# CS 370: OPERATING SYSTEMS
# [DEADLOCKS]

**Deadlock prevention**
Trying to prevent a deadlock?
examine the requirements
negate one of the four
*structurally* … that's all
and you're through

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT
COLORADO STATE UNIVERSITY

1

## Frequently asked questions from the previous class survey

- Niceness level who can set it?
  - Default is 0; root can set niceness level less than 0
- Is the weight table always the same?
- Are sched_latency and min_granularity every adjusted?
  - What happens when a process crashes? Does it use up its quanta?
- Why is the decision time in CFS O(1)?
- Can a new process cause starvation? [similar to I/O processes]
- Do idle threads have uses beyond energy savings?

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT DEADLOCKS L17.2

2

## Topics covered in this lecture

- Dealing with Deadlocks
- Deadlock Prevention
- Deadlock Avoidance



3

## DEADLOCKS VS. STARVATION



4

## Deadlocks vs. Starvation [1/2]

- Deadlocks and starvation are both **liveness** concerns
- Starvation
  - Task fails to make progress for an indefinite period of time
- Deadlock is a *form of starvation*, BUT with a stronger condition
  - A **group of tasks** forms a **cycle** where *none* of the tasks makes progress
    - Because each task is waiting for some other task in the cycle to take action



## Deadlocks vs. Starvation [2/2]

- Deadlock implies starvation (literally for the dining philosophers problem)
- Starvation DOES NOT imply deadlock

## Also …

- Just because a system can suffer deadlock or starvation does not mean that it always will
  - A system is *subject to starvation* if a task could starve in some circumstances
  - A system is *subject to deadlock* if a group of tasks could deadlock in some circumstances
- **Circumstances** impact whether a deadlock or starvation may occur
  - Choices made by scheduler, number of tasks, workload or sequence of requests, which tasks win races to acquire locks, order of task activations, etc.





# RESOURCE ALLOCATION GRAPH

## Resource allocation graph

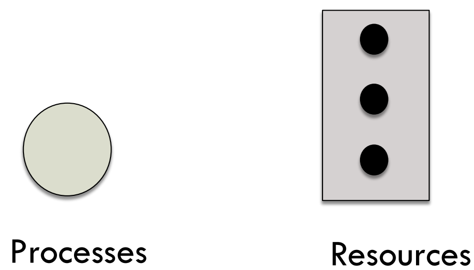
- Used to describe deadlocks precisely
- Consists of a set of vertices and edges
- Two different sets of nodes
  - P: the set of all **active processes** in system
  - R: the set of all **resource types** in the system



## Directed edges

- **Request** edge
  - $P_i$ has requested an instance of resource type $R_j$
  - Directed edge from process $P_i$ to resource $R_j$
  - Denoted $P_i \rightarrow R_j$
  - *Currently waiting* for that resource
- **Assignment** edge
  - Instance of resource $R_j$ assigned to process $P_i$
  - Directed edge from resource $R_j$ to process $P_i$
  - Denoted $R_j \rightarrow P_i$

## Representation of Processes and Resources

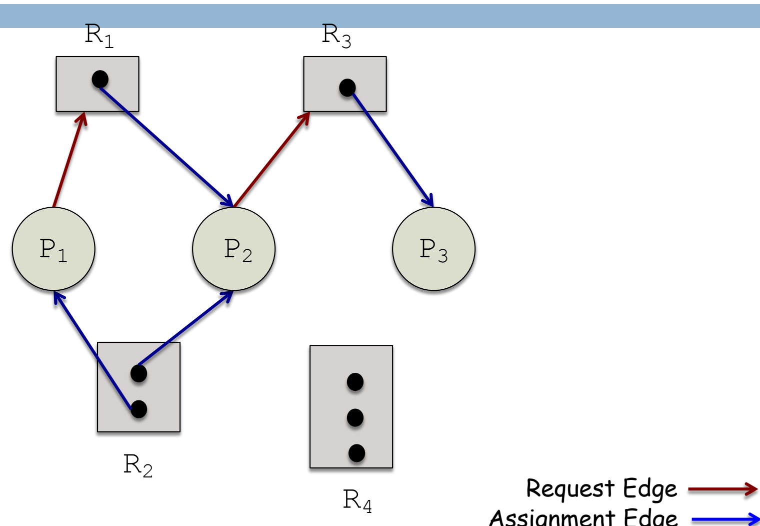



A resource type may have multiple instances



11

## Resource Allocation Graph example






12

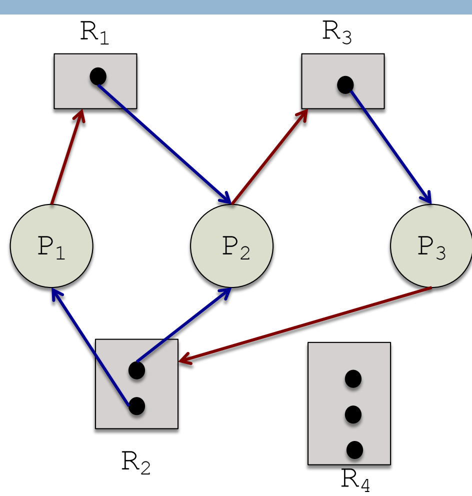
# Determining deadlocks

- If the graph contains **no cycles**?
  - <u>No process</u> in the system is deadlocked
- If there is a **cycle** in the graph?
  - If each resource type has *exactly one* instance
    - Deadlock ***<u>has</u>*** occurred
  - If each resource type has *multiple* instances
    - A deadlock ***<u>may have</u>*** occurred

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA | COMPUTER SCIENCE DEPARTMENT | DEADLOCKS | L17.13

13

# Resource Allocation Graph: Deadlock example




**Two cycles**

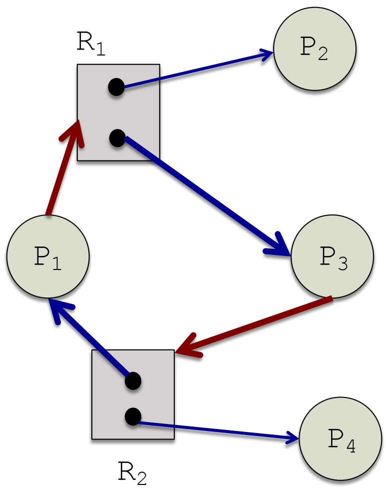
$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA | COMPUTER SCIENCE DEPARTMENT | DEADLOCKS | L17.14

14

## Resource Allocation Graph: Cycle but not a deadlock



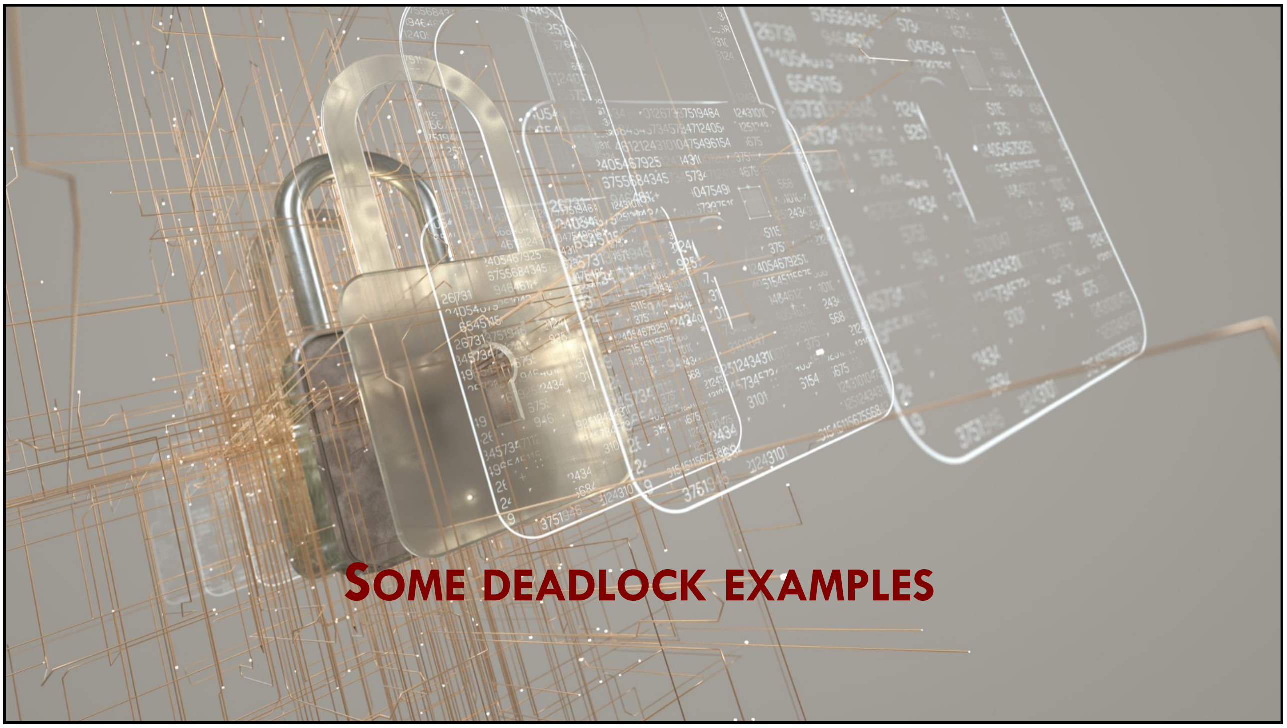

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

$P_4$ may release instance of $R_2$ allocate to $P_3$ and break cycle



## Resource Allocation Graphs and Deadlocks

- If the graph does not have a cycle
  - No deadlock
- If the graph does have a cycle
  - System may or may not be deadlocked



## SOME DEADLOCK EXAMPLES

17

## Law passed by Kansas Legislature … early 20th Century

*"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone"*

COLORADO STATE UNIVERSITY — Professor: SHRIDEEP PALLICKARA, COMPUTER SCIENCE DEPARTMENT — DEADLOCKS — L17.18

18

## Dining philosophers problem: Necessary conditions for deadlock (1)

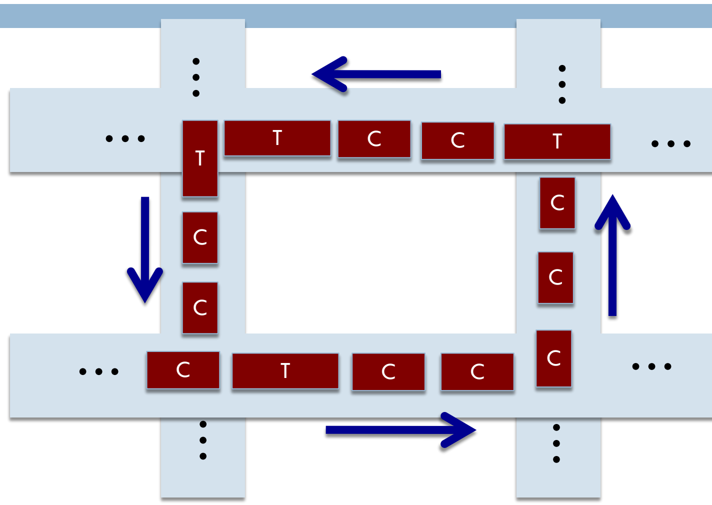
- Mutual exclusion
  - 2 philosophers *cannot share* the same chopstick
- Hold-and-wait
  - A philosopher *picks up one* chopstick at a time
  - Will not let go of the first while it *waits for the second* one



## Dining philosophers problem: Necessary conditions for deadlock (2)

- No preemption
  - A philosopher *does not snatch chopsticks* held by some other philosopher
- Circular wait
  - Could happen if each philosopher *picks chopstick with the same hand* first

## Is there a traffic deadlock here?








## The traffic scenario: Necessary Conditions (1)

- Mutual Exclusion
  - A vehicle needs its *own space*
  - We can't stack automobiles on top of each other
- Hold-and-wait
  - A vehicle does not move and *stays in place* if it cannot advance

## The traffic scenario: Necessary Conditions (2)

- No preemption
  - We *cannot move* an automobile to the side
- Circular-wait
  - Each vehicle is waiting for the one in front of it to advance



23

## DEALING WITH DEADLOCKS



24

## Four strategies for dealing with deadlocks

- Ignore the problem
  - May be if you ignore it, it will ignore you
- Deadlock prevention
  - By structurally negating one of the four required conditions
- Deadlock avoidance
  - By careful resource allocation
- Detection and Recovery
  - Let deadlocks occur, detect them, and take action



25


26

## Ostrich Algorithm

- Stick your head in the sand; pretend there is no problem at all
- Reactions
  - Mathematician: Unacceptable; prevent at all costs
  - Engineers: How often? Costs? Etc.



## OS suffer from deadlocks that are not even detected [1/3]

- Number of processes in the system
  - Total determined by slots in the process table
    - Slots are a finite resource
- Maximum number of open files
  - Restricted by size of the inode table
- Swap space on the disk

## OS suffer from deadlocks that are not even detected [2/3]

- Every OS table represents a **finite** resource
- Should we abolish all of these because collection of $n$ processes
  1. Might claim $1/n$ th of the total AND
  2. Then try to claim another one
- Most users prefer occasional deadlock to a restrictive policy
  - E.g., All users: 1 process, 1 open file …. one everything is far too restrictive



## OS suffer from deadlocks that are not even detected [3/3]

- If deadlock elimination is free
  - No discussions
- But the price is often high
  - Inconvenient restrictions on processes
- Tradeoff
  - Between **convenience** and **correctness**

# DEADLOCK CHARACTERIZATION

COMPUTER SCIENCE DEPARTMENT    COLORADO STATE UNIVERSITY

31

## Deadlocks: Necessary Conditions (I)

- **Mutual Exclusion**
  - At least one resource held in *nonsharable* mode
  - When a resource is being used
    - Another requesting process must wait for its release
- **Hold-and-wait**
  - A process must hold one resource
  - Wait to acquire additional resources
    - Which are currently held by other processes

COLORADO STATE UNIVERSITY    Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT    DEADLOCKS    L17.32

32

## Deadlocks: Necessary Conditions (II)

- **No preemption**
  - Resources cannot be preempted
  - Only voluntary release by process holding it
- **Circular wait**
  - A set of {$P_0$, $P_1$, …, $P_n$} waiting processes must exist
    - $P_0 \rightarrow P_1$; $P_1 \rightarrow P_2$, …, $P_n \rightarrow P_0$
  - Implies hold-and-wait



33

Hanging on
You're all that's left to hold on to
I'm still waiting
I'm hanging on
You're all that's left to hold on to
Red Hill Mining Town, The Joshua Tree, U2

## DEADLOCK PREVENTION



34

## Deadlock Prevention

- Ensure that **one** of the necessary conditions for deadlocks *cannot* occur
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait



## Mutual exclusion must hold for non-sharable resources, but …

- Sharable resources do not require mutually exclusive access
  - *Cannot be involved* in a deadlock
- A process never needs to wait for sharable resource
  - Read-only files
- Some resources are *intrinsically nonsharable*
  - So, denying mutual exclusion often not possible

## Deadlock Prevention: Ensure hold-and-wait never occurs in the system [Strategy 1]

- Process must request and be allocated all its resources **before** execution
  - Resource requests must precede other system calls
- E.g., copy data from DVD drive, sort file, & print
  - Printer needed only at the end
  - BUT process will hold printer for the *entire* execution





## Deadlock Prevention: Ensure hold-and-wait never occurs in the system [Strategy 2]

- Allow a process to request resources *only when it has none*
  - *Release* all resources, *before requesting* additional ones
- E.g., copy data from DVD drive, store file, & print
  - First request DVD and disk file
    - Copy and release resources
  - Then request file and printer

## Disadvantages of protocols targeting hold-and-wait

- **Low resource utilization**
  - Resources are allocated but unused for long durations
- **Starvation**
  - If a process needs several popular resources
    - Popular resource might always be *allocated to some other* process



## Deadlock Prevention: Eliminate the preemption constraint [1/2]

- {C1} If a process is holding some resources
- {C2} Process requests another resource
  - Cannot be immediately allocated
- All resources currently held by process is **preempted**
  - Preempted resources added to list of resources process is waiting for

## Deadlock Prevention: Eliminate the preemption constraint [2/2]

- Process requests resources that are not currently available
  - If resources are allocated to another waiting process?
    - Preempt resources from the second process and assign it to the first one
- Often applied when resource state can be ***saved and restored***
  - CPU registers and memory space
  - Unsuitable for tape drives



## Deadlock Prevention: Eliminating Circular wait

- Impose **total ordering** of all resource types
  - Assign each resource type a unique number
  - One-to-one function `F:R→N`

```
F(tape drive) = 1;
F(printer) = 12
```

① Request resources in ***increasing* order**

② If several instances of a resource type needed?
  - Single request for all them must be issued

## Requesting resources in an increasing order of enumeration

- Process initially requested $R_i$
- This process can now request $R_j$ ONLY IF
  $F(R_j) > F(R_i)$
- Alternatively, process requesting $R_j$ must have released resources $R_i$ such that
  $F(R_i) >= F(R_j)$
- Eliminates circular wait



## Hierarchy of resources and deadlock prevention

- Hierarchy by itself does not prevent deadlocks
  - Developed programs **must follow ordering**
- **F** *based on* ***order*** *of usage* of resources
  - Tape drive needed before printing
    - `F(tape drive) < F(printer)`

## Deadlock Prevention: **Summary**

- Prevent deadlocks by **restraining** how requests are made
  - Ensure at least 1 of the 4 conditions *cannot* occur
- Side effects:
  - Low device utilization
  - Reduced system throughput



## Dining Philosophers: Deadlock prevention strategies [1/2]

- Mutual exclusion
  - Philosophers can *share* a chopstick
- Hold-and-wait
  - Philosopher should release the first chopstick if it cannot obtain the second one

## Dining Philosophers: Deadlock prevention strategies [2/2]

- Preemption
  - Philosophers can *forcibly take* each other's chopstick
- Circular-wait
  - Number the chopsticks
  - Pick up chopsticks in ascending order
    - Pick the lower numbered one before the higher numbered one

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT DEADLOCKS L17.47

47

## DEADLOCK AVOIDANCE

COMPUTER SCIENCE DEPARTMENT COLORADO STATE UNIVERSITY

48

## Deadlock avoidance

- Require *additional* information about **how** resources are to be requested
- Knowledge about sequence of requests and releases for processes
  - Allows us to decide if resource allocation *could cause a future deadlock*
  - Process P: Tape drive, then printer
  - Process Q: Printer, then tape drive



## Deadlock avoidance: Handling resource requests

- For each resource request:
  - Decide whether or not process should wait
    - To avoid possible **future** deadlock
- Predicated on:
  1. Currently available resources
  2. Currently allocated resources
  3. Future requests and releases of each process

## Avoidance algorithms differ in the amount and type of information needed

- **Resource allocation state**
  - Number of available and allocated resources
  - Maximum demands of processes
- Dynamically **examine** resource allocation state
  - Ensure circular-wait cannot exist
- Simplest model:
  - Declare maximum number of resources for each type
  - Use information to avoid deadlock



## Safe sequence

- **Sequence** of processes <$P_1$, $P_2$, …, $P_n$> for the current allocation state
- Resource requests made by $P_i$ can be satisfied by:
  - Currently available resources
  - Resources held by $P_j$ where $j < i$
    - If needed resources not available, $P_i$ can wait
  - In general, when $P_i$ terminates, $P_{i+1}$ can obtain its needed resources
- If no such sequence exists: system state is **unsafe**

## Deadlock avoidance: Safe states

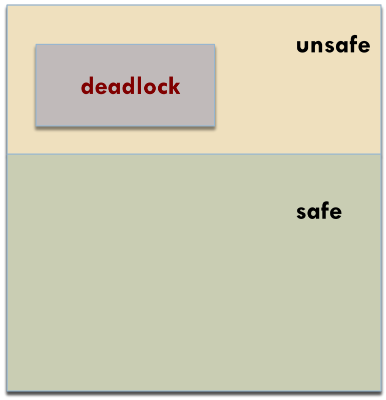
- If the system can:
  1. Allocate resources to each process in **some order**
     - Up to the *maximum* for the process
  2. Still avoid deadlock



## Safe states and deadlocks

- A system is safe ONLY IF there is a **safe sequence**
- A safe state is not a deadlocked state
  - Deadlocked state is an unsafe state
  - Not all unsafe states are deadlocks

## State spaces






55

## Unsafe states

- An unsafe state *may lead* to deadlock
- **Behavior** of processes controls unsafe states
- Cannot prevent processes from requesting resources such that deadlocks occur



56

## Example: 12 Tape drives available in the system

| | Maximum Needs | Current Allocation |
|---|---|---|
| $P_0$ | 10 | 5 |
| $P_1$ | 4 | 2 |
| $P_2$ | 9 | 2 |

**Before T0**:
3 drives available

Safe sequence
<$P_1$, $P_0$, $P_2$>

- At time **T0** the system is in a safe state
- $P_1$ can be given 2 tape drives
- When $P_1$ releases its resources; there are 5 drives
- $P_0$ uses 5 and subsequently releases them (# 10 now)
- $P_2$ can then proceed



## Example: 12 Tape drives available in the system

| | Maximum Needs | Current Allocation |
|---|---|---|
| $P_0$ | 10 | 5 |
| $P_1$ | 4 | 2 |
| $P_2$ | 9 | 2 |

**Before T1**:
3 drives available

- At time **T1**, $P_2$ is allocated 1 tape drive

## Example: 12 Tape drives available in the system

| | Maximum Needs | Current Allocation |
|---|---|---|
| $P_0$ | 10 | 5 |
| $P_1$ | 4 | 2 |
| $P_2$ | 9 | 3 |

**After T1**:
2 drives available

- At time **T1**, $P_2$ is allocated 1 tape drive
- Only $P_1$ can proceed
- When $P_1$ releases its resources; there are 4 drives
  - $P_0$ needs 5 and $P_2$ needs 6
- **Mistake** in granting $P_2$ additional tape drive



## Crux of deadlock avoidance algorithms

- **Ensure** that the system will always remain in a safe state
- Resource allocation request **granted** only if it will leave the system in a safe state

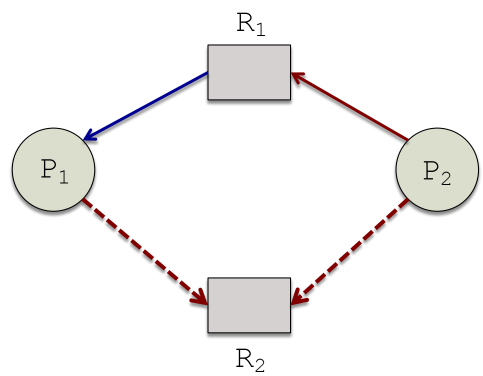
## RESOURCE ALLOCATION GRAPH ALGORITHM



## Claim edges

- Indicates that a process $P_i$ may request a resource $R_j$ at some time in the future
- Representation:
  - Same direction as request
  - **Dotted line**

## Resource allocation graph with a claim edge






63

## Conversion of claim edges

- When process $P_i$ requests resource $R_j$
  - Claim edge converted to a request edge
- When resource $R_j$ released by $P_i$
  - The assignment edge $R_j \rightarrow P_i$ is **reconverted** to a claim edge $P_i \rightarrow R_j$



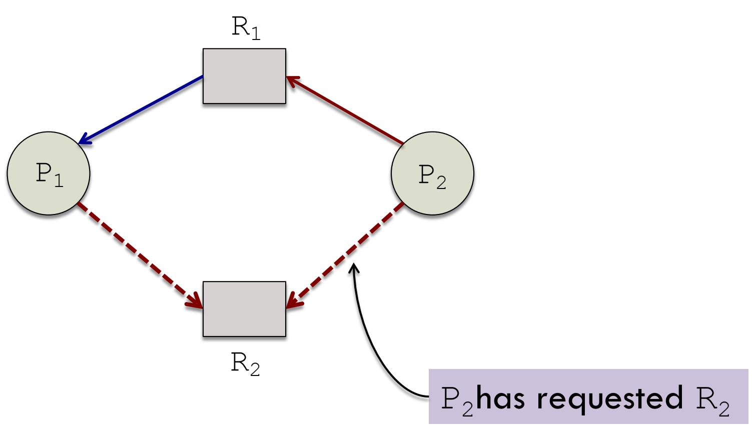
64

## Allocating resources

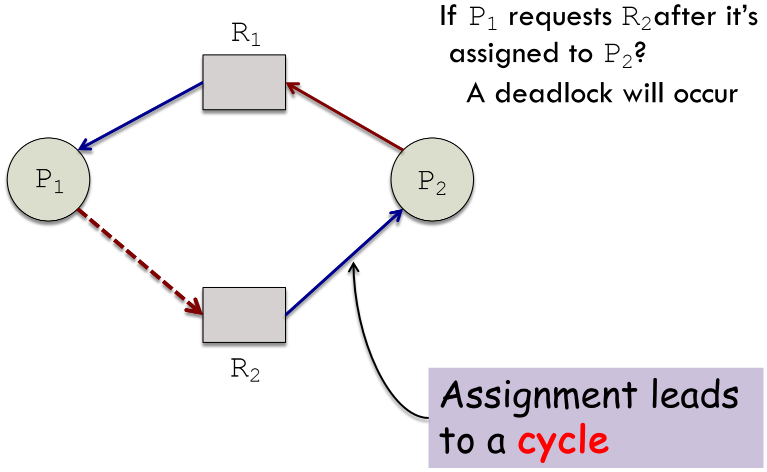
- When process $P_i$ requests resource $R_j$
- Request granted only if
  - Converting claim edge to $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ *does not result* in a **cycle**



## Using the allocation graph to allocate resources safely



## Using the allocation graph to allocate resources safely






67

## Resource allocation graph algorithm

- Not applicable in systems with multiple resource instances



68

## The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 7]
- *Andrew S Tanenbaum. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 6]*