

Operating Systems:

Chap 6:

DeadLocks avoidance:

The Banker's Algorithm

CS400

Week 13: 8th April

Spring 2020

Oliver BONHAM-CARTER

Participation 6 and 7: Instructions at the end of slides.

How to Avoid Deadlocks

- We must select our processes according to how they consume their resources.
- Ex: if a resource is not available for a particular process, then we should not choose it to run.
- Running this resource will force it to hold a resource and prevent other processes from using that resource.

The Banker's Algorithm

"The avoidance algorithm"

- Deadlock Handling
- Choose processes for which resources are available to for it to run to completion.
- A resource allocation and deadlock avoidance algorithm
- Bankers algorithm is used to schedule processes according to their required resources.
- Bankers algorithm produces a safe sequence of processes to run as a output if all the resources can be executed
- Return an error signal if no safe sequence of the processes is available (possible).

Inventor's Page



Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes. Science is not about tools. It is about how we use them, and what we find out when we do.

— *Edsger Dijkstra* —

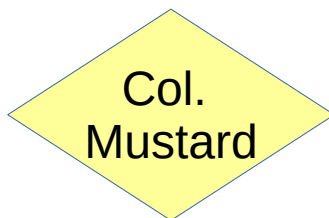
- The Banker's Algorithm: Developed by Edsger Dijkstra

Basic Banker's Dilemmas

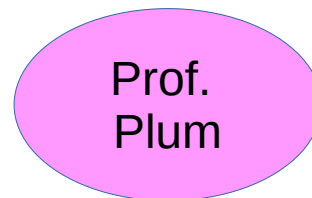


- Say you have to loan money to three people to complete some task.
- Each person may already have own money but will need a small loan from you to complete the task
- You will get back your loan money (and theirs too) ONLY when their task is completed.

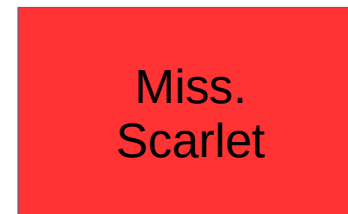
Basic Example



Col.
Mustard



Prof.
Plum



Miss.
Scarlet

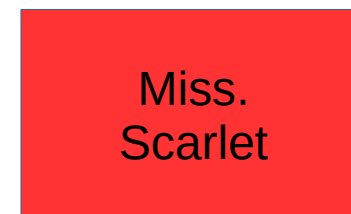
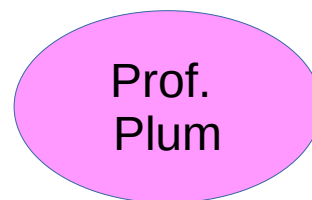
Required:	8	13	10
Already given:	6	8	7
Needs:	2	5	3

The banker starts with \$24 to lend
and begins lending:

$$\$24 - \$6 - \$8 - \$7 = \$3 \text{ (left over)}$$

- Three people to which money is lent to help them finish “tasks”.

A Loan



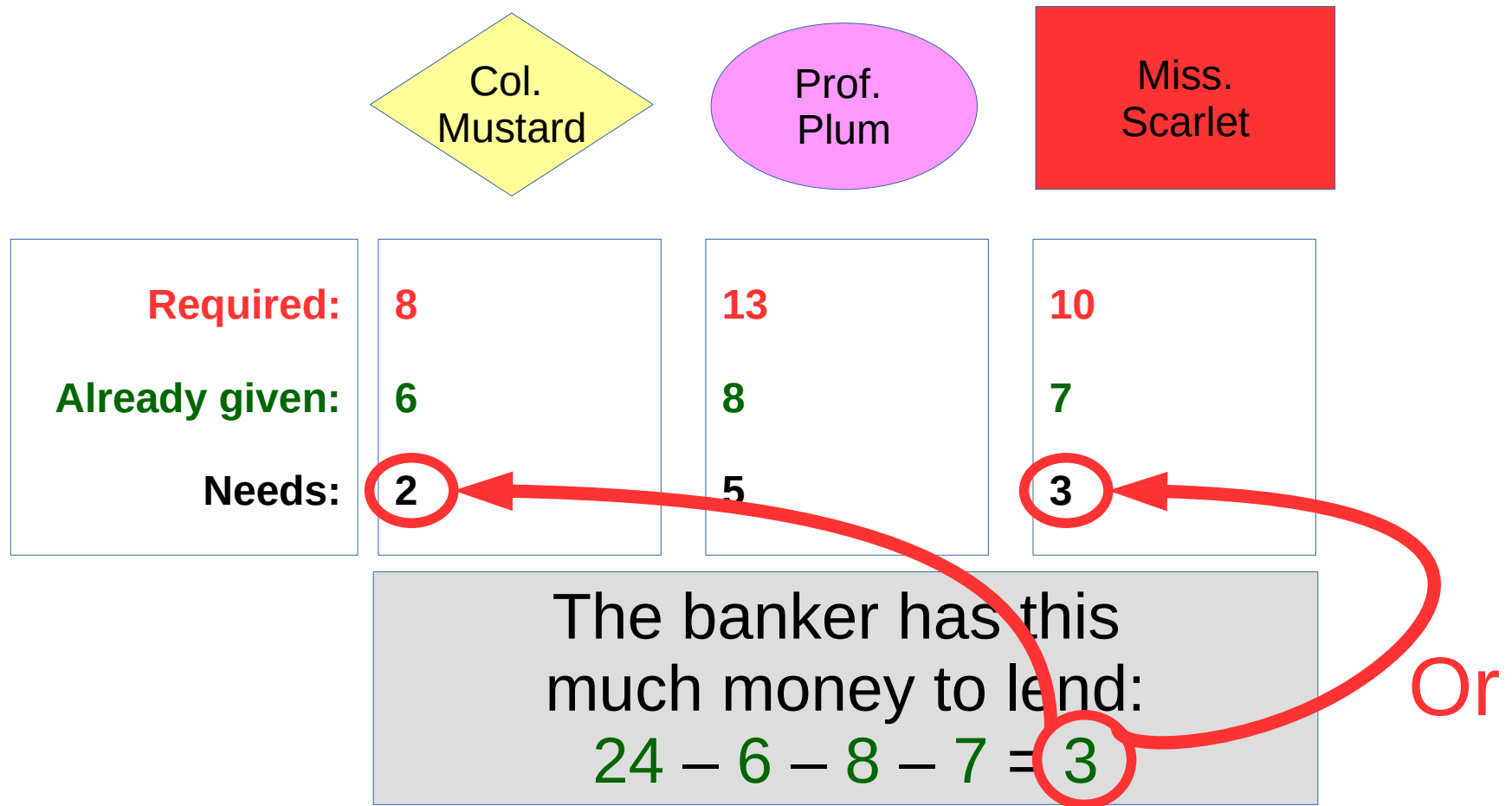
Required:	8	13	10
Already given:	6	8	7
Needs:	2	5	3

So, after lending, the
banker still has \$3 to lend.

$$24 - 6 - 8 - 7 = 3$$

- To whom can the banker give her last \$3?

A Stable (Safe State)



- The Banker can give her last \$3 to Miss Scarlet or \$2 to Col. Mustard for them to complete tasks and then repay the Bank.

The Banker Lends to Miss Scarlet

	Col. Mustard	Prof. Plum	Miss. Scarlet
Required:	8	13	10
Already given:	6	8	7
Needs:	2	5	3 - 3 = 0

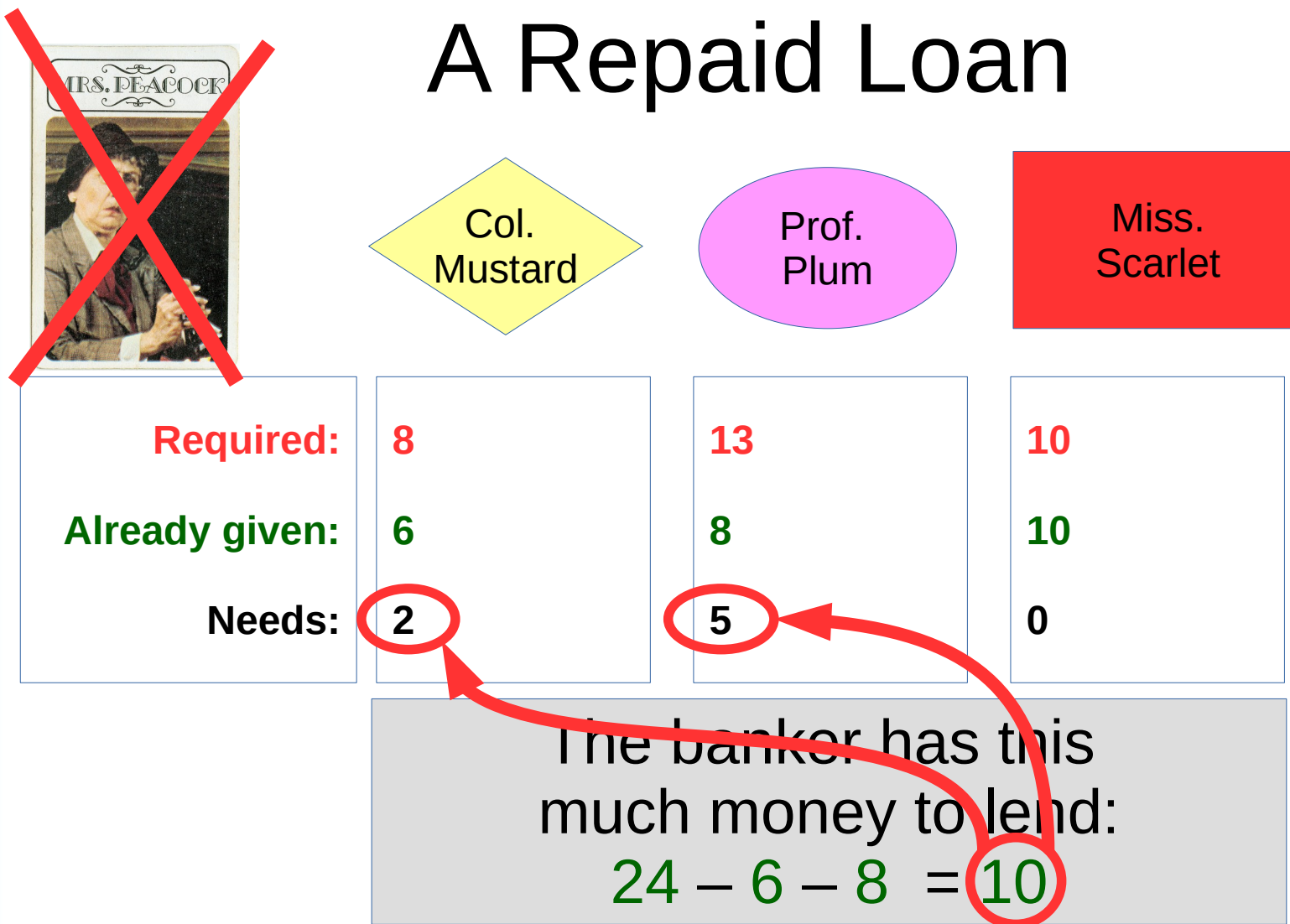
The banker has this much money to lend:
 $24 - 6 - 8 - 7 - 3 = 0$

- Remaining money lent to Miss Scarlet; No more money left to lend.
- Miss Scarlet is now able to complete her task while the others wait

Miss Scarlet Completes her Task in the Dining Room with the Candlestick

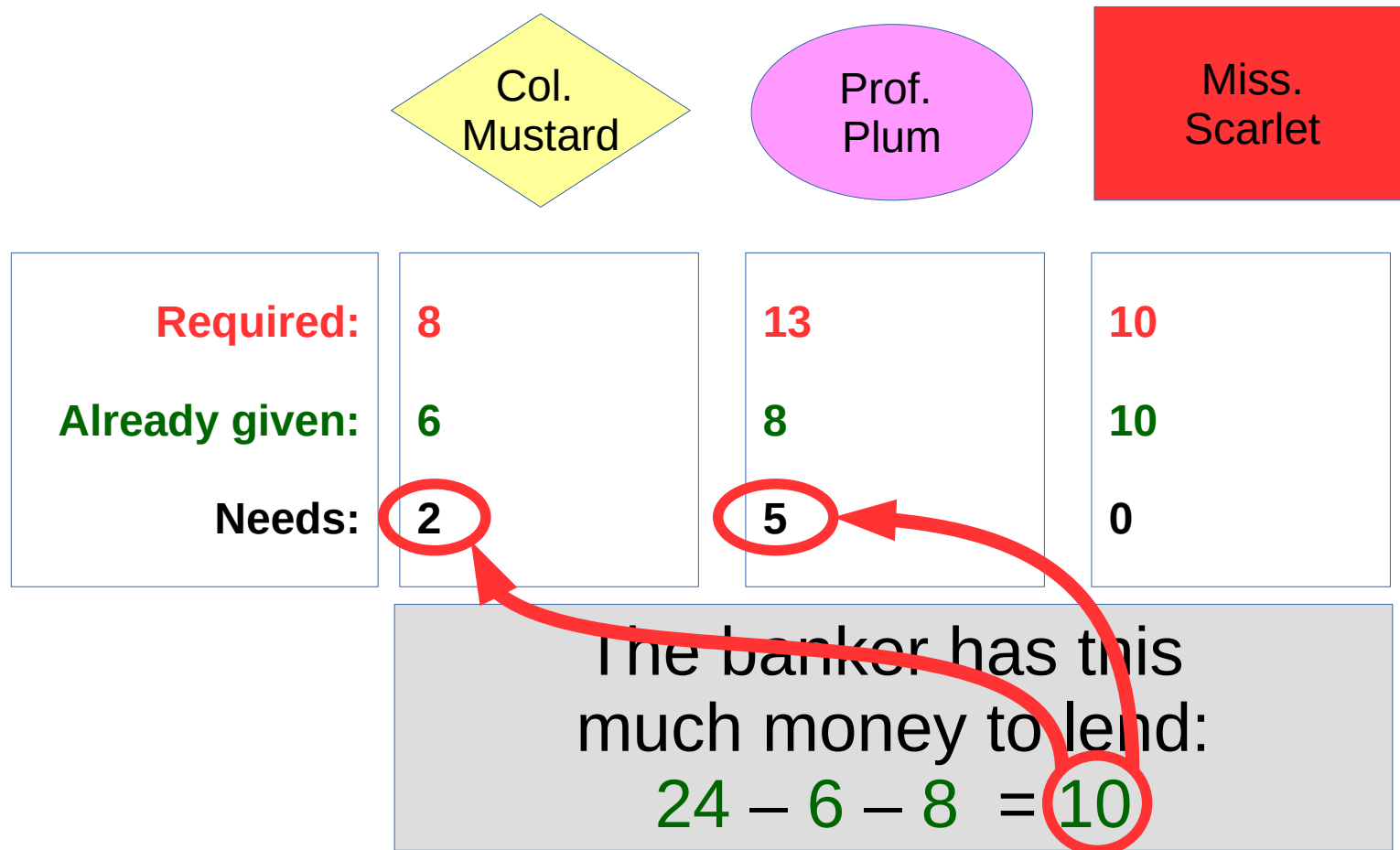


A Repaid Loan



- Miss Scarlet repays \$10 from her loan to the banker.
- Can now lend \$2 to Col Mustard and \$5 to Prof. Plum to complete their “tasks”.

Output of Algorithm: A Series of Processes



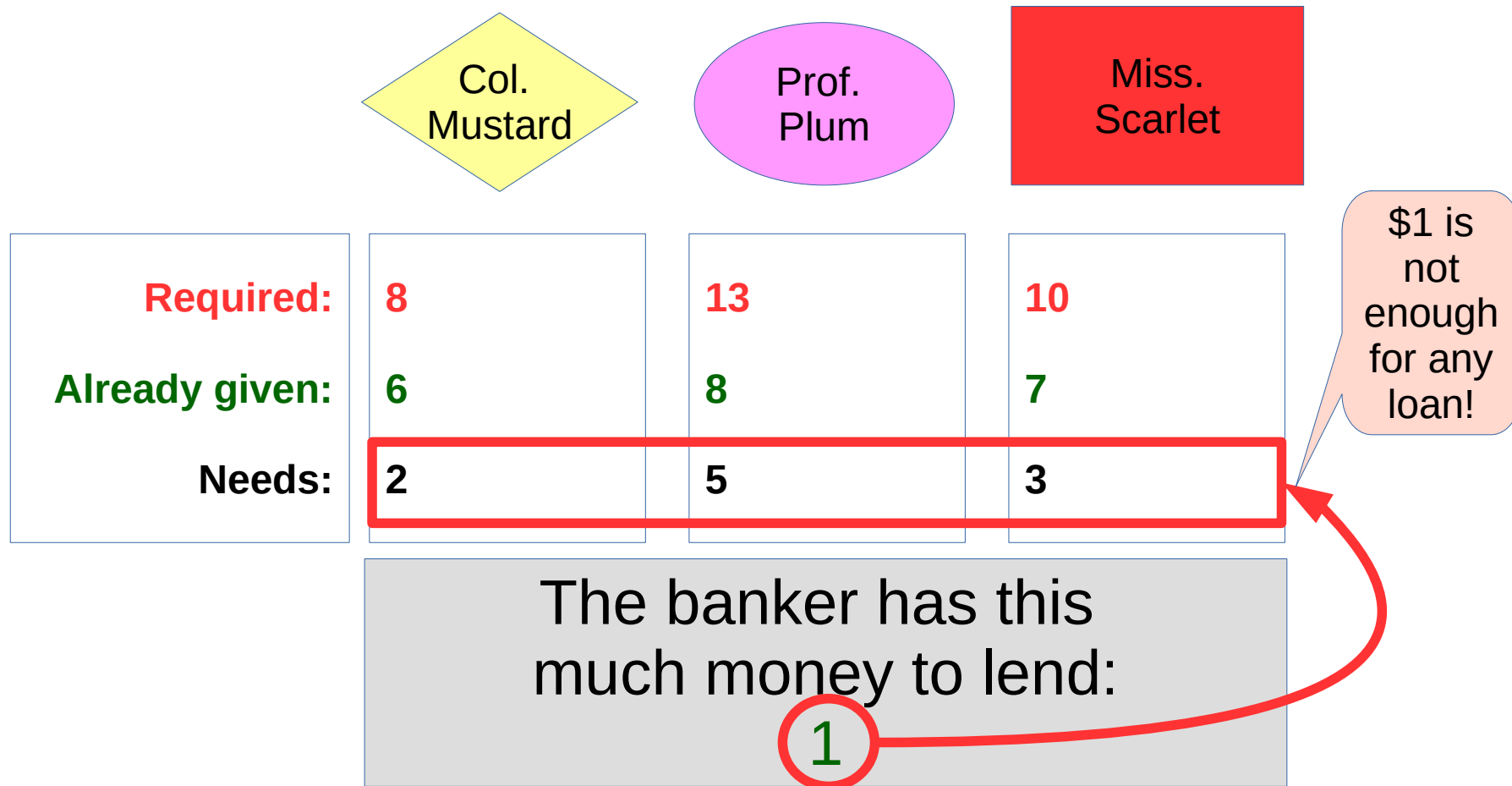
- Since the Banker can still pay, she is in a safe state.
- Can now lend \$2 to Col Mustard and \$5 to Prof. Plum to complete their "tasks". *Seq: Miss Scarlet AND (Col. Mustard AND/OR Prof Plum)*

Stable State System: When the Other Tasks Can Be Completed From Loans



- Miss Scarlet completes task first, followed by Prof Plum and then Col. Mustard
- Completed tasks result in returned loans to Banker.

Unstable State



- The Banker cannot make loans to anyone.
- An unstable state: no sequence of processes to run

Back to the Algorithm

- Processes and their Resources:
 - Printers, storage media, memory, CD Drives, etc.
- Terms (Data Structures):
 - Available resources: the types available to processes
 - **Need Matrix**: What is the need of the current processes at the current state
 - **Claim Matrix**: Max requirement of resources by processes
 - **Allocation Matrix**: at current state, number of resources being used at current state
 - $\text{Need} = \text{Claim} - \text{Allocation}$

Example

- Consider a system with three resource types (X, Y, Z) available to use by processes.
- Each resource type has five instances (each).

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

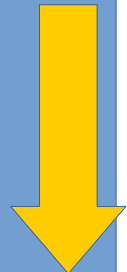
**Which process
will finish last?**

Find Available Resources

- How many resources are being used currently?

Allocation Matrix

		X	Y	Z
+	P0	1	2	1
+	P1	2	0	1
+	P2	2	2	1

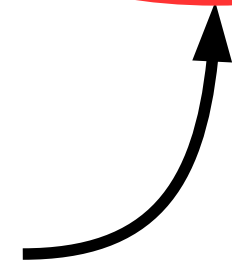


Used resources: 5 4 3

Total res: 5 5 5
Used res: - 5 4 3

0 1 2

Currently
available
resources
that we
can use



P1 is Chosen

- Currently available resources: $\{x, y, z\} \leftarrow \{0, 1, 2\}$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

We cannot satisfy
this resource: we
have 2 instances of Z,
not 3, as required.

We can satisfy
this resource

Build run sequence: $\{P1, \dots\}$

P1 Runs, Resources Returned

$$\begin{array}{r}
 \text{Allocated P1 res:} \quad 2 \ 0 \ 1 \\
 \text{Returned Curr res:} \quad + \ 0 \ 1 \ 2 \\
 \hline
 2 \ 1 \ 3
 \end{array}$$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

Returned
and available
resources
after running
P1.

P0 is Chosen

- Currently available resources: $\{x, y, z\} \leftarrow \{2, 1, 3\}$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

X	Y	Z
1	0	3
0	1	2
1	2	0

We can now
satisfy this
resource

We cannot satisfy
this resource: we
have 1 instance of Y,
not 2, as required.

Build run sequence: $\{P1, P0, \dots\}$

Resources for P0

- Currently available resources: $\{x, y, z\} \leftarrow \{1, 1, 0\}$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

Current res: 2 1 3
 Required Curr res: - 1 0 3

1 1 0

After giving P0 its required resources, we are left with these resources: $\{1\ 1\ 0\}$.

P0 Runs, Resources Returned

- Currently available resources: $\{x, y, z\} \leftarrow \{1, 1, 0\}$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

Allocated P0 res:

1 2 1

Needed res:

+ 1 0 3

2 2 4

P0 releases its
resources: $\{2, 2, 4\}$,
currently available
for other processes.

Update System Resources

- Currently available resources: $\{x, y, z\} \leftarrow \{3, 3, 4\}$

Released from p0 : 2 2 4

Current res : 1 1 0

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

3 3 4

Current
resources
available for
processes

P2 is Chosen

- Currently available resources: $\{x, y, z\} \leftarrow \{3, 3, 4\}$

Allocation Matrix

	X	Y	Z
P0	1	2	1
P1	2	0	1
P2	2	2	1

Need Matrix

	X	Y	Z
P0	1	0	3
P1	0	1	2
P2	1	2	0

We now
have enough
resources to
handle this resource

Build run sequence: $\{P1, P0, P2\}$

Sequence of Processes



- Run sequence: P1, P0, P2
- Using this sequence, we should be able to avoid deadlock.

Participation 06-07: Instructions

- Copy the directory (**06-07_part/**) into your general participation repository root for your submission. Please work on your files from the general participation directory.
- General Participation Repository
 - <https://classroom.github.com/a/S8lbI9Z5>

**Submit by
15th April at midnight**

THINK

Participation 6: Writing

- Read over these slides
- Respond to **questions in blue** in the file [06-07_part/writing/reflections.md](#)
- **Q0. What is a deadlock of processes**
- **Q1. How could they occur? Provide a general example.**
- **Q2. How can they be avoided?**
- **Q3. What is the Banker's Algorithm?**
- **Q4. How can the Banker's Algorithm help to stop deadlocks?**

Participation 7: Code Analysis

- Go to the Oracle Docs deadlock simulation tutorial
 - <https://docs.oracle.com/cd/E19205-01/820-0619/geosb/index.html>
- There are three source codes discussed. The code is already in files in your directory:
 - `06-07_part/src/din_philo.c`
 - `06-07_part/src/din_philo_fix1.c`
 - `06-07_part/src/din_philo_fix2.c`
- Compile each file using your “gcc” Docker:
 - `gcc myFile.c -pthread -o myFile`

Participation 7: Code Analysis

- Run each code and study the output.
- Explain in clear and meaningful language what each source code is *actually* demonstrating.
 - In particular, what do the ***philosophers*** and ***chopsticks*** metaphors represent?
- Place this explanation in [06-07_part/writing/codeExp.md](#)
- Submit your writing