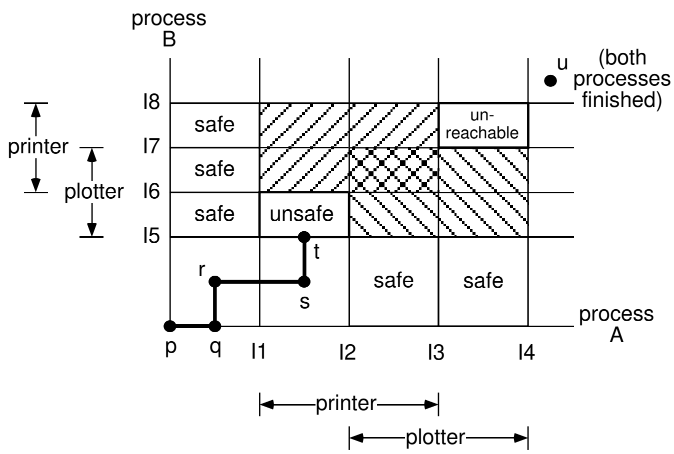
Deadlock Avoidance — Motivation



- Example to motivate a D.A. algorithm:
  - state p — neither process running
  - state q — scheduler ran A
  - state r — scheduler ran B
  - state s — scheduler ran A, A requested and received printer
  - state t — schedule ran B, B just requested and received plotter

Deadlock Avoidance — Motivation (cont.)

- Look at shaded areas:
  - The one shaded "\\\" represents both processes using printer at same time — this is not allowed by mutual exclusion
  - Other ("///") is similar, involving plotter
- Look at box marked "unsafe"
  - If OS enters this box, it will eventually deadlock because it will have to enter a shaded (illegal mutual exclusion) region
    - All paths must proceed up or right (why?)
  - Box is unsafe — should not be entered!
    - From state t, must avoid the unsafe area by going to the right (up to I4) (blocking B)
- At state t, the OS must decide whether or not to grant B's request
  - A good choice will avoid deadlock!
  - Need to know resource needs in advance

Deadlock Avoidance — Safe and Unsafe States

	Has	Max
A	3	9
B	2	4
C	2	7

free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

free: 1
(b)

	Has	Max
A	3	9
B	0	—
C	2	7

free: 5
(c)

	Has	Max
A	3	9
B	0	—
C	7	7

free: 0
(d)

	Has	Max
A	3	9
B	0	—
C	0	—

free: 7
(e)

	Has	Max
A	9	9
B	0	—
C	0	—

free: 1
(f)

	Has	Max
A	0	—
B	0	—
C	0	—

free: 10
(g)

- State (a) is *safe*, meaning there **exists** a sequence of allocations that allows **all** processes to complete:
  - B runs, asks for 2 more resources, 1 free
    - B finishes, releases its resources, 5 free
  - C runs, asks for 5 more resources, 0 free
    - C finishes, releases its resources, 7 free
  - A runs, gets 6 more, everyone done…

Deadlock Avoidance — Safe and Unsafe States (cont.)

	Has	Max
A	3	9
B	2	4
C	2	7

free: 3
(a)

	Has	Max
A	4	9
B	2	4
C	2	7

free: 2
(b)

	Has	Max
A	4	9
B	4	4
C	2	7

free: 0
(c)

	Has	Max
A	4	9
B	0	—
C	2	7

free: 4
(d)

- Suppose we start in state (a), and reach state (b) by giving A another resource
  - B runs, asks for 2 more resources, 0 free
    - B finishes, releases its resources, 4 free
  - C can't run — might want 5 resources
    - Same for A
- State (b) is unsafe, meaning that from there, deadlock **may** eventually occur
    - State (b) is not a deadlocked state — the system can still run for a bit
    - Deadlock may not occur — A might release one of its resources before asking for more, which allows C to complete

The Banker's Algorithm for Single Resources (Dijkstra, 1965)

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

free: 10
(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

free: 2
(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

free: 1
(c)

- A banker has granted lines of credit to customers A, B, C, and D (unit is $1000)
  - She knows it's not likely they will all need their maximum credit at the same time, so she keeps only 10 units of cash on hand
  - At some point in time, the bank is in state (b) above, which is *safe*
    - Can let C finish, have 4 units available
    - Then let B or D finish, etc.
  - But… if banker gives B one more unit (state (c) above), state would be *unsafe* — if everyone asks for maximum credit, **no** requests can be fulfilled

The Banker's Algorithm for Single Resources (cont.)

- Resource-request algorithm:
  - The banker considers each request as it occurs, determining whether or not fulfilling it leads to a safe state
    - If it does, the request is granted
    - Otherwise, it is postponed until later
- Safety algorithm:
  - To determine if a state is safe, the banker checks to see if she has enough resources to satisfy <u>some</u> customer
    - If so, she assumes those loans will be repaid (i.e., the process will use those resources, finish, and release all of its resources), and she checks to see if she has enough resources to satisfy another customer, etc.
  - If <u>all</u> loans can eventually be repaid, the state is safe and the initial request can be granted

Evaluation of Deadlock Avoidance Using the Banker's Algorithm

- Advantages:
  - No need to preempt resources and rollback state (as in deadlock detection & recovery)
  - Less restrictive than deadlock prevention
- Disadvantages:
  - Maximum resource requirement for each process must be stated in advance
  - Processes being considered must be independent (i.e., unconstrained by synchronization requirements)
  - There must be a fixed number of resources (i.e., can't add resources, resources can't break) and processes (i.e., can't add or delete processes)
  - Huge overhead — must use the algorithm every time a resource is requested

Evaluating the Approaches to Dealing with Deadlock

- *The Ostrich Approach* — ignoring the problem
  - Good solution if deadlock isn't frequent
- *Deadlock prevention* — eliminating one of the 4 deadlock conditions
  - May be overly restrictive
- *Deadlock detection and recovery* — detect when deadlock has occurred, then break the deadlock
  - Tradeoff between frequency of detection and performance / overhead added
- *Deadlock avoidance* — only fulfilling requests that will not lead to deadlock
  - Need too much a priori information, not very dynamic (can't add processes or resources), huge overhead