Operating Systems

Deadlocks
Based on Ch. 7 of
OS Concepts by SGG

System Model

- Deadlocks can happen for many reasons
  - Sloppy programming
  - Hardware failure
  - Inexpert human intervention
  - Unusual sequence of allocations and deallocations
- We are interested only in the last mechanism.
- Allocation-deallocation has three phases
  - Request (and may be block)
  - Use
  - Release

Conditions for Deadlock

- The Master of the Obvious proclaimed that a deadlock can occur iff
  - Mutual exclusion
  - Hold and wait
  - No Preemption
  - Circular wait
- The “circular wait” means that there is a cycle in the resource allocation graph

Resource Allocation Graph

- It is a graph with two kinds of nodes
  - Processes
  - Resources
- There are two kinds of edges
  - Request edge (from a process to a resource)
  - Assignment edge (from a resource to a process)
- Can be a bit more complex if the resources have multiple identical copies.
  - We do not explore the more complex variety.

General Methods for Handling Deadlocks

- We can prevent deadlocks from occuring
- We can allow deadlocks to happen and recover afterwards
- Ignore the issue (it is the responsibility of the application)

Deadlock Prevention

- To prevent deadlocks we can make sure that at least one of the conditions proclaimed by the Master of the Obvious does not hold
- This may put unnecessary burden to the system or reduce parallelizability (is this a word?)

Deadlock Prevention

- Deny mutual exclusion
  - Plain silly.
- Deny Hold-and-Wait
  - Require a process to request all its resources in one system call and only when it has none.
  - Processes would have to request (and lock) many resources, just in case. Also starvation is possible.
- Allow Preemption
  - Plain old stealing. To make it work we have to be able to roll back changes.

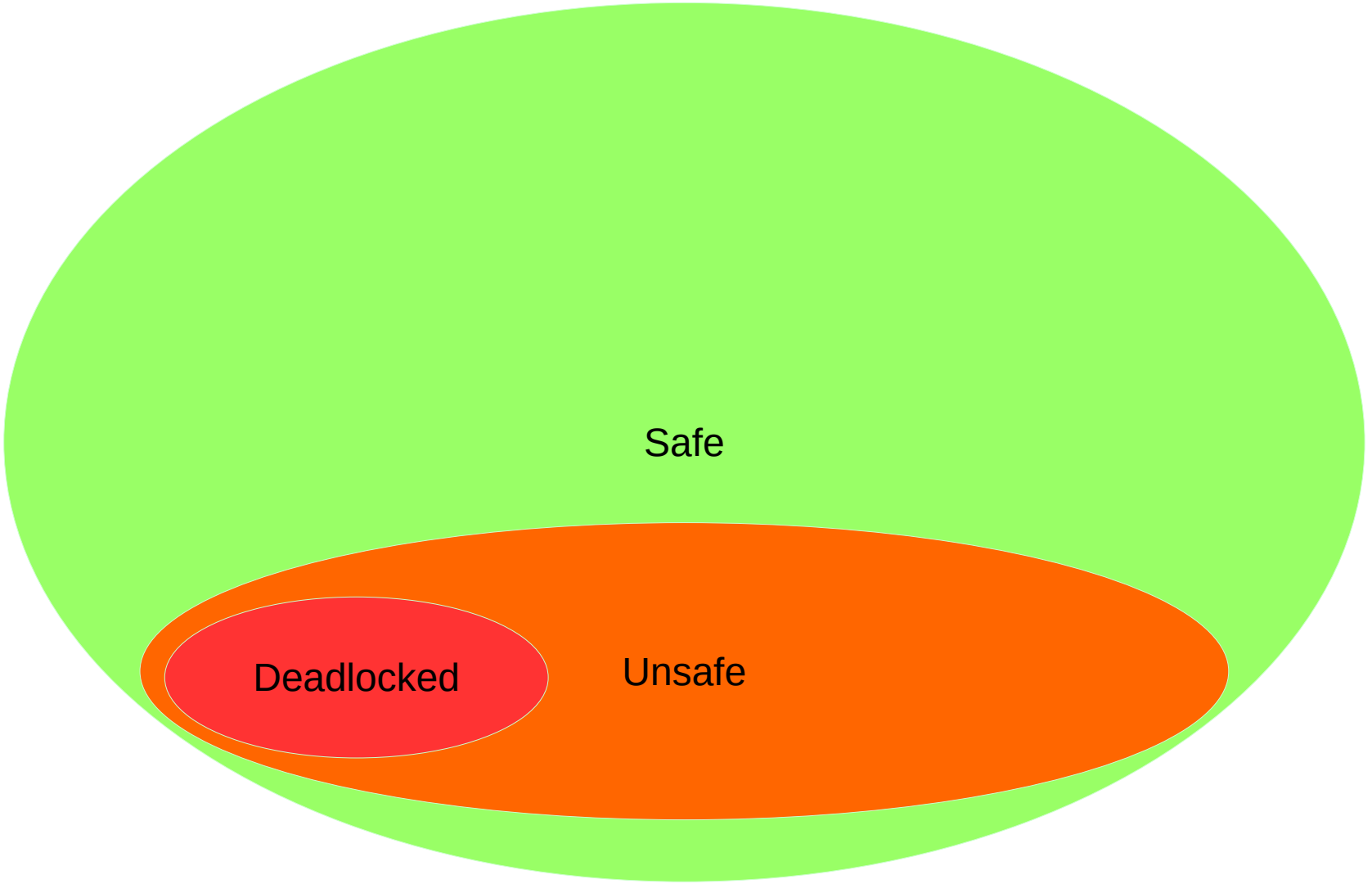
Deny Circular Waiting

- The trick is to arrange for a total order
  - For every pair of resources we know which one is before the other.
- Then impose the condition that resources have to be allocated in this order

Deadlock avoidance

- We may increase the flexibility of the system if we know what resources a process may use in the future
- A process “claims” a resource when it announces that it might request it in the future
- If the worst case senario would allow the process to complete we let the process start requesting resources. O/w we block it.

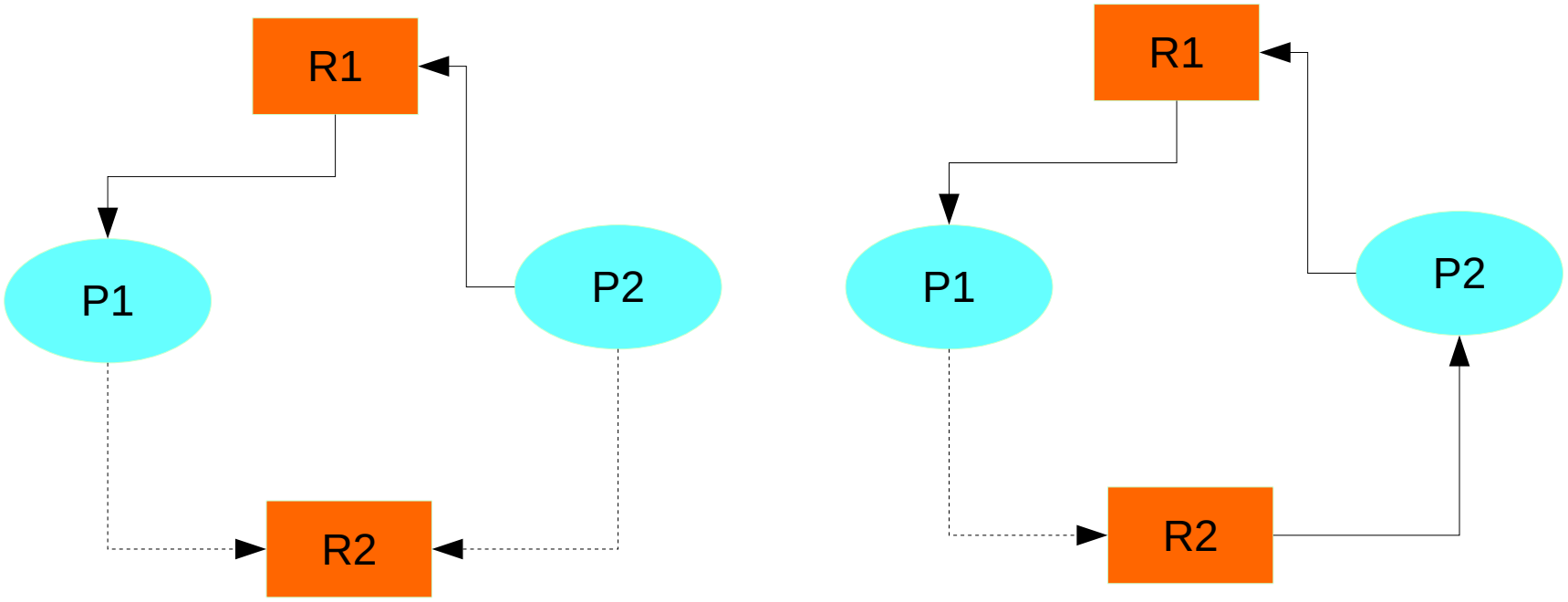
Safe State



Resource Allocation Graph

- We add a new type of edge called “claim”. If a process may allocate a resource sometime in the future we have an edge from the process to the resource.
- If a process P requests a resource R then the claim edge between them becomes request edge.
- If the resource R is allocated then the request edge becomes allocation edge (and flips)

Deadlock Avoidance using Graphs



Deadlock Detection

- We can run a cycle detection algorithm
- We can speed up the process by eliminating the resource nodes.
  - If process A waits for a resource allocated to process B, the new graph contains an edge from A to B

How often

- How often we invoke the deadlock detection algorithm
- Not very often, because it is expensive
- When system is mostly idle, makes sense
  - When a pool of processes are deadlocked, the system will be idle.

Recovery from Deadlock

- Kill all deadlocked processes
- Kill one at a time and release its resources until the system moves again
  - We need to select the order to minimize cost
  - Resources have to be in safe state
- Preempt one resource at a time and pass it on to a process that requested it