Cooperating Processes and Dead Lock

Dr Anas Aldabbagh

In this lesson

- Big Projects Problems
- Communication & Coordination
- Starvation and Deadlock
- Conditions for Deadlock
- Methods for Handling Deadlocks

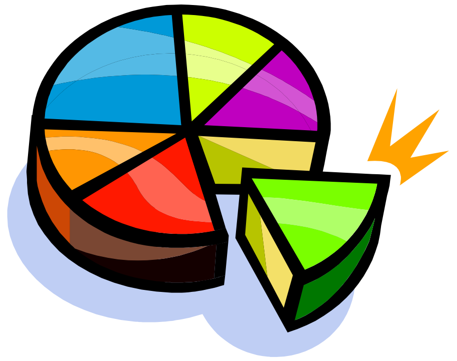
Programming in a Project Team

You just have to get your synchronization

- Big projects require more than one person (or long, long, long time)
  - For example, thousand of persons working for years to build a big OS.
  - It's very hard to make software project teams work correctly
- Deadlines missing might seems acceptable. However, it is very expensive as time-to-market is one of the most important things

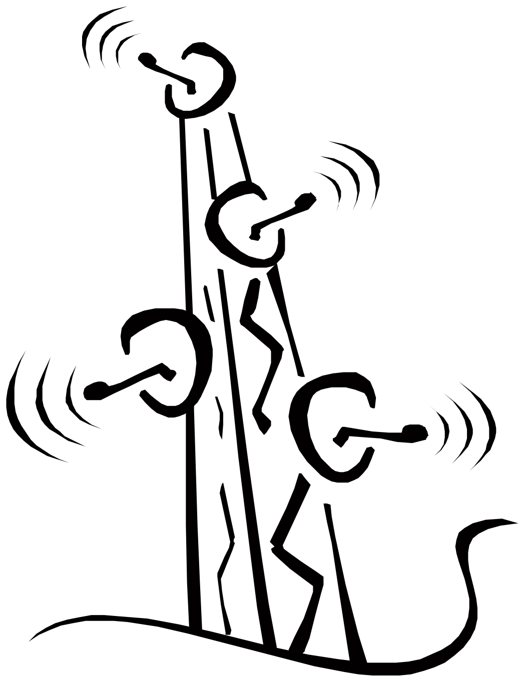
Big Projects Problems

- In big projects time esstimation is very hard. Gneraly, Programmers are often overly optimistic, thinking tasks will only take a short time (like "just two days"). This is why we emphasize starting projects early.
  - For example, grad students who always said they needed “10 minutes” to fix something—two hours later, he was still working on it.
- To solve this a project can be efficiently partitioned. Partitionable task decreases in time as you add people
  - But if you require communication:
    - Time reaches a minimum bound
    - With complex interactions, time increases



Communication

- More people mean more communication
  - ➢ Changes have to be propagated to more people
  - ➢ Think about person writing code for most fundamental component of system: everyone depends on them!
- Who makes decisions?
  - ➢ Individual decisions are fast but trouble
  - ➢ Group decisions take time
  - ➢ Centralized decisions require a big picture view (someone who can be the “system architect”)
- Often designating someone as the system architect can be a good thing
  - ➢ Better not be clueless
  - ➢ Better have good people skills
  - ➢ Better let other people do work

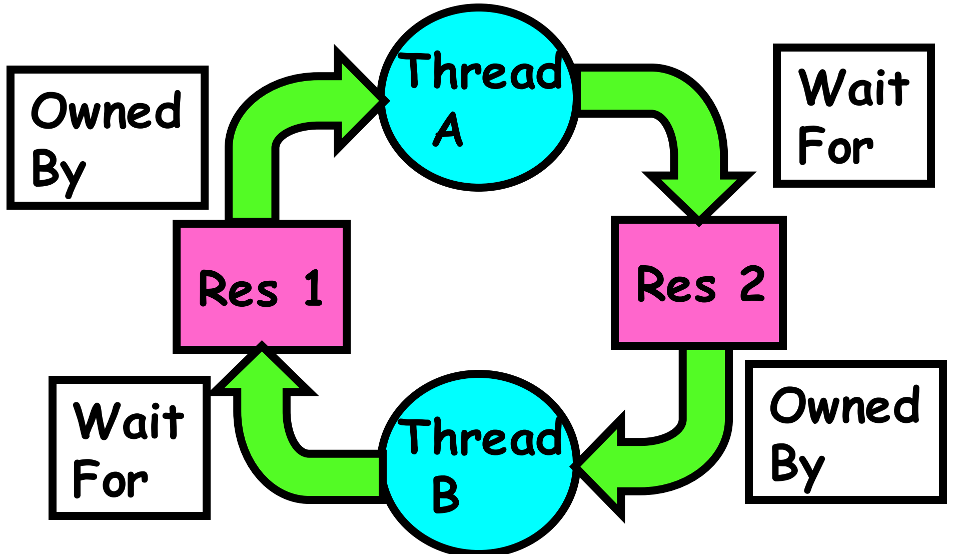


Coordination

- More people ⇒ no one can make all meetings!
  - They miss decisions and associated discussion
  - Example from earlier class: one person missed meetings and did something group had rejected
  - Why do we limit groups to 5 people?
    - ➢ You would never be able to schedule meetings otherwise
  - Why do we require 4 people minimum?
    - ➢ You need to experience groups to get ready for real world
- Hard to add people to existing group
    - ➢ Members have already figured out how to work together

Starvation and Deadlock

- Starvation: thread waits indefinitely
  - ➢ Example, low-priority thread waiting for resources constantly in use by high-priority threads
- Deadlock: circular waiting for resources
  - ➢ Thread A owns Res 1 and is waiting for Res 2
    Thread B owns Res 2 and is waiting for Res 1



Conditions for Deadlock

- Deadlock not always deterministic – Example:

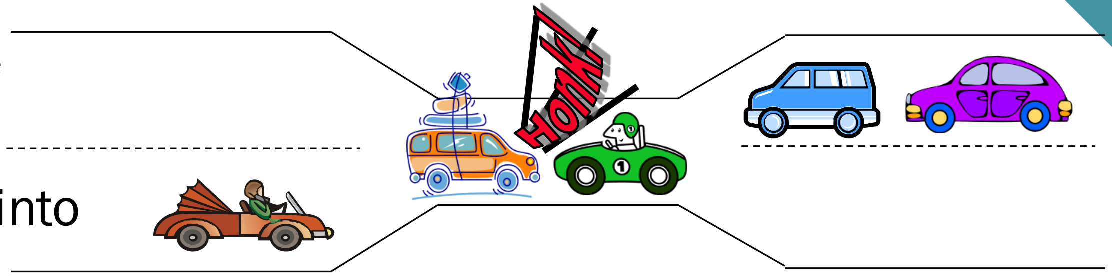
```
Thread A        Thread B
 x.P();          y.P();
 y.P();          x.P();
 y.V();          x.V();
 x.V();          y.V();
```

  - Deadlock won't always happen with this code
    - ➢ Have to have exactly the right timing
    - ➢ So you release a piece of software, and you tested it, and there it is, controlling a nuclear power plant…
- Deadlocks occur with multiple resources
  - ➢ Means you can't decompose the problem
  - ➢ Can't solve deadlock for each resource independently
- Example: System with 2 disk drives and two threads
  - Each thread needs 2 disk drives to function, however, each thread gets one disk and waits for another one.

Bridge Crossing Example

Each segment of road can be viewed as a resource

- Car must own the segment under them
- Must acquire segment that they are moving into



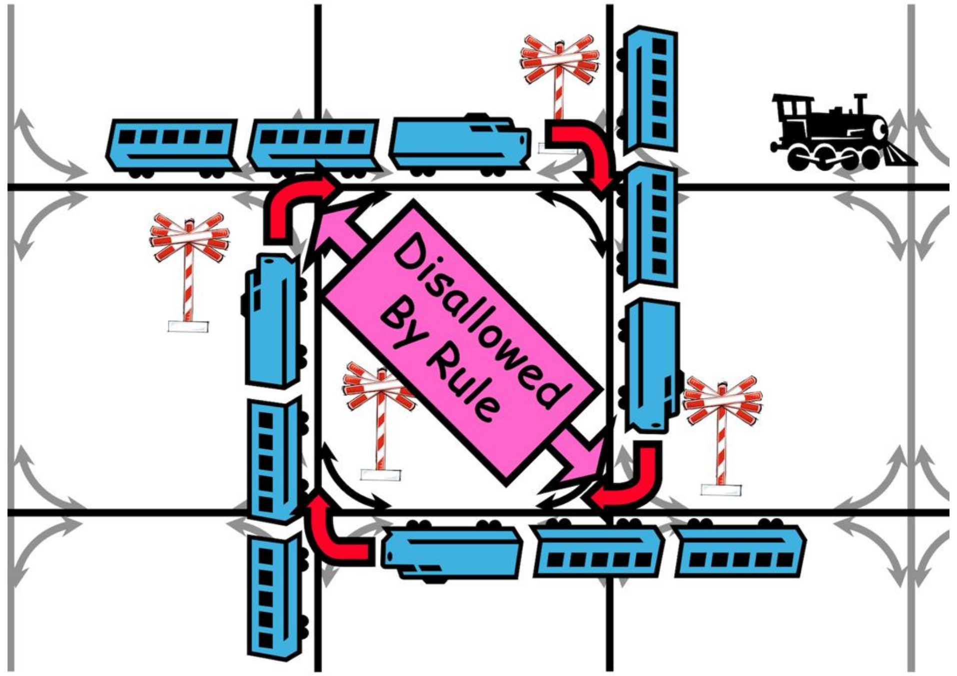
For bridge: must acquire both halves

- Traffic only in one direction at a time
- Problem occurs when two cars in opposite directions on bridge: each acquires one segment and needs next

If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)

Wormhole-Routed Network(Train Example)

- Circular dependency (Deadlock)
  - ➢ Each train wants to turn right, its blocked by other trains
  - ➢ Similar problem to multiprocessor networks
- Fix? Imagine grid extends in all four directions
  - ➢ Force ordering of channels (tracks)



Dining Lawyers Problem

Five chopsticks/Five lawyers (really cheap restaurant)

- Free-for all: Lawyer will grab any one they can
- Need two chopsticks to eat

What if all grab at same time?

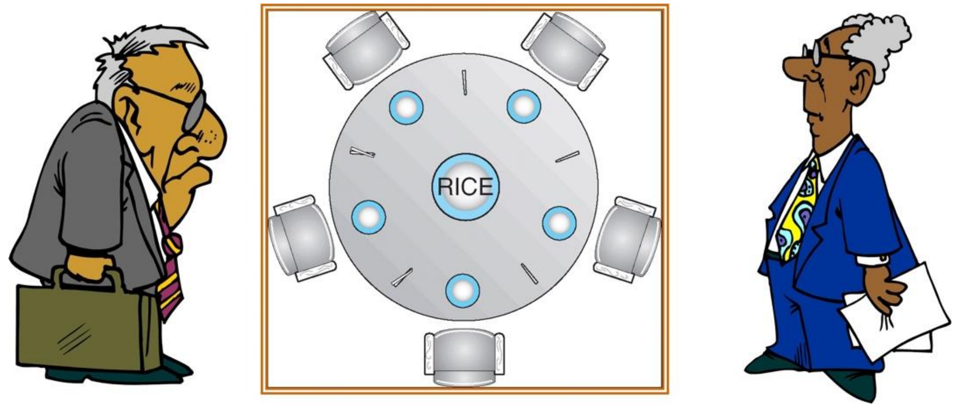
- Deadlock!

How to fix deadlock?

- Make one of them give up a chopstick (Hah!)
- Eventually everyone will get chance to eat

How to prevent deadlock?

- Never let lawyer take last chopstick if no hungry lawyer has two chopsticks afterwards



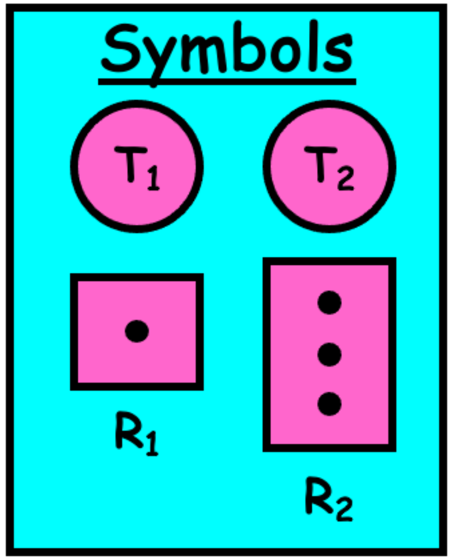
Four requirements for Deadlock

- Mutual exclusion
  - Only one thread at a time can use a resource.
- Hold and wait
  - Thread holding at least one resource is waiting to acquire additional resources held by other threads
- No preemption
  - Resources are released only voluntarily by the thread holding the resource, after thread is finished with it
- Circular wait
  - There exists a set $\{T_1, \ldots, T_n\}$ of waiting threads
    - T_1 is waiting for a resource that is held by T_2
    - T_2 is waiting for a resource that is held by T_3
    - …
    - T_n is waiting for a resource that is held by T_1

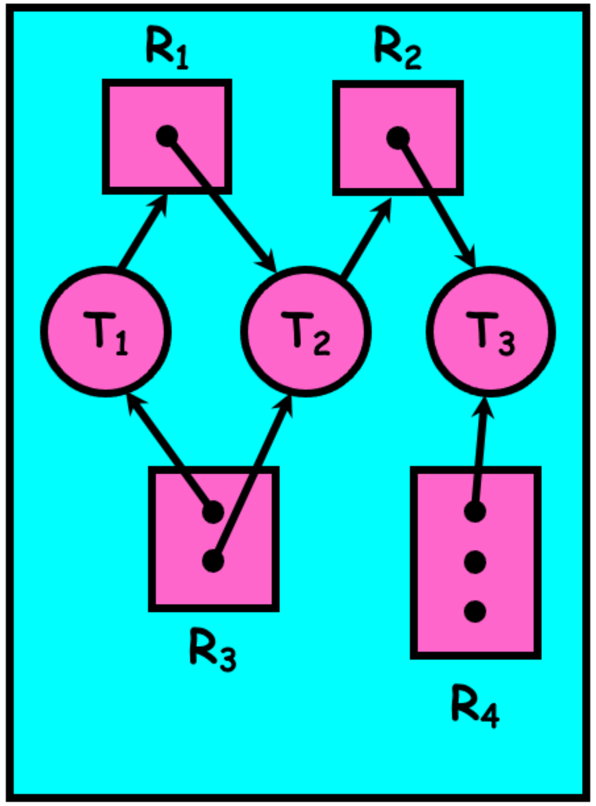
Resource-Allocation Graph

- System Model
  - A set of Threads $T_1, T_2, \ldots, T_n$
  - Resource types $R_1, R_2, \ldots, R_m$

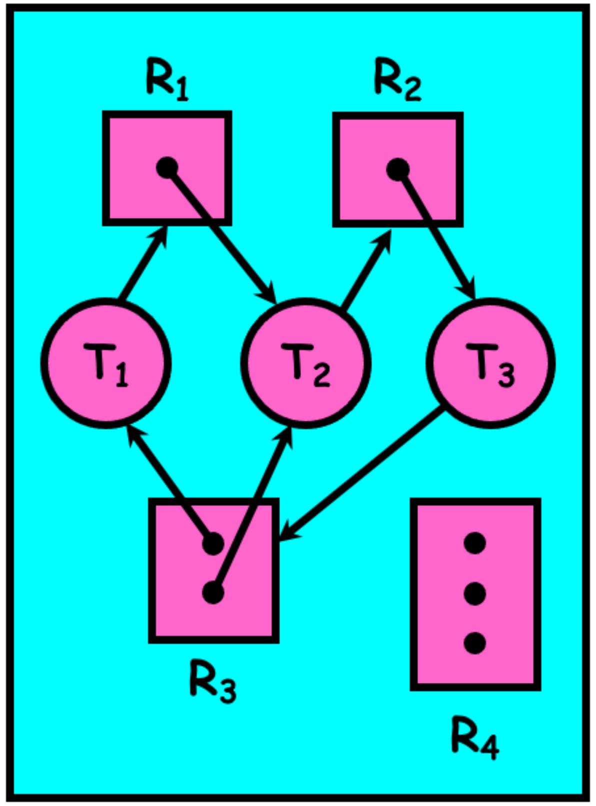
    CPU cycles, memory space, I/O devices
  - Each resource type R_i has W_i instances.
  - Each thread utilizes a resource as follows:
    - `Request() / Use() / Release()`
- Resource-Allocation Graph:
  - V is partitioned into two types:
    - $T = \{T_1, T_2, \ldots, T_n\}$, the set threads in the system.
    - $R = \{R_1, R_2, \ldots, R_m\}$, the set of resource types in system
  - request edge – directed edge $T_1 \rightarrow R_j$
  - assignment edge – directed edge $R_j \rightarrow T_i$



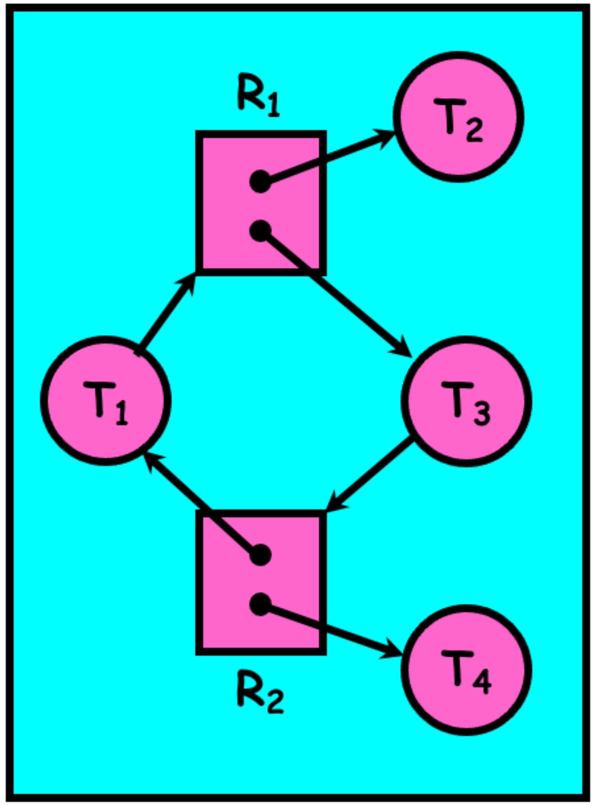
Examples



Simple Resource Allocation Graph

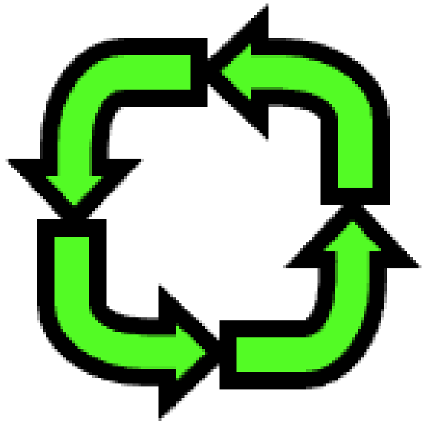


Allocation Graph With Deadlock



Allocation Graph With Cycle, but No Deadlock

What to do when detect deadlock?



- Terminate thread, force it to give up resources
  - In Bridge example, Godzilla picks up a car, hurls it into the river. Deadlock solved!
  - Shoot a dining lawyer
  - But not always possible – killing a thread holding a mutex leaves world inconsistent
- Preempt resources without killing off thread
  - Take away resources from thread temporarily
  - Doesn’t always fit with semantics of computation
- Roll back actions of deadlocked threads
  - Hit the rewind button on a Movie, pretend last few minutes never happened
  - For bridge example, make one car roll backwards (may require others behind him)
  - Common technique in databases (transactions)
  - Of course, if you restart in exactly the same way, may reenter deadlock once again
- Many operating systems use other options

Thank you