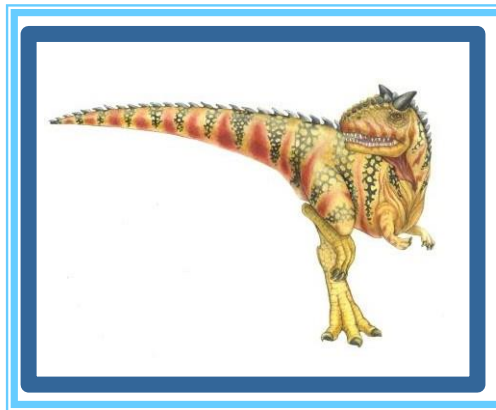
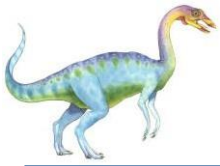


Chapter 6: Deadlocks

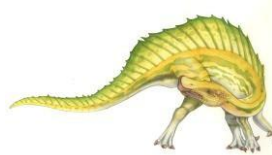


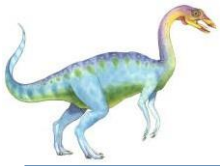


Chapter 6: Deadlocks



- The Deadlock Problem
- System Model
- Deadlock Characterization ลักษณะ Deadlocks
- Methods for Handling Deadlocks วิธีการจัดการ Deadlocks
- Deadlock Prevention การป้องกัน Deadlocks
- Deadlock Avoidance การหลีกเลี่ยง Deadlocks
- Deadlock Detection การตรวจจับ Deadlocks
- Recovery from Deadlock





Chapter Objectives

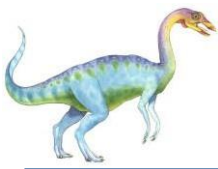
วัตถุประสงค์ของบท

เพื่อพัฒนาคำอธิบายของ **deadlocks** ซึ่งป้องกันไม่ให้เกิดของกระบวนการที่เกิดขึ้นพร้อมกันทำงานให้เสร็จสิ้น

- To develop a description of deadlocks, which prevent sets of concurrent processes from completing their tasks
- To present a number of different methods for preventing or avoiding deadlocks in a computer system

เพื่อนำเสนอวิธีการต่างๆ มากมายในการป้องกันหรือหลีกเลี่ยง **deadlocks** ในระบบคอมพิวเตอร์





The Deadlock Problem

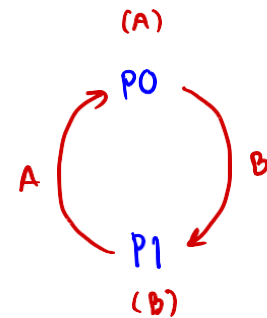
ปัญหา Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set
- Example ชุดของกระบวนการที่ถูกบล็อกแต่ละกระบวนการถือทรัพยากรและรอรับทรัพยากรที่ถือโดยกระบวนการอื่นในชุด
 - System has 2 disk drives ระบบมีดิสก์ไดรฟ์ 2 ตัว
 - P_1 and P_2 each hold one disk drive and each needs another one
- Example P_1 และ P_2 แต่ละตัวมีดิสก์ไดรฟ์หนึ่งตัว และแต่ละตัวต้องการอีกตัวหนึ่ง

- semaphores A and B , initialized to 1

เซมาฟอร์ A และ B เริ่มต้นเป็น 1

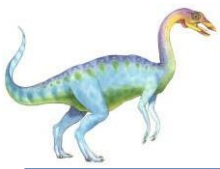
	P_0		P_1	
$A = 1 - 1 = 0$	wait (A);		wait(B)	$B = 1 - 1 = 0$
	wait (B);		wait(A)	



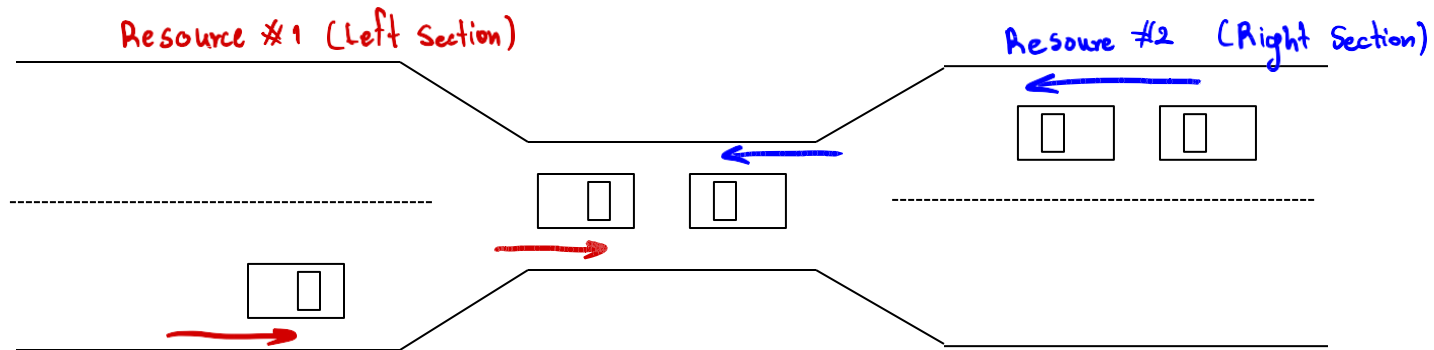
ทำงานไม่จบ While loop

acquire: อยากครอบครอง



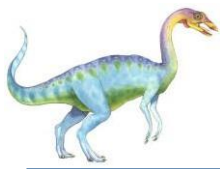


Bridge Crossing Example



- การจราจรในทิศทางเดียวเท่านั้น
- Traffic only in one direction
- แต่ละส่วนของบริดจ์สามารถดูได้เป็นทรัพยากร
- Each section of a bridge can be viewed as a resource
- หากเกิด deadlock สามารถแก้ไขได้หากมีรถคันหนึ่งสำรอง (จองทรัพยากรและการย้อนกลับ)
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
- อาจต้องสำรองขอมูลรถยนต์หลายคันหากเกิดการหยุดชะงัก
- Several cars may have to be backed up if a deadlock occurs
- ความมอดอยากเป็นไปได้
- Starvation is possible
- Ex Some car may always be preempted and never cross the bridge
- หมายเหตุ – ระบบปฏิบัติการส่วนใหญ่ไม่ได้ป้องกันหรือจัดการกับ deadlocks
- Note – Most OSes do not prevent or deal with deadlocks





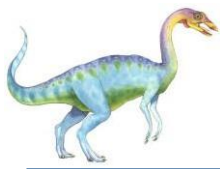
System Model



- ชนิดทรัพยากร
- Resource types R_1, R_2, \dots, R_m
 - รอบของ CPU, พื้นที่หน่วยความจำ, อุปกรณ์ I/O
 - CPU cycles, memory space, I/O devices*
 - ทรัพยากรแต่ละประเภท R_i มีอินสแตนซ์ W_i แต่ละกระบวนการใช้
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - request
 - use
 - release

มีทรัพยากร เป็น ชนิด
ชนิดที่ 1
ชนิดที่ 2





Deadlock Characterization

ข้อ ๗ A เงื่อนไขพร้อมกัน 4 เงื่อนไข Deadlock

Deadlock สามารถเกิดขึ้นได้หากมีเงื่อนไขสี่ประการเกิดขึ้นพร้อมกัน

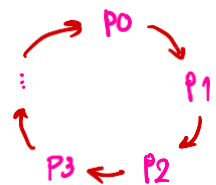
Deadlock can arise if four conditions hold simultaneously.

- การยกเว้นร่วมกัน: มีเพียงกระบวนการเดียวในแต่ละครั้งเท่านั้นที่สามารถใช้ทรัพยากรได้
- **Mutual exclusion:** only one process at a time can use a resource
- กตค้างไว้และรอ: กระบวนการที่ถือทรัพยากรอย่างน้อยหนึ่งรายการกำลังรอเพื่อรับทรัพยากรเพิ่มเติมที่กระบวนการอื่นถืออยู่
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- ไม่มีการจองล่วงหน้า: ทรัพยากรสามารถถูกปล่อยออกมาโดยสมัครใจโดยกระบวนการที่ถือครอง
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- ทรัพยากรนั้นเท่านั้น หลังจากที่มีกระบวนการนั้นได้เสร็จสิ้นภารกิจแล้ว
- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_{n-1}\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_0 .

มีข้อ: เกิด Deadlock

Process wait for resource

เกิดเพราะเป็นวงกลม



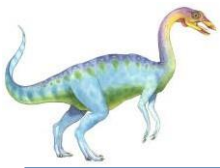
simultaneously : เกิดขึ้นในเวลาเดียวกัน

voluntarily : กระทำเสร็จแล้ว

กำลังรอทรัพยากรที่ถือโดย Pn และ P0 กำลังรอทรัพยากรที่ถือโดย P0

= willingly without being forced





Resource-Allocation Graph

กราฟการจัดสรรทรัพยากร

ชุดของจุดยอด V และชุดของขอบ E

A set of vertices V and a set of edges E .

V แบ่งออกเป็นสองประเภท:

- V is partitioned into two types:

$P = \{P_1, P_2, \dots, P_n\}$ ซึ่งเป็นเซตที่ประกอบด้วยกระบวนการทั้งหมดในระบบ

- $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system

$R = \{R_1, R_2, \dots, R_m\}$ ชุดที่ประกอบด้วยทรัพยากรทุกประเภทในระบบ

- $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system

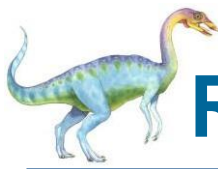
ขอคำขอ - ขอบกำกับ $P_i \rightarrow R_j$

- **request edge** – directed edge $P_i \rightarrow R_j$ P_i requests R_j

ขอการมอบหมาย - ขอบกำกับ $R_j \rightarrow P_i$

- **assignment edge** – directed edge $R_j \rightarrow P_i$ R_j is assigned (or allocated) to P_i





Resource-Allocation Graph (Cont.)



กระบวนการ

- Process



A vertex representing a process

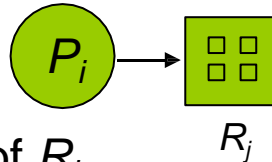
ประเภททรัพยากรที่มี 4 อินสแตนซ์

- Resource Type with 4 instances



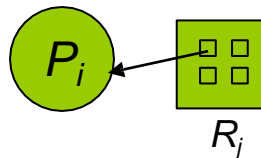
P_i ร้องขออินสแตนซ์ของ R_j

- P_i requests instance of R_j



P_i กำลังถือตัวอย่างของ R_j

- P_i is holding an instance of R_j

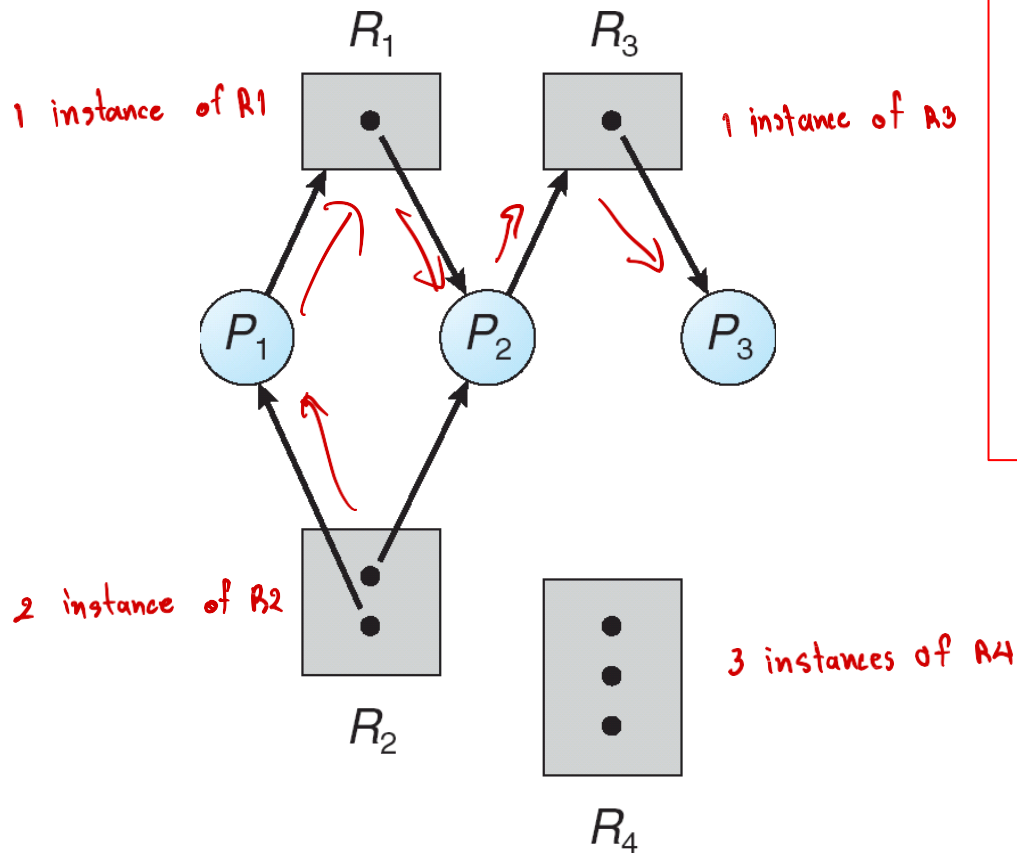




Example of a Resource Allocation Graph

There are 6 edges in the graph

E = the set request & assignment

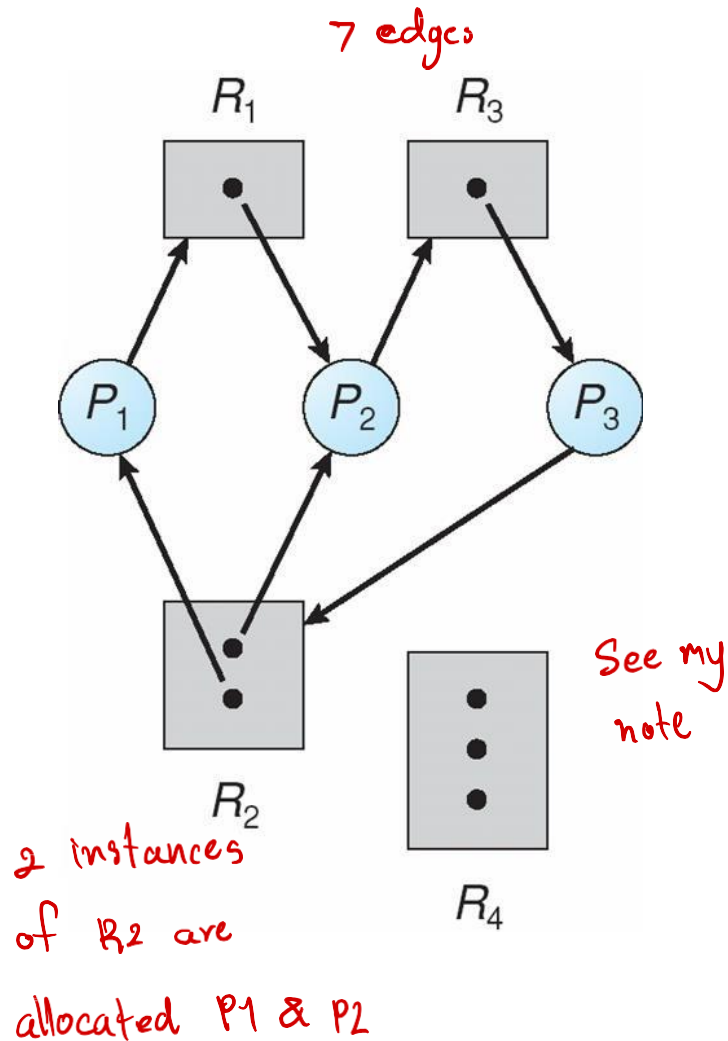


ได้กราฟ $E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3\}$





Resource Allocation Graph With A Deadlock



ได้กราฟคือ?

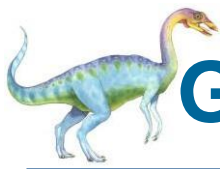
$=\{ P_1 \rightarrow R_1, P_2 \rightarrow R_3, P_3 \rightarrow R_2, R_1 \rightarrow P_2, R_2 \rightarrow P_1, R_2 \rightarrow P_2, R_3 \rightarrow P_3 \}$

กราฟนี้มีการเกิดเป็น Cycle (วงรอบ) 2 วง ได้แก่ วงที่ 1

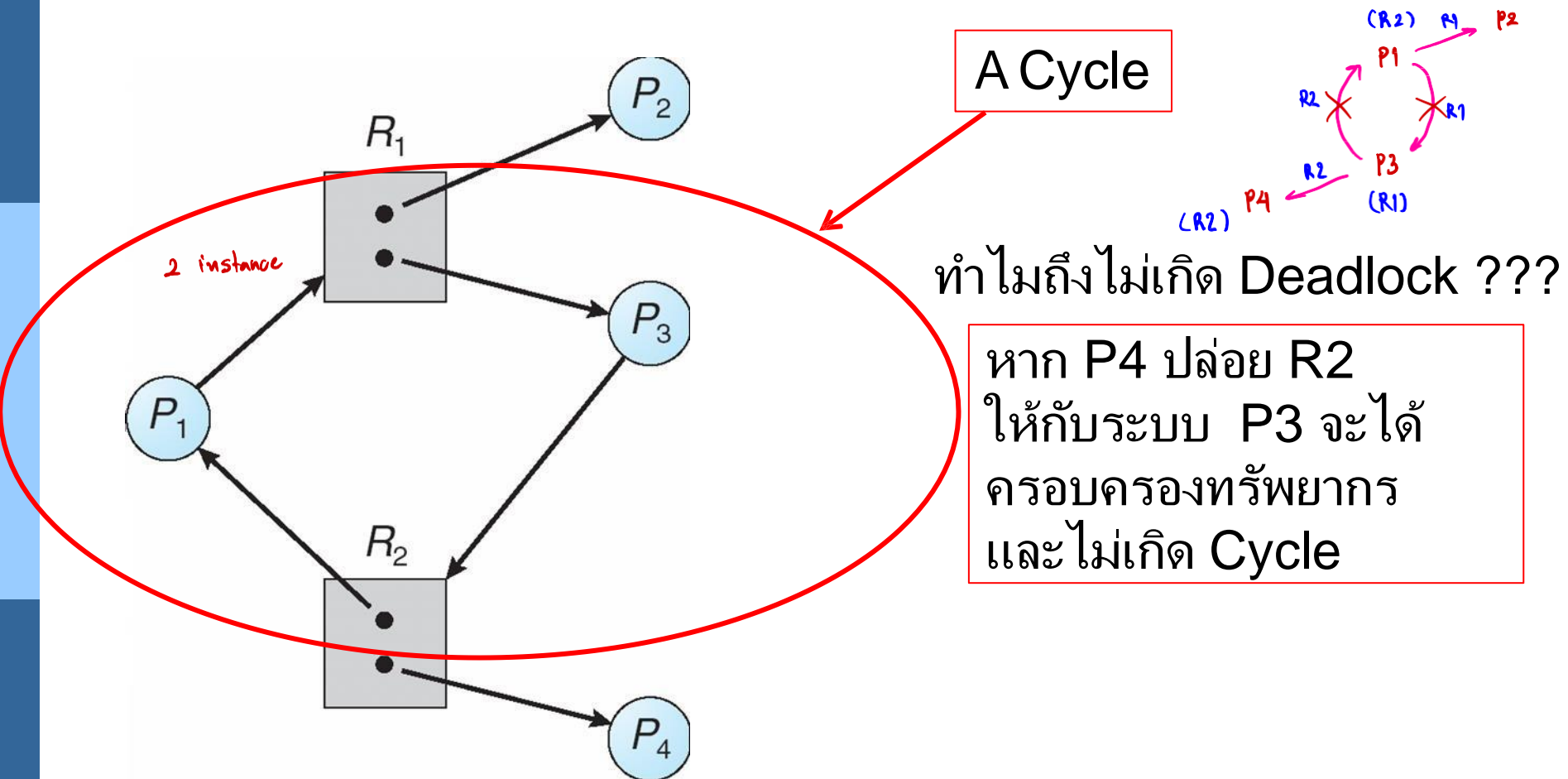
$P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

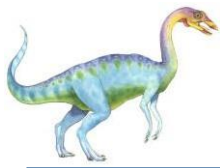
วงที่ 2 = $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$





Graph With A Cycle But No Deadlock





Basic Facts

ข้อเท็จจริงพื้นฐาน

ถ้ากราฟไม่มี cycles \Rightarrow ไม่มี deadlock

- If graph contains no cycles \Rightarrow no deadlock

ถ้ากราฟมี cycle \Rightarrow

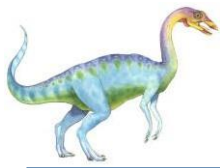
- If graph contains a cycle \Rightarrow

หากมีเพียงหนึ่งอินสแตนซ์ต่อประเภททรัพยากร เกิด deadlock

- if only one instance per resource type, then deadlock

- if several instances per resource type, possibility of deadlock หากมีหลายอินสแตนซ์ต่อประเภททรัพยากร อาจมีความเป็นไปได้ที่จะเกิด deadlock





Methods for Handling Deadlocks

ป้องกัน
หลีกเลี่ยง

The system can use
- a deadlock-prevention
schema
- or a deadlock-avoidance
schema

ตรวจสอบให้แน่ใจว่าระบบจะไม่เข้าสู่สถานะล๊อคตาย

- Ensure that the system will **never** enter a deadlock state

(กำหนดเงื่อนไขในการใช้ Resource)

อนุญาตให้ระบบเข้าสู่สถานะ deadlock จากนั้นจึงกู้คืน

- **Allow** the system to enter a **deadlock** state and then recover

(เมื่อเกิดปัญหา ตามแก้ไขทีหลัง)

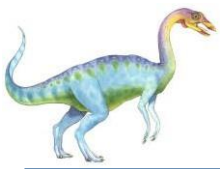
ละเว้นปัญหาและแสร้งทำเป็นว่าการหยุดชะงักไม่เคยเกิดขึ้นในระบบ ใช้โดยระบบปฏิบัติการส่วนใหญ่รวมถึง UNIX

- **Ignore** the problem and pretend that deadlocks never occur in the system; **used by most operating systems**, including UNIX

(มองข้ามปัญหา ทำเสมือนไม่มีการเกิด **Deadlock** ในระบบ

****** วิธีนี้เป็น 1 วิธีการที่ใช้ใน OS ส่วนใหญ่*****)**





Deadlock Prevention (ป้องกัน)

พิจารณาถึง การเกิด Deadlock ต้องมีเงื่อนไข ทั้ง 4 กรณีเกิดขึ้นพร้อมกัน

Restrain the ways request can be made ยับยั้งวิธีการร้องขอสามารถทำได้

การยกเว้นร่วมกัน – ไม่จำเป็นสำหรับทรัพยากรที่แบ่งปันได้ จะต้องถือเป็นทรัพยากรที่ไม่สามารถแบ่งปันได้

- ❑ **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources

ระงับและรอ - ต้องรับประกันว่าเมื่อใดก็ตามที่กระบวนการร้องขอทรัพยากร จะไม่เก็บทรัพยากรอื่นได้ไว้

- ❑ **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources

กำหนดให้กระบวนการ
ร้องขอและจัดสรร

- ❑ Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none

ทรัพยากรทั้งหมดก่อนที่
จะเริ่มดำเนินการ หรือ

อนุญาตให้กระบวนการ
ร้องขอทรัพยากรเฉพาะ

- ❑ Low resource utilization; starvation possible

เมื่อกระบวนการไม่มีเลย

(หาก Resource ว่างแต่ไม่สามารถให้ Process ถือครองได้เวลานานๆ สามารถนำ Resource มา

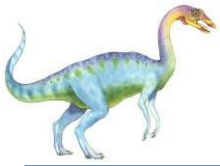
ใช้ประโยชน์เมื่อใช้เสร็จต้องรีบคืน เมื่อจะใช้ใหม่ต้อง Request

ใหม่

หากมี Process ต้องการใช้ Resource ที่ได้รับความนิยมมากๆ จะเกิด Starvation)

All or
nothing





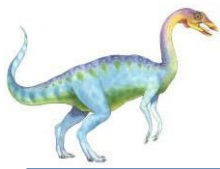
Deadlock Prevention (Cont.)



- **No Preemption** – หากกระบวนการที่ถือทรัพยากรบางส่วนร้องขอทรัพยากรอื่นที่ไม่สามารถจัดสรรได้ทันที ทรัพยากรทั้งหมดที่ถูกกักไว้ในปัจจุบันจะถูกปล่อยออกมา
 - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
ทรัพยากรที่จองไว้จะถูกเพิ่มลงในรายการทรัพยากรที่กระบวนการกำลังรออยู่
 - Preempted resources are added to the list of resources for which the process is waiting
กระบวนการจะเริ่มต้นใหม่เฉพาะเมื่อสามารถกู้คืนทรัพยากรเก่าได้ เช่นเดียวกับทรัพยากรใหม่ที่ร้องขอ
 - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – **impose a total ordering of all resource types**, and require that each process requests resources in an increasing order of enumeration
การรอแบบวงกลม – กำหนดลำดับรวมของทรัพยากรทุกประเภท และกำหนดให้แต่ละกระบวนการร้องขอทรัพยากรตามลำดับการแจกแจงที่เพิ่มขึ้น

Impose : กำหนด





Deadlock Avoidance

กำหนดให้ระบบต้องมีข้อมูลเบื้องต้นเพิ่มเติม

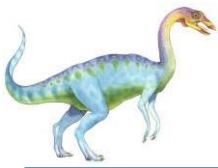
Requires that the system has some additional *a priori* information available.

using fact that are there

- Simplest and most useful model requires that each process **declare the maximum number of resources** of each type that it may need. โมเดลที่ง่ายที่สุดและมีประโยชน์มากที่สุดกำหนดให้แต่ละกระบวนการประกาศจำนวนทรัพยากรสูงสุดของแต่ละประเภทที่อาจต้องการ
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition. อัลกอริธึมการหลีกเลี่ยง deadlock จะตรวจสอบสถานะการจัดสรรทรัพยากรแบบไดนามิกเพื่อให้แน่ใจว่าจะไม่มีเงื่อนไขการรอแบบวนซ้ำ
- **Resource-allocation state** is defined by the **number of available and allocated resources**, and the **maximum demands** of the processes. สถานะการจัดสรรทรัพยากรถูกกำหนดโดยจำนวนของทรัพยากรที่มีอยู่และทรัพยากรที่จัดสรร และความต้องการสูงสุดของกระบวนการ

priori : บางส่วนก่อนหน้า





Safe State

พิจารณาทรัพยากร

เมื่อกระบวนการร้องขอทรัพยากรที่มีอยู่ ระบบจะต้องตัดสินใจว่าการจัดสรรทันทีจะทำให้ระบบอยู่ในสถานะที่ปลอดภัยหรือไม่

- When a **process** requests an available resource, system must decide if immediate allocation leaves the system in a **safe state**.

ระบบอยู่ในสถานะปลอดภัยหากมีลำดับที่ปลอดภัยของกระบวนการทั้งหมด

- System is in safe state **if there exists a safe sequence of all processes**.

- Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is safe if for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with **$j < i$** .

- If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished. *P_j will release its hold resources soon thus P_i gain them after that*

หากไม่มีความต้องการทรัพยากร P_i ในทันที P_i สามารถรอจนกว่า P_j ทั้งหมดจะเสร็จสิ้น

- When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

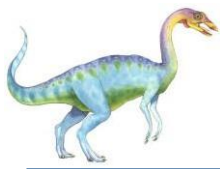
เมื่อ P_j เสร็จสิ้น P_i สามารถรับทรัพยากรที่จำเป็น ดำเนินการ คืนทรัพยากรที่จัดสรร และยุติ

- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

เมื่อ P_i ยุติลง P_{i+1} จะได้รับทรัพยากรที่จำเป็น และอื่นๆ

ลำดับ $\langle P_1, P_2, \dots, P_n \rangle$ นั้นปลอดภัยหากสำหรับแต่ละ P_i ทรัพยากรที่ P_i ยังคงสามารถร้องขอได้สามารถตอบสนองได้ด้วยทรัพยากรที่มีอยู่ในปัจจุบัน + ทรัพยากรที่ P_j ทั้งหมดถือครอง โดยที่ $j < i$





Basic Facts

ข้อเท็จจริงพื้นฐาน

หากระบบอยู่ในสถานะปลอดภัย \Rightarrow ไม่มี deadlocks

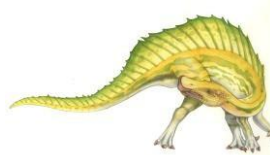
- If a system is in safe state \Rightarrow no deadlocks.

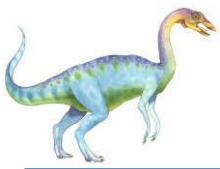
หากระบบอยู่ในสถานะไม่ปลอดภัย \Rightarrow อาจเกิดการ deadlock ได้

- If a system is in unsafe state \Rightarrow possibility of deadlock.

การหลีกเลี่ยง \Rightarrow ทำให้แน่ใจว่าระบบจะไม่เข้าสู่สถานะที่ไม่ปลอดภัย

- Avoidance \Rightarrow ensure that a system will never enter an unsafe state.

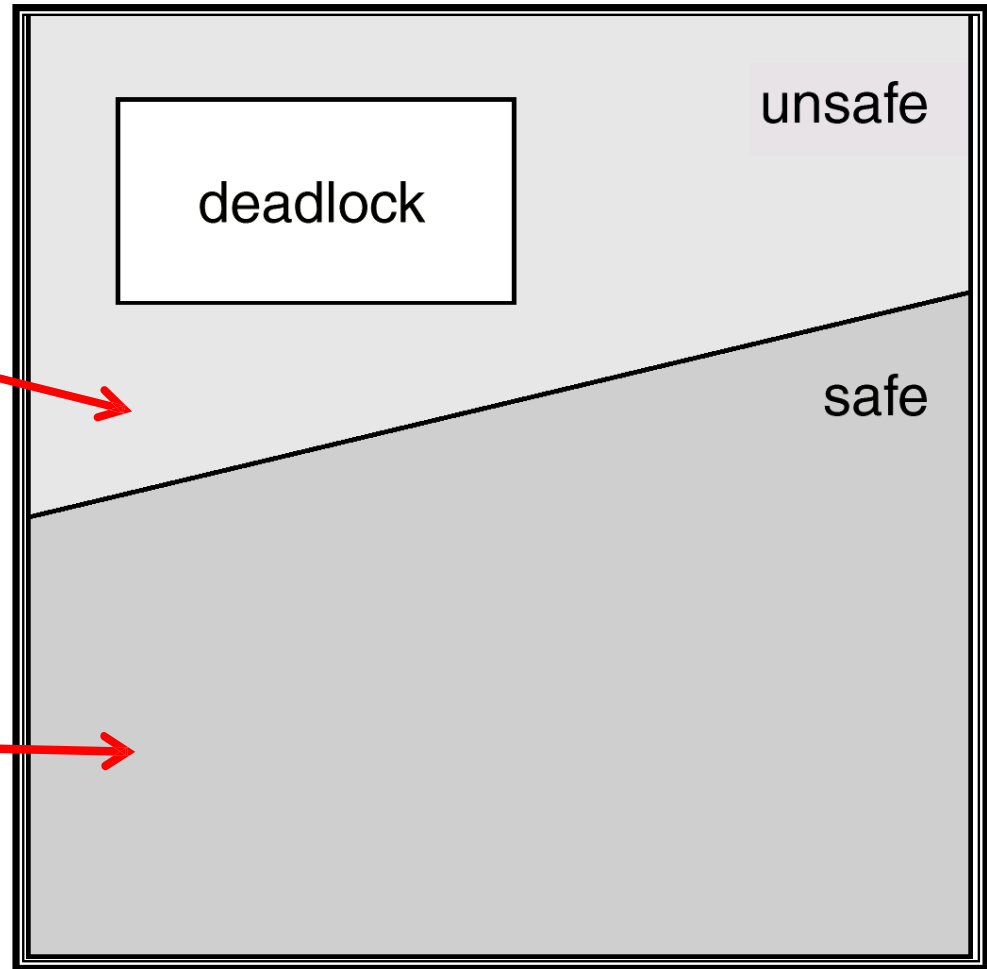


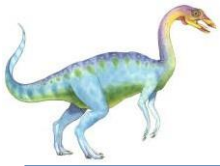


Safe, Unsafe , Deadlock State

อยู่ในสถานะ unsafe อาจจะ
เกิด Deadlock ได้

อยู่ในสถานะ safe ไม่เกิด
Deadlock แน่นนอน





Avoidance algorithms

อินสแตนซ์เดียวของประเภททรัพยากร

- Single instance of a resource type

ใช้กราฟการจัดสรรทรัพยากร

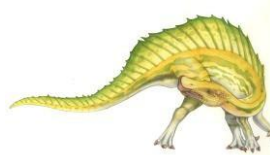
- Use a resource-allocation graph *กราฟทรัพยากร*

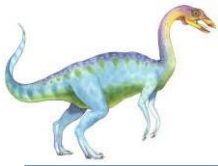
ประเภททรัพยากรหลายอินสแตนซ์

- Multiple instances of a resource type

ใช้ banker's algorithm

- Use the banker's algorithm





Resource-Allocation Graph Scheme

For deadlock avoidance with ONE instance of each resource type

Claim edge $P_j \rightarrow R_j$ ระบุว่ากระบวนการ P_j อาจร้องขอทรัพยากร R_j ; แสดงด้วยเส้นประ

- Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ; represented by a dashed line

Claim edge แปลงเป็น request edge เมื่อกระบวนการร้องขอทรัพยากร

- Claim edge converts to request edge when a process requests a resource

- Request edge converted to an assignment edge when the resource is allocated to the process

เมื่อทรัพยากรถูกปล่อยโดยกระบวนการ ขอบการมอบหมายจะแปลงกลับเป็นขอบการเคลม

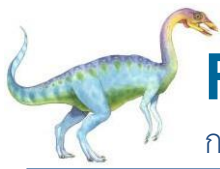
- When a resource is released by a process, assignment edge reconverts to a claim edge

ทรัพยากรจะต้องได้รับการอ้างสิทธิ์เป็นนิรันดร์ในระบบ

- Resources must be claimed *a priori* in the system

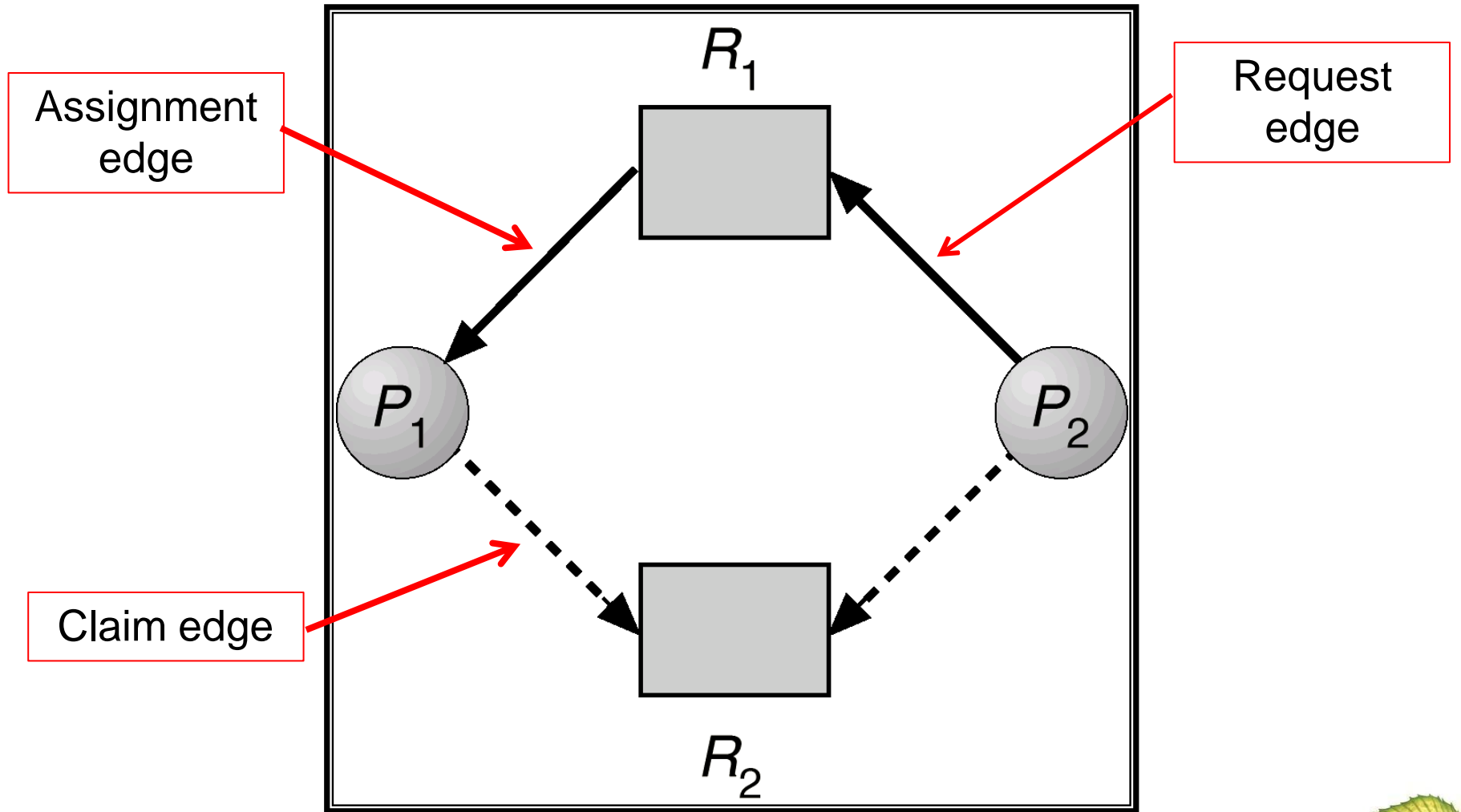
Claim edge : เส้นความต้องการ (แสดงโดยใช้เส้นประ)

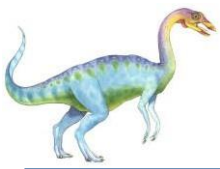




Resource-Allocation Graph For Deadlock Avoidance

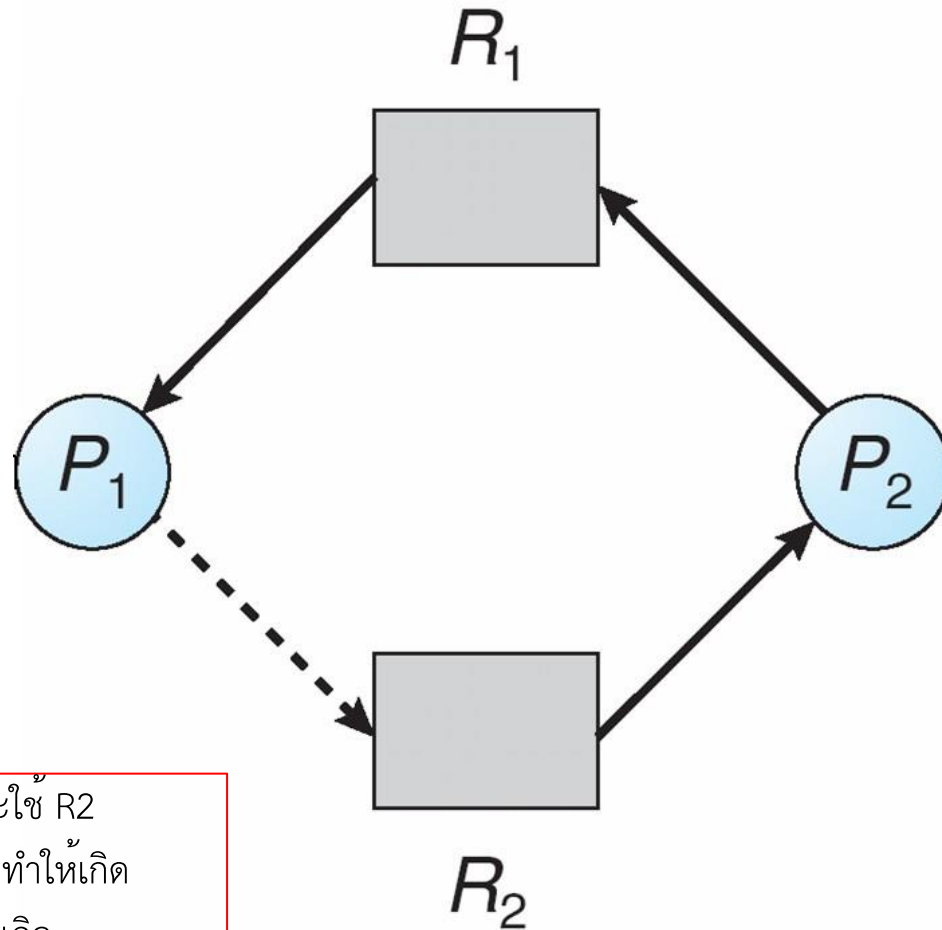
กราฟการจัดสรรทรัพยากรเพื่อหลีกเลี่ยง Deadlock





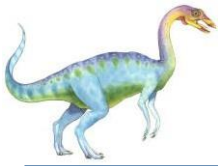
Unsafe State In Resource-Allocation Graph

สถานะที่ไม่ปลอดภัยในกราฟการจัดสรรทรัพยากร



หากมีความต้องการของ P_1 จะใช้ R_2
ระบบจะไม่อนุญาต เพราะอาจทำให้เกิด
Deadlock (ดูจากลักษณะอาจเกิด
วงรอบ ขึ้นได้)





Resource-Allocation Graph Algorithm

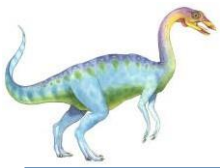
สมมติว่ากระบวนการ P_i ร้องขอทรัพยากร R_j

- Suppose that process P_i requests a resource R_j
- The request can be granted **only if** converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

สามารถให้คำขอได้เฉพาะในกรณีที่การแปลงขอบคำขอเป็นขอบที่ได้รับมอบหมายไม่ส่งผลให้เกิดการก่อตัวของวงจรในกราฟการจัดสรรทรัพยากร

ถ้าไม่ปลอดภัย ก็ไม่
อนุญาตให้





Banker's Algorithm

For deadlock avoidance with MULTIPLE instance of each resource type

- Multiple instances. หลายกรณี

แต่ละกระบวนการจะต้องเรียกร้องสิทธิ์การใช้งานสูงสุด

- **Each process must a priori claim maximum use.**

เมื่อกระบวนการร้องขอทรัพยากร กระบวนการอาจต้องรอ

- When a process requests a resource it may have to wait.

เมื่อกระบวนการได้รับทรัพยากรทั้งหมด กระบวนการจะต้องส่งคืนทรัพยากรเหล่านั้นภายในระยะเวลาจำกัด

- When a process gets all its resources it must return them in a finite amount of time.





Data Structures for the Banker's Algorithm

โครงสร้างข้อมูลสำหรับ Banker's Algorithm



ให้ n = จำนวนกระบวนการ และ m = จำนวนประเภททรัพยากร

Let **n** = number of processes, and **m** = number of resources types.

มีจำหน่าย: เวกเตอร์ความยาว m หากมี $[j] = k$ มีอินสแตนซ์ประเภททรัพยากร R_j จำนวน k รายการ

- **Available:** Vector of length m . If available $[j] = k$, there are k instances of resource type R_j available. (เก็บ Resource ที่ว่าง)

สูงสุด: เมทริกซ์ $n \times m$ หาก $\text{Max}[i, j] = k$ ดังนั้นกระบวนการ P_i อาจร้องขอได้สูงสุด k อินสแตนซ์ของทรัพยากรประเภท R_j

- **Max:** $n \times m$ matrix. If $\text{Max}[i, j] = k$, then process P_i may request at most k instances of resource type R_j . (เก็บจน. สูงสุดของ Resource ที่กระบวนการแต่ละตัวต้องการใช้)

การจัดสรร: เมทริกซ์ $n \times m$ หาก $\text{Allocation}[i, j] = k$ ดังนั้น P_i จะได้รับการจัดสรร k อินสแตนซ์ของ R_j

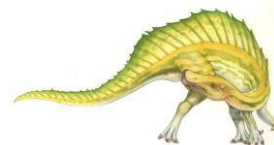
- **Allocation:** $n \times m$ matrix. If $\text{Allocation}[i, j] = k$ then P_i is currently allocated k instances of R_j . (เก็บจน. Resource ที่แต่ละกระบวนการครอบครองอยู่)

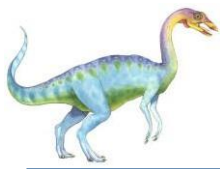
- **Need:** $n \times m$ matrix. If $\text{Need}[i, j] = k$, then P_i may need k more instances of R_j to complete its task. (เก็บจน. Resource ที่เหลืออยู่ที่แต่ละกระบวนการยังคงต้องการใช้เพื่อทำงานให้เสร็จสมบูรณ์)

ต้องการ: เมทริกซ์ขนาด $n \times m$ ถ้า $\text{Need}[i, j] = k$ ดังนั้น P_i อาจต้องใช้ R_j อีก k ตัวเพื่อทำงานให้สำเร็จ

$$\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j].$$

(ที่ขอใช้ Resource สูงสุด – ที่ได้ Resource ครอบครองแล้ว)





Safety Algorithm

ให้ *Work* และ *Finish* เป็นเวกเตอร์ที่มีความยาว m และ n ตามลำดับ

1. Let *Work* and *Finish* be **vectors of length m and n** , respectively.
Initialize:

Work = *Available* the number of resource available for P_i to use of its work

Finish [i] = false for $i = 1, 2, \dots, n$. for all processes P_i , initialize
the *Finish* status to false

2. Find an i such that both:

(a) *Finish* [i] = false

(b) $Need_i \leq Work$

ตรวจสอบเงื่อนไข

If (*Finish*[i]==false AND $Need_i \leq work$)

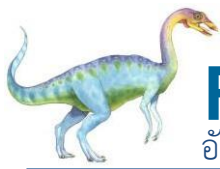
If no such i exists, go to step 4.

3. $Work = Work + Allocation_i$ After P_i finished its task, all allocated resource must return
Finish[i] = true to the system
go to step 2.

4. If ***Finish* [i] == true for all i** , then the system is in a **safe state**.

Resource มีสถานะที่ available เพื่อรอในการทำงานครั้งต่อไป





Resource-Request Algorithm for Process P_i

อัลกอริทึมการร้องขอทรัพยากรสำหรับกระบวนการ P_i

Request = ขอเวกเตอร์สำหรับกระบวนการ P_i ถ้า $Request_i[j] = k$ แล้วกระบวนการ P_i ต้องการอินสแตนซ์ของทรัพยากร

ประเภท R_j จำนวน k รายการ

หาก $Request_i \leq$ **Request** = request vector for process P_i . If $Request_i[j] = k$ then

Need_i ไปที่ขั้นตอนที่ 2 **process P_i wants k instances of resource type R_j .**

มีฉะนั้น จะเกิด

ข้อผิดพลาดเงื่อนไข

เนื่องจากกระบวนการ

เกินการเรียกร้องสูงสุด

→ 1. If $Request_i \leq Need_i$ go to step 2. **Otherwise**, raise error condition, since **process has exceeded its maximum claim.**

→ 2. If $Request_i \leq Available$, go to step 3. **Otherwise P_i must wait**, since **resources are not available.**

หาก $Request_i \leq$

Available ให้ไปที่

ขั้นตอนที่ 3 มีฉะนั้น

P_i จะต้องรอเนื่องจาก

ทรัพยากรไม่พร้อมใช้

งาน

3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$Available = Available - Request_i$; Fewer resources are left in the system

$Allocation_i = Allocation_i + Request_i$; More resources allocated to P_i

$Need_i = Need_i - Request_i$;

หากปลอดภัย \Rightarrow ทรัพยากรจะถูกจัดสรรให้กับ P_i

• If safe \Rightarrow the resources are allocated to P_i .

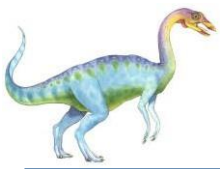
• If unsafe $\Rightarrow P_i$ must wait, and the **old** resource-allocation state is restored

หากไม่ปลอดภัย $\Rightarrow P_i$ ต้องรอ และสถานะการจัดสรรทรัพยากรเก่าจะถูกกู้คืน

exceed: มากเกิน

pretend: การอ้างถึง





Example of Banker's Algorithm

5 กระบวนการ P_0 ถึง P_4 ; ทรัพยากร 3 ประเภท A (10 อินสแตนซ์), B (5 อินสแตนซ์) และ C (7 อินสแตนซ์)

□ 5 processes P_0 through P_4 ; 3 resource types

A (10 instances), B (5 instances), and C (7 instances).

ภาพรวม ณ เวลา T_0 :

□ Snapshot at time T_0 :

รวมไว้ทั้งหมด

	<u>Allocation</u>	<u>Max</u>
	A B C	A B C
P_0	0 1 0	7 5 3
P_1	2 0 0	3 2 2
P_2	3 0 2	9 0 2
P_3	2 1 1	2 2 2
P_4	0 0 2	4 3 3

ที่เหลืออยู่ในระบบ

<u>Available</u>
A B C
3 3 2

System j of R_j is composed of

1 System-A = 10 (instance)

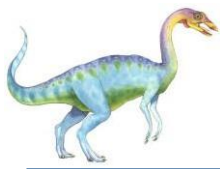
2 System-B = 5

3 System-C = 7

1 bug P

see my note





Example (Cont.)

need more form

- The content of the matrix. **Need** is defined to be **Max – Allocation**.
เนื้อหาของเมทริกซ์ ความต้องการถูกกำหนดให้เป็น Max – Allocation

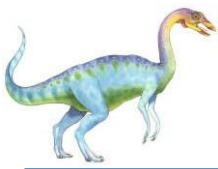
	<u>Need</u>			
	A	B	C	
	7	5	3	– 0 1 0
P_0	7	4	3	
P_1	1	2	2	
P_2	6	0	0	
P_3	0	1	1	
P_4	4	3	1	– 4 3 3 – 0 0 2

Found at least 1 safe sequence'

- The system is in a **safe state** since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria.

ระบบอยู่ในสถานะปลอดภัยเนื่องจากลำดับ $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ เป็นไปตามเกณฑ์ความปลอดภัย






Example P_1 Request (1,0,2)



ตรวจสอบว่า Request \leq Available (นั่นคือ $(1,0,2) \leq (3,3,2) \Rightarrow \text{true}$)

- Check that Request \leq Available (that is $(1,0,2) \leq (3,3,2) \Rightarrow \text{true}$).

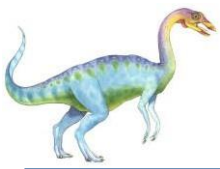
	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
$2\ 0\ 0 + 1\ 0\ 2$	P_0 0 1 0 ✓	7 4 3	2 3 0 ← $3\ 3\ 2 - 1\ 0\ 2$
	P_1 3 0 2	0 2 0 ←	
	P_2 3 0 2	6 0 0	?? 
	P_3 2 1 1	0 1 1	
	P_4 0 0 2	4 3 1	

$122 - 102$

การดำเนินการอัลกอริธึมความปลอดภัยแสดงให้เห็นว่าลำดับ $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ เป็นไปตามข้อกำหนดด้านความปลอดภัย

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.
- Can request for $(3,3,0)$ by P_4 be granted? สามารถขอ $(3,3,0)$ โดย P_4 ได้หรือไม่?
- Can request for $(0,2,0)$ by P_0 be granted? สามารถขอ $(0,2,0)$ โดย P_0 ได้หรือไม่?





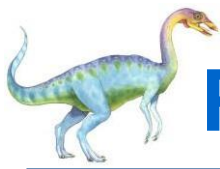
Practice: Banker's Algorithm

□ Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- จาก Snapshot T_0 จงแสดงวิธีทำเพื่อที่จะตรวจสอบว่า ระบบจะอยู่ในสถานะที่ปลอดภัย (safe state) หรือไม่ ?? เมื่อมีลำดับของโพรเซส คือ P_0, P_1, P_2, P_3, P_4





Practice: Banker's Algorithm (Solution)

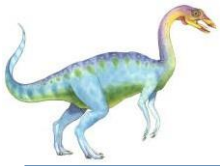
- ค้นหา Need ของแต่ละ Process ก่อน แล้วจึงมาดำเนินการตาม Banker Algorithm

Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	3 3 2
P_1	2 0 0	3 2 2	1 2 2	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

ระบบจะอยู่ในสถานะที่ปลอดภัย (safe state) หรือไม่ ?? เมื่อมีลำดับของโปรเซส คือ P_3, P_2, P_1, P_4, P_0





Deadlock Detection

หากไม่มีการ Protection และ Avoidance

อนุญาตให้ระบบเข้าสู่สถานะ deadlock

- **Allow system to enter deadlock state**

อัลกอริธึมการตรวจจับ

- **Detection algorithm** *พหุคูณ*

โครงการฟื้นฟู

- **Recovery scheme** *ทฤษฎี*

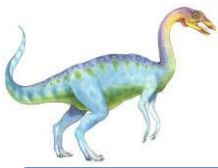




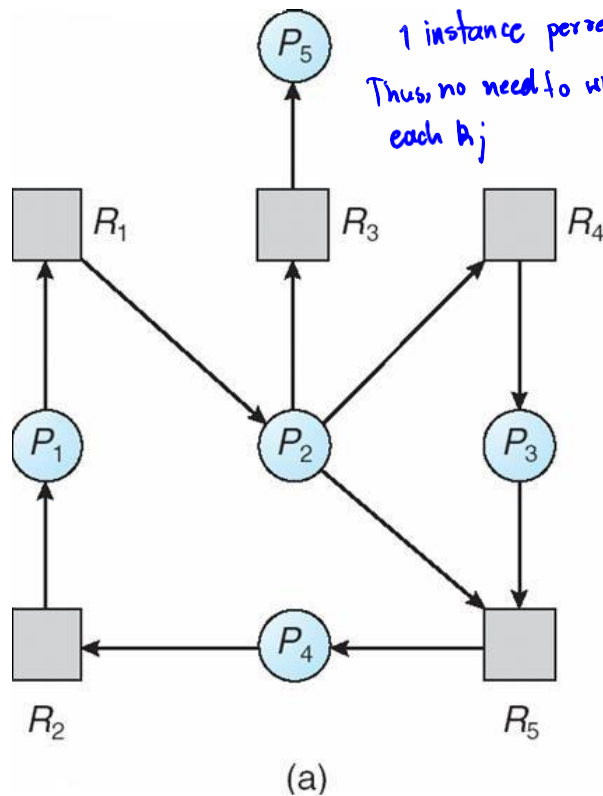
Single Instance of Each Resource Type

- Maintain **wait-for graph**
 - **Nodes are processes.**
 - $P_i \rightarrow P_j$ ถ้า P_i กำลังรอ P_j
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
 - Periodically invoke an **algorithm that searches for a cycle in the graph.**
 - An algorithm to detect a cycle in a graph requires an **order of n^2** operations, where n is the number of vertices in the graph.
- เปลี่ยนเป็น กราฟการรอคอยทรัพยากร (wait-for graph)
- โหนดเป็นกระบวนการ
- จะไม่มีโหนดของ Resource แล้ว
- ตรวจสอบเป็นระยะ เรียกใช้อัลกอริทึมที่ค้นหาวงจรในกราฟเป็นระยะ
- $n = \text{จำนวนของกราฟ} / \text{โหนด}$
- อัลกอริทึมในการตรวจจับวงจรในกราฟจำเป็นต้องมีลำดับการดำเนินการ n^2 โดยที่ n คือจำนวนจุดยอดในกราฟ



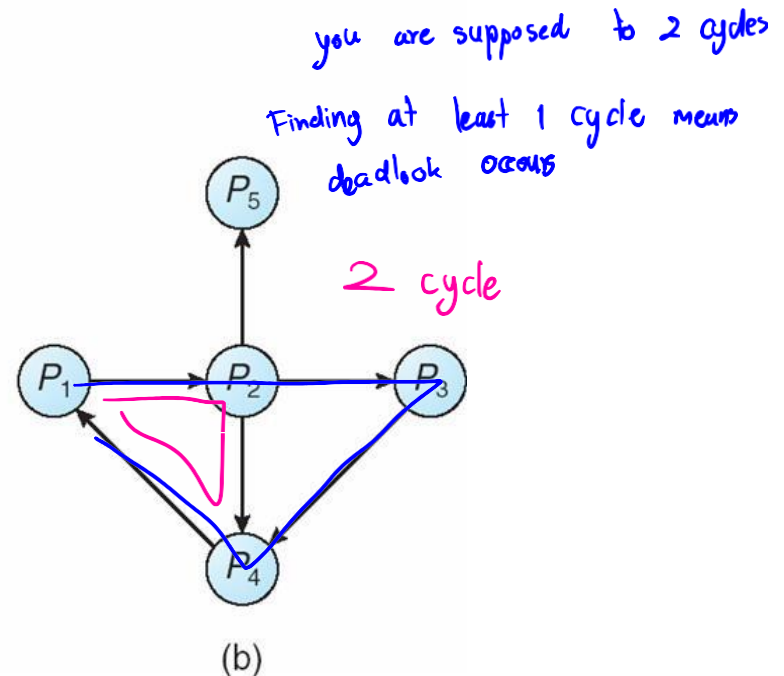


Resource-Allocation Graph and Wait-for Graph



1 instance per resource type
Thus, no need to write a black dot in
each R_j

Resource-Allocation Graph

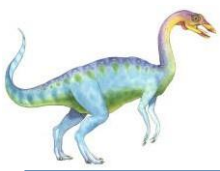


you are supposed to 2 cycles
Finding at least 1 cycle means
deadlock occurs

2 cycle

Corresponding wait-for graph





Several Instances of a Resource Type

m — # of resource types

n — # of processes

พร้อมใช้งาน: เวกเตอร์ความยาว m ระบุจำนวนทรัพยากรที่มีอยู่ของแต่ละประเภท

- **Available:** A vector of length m indicates the number of **available resources** of each type.

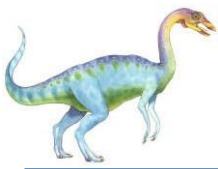
การจัดสรร: เมทริกซ์ขนาด $n \times m$ กำหนดจำนวนทรัพยากรของแต่ละประเภทที่ได้รับการจัดสรร

- **Allocation:** An $n \times m$ matrix defines the number of resources of each type **currently allocated** to each process.

- **Request:** An $n \times m$ matrix indicates the **current request** of each process. If $Request[ij] = k$, then process P_i is requesting k more instances of resource type R_j .

คำขอ: เมทริกซ์ขนาด $n \times m$ บ่งชี้ถึงคำขอปัจจุบันของแต่ละกระบวนการ หากคำขอ $[ij] = k$ แสดงว่ากระบวนการ P_i กำลังขออินสแตนซ์ประเภททรัพยากร R_j เพิ่มเติมนีก k รายการ





Detection Algorithm

ปล่อยให้ Work และ Finish เป็นเวกเตอร์ที่มีความยาว m และ n ตามลำดับ

1. Let Work and Finish be vectors of length m and n , respectively Initialize:

(a) Work = Available

(b) For $i = 1, 2, \dots, n$, if Allocation_i $\neq 0$, then
Finish[i] = false; otherwise, Finish[i] = true.

ของ Banker's Algo
For $i=1, 2 \dots n$ then
Finish[i]=false;

2. Find an index i such that both:

(a) Finish[i] == false

(b) Request_i \leq Work

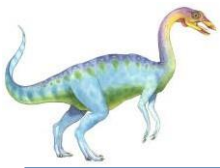
ตรวจสอบเงื่อนไข
If (Finish[i]==false AND ~~Need_i~~ \leq work)

P_i does not finish yet

P_i 's additional
request \leq available

If no such i exists, go to step 4.





Detection Algorithm (Cont.)



3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2

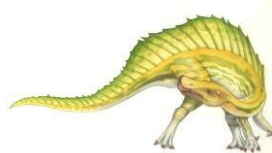
After P_i finished, all allocated resource must be returned to the system

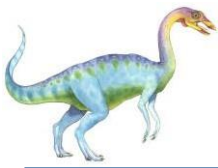
4. If $Finish[i] == false$, for some i , $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked

หาก $Finish[i] == false$ สำหรับ i บางตัว $1 \leq i \leq n$ แสดงว่าระบบอยู่ในสถานะ deadlock ยิ่งไปกว่านั้น หาก $Finish[i] == false$ แสดงว่า P_i เกิด deadlocked

Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state

อัลกอริทึมต้องมีลำดับการดำเนินการ $O(m \times n^2)$ เพื่อตรวจสอบว่าระบบอยู่ในสถานะล๊อคตายตัวหรือไม่





Example of Detection Algorithm

ห้ากระบวนการ P_0 ถึง P_4 ; ทรัพยากรสามประเภท A (7 อินสแตนซ์), B (2 อินสแตนซ์) และ C (6 อินสแตนซ์)

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances)

ภาพรวม ณ เวลา T_0 :

- Snapshot at time T_0 :

จัดภาพ
งา

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Similar to Need_i in Banker.

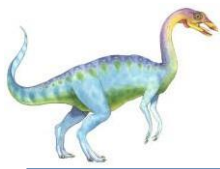
see my not

ลำดับ $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ จะส่งผลให้ $Finish[i] = \text{true}$ สำหรับ i ทั้งหมด

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i

thus, the system detects
detects delay in a day





Example (Cont.)

P2 ขออินสแตนซ์เพิ่มเติมประเภท C

- P_2 requests an additional instance of type C

สรุป 4 ฟังก์ชัน

Request

A B C

P_0 0 0 0

P_1 2 0 1

P_2 0 0 1

P_3 1 0 0

P_4 0 0 2

$(7, 2, 6) - (7, 2, 6)$
 $(0, 0, 0)$

สถานะของระบบ?

- State of system?

สามารถเรียกคืนทรัพยากรที่ถือครองโดยกระบวนการ P_0 แต่มีทรัพยากรไม่เพียงพอที่จะตอบสนองกระบวนการอื่น ๆ คำขอ

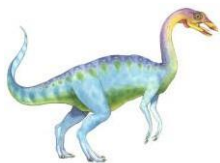
- Can reclaim resources held by process P_0 , **but insufficient resources to fulfill other processes**; requests

- **Deadlock exists**, consisting of processes P_1 , P_2 , P_3 , and P_4

Deadlock มีอยู่ ประกอบด้วยกระบวนการ P_1 , P_2 , P_3 และ P_4

insufficient : ไม่เพียงพอ





Practice: Deadlock Detection

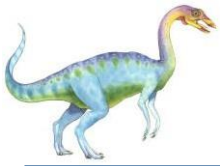
□ Snapshot at time T_0 :

	<u>Allocation</u>	<u>Request</u>	<u>Available</u>	
	A B C	A B C	A B C	
P_0	0 1 0	0 0 0	0 0 0 + 0 1 0	1
P_1	2 0 0	2 0 2	0 1 0 + 2 0 2	2
P_2	3 0 3	0 1 0 (0,0,0)	3 1 3 + 2 1 1	4
P_3	2 1 1	<div style="border: 2px solid purple; padding: 2px;">1 2 3</div> (1,0,0) → (1,2,3)	5 2 4 + 0,0,2	5
P_4	0 0 2	0 0 2		
	7 2 6			

Not addition request
but just change P_3 's request

□ จาก Snapshot T_0 เมื่อ P_3 ต้องการ (Request) ทรัพยากรชนิดต่างๆ (Resource type) A = 1, B=2, และ C =3 เมื่อลำดับการทำงานของโพรเซส คือ P_0 , P_2 , P_3 , P_1 และ P_4 ให้นักศึกษาแสดงวิธีทำในการตรวจสอบสถานะของระบบว่า จะมีสถานะเป็นอย่างไร??





Detection-Algorithm Usage

การเรียกใช้เมื่อใดและบ่อยเพียงใดขึ้นอยู่กับ:

- When, and how often, to invoke depends on:

มีโอกาสเกิด deadlock บ่อยแค่ไหน?

- **How often** a deadlock is likely to occur?

จะต้องย้อนกลับกี่กระบวนการ?

- **How many processes** will **need to be rolled back**?

หนึ่งอันสำหรับแต่ละรอบที่ไม่ต่อเนื่องกัน

- ▶ one for each disjoint cycle

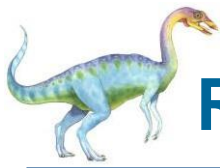
one-by-one unit each

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock

หากอัลกอริทึมการตรวจจับถูกเรียกใช้โดยพลการ อาจมีหลาย cycles ในกราฟทรัพยากร ดังนั้นเราจึงไม่สามารถบอกได้ว่ากระบวนการใดจากหลายกระบวนการที่ deadlocked “สาเหตุ” ของ deadlocked

arbitrarily : ไม่มีกฎเกณฑ์





Recovery from Deadlock: Process Termination

ยกเลิกกระบวนการที่ deadlocked ทั้งหมด

(ยกเลิกกระบวนการที่เกิด deadlock)

- Abort all deadlocked processes

ยกเลิกทีละกระบวนการจนกว่าวงจรการ deadlock จะหมดไป

- Abort one process at a time until the deadlock cycle is eliminated

เราควรเลือกที่จะยกเลิกตามลำดับไหน *อันดับไหน abort?*

- In which order should we choose to abort?

ลำดับความสำคัญของกระบวนการ

- Priority of the process *ตามลำดับความสำคัญ*

กระบวนการคำนวณนานเท่าใด และใช้เวลาไม่นานเท่าใดจึงจะเสร็จสิ้น

- How long process has computed, and how much longer to completion *process มีงานในมือทำงานนานขนาดไหน*

ทรัพยากรที่กระบวนการใช้

- Resources the process has used *ทรัพยากรที่ใช้แล้ว*

กระบวนการทรัพยากรต้องเสร็จสิ้น

- Resources process needs to complete *จะเหลือ*

จะต้องยุติกระบวนการกี่กระบวนการ

- How many processes will need to be terminated

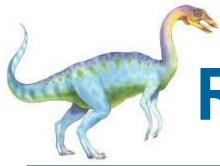
กระบวนการเป็นแบบโต้ตอบหรือเป็นชุด?

- Is process interactive or batch?

มีกระบวนการแบบโต้ตอบหรือไม่

จำนวนที่ processes batch processes are likely to be terminated first





Recovery from Deadlock: Resource Preemption

ข้อได้เปรียบจากการดักเหวี่ยง

การเลือกใช้ จะต้องพิจารณาผลที่จะเกิดขึ้น 3 ข้อ ดังนี้

การเลือกเหยื่อ – ลดต้นทุนให้เหลือน้อยที่สุด

□ **Selecting a victim** – minimize cost

ย้อนกลับ – กลับสู่สถานะที่ปลอดภัย รีستาร์ทกระบวนการสำหรับสถานะนั้น

□ **Rollback** – return to some safe state, restart process for that state

ความอดอยาก – กระบวนการเดียวกันอาจถูกเลือกให้เป็นเหยื่อเสมอ โดยรวมจำนวนการย้อนกลับไว้ในปัจจัยด้านต้นทุน

□ **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

We must ensure that a process can be picked as a victim only a (small) finite number of times.
The most common SOLUTION is to include
the number of rollback in the factor.

Victim : ผู้รับเคราะห์ (process)



End of Chapter 6

