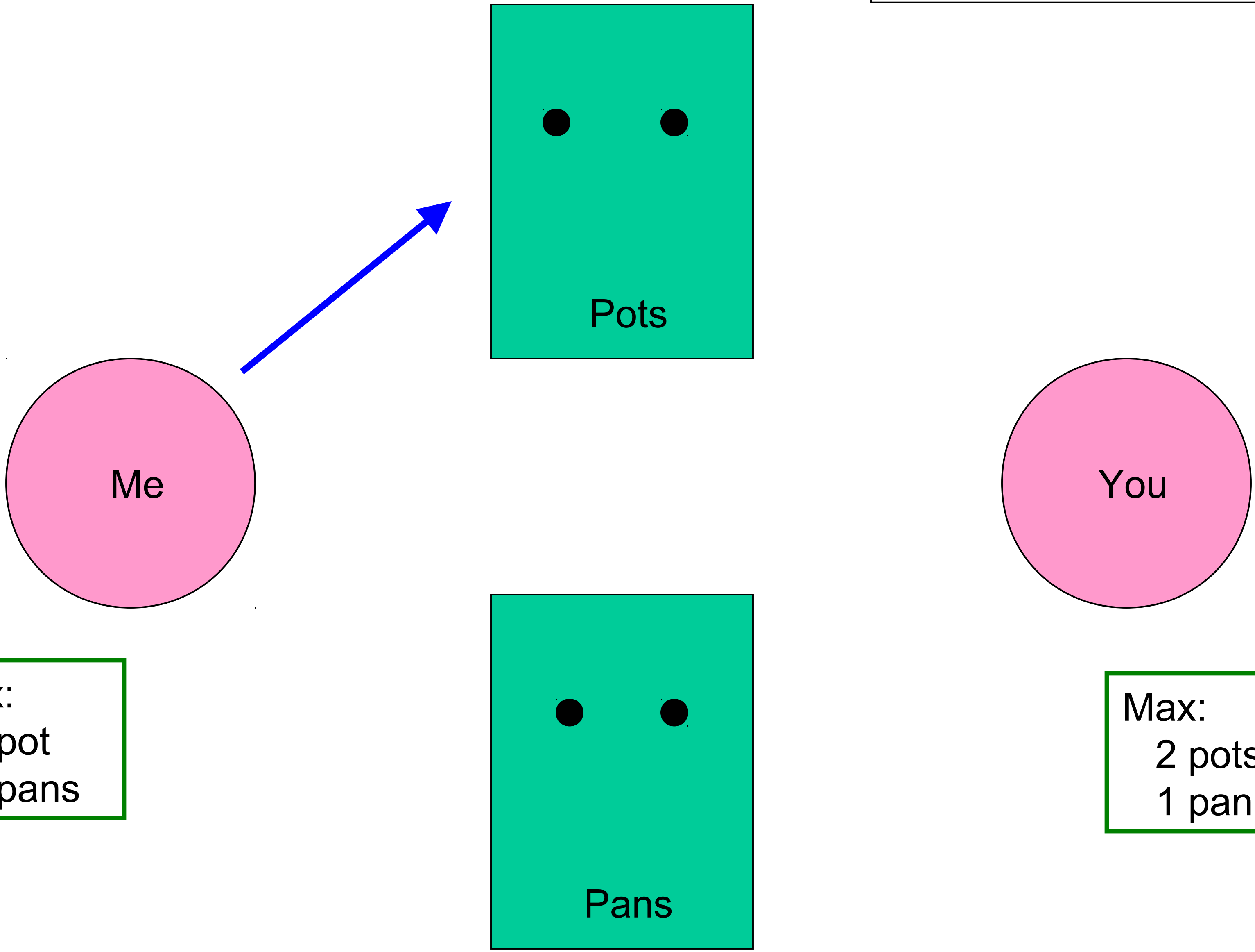
- if so, eliminate it
  - how?

Avoidance: Banker's Algorithm Example

- Background:
  - The set of controlled resources is known to the system
  - The number of units of each resource is known to the system
  - Each application must declare its maximum possible requirement of each resource type

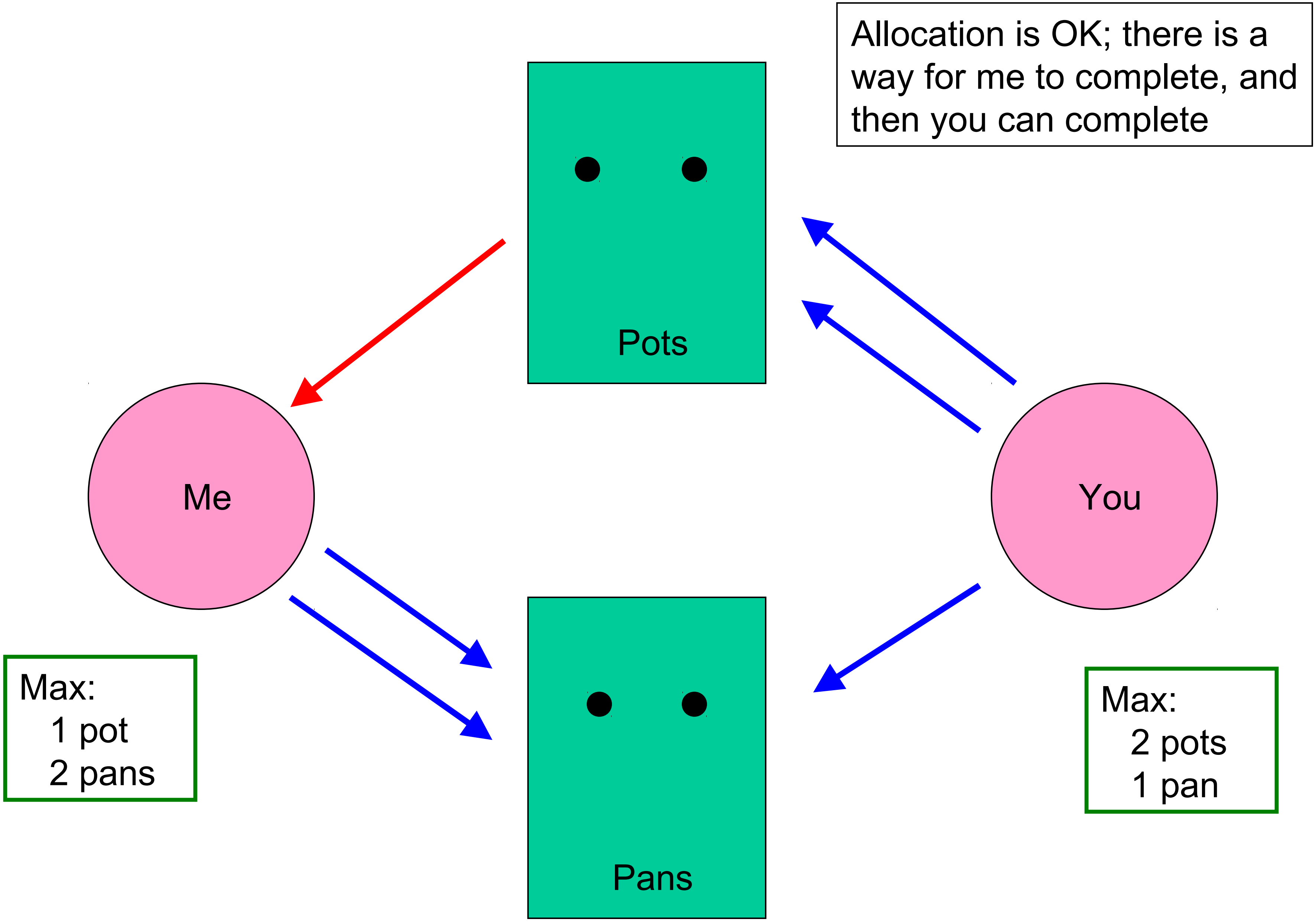
- Then, the system can do the following:
  - When a request is made
    - pretend you granted it
    - pretend all other legal requests were made
    - can the graph be reduced?
      - if so, allocate the requested resource
      - if not, block the thread until some thread releases resources and try pretending again

1. I request a pot

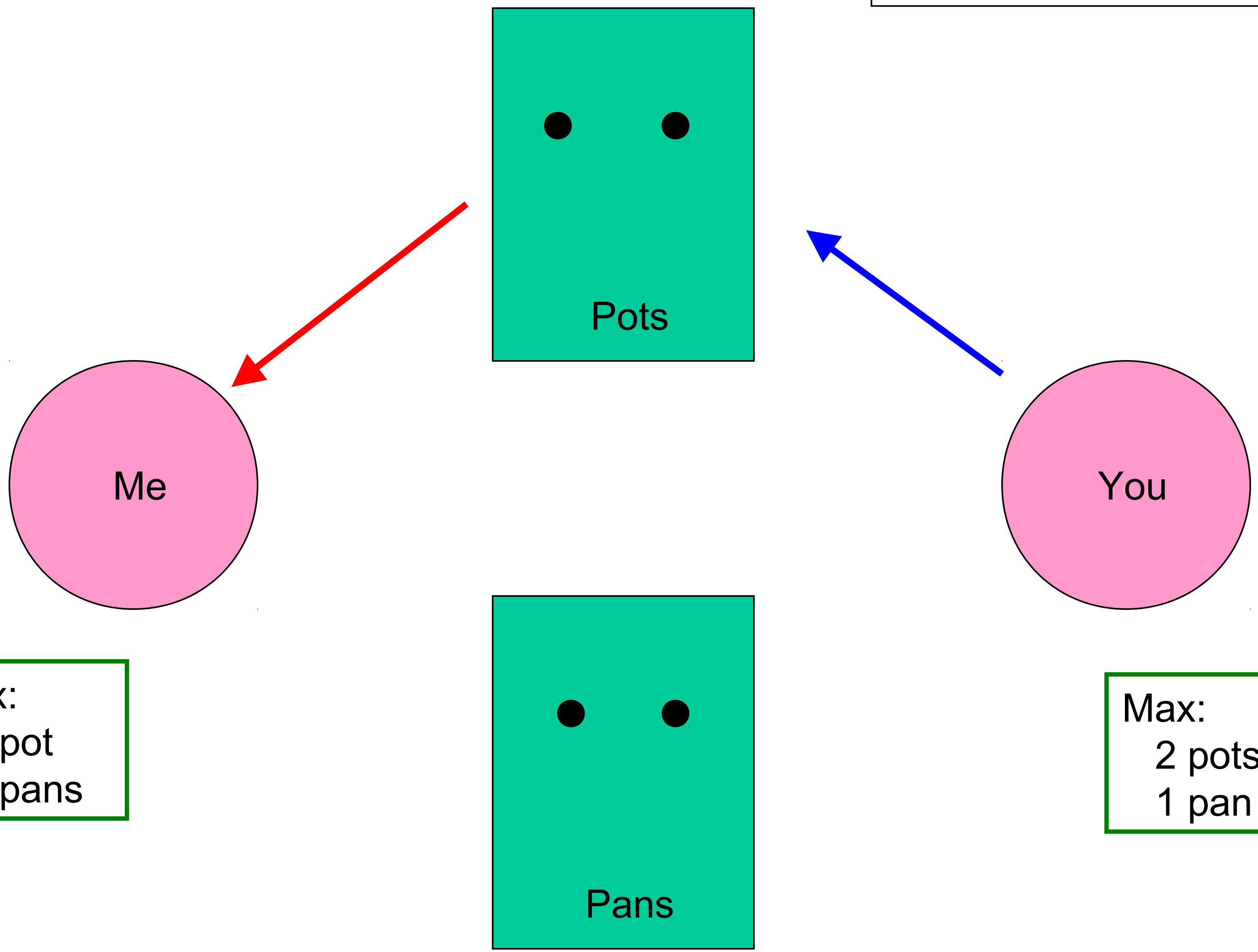


Max:
1 pot
2 pans

Max:
2 pots
1 pan

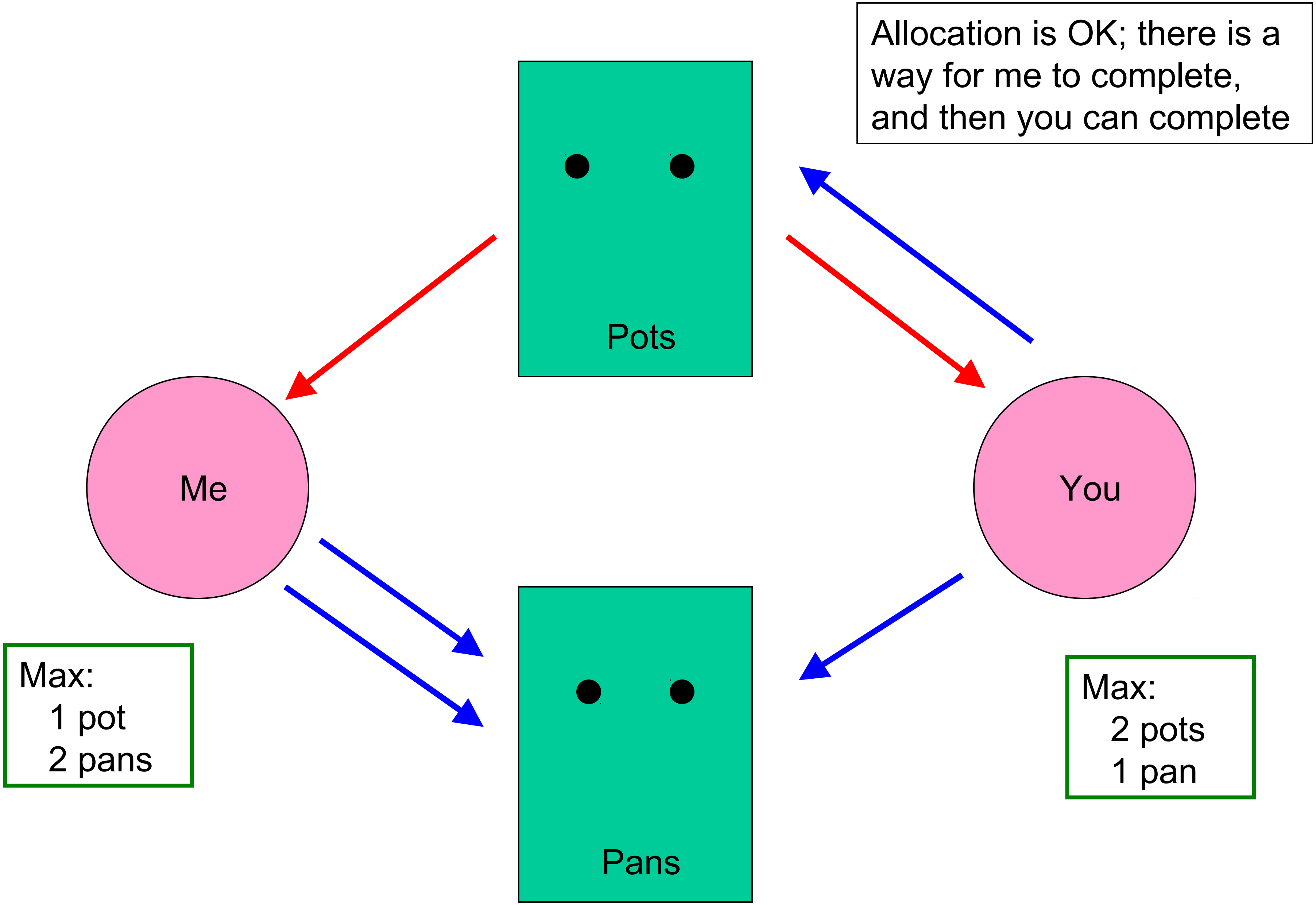

Allocation is OK; there is a way for me to complete, and then you can complete
Pots
Me
You
Pans
Max:
1 pot
2 pans
Max:
2 pots
1 pan

2. You request a pot



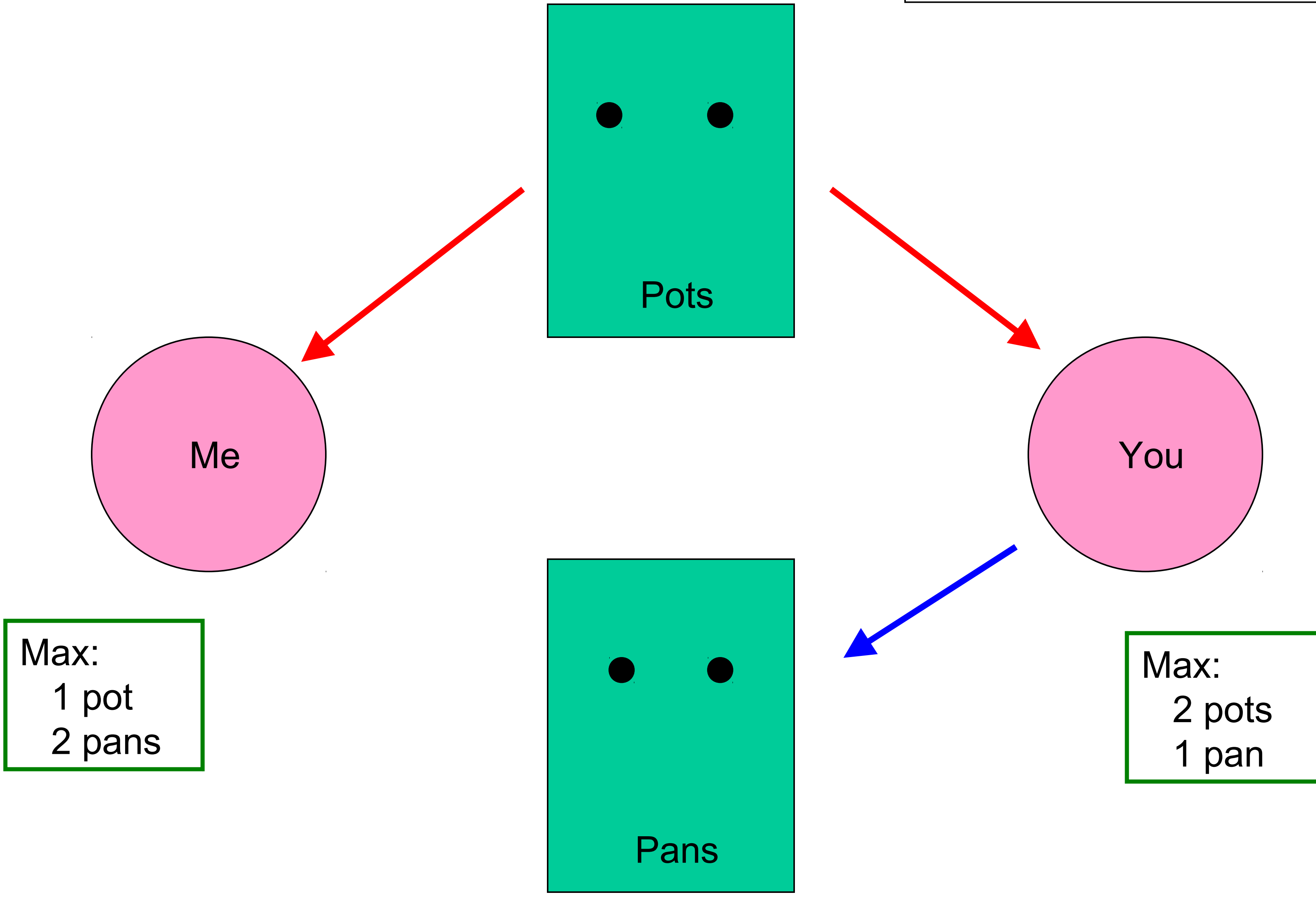
Max:
1 pot
2 pans

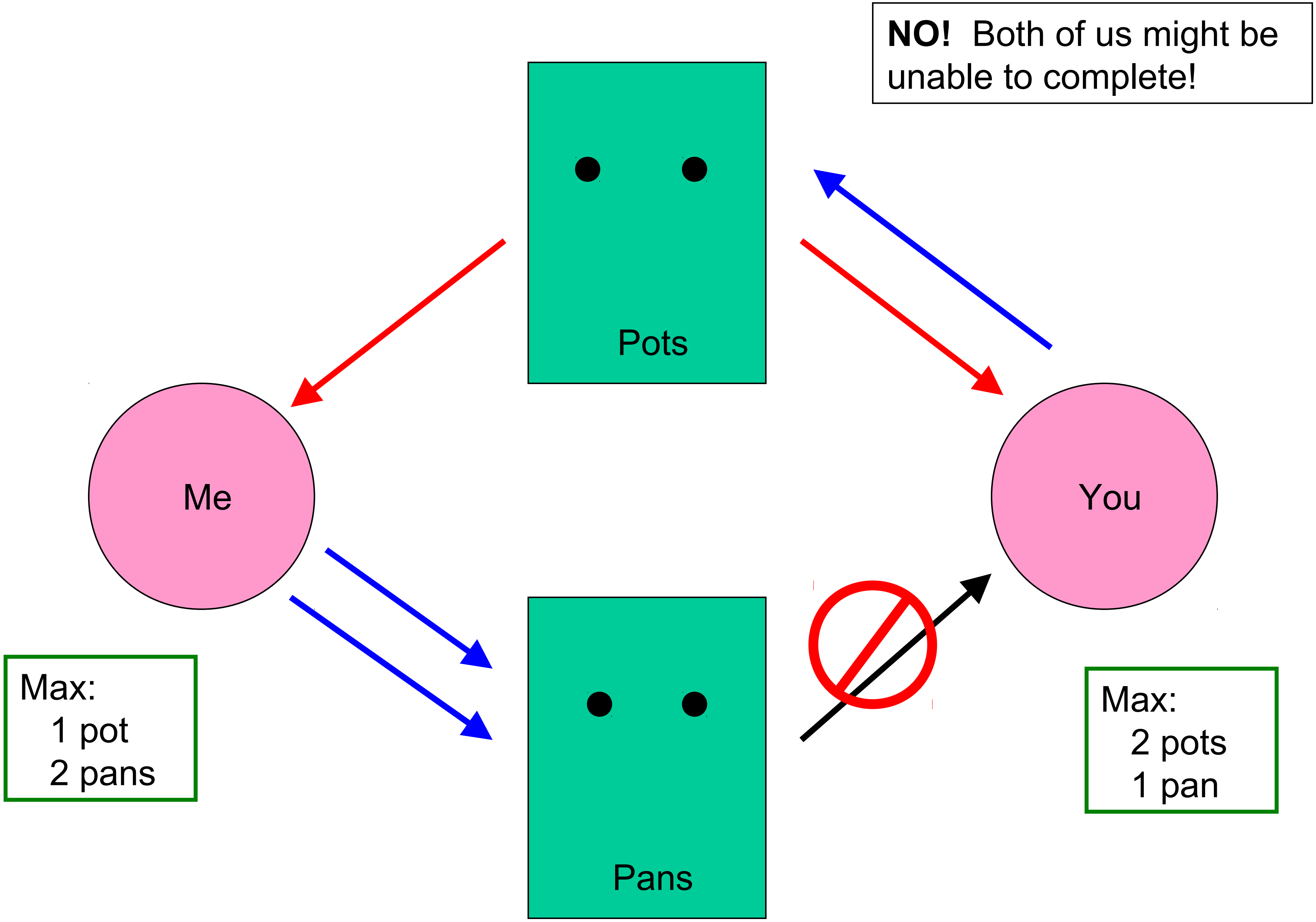
Max:
2 pots
1 pan

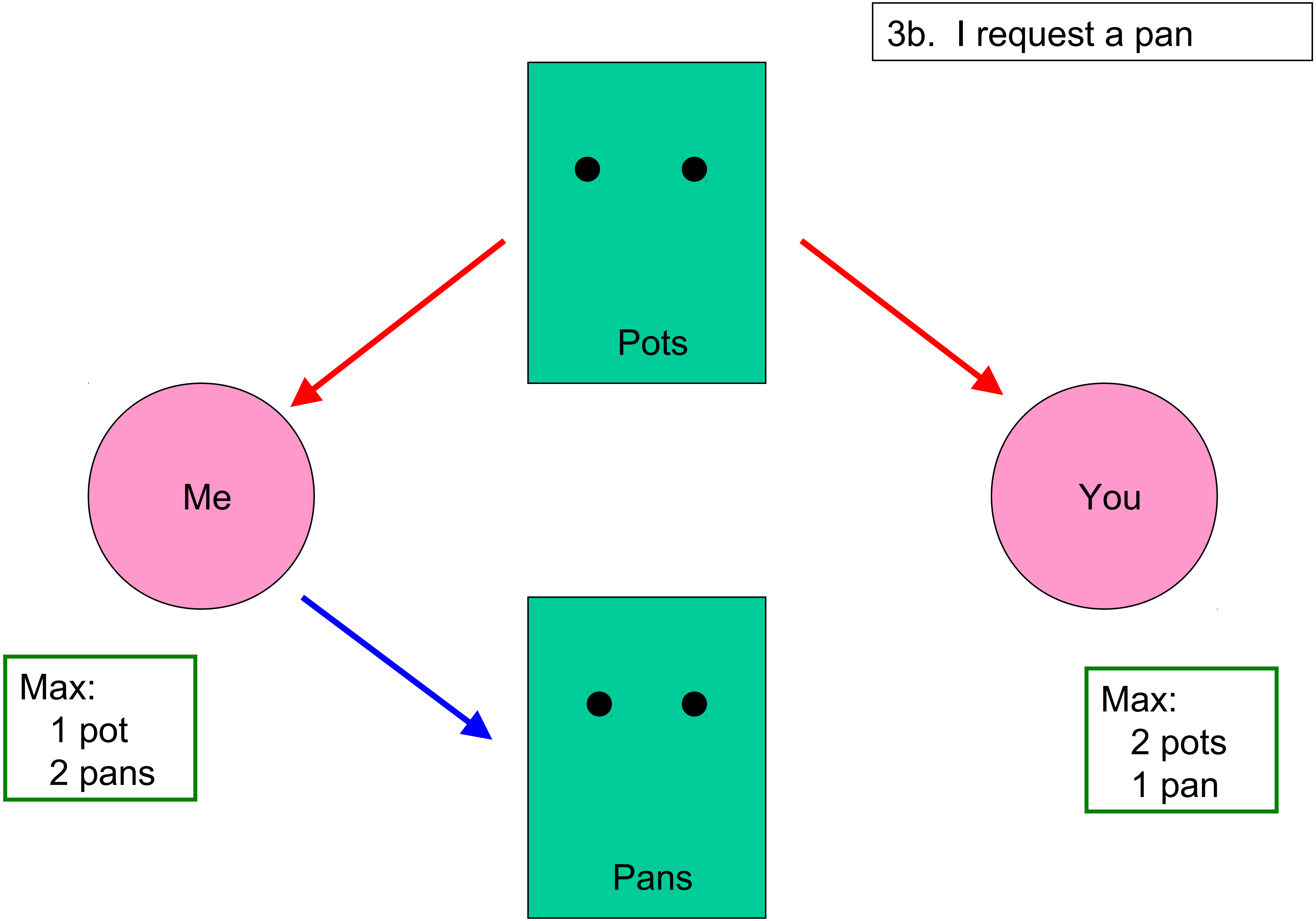


Allocation is OK; there is a way for me to complete, and then you can complete
Pots
Me
You
Pans
Max:
1 pot
2 pans
Max:
2 pots
1 pan

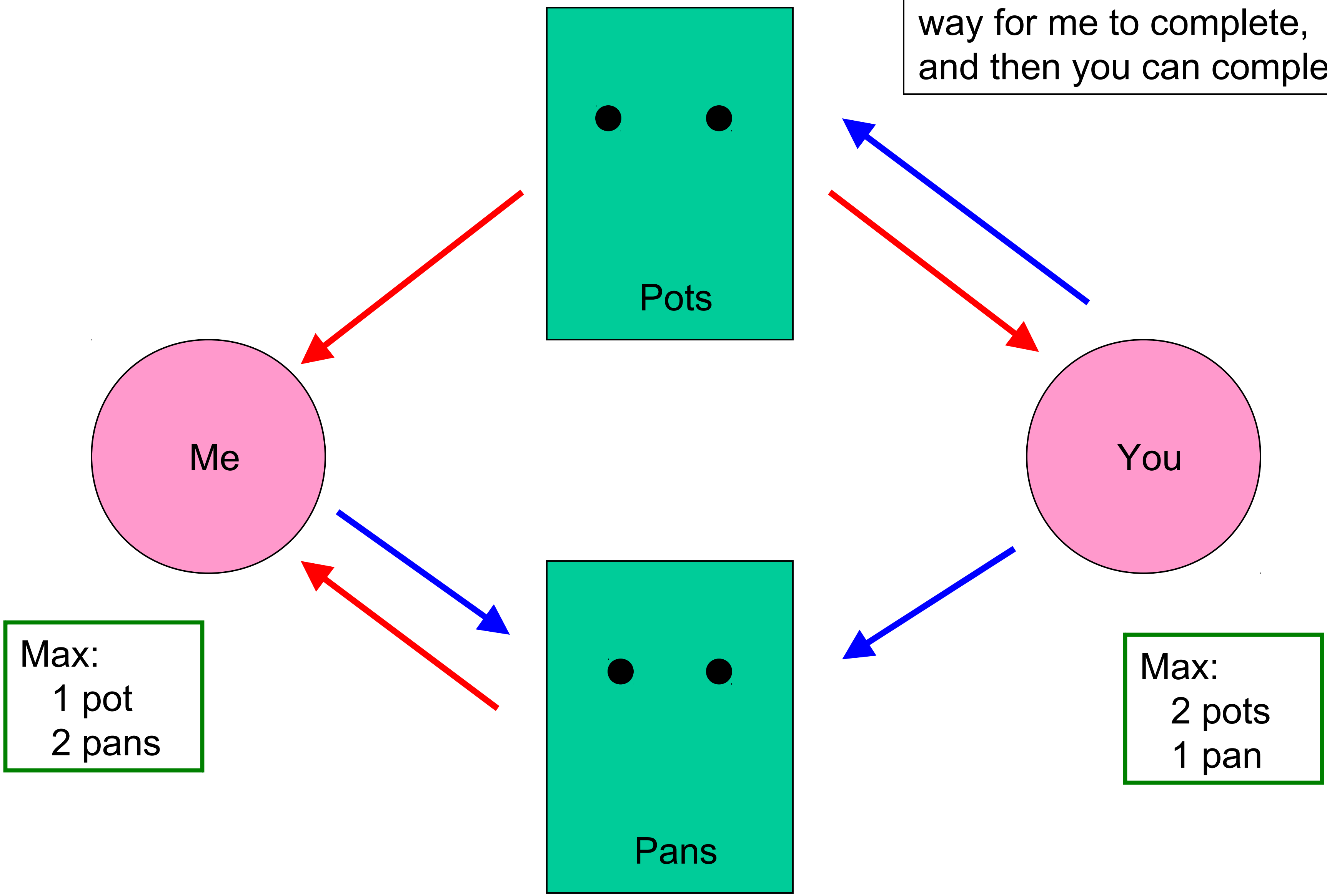
3a. You request a pan




NO! Both of us might be unable to complete!
Pots
Me
You
Pans
Max:
1 pot
2 pans
Max:
2 pots
1 pan


3b. I request a pan
Pots
Me
You
Pans
Max:
1 pot
2 pans
Max:
2 pots
1 pan

Allocation is OK; there is a way for me to complete, and then you can complete



Summary

- Deadlock is bad!
- We can deal with it either statically (prevention) or dynamically (avoidance and detection)
- In practice, ordering resources (locks) is the technique you'll encounter most often