Principles of Operating Systems

CS 446/646

2. Processes

a. Process Description & Control

b. Threads

c. Concurrency

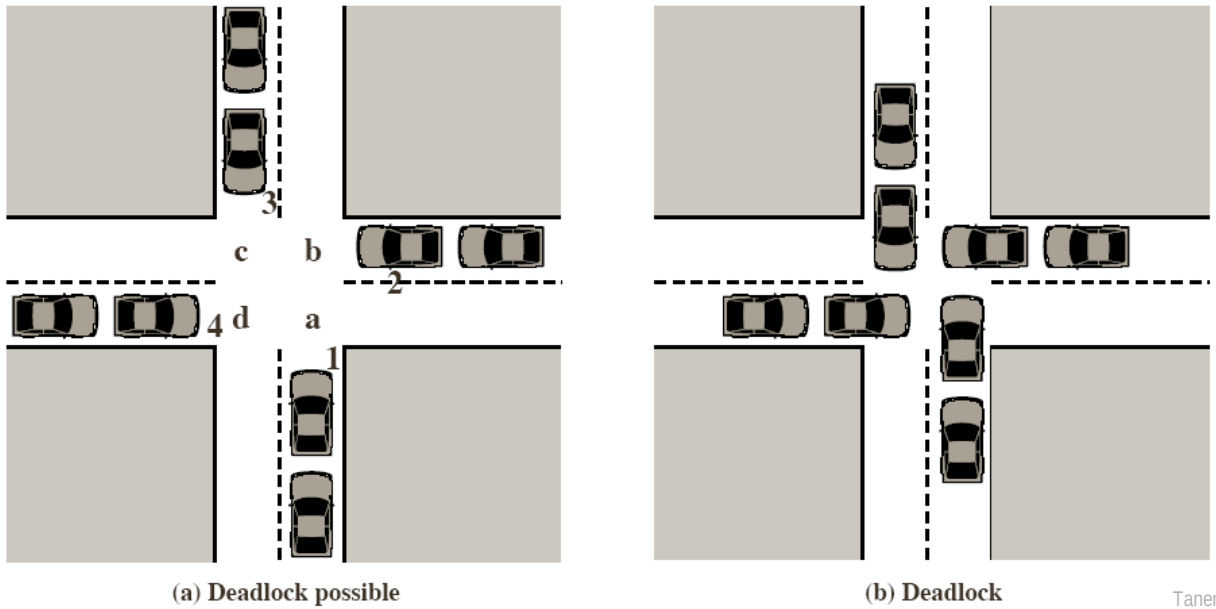
d. Deadlocks

- ✓ Deadlock principles: diagrams and graphs
- ✓ Deadlock prevention: changing the rules
- ✓ Deadlock avoidance: optimizing the allocation
- ✓ Deadlock detection: recovering after the facts

2.d Deadlocks

Deadlock principles: diagrams and graphs

- A deadlock is a permanent blocking of a set of threads
  - ✓ a deadlock can happen while threads/processes are competing for system resources or communicating with each other
  - ✓ there is no universal efficient solution against deadlocks



(a) Deadlock possible (b) Deadlock

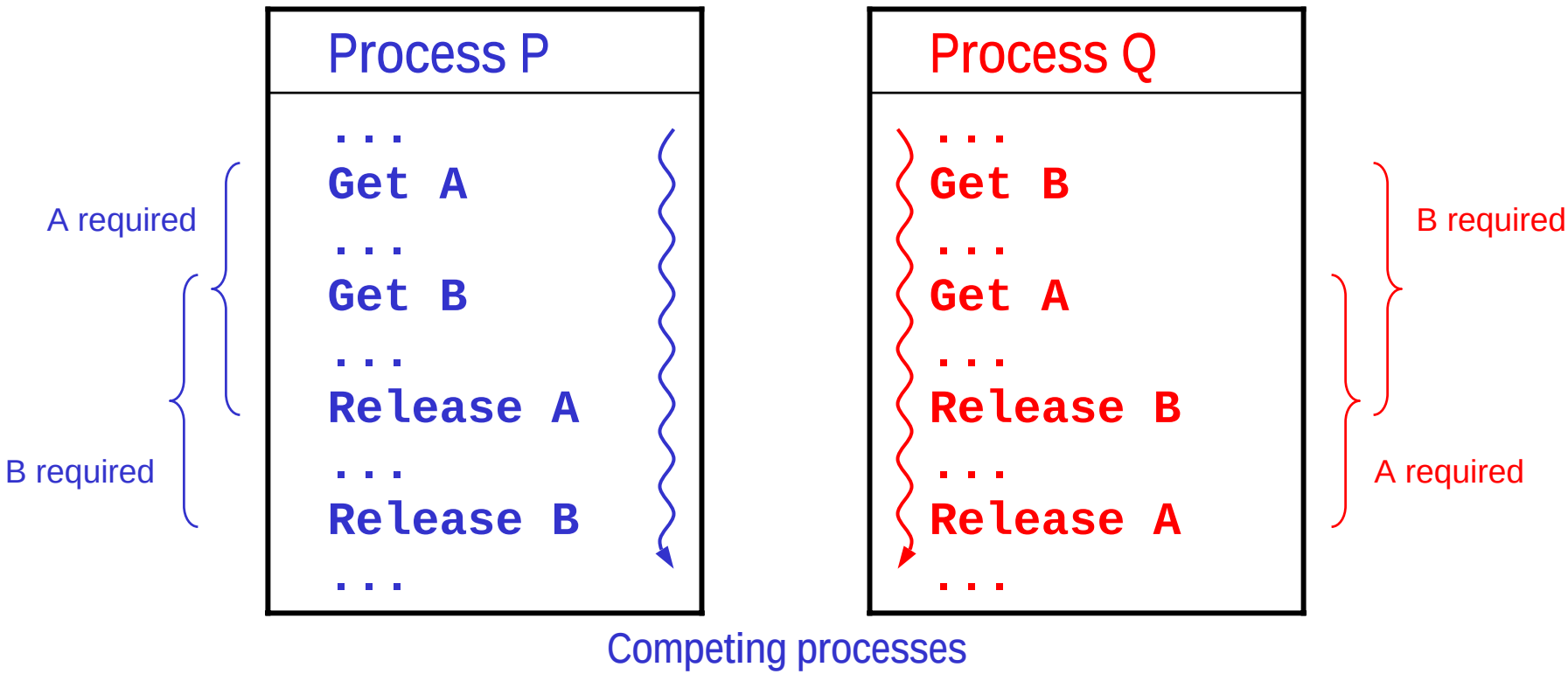
Tanenbaum, A. S. (2001)
Modern Operating Systems (2nd Edition).

Illustration of a deadlock

2.d Deadlocks

Deadlock principles: diagrams and graphs

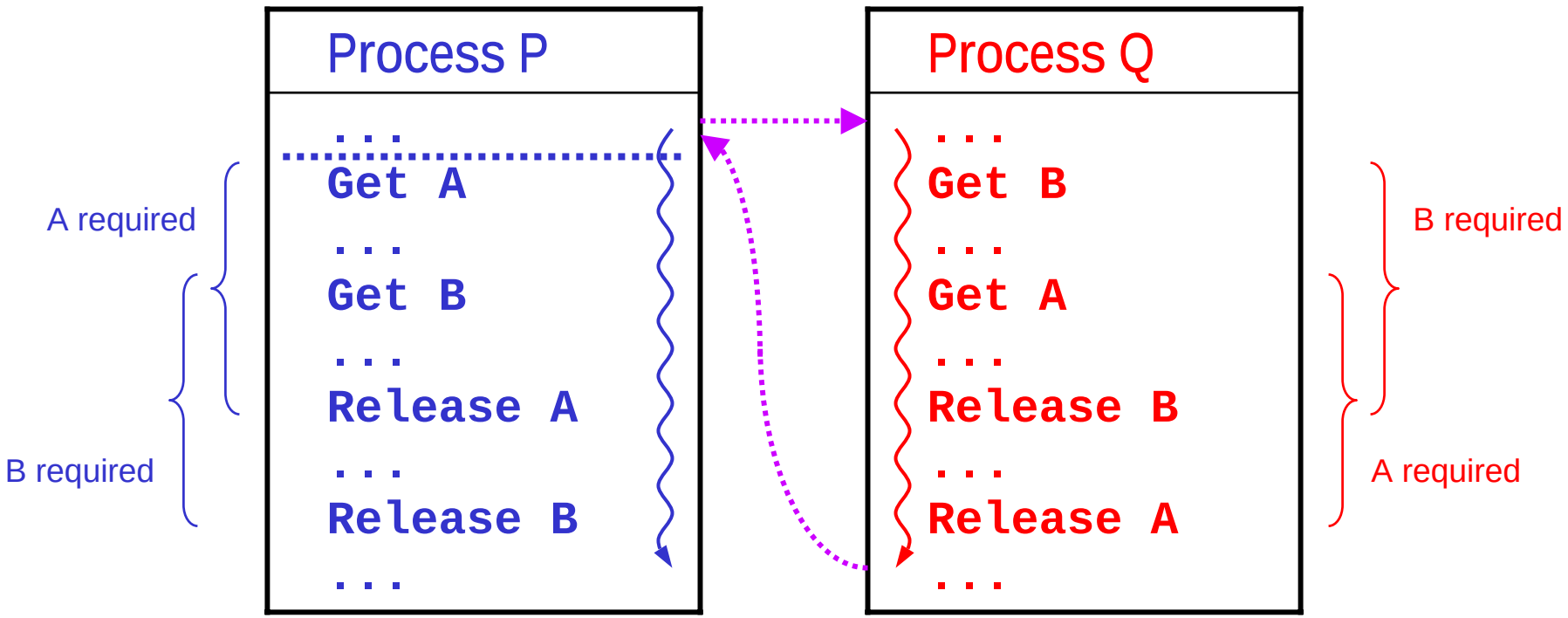
- Illustration of a deadlock
  - ✓ two processes, P and Q, compete for two resources, A and B
  - ✓ each process needs exclusive use of each resource



2.d Deadlocks

Deadlock principles: diagrams and graphs

- Illustration of a deadlock — scheduling path 1 ☺
  - ✓ Q executes everything before P can ever get A
  - ✓ when P is ready, resources A and B are free and P can proceed

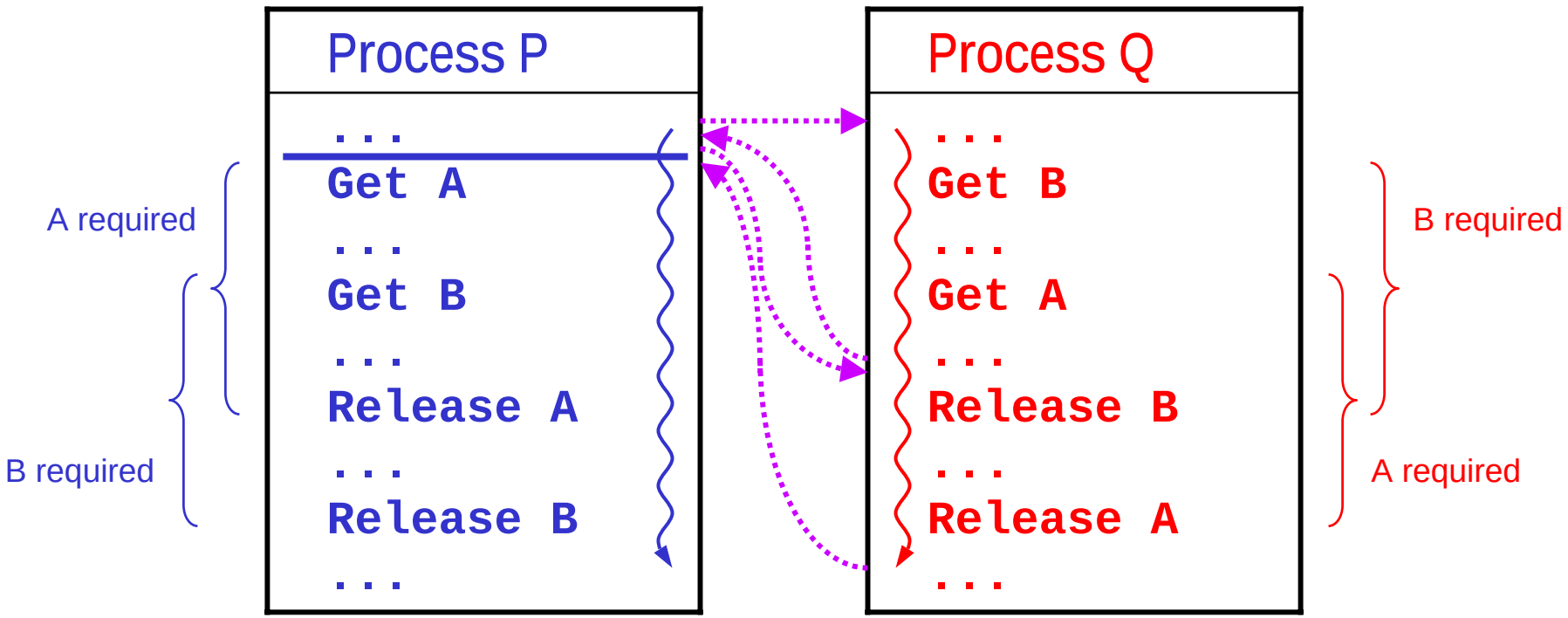


Happy scheduling 1

2.d Deadlocks

Deadlock principles: diagrams and graphs

- Illustration of a deadlock — scheduling path 2 ☺
  - ✓ Q gets B and A, then P is scheduled; P wants A but is blocked by A's mutex; so Q resumes and releases B and A; P can now go

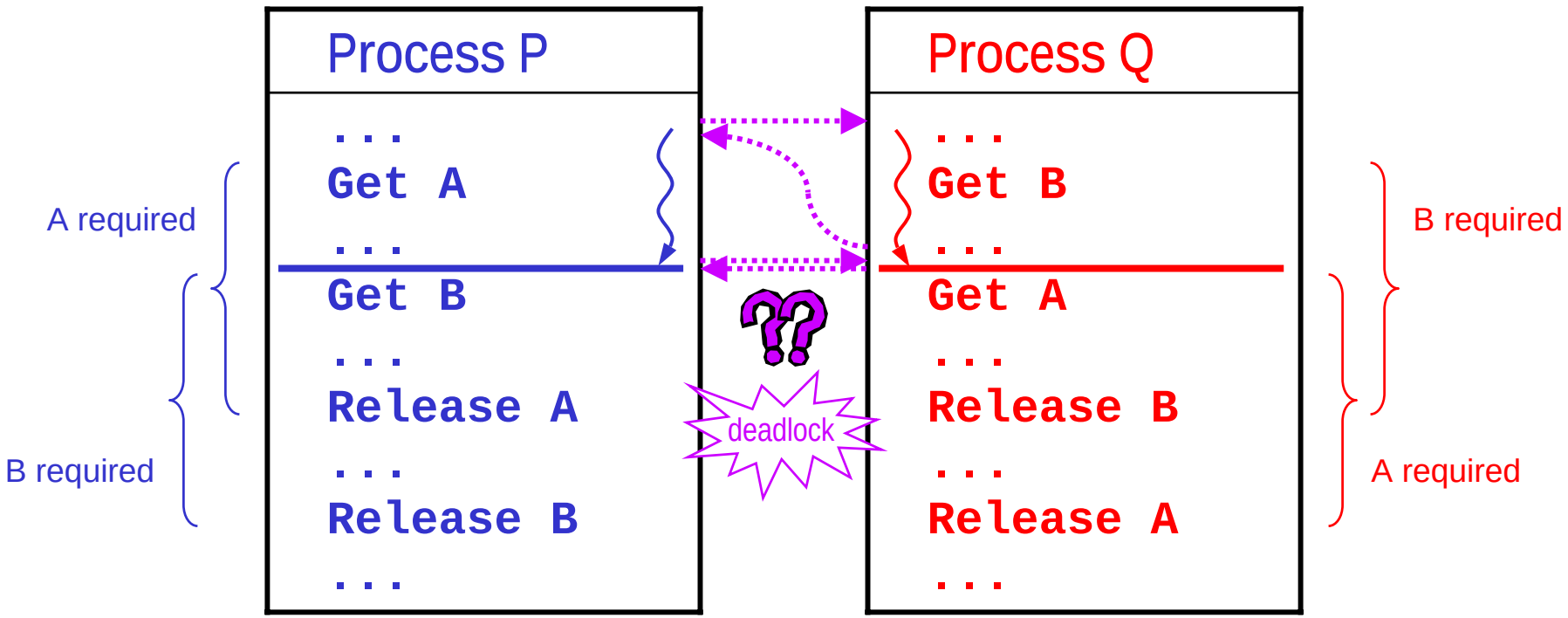


Happy scheduling 2

2.d Deadlocks

Deadlock principles: diagrams and graphs

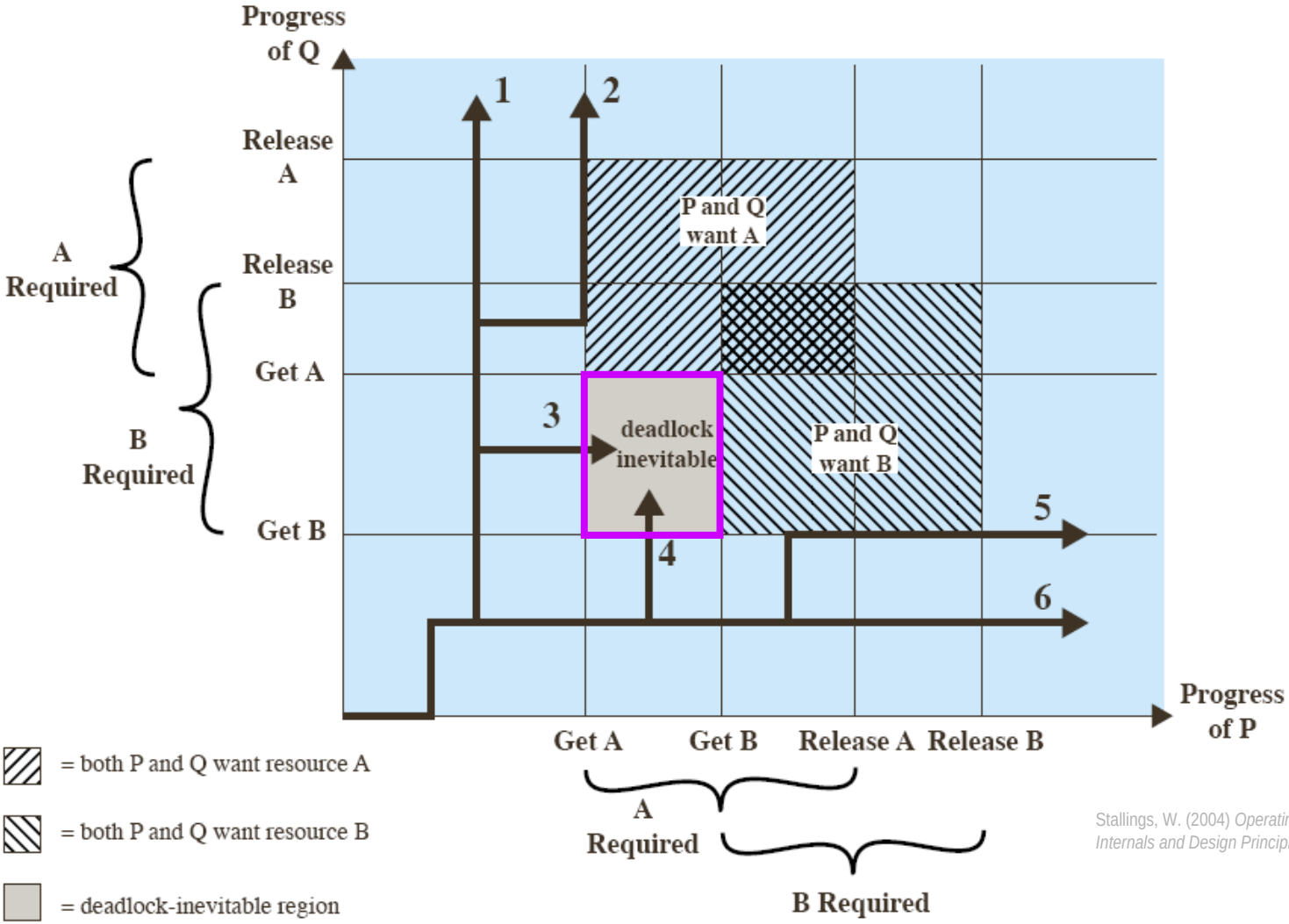
- Illustration of a deadlock — scheduling path 3 ☹
  - ✓ Q gets only B, then P is scheduled and gets A; now both P and Q are blocked, each waiting for the other to release a resource



Bad scheduling → deadlock

2.d Deadlocks

Deadlock principles: diagrams and graphs



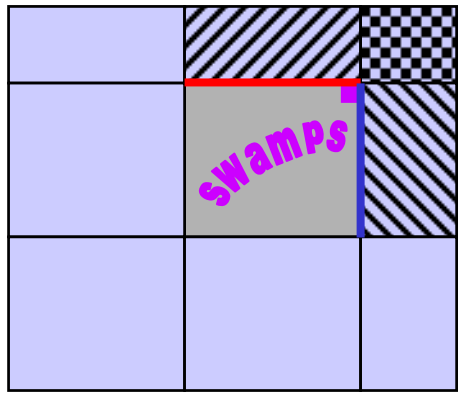
Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

Joint progress diagram

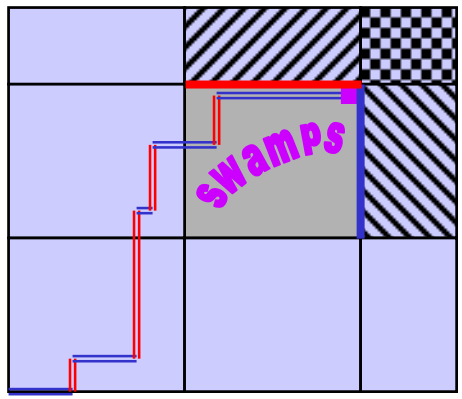
2.d Deadlocks

Deadlock principles: diagrams and graphs

- Deadlocks depend on the program and the scheduling
  - ✓ program design
    - the order of the statements in the code creates the "landscape" of the joint progress diagram
    - this landscape may contain gray "swamp" areas leading to deadlock



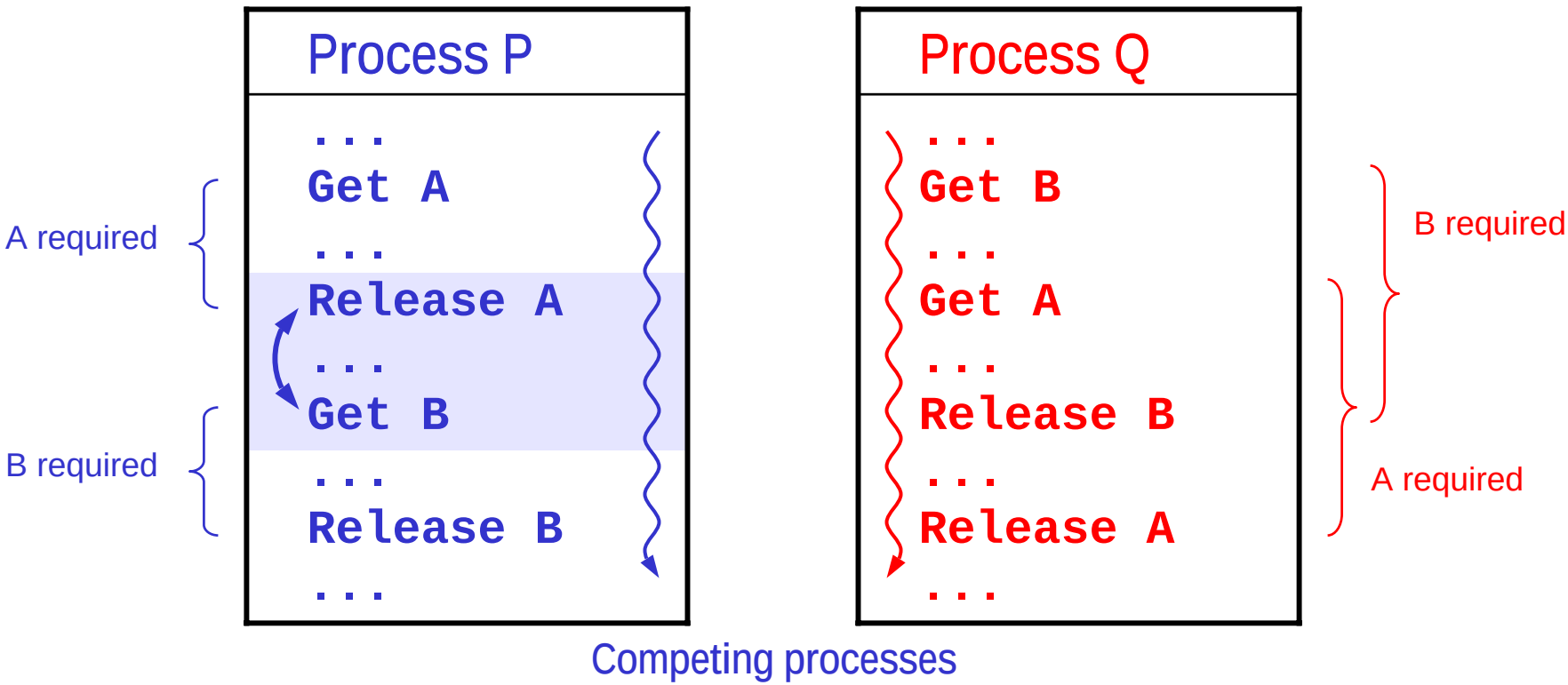
  - ✓ scheduling condition
    - the interleaved dynamics of multiple executions traces a "path" in this landscape
    - this path may sink in the swamps



2.d Deadlocks

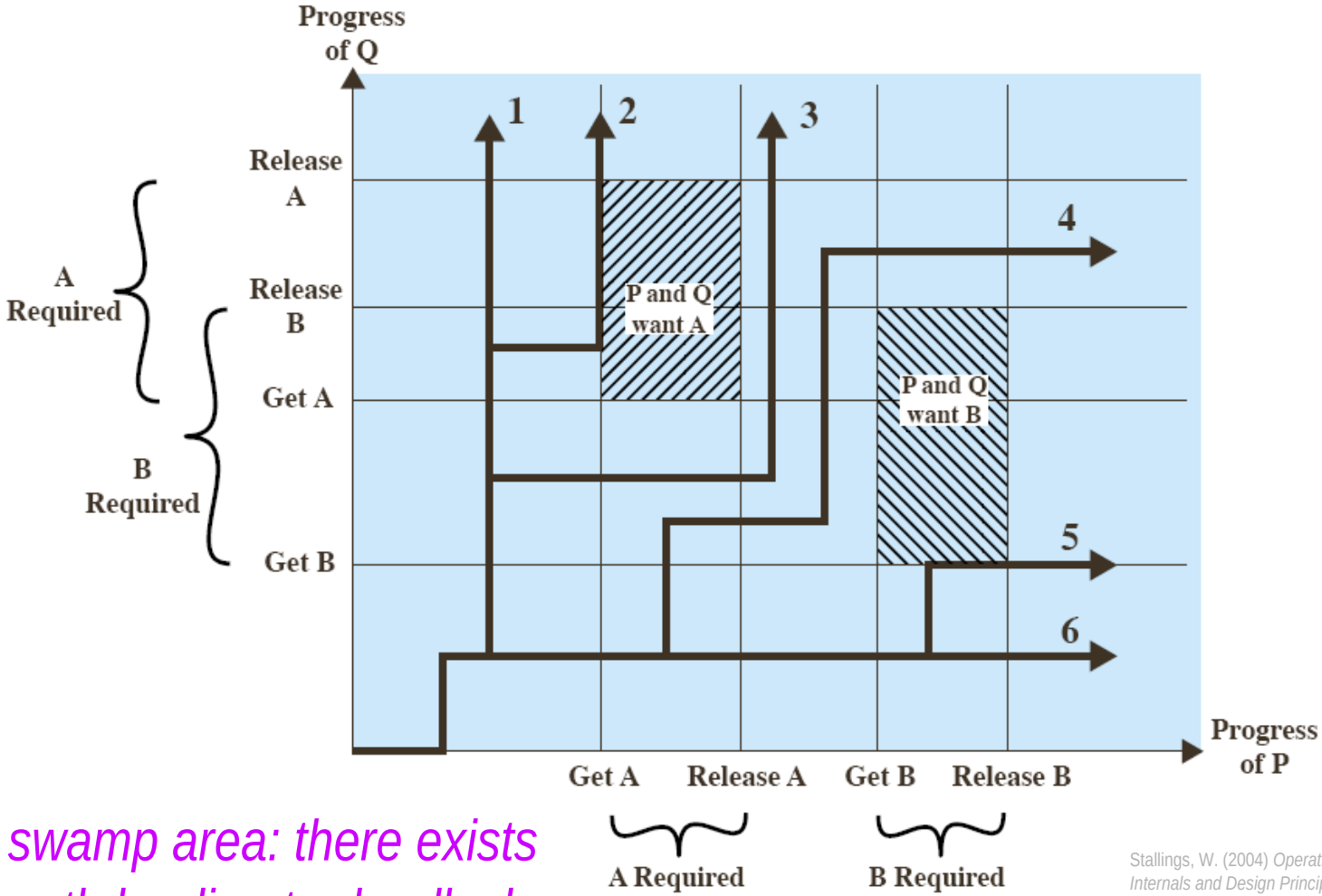
Deadlock principles: diagrams and graphs

- Changing the program changes the landscape
  - ✓ here, P releases A before getting B
  - ✓ deadlocks between P and Q are not possible anymore



2.d Deadlocks

Deadlock principles: diagrams and graphs



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

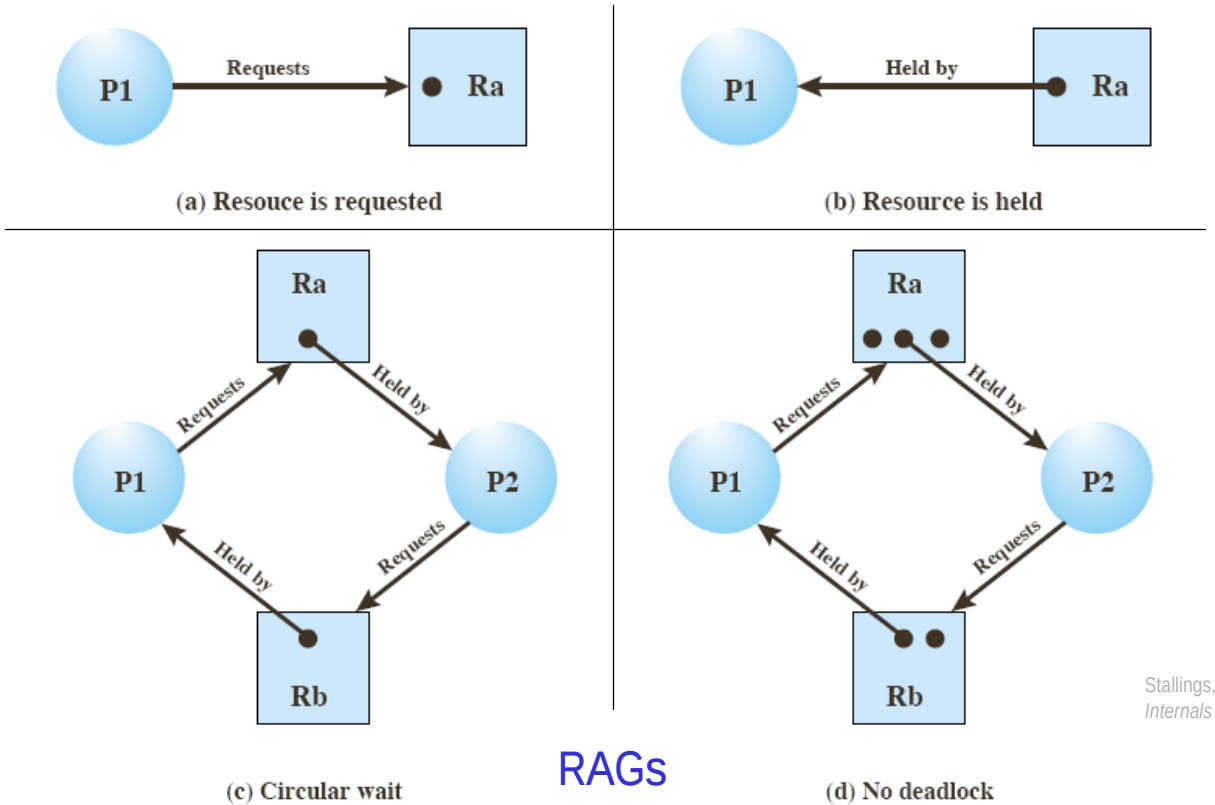
→ *no swamp area: there exists no path leading to deadlock*

Joint progress diagram

2.d Deadlocks

Deadlock principles: diagrams and graphs

- Snapshot of concurrency: Resource Allocation Graph
  - ✓ a resource allocation graph is a directed graph that depicts a state of the system of resources and processes



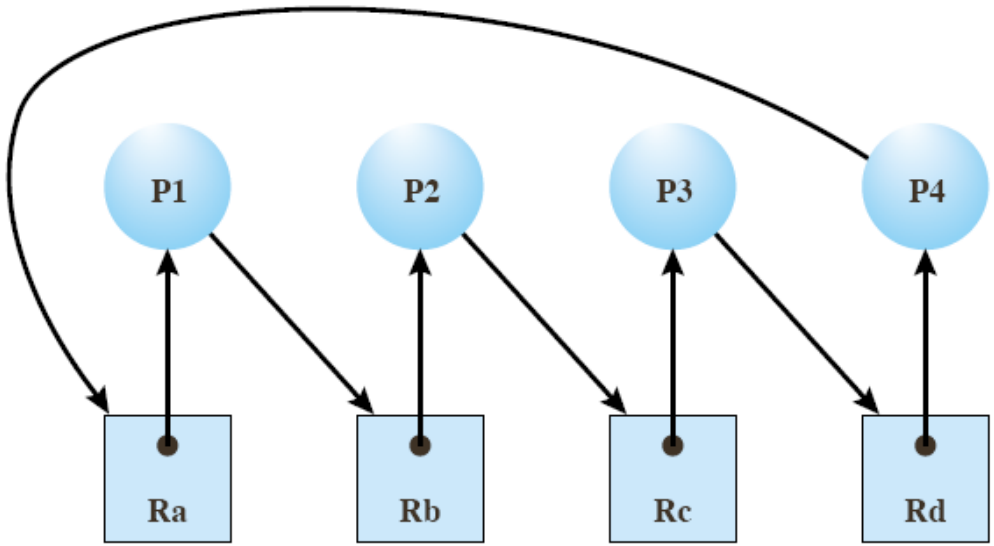
RAGs

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

2.d Deadlocks

Deadlock principles: diagrams and graphs

- Resource allocation graphs & deadlocks
  - ✓ there is deadlock when a closed chain of processes exists
  - ✓ each process holds at least one resource needed by the next process



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

A deadlock's RAG

2.d Deadlocks

Deadlock principles: diagrams and graphs

➢ Design conditions for deadlock (create the swamps)

1. **mutual exclusion** — the design contains protected critical regions; only one process at a time may use these
2. **hold & wait** — the design is such that, while inside a critical region, a process may have to wait for another critical region
3. **no resource preemption** — there must not be any hardware or O/S mechanism forcibly removing a process from its CR

\+ Scheduling condition for deadlock (go to the swamps)

4. **circular wait** — two or more hold-&-wait's are happening in a circle: each process holds a resource needed by the next

= Deadlock!

2.d Deadlocks

Deadlock principles: diagrams and graphs

- Three strategies for dealing with deadlocks
  - ✓ deadlock prevention — changing the rules
    - one or several of the deadlock conditions 1., 2., 3. or 4. are removed *a priori* (design decision)
  - ✓ deadlock avoidance — optimizing the allocation
    - deadlock conditions 1., 2., 3. are maintained but resource allocation follows extra cautionary rules (runtime decision)
  - ✓ deadlock detection — recovering after the facts
    - no precautions are taken to avoid deadlocks, but the system cleans them periodically ("deadlock collector")

2.d Deadlocks

Deadlock prevention: changing the rules

- Remove one of the design or scheduling conditions?
  - ✓ remove "mutual exclusion"?
    - → *not possible: must always be supported by the O/S*
  - ✓ remove "hold & wait"?
    - require that a process gets all its resources at one time
    - → *inefficient and impractical: defeats interleaving, creates long waits, cannot predict all resource needs*
  - ✓ remove "no preemption" = allow preemption?
    - require that a process releases and requests again → *ok*
  - ✓ remove "circular wait"?
    - ex: impose an ordering of resources → *inefficient, again*

2.d Deadlocks

Deadlock avoidance: optimizing the allocation

- Allow all conditions, but allocate wisely
  - ✓ given a resource allocation request, a decision is made dynamically whether granting this request can potentially lead to a deadlock or not
    - do not start a process if its demands might lead to deadlock
    - do not grant an incremental resource request to a running process if this allocation might lead to deadlock
  - ✓ avoidance strategies requires knowledge of future process request (calculating “chess moves” ahead)

2.d Deadlocks

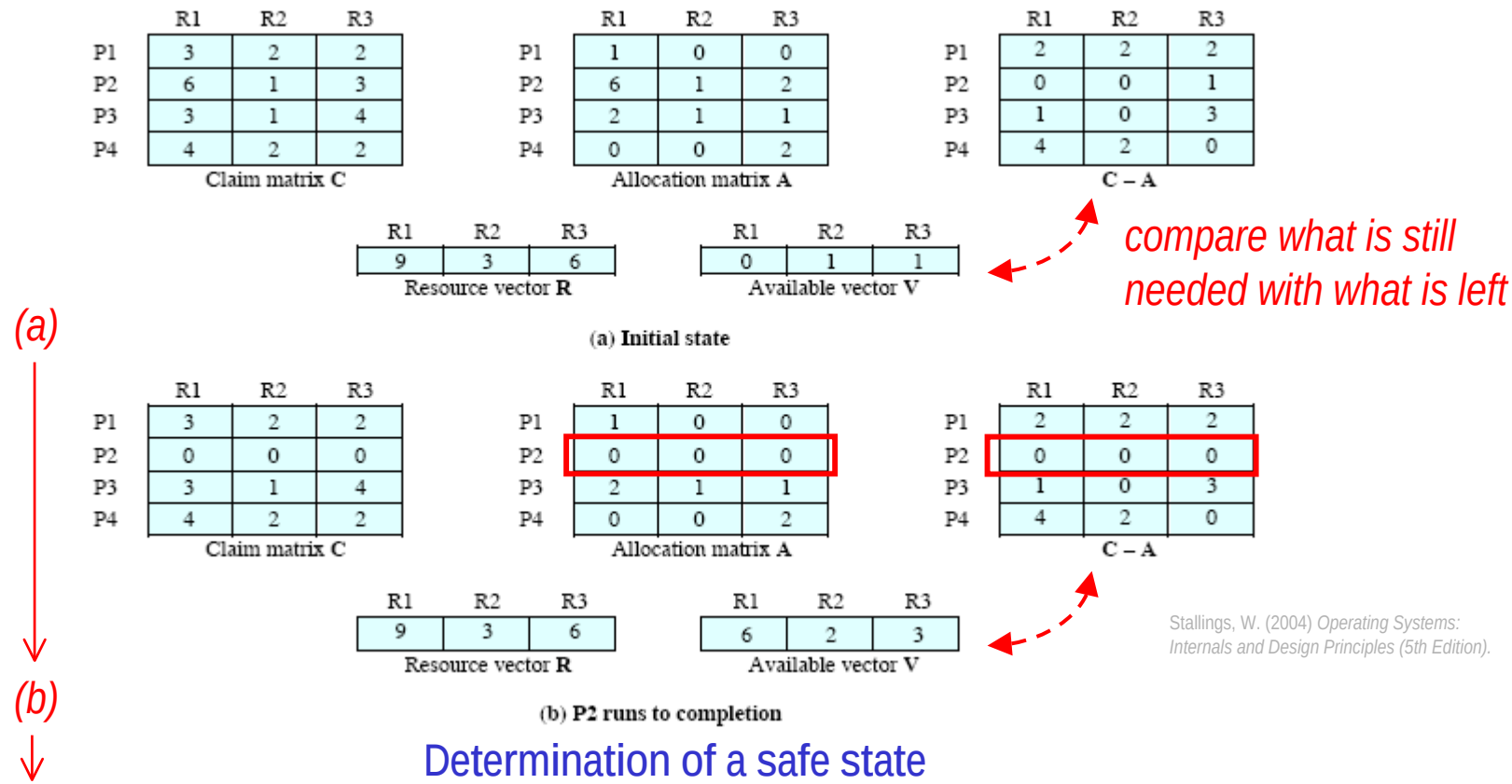
Deadlock avoidance: optimizing the allocation

- Resource allocation denial: the “banker's algorithm”
  - ✓ at any time, the state of the system is the current allocation of multiple resources to multiple processes
    - a safe state is where there is at least one sequence that does not result in deadlock
    - an unsafe state is a state where there is no such sequence
  - ✓ analogy = banker refusing to grant a loan if funds are too low to grant more loans + uncertainty about how long a customer will repay

2.d Deadlocks

Deadlock avoidance: optimizing the allocation

- Resource allocation denial: the “banker's algorithm”
  - ✓ can a process run to completion with the available resources?



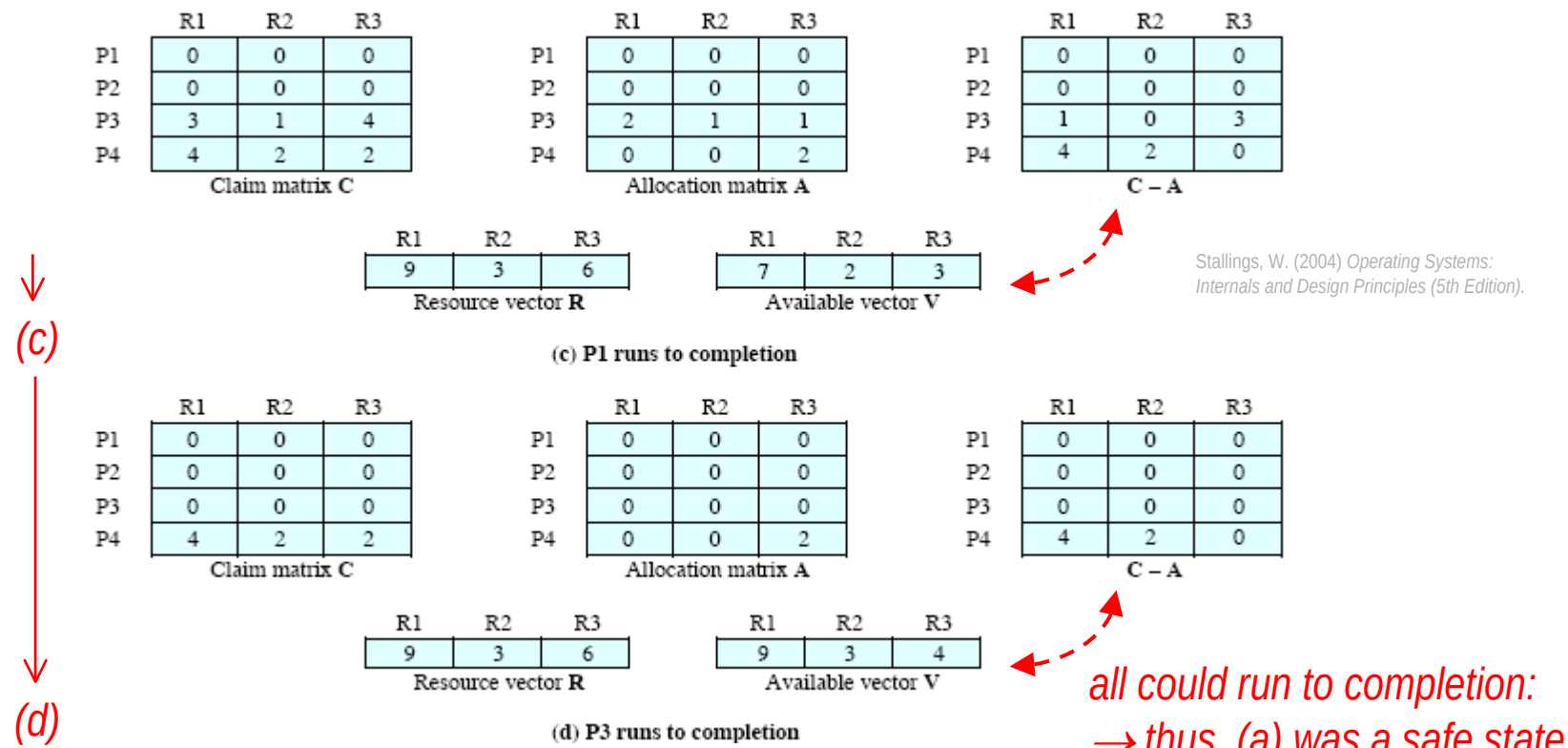
Determination of a safe state

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

2.d Deadlocks

Deadlock avoidance: optimizing the allocation

- Resource allocation denial: the "banker's algorithm"
  - ✓ idea: refuse to allocate if it may result in deadlock



Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

all could run to completion:
→ thus, (a) was a safe state

Determination of a safe state (cont'd)

2.d Deadlocks

Deadlock avoidance: optimizing the allocation

- Resource allocation denial: the "banker's algorithm"
  - ✓ idea: refuse to allocate if it may result in deadlock

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
1	1	2

Available vector **V**

Stallings, W. (2004) *Operating Systems: Internals and Design Principles (5th Edition).*

(a) Initial state

(a) safe ← (a')

(b') unsafe

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(b) P1 requests one unit each of R1 and R3

potential for deadlock (we don't know how long Ri will be kept)
→ thus, (b') is an unsafe state: don't allow (b') to happen

Determination of an unsafe state

2.d Deadlocks

Deadlock detection: recovering after the facts

Principles of Operating Systems

CS 446/646

2. Processes

a. Process Description & Control

b. Threads

c. Concurrency

d. Deadlocks

- ✓ Deadlock principles: diagrams and graphs
- ✓ Deadlock prevention: changing the rules
- ✓ Deadlock avoidance: optimizing the allocation
- ✓ Deadlock detection: recovering after the facts

Principles of Operating Systems

CS 446/646

2. Processes

a. Process Description & Control

b. Threads

c. Concurrency

d. Deadlocks

Principles of Operating Systems

CS 446/646

0. Course Presentation
1. Introduction to Operating Systems
2. Processes
3. **Memory Management**
4. **CPU Scheduling**
5. **Input/Output**
6. **File System**
7. **Case Studies**