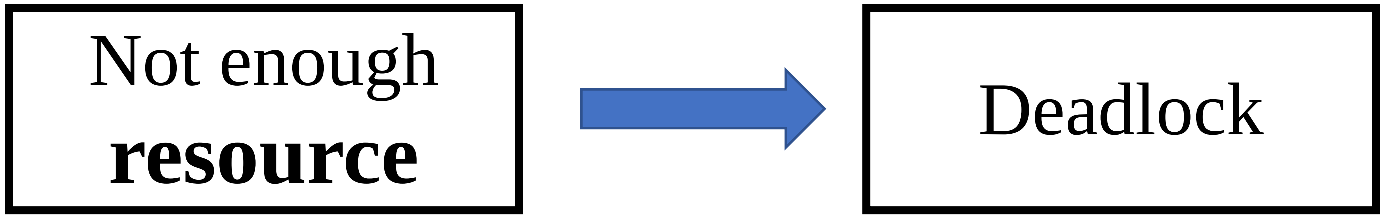
Operating Systems

Jinghui Zhong （钟竞辉）
Office： B3-515
Email ： jinghuizhong@scut.edu.cn

What is deadlock?

- **Definition**

A set of processes(e.g.,) is in a ***deadlock state*** when every process in the set is waiting for a **resource** that can only be released by another process in the set.



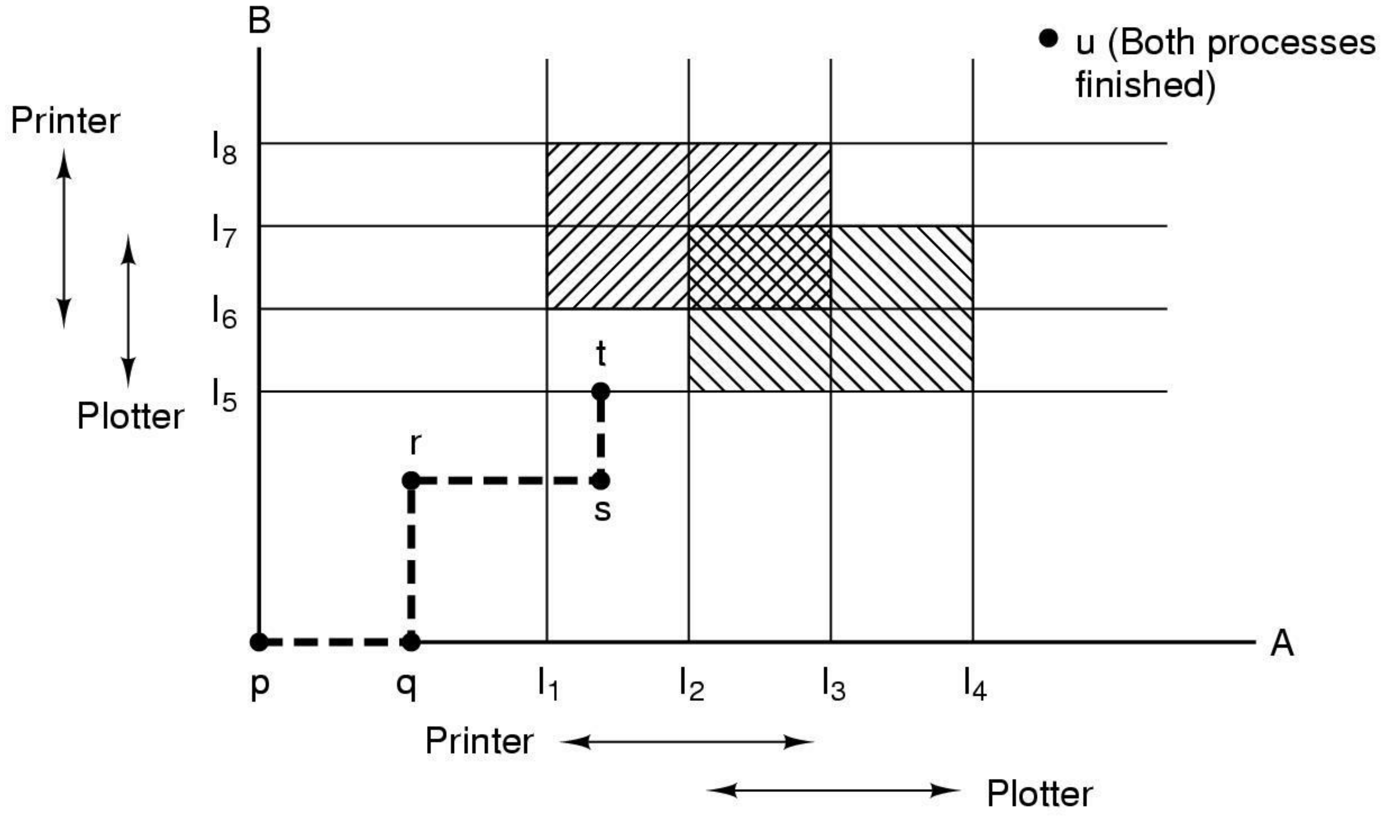
Can we avoid deadlock?

- **Motivation:** In most systems, resources are requested one at a time. It is desirable that the system can decide whether granting a resource is safe or not.

- **Question:** Is there an algorithm to make the correct decision?

- **Answer:** Yes, we can avoid deadlock, but only if certain information is available in advance.

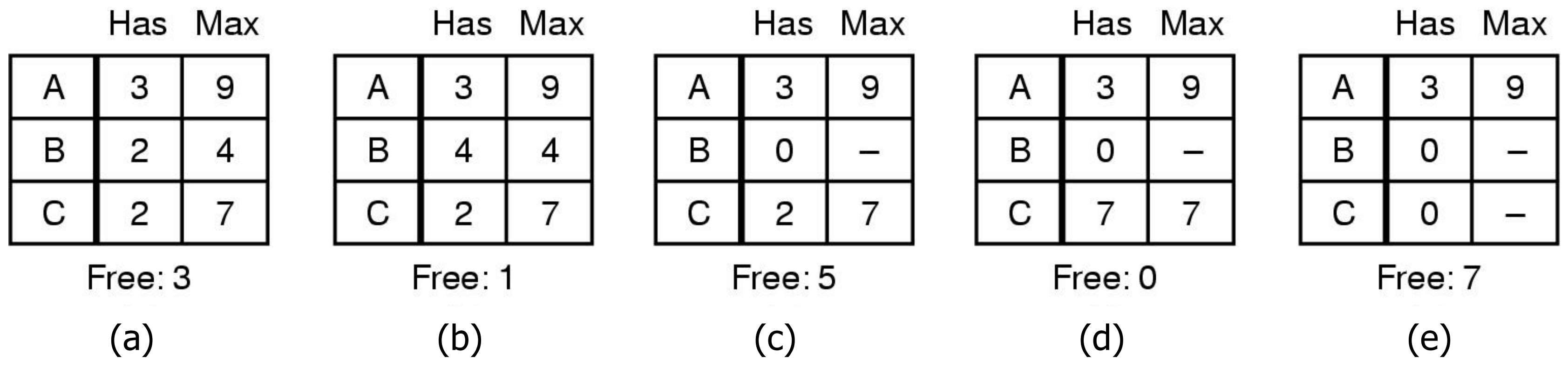
Resource Trajectories



If the system ever enters the box bounded by $[I_1, I_2, I_5, I_6]$, it will eventually deadlock. Why?

Safe and Unsafe States

- A state is safe if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources.



	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)

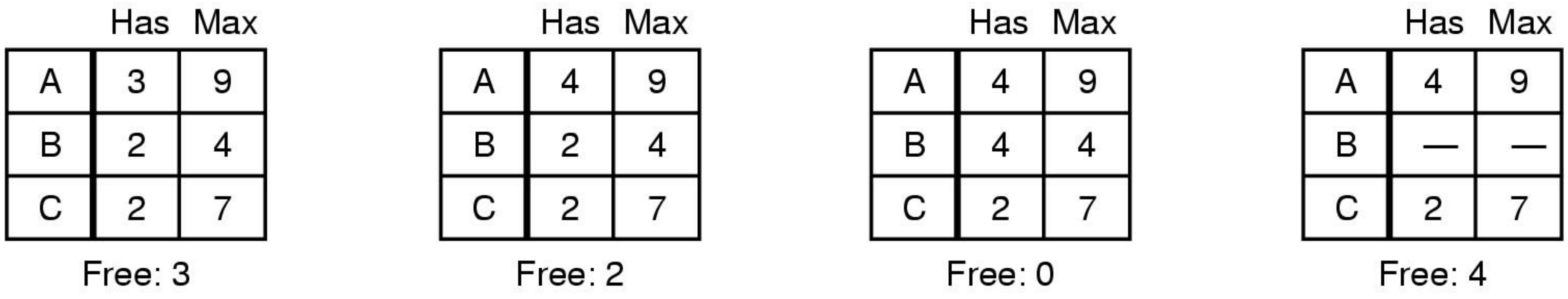
	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Demonstration that the state in (a) is safe

Safe and Unsafe States



	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	—	—
C	2	7

Free: 4

(d)

Demonstration that the state in (b) is not safe

Safe and Unsafe States

The difference between a safe state and an unsafe state is that:

from a safe state the system can guarantee that all processes will finish; But from an unsafe state, no such guarantee can be given.

The Banker's Algorithm for a Single Resource

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

Free: 10

(a) safe;

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

Free: 2

(b) safe;

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

Free: 1

(c) unsafe.

- Check to see if granting the request leads to an unsafe state. If it does, the request is denied. If granting the request leads to a safe state, it is carried out.

Banker’s Algorithm

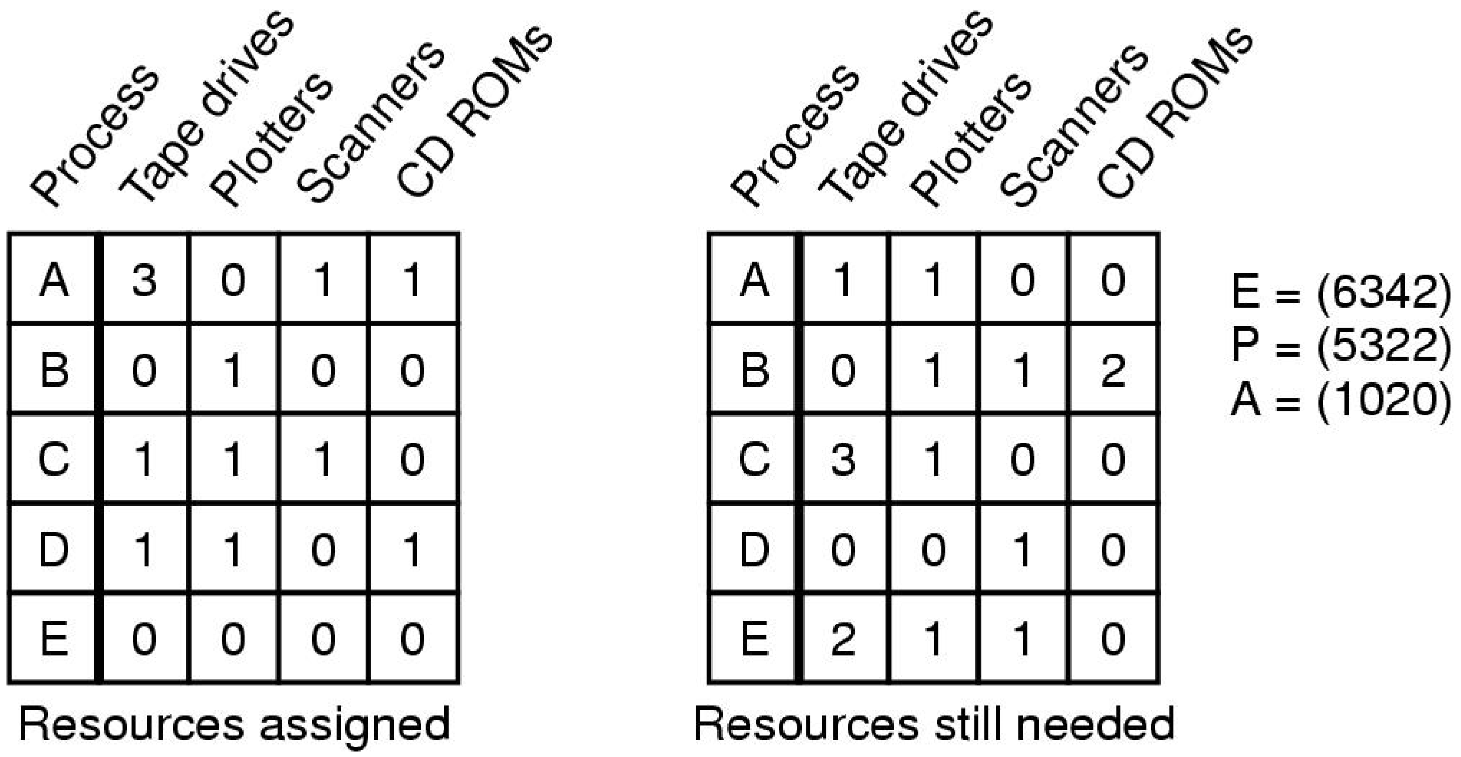
Step 1: Look for a new row in R which is smaller than A. If no such row exists the system will eventually deadlock ==> not safe.

Step 2: If such a row exists, the process may finish. mark that process (row) as terminate and add all of its resources to A.

Step 3: Repeat Steps 1 and 2 until all rows are marked == > safe state

or not marked == > not safe.

Banker's Algorithm for Multiple Resources



Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Example of banker's algorithm with multiple resources

Banker's Algorithm Summary

- **In theory :** the algorithm is wonderful.
- **In practice:** it is essentially useless because processes rarely know in advance what their maximum resource needs will be.
- Thus, in practice, few systems use the banker’s algorithm for avoiding deadlocks.

Starvation

- **Definition:** a process is perpetually denied necessary resources to process its work.

Example: A system always chooses the process with the shortest file to execute. If there is a constant stream of processes with short files, the process with long file will never be executed.

Starvation vs Deadlock

Starvation: thread waits indefinitely

Deadlock: circular waiting for resource.

What are the differences between them?

① Deadlock ➔ starvation but not vice versa.

② Starvation can end，but deadlock can’t end without external intervention.

Some notes of Deadlock

Deadlock is not always deterministic.

```
void process_A(void) {                void process_B(void) {
   down(resource_1);                      down(resource_2);
   down(resource_2);                      down(resource_1);
   use_both_resources();                  use_both_resources();
   up(resource_2);                        up(resource_1);
   up(resource_1);                        up(resource_2);
}                                           }
```

- Deadlock won’t always happen with this code.
- Have to have exactly the right time;

Deadlock Detection, Deadlock Avoidance, and Deadlock Prevention

- **Deadlock Detection**: To make sure whether there is a deadlock now.

- **Deadlock Avoidance:** to ensure the system won't enter an unsafe state. The system dynamically considers every request and decides whether it is safe to grant it at this point.

- **Deadlock Prevention:** to ensure that at least one of the necessary conditions for deadlock can never hold.

Necessary Conditions for Deadlock

Mutual exclusion

Resources are held by processes in a non-sharable (exclusive) mode.

Hold and Wait

A process holds a resource while waiting for another resource.

No Preemption

There is only voluntary release of a resource - nobody else can make a process give up a resource.

Circular Wait

Process A waits for Process B waits for Process C .... waits for Process A.

ALL of these four **must** happen simultaneously for a deadlock to occur.

Operating System Structure

- **Jigsaw reading**

Describe the key ideas of the following methods

① **Attacking the mutual-exclusion condition (Group 1~4)**

② **Attacking the hold-and-wait condition (Group 5~8)**

③ **Attacking the No-Preemption condition (Group 9~12)**

④ **Attacking the Circular wait condition (Group 13-18)**

Attacking the Mutual Exclusion Condition

- **No resource were assigned exclusively to a single process.**
- **Problem?**

This method is generally impossible, because the mutual-exclusion condition must hold for non-sharable resources. e.g., a printer can not be shared by multiple processes simultaneously.

Attacking the Hold and Wait Condition

- **Require processes to request resources before starting**

  a process never has to wait for what it needs

  e.g. In the dining philosophers’ problem, each philosopher is required to pick up both forks at the same time. If he fails, he has to release the fork(s) (if any) he has acquired.

- **Problems?**

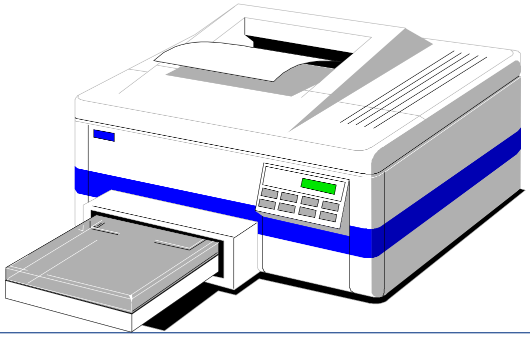
  It is difficult to know required resources at start of run

Attacking the No Preemption Condition

- **If a process is holding some resources and requests another resource that cannot be allocated to it, then all resources are released.**
- **Problem?**

The method can be applied to resources whose state can be save and restored later, e.g., memory.

It cannot be applied to resources such as printers.

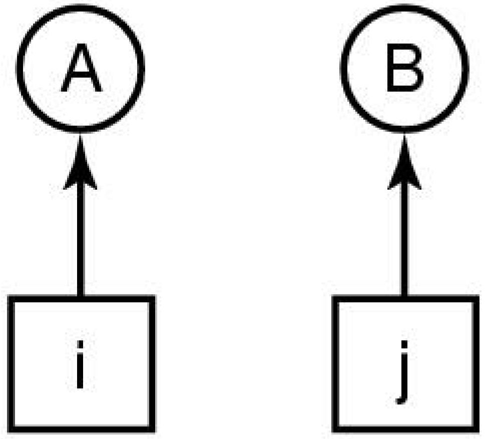


Attacking the Circular Wait Condition

- A process is entitled only to a single resource at any moment.
- Impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

(a)



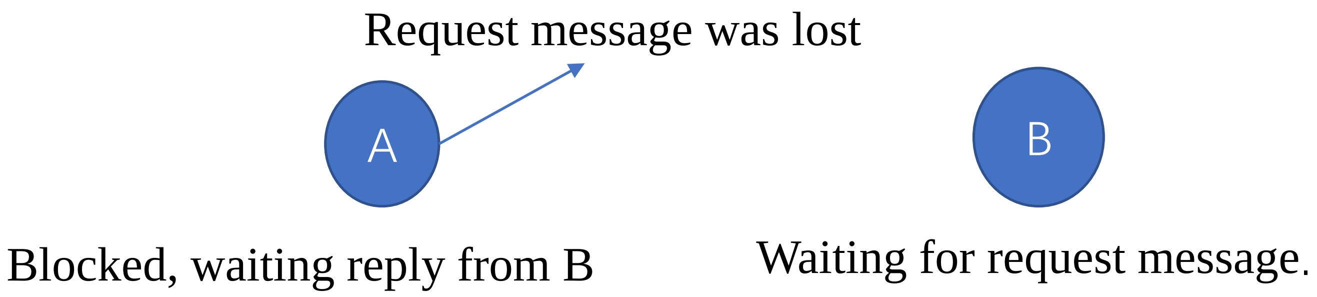
(b)

Other Issues: Two-Phase Locking

- Phase One
  - ✓process tries to lock all records it needs, one at a time
  - ✓if needed record found locked, restart
  - ✓(no real work done in phase one)
- If phase one succeeds, it starts second phase,
  - ✓performing updates
  - ✓releasing locks
- Similar to requesting all resources at once

Non-resource Deadlocks

- Communication Deadlock
  - ✓E.g., each is waiting for the other to do some task



- Can happen with semaphores

```
void process_A(void) {
  down(resource_1);
  down(resource_2);
  use_both_resources();
  up(resource_2);
  up(resource_1);
}
```

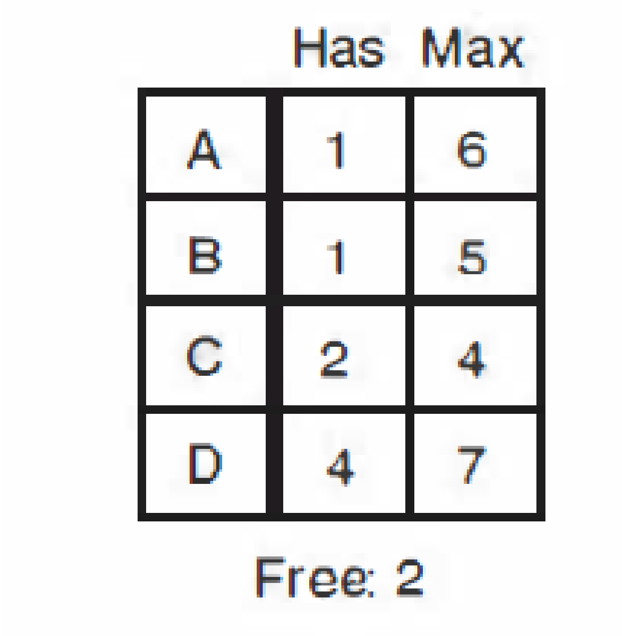
```
void process_B(void) {
  down(resource_2);
  down(resource_1);
  use_both_resources();
  up(resource_1);
  up(resource_2);
}
```

Check Points

The banker’s algorithm is being run in a system with *m* resource classes and *n* processes? In the limit of large *m* and *n*, the number of operations that must be performed to check a state for safety is proportional to $m^a n^b$. What are the values of *a* and *b*?

Check Points

If *D* asks for one more unit, does this lead to a safe state or an unsafe one? What if the request came from *C* instead of *D*?



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

Free: 2

Check Points

A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.

Check Points

A computer has six tape drives, with *n* processes competing for them. Each process may need two drives. For which values of *n* is the system deadlock free?

Check Points

Consider p processes each needing a maximum of m resources and a total of r resources available. What condition must hold to make the system deadlock free?

Check Points

- Consider a system consisting of *m* resources of the same type being shared by *n* processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:

a) The maximum need of each process is between one resource and *m* resources.

b) The sum of all maximum needs is less than $m + n$.