# Deadlock

## Detection and Breaking

Next: deadlock *detection* systems allow deadlocks to happen but rely on noticing and breaking them

# Deadlock

## Detection and Breaking

Next: deadlock *detection* systems allow deadlocks to happen but rely on noticing and breaking them

The hope is that detection and breaking will be cheaper than avoidance: this is not always clear

# Deadlock

## Detection and Breaking

Next: deadlock *detection* systems allow deadlocks to happen but rely on noticing and breaking them

The hope is that detection and breaking will be cheaper than avoidance: this is not always clear

The earliest detection system was *Detection by Operator*

# Deadlock

## Detection and Breaking

Next: deadlock *detection* systems allow deadlocks to happen but rely on noticing and breaking them

The hope is that detection and breaking will be cheaper than avoidance: this is not always clear

The earliest detection system was *Detection by Operator*

“The machine seems to have stopped…”

# Deadlock

## Detection and Breaking

The chief method employed is to spot when the circular wait happens

# Deadlock

## Detection and Breaking

The chief method employed is to spot when the circular wait happens

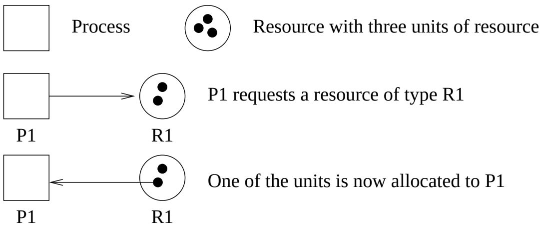
One method for deadlock detection uses *resource request and allocation graphs* (RRAG)

# Deadlock

## Detection and Breaking

The chief method employed is to spot when the circular wait happens

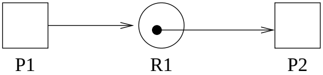
One method for deadlock detection uses *resource request and allocation graphs* (RRAG)




RRAGs

# Deadlock

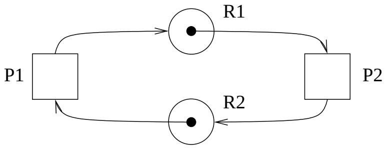
## Detection and Breaking




P1 requests from R1, but it has no free units, so P1 will be blocked

# Deadlock

## Detection and Breaking




Circular Wait

P1 requests from R1, but it has been allocated to P2;
P2 requests from R2, but it has been allocated to R1:
deadlock

# Deadlock

## Detection and Breaking

So deadlock detection is just finding these kinds of loops in RRAGs

# Deadlock

## Detection and Breaking

So deadlock detection is just finding these kinds of loops in RRAGs

This can be done by *graph reduction*

# Deadlock

## Detection and Breaking

So deadlock detection is just finding these kinds of loops in RRAGs

This can be done by *graph reduction*

For each process repeatedly

1. Remove all request links from the process to resources that are available (perhaps available after a reduction step)
2. When there are no requests links left, remove all links from allocated units of resource to the process

# Deadlock

## Detection and Breaking

If we can reduce a RRAG by all processes, then there is no deadlock

# Deadlock

## Detection and Breaking

If we can reduce a RRAG by all processes, then there is no deadlock

1. Removing request links to available resources is allocating the requested resources to the process

# Deadlock

## Detection and Breaking

If we can reduce a RRAG by all processes, then there is no deadlock

1. Removing request links to available resources is allocating the requested resources to the process
2. Removing allocated links is the process finishing and returning its resources

# Deadlock

## Detection and Breaking

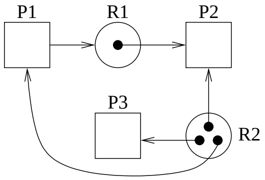
If we can reduce a RRAG by all processes, then there is no deadlock

1. Removing request links to available resources is allocating the requested resources to the process
2. Removing allocated links is the process finishing and returning its resources

An example:

# Deadlock

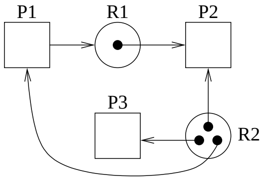
## Detection and Breaking




A RAAG

# Deadlock

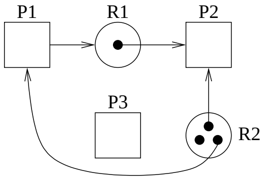
## Detection and Breaking




We can satisfy P3's requests (none)

# Deadlock

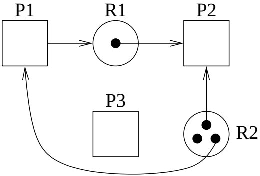
## Detection and Breaking




Reduce P3's allocations

# Deadlock

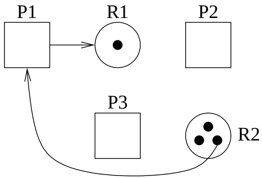
## Detection and Breaking




We can satisfy P2's requests

# Deadlock

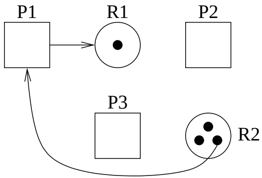
## Detection and Breaking




Reduce P2

# Deadlock

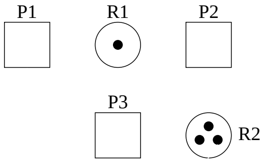
## Detection and Breaking




Now P1's requests can be granted

# Deadlock

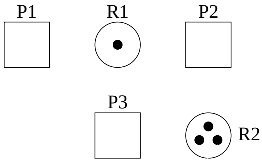
## Detection and Breaking




Reduce P1

# Deadlock

## Detection and Breaking

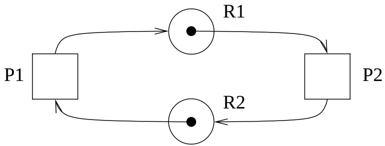



Reduce P1

No more links, so this graph has been completely reduced and there will be no deadlock

# Deadlock

## Detection and Breaking



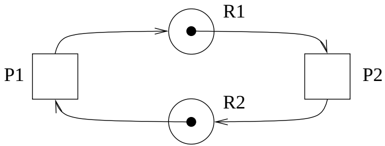

Circular Wait

We can't reduce this as no request links are removable

□

# Deadlock

## Detection and Breaking




Circular Wait

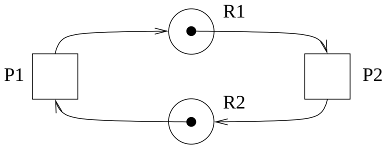
We can't reduce this as no request links are removable

Thus this is deadlock

□

# Deadlock

## Detection and Breaking




Circular Wait

We can't reduce this as no request links are removable

Thus this is deadlock

An advantage of this technique is that it isolates the parts that are deadlocking: we can see them in the graph