Deadlocks

Deadlocks: overview

- Resources
- Why do deadlocks occur?
- Dealing with deadlocks
  - Ignoring them: ostrich algorithm
  - Detecting & recovering from deadlock
  - Avoiding deadlock
  - Preventing deadlock

Resources

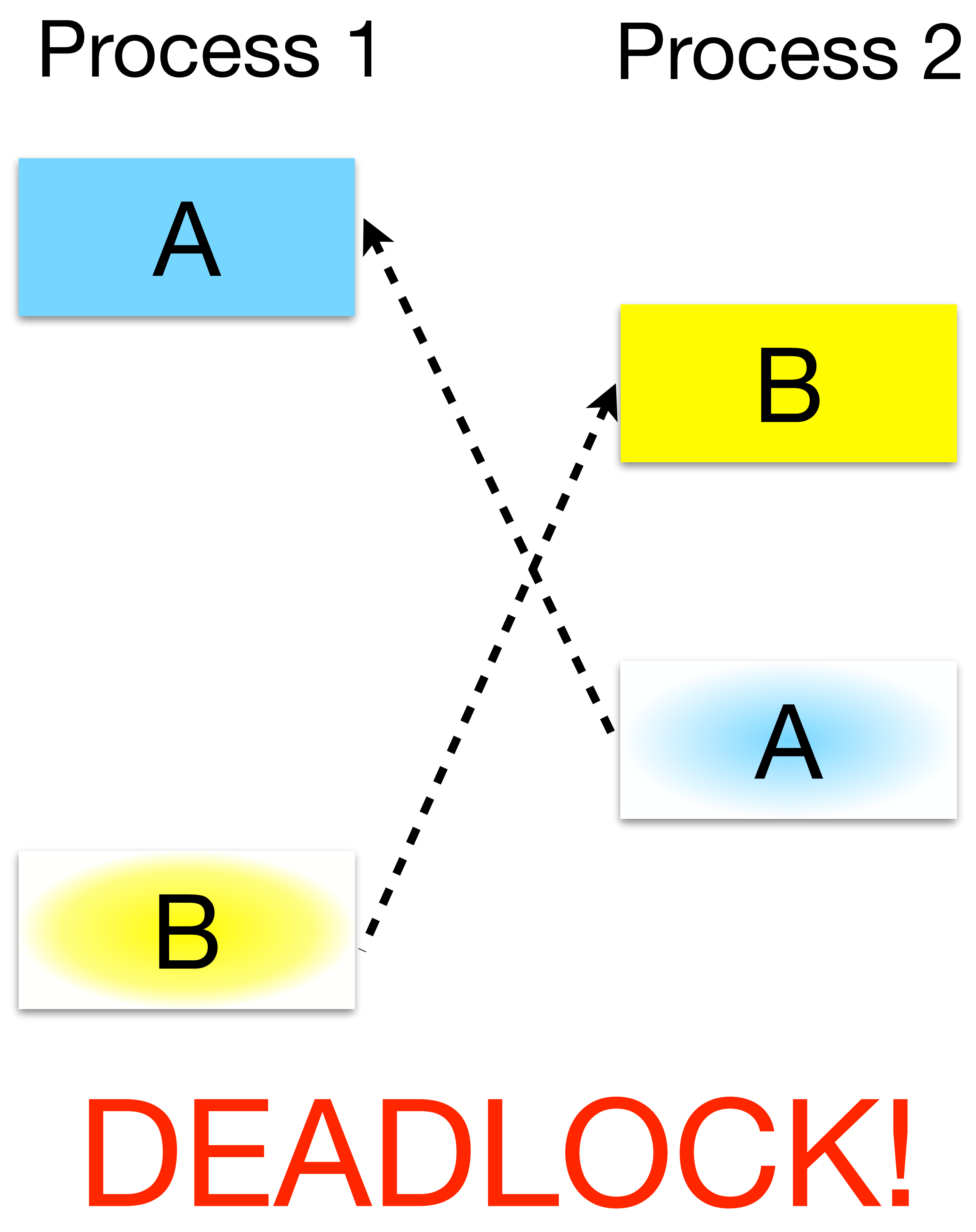
- Resource: something a process or thread uses
  - Usually limited (at least somewhat)
- Examples of computer resources
  - Printers
  - Semaphores / locks
  - Memory
  - Database tables
- Processes need access to resources in a reasonable order
- Two types of resources:
  - Preemptable resources: can be taken away from a process with no ill effects
  - Nonpreemptable resources: will cause the process to fail if taken away

Using resources

- Sequence of events required to use a resource
  - Request the resource
  - Use the resource
  - Release the resource
- Can’t use the resource if request is denied
  - Requesting process has options
    - Block and wait for resource
    - Continue (if possible) without it: may be able to use an alternate resource
    - Process fails with error code
  - Some of these may be able to prevent deadlock…

When do deadlocks happen?

- Suppose
  - Process 1 holds resource A and requests resource B
  - Process 2 holds B and requests A
  - Both can be blocked, with neither able to proceed
- Deadlocks occur when …
  - Processes are granted exclusive access to devices or software constructs (resources)
  - Each deadlocked process needs a resource held by another deadlocked process



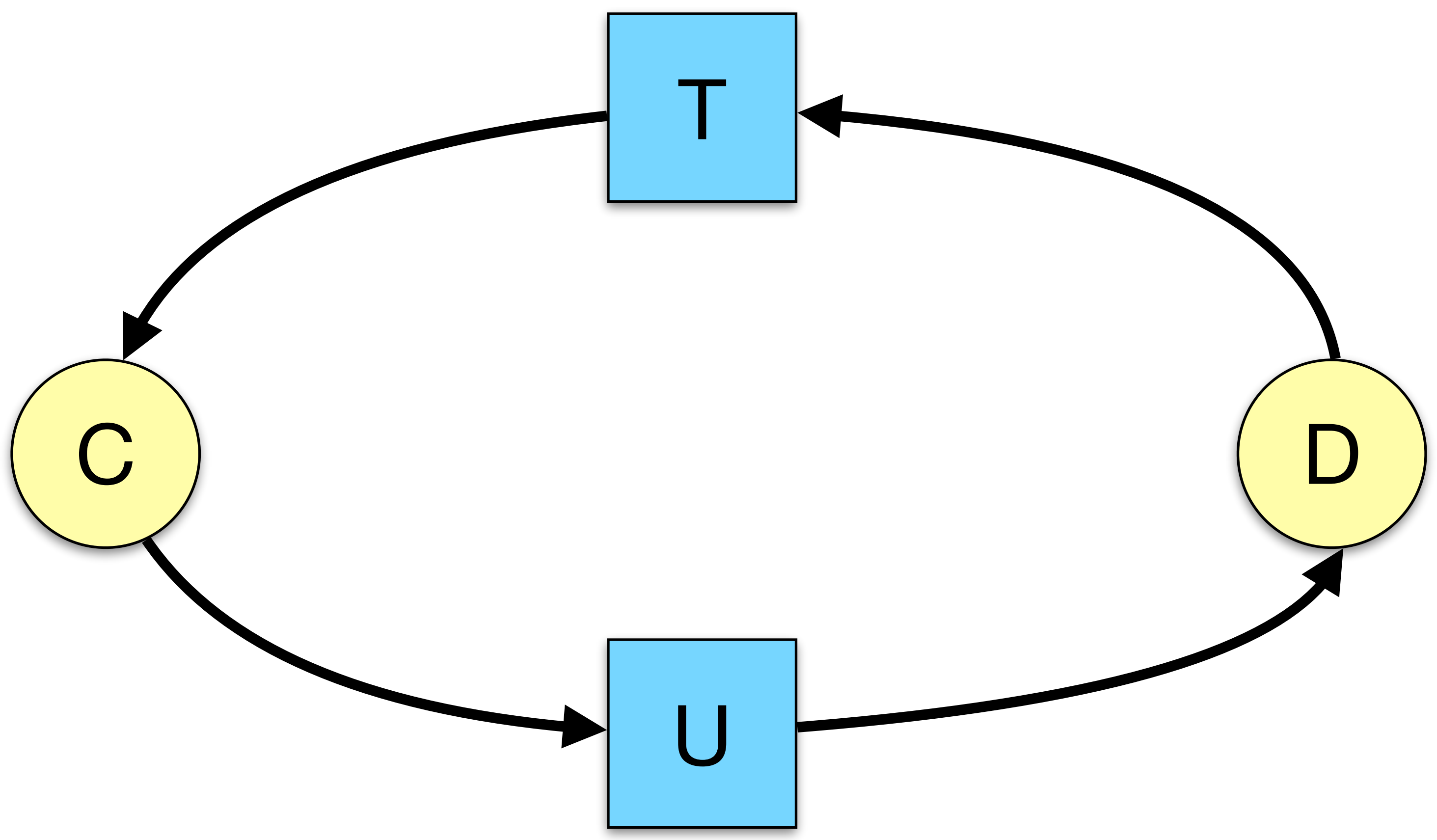
What is a deadlock?

- Formal definition:
  "A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause."
- Usually, the event is release of a currently-held resource
- In deadlock, none of the processes can
  - Run
  - Release resources
  - Be awakened

Four conditions for deadlock

- Mutual exclusion
  - Each resource is assigned to at most one process
- Hold and wait
  - A process holding resources can request more resources
- No preemption
  - Previously granted resources cannot be forcibly taken away
- Circular wait
  - There must be a circular chain of 2 or more processes where each is waiting for a resource held by the next member of the chain

Resource allocation graphs



- Resource allocation modeled by directed graphs
- Example 1:
  - Resource R assigned to process A
- Example 2:
  - Process B is requesting / waiting for resource S
- Example 3:
  - Process C holds T, waiting for U
  - Process D holds U, waiting for T
  - C and D are in deadlock!

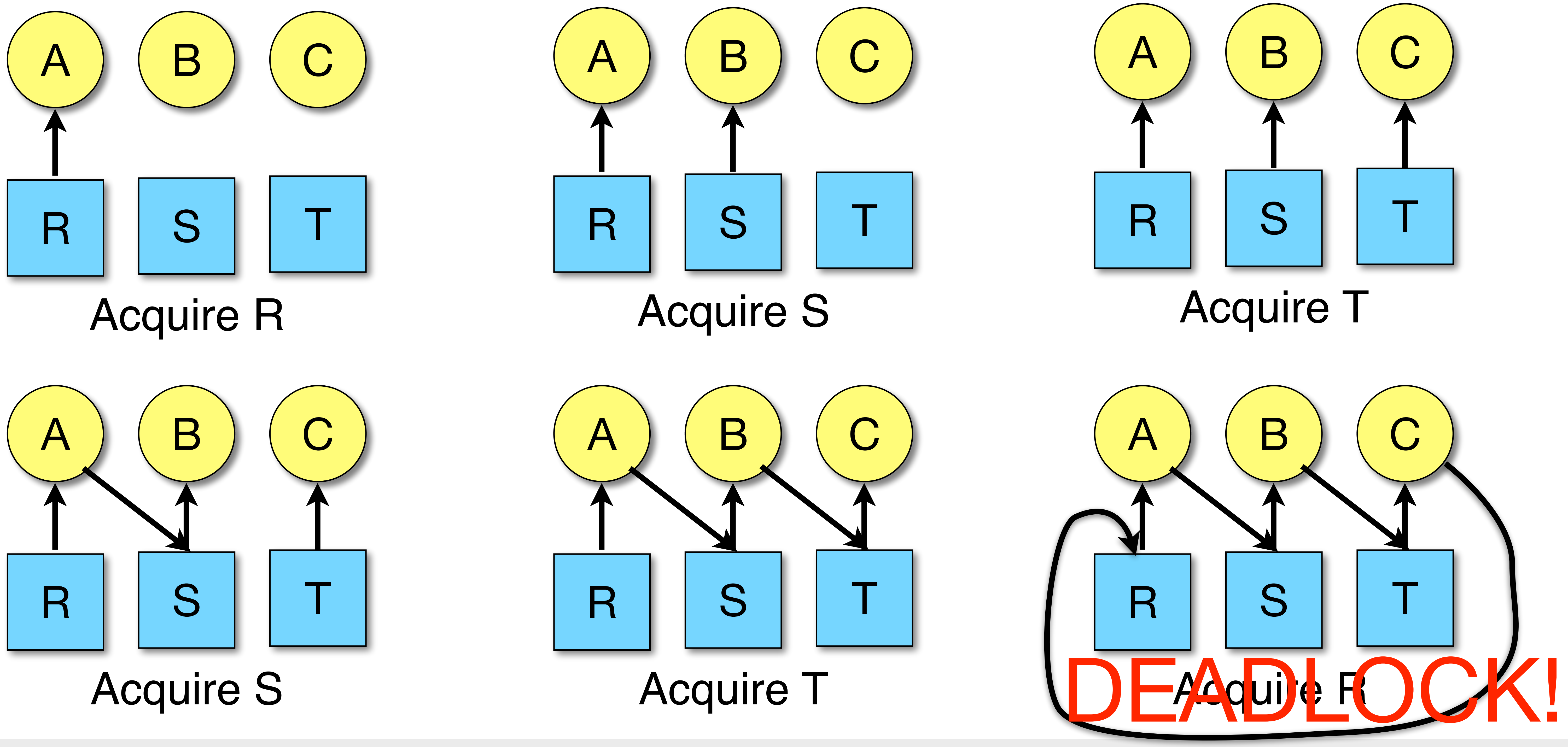
Dealing with deadlock

- How can the OS deal with deadlock?
  - Ignore the problem altogether!
    - Hopefully, it'll never happen…
  - Detect deadlock & recover from it
  - Dynamically avoid deadlock
    - Careful resource allocation
  - Prevent deadlock
    - Remove at least one of the four necessary conditions

- We'll explore these tradeoffs

Getting into deadlock

A	B	C
Acquire R	Acquire S	Acquire T
Acquire S	Acquire T	Acquire R
Release R	Release S	Release T
Release S	Release T	Release R



The Ostrich Algorithm

- Pretend there's no problem
- Reasonable if
  - Deadlocks occur very rarely
  - Cost of prevention is high
- UNIX™ and Windows™ take this approach
  - Resources (memory, CPU, disk space) are plentiful
  - Deadlocks over such resources rarely occur
  - Deadlocks typically handled by rebooting
- Trade off between convenience and correctness

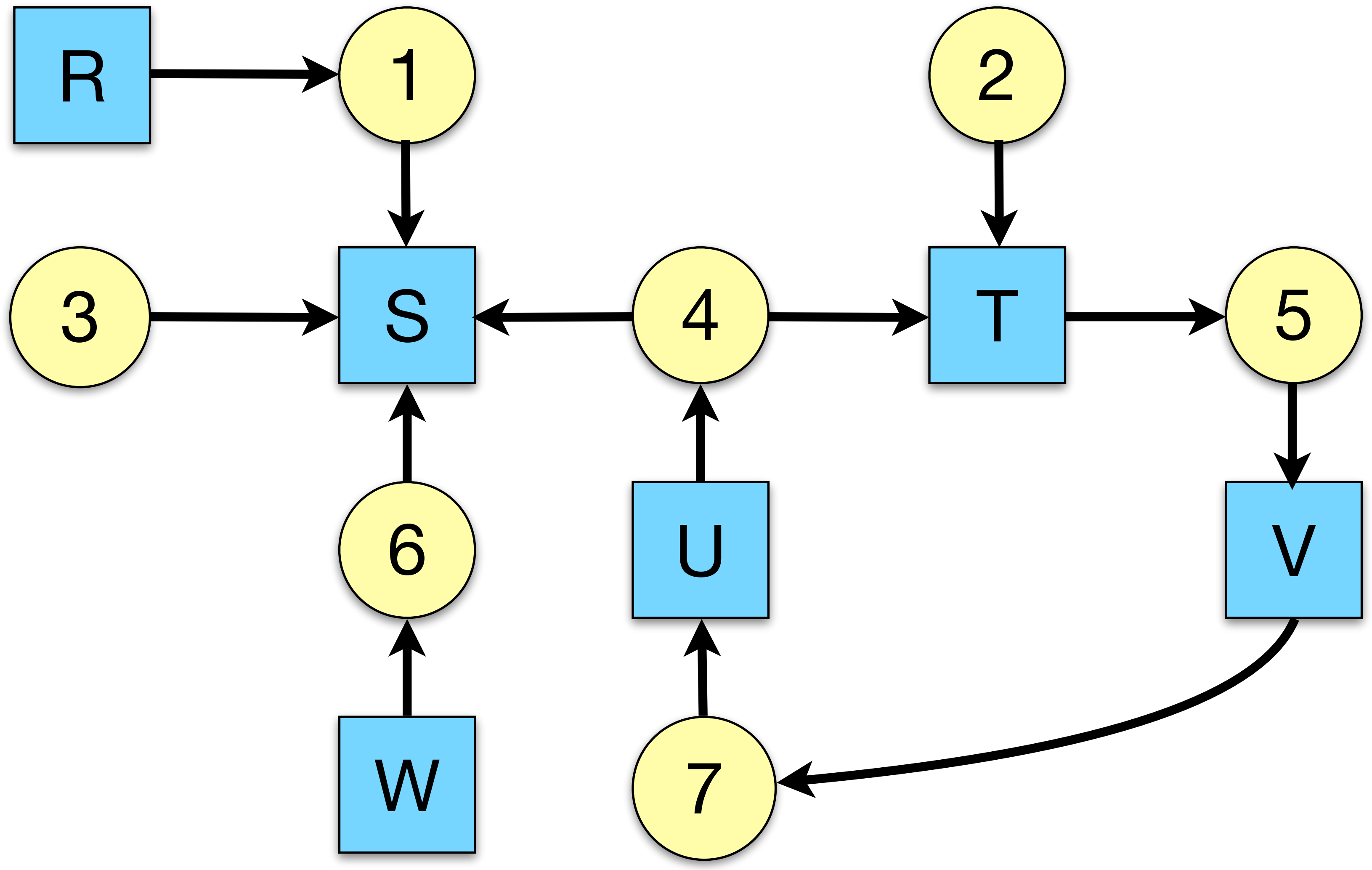
Not getting into deadlock…

- Many situations may result in deadlock (but don't have to)
  - In previous example, A could release R before C requests R, resulting in no deadlock
  - Can we always get out of it this way?
- Find ways to:
  - Detect deadlock and reverse it
  - Stop it from happening in the first place

Detecting deadlocks using graphs

- Process holdings and requests in the table and in the graph (they're equivalent)
- Graph contains a cycle ⇒ deadlock!
  - Easy to pick out by looking at it (in this case)
  - Need to mechanically detect deadlock
- Not all processes are deadlocked (1, 3, 6 not in deadlock)

Process	Holds	Wants
1	R	S
2		T
3		S
4	U	S,T
5	T	V
6	W	S
7	V	U



Recovering from deadlock

- Recovery through preemption
  - Take a resource from some other process
  - Depends on nature of the resource and the process
- Recovery through rollback
  - Checkpoint a process periodically
  - Use saved state to restart the process if it’s in deadlock
  - May present a problem if the process affects lots of “external” things
- Recovery through killing processes
  - Crudest but simplest way to break a deadlock: kill one of the processes in the deadlock cycle
  - Other processes can get its resources
  - Try to choose a process that can be rerun from the start
    - Pick one that hasn’t run too far already

Preventing deadlock

- Deadlock can be completely prevented!
- Ensure that at least one of the conditions for deadlock never occurs
  - Mutual exclusion
  - Circular wait
  - Hold & wait
  - No preemption
- Not always possible…

Eliminating mutual exclusion

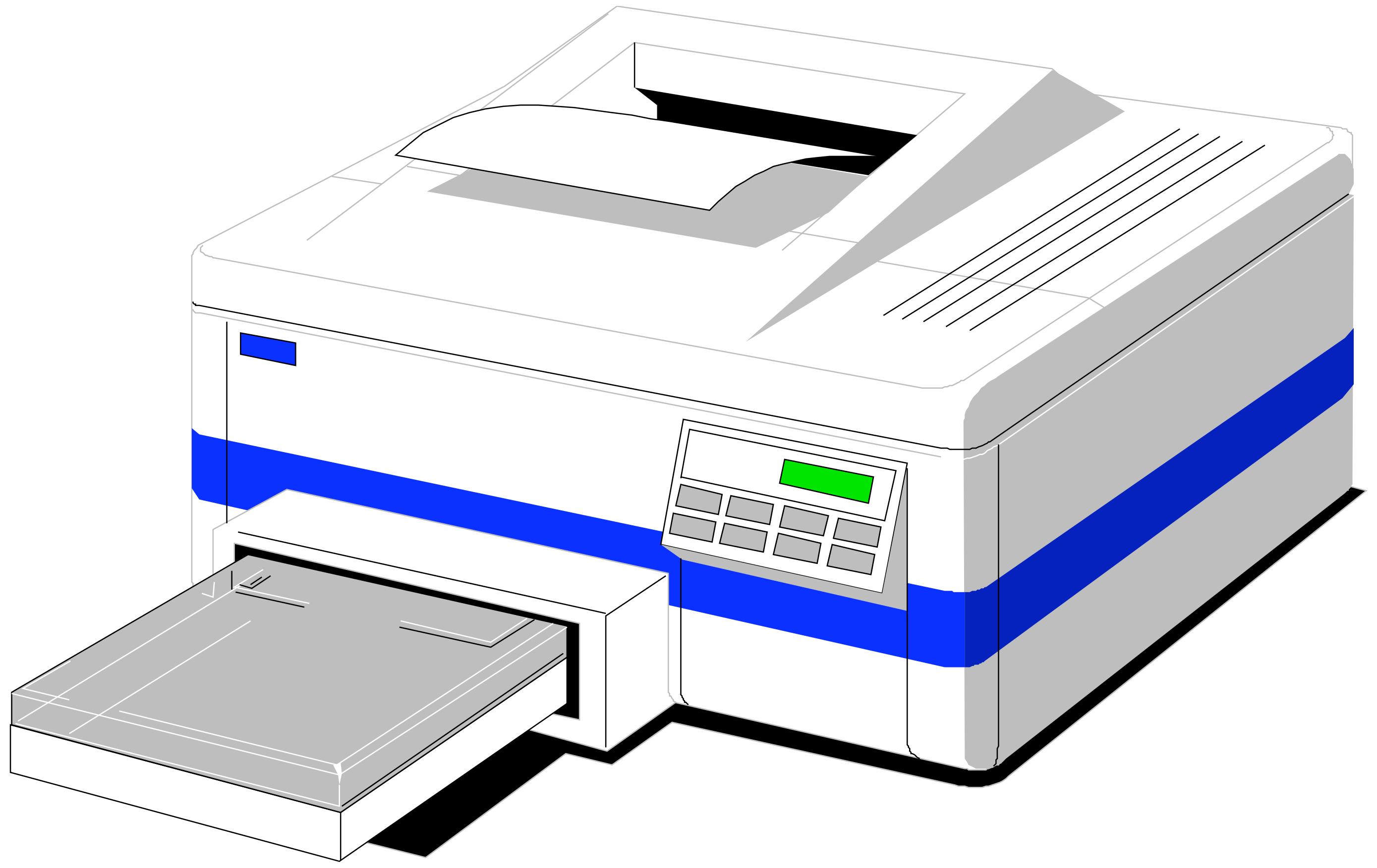
- Some devices (such as printer) can be spooled
  - Only the printer daemon uses printer resource
  - This eliminates deadlock for printer
- Not all devices can be spooled
- Principle:
  - Avoid assigning resource when not absolutely necessary
  - As few processes as possible actually claim the resource

Attacking “hold and wait”

- Require processes to request resources before starting
  - A process never has to wait for what it needs
- This can present problems
  - A process may not know required resources at start of run
  - This also ties up resources other processes could be using
    - Processes will tend to be conservative and request resources they might need
- Variation: a process must give up all resources before making a new request
  - Process is then granted all prior resources as well as the new ones
  - Problem: what if someone grabs the resources in the meantime—how can the process save its state?

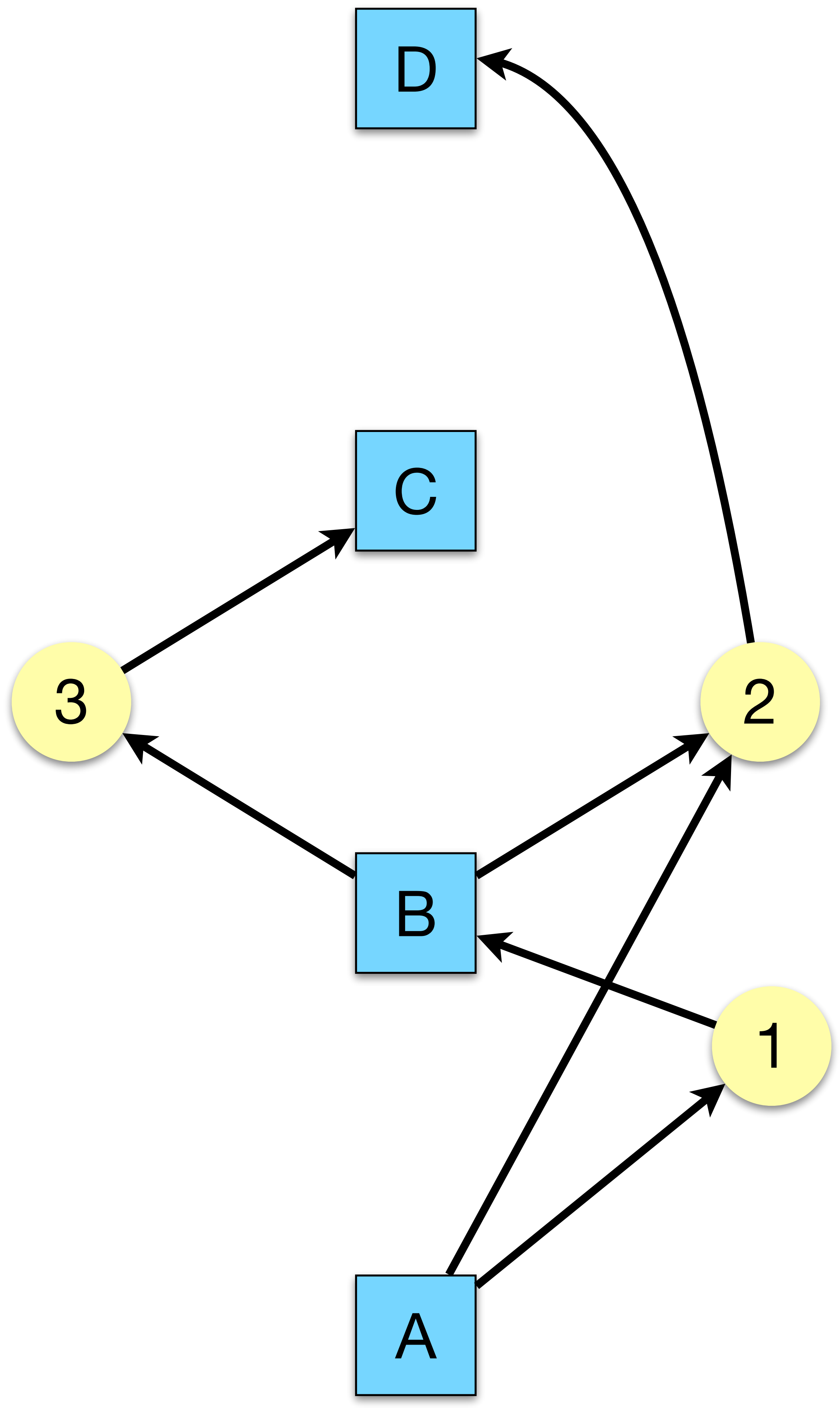
Attacking "no preemption"

- This is not usually a viable option
- Consider a process given the printer
  - Halfway through its job, take away the printer
  - Confusion ensues!
- May work for some resources
  - Forcibly take away memory pages, suspending the process
  - Process may be able to resume with no ill effects



Attacking "circular wait"

- Assign an order to resources
- Always acquire resources in numerical order
  - Need not acquire them all at once!
- Circular wait is prevented
  - A process holding resource *n* can't wait for resource *m* if $m < n$
  - No way to complete a cycle!
    - Place processes above the highest resource they hold and below any they're requesting
    - All arrows point up!



Deadlock prevention: summary

Condition	Prevented by
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away if there's not a complete set
Circular wait	Order resources numerically