About the midterm exam

- Basic concepts
- Questions have similar styles as our Assignments
- Keep your answer concisely and list the point one by one

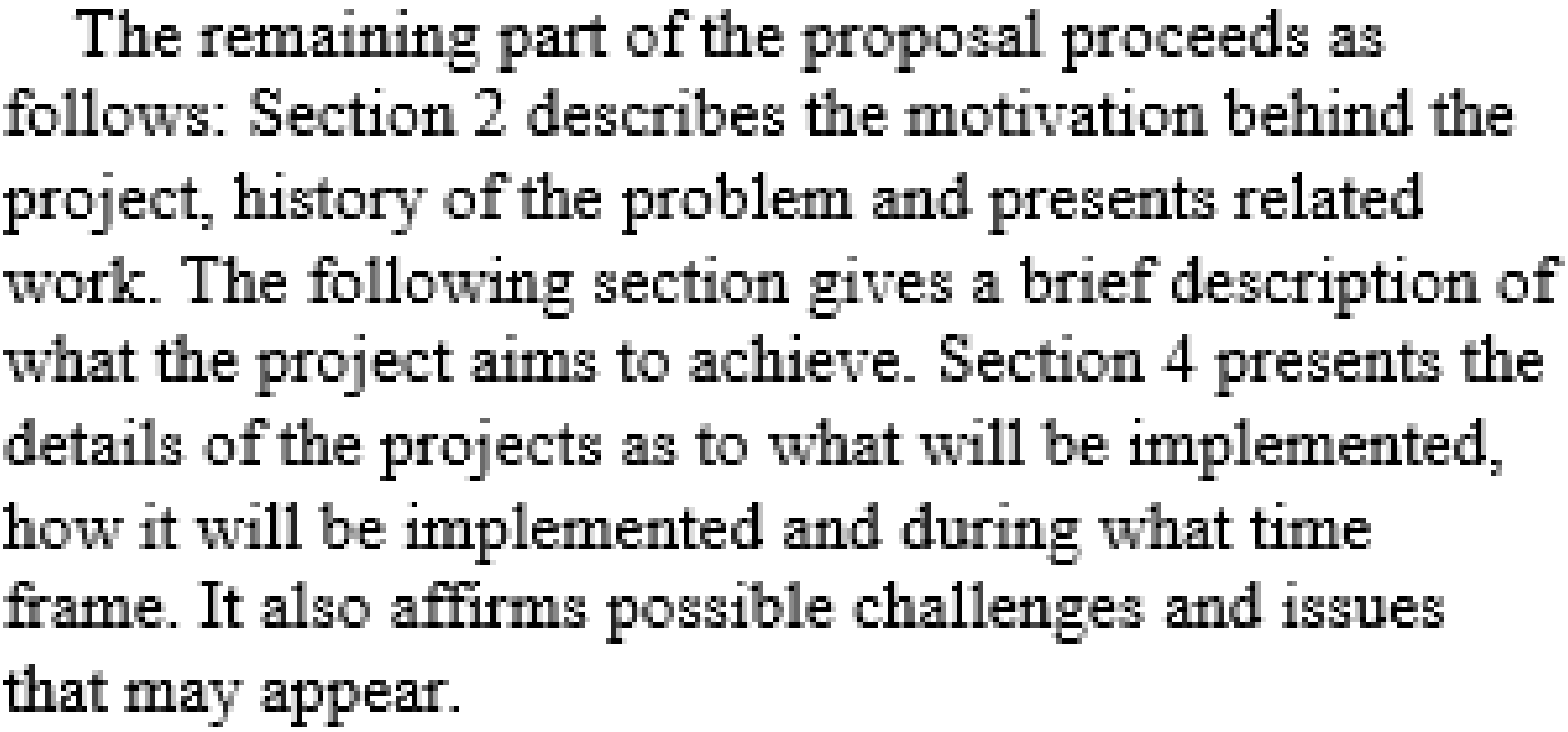
Common Issues in Proposals

- Some proposals have low research merits (e.g., only comparing some existing algorithms)
- Inappropriately capitalizing the first letter of some words such as “Distributed Computing” even in the middle of a sentence.
- Obvious grammar errors and long sentences (e.g., see next slide)
- Copy a figure from other papers or figures are very large
- Forget to embed references into the body of text or put references in a wrong position.
- Poor formatting (e.g., inconsistent in capitalization, zigzag)

Example of A Very Long Sentence

- Problem with the above system is its static and such as occlusion and shadow overlapping exist in this algorithm these problems can be handled by applying efficient shadow removal and occlusion removal techniques however they are not efficient due to several shortcomings in the algorithm itself that is every shadow of the vehicle is treated as an object which leads to wrong computation of the traffic density.

Zigzag



The remaining part of the proposal proceeds as follows: Section 2 describes the motivation behind the project, history of the problem and presents related work. The following section gives a brief description of what the project aims to achieve. Section 4 presents the details of the projects as to what will be implemented, how it will be implemented and during what time frame. It also affirms possible challenges and issues that may appear.

Examples of Poor Formatting

- DYNAMIC TRAFFIC SWITCHING USING PROXIMITY SENSORS
- Ultra-Large-Scale Systems: Geographical Analytics and its growing superiority in the modern world
- Forget to indent the first line of a paragraph
- It should be “Xie et al. [5]” instead of “Tao Xie et al. [5]”
- Many short paragraphs (see next slide)

Many Short Paragraphs

Like most replication algorithms, the DAR algorithm seeks to balance the server load, and it is successful under simplified and constrained conditions [1]. Server load is characterized by the number of movie requests that cannot be fulfilled by peers, and must be fulfilled by a centralized server.

When a peer watches a movie and has to decide to replicate the movie or not it has to talk to each peer streaming the movie at that point in time and then calculate the average streaming rate of the movie.

The subject of interest in this project is how peers exchange information concerning movie download speeds with other peers. One existing solution is to store global information in a central server or servers. However, servers become over loaded when large numbers of peers exchange information via a centralized server [2].

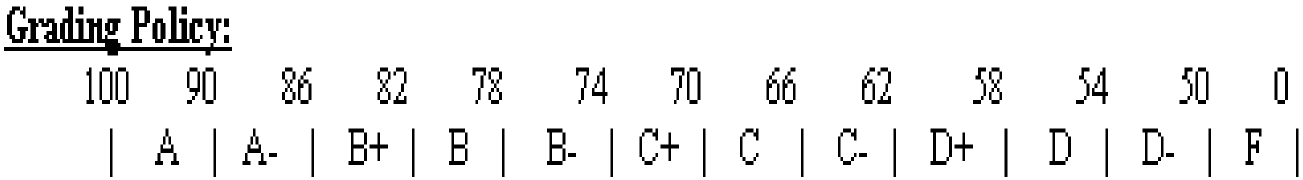
The other option is searching the peer to peer network to retrieve the information. Many original open source p2p networks used simple search algorithms. For example Gnuttela protocol implemented a modified Breadth-First Search (BFS) mechanism[5]. More recent P2P networks implement distributed hash tables as an overlay network to more efficiently route information among peers [2].

Regardless of the methodology employed for communication among peers in a network, reducing the number of peers to be queried is the problem we address is this project.

Research Project

- Your idea is critical
- The new the better
- Your efforts in terms of implementation and experiment results are very important.
- Results: The more the better
- Your writing performance will also be evaluated

Final grading policy



Grading Policy:

100 90 86 82 78 74 70 66 62 58 54 50 0

| A | A- | B+ | B | B- | C+ | C | C- | D+ | D | D- | F |

Election Requirements

- Unique process must be selected even if multiple runs initiated concurrently
- Pick process with largest identifier
  - identifiers under our control
  - unique, total ordering
- elected$_k$ - track elected process

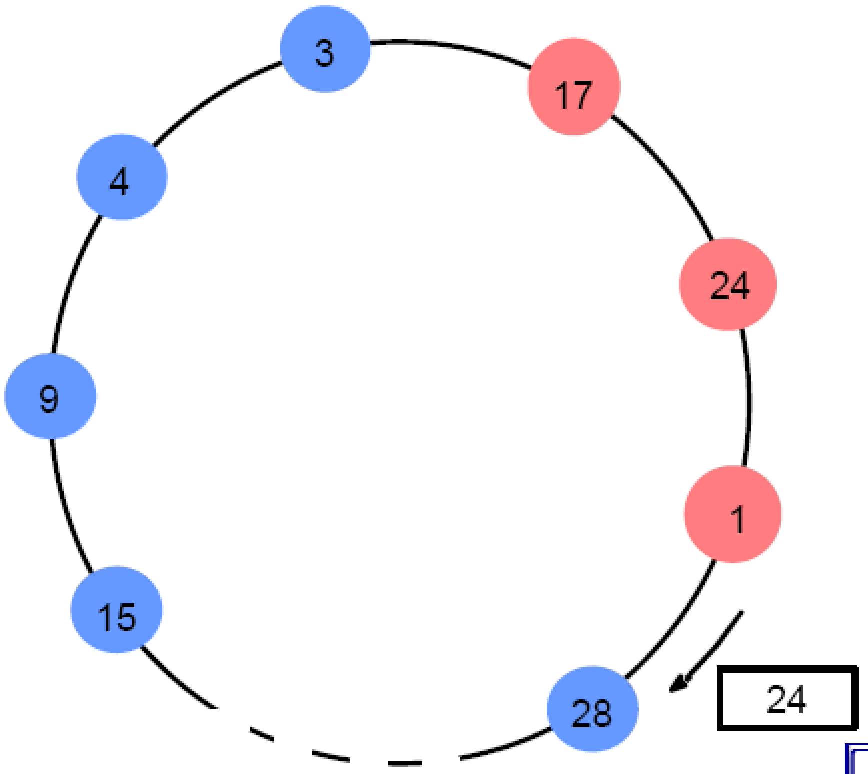
E1 (safety)	elected$_k$ is either $\perp$ or non-crashed process P with the largest identifier
E2 (liveness)	every process p either picks elected$_k \neq \perp$ or crashes

© Pearson Education 2005

Ring-Based Election Chang and Roberts

Notes:

- The election was started by process 17.
- The highest process identifier encountered so far is 24.
- Participant processes are shown **darkened**



© Pearson Education 2005

Ring Election Algorithm

```
Initialize
    participate := FALSE;
    coordinate := FALSE;
    elected := $\perp$;
On calling election
    participate := TRUE;
    Send election message $<p_i>$ to
    next process
Elected message $[p_i]$ at $p_k$
    participate := FALSE;
    elected := $p_i$ ;
    if (not coordinate)
        Send $[p_i]$ to next process;
    end if
```

```
Election message $<p_i>$ at $p_k$
    if ($p_i > p_k$) then
        Send $<p_i>$ to next process;
    else if ($p_i = p_k$) then
        coordinator := TRUE;
        participate := FALSE;
        elected := $p_k$ ;
        Send $[p_k]$ to next process;
    else if (not participate)
        participate := TRUE;
        Send $<p_k>$ to next process;
    end if
```

© Pearson Education 2005

E1 Satisfied
Proof by contradiction

- Suppose two processes p and q both set themselves as coordinators
  - Since only the winner of comparison is forwarded and identifiers are unique:
    - p is the largest of all identifiers
    - q is the largest of all identifiers
  - a contradiction
- process crashes during run?
  - Ring broken, none elected

© Pearson Education 2005

E2 is Satisfied
Direct Proof

- If there are no failures
  - all messages traverse the ring
  - comparison logic and the role of *participate* ensure that
    - largest identifier circulates
    - only one run is maintained
    - therefore, process with largest identifier sets itself as the coordinator and sends *elected* message
    - therefore, all processes set *elected* to be largest identifier
- If there are failures
  - none elected

© Pearson Education 2005

Characterizing Ring Election Algorithm

- Characteristics
  - Bandwidth
    - the number of messages sent to reach agreement
  - Turnaround Time
    - number of serialized messages transmitted from initiation to termination of a single run

- Characteristics of Ring Election
  - Bandwidth
  - Turnaround Time
- Failure Handling
  - election essentially stops
  - detection and reconfiguration needed to handle failure

© Pearson Education 2005

Ring-based algorithm

- Performance:
  - one election, best case, when?
    - ? *election* messages N
    - ? *elected* messages N
    - turnaround: ? messages $2N$
  - one election, worst case, when?
  - Drawback?

can't tolerate failures,
not very practical

- $2N$ - 1 election messages
- N *elected* messages
- turnaround: $3N$ - 1 messages

Bully Algorithm
Garcia and Molina

- Goal is handling process failures
- Assumptions
  - synchronous system
  - no message failures
  - processes know set of processes and identifiers
- Timeout used to identify process failures

© Pearson Education 2005

Bully Algorithm Basics

- Reliable Fault Detector
  - timeout = $2T_{trans} + T_{pr}$
- Types of Messages
  - election message
    - upon detection of coordinator failure
  - answer message
    - acknowledgment
  - coordinator message
    - announce new coordinator

- Basic Idea
  - process with highest id can elect itself
  - processes with lower ids ask higher processes
    - if answer does not come, elect self
    - if answer comes
      - await coordinator message
      - restart election if necessary

Bully Algorithm

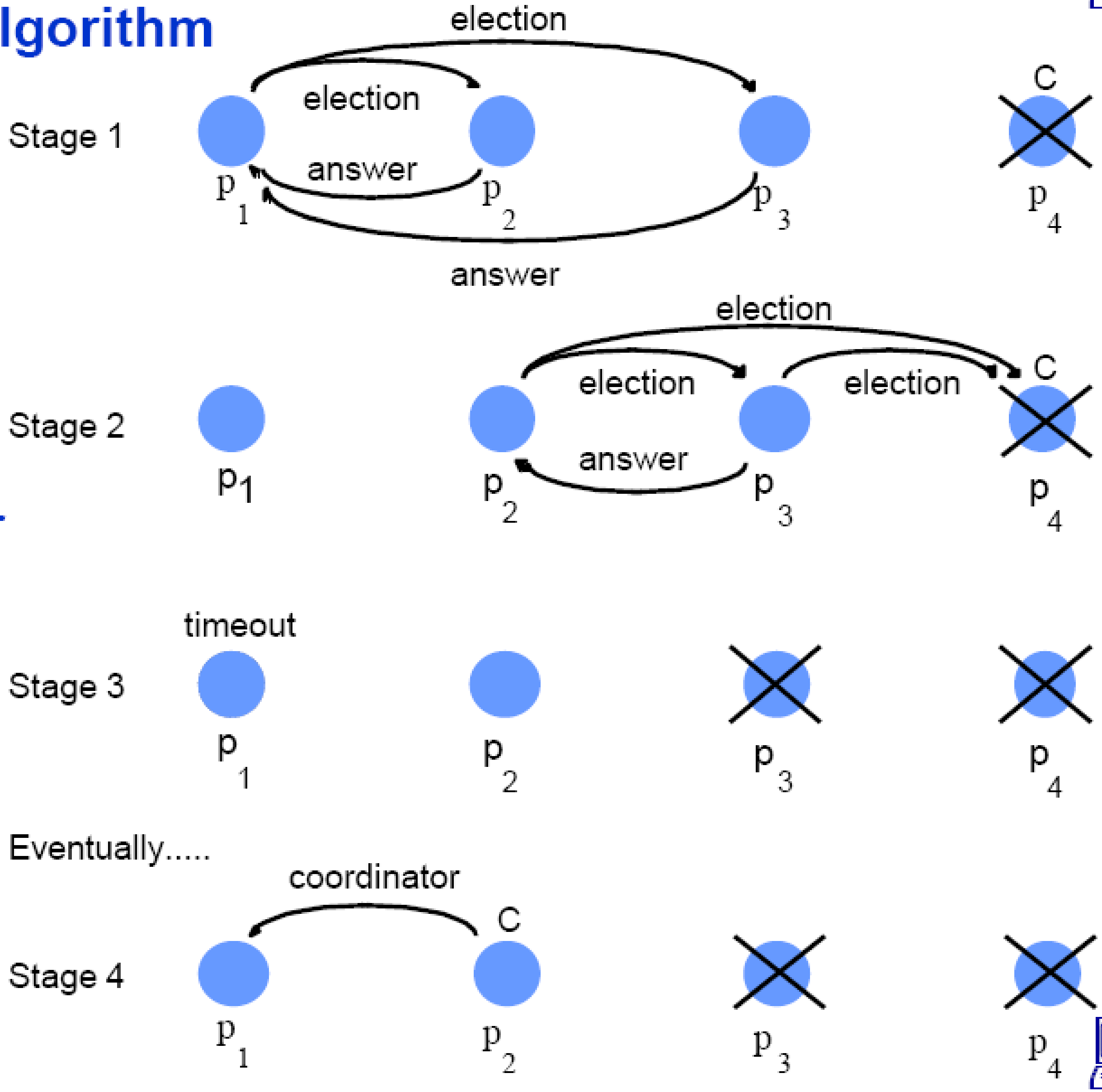
```
Initialize
    participate := FALSE;
    elected := ⊥;
    call election;
On calling election
    participate := TRUE;
    Send election message <$p_i$ > to each process with higher id
    if ( no answer within time $T$ )
      elected := $p_i$;
      Send coordinator message to each process with lower id;
    else if ( no coordinator within time $T'$ )
      begin election;
    end if
```

```
Election message <$p_i$ > at $p_k$
    Send answer message to $p_i$;
    if (not participate) then
      call election;
    end if
Coordinator message [$p_i$ ] at $p_k$
    participate := FALSE;
    elected := $p_i$ ;
```

© Pearson Education 2005

The Bully Algorithm

The election of coordinator p_2, after the failure of p_4 and then p_3



© Pearson Education 2005

Bully Algorithm Safety Property E1

- Assume no process crashes
  - consider any two (p and q) processes
  - let p be higher id process
  - channel failure is impossible
  - q cannot be elected, since p will *answer*
  - highest id process elected

- What if processes crash or timeout values fail?
  - Process crash
    - say coordinator p crashes
    - q decides it is new coordinator
    - new process spawned to replace p also elects itself
    - processes may receive [p] and [q] in different orders
  - timeout value fails
    - new coordinator elected
    - old coordinator is still alive

© Pearson Education 2005

Bully Algorithm Liveness Condition E2

- Messages do not fail
  - all *elected* messages reach alive destinations
  - all *answer* and *coordinator* messages also reach alive destinations
  - timeouts will detect failed processes
  - by earlier argument, highest id process that is alive will elect itself as the coordinator

© Pearson Education 2005

The bully algorithm

- properties:
  - safety:
    - a lower-id process always yields to a higher-id process
    - However, during an election, if a failed process is replaced
      - the low-id processes might have two different coordinators: the newly elected coordinator and the new process,
    - failure detection might be unreliable
  - liveness: all processes participate and know the coordinator at the end

Multicast Communication

- Multicast
  - group communication
  - requires coordination and agreement
  - issues
    - delivery guarantees
    - time and bandwidth needs
    - group membership rules

© Pearson Education 2005

Multicast Communication

- Basics
  - set of member processes is known
  - one multicast operation to send to each member in group
    - allows improved efficiency
    - allow stronger delivery guarantees
- Efficiency
  - same message is to be delivered to all in group
  - one copy per communication link over distribution tree
  - network hardware support where available
- Delivery Guarantees
  - multiple independent sends offer no control
  - multicast model can offer additional controls

© Pearson Education 2005

System Model

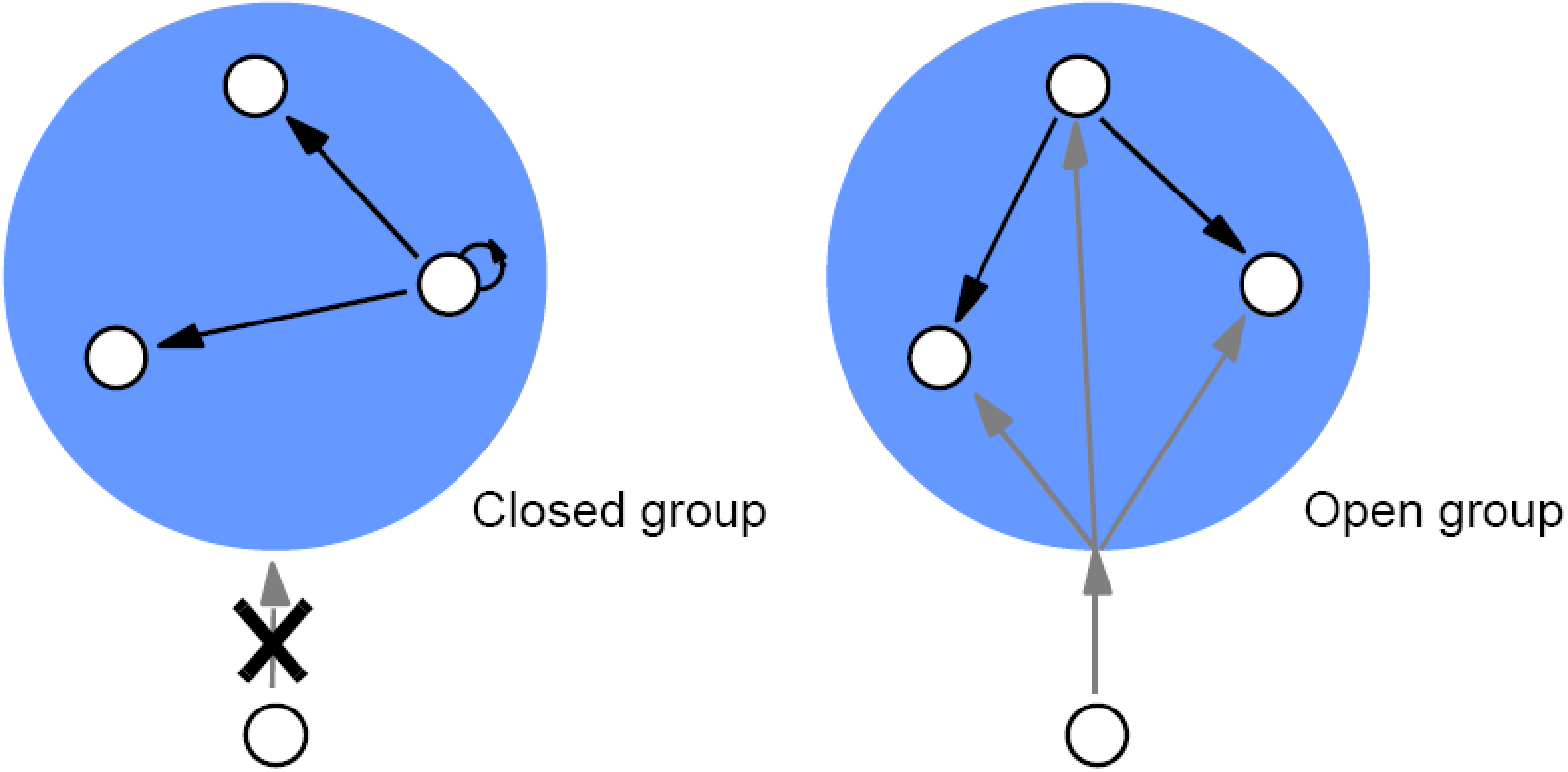
- Basics
  - Set or processes
  - reliable channels
  - processes can crash
  - process may belong to more than one group

- Operations
  - multicasting
    - **multicast(g,m) sends the message m to all members of group g**
  - delivering

    Why not “receiving”?
    - **deliver(m) delivers message to calling process**
  - message m identifies sender(m) and group(m)

© Pearson Education 2005

Open and Closed Groups



© Pearson Education 2005

Basic Multicast

- Basic Multicast
  - multicast primitive with guarantee that correct process will eventually deliver the message
  - unless multicaster crashes
  - operations
    - **B-multicast**
    - **B-deliver**

- Implementation
  - use a reliable one-to-one send as the underlying mechanism
    - **B-multicast(m)**
      - send(p,m) for each p in group g
    - **On receive(m) at p**
      - B-deliver(m) at p
  - problem
    - **ack-implosion**

© Pearson Education 2005

Reliable Multicast

- Reliable Multicast Operations
  - **R-multicast**
  - **R-deliver**
- Reliable Multicast Properties
- Integrity
  - **A correct process p delivers a message m at most once, where p is in group(m) and m was multicast by sender(m)**
- Validity
  - **If a correct process multicasts a message m then it will eventually deliver m**
- Agreement
  - **If a correct process delivers message m, then all other correct processes in group(m) will eventually deliver m**

© Pearson Education 2005

Reliable Multicast Algorithm A

```
On initialization
    Received := {};

For process p to R-multicast message m to group g
    B-multicast(g, m);        // $p \in g$ is included as a destination

On B-deliver(m) at process q with g = group(m)
    if ($m \notin$ Received)
    then
            Received := Received $\cup$ {m};
            if ($q \neq p$) then B-multicast(g, m); end if
            R-deliver m;
    end if
```

Reliable Multicast Algorithm A Integrity and Validity

- Integrity
  - B-multicast channel reliability
  - therefore, there is a B-deliver(m)
  - check for receipt of m in the algorithm
    - **R-deliver executed on first receipt of m**
    - **additional copies of m are ignored**

- Validity
  - B-multicast channel reliability
  - therefore, there is a B-deliver(m) at p
  - since, Received is initially empty
    - **p add m to Received**
    - **p R-delivers m**

© Pearson Education 2005

Each message is sent |g| times to each process.

Reliable Multicast Algorithm A Agreement

- B-deliver at process p implies R-deliver at process p
  - Obvious from algorithm
- Suppose correct processes p and q differ on R-delivery
  - without loss of generality, p does R-deliver and q does not
    - only happen if no B-deliver at q
    - impossible due to B-multicast channel reliability

Efficiency?

Reliable Multicast over IP Multicast

- IP Multicast Properties
  - messages may be lost
  - no acknowledgments
- How can we add reliability on top of IP multicast?
  - acknowledgments
  - efficiency of UDP lost?
- How can we maintain the efficiency of UDP?
  - piggy back acknowledgments
    - no separate acks
  - negative acknowledgments
    - indicate that expected message did not arrive

© Pearson Education 2005

Reliable Multicast Algorithm B

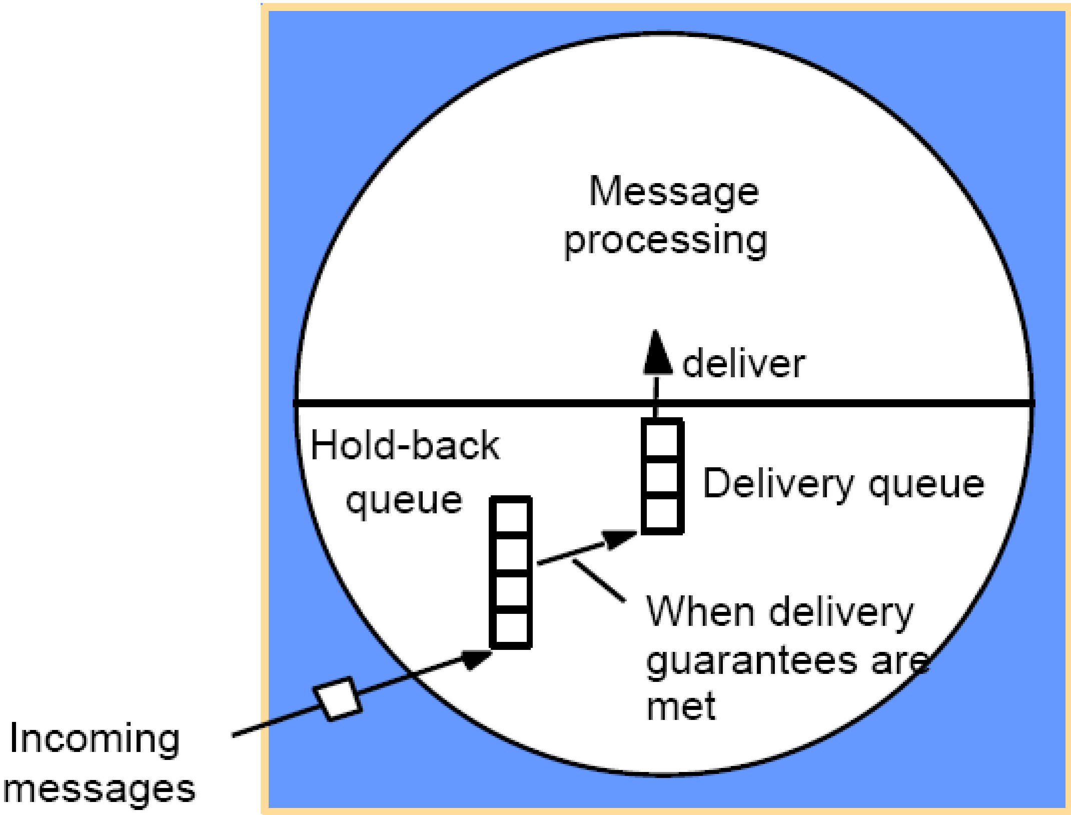
```
Initialize
    S.g := 0; // group sequence #
    R[g,p] := -1; // message seq #
    F.g := self; // from process
R-multicast(g,m)
    piggy back S.g on m;
    piggy back <F.g, R[g,F.g] > on m;
    IP-multicast(g,m);
    S.g := S.g + 1;
```

Holdback Queue Handling?

Negative acknowledgment handling?

```
On IP-receive(m) at q
    Sm := group-seq(m);
    Rm := msg-seq(m);
    Fm := from(m);
    if (Sm < R[g,Fm] + 1) then
        discard m;
    else if (Sm = R[g,Fm] + 1) then
        R-deliver m;
        R[g,Fm] := R[g,Fm] + 1;
        F.g := Fm;
    else if (Sm > R[g,Fm] + 1 or
        Rm > R[g,Fm] ) then
        add m to holdback queue;
        send negative acks to …;
    end if
```

Holdback Queue for Arriving Multicast Messages



© Pearson Education 2005

Reliable Multicast Algorithm B Integrity and Validity

- Integrity
  - duplicate detection
    - **duplicates are detected and discarded**
  - IP-multicast property
    - **corrupt messages are discarded**
    - **rerequested as necessary**

- Validity
  - IP-multicast may drop messages
  - correct process and reliable channels imply
    - **missing messages detected**
    - **negative acknowledgment sent**

© Pearson Education 2005

Reliable Multicast Algorithm B Agreement

- Suppose correct processes p and q disagree on a message m
  - p delivered m but q did not
  - q will send a negative acknowledgment for m
  - Can some process redeliver m?
    - **only if a copy of message m exists at some process**
    - **must hold delivered messages indefinitely for agreement to hold**

© Pearson Education 2005

Reliable Multicast Uniform Properties

- Uniform Property
  - a uniform property holds independent of the correctness of processes
  - we can define uniform versions of any desired property
- Uniform Agreement
  - If any process deliver message m, then all correct processes in group(m) will eventually deliver m

- Consider algorithm A for Reliable Multicast
  - every process B-multicasts message prior to R-delivery
  - therefore, algorithm A satisfies the Uniform Agreement property

© Pearson Education 2005