Advanced Programming

Process-based Parallelism

IMPROVE APPLICATION PERFORMANCE

- Start reviewing all the key algorithms you have learned and their find their worst -case performance
- Look for different ways to improve the performance of an algorithm – For example Bubble sort->Merge Sort -> Quick Sort
- Choose the data structure wisely – cost of accessing an element close to $O(1)$

CACHING

```python
import time

# Function to calculate numbers recursively
def fibonacci_recursive(n):
    if n <= 1:
        return n
    else:
        return fibonacci_recursive(n-1) + \
               fibonacci_recursive(n-2)

# Function to calculate with caching
fib_cache = {}
def fibonacci_with_cache(n):
    if n in fib_cache:
        return fib_cache[n]
    if n <= 1:
        return n
    else:
        fib_cache[n] = fibonacci_with_cache(n-1) + \
                       fibonacci_with_cache(n-2)
        return fib_cache[n]
```

```python
if __name__ == '__main__':
    # Calculate without caching
    start_time = time.time()
    fibonacci_recursive(35)
    end_time = time.time()
    print(f"Without caching = \
          {end_time - start_time:.4e} s")

    # Calculate with caching
    start_time = time.time()
    fibonacci_with_cache(35)
    end_time = time.time()
    print(f"With caching =
          {end_time - start_time:.4e}”)
```

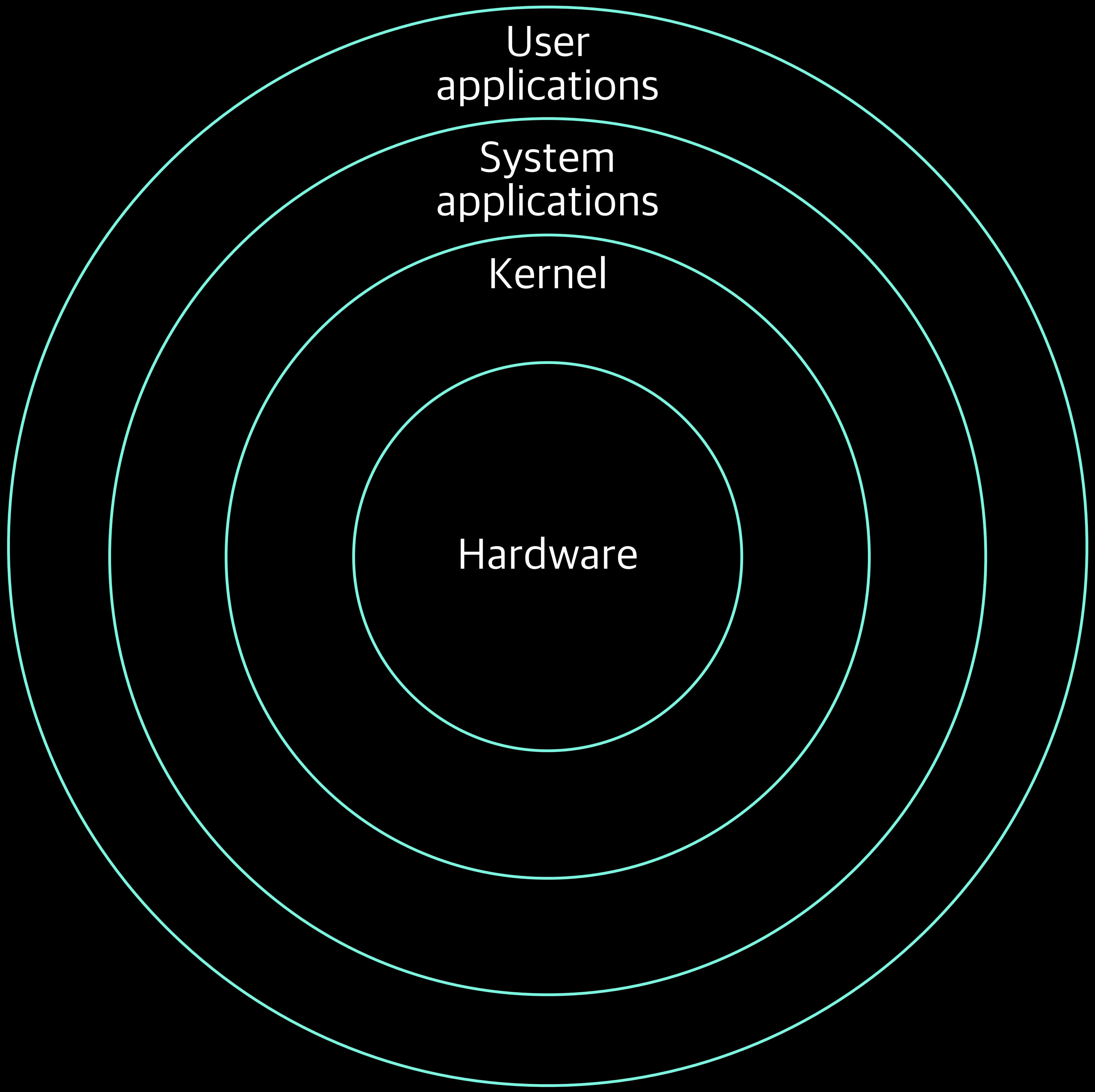
```
Without caching = 9.2852e-01 s

   With caching = 2.0981e-05 s
```

CACHING USING A DECORATOR

```python
from functools import lru_cache
import time

# Function to calculate Fibonacci numbers
@lru_cache(maxsize=None)  # No maximum cache size
def fibonacci(n):
  if n <= 1:
    return n
  else:
    return fibonacci(n-1) + fibonacci(n-2)

if __name__ == '__main__':
    # Without caching
    start_time = time.time()
    fibonacci(35)
    end_time = time.time()
    print(f"{'Without caching':<16}: {end_time - start_time:0.10e}")

    # With caching
    start_time = time.time()
    fibonacci(35)
    end_time = time.time()
    print(f"{'With caching':<16}: {end_time - start_time:0.10e}")
```

```
Without caching : 7.8678131104e-06
With caching    : 0.0000000000e+00
```

ARCHITECTURE OF A LINUX SYSTEM



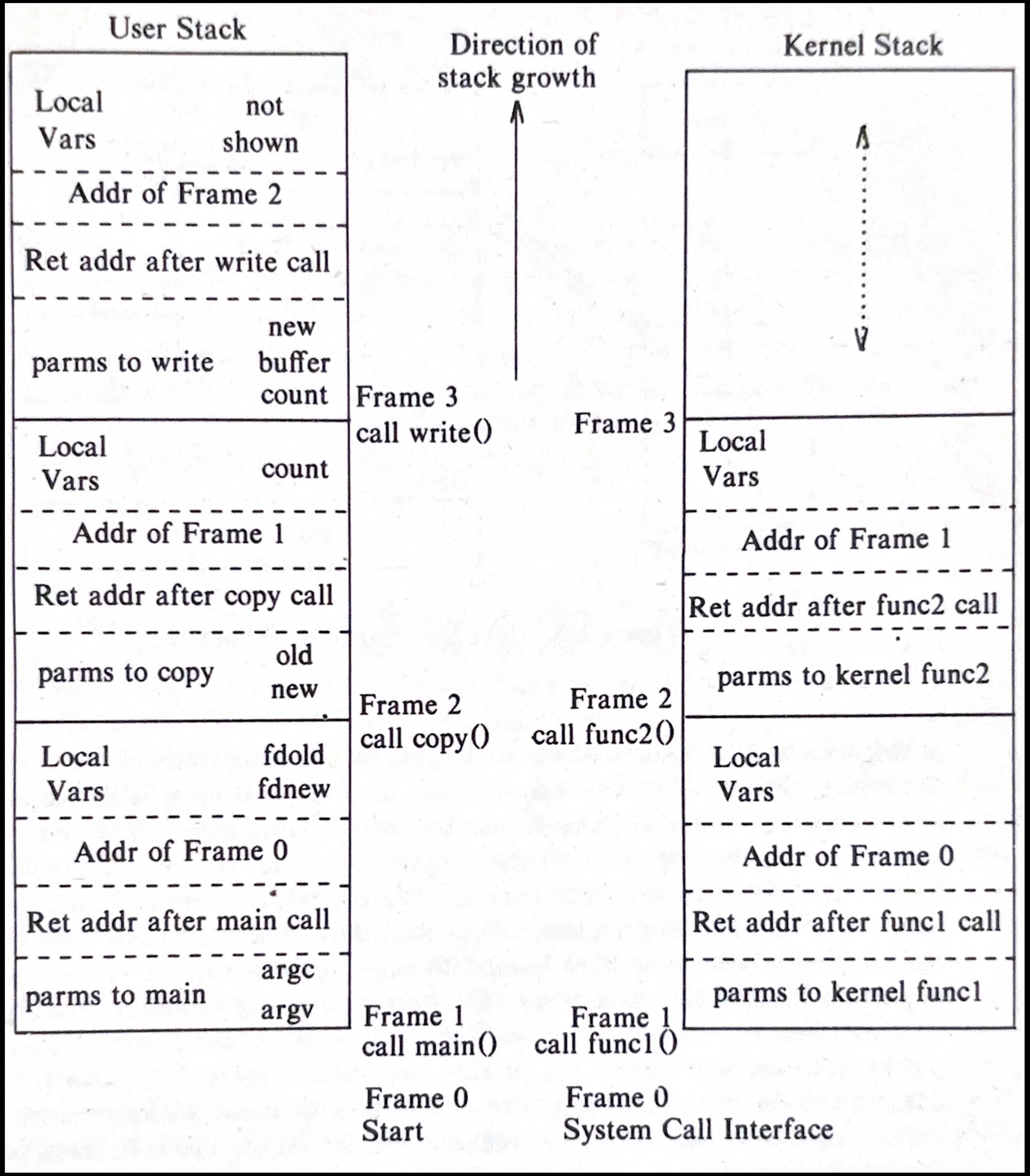
PROCESS

- A process is an active entity in an OS
- Fundamental unit of a program in an execution state
  - Consists of regions
    - patterns of bytes interpreted as instructions by CPU, called as text
    - Data
    - Stack
- Self-contained
- Reads and writes its data and stack
- Cannot read and write another process directly
- Communicates with other processes through messages using system calls

PROCESS

```python
import sys

def copy(source_file, destination_file):
    try:
        with open(source_file, 'r') as src, \
        open(destination_file, 'w') as dst: \
             dst.write(src.read())
        print(f"File copied successfully from
          {source_file} to {destination_file}")
    except FileNotFoundError:
        print(f"Error: Source file '{source_file}'
          not found.")
    except IOError as e:
        print(f"Error accessing files: {e}")

if __name__ == '__main__':
    if len(sys.argv) != 3:
        print(f"Usage: python {sys.argv[0]}
          <source_file> <destination_file>")
        sys.exit(1)

    else:
        copy(sys.argv[1], sys.argv[2])
```

USER AND KERNEL STACK FOR COPY



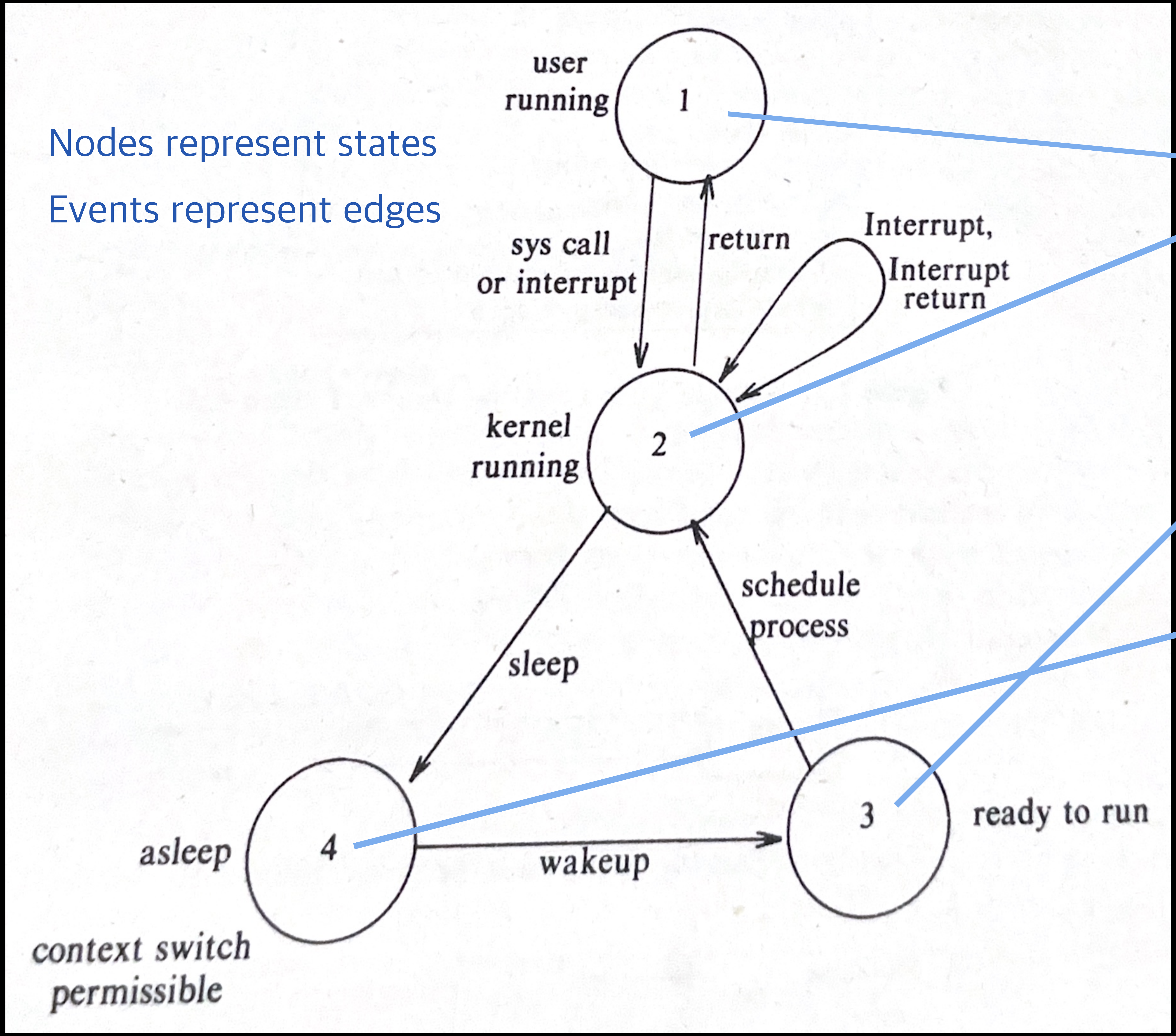

```python
import sys

def copy(source_file, destination_file):
    try:
        with open(source_file, 'r') as src, \
        open(destination_file, 'w') as dst: \
            dst.write(src.read())
        print(f"File copied successfully from
              {source_file} to {destination_file}")
    except FileNotFoundError:
        print(f"Error: Source file '{source_file}'
              not found.")
    except IOError as e:
        print(f"Error accessing files: {e}")

if __name__ == '__main__':
    if len(sys.argv) != 3:
        print(f"Usage: python {sys.argv[0]}
              <source_file> <destination_file>")
        sys.exit(1)

    else:
        copy(sys.argv[1], sys.argv[2])
```

Source: The design of UNIX operating system, M.J. Bach

PROCESS STATES

- Currently executing in user mode
- Currently executing in kernel mode
- Waiting – Not in execution mode – waiting for the scheduler to allocate CPU
- Sleeping – Waiting for an I/O to complete

PROCESS STATES AS DIGRAPH



A process may be in state 1 or 2

Many process may be in this state

Waiting for I/O to complete

CONTEXT SWITCH

- In any multitasking operating system, multiple processes can run simultaneously
- A single CPU can only execute one process at a time
- The context of a process its state – its text, values, data structure and the machine register values, content of user and kernel stack frames
- Context switching saves the context of a currently running process at time t_1 in order to continue at a future time t_n
- Allows the OS to switch between processes efficiently

- Allows all processes to share a single CP
- Facilitates efficient CPU utilisation
- Increased CPU Usage
- frequency of context switching significantly affects performance

CONCURRENCY AND PARALLELISM

- Creates an impression that multiple are running at the same time
  - Manages multiple tasks
  - Tasks seem to run at or
  - Achieved through conte switching on a single CPU
  - Concurrency exhibits non-deterministic control flow
  - Browser - download and browsing

Due to

Asynchronous Execution
Context-switching
Resource sharing
Synchronisation

- Runs multiple computations simultaneously
- Tasks truly run at once
- Multiple CPU cores/distributed systems

To learn more about Concurrent programming, listen to this lecture

https://www.youtube.com/watch?v=XbdDSUI8NXE

TO BE CONTINUED