Processes and Threads

Learning Outcomes

- An understanding of fundamental concepts of processes and threads

THE UNIVERSITY OF
NEW SOUTH WALES

Major Requirements of an Operating System

- Interleave the execution of several processes to maximize processor utilization while providing reasonable response time

- Allocate resources to processes

- Support interprocess communication and user creation of processes

THE UNIVERSITY OF
NEW SOUTH WALES

Processes and Threads

- ## Processes:
    - Also called a task or job
    - Execution of an individual program
    - "Owner" of resources allocated for program execution
    - Encompasses one or more threads

- ## Threads:
    - Unit of execution
    - Can be traced
        - list the sequence of instructions that execute
    - Belongs to a process

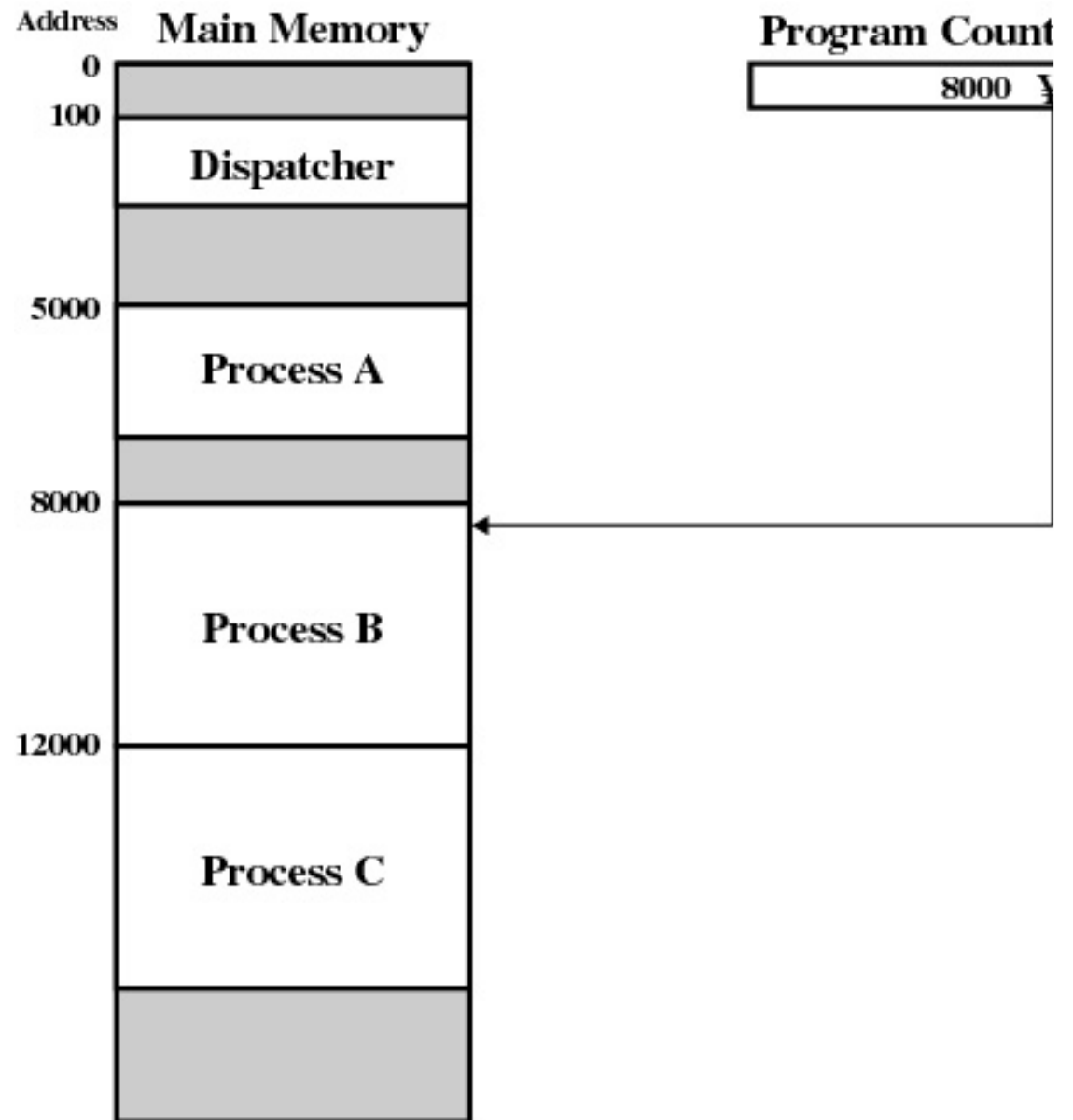Execution snapshot of three single-threaded processes (No Virtual Memory)

Figure 3.1 Snapshot of Example Execution (Figure 3. at Instruction Cycle 13

Logical Execution Trace

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A　　　**(b) Trace of Process B**　　　**(c) Trace of Process C**

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.2　Traces of Processes of Figure 3.1

Combined Traces
(Actual CPU Instructions)

What are the shaded sections?

1	5000		27	12004
2	5001		28	12005
3	5002		----------------------Time out	
4	5003		29	100
5	5004		30	101
6	5005		31	102
----------------------Time out			32	103
7	100		33	104
8	101		34	105
9	102		35	5006
10	103		36	5007
11	104		37	5008
12	105		38	5009
13	8000		39	5010
14	8001		40	5011
15	8002		----------------------Time out	
16	8003		41	100
------------------I/O request			42	101
17	100		43	102
18	101		44	103
19	102		45	104
20	103		46	105
21	104		47	12006
22	105		48	12007
23	12000		49	12008
24	12001		50	12009
25	12002		51	12010
26	12003		52	12011
			----------------------Time out	

100 = Starting address of dispatcher program

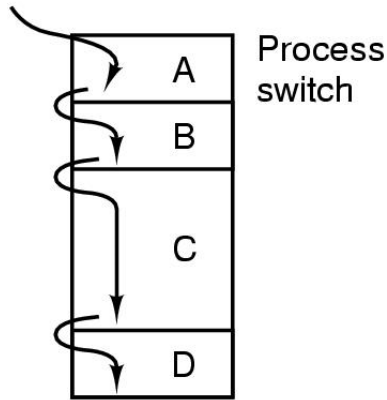shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

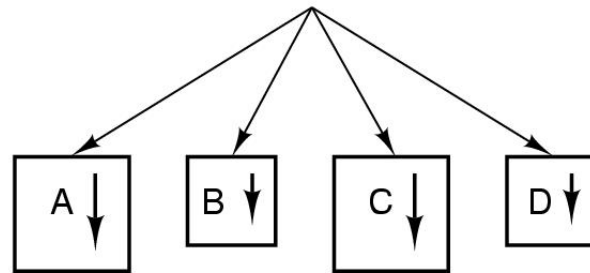**Figure 3.3 Combined Trace of Processes of Figure 3.1**
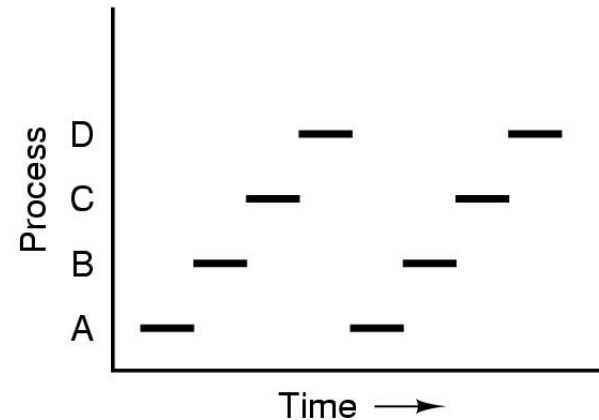
Summary: The Process Model

One program counter

Process switch

A
B
C
D

(a)

Four program counters

A
B
C
D

(b)

Process
D
C
B
A

Time

(c)

- Multiprogramming of four programs
- Conceptual model of 4 independent, sequential processes (with a single thread each)
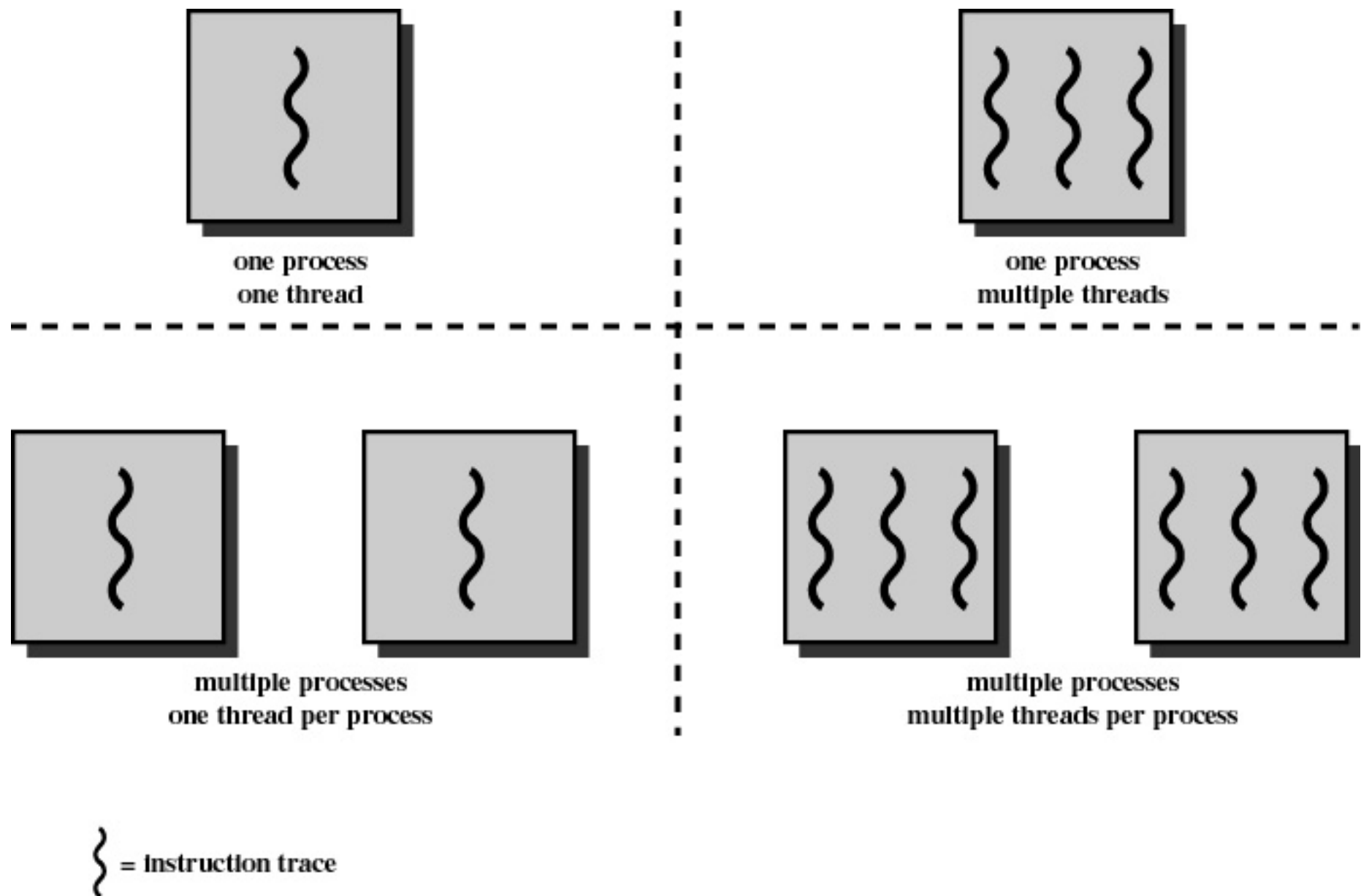- Only one program active at any instant

one process
one thread

one process
multiple threads

multiple processes
one thread per process

multiple processes
multiple threads per process

{ = instruction trace

Figure 4.1   Threads and Processes [ANDE97]

Process and thread models of selected OSes

- Single process, single thread
  - MSDOS
- Single process, multiple threads
  - OS/161 as distributed
- Multiple processes, single thread
  - Traditional unix
- Multiple processes, multiple threads
  - Modern Unix (Linux, Solaris), Windows

Note: Literature (incl. Textbooks) often do not cleanly distinguish between processes and threads (for historical reasons)

Process Creation

Principal events that cause process creation

1. System initialization

   - Foreground processes (interactive programs)

   - Background processes

     - Email server, web server, print server, etc.
     - Called a *daemon* (unix) or *service* (Windows)

2. Execution of a process creation system call by a running process

   - New login shell for an incoming telnet/ssh connection

3. User request to create a new process

4. Initiation of a batch job

Note: Technically, all these cases use the same system mechanism to create new processes.

Process Termination

Conditions which terminate processes
1. Normal exit (voluntary)
2. Error exit (voluntary)
3. Fatal error (involuntary)
4. Killed by another process (involuntary)

Process/Thread States
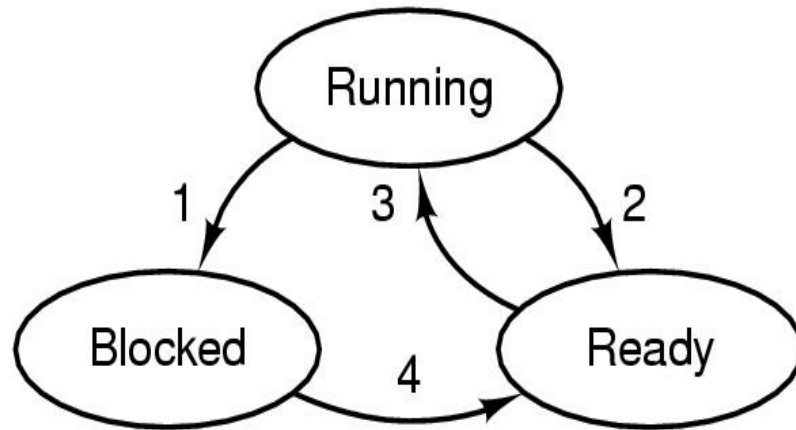
Running

1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Blocked   Ready

- Possible process/thread states
  - running
  - blocked
  - ready
- Transitions between states shown

Some Transition Causing Events

Running ▷Ready

- Voluntary `Yield()`

- End of timeslice

Running ▷Blocked

- Waiting for input

  - File, network,

- Waiting for a timer (alarm signal)

- Waiting for a resource to become available

THE UNIVERSITY OF
NEW SOUTH WALES

Dispatcher

- **Sometimes also called the *scheduler***
  - The literature is also a little inconsistent on with terminology.

- **Has to choose a *Ready* process to run**
  - How?
  - It is inefficient to search through all processes

The Ready Queue
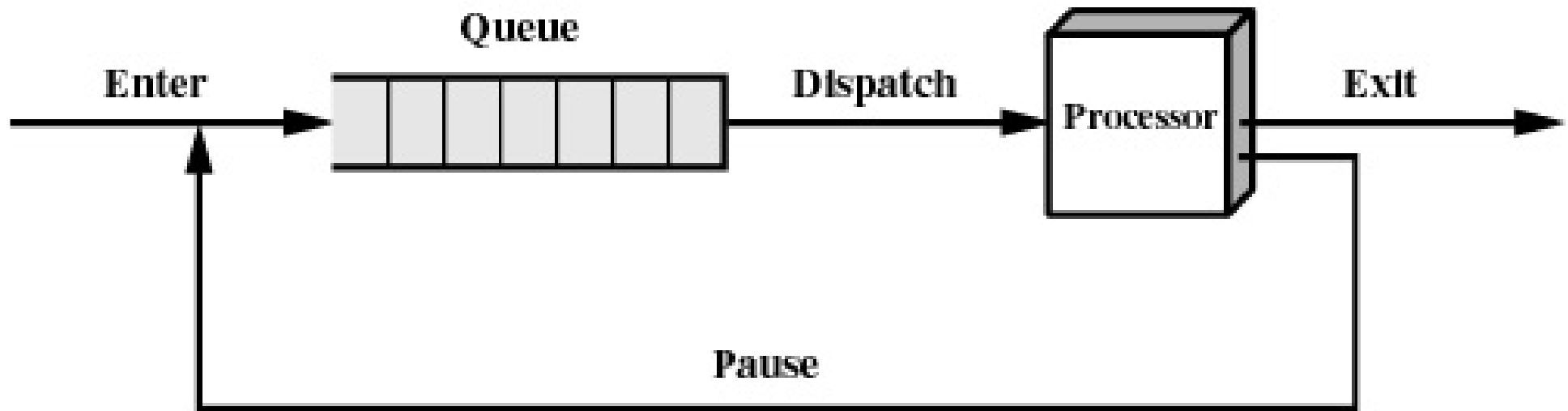
(b) Queuing diagram

What about blocked processes?

- When an *unblocking* event occurs, we also wish to avoid scanning all processes to select one to make *Ready*
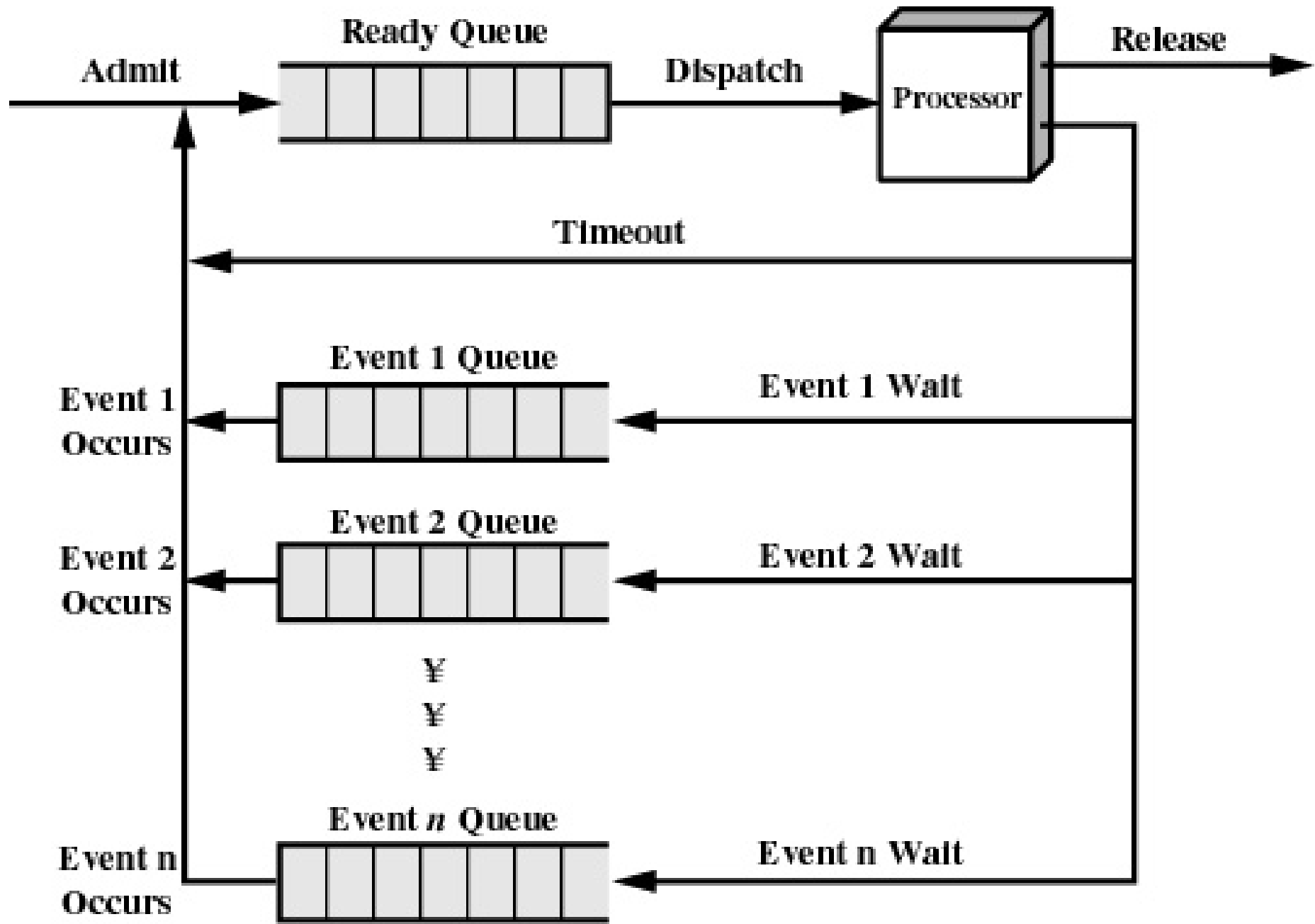
THE UNIVERSITY OF
NEW SOUTH WALES

Using Two Queues
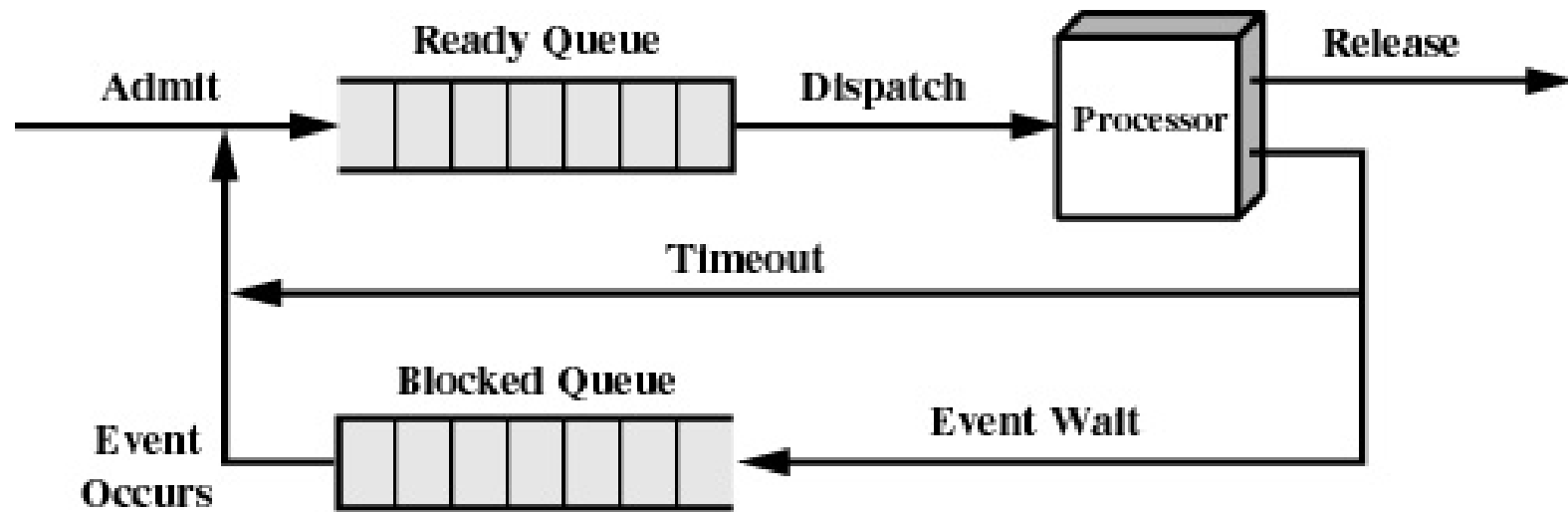
(a) Single blocked queue

(b) Multiple blocked queues

Implementation of Processes

- A processes' information is stored in a *process control block* (PCB)

- The PCBs form a *process table*
  - Sometimes the kernel stack for each process is in the PCB
  - Sometimes some process info is on the kernel stack
    - E.g. registers in the *trapframe* in OS/161
  - Reality is much more complex (hashing, chaining, allocation bitmaps,…)

P7
P6
P5
P4
P3
P2
P1
P0

THE UNIVERSITY OF
NEW SOUTH WALES

Implementation of Processes

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Example fields of a process table entry

THE UNIVERSITY OF
NEW SOUTH WALES

Threads
The Thread Model
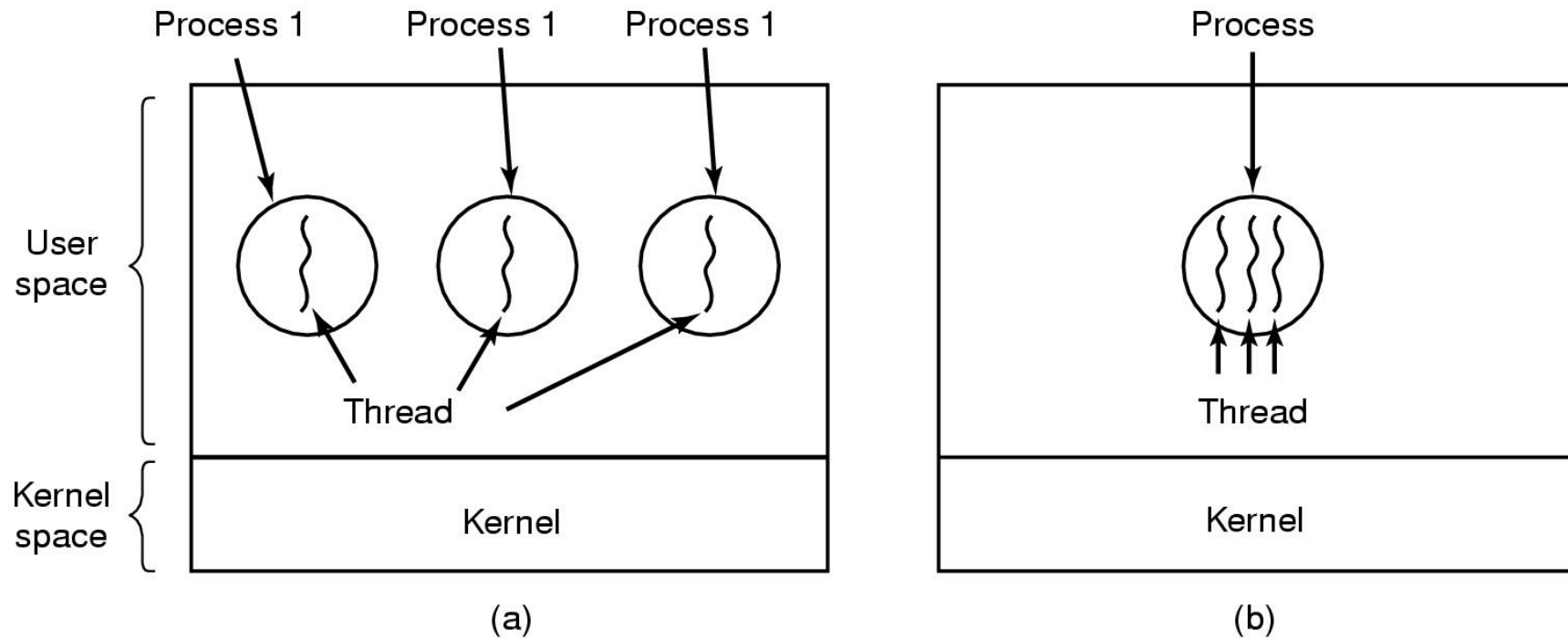
(a) Three processes each with one thread
(b) One process with three threads

The Thread Model – Separating execution from the environment.

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- Items shared by all threads in a process
- Items private to each thread
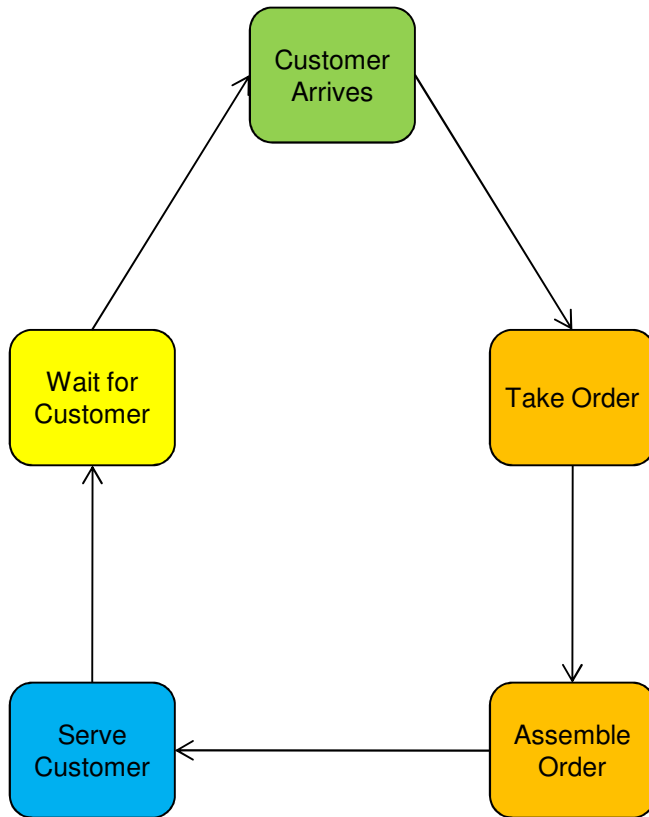
Threads Analogy
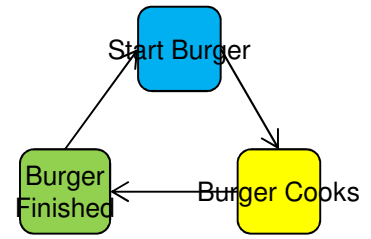

The Hamburger Restaurant

Single-Threaded Restaurant
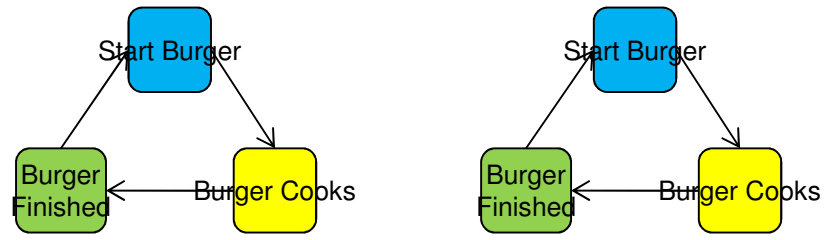
Customer Arrives → Take Order → Start Fries → Fries Cook

Fries Cook → Fries Finish → Start Burger → Burger Cooks

Burger Cooks → Burger Finished → Assemble Order → Serve Customer → Wait for Customer → Customer Arrives

Blocking operations delay all activities

THE UNIVERSITY OF NEW SOUTH WALES

Multithreaded Restaurant

Customer Arrives → Take Order → Assemble Order → Serve Customer → Wait for Customer → Customer Arrives

Start Fries → Fries Cook → Fries Finish → Start Fries

Start Burger → Burger Cooks → Burger Finished → Start Burger

Note: Ignoring synchronisation issues for now

THE UNIVERSITY OF
NEW SOUTH WALES

26

Multithreaded Restaurant with more worker threads

THE UNIVERSITY OF NEW SOUTH WALES
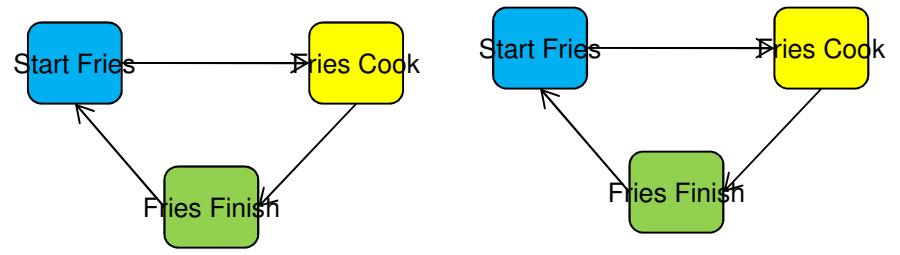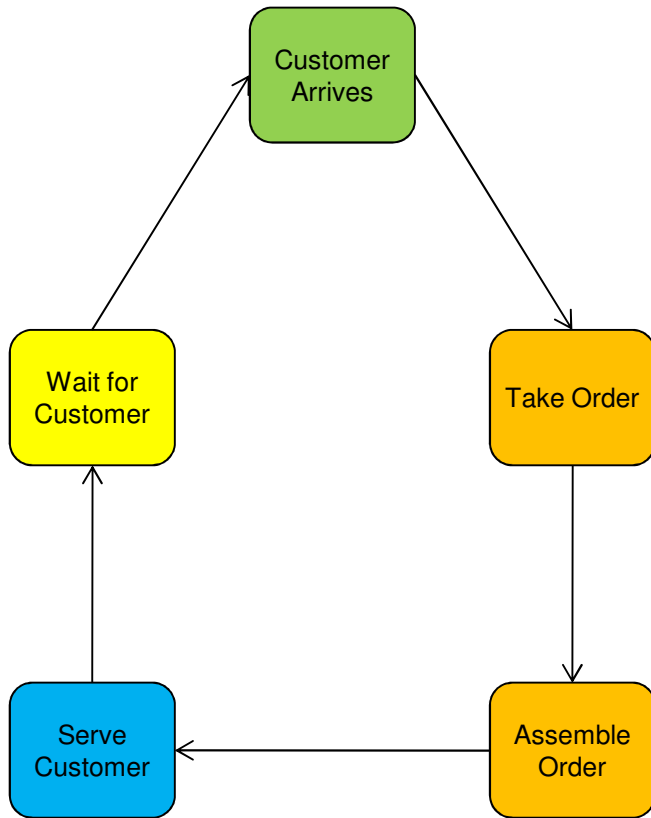
Finite-State Machine Model
(Event-based model)

Input Events
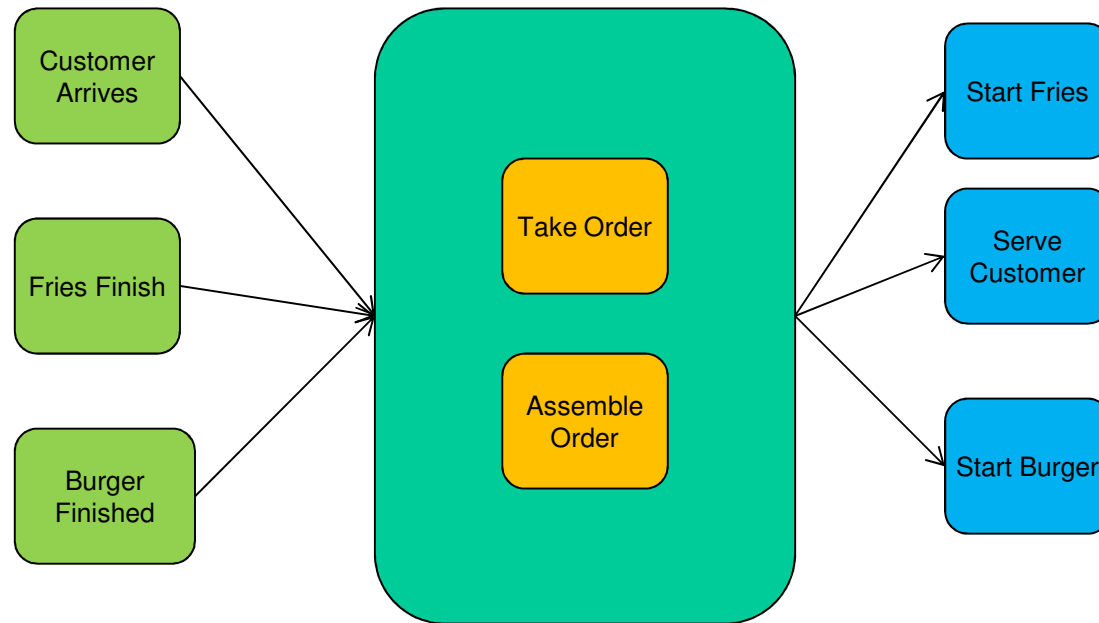
Non-Blocking actions

External activities

Customer Arrives

Fries Finish

Burger Finished

Take Order

Assemble Order

Start Fries

Serve Customer

Start Burger

Fries Cook

Burger Cooks

Wait for Customer

THE UNIVERSITY OF
NEW SOUTH WALES

28
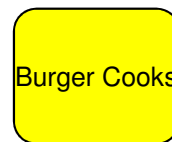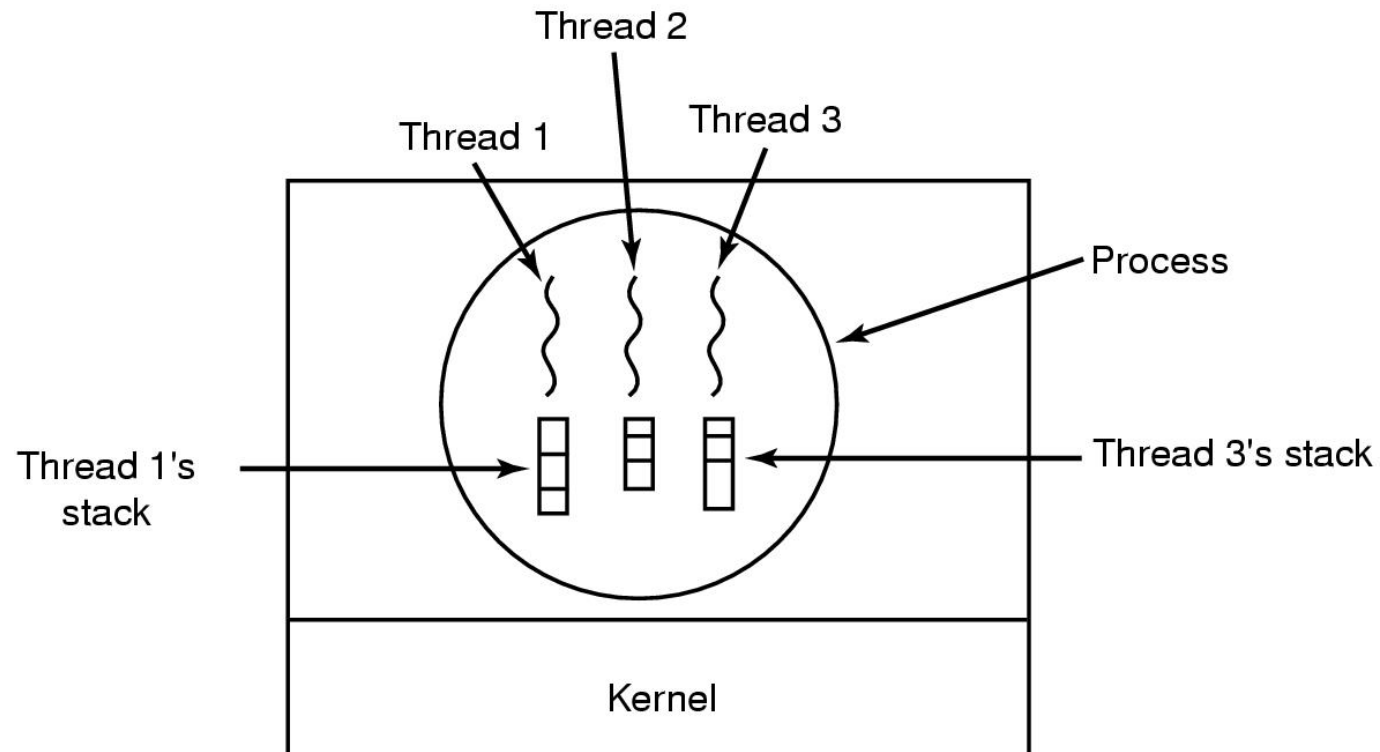
# The Thread Model

Each thread has its own stack

29

Thread Model

- Local variables are per thread
  - Allocated on the stack
- Global variables are shared between all threads
  - Allocated in data section
  - Concurrency control is an issue
- Dynamically allocated memory (malloc) can be global or local
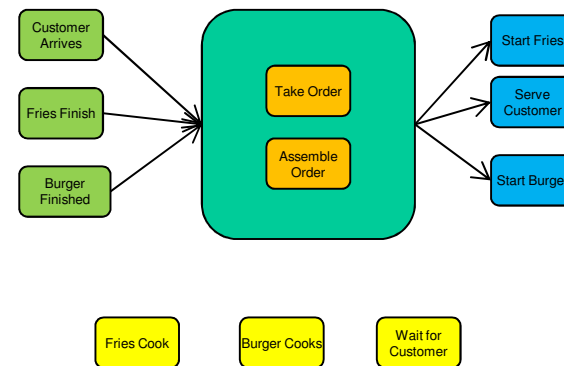 - Program defined (the pointer can be global or local)

Observation: Computation State

Thread Model

Finite State (Event) Model
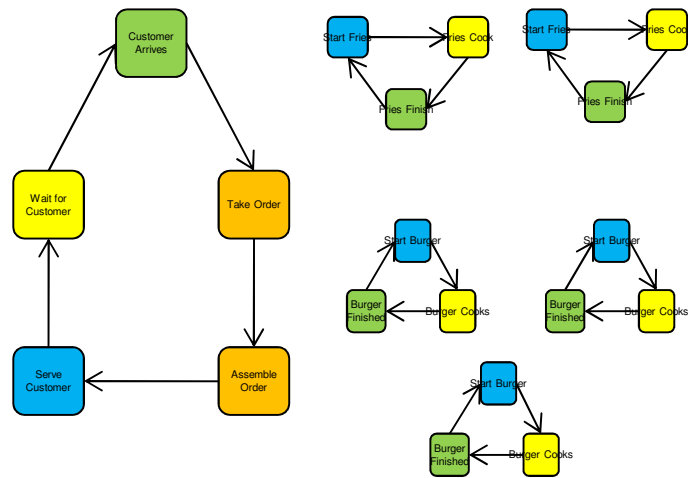
- State implicitly stored on the stack.

- State explicitly managed by program

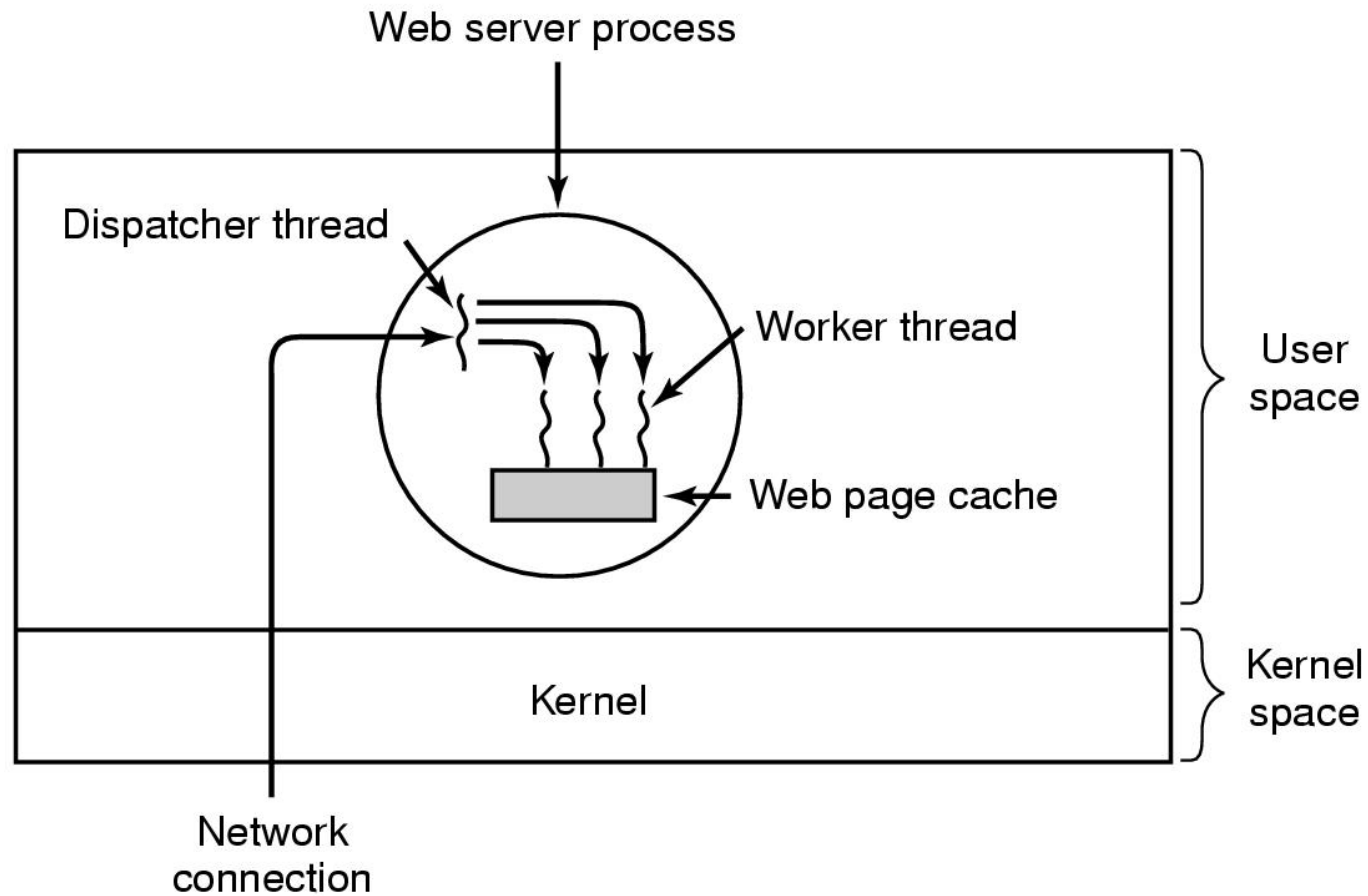THE UNIVERSITY OF
NEW SOUTH WALES
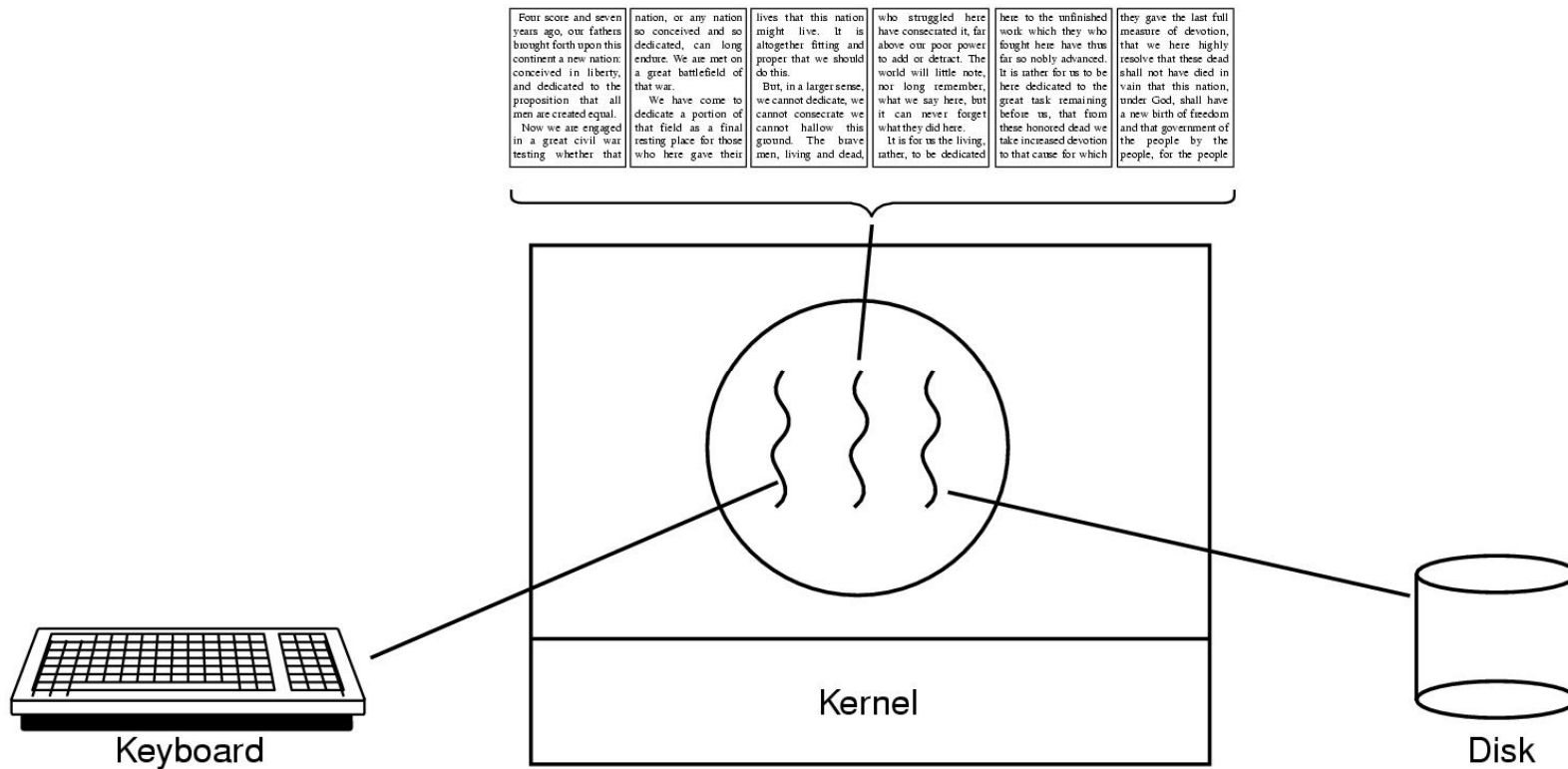
Thread Usage

A word processor with three threads

Thread Usage

A multithreaded Web server

Thread Usage

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}

          (a)
```

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
                (b)
```

- Rough outline of code for previous slide
  - (a) Dispatcher thread
  - (b) Worker thread – can overlap disk I/O with execution of other threads

Thread Usage

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

Three ways to construct a server

Summarising "Why Threads?"

- Simpler to program than a state machine
- Less resources are associated with them than a complete process
  - Cheaper to create and destroy
  - Shares resources (especially memory) between them
- Performance: Threads waiting for I/O can be overlapped with computing threads
  - Note if all threads are *compute bound*, then there is no performance improvement (on a uniprocessor)
- Threads can take advantage of the parallelism available on machines with more than one CPU (multiprocessor)

THE UNIVERSITY OF
NEW SOUTH WALES