CS 425/ECE 428/CSE424 Distributed Systems (Fall 2009)

Lecture 11
Failure Detectors
(Sections 12.1 and part of 2.3.2)
Klara Nahrstedt

Acknowledgement

- The slides during this semester are based on ideas and material from the following sources:
  - Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.
  - Slides from Professor S. Gosh's course at University o Iowa.

Administrative

- HW 2 posted September 22, Tuesday
  - Deadline, October 6 (Tuesday), 2pm (at the beginning of the class)

Plan for Today

- Failure detectors
- Properties – completeness & accuracy
- Two failure detector algorithms
  - Heart-beating and Ping-Ack
- Distributed Failure Distribution through heart-beating algorithms
  - Centralized, Ring, All-to-all
- Accuracy metrics
- Other Types of Failures

Two Different System Models

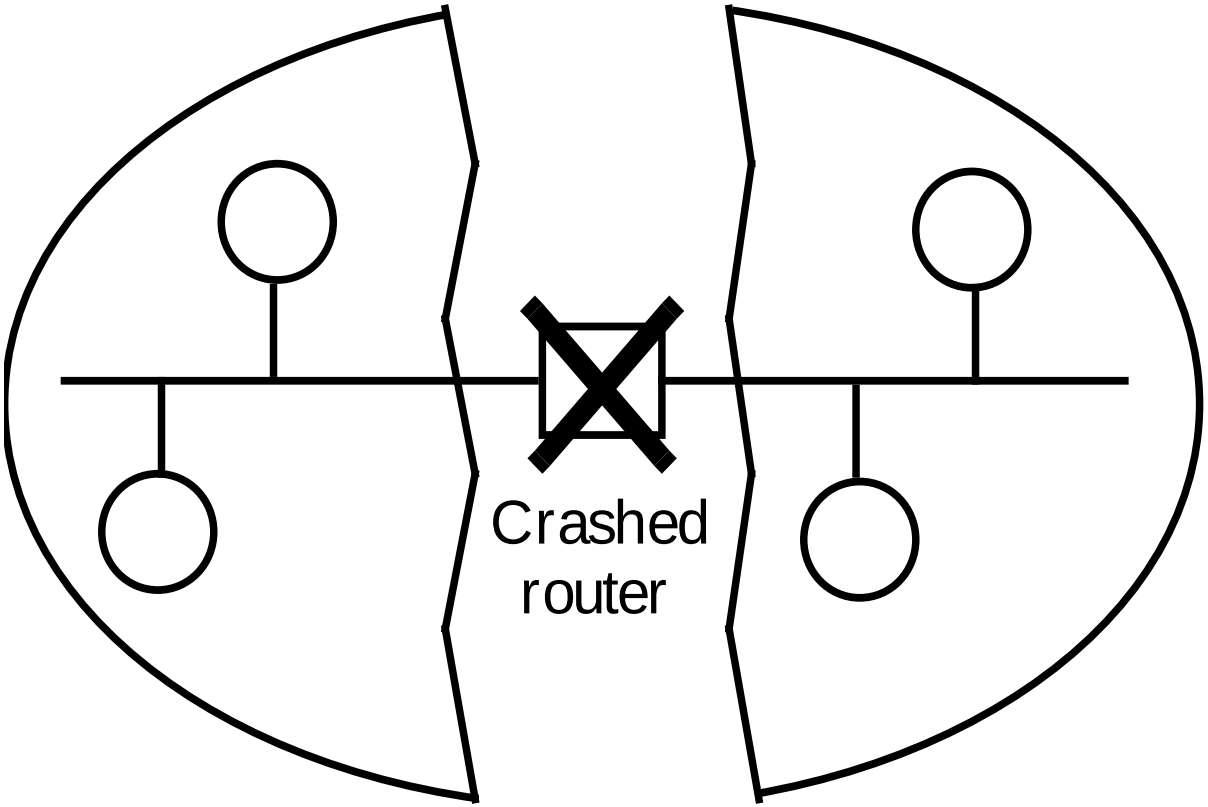
- **Synchronous Distributed System**
  - Each message is received within bounded time
  - Each step in a process takes ***lb*** < ***time*** < ***ub***
  - (Each local clock's drift has a known bound)
- **Asynchronous Distributed System**
  - No bounds on process execution
  - No bounds on message transmission delays
  - (The drift of a clock is arbitrary)

The Internet is an asynchronous distributed system

Failure Model

- **Process omission failure**
  - **Crash-stop (fail-stop) –** a process halts and does not execute any further operations
  - **Crash-recovery** – a process halts, but then recovers (reboots) after a while
- *Crash-stop* failures can be detected in synchronous systems
- Next: detecting *crash-stop* failures in asynchronous systems

Network partition



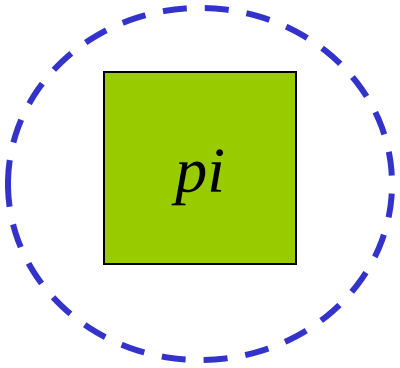
What's a failure detector?

What's a failure detector?

Crash-stop failure

What's a failure detector?

needs to know about *pj*'s failure

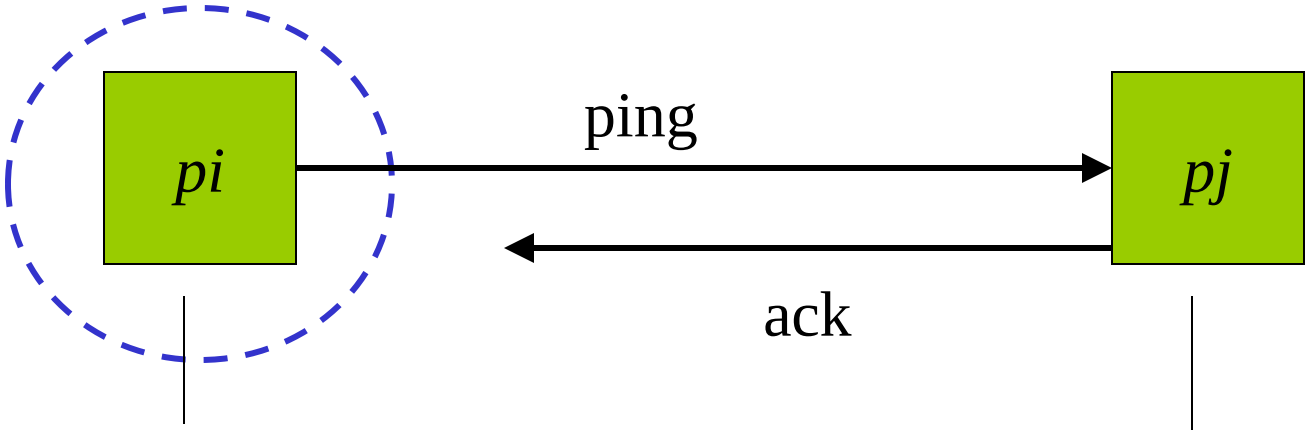


Crash-stop failure

I. Ping-Ack Protocol

needs to know about *pj*'s failure

If pj fails, within T time units, pi will send it a ping message, and will time out within another T time units. Detection time = 2T



- *pi* queries *pj* once every *T* time units

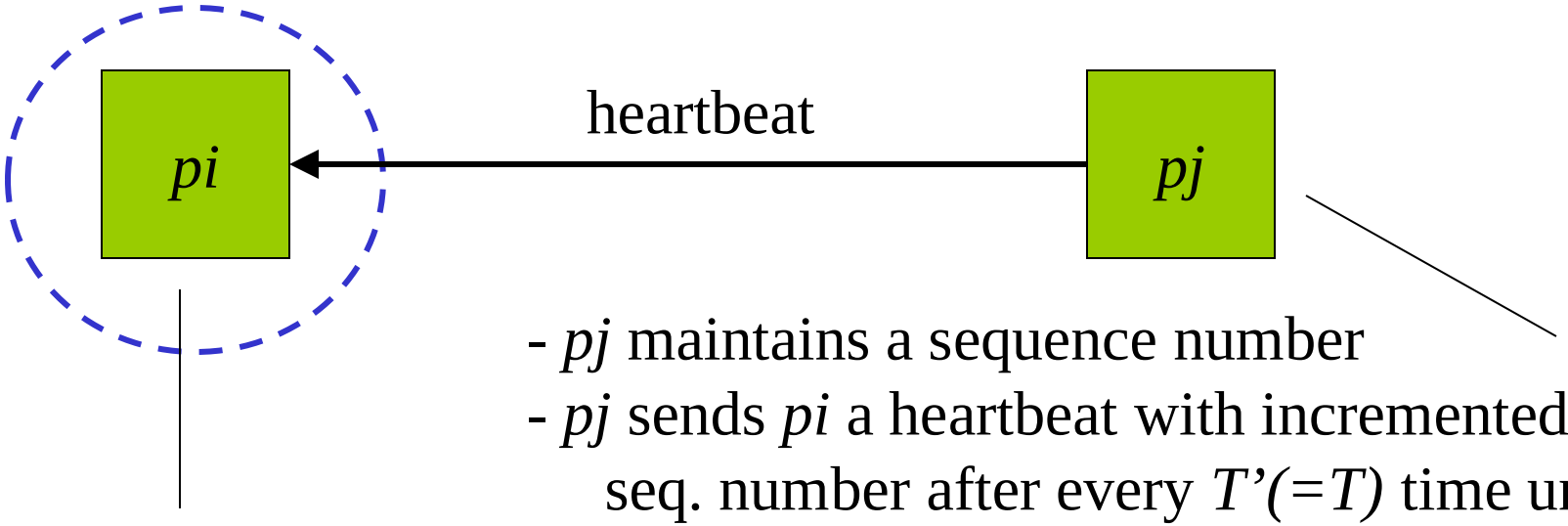
- *pj* replies

- if *pj* does not respond within *T* time units, pi marks *pj* as failed

II. Heart-beating Protocol

needs to know about *pj*'s failure

In reality, detection time
is also T time units (why?)



- *pj* maintains a sequence number
- *pj* sends *pi* a heartbeat with incremented seq. number after every *T'(=T)* time units

-if *pi* has not received a new heartbeat for the past *T* time units, *pi* declares *pj* as failed

*If pj has sent x heartbeats until the time it fails, then pi will timeout within (x+1)*T time units in the worst case, and will detect pj as failed.*

Failure Detector Properties

- **Completeness** = every process failure is eventually detected (no misses)
- **Accuracy** = every detected failure corresponds to a crashed process (no mistakes)
- Given a failure detector that satisfies both Completeness and Accuracy
  - One can show that Consensus is achievable
  - **FLP => one cannot design a failure detector (for an asynchronous system) that guarantees both above properties**

Completeness or Accuracy?

- Most failure detector implementations are willing to **tolerate some inaccuracy**, but **require 100% completeness**
- Plenty of distributed apps designed assuming 100% completeness, e.g., p2p systems
  - "Err on the side of caution".
  - Other processes need to make repairs whenever a failure happens
- **Heart-beating** – satisfies completeness but not accuracy (why?)
- **Ping-Ack** – satisfies completeness but not accuracy (why?)

Completeness or Accuracy?

- Both Heart-beating and Ping-Ack provide
  - *Probabilistic* accuracy (for a process detected as failed, with some probability close to 1.0, it is true that it has actually crashed).
  - That was for asynchronous systems
- Heart-beating and Ping-ack can **satisfy <u>both</u> completeness and accuracy in *synchronous* systems** (why?)

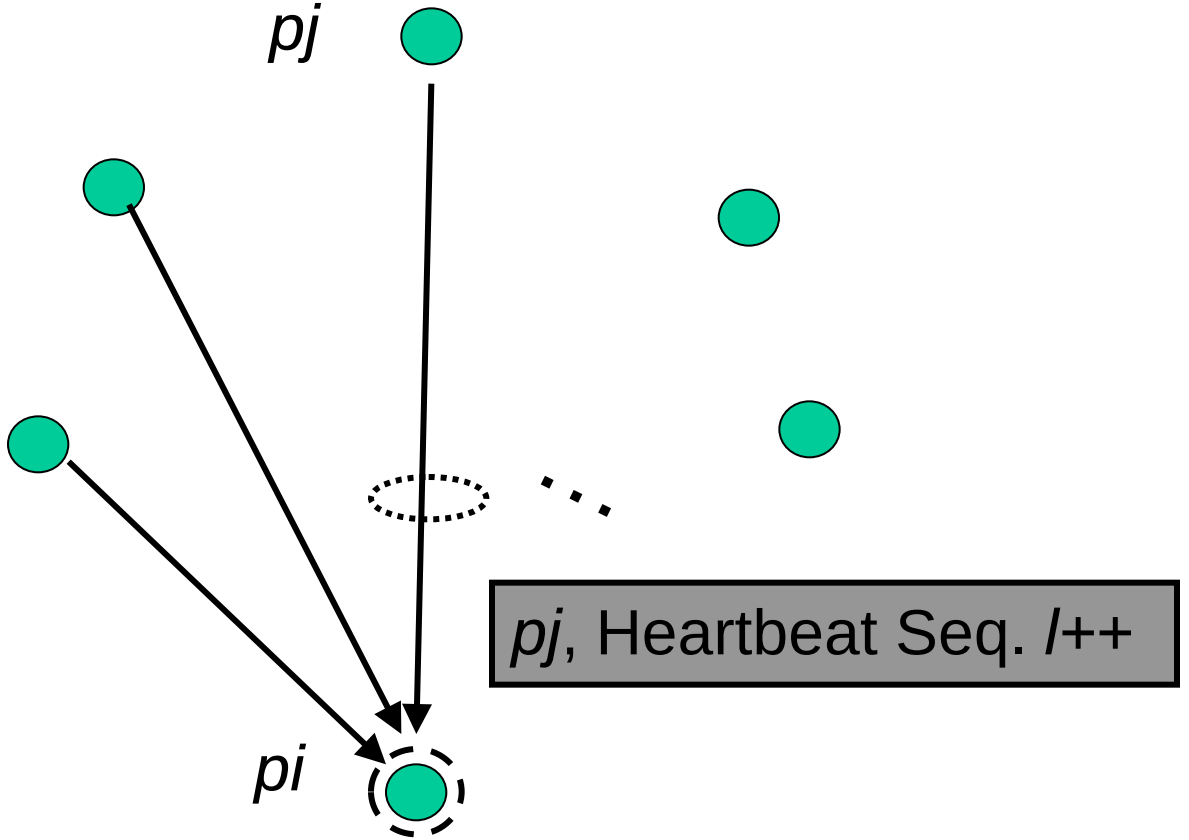
Failure Detection in a Distributed System

- Difference from original failure detection is
  - we want not one process (*pi*), but *all* processes in system to know about failure
- ➔ May need to combine failure detection with **a *dissemination protocol***
  - What's an example of a dissemination protocol?

Failure Detection in a Distributed System

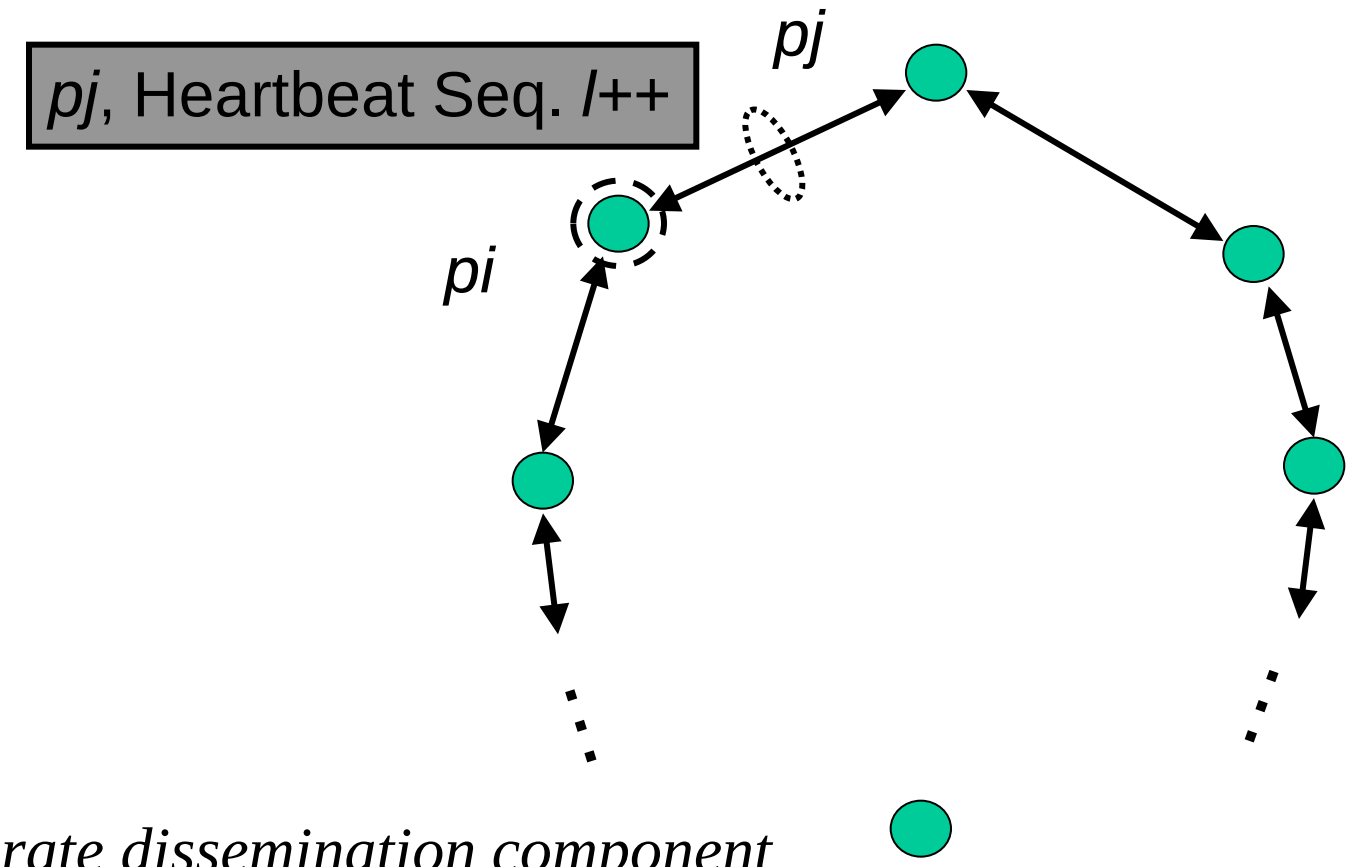
- Difference from original failure detection is
  - we want not one process (*pi*), but *all* processes in system to know about failure
- ➔ May need to combine failure detection with a ***dissemination protocol***
  - What's an example of a dissemination protocol?
    - **A reliable multicast protocol!**

Centralized Heart-beating



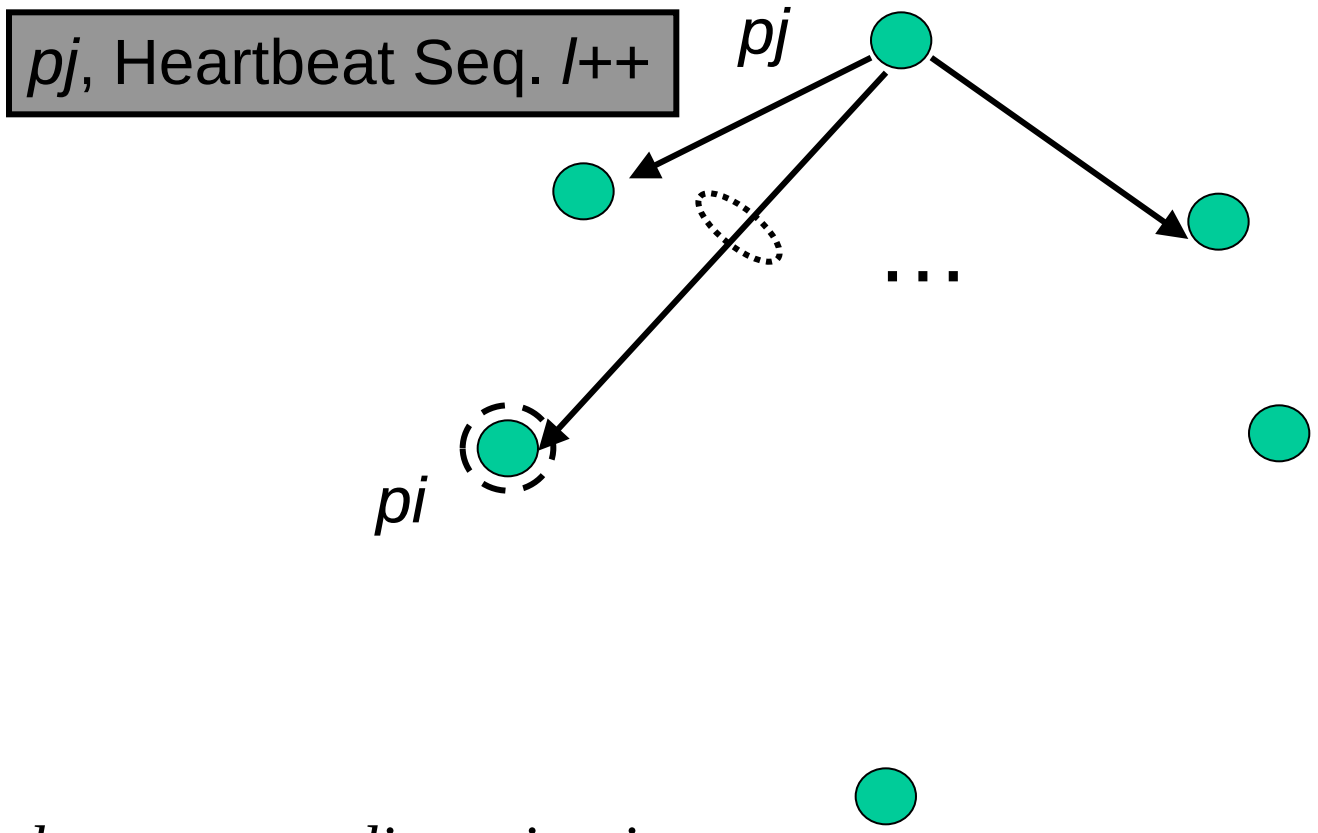
Needs a separate dissemination component
Downside?

Ring Heart-beating



Needs a separate dissemination component
Downside?

All-to-All Heart-beating



Does not *need a separate dissemination component*

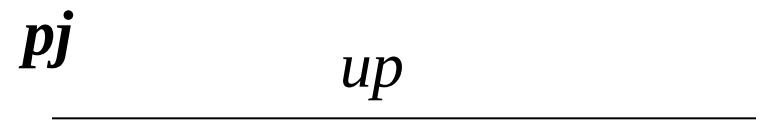
Downside?

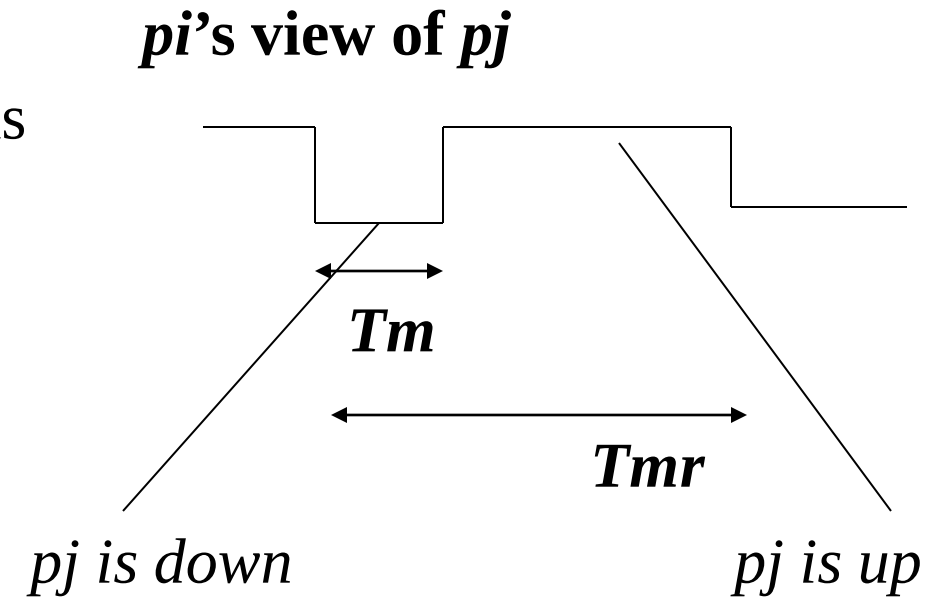
Efficiency of Failure Detector: Metrics

- **Measuring Speed**: Detection Time
  - Time between a process crash and its detection
  - Determines speed of failure detector
- **Measuring Accuracy**: depends on distributed application

Accuracy Metrics

- ***Tmr***: **Mistake recurrence time**
  - Time between two consecutive mistakes
- ***Tm*: Mistake duration time**
  - Length of time for which correct process is marked as failed (for crash-recovery model)

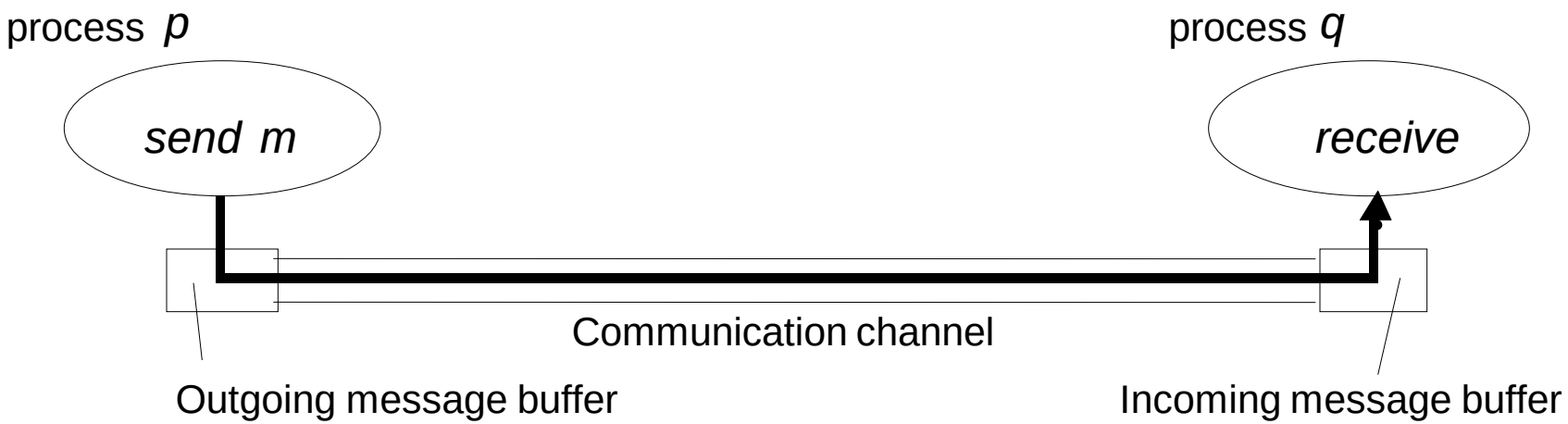




More Accuracy Metrics

- **Number of false failure detections** per time unit (**false positives**)
  - System reported failure, but actually the process was up
  - Failure detector is inaccurate
- **Number of not detected failures** (**false negatives**)
  - System did not report failure, but the process failed
  - Failure detector is incomplete

Processes and Channels



Other Failure Types

Communication Omission Failures

- **Send-omission**: loss of messages between the sending process and the outgoing message buffer (both inclusive)
  - What might cause this?
- **Channel omission**: loss of message in the communication channel.
  - What might cause this?
- **Receive-omission:** loss of messages between the incoming message buffer and the receiving process (both inclusive)
  - What might cause this?

Other Failure Types

❑ Arbitrary Failures (Byzantine)

- ❑ Arbitrary process failure: arbitrarily omits intended processing steps or takes unintended processing steps.
- ❑ Arbitrary channel failures: messages may be corrupted, duplicated, delivered out of order, incur extremely large delays; or non-existent messages may be delivered.

- ➢ Above two are Byzantine failures, e.g., due to hackers, man-in-the-middle attacks, viruses, worms, etc.
- ➢ A variety of Byzantine fault-tolerant protocols have been designed in literature!
  - ➢ "Scaling Byzantine Fault-tolerant replication in WAN", DSN 2006
  - ➢ A Byzantine Fault-Tolerant Mutual Exclusion Algorithm and its Application to Byzantine Fault-tolerant Storage Systems", (in ICDCS Workshop ADSN 2005)

Omission and Arbitrary Failures

Class of failure	*Affects*	*Description*
Fail-stop or Crash-stop	Process	Process halts and remains halted. Other processes may detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing Failures

- In synchronous distributed systems - applicable
  - Need **time limits** on process execution time, message delivery time, clock drift rate
- In asynchronous distributed systems - not applicable
  - Server may respond too slowly, but we cannot say if it is timing failure since no guarantee is offered
- In real-time OS - applicable
  - Need timing guarantees, hence may need redundant hardware
- In multimedia distributed systems – applicable
  - Timing important for multimedia computers with audio/video channels

Timing Failures

Class of Failure	Affects	Description
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time
Performance	Process	Process exceeds the bounds on the interval between two steps
Performance	Channel	A message's transmission takes longer than the stated bound

Masking Failures

- It is possible to construct **reliable services** from components that exhibit failures
  - Example: multiple servers that hold replicas of data can continue to provide a service when one of the crashes
- **Knowledge of failure characteristics** of a component can enable a new service when one of them crashes
- Service **masks** failure either by **hiding** it altogether or **converting** it into more acceptable type of failure
  - Example: checksums mask corrupted messages , effectively converting an arbitrary failure into omission failure
- **Reliability of one-to-one communication**
  - Defined in terms of validity and integrity
    - **Validity** – any message in the outgoing message buffer is eventually delivered to the incoming message buffer
    - **Integrity** – message received is identical to one sent, and no messages are delivered twice

Summary

- Failure detectors are required in distributed systems to maintain liveness in spite of process crashes
- Properties – completeness & accuracy, together unachievable in asynchronous systems
- Most apps require 100% completeness, but can tolerate inaccuracy
- 2 failure detector algorithms – Heart-beating and Ping-Ack
- Distributed Failure Distribution through heart-beating algorithms: Centralized, Ring, All-to-all
- Accuracy metrics
- Other Types of Failures

Next…

- Reading for Next Lecture: Peer-to-Peer Systems
  Two papers on website
  - Gnutella Protocol Specification v0.4
  - Chord: a scalable peer to peer lookup service

- Print and bring a personal copy of each paper to class.