

Deadlock

CS450 : Saelee

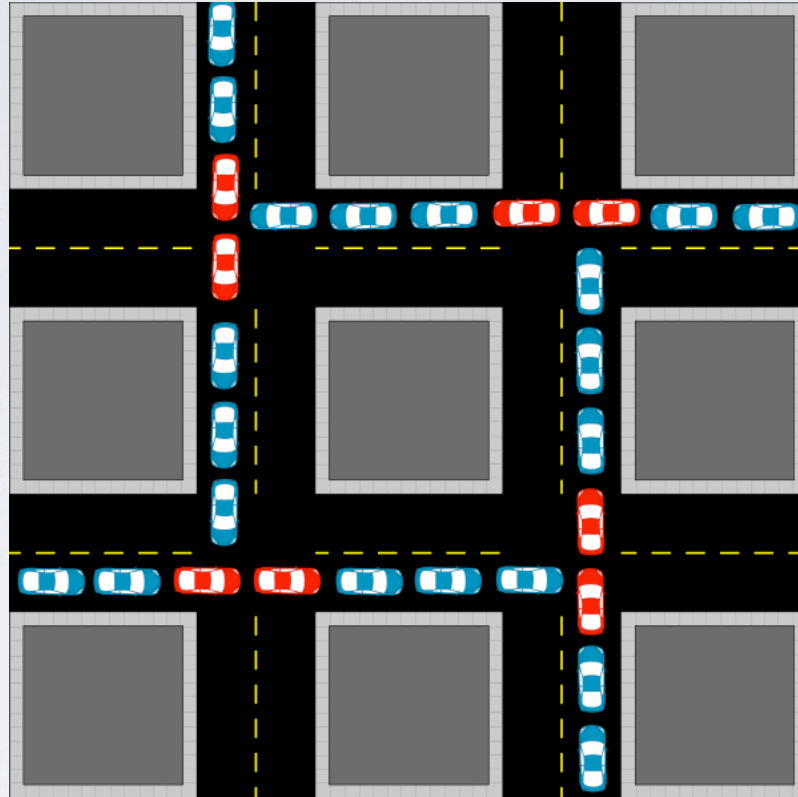
“

deadlock |'ded,läk|

noun

- 1 [in sing.] a situation, typically one involving opposing parties, in which no progress can be made : *an attempt to break the deadlock.*

New Oxford American Dictionary



Traffic Gridlock

```
mtx_A.lock()  
mtx_B.lock()  
  
# critical section  
  
mtx_B.unlock()  
mtx_A.unlock()
```

```
mtx_B.lock()  
mtx_A.lock()  
  
# critical section  
  
mtx_B.unlock()  
mtx_A.unlock()
```

Software Gridlock

shared resource(s)

necessary conditions

1. mutual exclusion

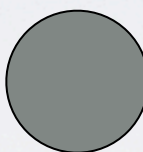
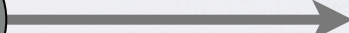
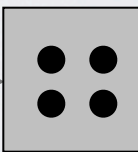
2. hold and wait

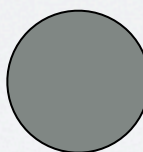
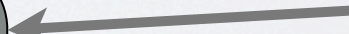
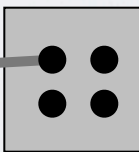
3. no preemption

4. circular wait

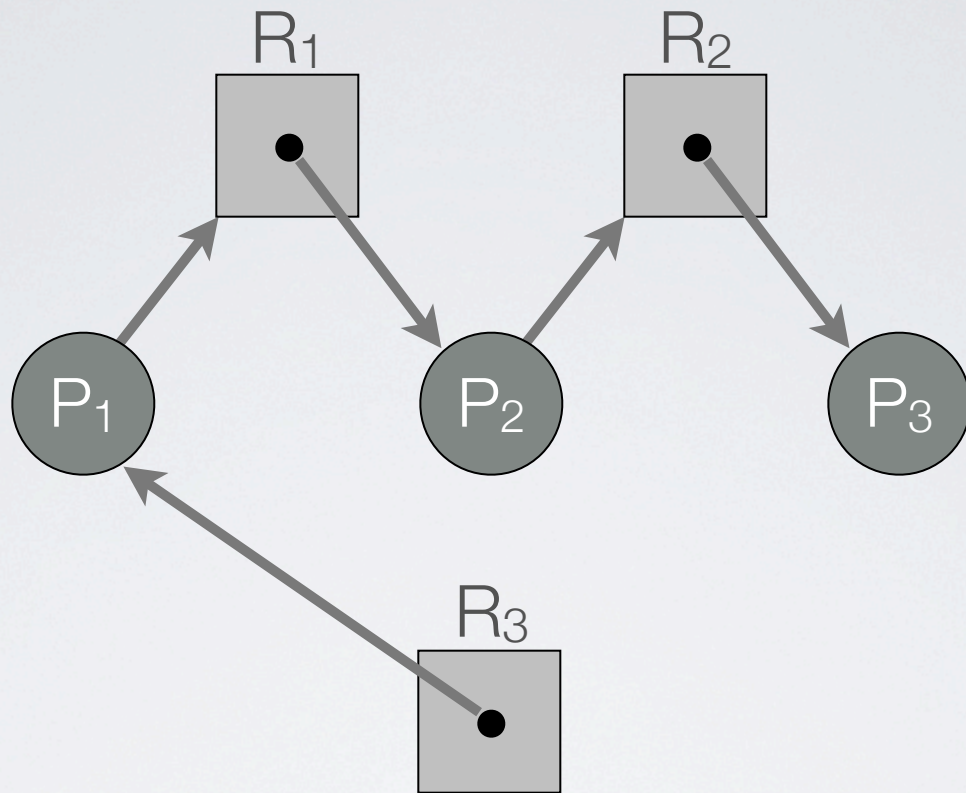
Process : 

Resource : 

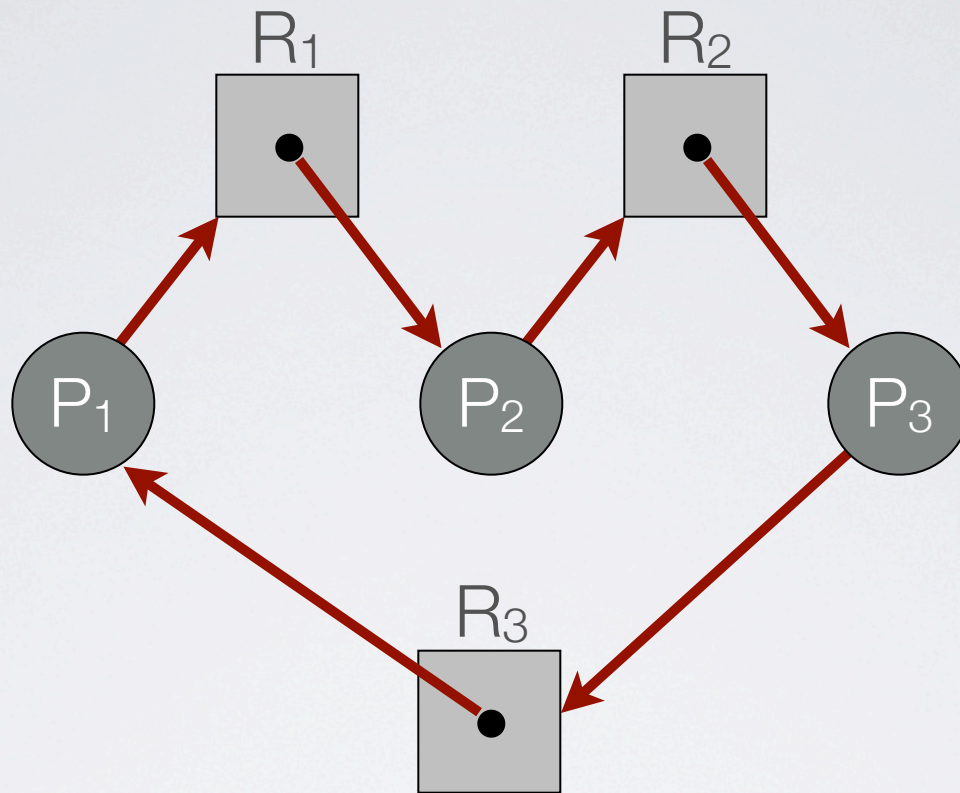
Request :   

Allocation :   

Resource Allocation Graphs

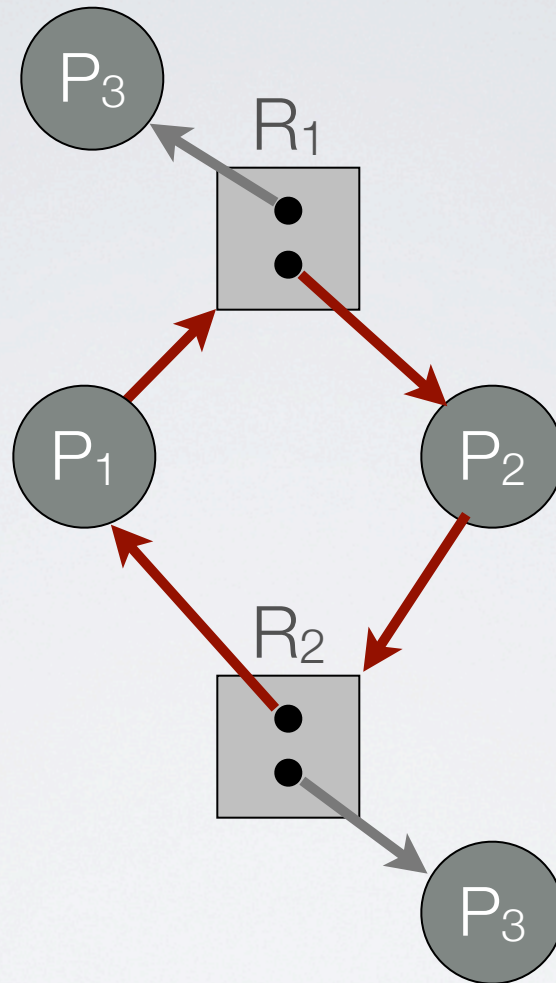


Safe



Deadlock

single instance resources:
cycle → deadlock



Cycle without Deadlock

Approaches

1. Ignore it
2. Avoidance
3. Detection & Recovery

(2)

(a)

remove necessary conditions

1. mutual exclusion

resource spooling

limited application

2. hold and wait

all-or-nothing

starvation
low utilization

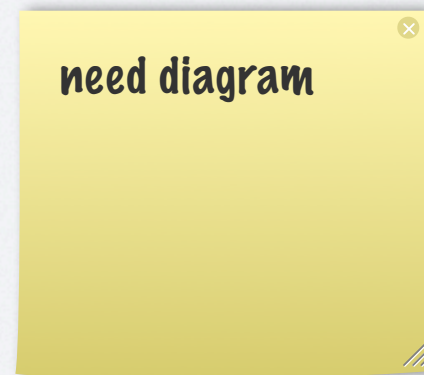
3. no preemption

allow preemption

mechanism?
livelock

4. circular wait

order resource types
request in order

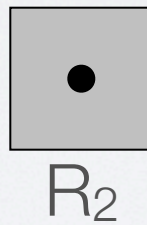
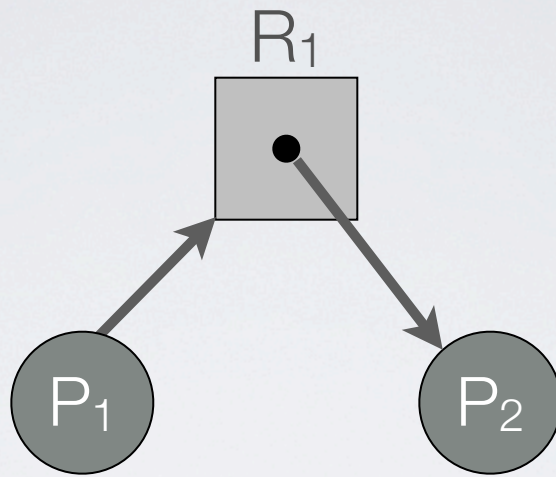


impractical

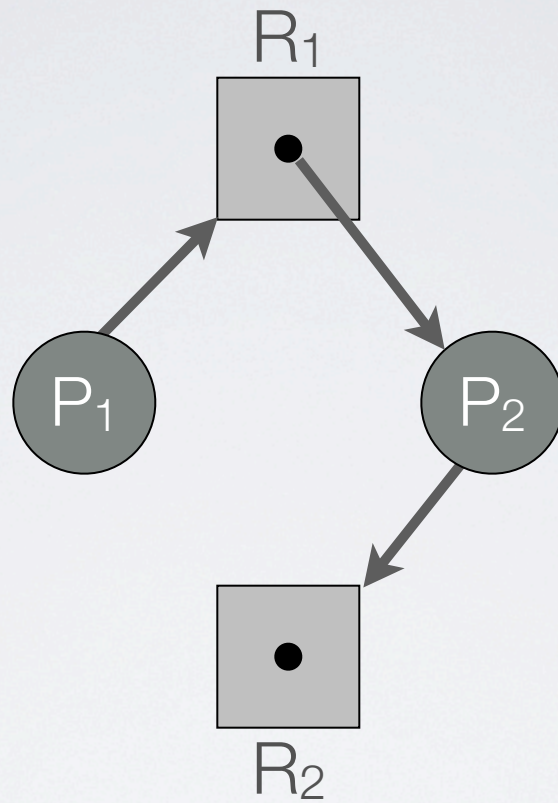
(b)

avoidance algorithm

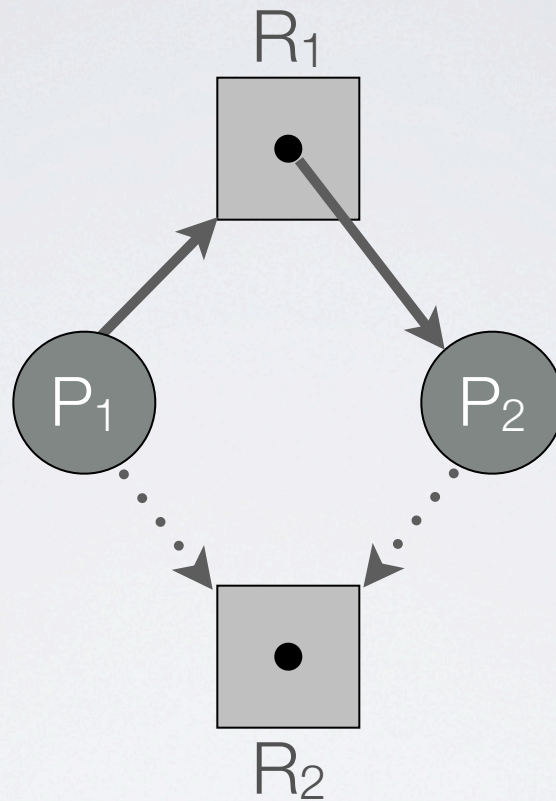
avoid circular wait
maintain “safe” state
no possibility of deadlock



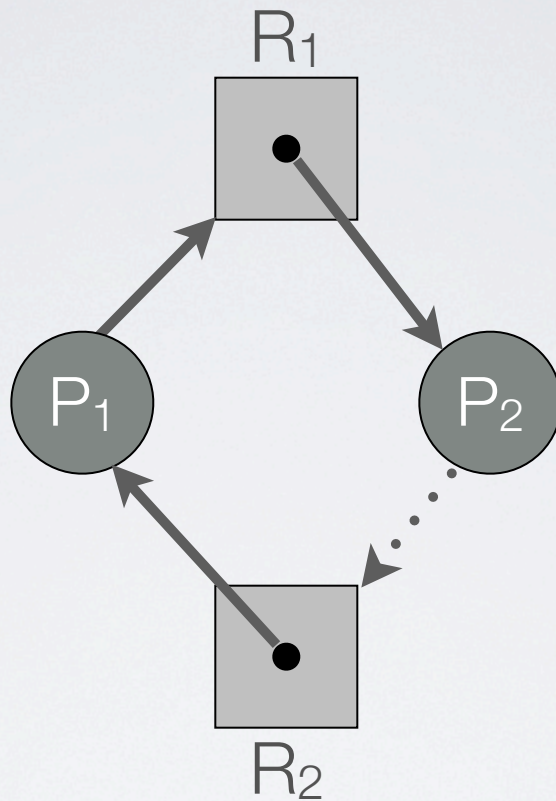
safe?



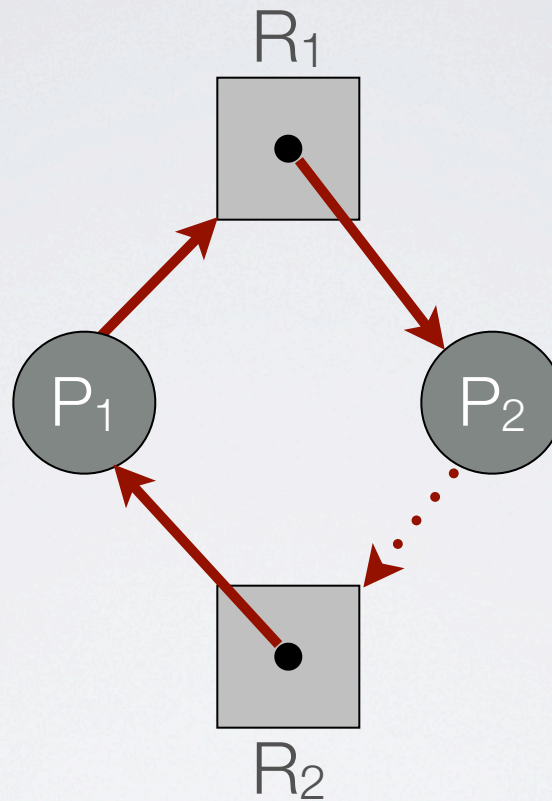
safe?



“claim” edges



P_1 requests R_2



unsafe state

multiple instance resources?

false positives

Banker's algorithm

Safe State

- There exists a sequence $\langle P_1, P_2, \dots, P_n \rangle$, where each P_k can complete with:
 - currently available (free) resources
 - resources held by $P_1 \dots P_{k-1}$

Data Structures

Processes $P_1 \dots P_n$, Resources $R_1 \dots R_m$

- $\text{available}[j]$ = # of R_j available
- $\text{max}[i][j]$ = max # of R_j required by P_i
- $\text{allocated}[i][j]$ = # of R_j allocated to P_i
- $\text{need}[i][j]$ = $\text{max}[i][j] - \text{allocated}[i][j]$

Safety Algorithm

1. $\text{finish}[i] \leftarrow \text{false} \forall i \in 1 \dots n$
 $\text{work} \leftarrow \text{available}$
2. Find $i : \text{finish}[i] = \text{false} \ \& \ \text{need}[i][j] \leq \text{work}[j] \ \forall j$
If none, go to 4.
3. $\text{work} \leftarrow \text{work} + \text{allocated}[i]$; $\text{finish}[i] \leftarrow \text{true}$
Go to 2.
4. Safe state iff $\text{finish}[i] = \text{true} \ \forall i$

request array

$\text{request}[j] = \# R_j \text{ requested}$

Resource Request from P_i

1. If $\text{request}[j] \leq \text{need}[i][j] \quad \forall j$, continue, else error
2. If $\text{request}[j] \leq \text{available}[j] \quad \forall j$, continue, else block
3. Run safety algorithm with:
 - $\text{available} \leftarrow \text{available} - \text{request}$
 - $\text{allocated}[i] \leftarrow \text{allocated}[i] + \text{request}$
 - $\text{need}[i] \leftarrow \text{need}[i] - \text{request}$

3 resources: A (10), B (5), C (7)

	Max			Allocated			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	7	5	3	0	1	0	3	3	2	7	4	3
P ₁	3	2	2	2	0	0				1	2	2
P ₂	9	0	2	3	0	2				6	0	0
P ₃	2	2	2	2	1	1				0	1	1
P ₄	4	3	3	0	0	2				4	3	1

- Safe state: $\langle P_1, P_3, P_0, P_2, P_4 \rangle$
- P₁ requests $\langle 1, 0, 2 \rangle$

unreasonable assumption
max resource requirements

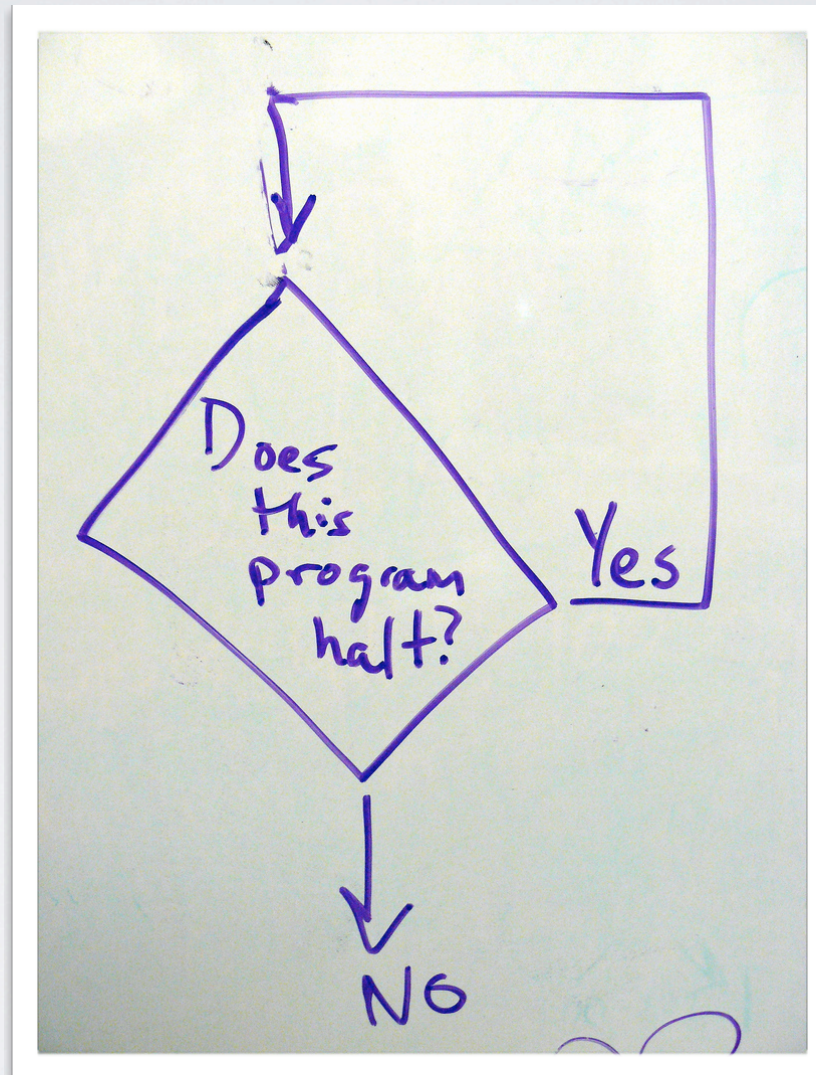
decision theory

halting problem

halt(f, x)

```
def reverse(f,x)
  if (halt(f,x))
    while (1) ;
  else
    return 1
end
```

halt(reverse, x) ?



proof by contradiction
“undecidable” problem

without resource information,
deadlock detection
is provably **impossible**

(3)

detection & recovery

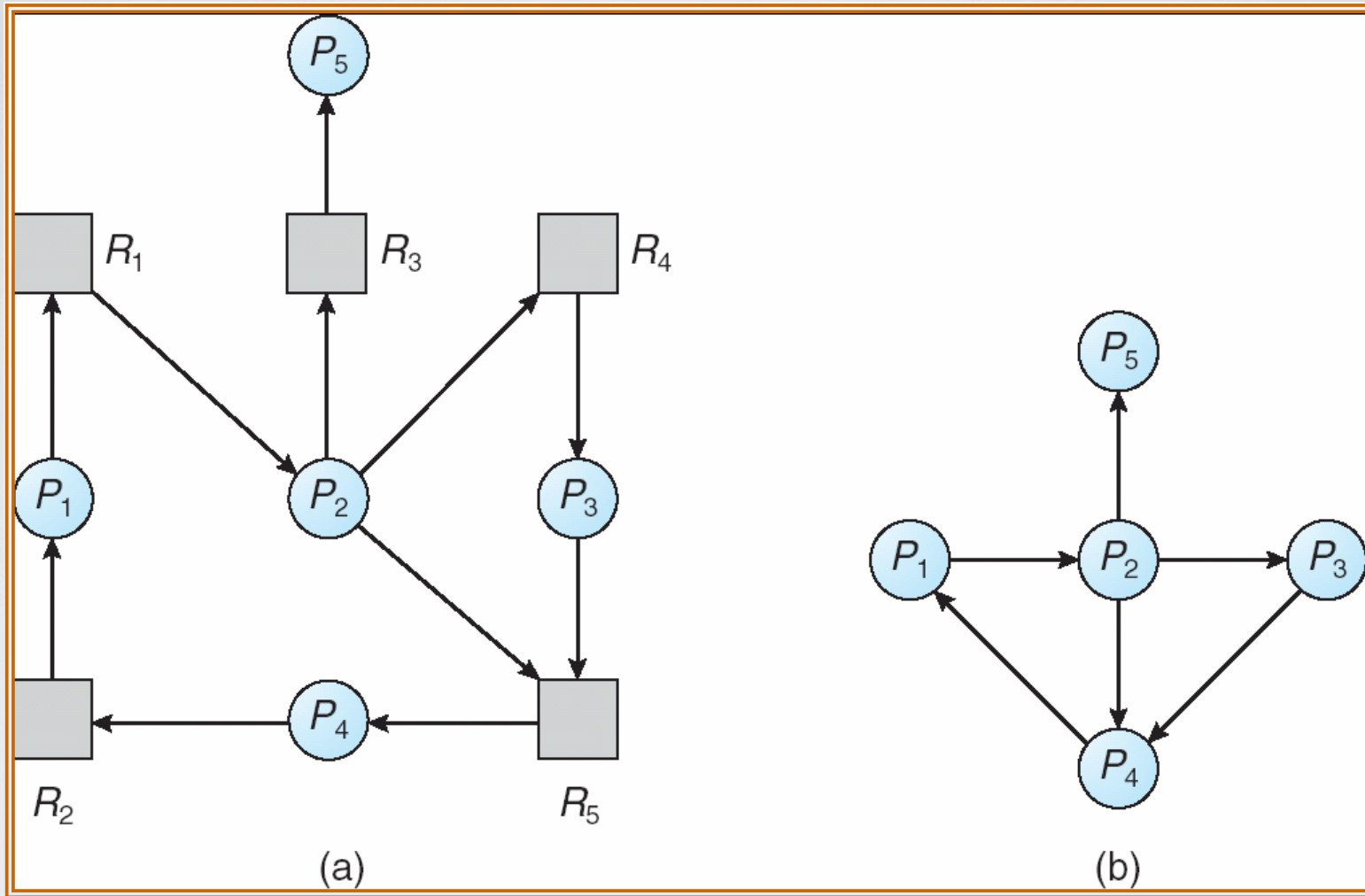
cycle detection algorithm

$$O(N^2)$$

$N = \#$ vertices

reduce vertices

“wait-for” graph



Resource Allocation
Graph

Wait-for
Graph

multiple instances?

modified Banker's algorithm

“max” not needed
track all pending requests

request matrix

$\text{request}[i][j] = \# R_j \text{ requested by } P_i$

Detection Algorithm

1. $\text{finish}[i] \leftarrow \text{all_nil?}(\text{allocated}[i]) \quad \forall i \in 1 \dots n$
 $\text{work} \leftarrow \text{available}$
2. Find $i : \text{finish}[i] = \text{false} \ \& \ \text{request}[i][j] \leq \text{work}[j] \quad \forall j$
If none, go to 4.
3. $\text{work} \leftarrow \text{work} + \text{allocated}[i]; \text{finish}[i] \leftarrow \text{true}$
Go to 2.
4. If $\text{finish}[i] \neq \text{true} \quad \forall i$, system is deadlocked.

assumption

processes complete with
current resources

3 resources: A (7), B (2), C (6)

	Allocated			Request			Available		
	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			

- Safe state: $\langle P_0, P_2, P_1, P_3, P_4 \rangle$
- P₂ requests $\langle 0, 0, 1 \rangle$

recovery

termination / preemption

rollback

selection criteria

minimizing cost

processes
process lengths
resources held
resources needed
arbitrary priority

starvation

livelock