THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

OPERATING SYSTEMS

Workshop 0

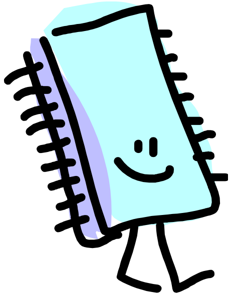
Much of the material on these slides comes from the recommended textbook by William Stallings

Week 01 Workshop 0 Outline

Operating System Overview

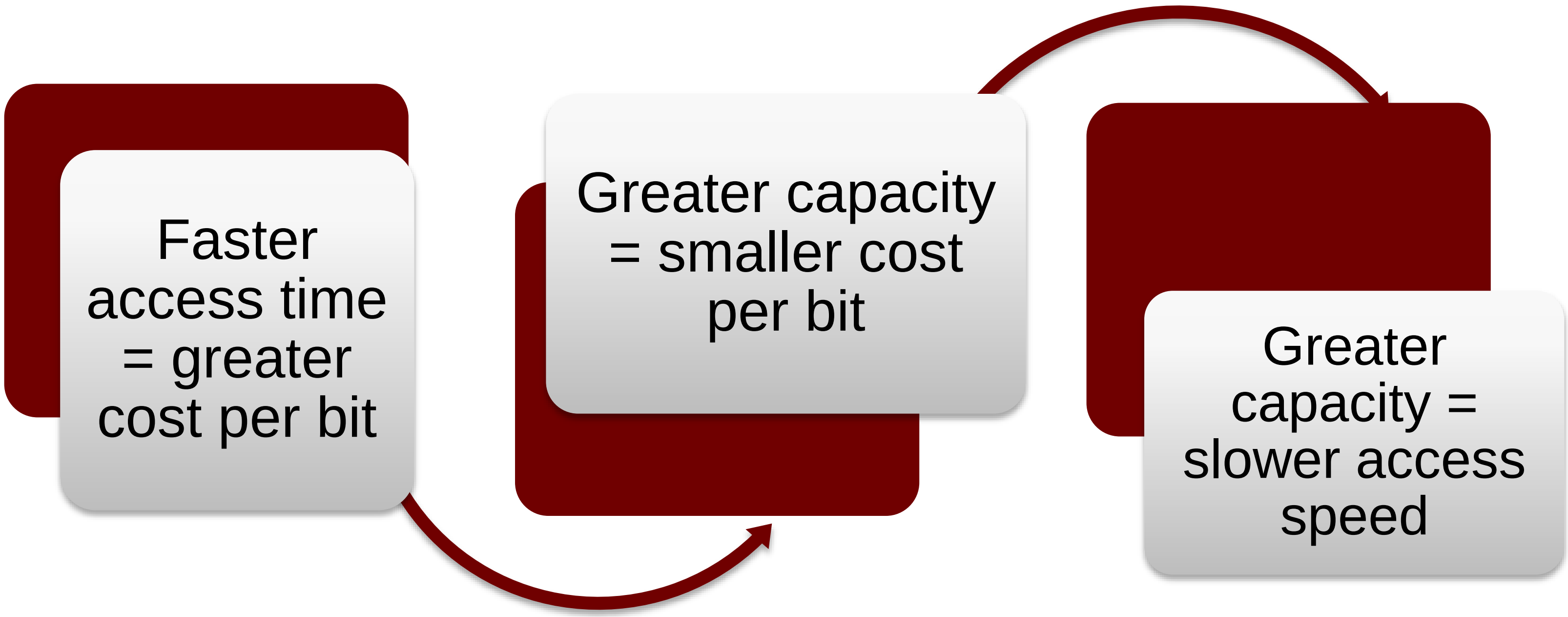
- ❑ Memory Hierarchy
- ❑ Cache Memory
- ❑ I/O Techniques
- ❑ Multiprocessors and Multicores
- ❑ Evolution of OS
- ❑ Serial Processing
- ❑ Batch Processing
- ❑ User Mode VS Kernel Mode
- ❑ Multi-Programming VS Uni-programming
- ❑ Time Sharing
- ❑ Fault Tolerance

Memory Hierarchy



- Major constraints in memory
    - amount
    - speed
    - expense
- Memory must be able to keep up with the processor
- Cost of memory must be reasonable in relationship to the other components

Memory Relationships



Faster access time = greater cost per bit

Greater capacity = smaller cost per bit

Greater capacity = slower access speed

The Memory Hierarchy

- Going down the hierarchy:
  - decreasing cost per bit
  - increasing capacity
  - increasing access time
  - decreasing frequency of access to the memory by the processor

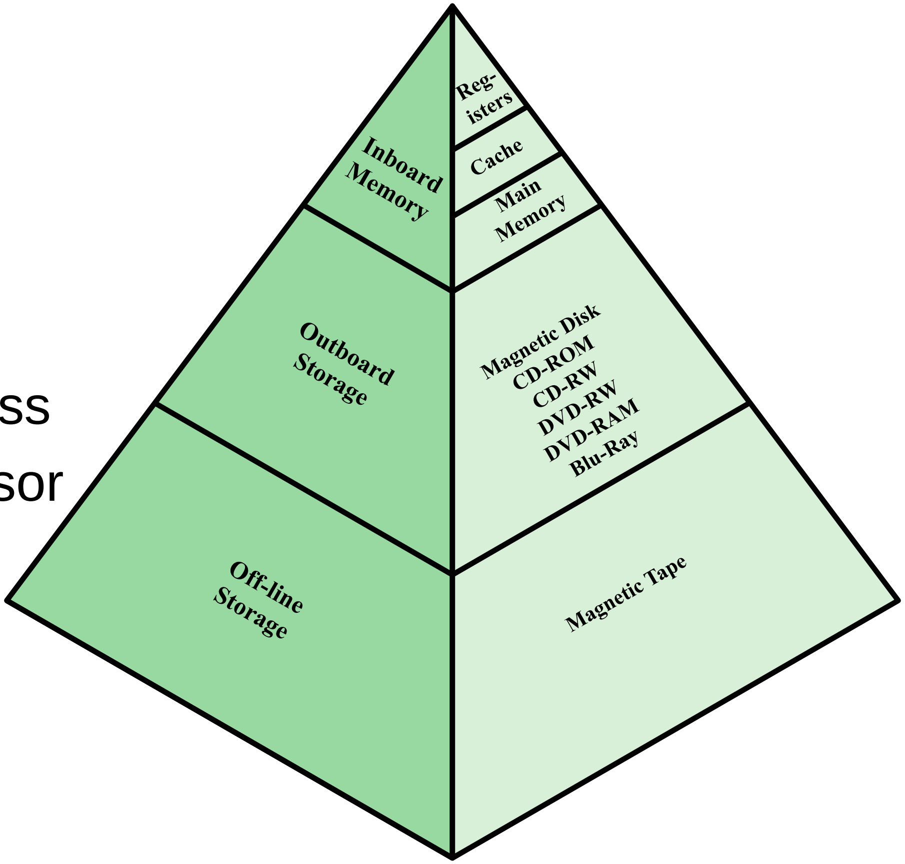
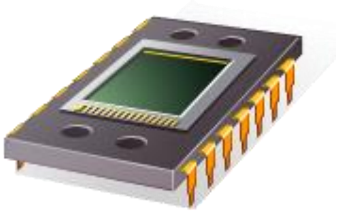


Figure 1.14 The Memory Hierarchy

Secondary Memory

- Also referred to as auxiliary memory
  - external
  - nonvolatile
  - used to store program and data files

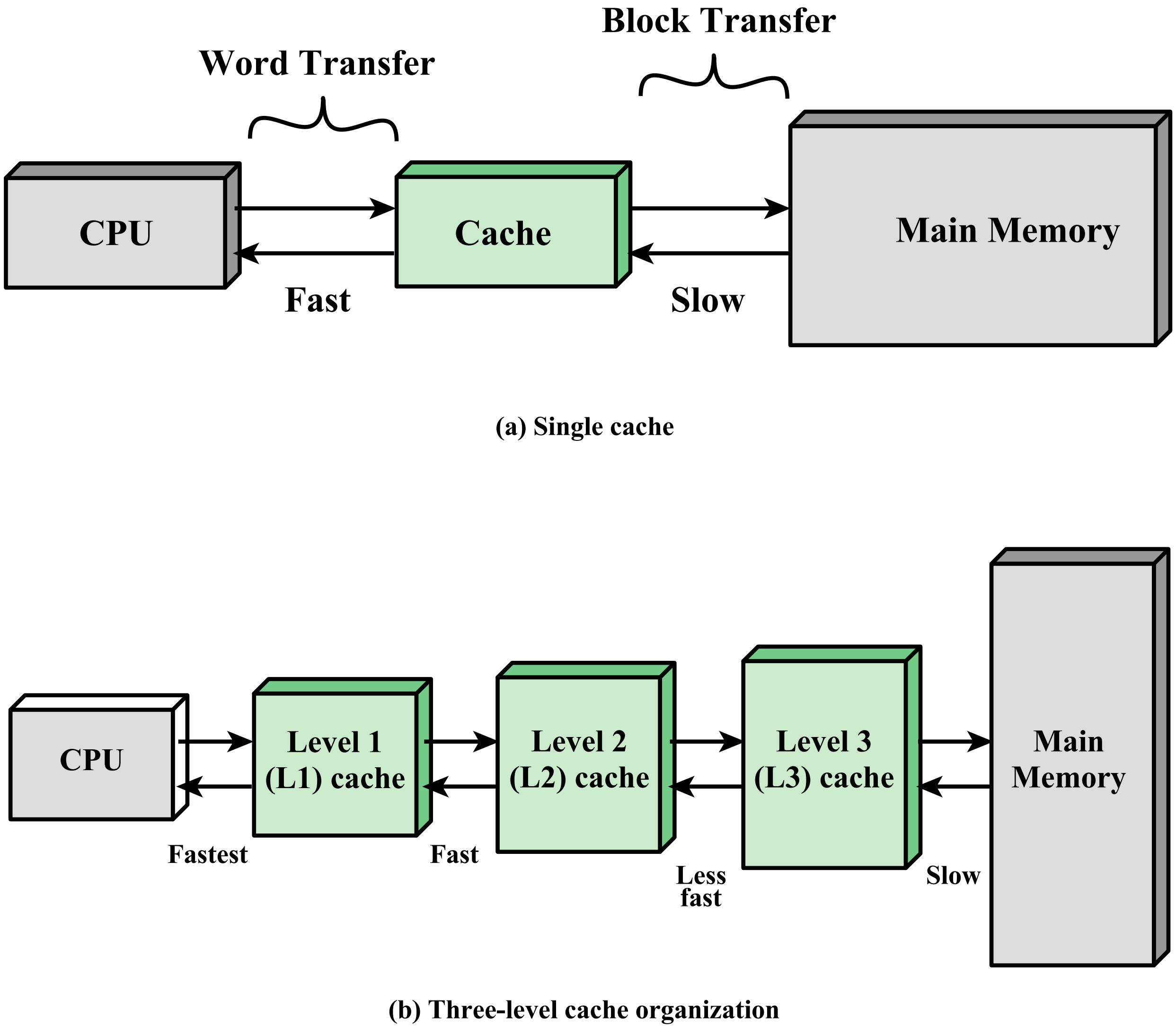
Cache Memory



- Invisible to the OS
- Interacts with other memory management hardware
- Processor must access memory at least once per instruction cycle
- Processor execution is limited by memory cycle time
- Exploit the principle of locality with a small, fast memory

Principle of Locality

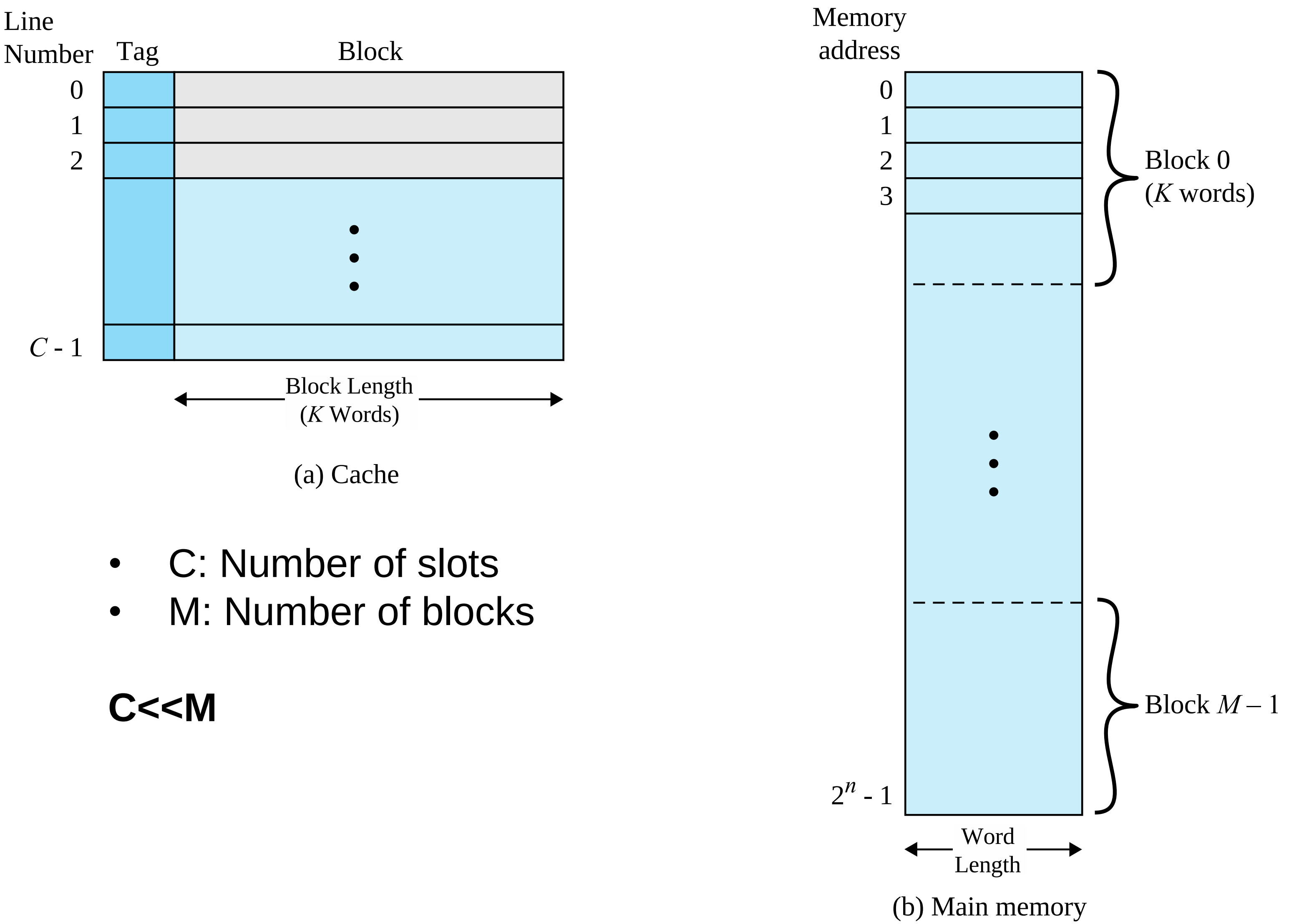
- Memory references by the processor tend to cluster
- Data is organized so that the percentage of accesses to each successively lower level is substantially less than that of the level above
- Can be applied across more than two levels of memory



(a) Single cache

(b) Three-level cache organization

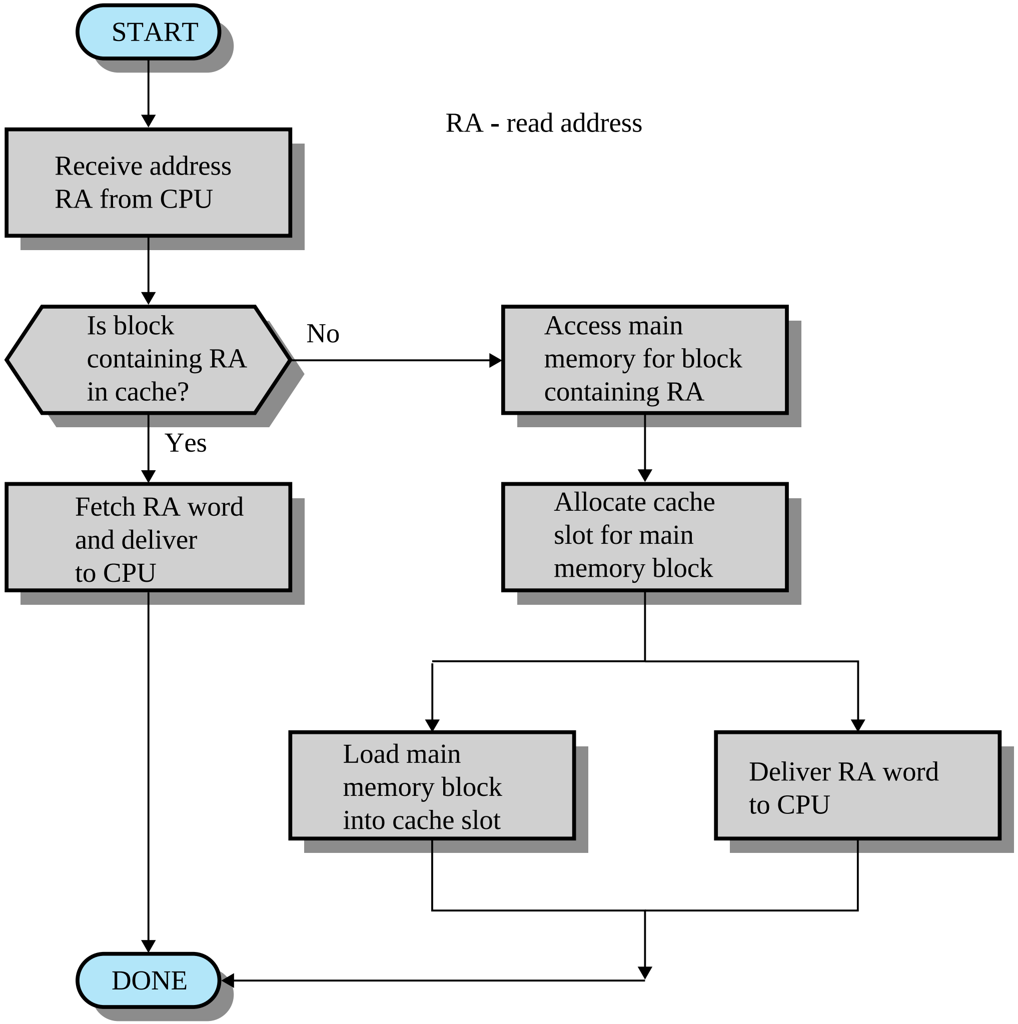
Figure 1.16 Cache and Main Memory



- C: Number of slots
- M: Number of blocks

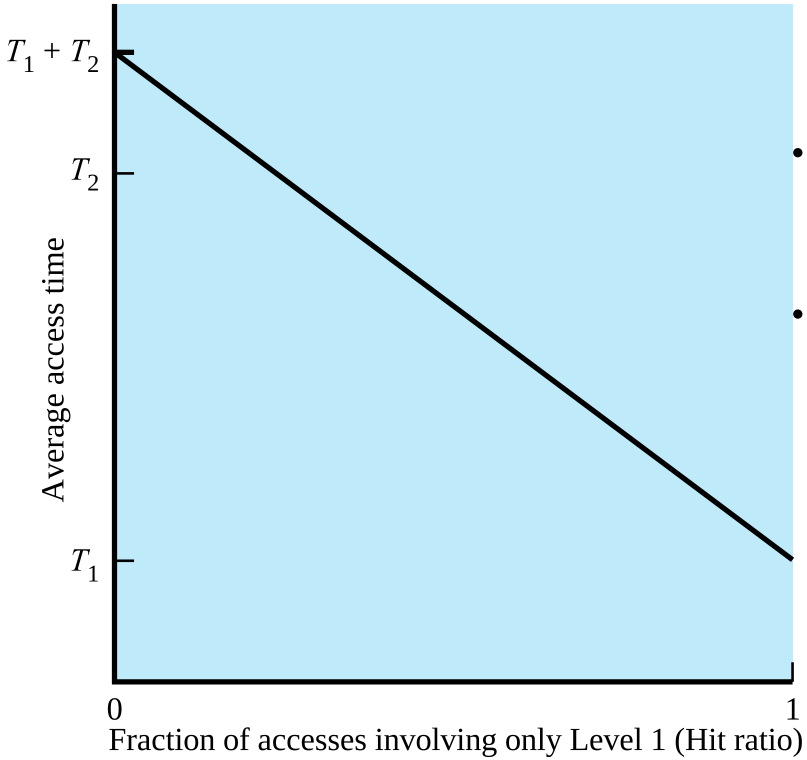
C<<M

Figure 1.17 Cache/Main-Memory Structure


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache slot for main memory block
Load main memory block into cache slot
Deliver RA word to CPU
DONE

RA - read address

Figure 1.18 Cache Read Operation

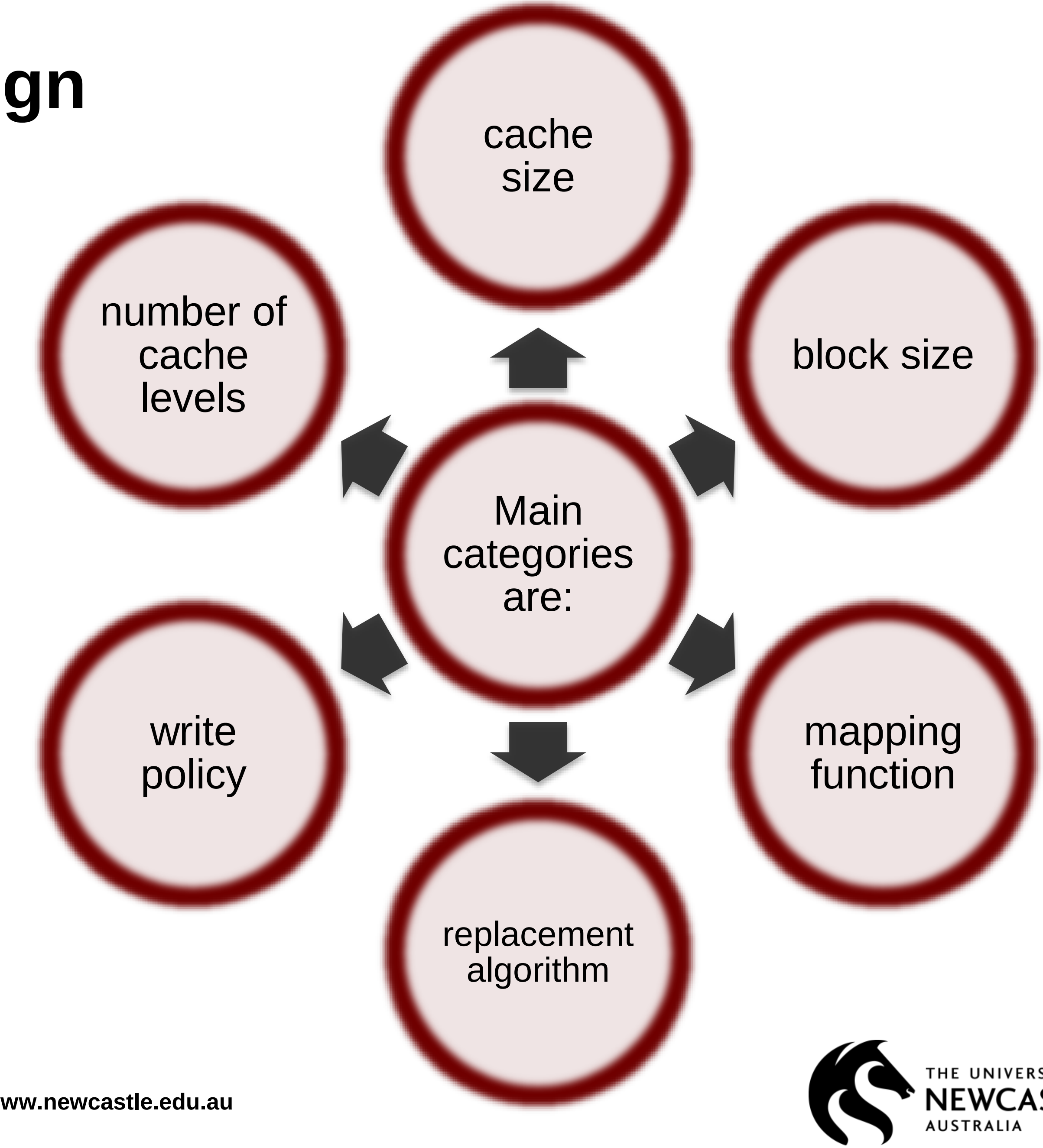


- Level 1
  - 1,000 bytes
  - Access time of T_1=0.1 μs
- Level 2
  - 100,000 bytes
  - Access time of T_2=1 μs.

If H = 0.95 , the average time to access a byte is

(0.95) (0.1 μs) + (0.05) (0.1 μs + 1 μs) = 0.095 + 0.055 = 0.15 μs

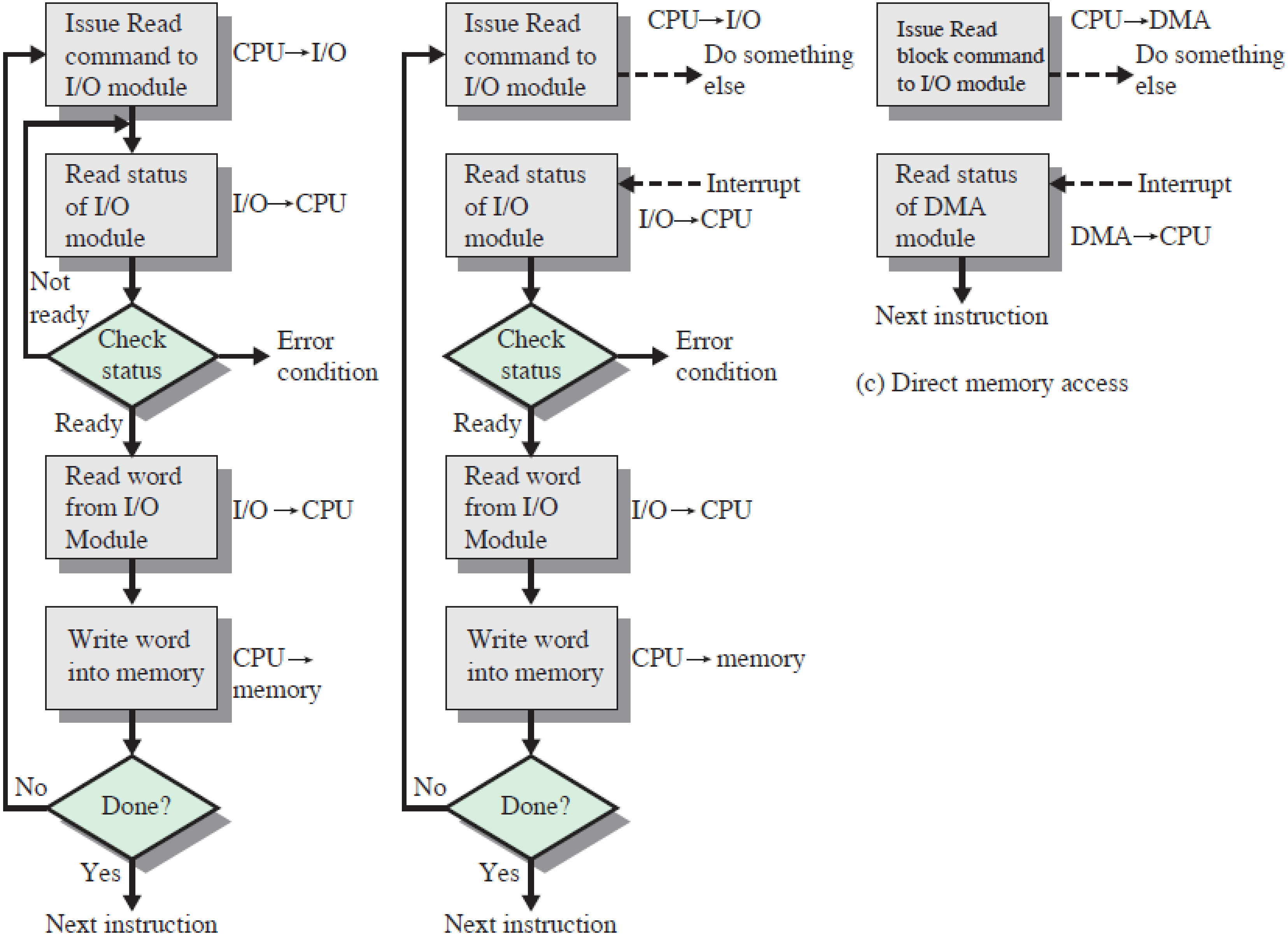
Cache Design



I/O Techniques

- When the processor encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module

- Three techniques are possible for I/O operations:
  - Programmed I/O
  - Interrupt-Driven I/O
  - Direct Memory Access (DMA)



(a) Programmed I/O

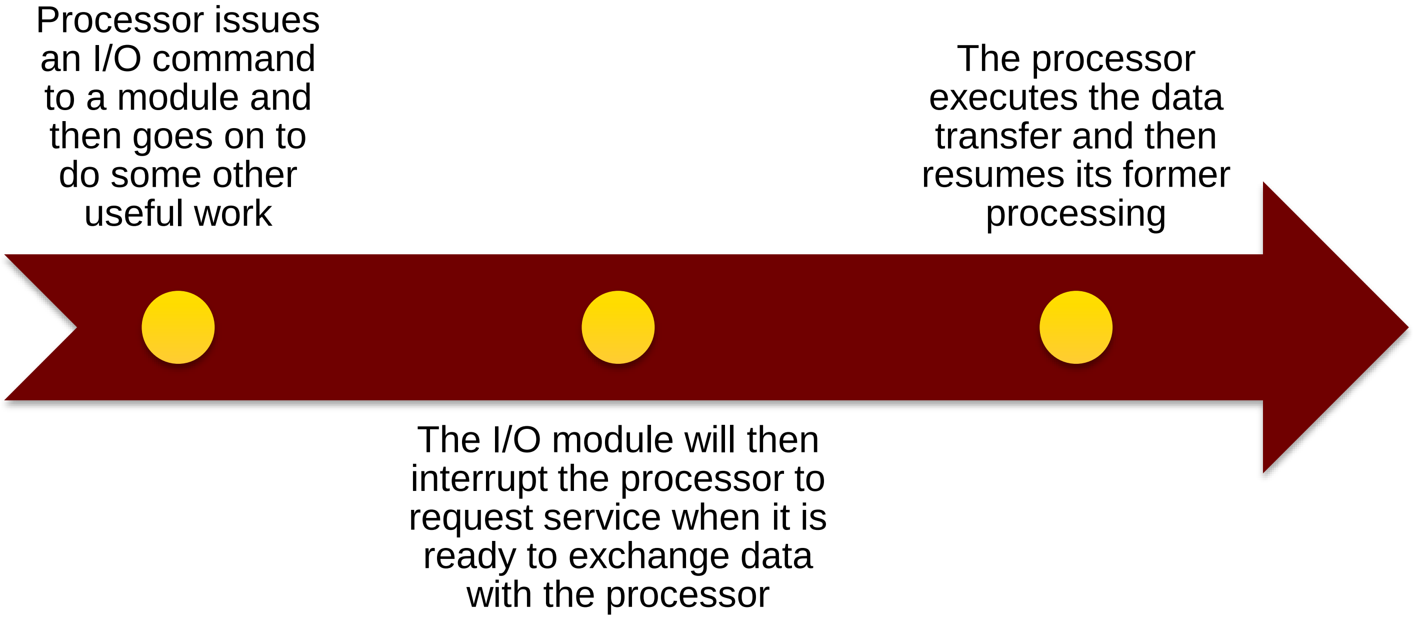
(b) Interrupt-driven I/O

(c) Direct memory access

Programmed I/O

- The I/O module performs the requested action then sets the appropriate bits in the I/O status register
- The processor periodically checks the status of the I/O module until it determines the instruction is complete
- With programmed I/O the performance level of the entire system is severely degraded

Interrupt-Driven I/O



Processor issues an I/O command to a module and then goes on to do some other useful work

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor

The processor executes the data transfer and then resumes its former processing

More efficient than Programmed I/O but still requires active intervention of the processor to transfer data between memory and an I/O module

Drawbacks

- Transfer rate is limited by the speed with which the processor can test and service a device
- The processor is tied up in managing an I/O transfer
  - a number of instructions must be executed for each I/O transfer

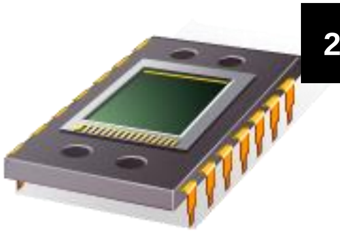
Direct Memory Access (DMA)

- Performed by a separate module on the system bus or incorporated into an I/O module
- When the processor wishes to read or write data it issues a command to the DMA module containing:
  - whether a read or write is requested
  - the address of the I/O device involved
  - the starting location in memory to read/write
  - the number of words to be read/written

Direct Memory Access

- Transfers the entire block of data directly to and from memory without going through the processor
    - processor is involved only at the beginning and end of the transfer
    - processor executes more slowly during a transfer when processor access to the bus is required
- More efficient than interrupt-driven or programmed I/O

Symmetric Multiprocessors (SMP)



- A stand-alone computer system with the following characteristics:
  - two or more similar processors of comparable capability
  - processors share the same main memory and are interconnected by a bus or other internal connection scheme
  - processors share access to I/O devices
  - all processors can perform the same functions
  - the system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels

SMP Advantages

- Performance
  - a system with multiple processors will yield greater performance if work can be done in parallel
- Availability
  - the failure of a single processor does not halt the machine
- Incremental Growth
  - an additional processor can be added to enhance performance
- Scaling
  - vendors can offer a range of products with different price and performance characteristics

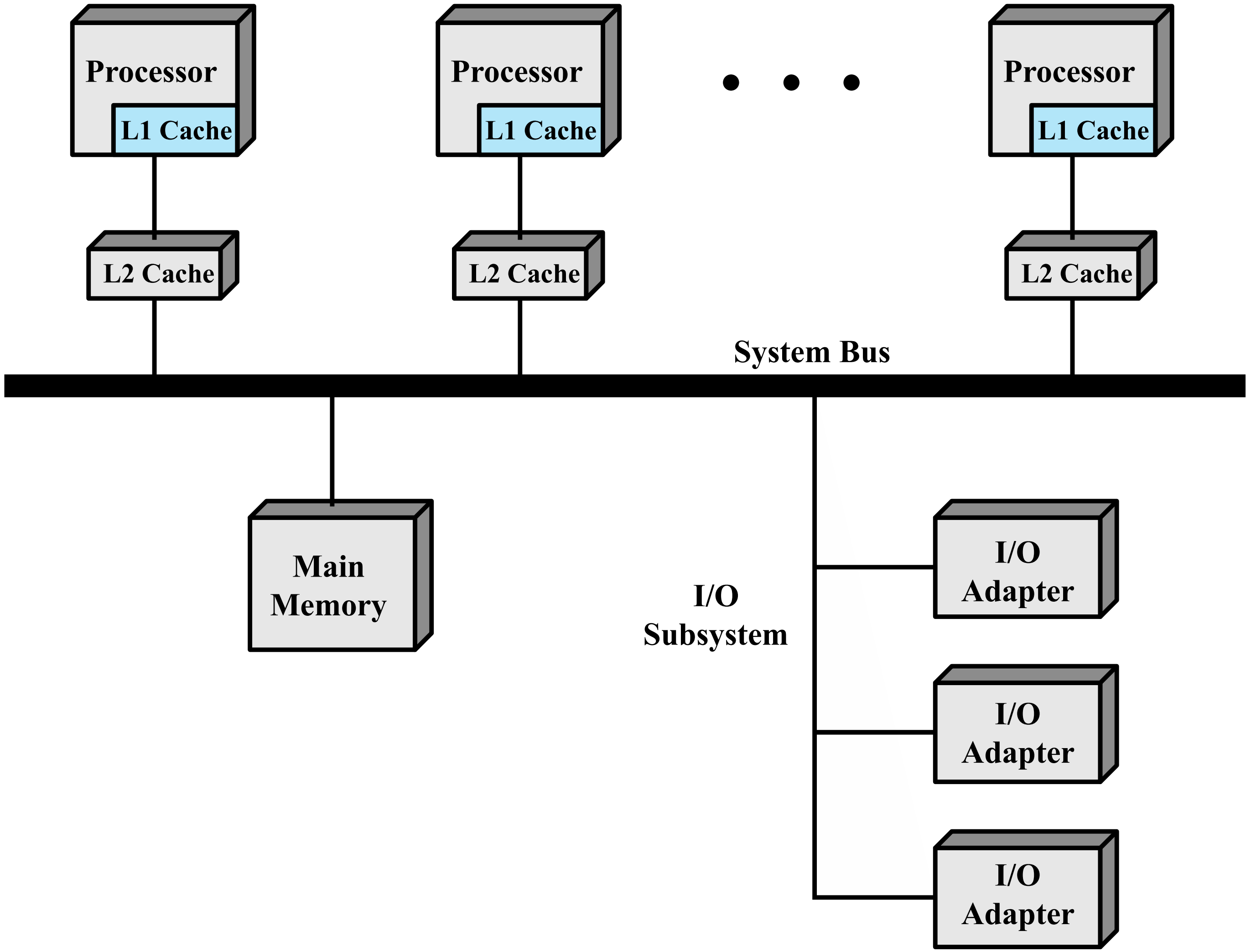
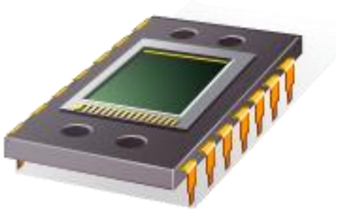


Figure 1.19 Symmetric Multiprocessor Organization

Multicore Computer



- Also known as a chip multiprocessor
- Combines two or more processors (cores) on a single piece of silicon (die)
  - each core consists of all of the components of an independent processor
- In addition, multicore chips also include L2 cache and in some cases L3 cache

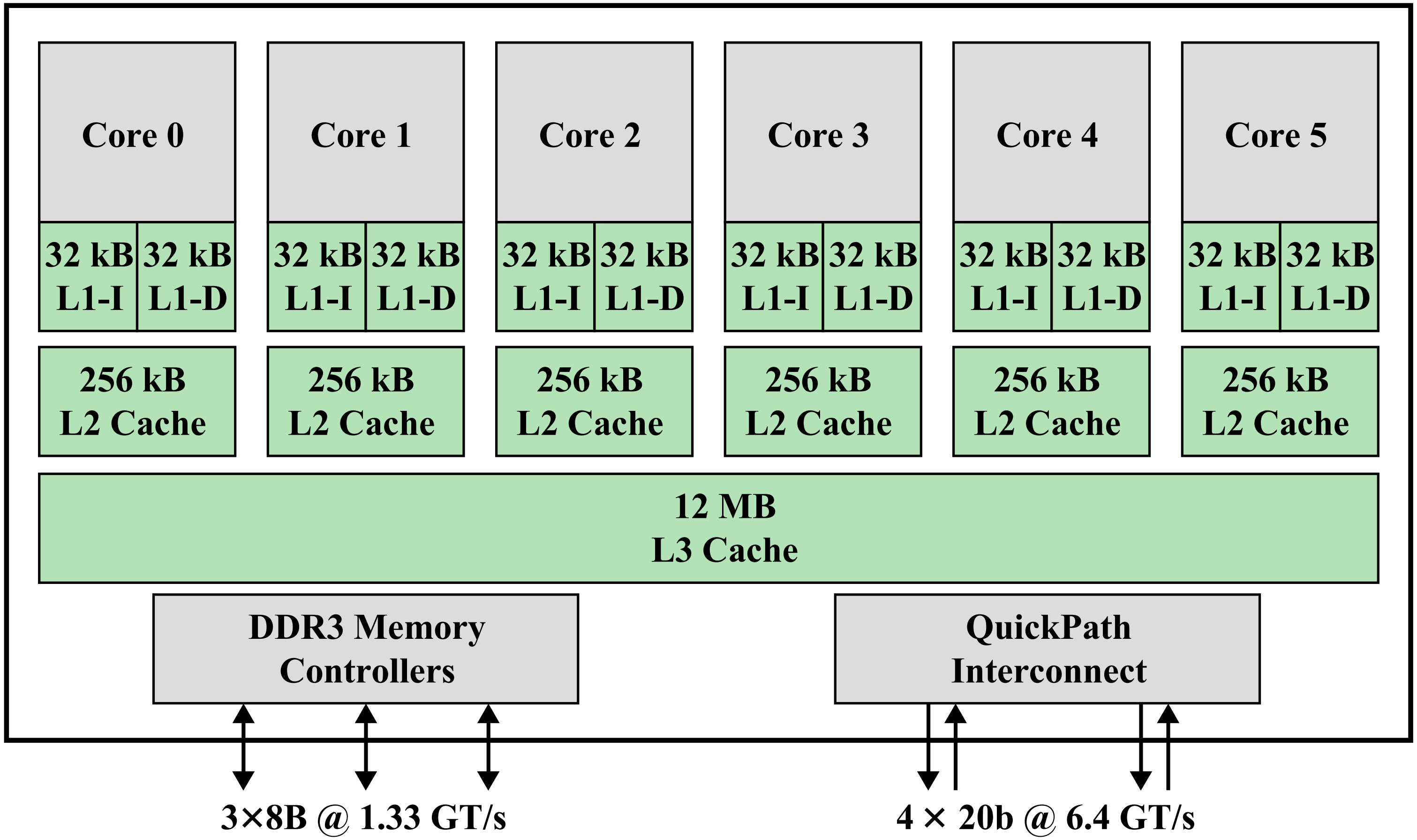
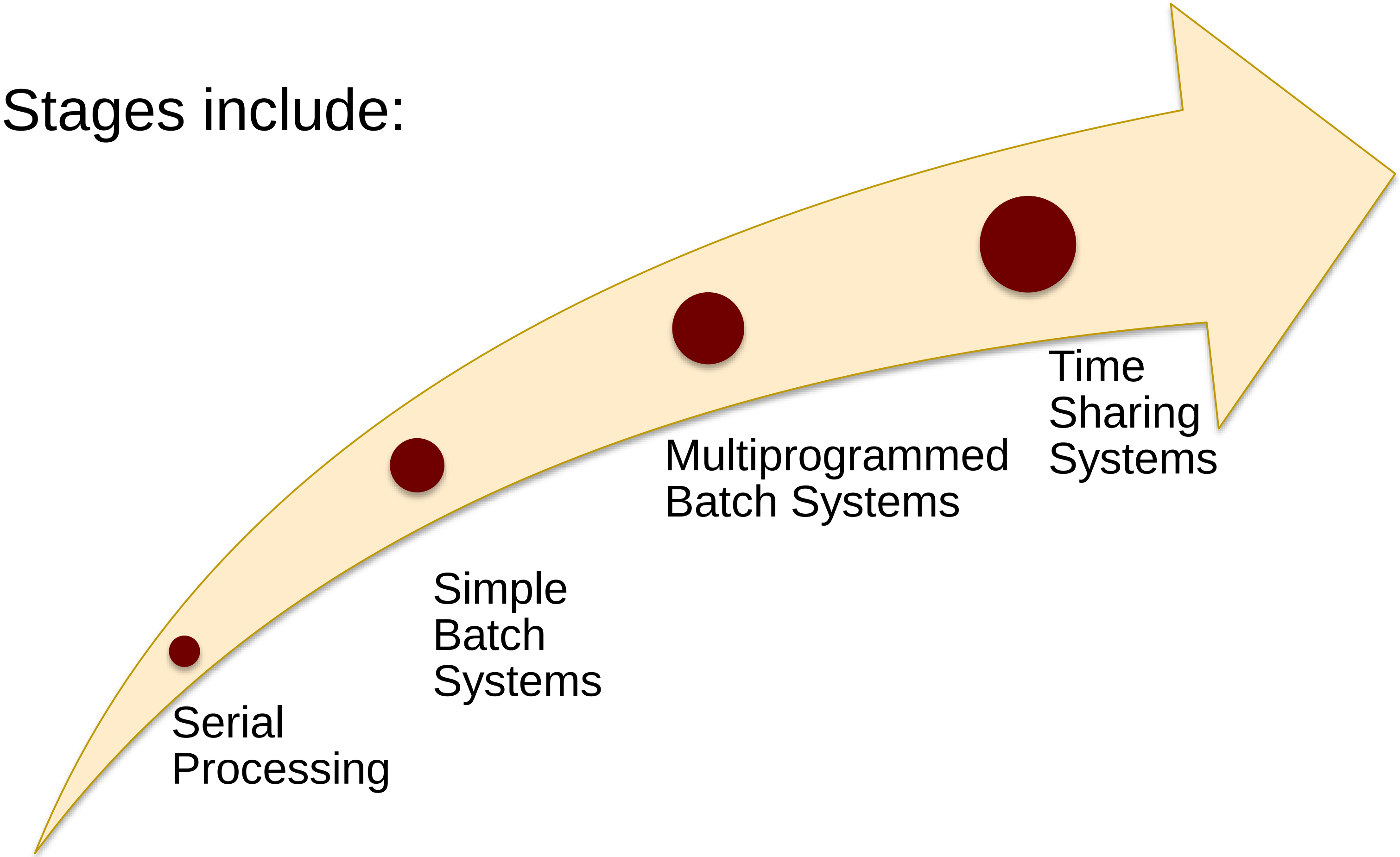


Figure 1.20 Intel Core i7-990X Block Diagram

Evolution of Operating Systems

- Stages include:



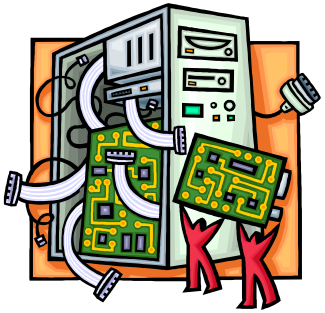
Serial Processing

Simple Batch Systems

Multiprogrammed Batch Systems

Time Sharing Systems

Serial Processing



Earliest Computers

- No operating system
- Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in “series”

Problems

- Scheduling:
  - most installations used a hardcopy sign-up sheet to reserve computer time
  - time allocations could run short or long, resulting in wasted computer time
- Setup time
  - a considerable amount of time was spent just on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - important to maximize processor utilization
- Monitor
  - user no longer has direct access to processor
  - job is submitted to computer operator who batches them together and places them on an input device
  - program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- ***Resident Monitor*** is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

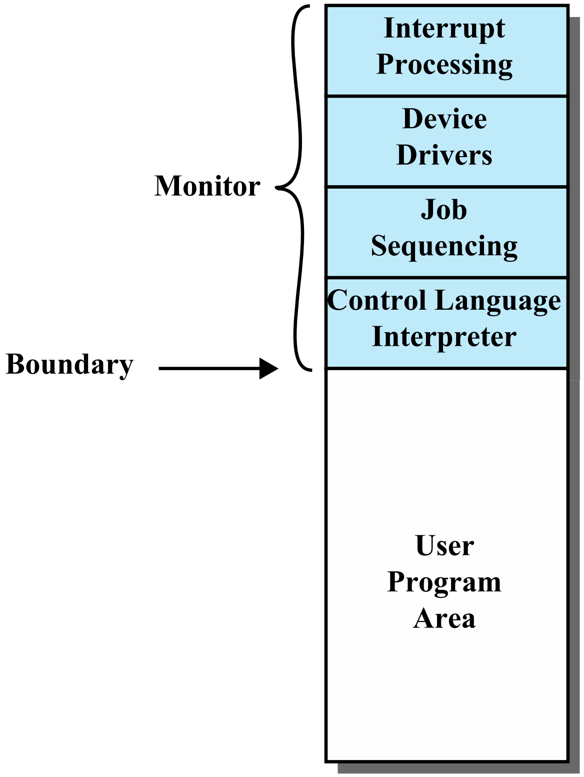
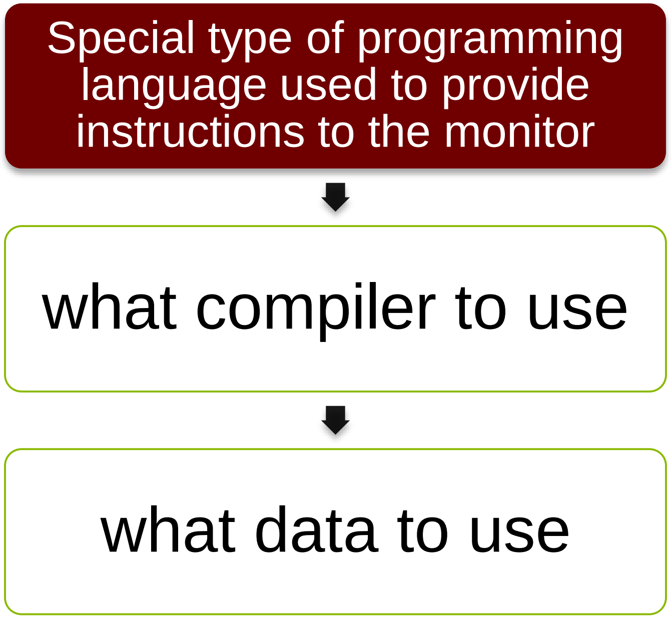


Figure 2.3 Memory Layout for a Resident Monitor

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "*control is passed to a job*" means processor is fetching and executing instructions in a user program
- "*control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)



Special type of programming language used to provide instructions to the monitor

what compiler to use

what data to use

$JOB

$FTN

- •
- •
- •

FORTRAN instruction

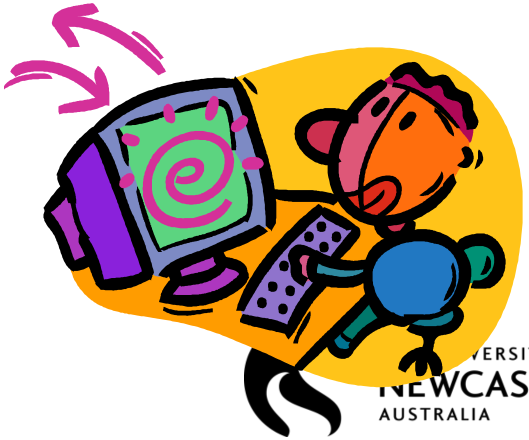
$LOAD

$RUN

- •
- •
- •

Data

$END



Desirable Hardware Features

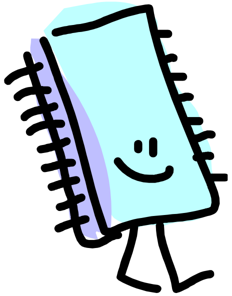
- Memory protection for monitor
  - while the user program is executing, it must not alter the memory area containing the monitor
- Timer
  - prevents a job from monopolizing the system
- Privileged instructions
  - can only be executed by the monitor
- Interrupts
  - gives OS more flexibility in controlling user programs

Modes of Operation

- **User Mode**
  - user program executes in user mode
  - certain areas of memory are protected from user access
  - certain instructions may not be executed
- **Kernel Mode**
  - monitor executes in kernel mode
  - privileged instructions may be executed
  - protected areas of memory may be accessed

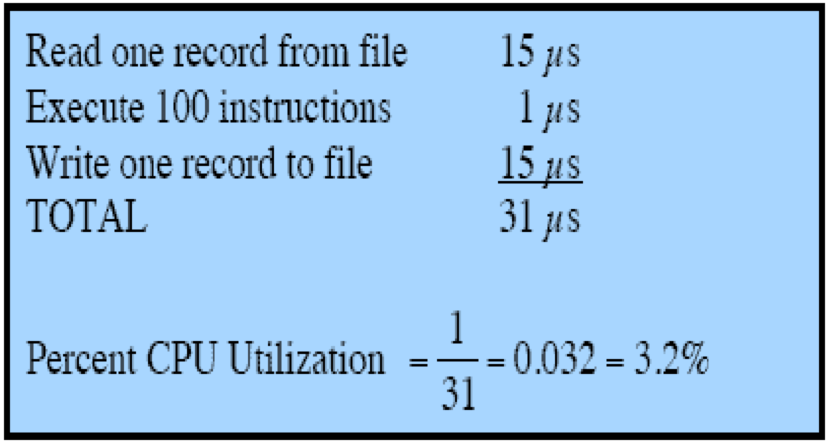
Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
    - some main memory is now given over to the monitor
    - some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer



Multiprogrammed Batch Systems

- Processor is often idle
    - even with automatic job sequencing
    - I/O devices are slow compared to processor



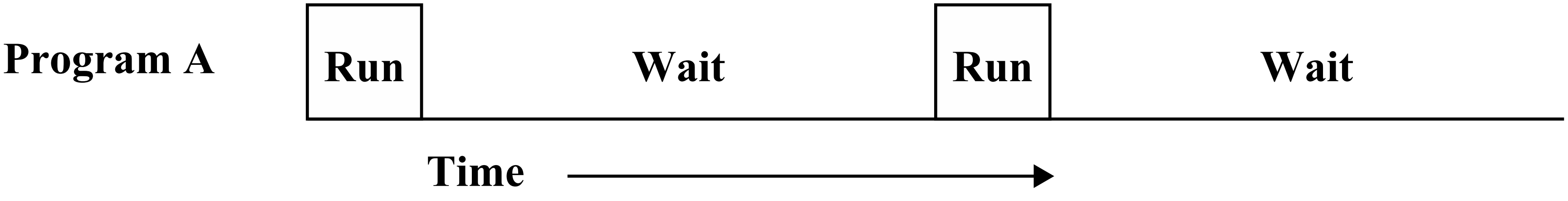
Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Figure 2.4 System Utilization Example

Uniprogramming

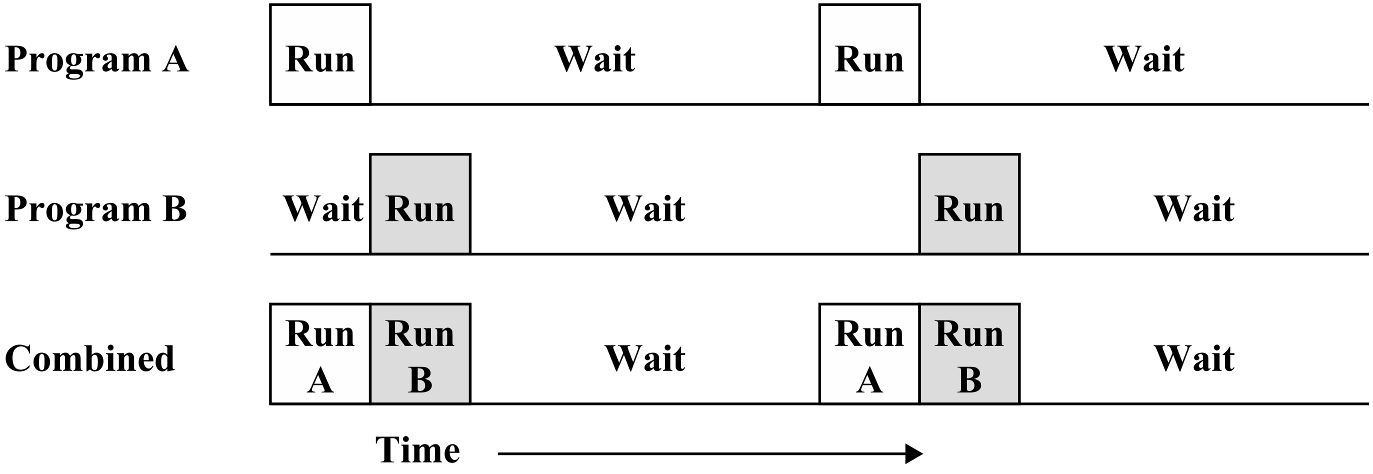
- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding



(a) Uniprogramming

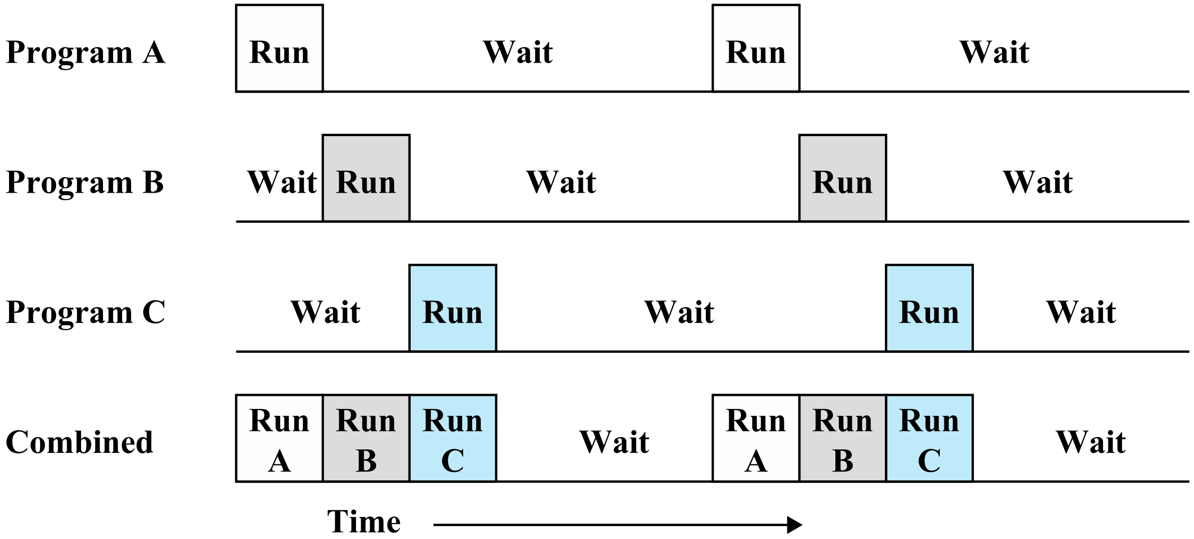
Multiprogramming

- There must be enough memory to hold the OS (resident monitor) and more than one user program
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O



(b) Multiprogramming with two programs

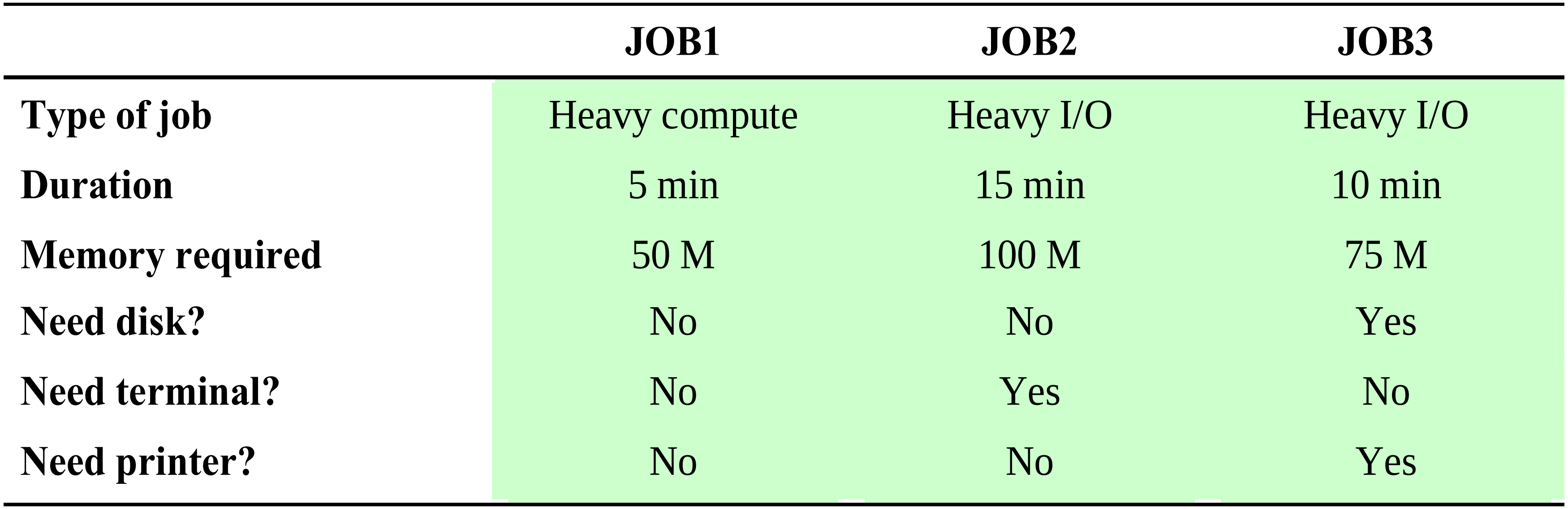
Multiprogramming



(c) Multiprogramming with three programs

- Multiprogramming
  - also known as multitasking
  - memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example



	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 2.1 Sample Program Execution Attributes

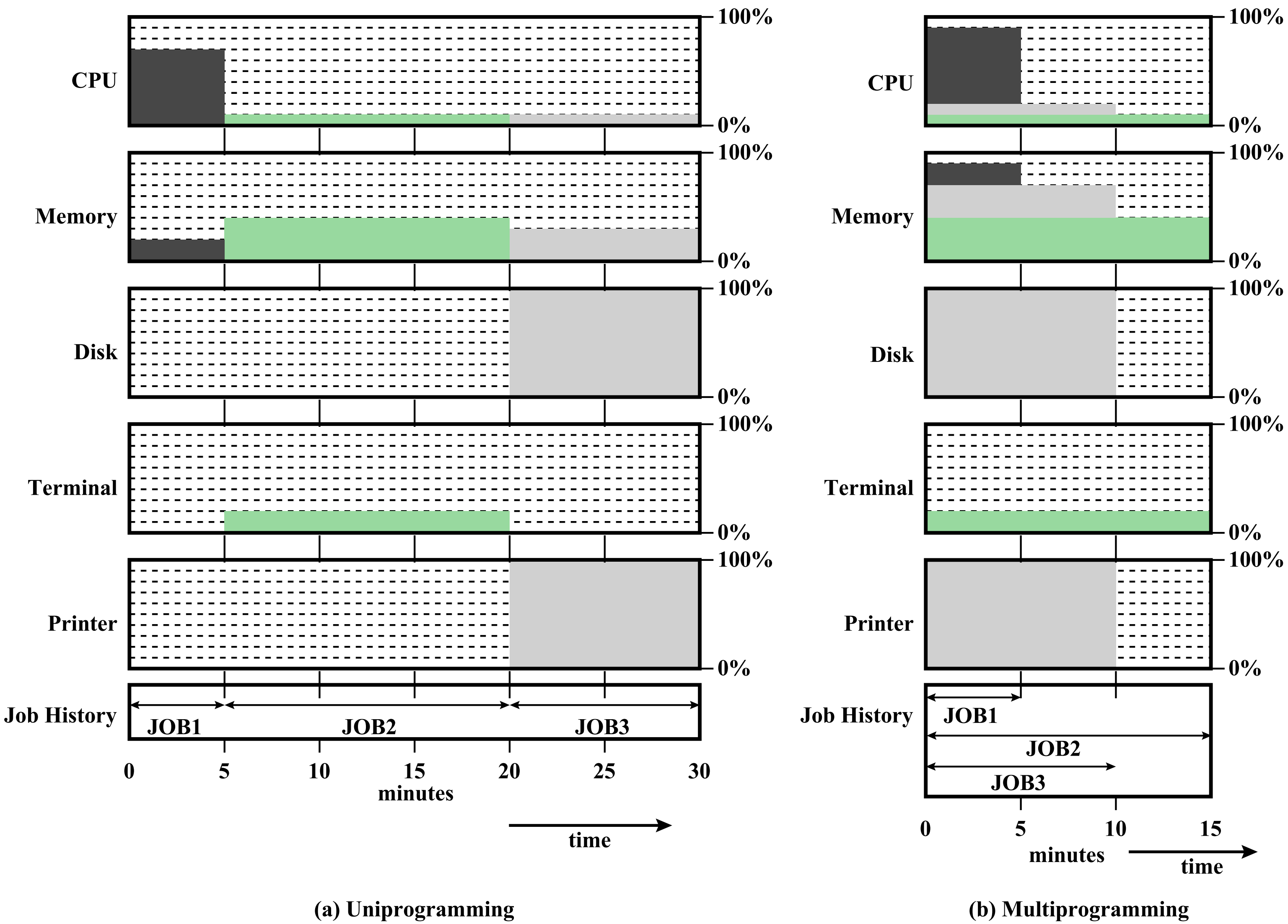
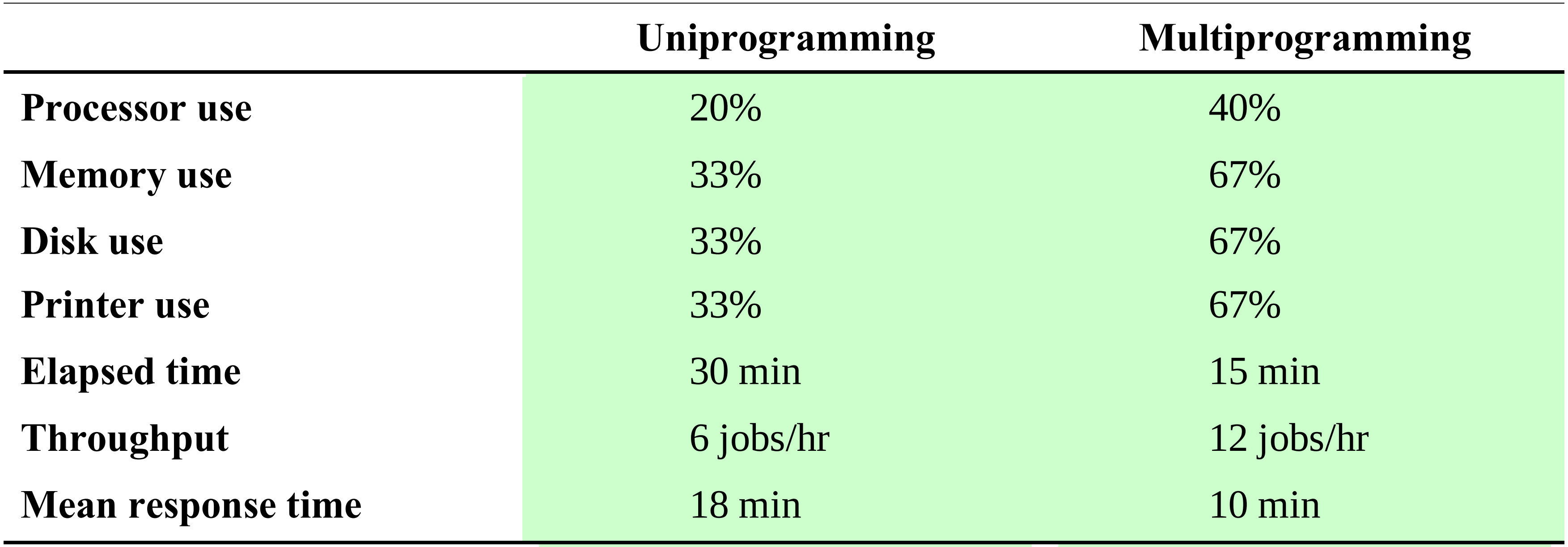


Figure 2.6 Utilization Histograms

Effects on Resource Utilization



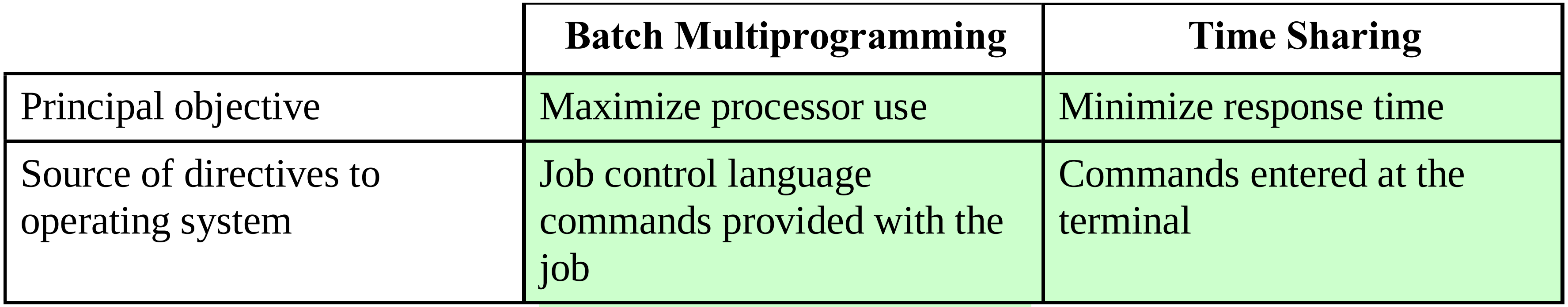
	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a **short burst** or **quantum** of computation

Batch Multiprogramming vs. Time Sharing



	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 2.3 Batch Multiprogramming versus Time Sharing

Compatible Time-Sharing Systems (CTSS)

- One of the first time-sharing operating systems (at MIT by Project MAC)

- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that. To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

- System clock generates interrupts at a rate of approximately one every 0.2 seconds. At each interrupt OS regained control and could assign processor to another user (**Time Slicing**)

- At regular time intervals the current user would be preempted and another user loaded in. Old user programs and data were written out to disk. Old user program code and data were restored in main memory when that program was next given a turn.

Memory Requirements

- JOB1: 15000
- JOB2: 20000
- JOB3: 5000
- JOB4: 10000

Loading order

J1 > J2 > J3 > J1 > J4 > J2

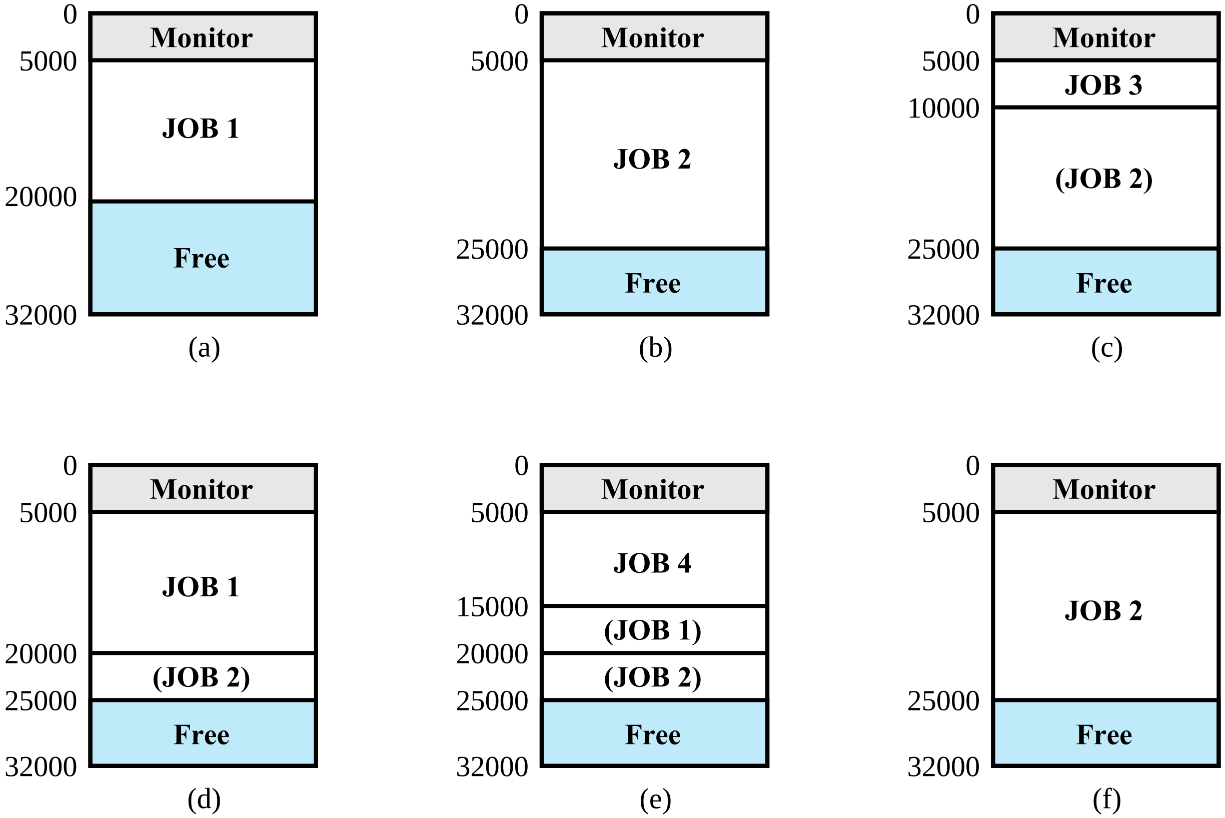


Figure 2.7 CTSS Operation

Major Achievements

- Operating Systems are among the most complex pieces of software ever developed

Major advances in development include:

- Processes
- Memory management
- Information protection and security
- Scheduling and resource management

Different Architectural Approaches

- Demands on operating systems require new ways of organizing the OS
- Different approaches and design elements have been tried:
  - monolithic
  - microkernel architecture
  - multithreading
  - symmetric multiprocessing
  - distributed operating systems
  - object-oriented design

Fault Tolerance

- Refers to the ability of a system or component to continue normal operation despite the presence of hardware or software faults
- Typically involves some degree of redundancy
- Intended to increase the reliability of a system
  - typically comes with a cost in financial terms or performance
- The extent adoption of fault tolerance measures must be determined by how critical the resource is

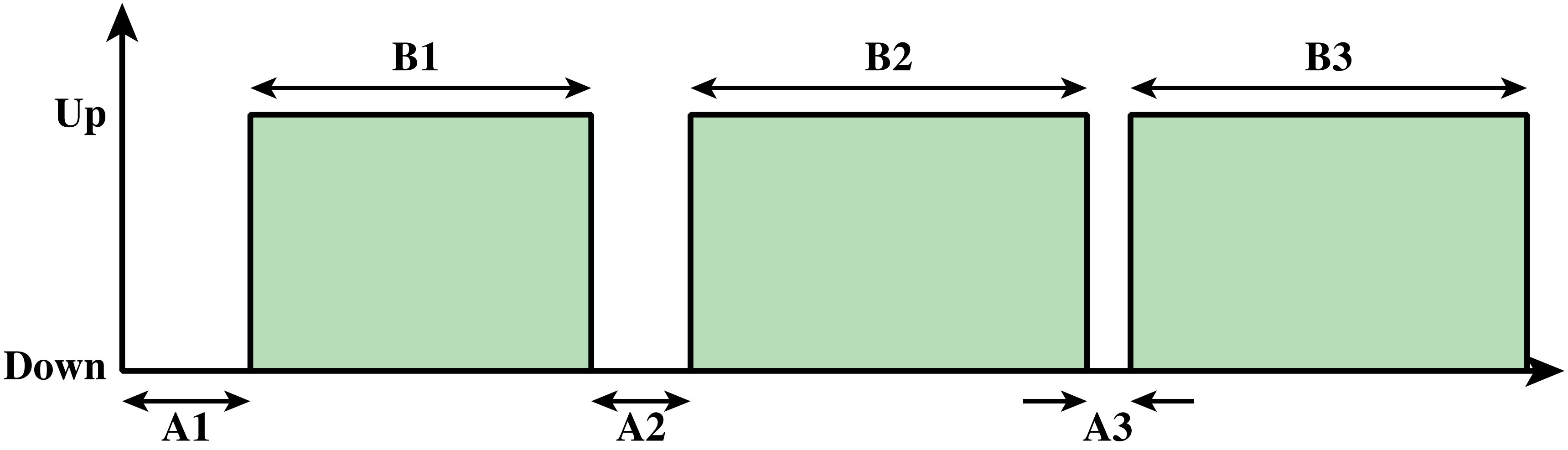
Fundamental Concepts

- The basic measures are:
  - Reliability
    - ***R(t):*** defined as the probability of its correct operation up to time *t* given that the system was operating correctly at time *t=0*
  - Mean time to failure (MTTF)

$$MTTF = \int_{0}^{\infty} R(t)$$

  - Mean Time To Repair (MTTR) is the average time it takes to repair or replace a faulty element
  - Availability (A)
    - defined as the fraction of time the system is available to service users' requests

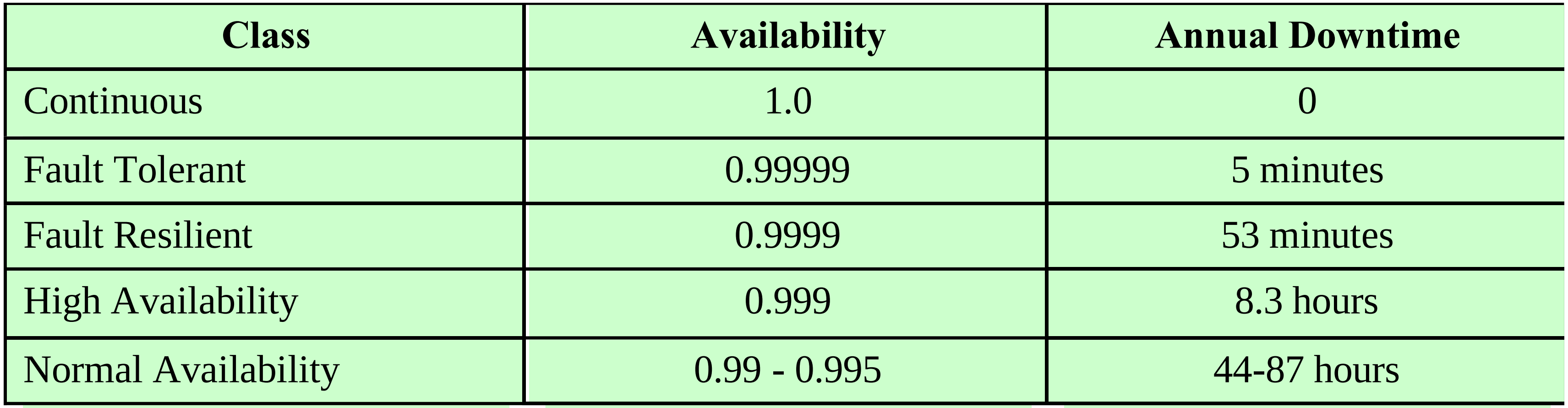
$$A = \frac{MTTF}{MTTF + MTTR}$$



$$MTTF = \frac{B1 + B2 + B3}{3} \qquad MTTR = \frac{A1 + A2 + A3}{3}$$

Figure 2.13 System Operational States

Availability Classes



Class	Availability	Annual Downtime
Continuous	1.0	0
Fault Tolerant	0.99999	5 minutes
Fault Resilient	0.9999	53 minutes
High Availability	0.999	8.3 hours
Normal Availability	0.99 - 0.995	44-87 hours

Table 2.4 Availability Classes

Operating System Mechanisms

- A number of techniques can be incorporated into OS software to support fault tolerance:
    - Process isolation
    - Concurrency control
    - Virtual machines
    - Checkpoints and rollbacks

Summary

- Memory hierarchy is the solution to the conflicting trade-off among cost, capacity, access time
- Cash is invisible to OS but principles of cash is applied in virtual memory
- DMA frees processor from the burden of I/O
- OS evolved from batch processing to multi-programming to time-sharing

- Fault tolerance is also a consideration in OS design

References

- **Operating Systems – Internal and Design Principles**
- By William Stallings
  - Chapter 2