Safe to Unsafe Transition

Current safe state doesn't imply future states are safe:

	Current Loan	Max Need
User1	1	4
User2	4	6
User3	5	8

Available 2

Suppose User3 requests and gets one more resource:

	Current Loan	Max Need
User1	1	4
User2	4	6
User3	6	8

Available 1

Essence of Banker's Algorithm

Find schedule to complete jobs:

schedule exists iff safe

Method:

"Pretend" you are the CPU.

1. Scan table row by row and find a job that can finish.

2. Add finished job's resources to number available.

Repeat 1 and 2 until:

- no more jobs can finish *(unsafe)*, or
- all jobs finish *(safe)*.

Banker's Algorithm

Constants:
- N *{number of processes}*
- Total_Units
- Max_Demand [i]

Variables:
- i *{denotes a process}*
- Unused_units
- Allocated [i]
- Claim [i]
- Cannot_Finish [i]

Function:

Claim [i] =
  Max_Demand[i] - Allocated[i]

Banker's Algorithm

```
Begin
  Unused_units == Total_Units;

  For I = 1 to N Do
     Begin
     Unused_Units = Unused_Units - Allocated [i];
     Cannot_Finish [i] = TRUE;
     End;

  i = 1;

  While ( i<=N ) Do
     Begin
       If ( Cannot_Finish [i] AND Claim [i] <= Unused_Units )
          Then Begin
             Cannot_Finish [i] = False;
             Unused_units = Unused_Units + Allocated [i];
             I = 1;
             End;
          Else i = i+1;
     End;

  If ( Unused_Units == Total_Units )
     Then Return ( SAFE )
     Else Return ( UNSAFE );
End;
```

Initialize

Find schedule to complete all jobs

Banker's Example # 1

Total_Units = 10 units
N = 3 processes

Process:	1	2	3	1
Request:	2	3	4	1

Can the fourth request by satisfied?

Process	MaxDemand	Allocated	Claim	MNF
1	4			
2	4			
3	8			

Unused_Units =
i =

Banker's Example #2

Total_Units = 10 units
N = 3 processes

Process:	1	2	3	1
Request:	4	1	1	2

Can the fourth request by satisfied?

Process	MaxDemand	Allocated	Claim	MNF
1	10			
2	6			
3	3			

Unused_Units =
i =

Banker's Algorithm: Summary

<u>(+) PRO's:</u>

+ Deadlock never occurs.
+ More flexible, efficient than deadlock prevention. (Why?)

<u>(−) CON's:</u>

- Must know max use of each resource when job starts.
  - => No truly dynamic allocation
- Process could block even though deadlock would never occur.