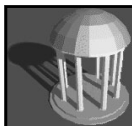


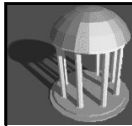
COMP 190 – Enterprise Systems

Fault-Tolerant and Highly-Available Distributed Systems

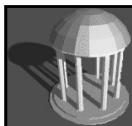
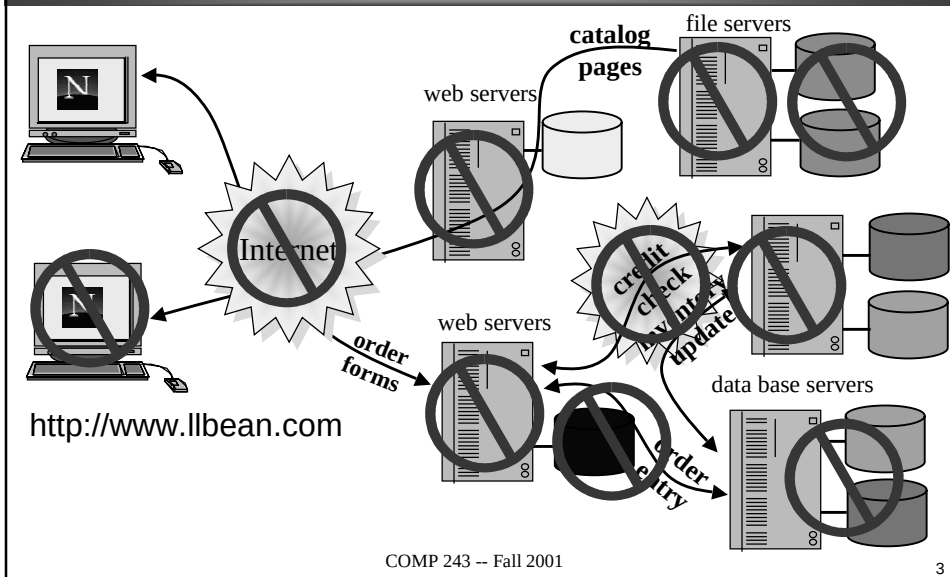


Remember the Goal:

- ◆ Provision of an Ongoing, Reliable, Responsive Service to Many Users
 - ◆ Ongoing  Highly-Available
 - ◆ Reliable  Fault-Tolerant
- ◆ Highly-Available: almost never “down” for recovery or repairs
- ◆ Fault-Tolerant: provides correct results even when there are failures

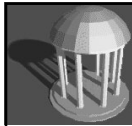


Can Your Customer Place an Order?



Basic Model of Failures

- ◆ A failure can be modeled as a random event, X , with some probability, $P(X)$, that the event will occur in a given interval of time (e.g., hour, day, year, etc.).
- ◆ In our model, we assume:
 - ◆ Failures of individual system components are independent
 - ◆ The probabilities, $P(X)$, are small (< 0.1)
 - choose time intervals where this is true
 - ◆ Failure distribution functions are “memoryless”
 - e.g., Poisson



Basic Model of Failures

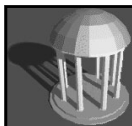
For a set of components, A, B, C,, with probabilities of failure in some interval of $P(A)$, $P(B)$, $P(C)$, respectively:

The probability that **all** will fail in the interval
 $= P(A) * P(B) * P(C) * \dots\dots\dots$

The probability that **one** will fail in the interval
 $= P(A) + P(B) + P(C) + \dots\dots\dots$

The mean time to failure, MTTF, of a component, A, with probability of failure in some interval, $P(A)$, is $1/P(A)$

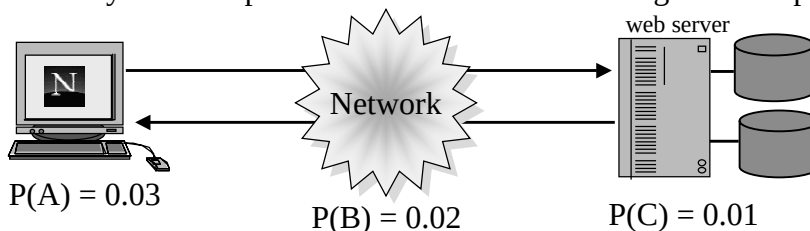
The MTTF of **one of** components A, B, C,
 $= 1/[P(A) + P(B) + P(C) + \dots\dots\dots]$



Basic Model of Failures

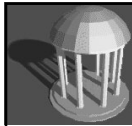
Example: for 24 hour period

Three system components are needed to browse a given web page:



What is the probability that I will not be able to read this web page during some 24 hour period?

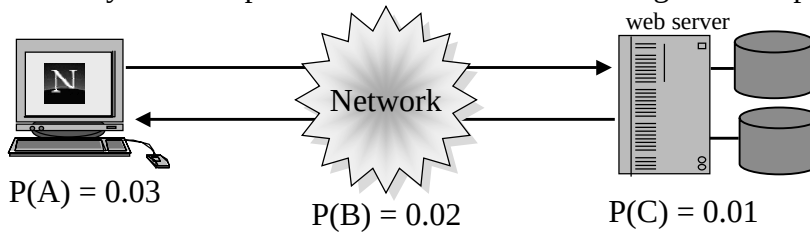
= probability that one component fails
 $= P(A) + P(B) + P(C)$
 $= 0.03 + 0.02 + 0.01$
 $= 0.06$



Basic Model of Failures

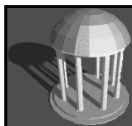
Example: for 24 hour period

Three system components are needed to browse a given web page:



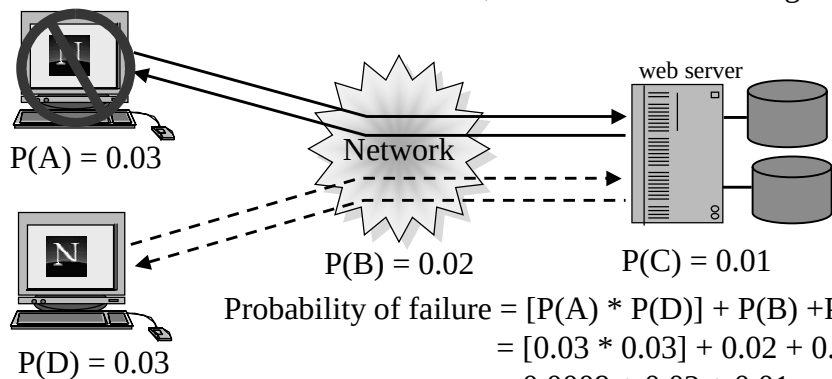
What is the mean time to when the web page will be unavailable (MTTF)?

$$\begin{aligned} &= 1/[\text{probability that one component fails}] \\ &= 1/0.06 \\ &= \text{about 17 days} \end{aligned}$$



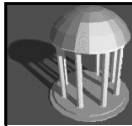
Basic Model of Failures

If I can use either of two workstations, how does MTTF change?



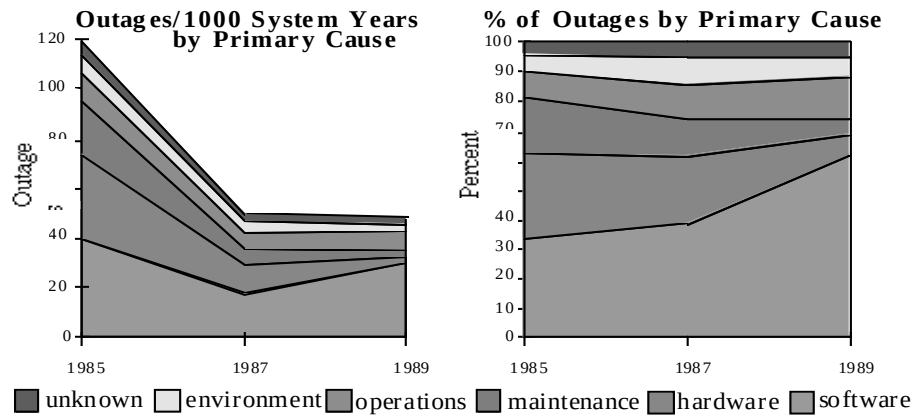
$$\begin{aligned} \text{Probability of failure} &= [P(A) * P(D)] + P(B) + P(C) \\ &= [0.03 * 0.03] + 0.02 + 0.01 \\ &= 0.0009 + 0.02 + 0.01 \\ &= 0.0309 \end{aligned}$$

$$\text{MTTF} = 1/0.0309 = 32 \text{ days}$$



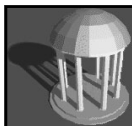
Studies of Real Systems (Gray)

Study of Tandem Customers, 1985-1990



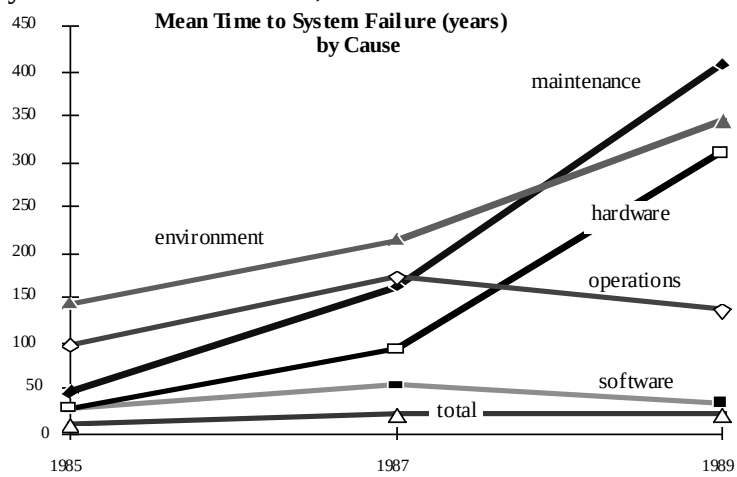
COMP 243 -- Fall 2001

9



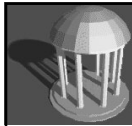
Studies of Real Systems (Gray)

Study of Tandem Customers, 1985-1990



COMP 243 -- Fall 2001

10

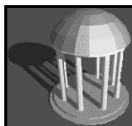


“Repair” Failures for Availability

- ◆ Along with MTTF, availability also depends on the mean time to “repair” (MTTR), the time it takes to restore service after a failure.
- ◆ System availability is usually expressed as:
 - ◆ $MTTF / (MTTF + MTTR)$
- ◆ For our web browsing examples if it typically takes 2 days to repair (or replace) a workstation,
 - ◆ availability = $17 / (17 + 2) = 89.5\%$ (for one workstation)
 - ◆ availability = $32 / (32 + 2) = 94\%$ (for two workstations)
- ◆ Availability of 99.999% is “high availability”
About 6 minutes of “down time” per year

COMP 243 -- Fall 2001

11

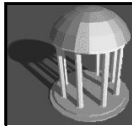


Repair for Availability

- ◆ If the MTTF for some software process is **100 hours**, then 99.999% availability can be achieved only if MTTR is **3.6 seconds**.
- ◆ You can’t reboot a machine in 3.6 seconds!
- ◆ Designing for availability means designing for “repair”, e.g., redundancy and rapid “take-over”
- ◆ Repair must be **fault-tolerant** -- it must not introduce errors (or more failures) that prevent correct operation of the system.

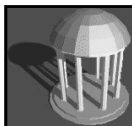
COMP 243 -- Fall 2001

12



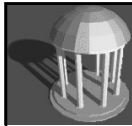
Failure Types and examples of MTTF/MTTR

◆ Disk (Server)	◆ 200,000 hours/24 hours
◆ Disk (PC)	◆ 20,000 hours/24 hours
◆ Processor (Server)	◆ 100,000 hours/24 hours
◆ Processor (PC)	◆ 50,000 hours/24 hours
◆ LAN	◆ 1000 hours/2 hours
◆ WAN	◆ 2000 hours/8 hours
◆ Power	◆ 3000 hours/1 hour
◆ Software Process	◆ 3000 hours/3 hours

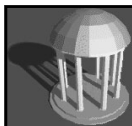
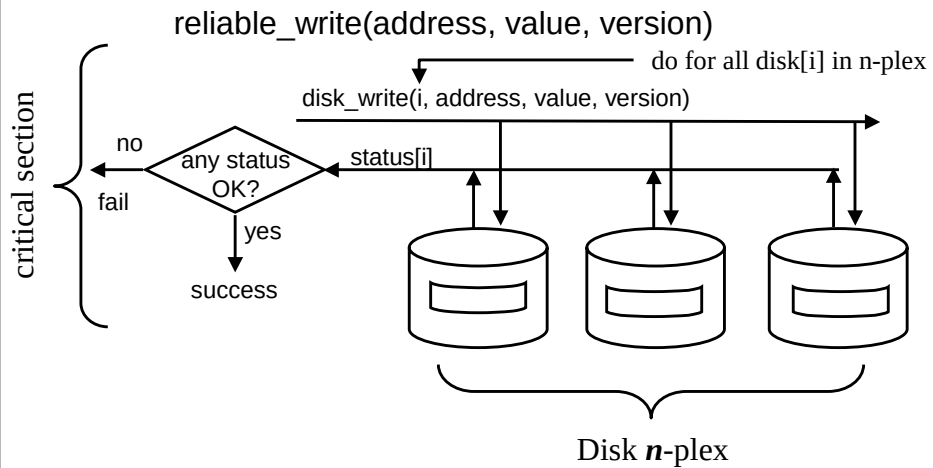


The Building Blocks of Availability

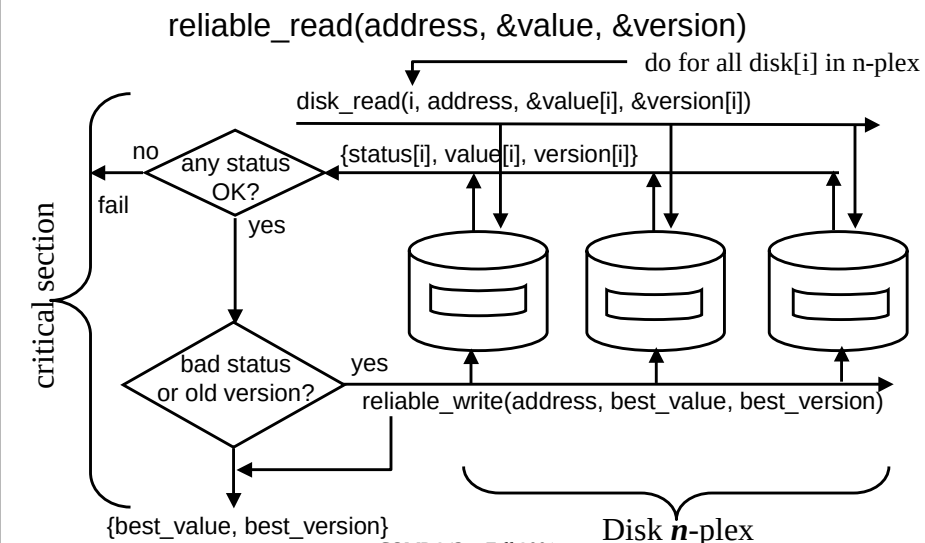
- ◆ Reliable Messages
 - ◆ Sequence numbers, timeout, and retransmit
- ◆ Highly-Available Storage
- ◆ Highly-Available Processes

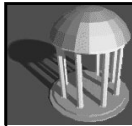


Storage reliable_write()



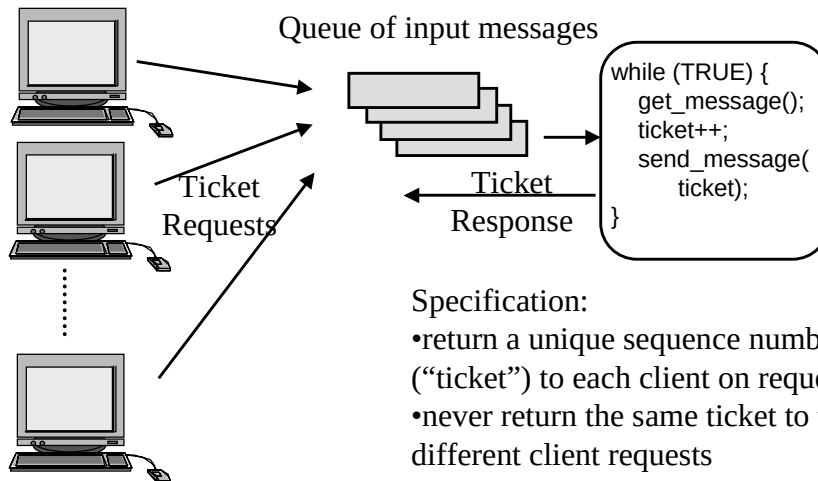
Storage reliable_read()





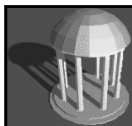
An Example: The Ticket Generator

How can we make a fault-tolerant, highly available ticket generator?

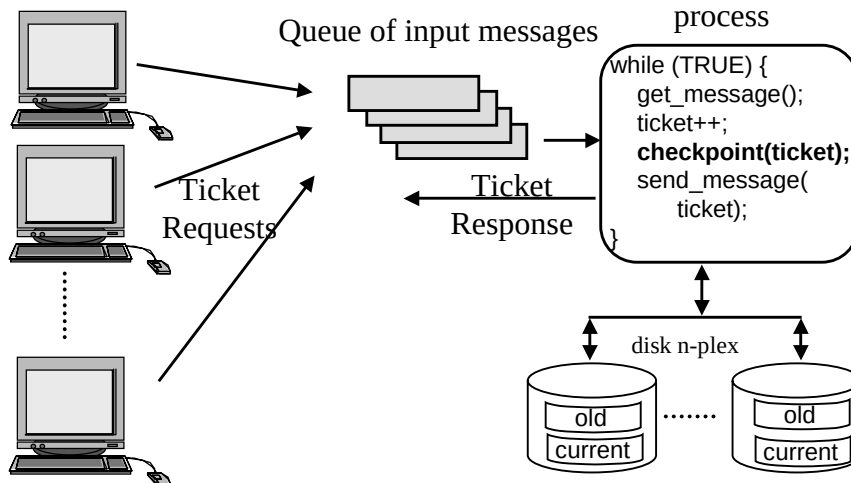


COMP 243 -- Fall 2001

17

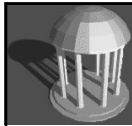


Solution 1: Process CheckPoint-Restart



COMP 243 -- Fall 2001

18

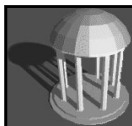


Solution 1: Process CheckPoint-Restart (adapted from Gray & Reuter, sect. 3.7.3.3)

```
checkpoint_restart_tickets {
/* restart logic, recover current number from checkpoint */
for (old=0;old<=1,old++) {
    if (reliable_read(address[old], &value[old], &version[old]) == FAIL)
        panic();
    if (value[1] < value[0]) {
        old = 1; ticket = value[0];
    } else {
        old = 0; ticket = value[1];
    }
/* normal processing logic -- checkpoint ticket before reply */
while (TRUE) {
    get_message(&msg); /* blocks until message in queue */
    ticket++; version++;
    if (reliable_write(address[old], ticket, version) == FAIL)
        panic();
    old = (old + 1) % 2;
    send_message(msg.sender, ticket);
}
}
```

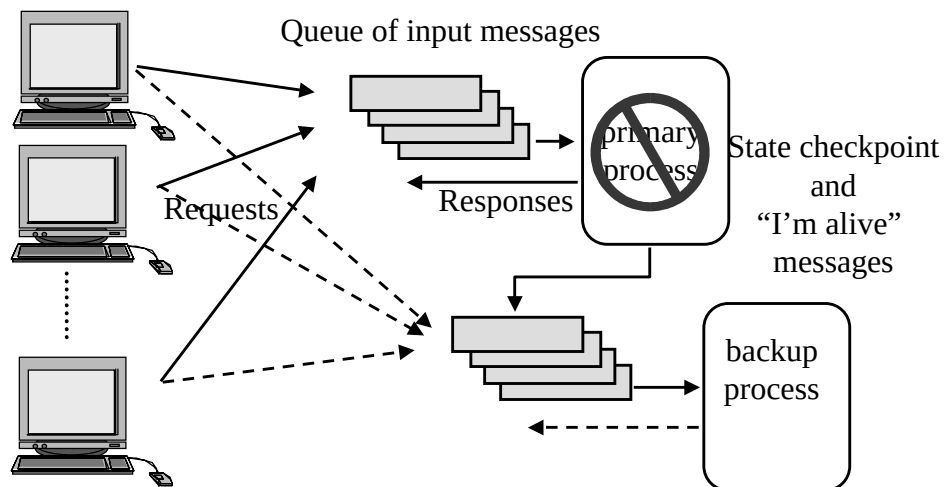
Why do we need to keep
2 values on disk?

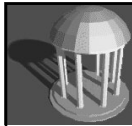
***Achieved fault-tolerance but
did little for MTTR!***



Solution 2: Primary-Backup Process Pairs

Works for single failures at a time (server process)



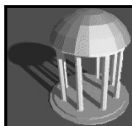


Solution 2: Primary-Backup Process Pairs (adapted from Gray & Reuter, sect. 3.7.3.4)

```
process_pair_tickets { /* select primary or backup role */
last_contact = time_now(); /* initialize for later test */
if (default_backup == MY_ID) /* going to default to primary or backup? */
    sleep(2 * SLEEP_INTERVAL); /* will see primary's "I'm Alive" and be backup*/
else sleep (SLEEP_INTERVAL); /* will become primary if backup not running*/
while(TRUE) { /* main backup loop */
    while (message_available(from_pair)) { /* can't let get_next... block, why ?? */
        get_next_message(&msg); /* checkpoint/alive from primary */
        last_contact = time_now(); /* note current time of last contact */
        if (msg.ticket > ticket) { /* critical -- must know if state updated */
            ticket = msg.ticket; /* use ticket # as state sequence # */
            last_client = msg.client; /* save as part of new state */
        }
    } /* end while loop looking for queued messages */
    if ((time_now() - last_contact) > SLEEP_INTERVAL)
        break; /* exit backup loop and prepare to be primary */
    else
        sleep(SLEEP_INTERVAL); /* pause before looking again */
} /*end main backup loop */
```

COMP 243 -- Fall 2001

21

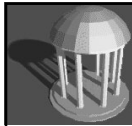


Solution 2: Primary-Backup Process Pairs (continued)

```
/* prepare to be primary , tell all clients to use this address */
notify_clients(client_list, MY_ID); /* how done depends on type of communications */
if (last_client != NULL) /* respond to last client; duplicates OK */
    send_message(last_client, ticket);
/* now in primary main loop */
while(TRUE) {
    while (message_available(from_client)) { /* can't let get_next... block, why ?? */
        get_next_message(&msg); /* request from client */
        checkpoint_msg.client = msg.client; /* save last client */
        checkpoint_msg.ticket = ticket++; /* increment and save new ticket state */
        send_message(pair, checkpoint_msg); /* send state to backup */
        send_message(msg.client, ticket); /* respond to client with ticket */
    } /* end loop checking for requests */
    send_message(pair, checkpoint_msg); /* send old checkpoint for "I'm alive" */
} /*end primary main loop */
```

COMP 243 -- Fall 2001

22



System Pairs

- ◆ Run primary and backup processes on independent systems to provide high availability
 - ◆ $\text{MTTF}(\text{both CPUs fail}) = \text{MTTF}(\text{either CPU fails})^2$
 - ◆ $\text{MTTF}(\text{power fails to both systems}) = \text{MTTF}(\text{power fails to either system})^2$
 - ◆ $\text{MTTF}(\text{network connections to both systems fail}) = \text{MTTF}(\text{network connection to either system})^2$
 - ◆ ...etc.
- ◆ MTTF increases exponentially, and MTTR can be small (50 - 5000 milliseconds) depending on network latency.