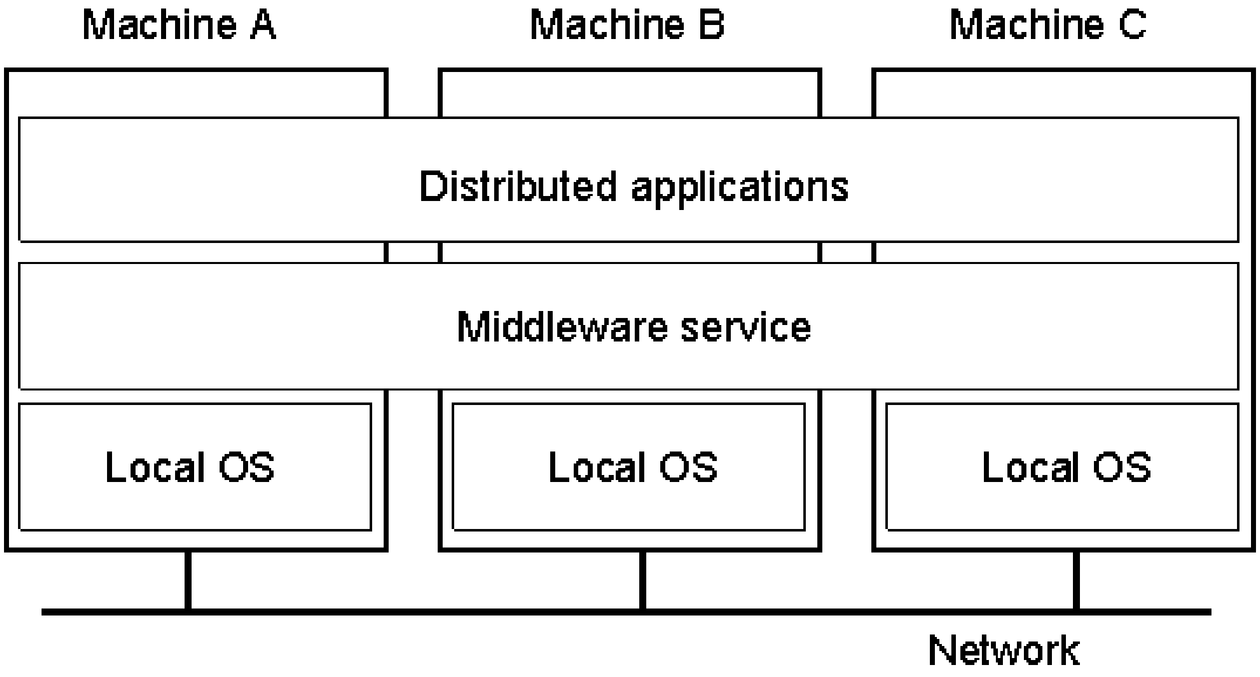
Definition of a Distributed System

- A distributed system is:

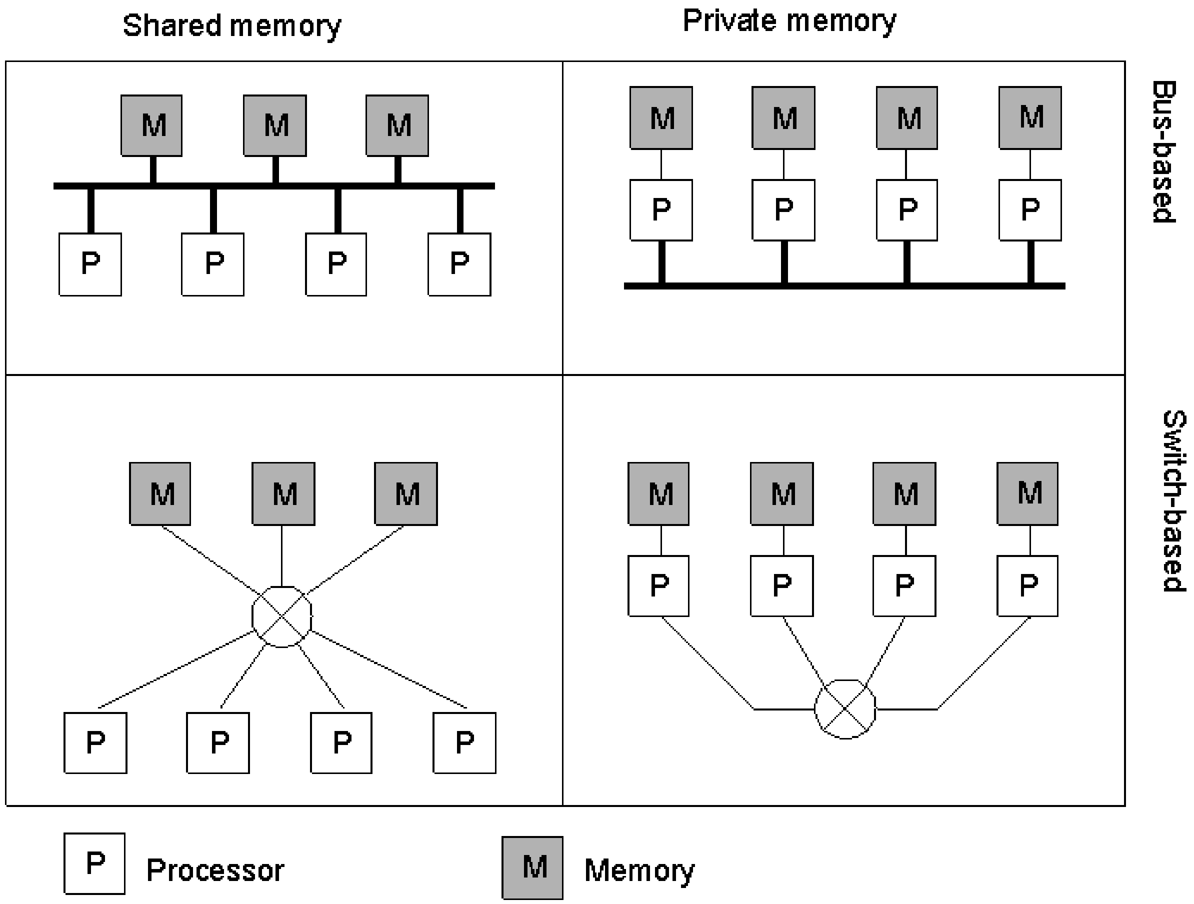
A collection of independent computers that appears to its users as a single coherent system.

Distributed System Organization



- Example of middleware-based organization of a distributed system.
- The thickness of the middleware layer can range from extremely thin to very thick depending on the degree of integration of a particular system

Hardware Concepts



Different basic organizations and memories in distributed computer systems

Group Discussion

- Topics to discuss:
  - Name one or two Distributed Systems based on your impression or past experiences
  - What are good things about these systems?
  - Anything in these systems demands improvement?

- Format:
  - 4-5 students form a group
  - Feel free to move around
  - Discuss for 2-3 minutes
  - One representative from each group will talk about your ideas

Issues of Distributed Systems

- Distributed systems introduce a whole new set of design issues w.r.t traditional system design
- Scalability
- Transparency
- On multi-computers:
  - Lack of common address space
  - Lack of common clock

Scalability Problems

Concept	**Example**
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Examples of scalability limitations.

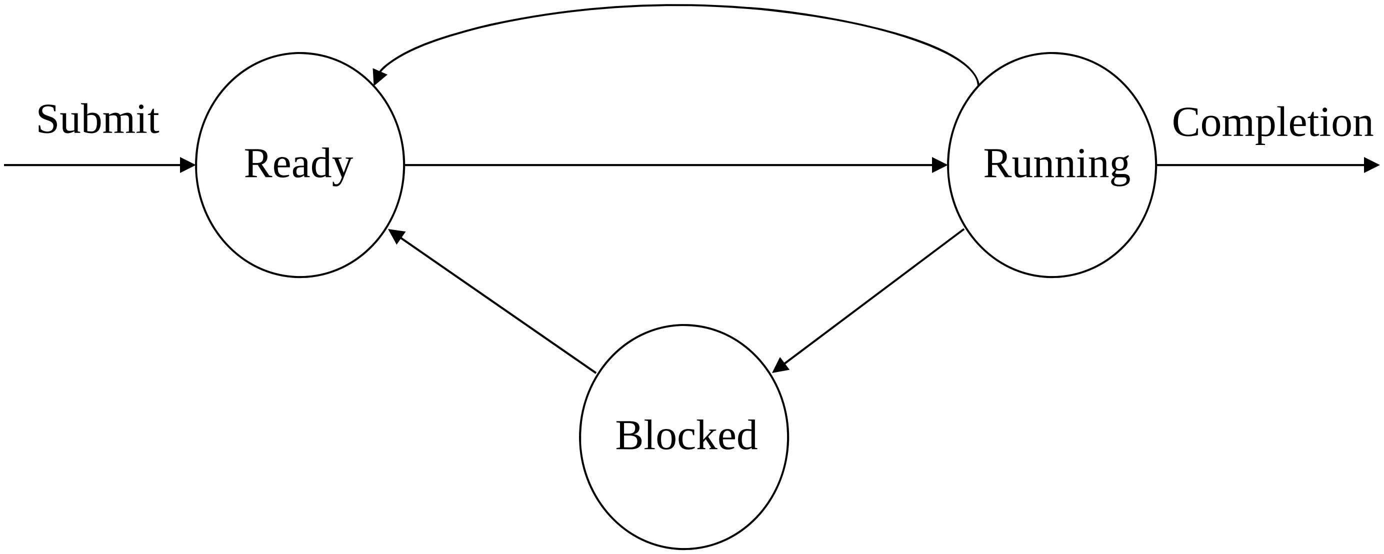
Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that there may be multiple copies of a resource
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Different forms of transparency in a distributed system.

Concept of a process

- In the context of this course a process is a program whose execution is in progress.
- States of a process: running, ready, blocked



Concurrent processes

- In a multiprocessor system two or more processes can be in execution at the same time
  - physical concurrency - as opposed to logical concurrency achieved by interleaving process execution
- Concurrent processes interaction:
  - shared variables
  - message passing
- If no interaction, their execution is functionally the same as their serial execution
- Group discussion:
  - Real life analogies? (Focus on concurrency, interaction, shared resources, any potential issues?)

The critical section problem

- A critical section is a code segment of a concurrent process in which a shared resource is accessed
- Concurrent access to a shared variable is potentially dangerous
  - example: if a=0, what is the result of the command a=a+1 executed simultaneously by processes A and B?
  - a common solution is the mutual exclusion i.e. serialization of accesses

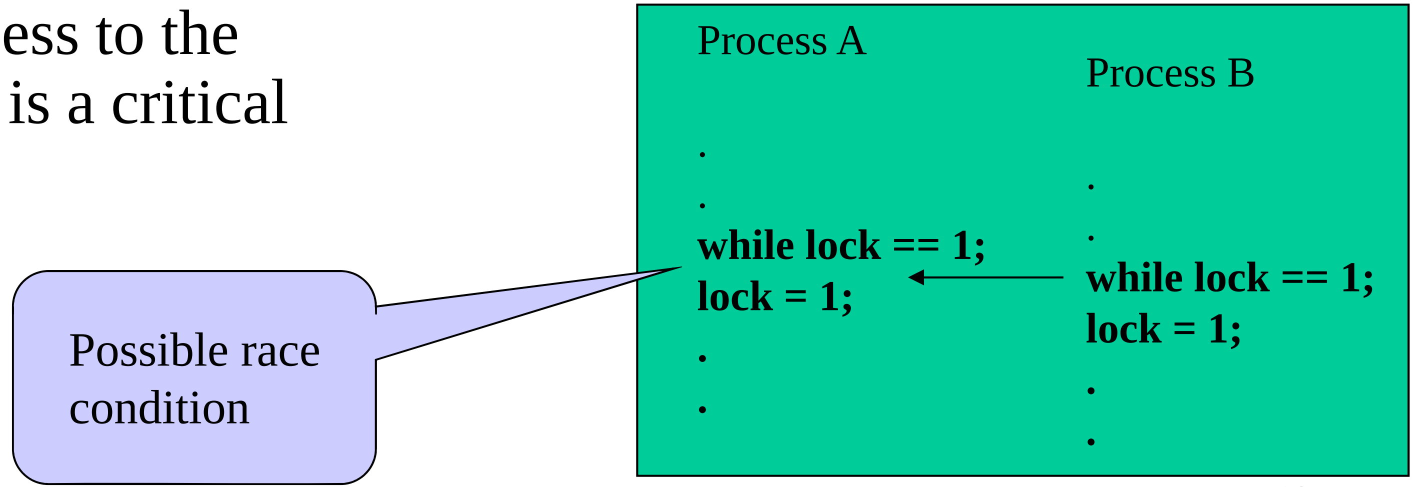
Early Solutions

- Busy Waiting
  - Wastes cycles
- Disabling Interrupts
  - Only applicable to uniprocessor
- A special test-and-set instruction

Example of busy waiting on a lock (1/2)

- One could think of using a variable as a flag to be checked upon entering a critical section ...
- … but access to the lock itself is a critical section!

```
Shared integer lock = 0;
Process i
            .
            .
            while lock == 1;
            lock = 1;
            execute CS;
            lock = 0;
            .
```



```
Process A
                          Process B

.
.                         .
while lock == 1;          .
lock = 1;           <-    while lock == 1;
.                         lock = 1;
.                         .
                          .
```

Possible race condition

Example of busy waiting on a lock (2/2)

- The correct implementation uses a test-and-set instruction to avoid race conditions

Semantics of test-and-set instruction

```
int test-and-set (int a) {
    int rv = a;
    a = 1;
    return rv;
}
```

Correct lock implementation

```
Process A

Shared integer lock = 0;
.
.
While( test-and-set(lock) ==1)
   ;
.
.
```