CSE 451: Operating Systems
Winter 2009

Module 9
Deadlock

Mark Zbikowski

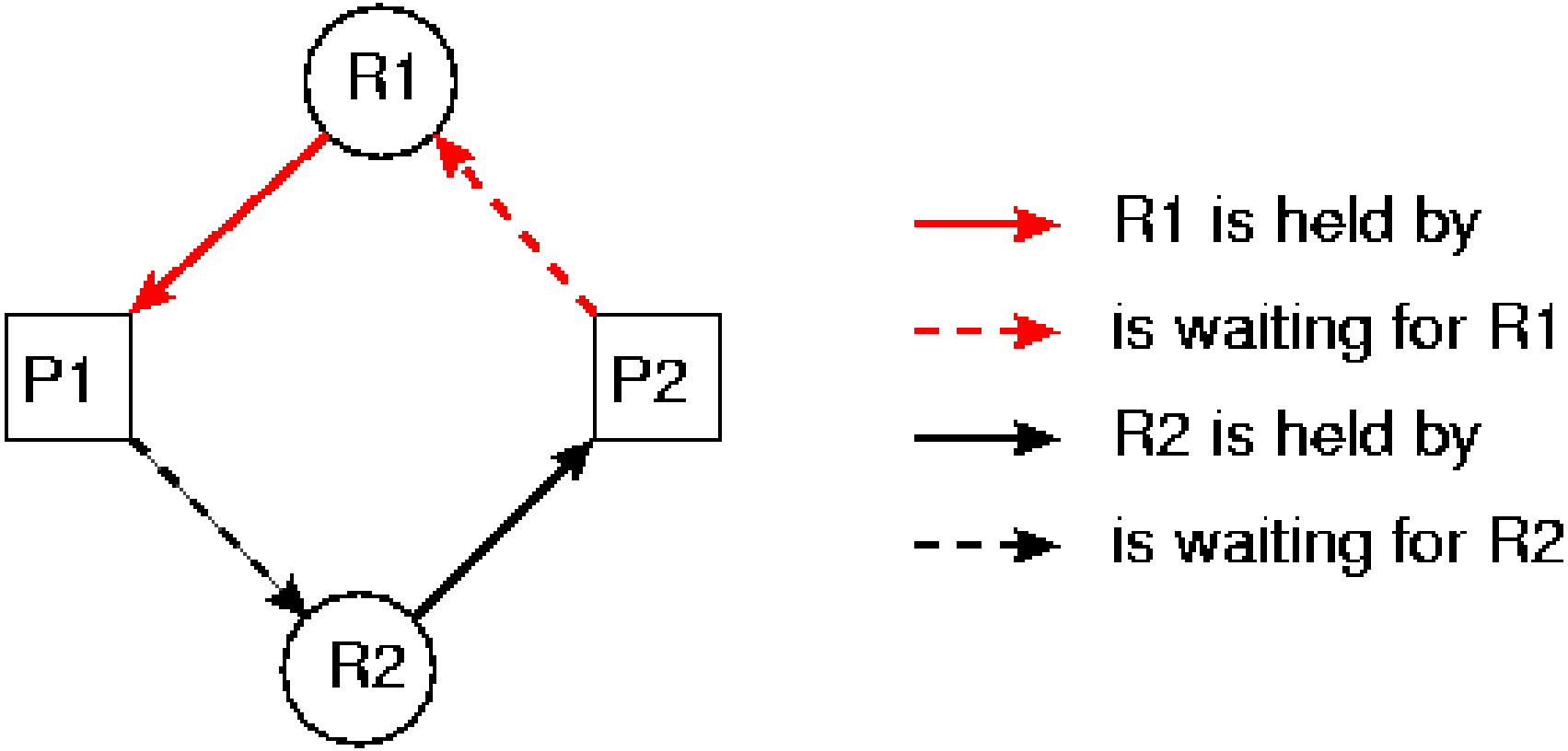
Definition

- 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

1. Mutual Exclusion
   - Resource owned by one thread
2. Hold and Wait
   - Thread may hold a resource while waiting
3. No Preemption
   - Ownership is never transferred
4. Circular Wait
   - Irreducible cycle in hold/wait graph

Resource graph

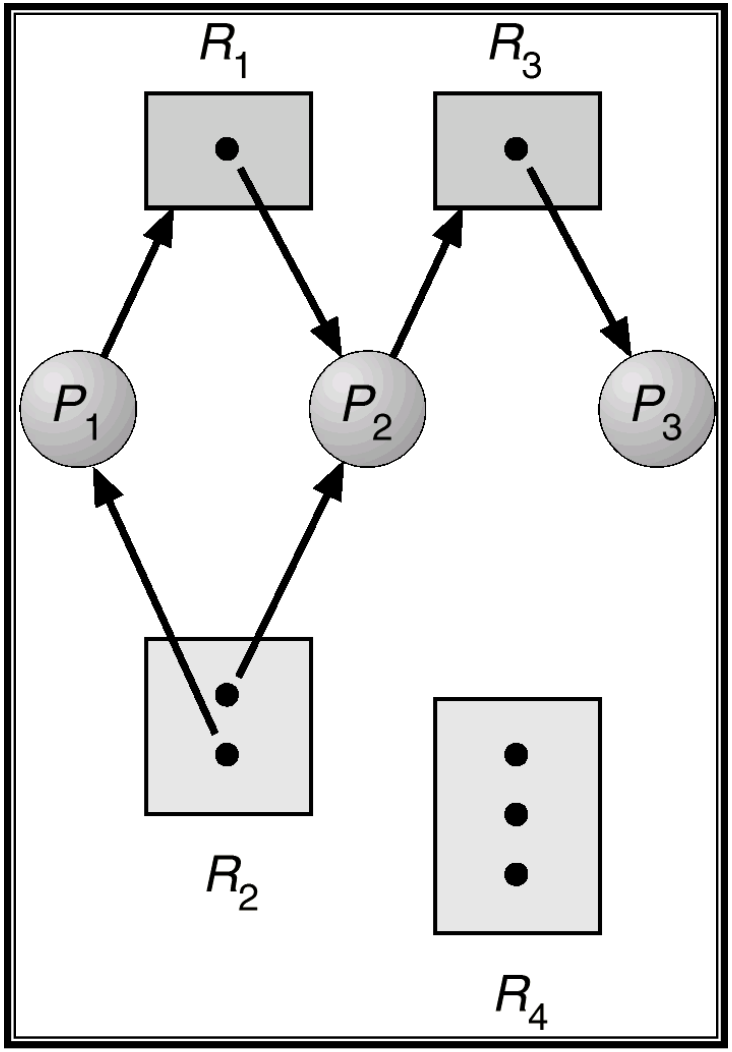


- A deadlock exists if there is an *irreducible cycle* in the resource graph (such as the one above)

Graph reduction

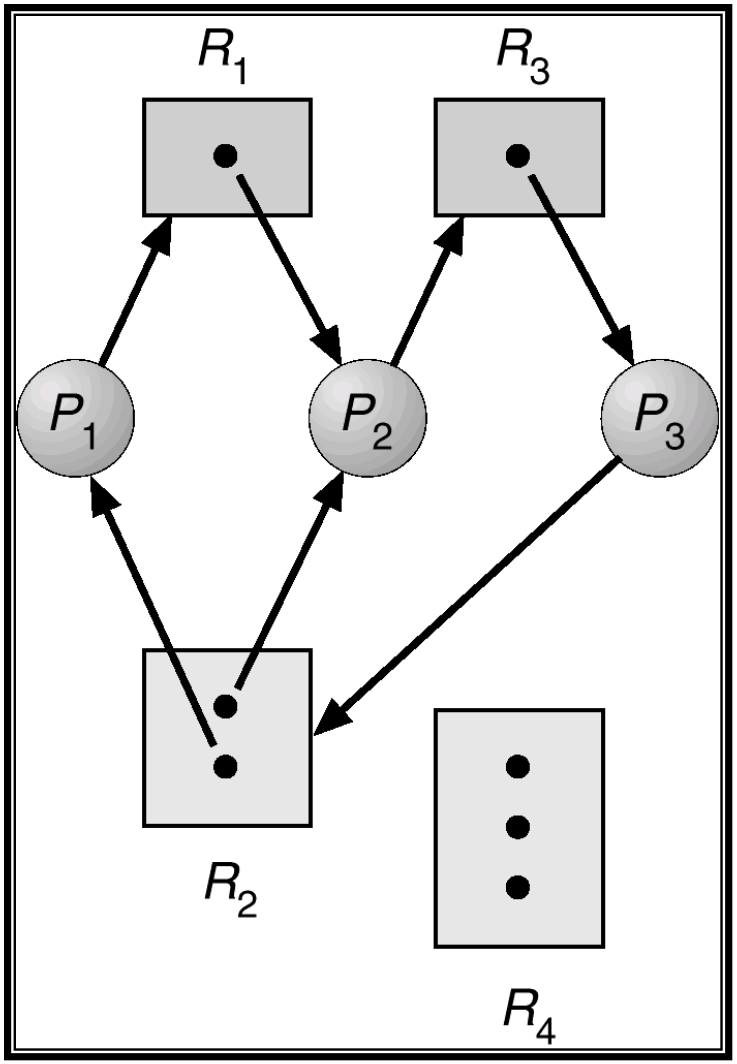
- A graph can be *reduced* by a thread if all of that thread's requests can be granted
  - in this case, the thread eventually will free all resources and not be waiting on any – all arcs (allocations) to it in the graph are deleted
- Miscellaneous theorems (Holt, Havender):
  - There are no deadlocked threads iff the graph is completely reducible
  - The order of reductions is irrelevant
- (Detail: resources with multiple units)

Resource allocation graph with no cycle

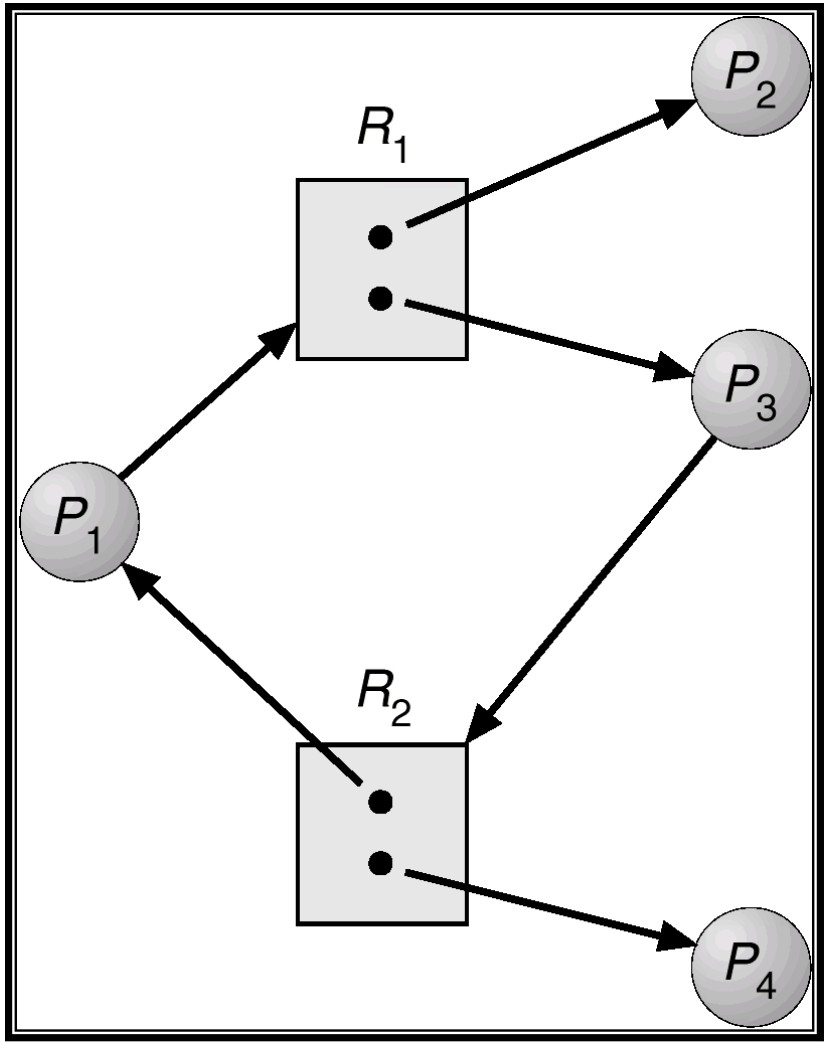


What would cause a deadlock?

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 (static), or
  - Avoidance (dynamic), or
  - detection (and recovery)

Prevention (static)

- Hold and Wait
  - each thread obtains all resources at the beginning; blocks until all are available
    - drawback?
- Circular Wait
  - resources are ordered; each thread obtains them in sequence (which means acquiring some before they are actually needed)
    - why does this work?
    - pros and cons?

Avoidance (dynamic)

- 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…

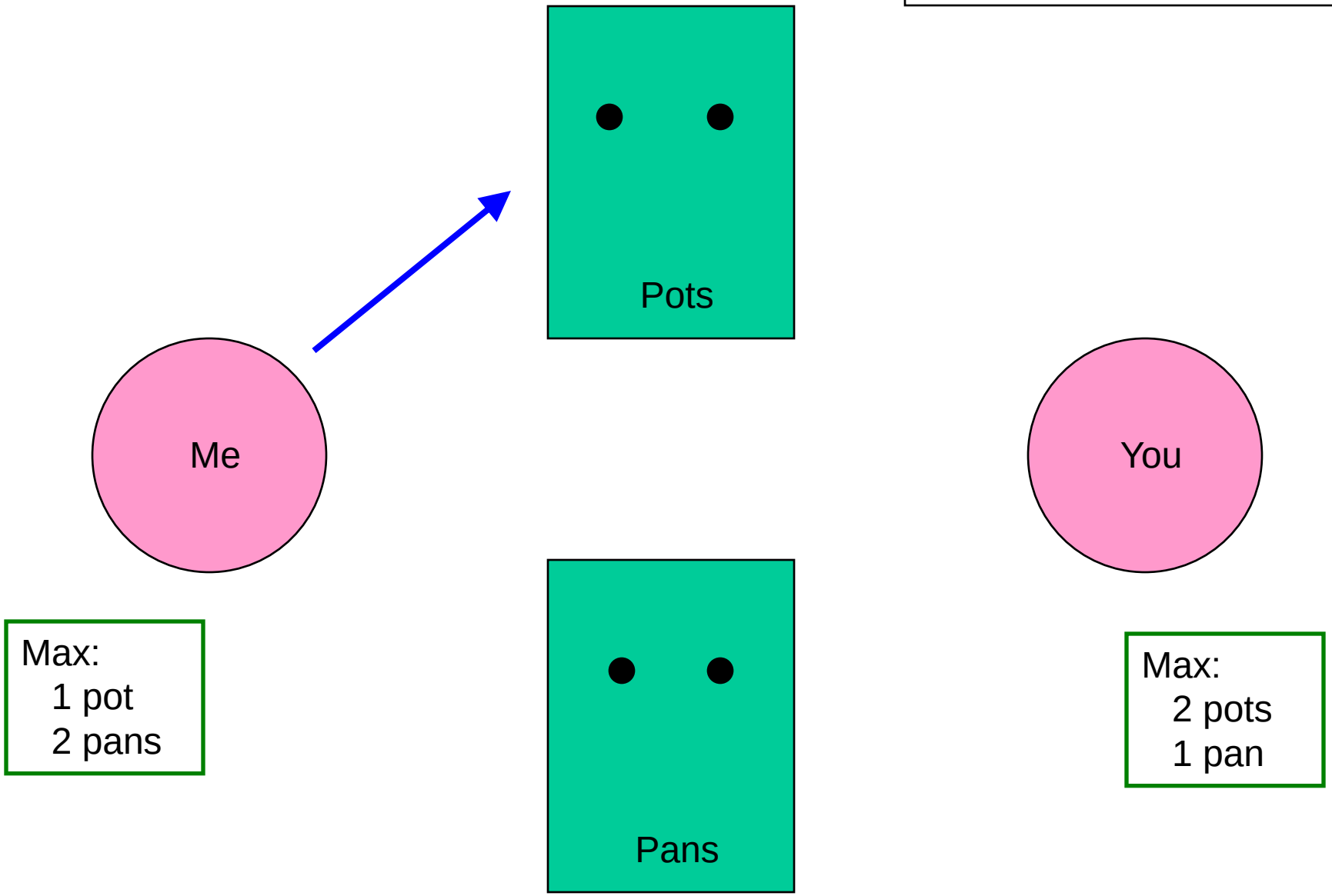
Detection and recovery

- every once in a while, check to see if there's a deadlock
  - how?
- if so, eliminate it
  - how?

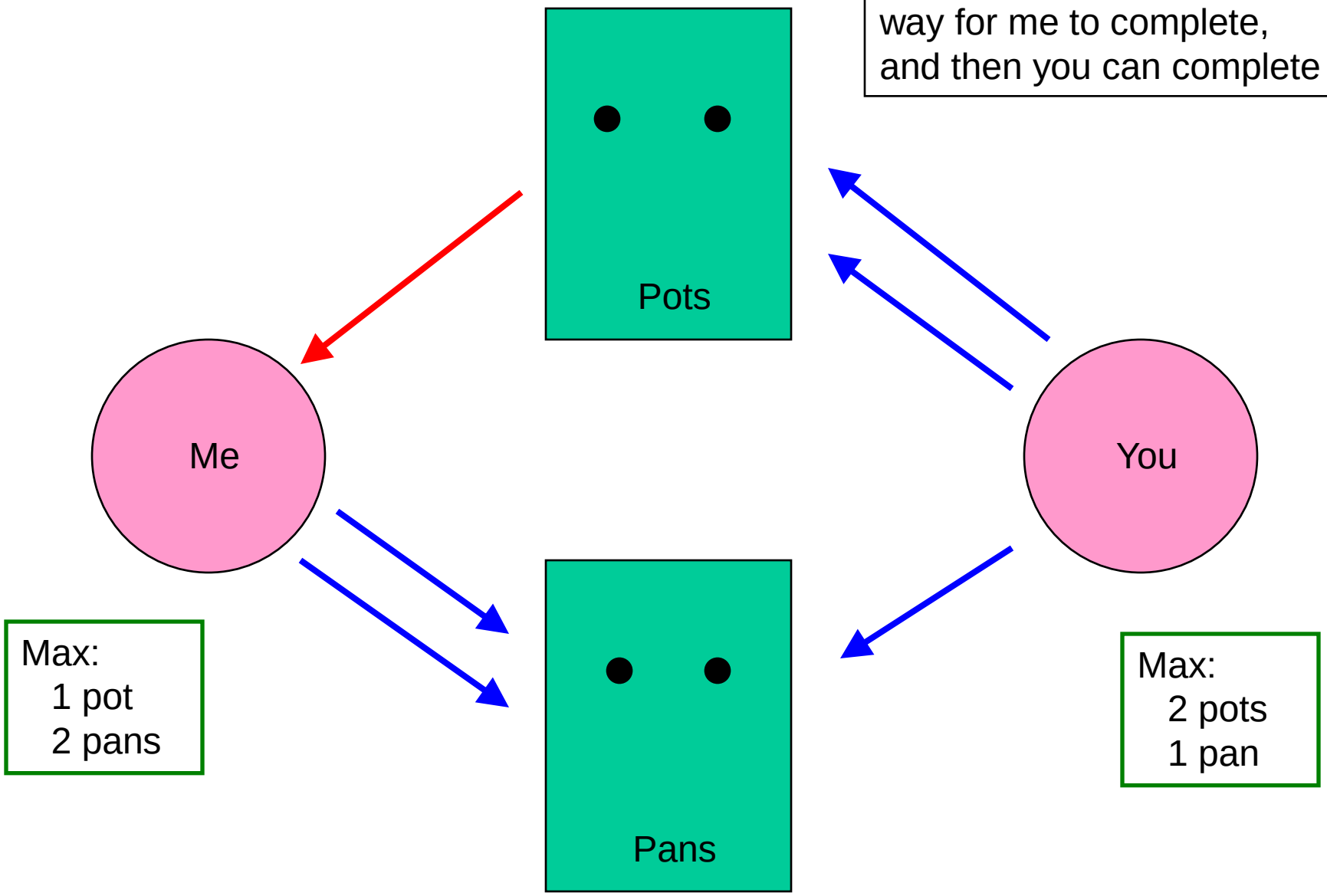
Avoidance: Banker's Algorithm example

- 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

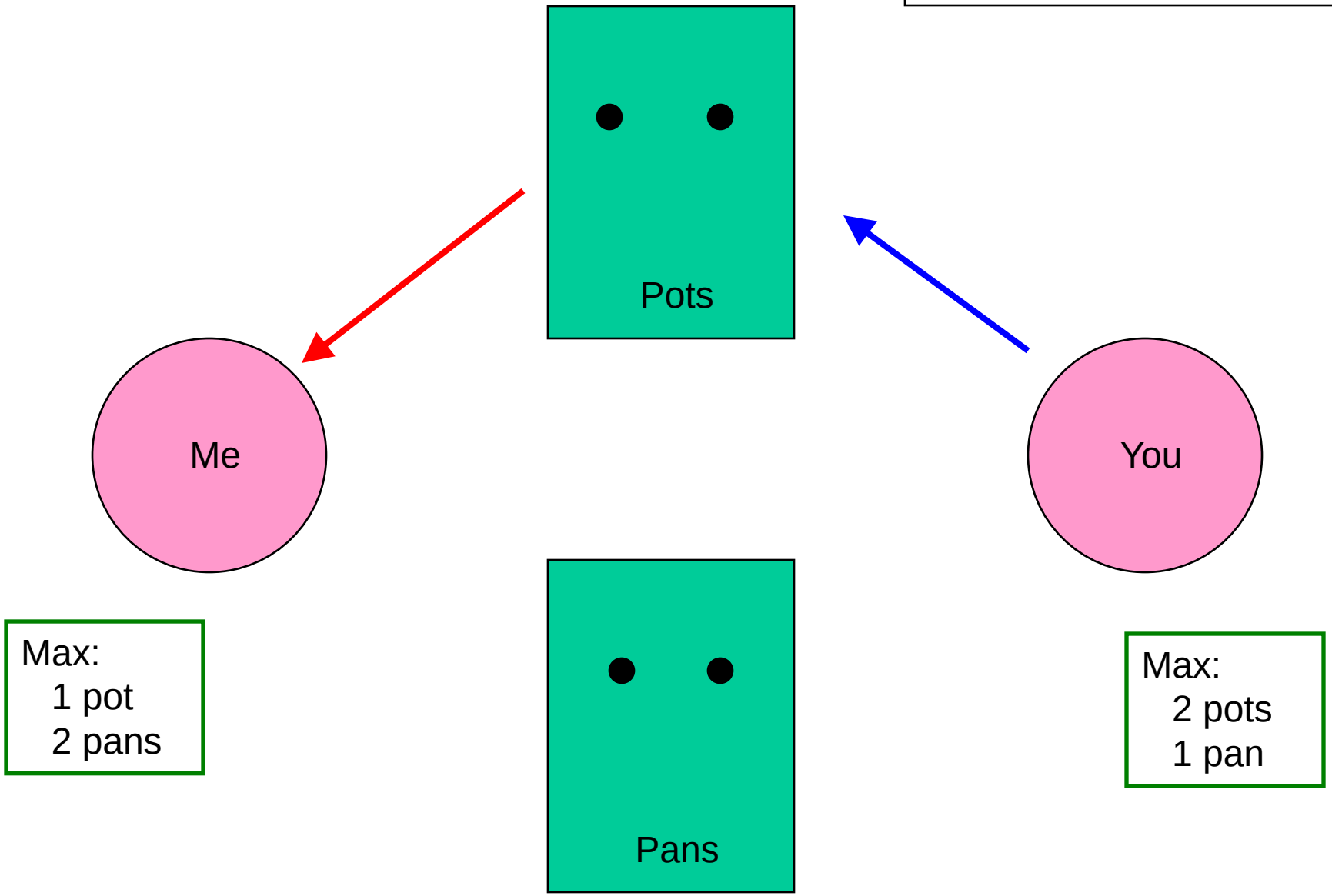
1. I request a pot

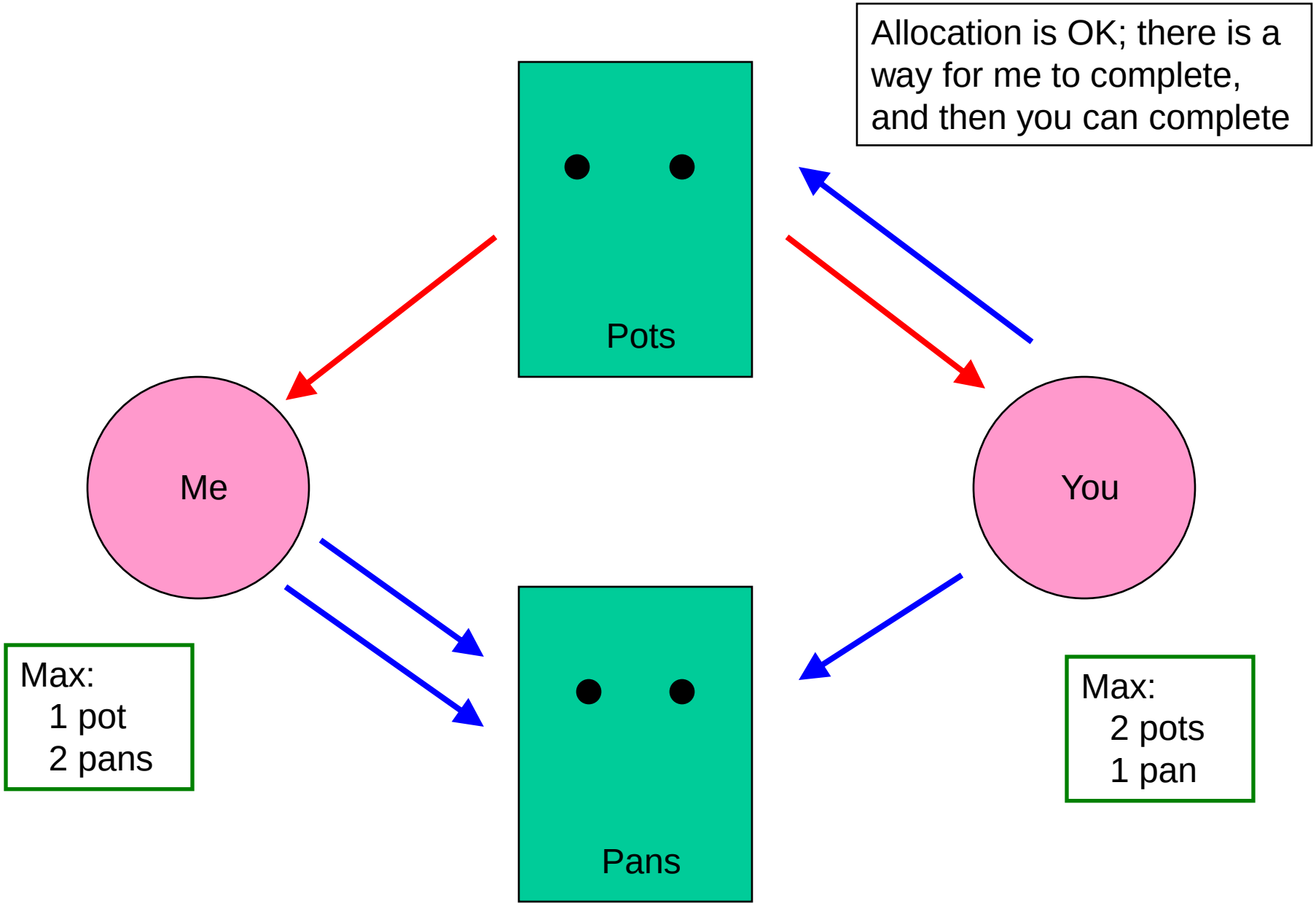


Allocation is OK; there is a way for me to complete, and then you can complete

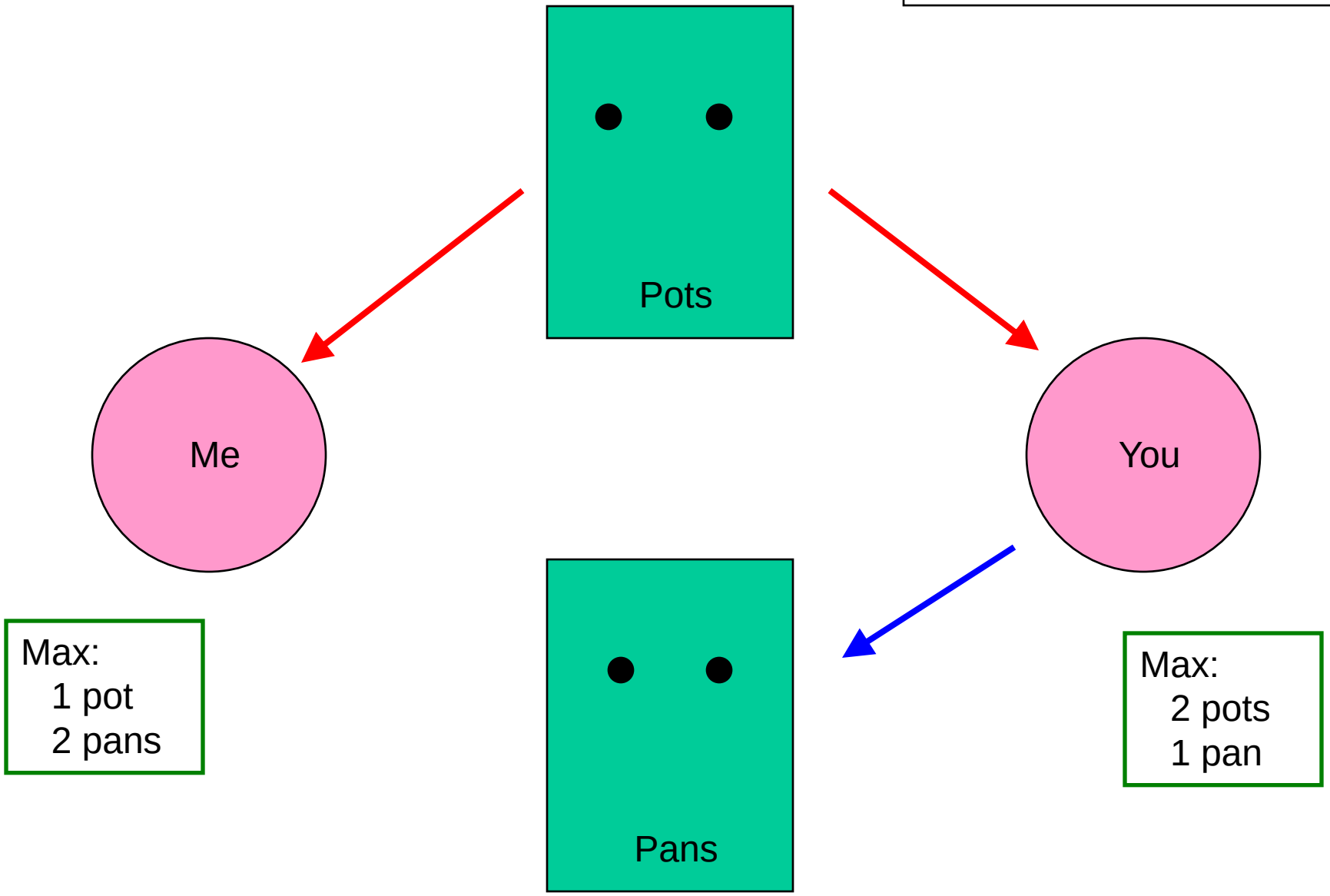


2. You request a pot

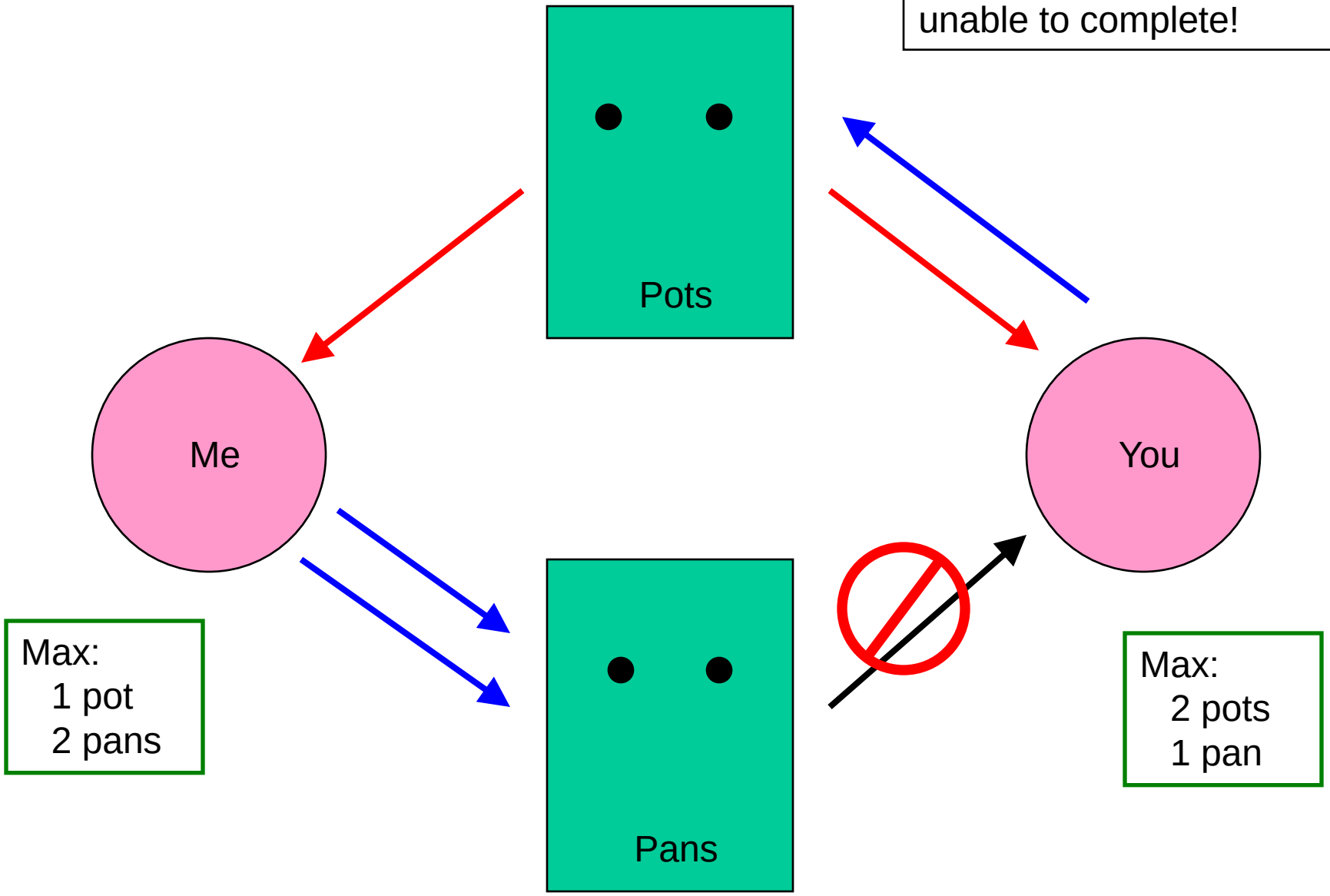



Allocation is OK; there is a way for me to complete, and then you can complete
Pots
Me
You
Pans
Max:
1 pot
2 pans
Max:
2 pots
1 pan

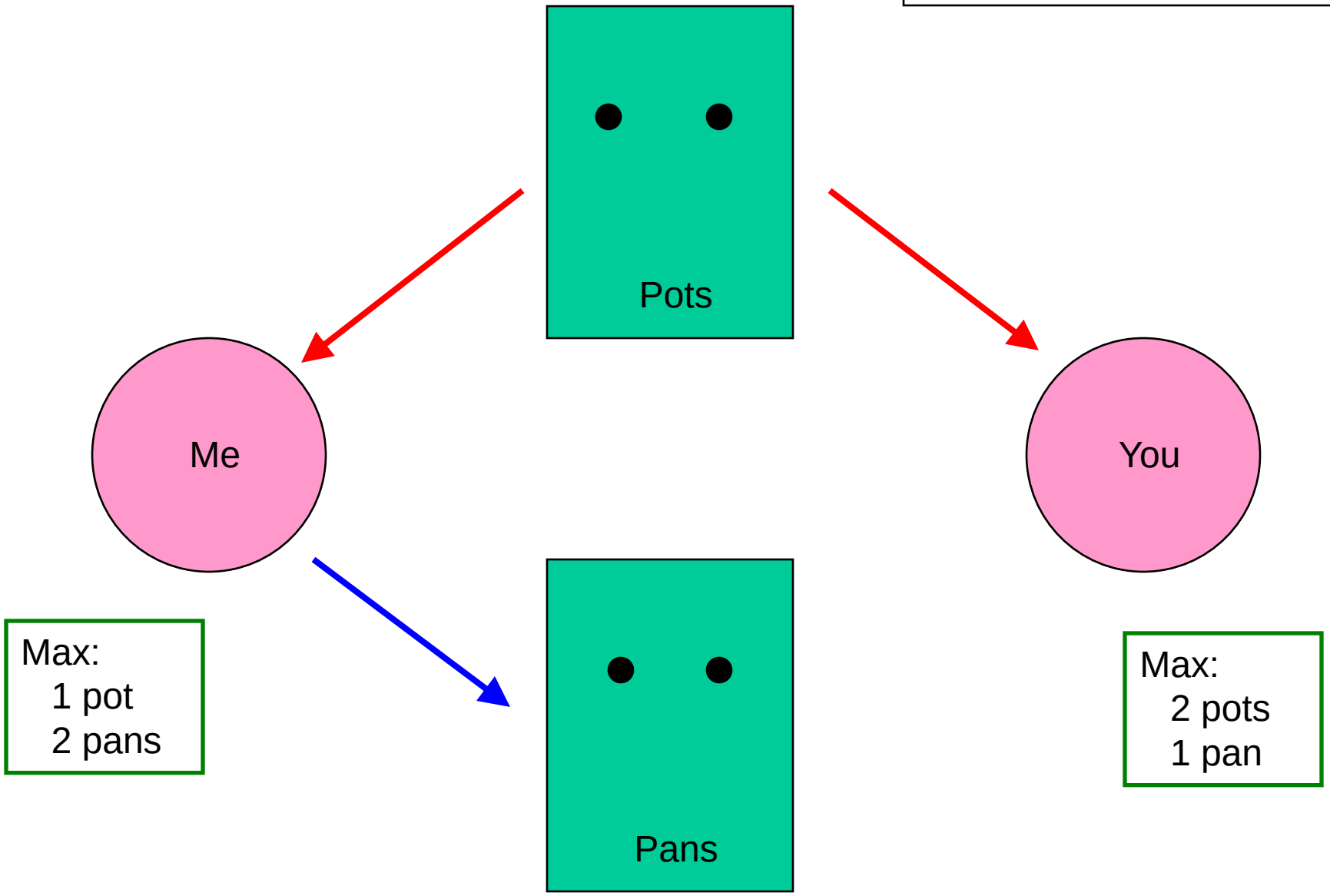
3a. You request a pan



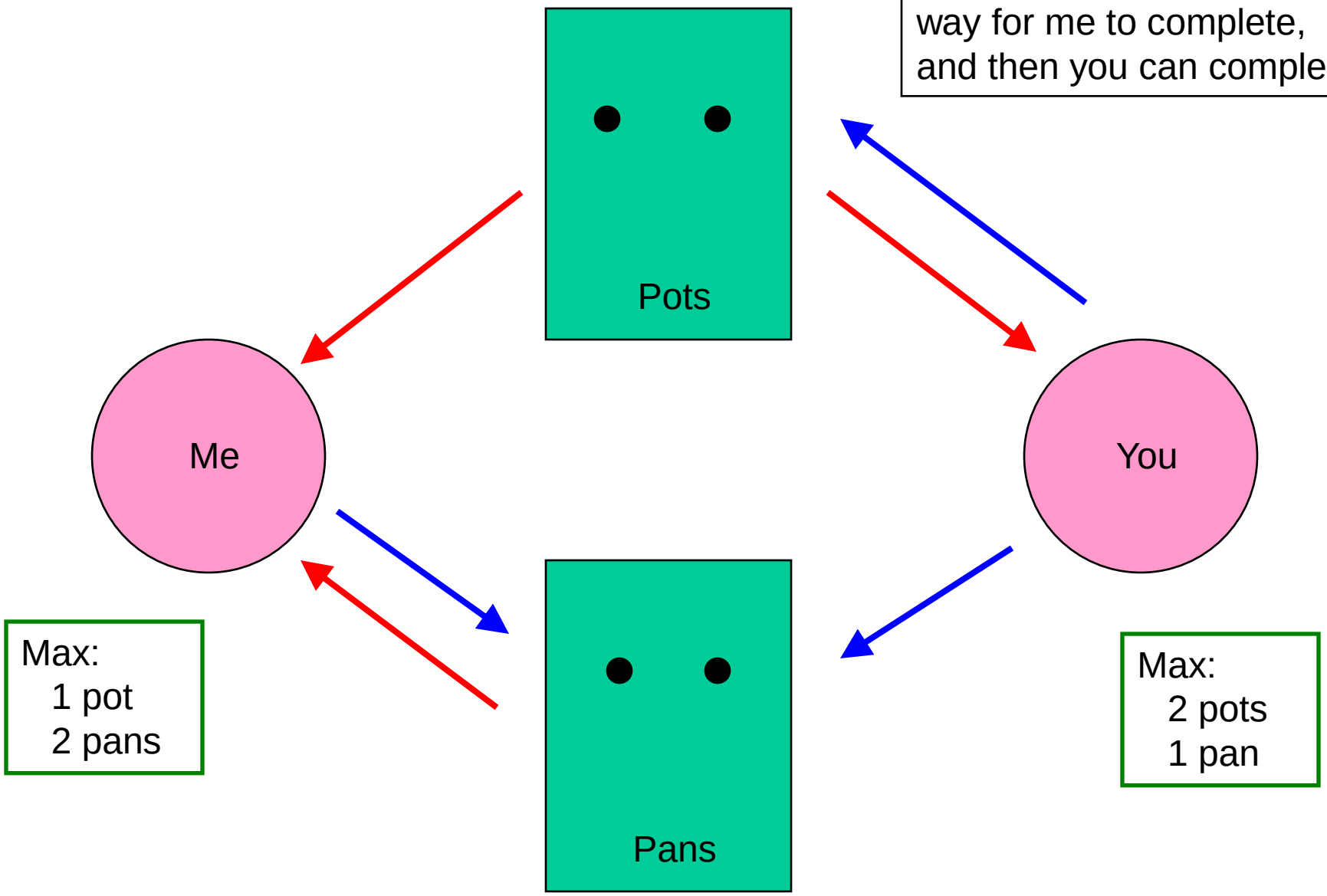
NO! Both of us might be unable to complete!



3b. I request a pan



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



Current practice

- Microsoft SQL Server
  - “The SQL Server Database Engine automatically detects deadlock cycles within SQL Server. The Database Engine chooses one of the sessions as a deadlock victim and the current transaction is terminated with an error to break the deadlock.”
- Oracle
  - As Microsoft SQL Server, plus “Multitable deadlocks can usually be avoided if transactions accessing the same tables lock those tables in the same order... For example, all application developers might follow the rule that when both a master and detail table are updated, the master table is locked first and then the detail table.”

- Windows internals (Linux no different)
  - "… the NT kernel architecture is a deadlock minefield. With the multi-threaded re-entrant kernel there is plenty of deadlock potential."
  - "Lock ordering is great in theory, and NT was originally designed with mutex levels, but they had to be abandoned. Inside the NT kernel there is a lot of interaction between memory management, the cache manager, and the file systems, and plenty of situations where memory management (maybe under the guise of its modified page writer) acquires its lock and then calls the cache manager. This happens while the file system calls the cache manager to fill the cache which in turn goes through the memory manager to fault in its page. And the list goes on."
  - Enhancements to semaphores for file systems:
    - Reacquire by owning thread
    - Test for ownership before acquiring

Summary

- Deadlock is bad!
- We can deal with it either statically (prevention) or dynamically (avoidance and detection)
- In practice, you'll encounter lock ordering, periodic deadlock detection/correction, and minefields