Deadlocks

COMP650

Deadlock

- A deadlock occurs when two or more tasks are waiting for each other and they cannot proceed.
- Deadlocks suspend all involved tasks.
- Livelock occurs when some tasks starve another and keep it from completing, but the tasks are still running.

Necessary Conditions for Deadlock

- Mutual Exclusion
- No Preemption
- Hold and Wait
- Circular Wait Graph

These are necessary but not sufficient requirements for deadlock.

Mutual Exclusion

- Deadlock will only occur if a resource cannot be simultaneously used by more than one task.
- Some resources can easily be shared
  - Read only files
  - Code with no shared data

No Preemption

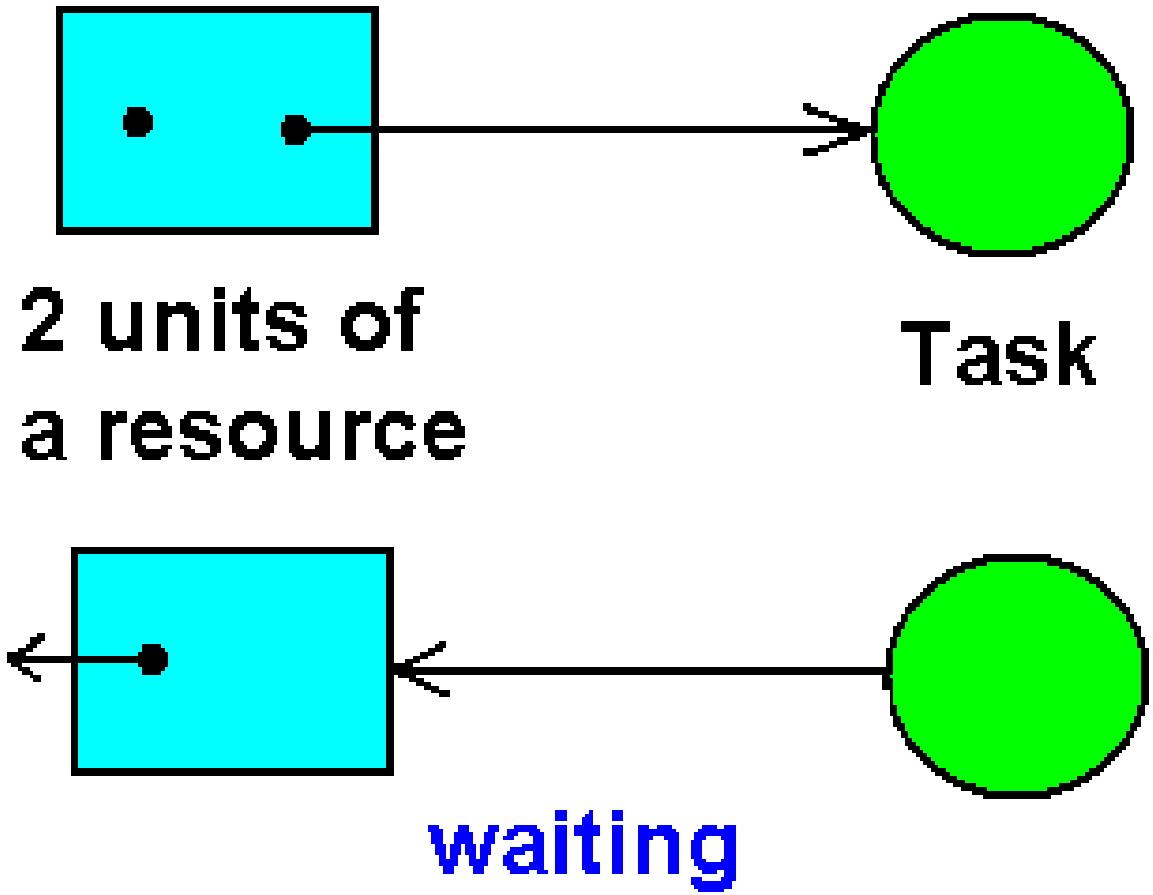
- Deadlock cannot occur if a task can take a resource from another task.
- If high priority tasks can take resources from low priority tasks, the highest priority task will always be able to finish, followed by the next highest priority and so forth.
- Some resources are easy to preempt, like the CPU or RAM.

Hold and Wait

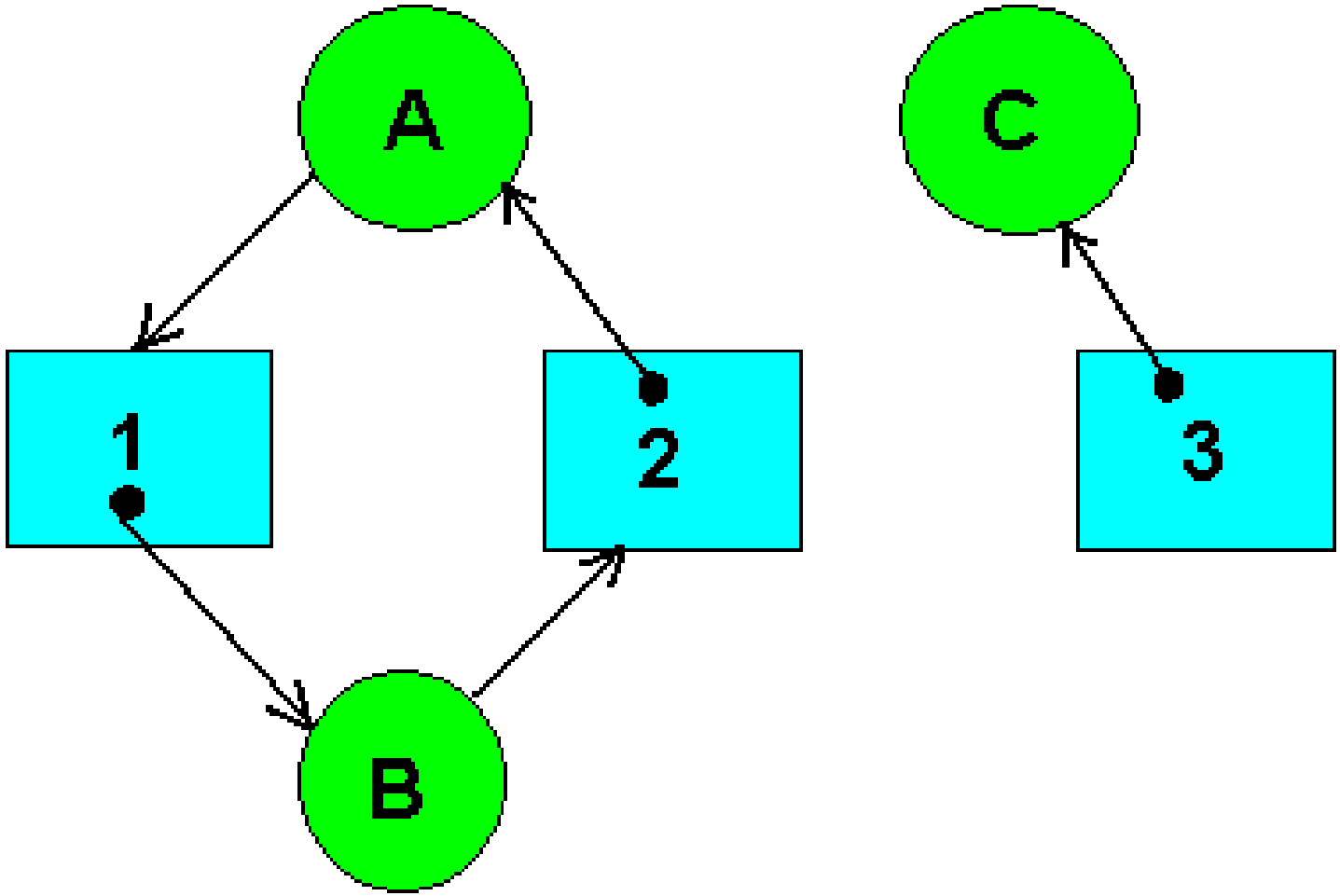
- Deadlock will only occur if a task has exclusive access to a resource and must wait to get access to another resource.
- If a task acquires all the resources it needs at one time, deadlock will not occur.

Circular Wait Graph

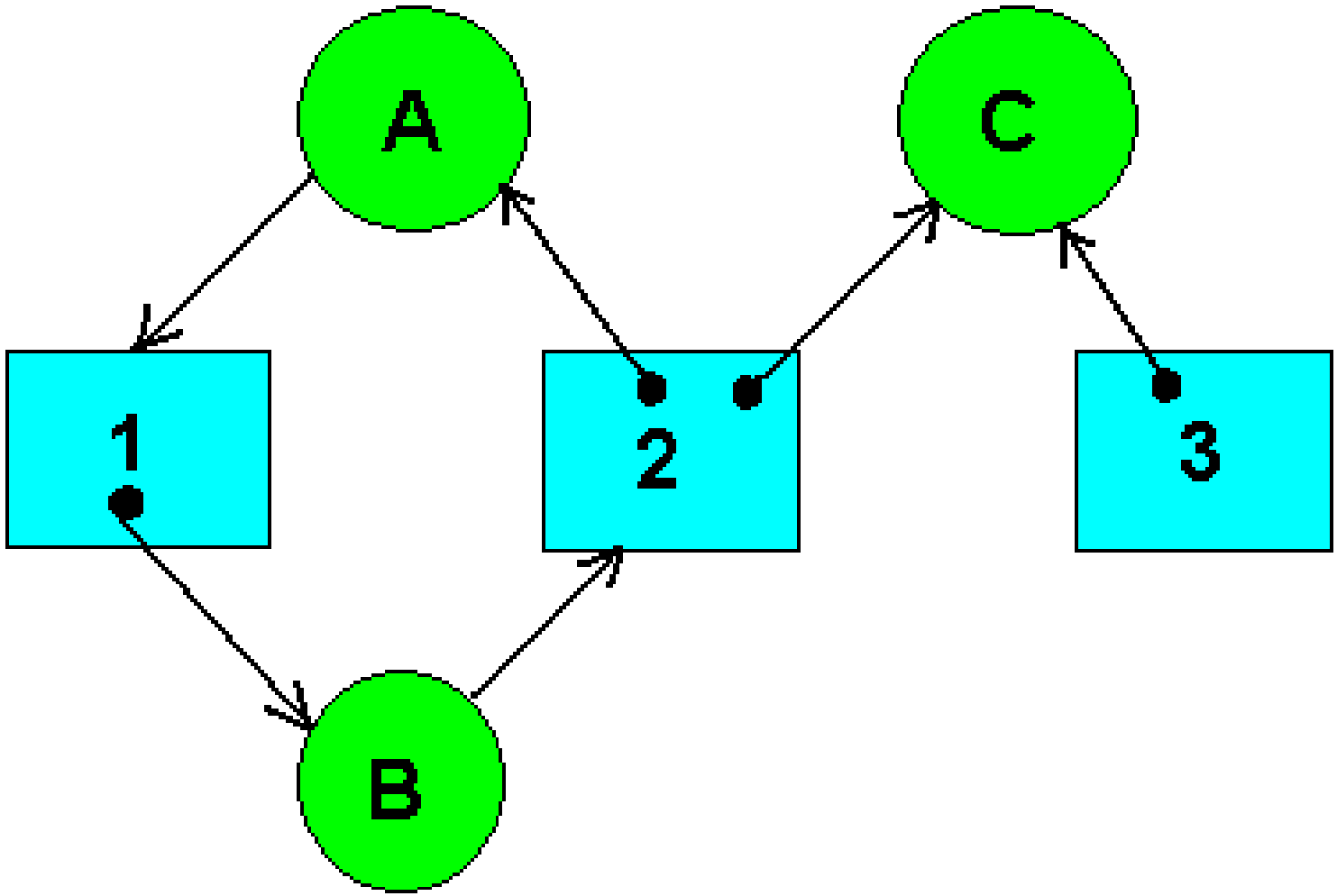
- A deadlock can not occur unless the graph of tasks and resources contains a cycle.



Deadlock Cycle



Non-Deadlock Cycle



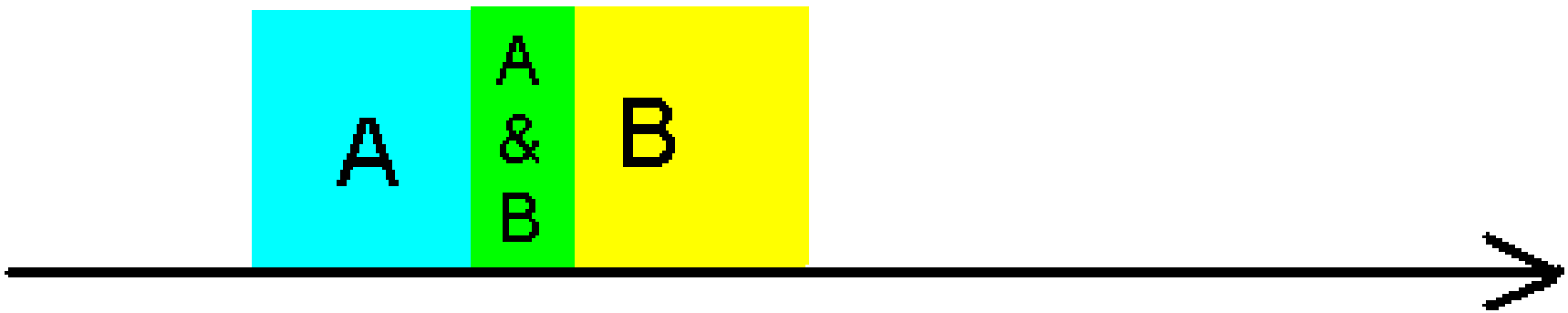
Handling Deadlock

- Ignore deadlocks until they occur, then terminate tasks until the problem goes away.
- Avoid entering into a state that might deadlock. Banker's algorithm does this.
- Design the system so that deadlock cannot occur.

Execution Timeline

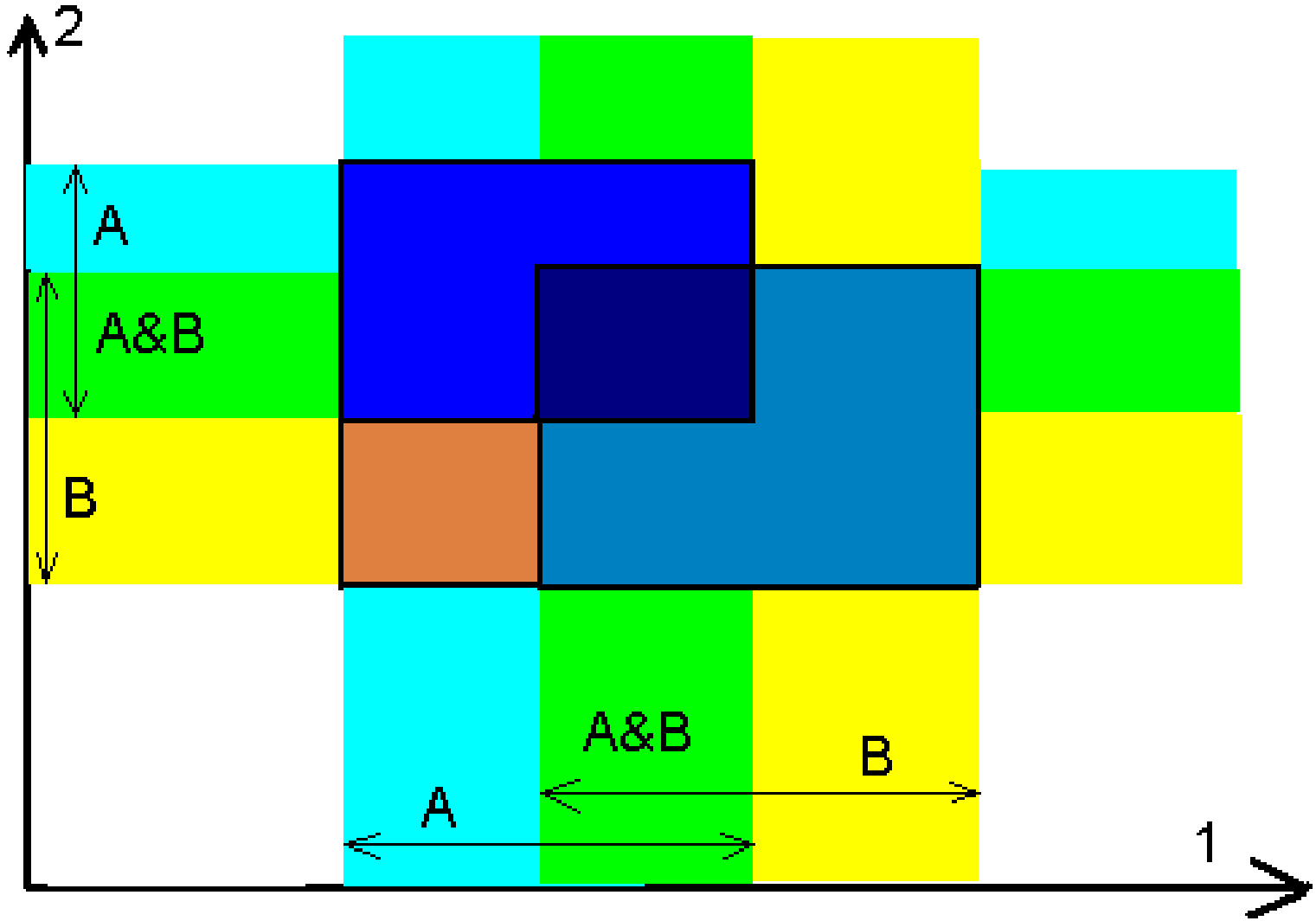
The program acquires resource A, releases it and then acquires resource B.

Execution Timeline



The program acquires resource A then resource B before releasing A.

Unsafe Zone



Deadlock Proofing

- Design the system so that at least one of the necessary condition cannot hold.
- When proving a system cannot deadlock, show that one of the conditions cannot hold.