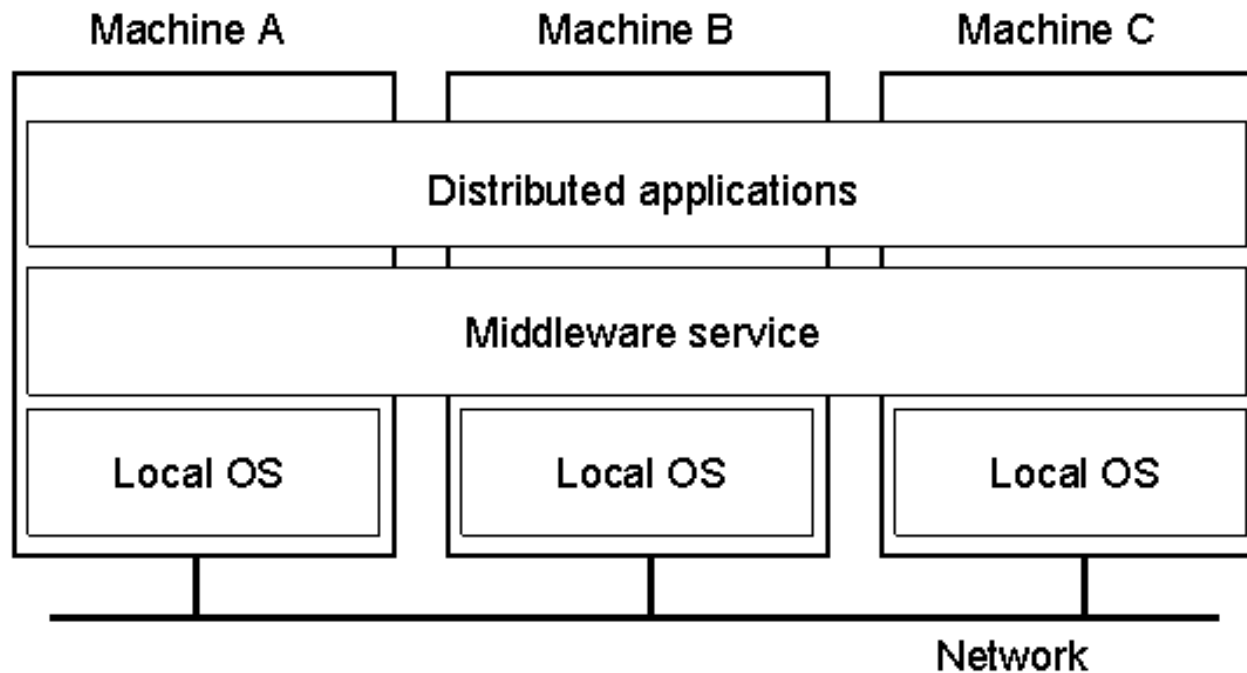
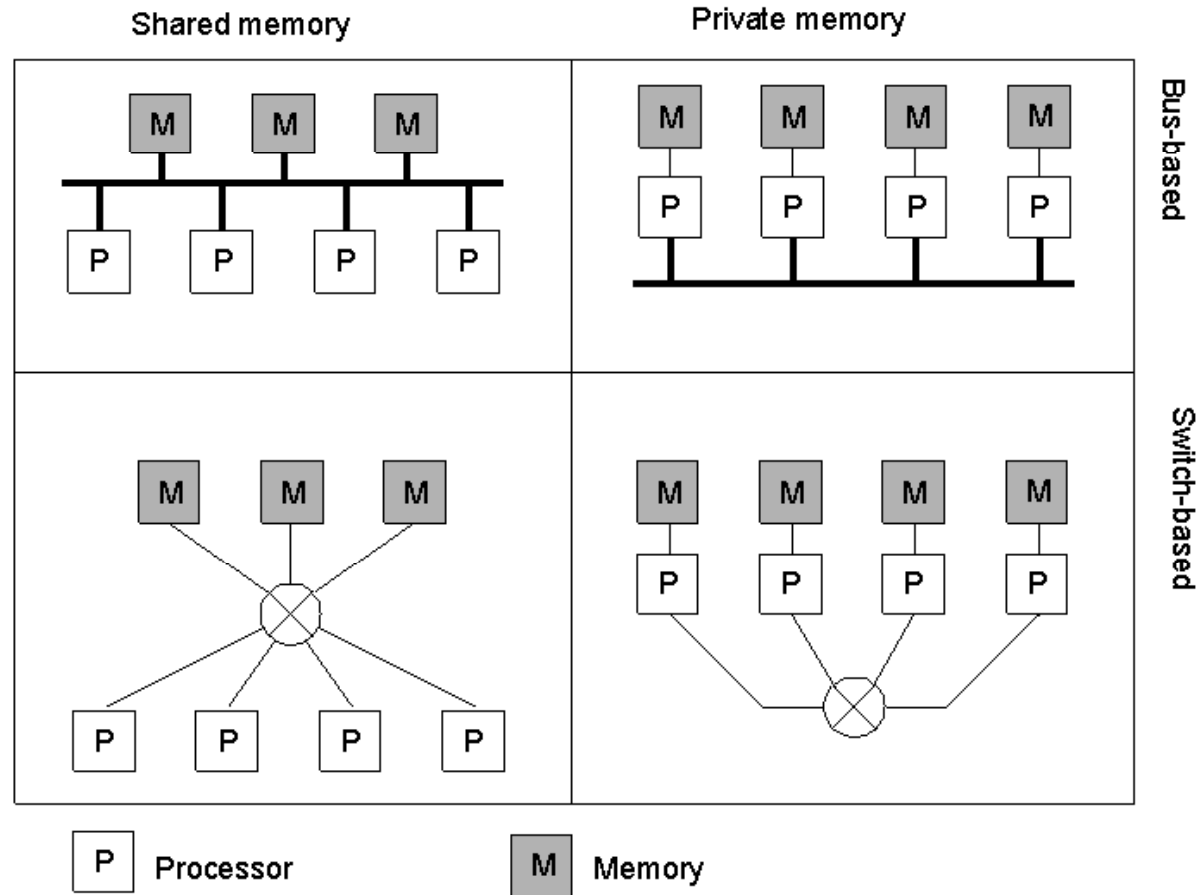


# Distributed System Organization



- 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



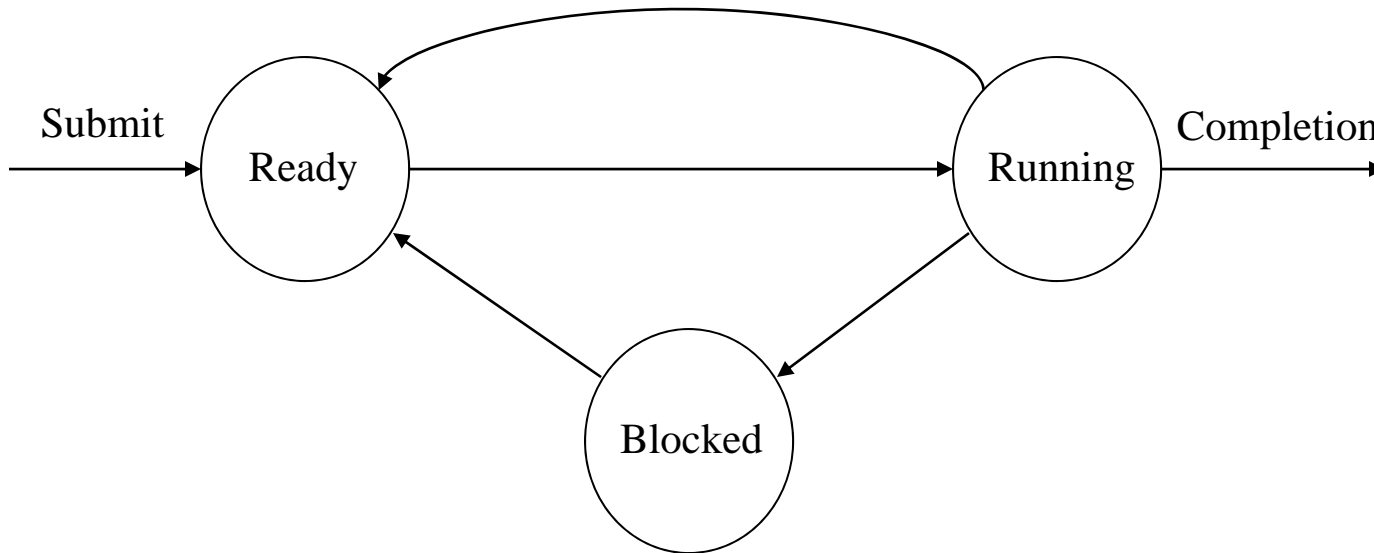
Different basic organizations and memories in distributed computer systems

# Distributed systems issues

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

# 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 processor interaction:
  - shared variables
  - message passing
- If they don't interact their execution is functionally the same as their serial execution

# 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 uniprocessors
- 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 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;	←	while lock == 1;
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

Semantic 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)  
    ;  
.  
.
```

# Locks: pros and cons

- Pros:
  - simple and fast
  - ubiquitous: every processor has a test-and-set or equivalent operation
- Cons:
  - busy waiting is wasteful of resources (CPU cycles, memory bandwidth)

# Semaphores - definition

- Proposed by Dijkstra, it was the first high level constructs used to synchronize concurrent processes.
- A semaphore  $S$  is an integer variable on which two atomic operations are defined,  $P(S)$  and  $V(S)$ , and with an associated queue.
- $P$  and  $V$  semantic:

```
P(S) : if  $S \geq 1$  then  $S := S - 1$   
      else <block and enqueue the process>;
```

```
V(S) : if <some process is blocked on the queue> then  
      <unblock a process>  
      else  $S := S + 1$ ;
```

# Semaphores - properties

- The P operation may block a process, but V does not
- Two type of semaphores
  - binary: intial value is 1
  - resource counting: any initial value
- P and V are atomic operations

```
P(S): if  $S \geq 1$  then  $S := S - 1$   
      else <block and enqueue the process>;
```

```
V(S): if <some process is blocked on the queue> then  
      <unblock a process>  
      else  $S := S + 1$ ;
```

# Example of use

**Shared var mutex: semaphore = 1;**

**Process *i***

**begin**

•

•

**P(mutex);**

*execute CS;*

**V(mutex);**

•

•

**End;**

# Other synchronization problems

- Semaphore can be used in other synchronization problems besides Mutual Exclusion
- The Producer-Consumer problem
  - a finite buffer pool is used to exchange messages between producer and consumer processes
- The Readers-Writers Problem
  - reader and writer processes accessing the same file
- The Dining Philosophers Problem
  - five philosophers competing for a pair of forks

