CS 425 / ECE 428 Distributed Systems Fall 2026

Aishwarya Ganesan

w/ Ram Kesavan

Ack: Indranil Gupta (Indy)

Lecture 5-6: Failure Detection and Membership

Announcements

- MP1: Due Sunday 9/13, demos Monday 9/14
  - VMs distributed last week: see Piazza
  - GitHub course repo: see Piazza
    - IT does not handle requests on weekends
  - MP1 Demo, Signup, and Submission instructions release on Piazza
    - ✓ (Due by Fri Sep 11) [MP Demo Signup Sheet]
    - ✓ (Due by Sun Sep) [a. MP1 Submission Form & b. push MP code to GitHub]
    - ✓ (Due by Sun Sep 13) [MP1 Report Submission on Gradescope]
    - ✓ (On Mon Sep 14) Demos
- HW1: due 9/20
- Check Piazza often! It's where all the announcements are at!

Failures are the Norm

… not the exception, in datacenters.

Say, the rate of failure of one machine (OS/disk/motherboard/network, etc.) is once every 10 years (120 months) on average.

When you have 120 servers in the DC, the mean time to failure (MTTF) of the next machine is 1 month.

When you have 12,000 servers in the DC, the MTTF is about once every 7.2 hours!

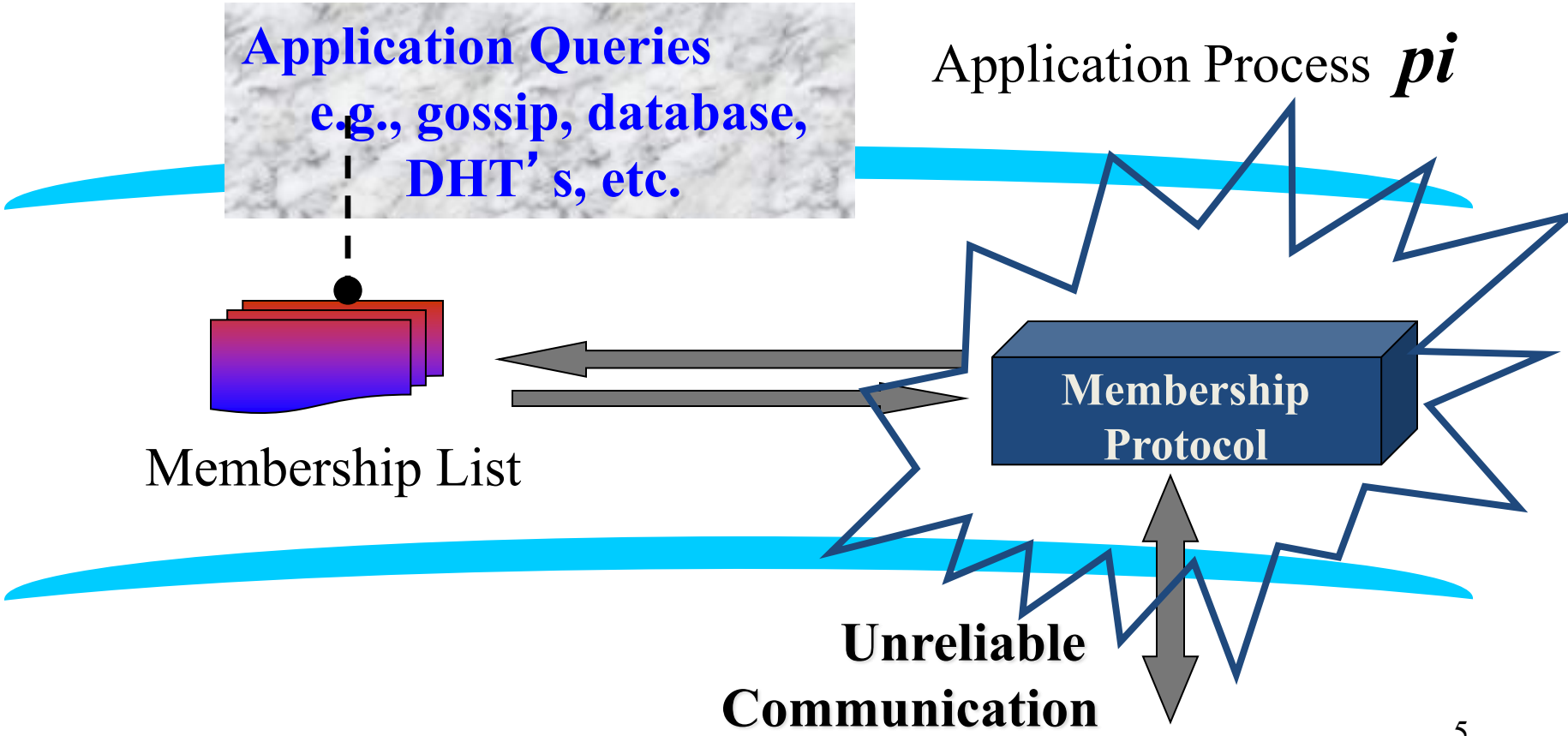
Soft crashes and failures are even more frequent!

Target Settings

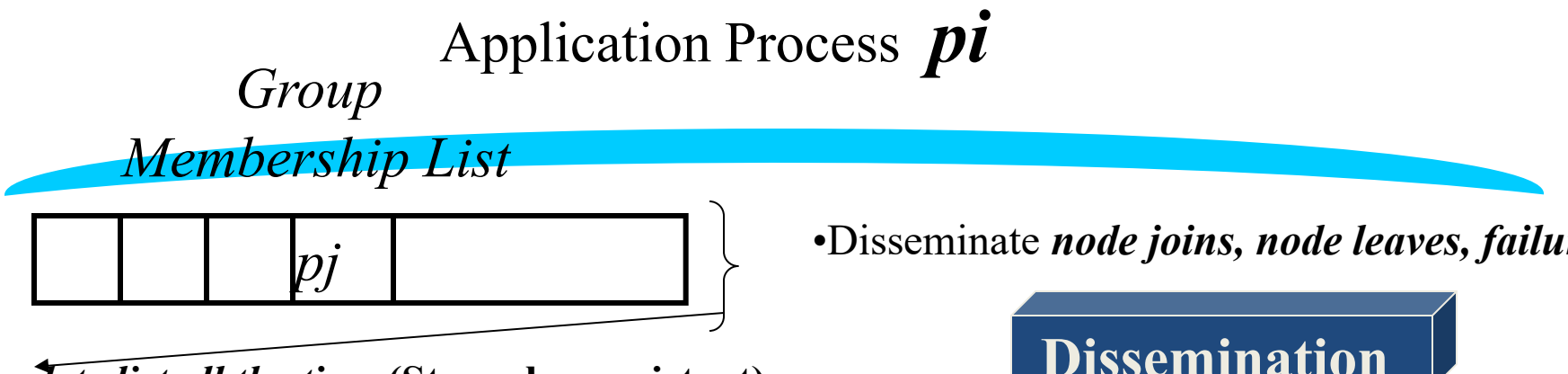
- Process ‘group’-based systems
  - Clouds/Datacenters
  - Replicated servers
  - Distributed databases

- Fail-stop (crash) process failures
  - (not fail-recover failures)

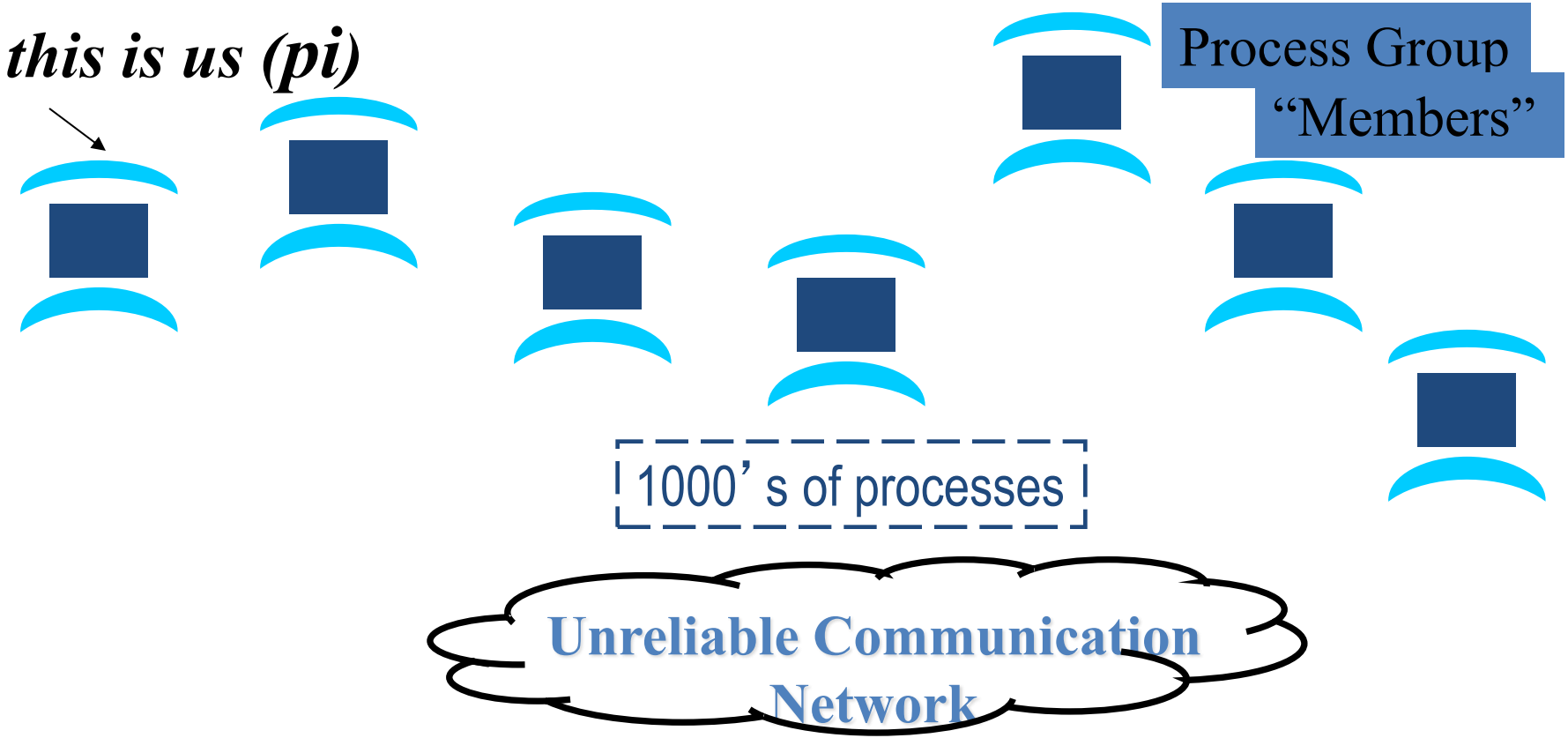
Group Membership Service



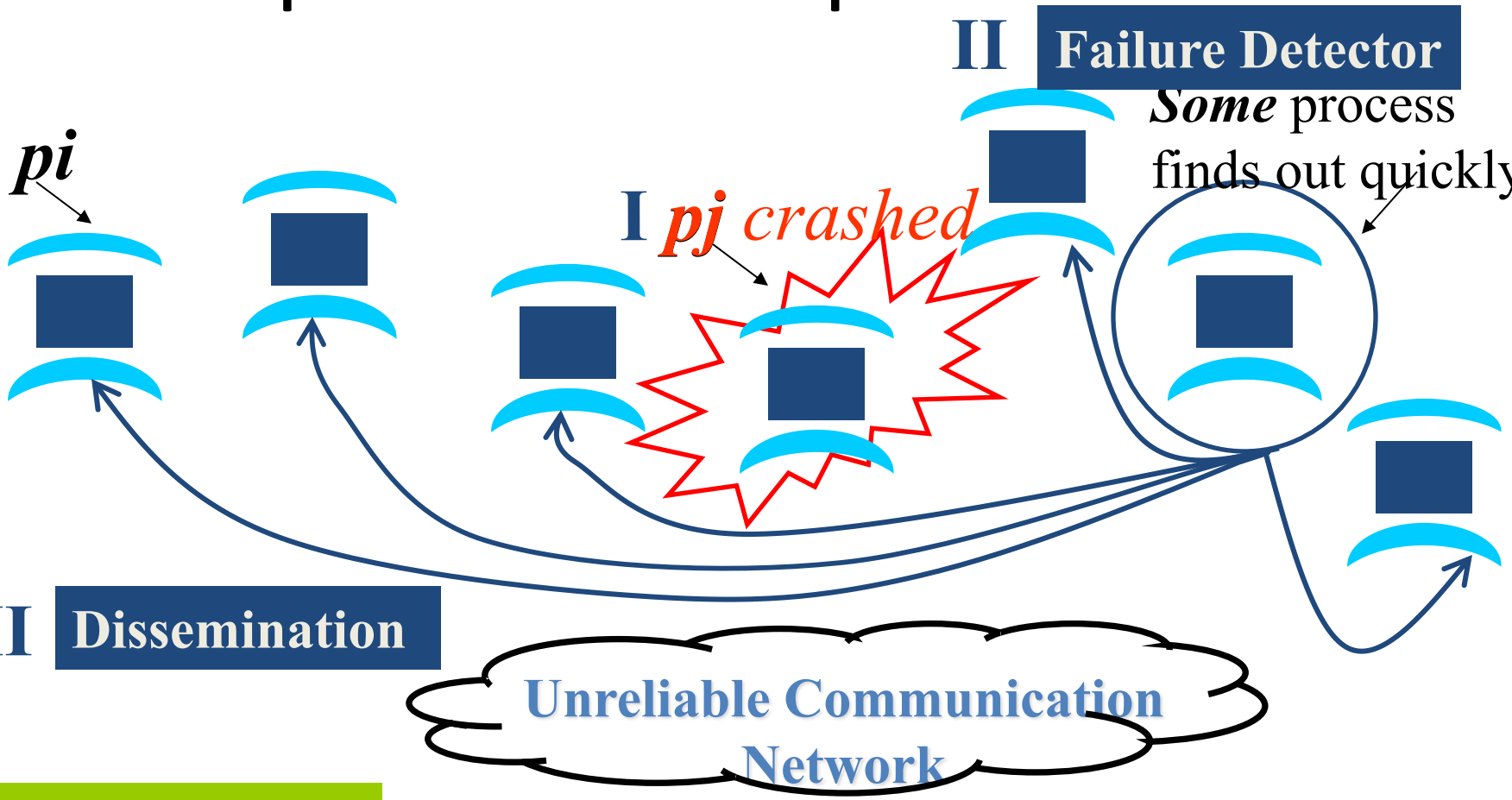
Two sub-protocols



Large Group: Scalability A Goal



Group Membership Protocol



Fail-stop Failures only

Next

- How do you design a group membership protocol?

I. *pj* crashes

- Nothing we can do about it!
- A frequent occurrence
- Common case rather than exception
- Frequency goes up linearly with size of datacenter

II. Distributed Failure Detectors: Desirable Properties

- Completeness = each failure is detected
- Accuracy = there is no mistaken detection
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Distributed Failure Detectors: Properties

- Completeness
- Accuracy

Impossible together in lossy networks [Chandra and Toueg]

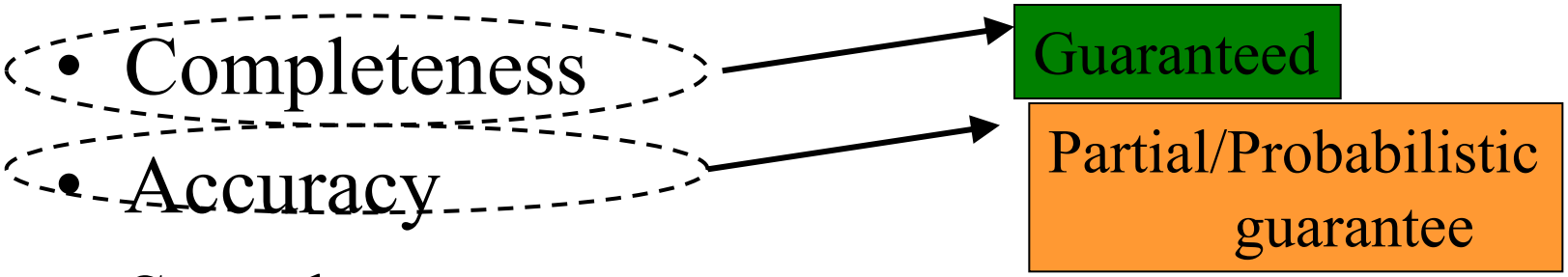
What is a 100% complete failure detector? *Hint*: trivial

What is a 100% accurate failure detector?

Which is more important? Completeness or Accuracy?

If possible, then can solve consensus! (but consensus is known to be unsolvable in asynchronous systems)

What Real Failure Detectors Prefer



- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

What Real Failure Detectors Prefer

- Completeness
- Accuracy

Guaranteed

Partial/Probabilistic guarantee

- Speed
  - Time to first detection of a failure

Time until ***some non-faulty*** process detects the failure

- Scale
  - Equal Load on each member
  - Network Message Load

What Real Failure Detectors Prefer

- Completeness
- Accuracy

Guaranteed

Partial/Probabilistic guarantee

- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Time until ***some non-faulty*** process detects the failure

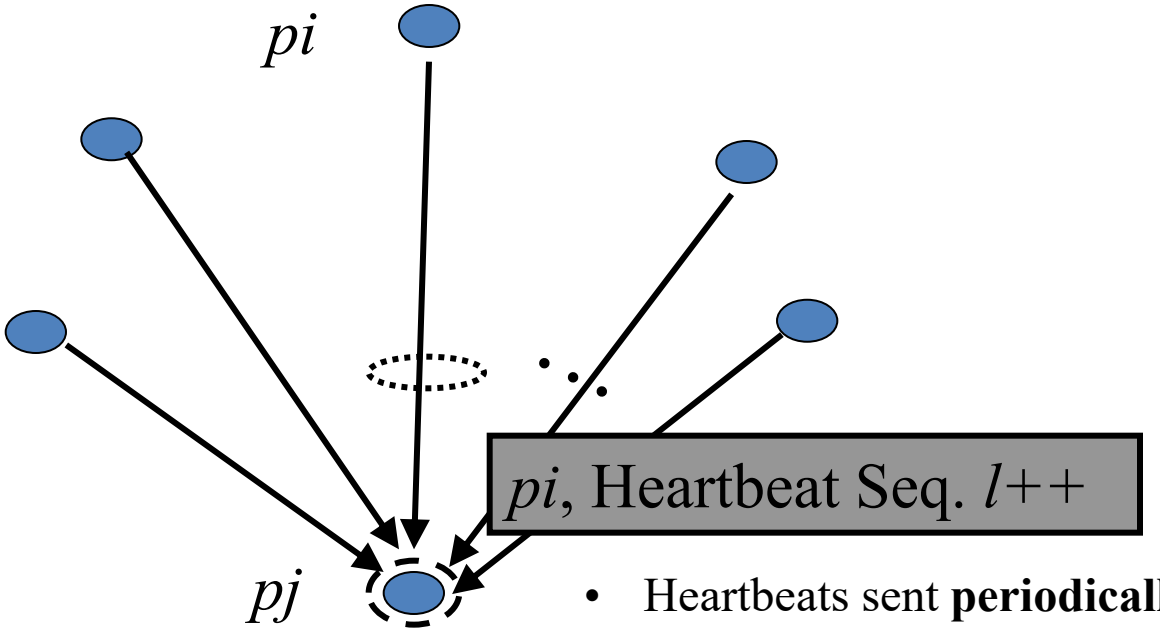
No bottlenecks/single failure point

Failure Detector Properties

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

In spite of arbitrary simultaneous process failures

Centralized Heartbeating



- Heartbeats sent **periodically**

•If heartbeat not received from *pi* within **timeout**, mark *pi* as failed

Completeness of centralized heartbeating:
what is the maximum number of process failures the failure detector can tolerate?

Responses hidden

Anonymous 151

0

1

N-1

N

In an asynchronous system, (Pj) uses centralized heartbeating and marks (Pi) as failed Which of the following could have occurred? Mark all that apply.

Responses hidden

Anonymous 153

Pi actually failed

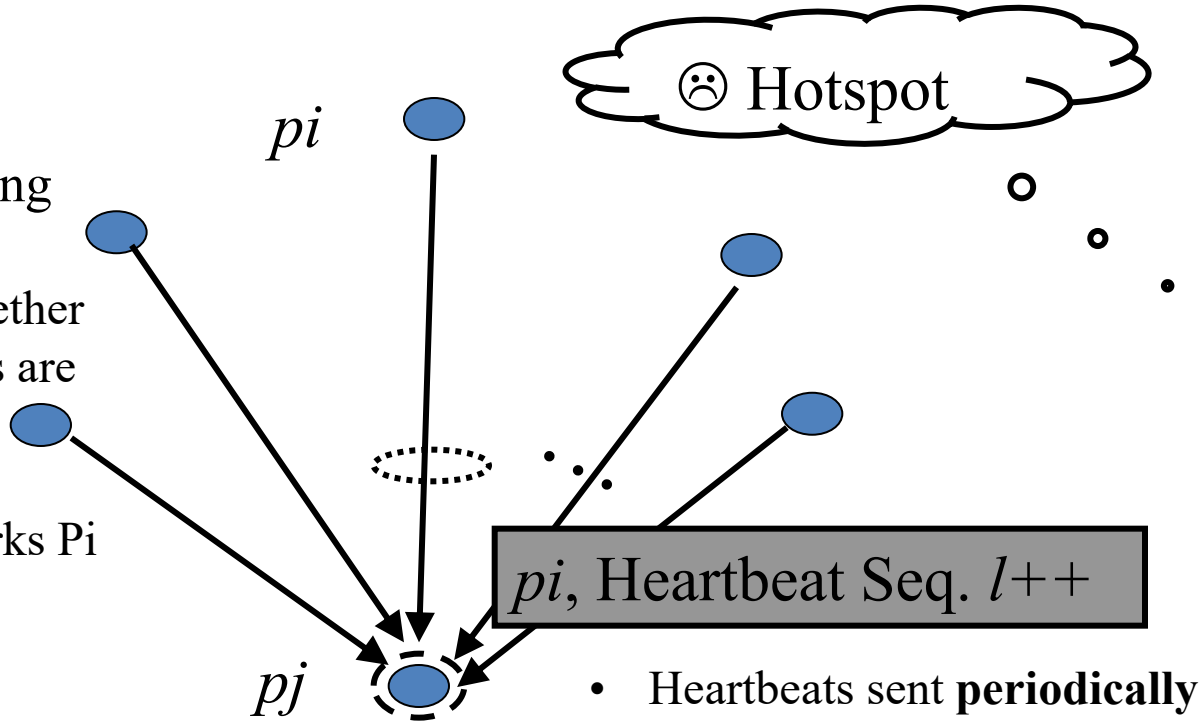
Pi did not fail, but its heartbeat message was delayed by the network.

Pi did not fail, but its heartbeat messages were dropped by the network

Pi did not fail, but it was running very slowly and had not sent the expected heartbeat yet.

Which failure-detector property is violated in cases B–D?

Centralized Heartbeating



Cannot tolerate Pj failing

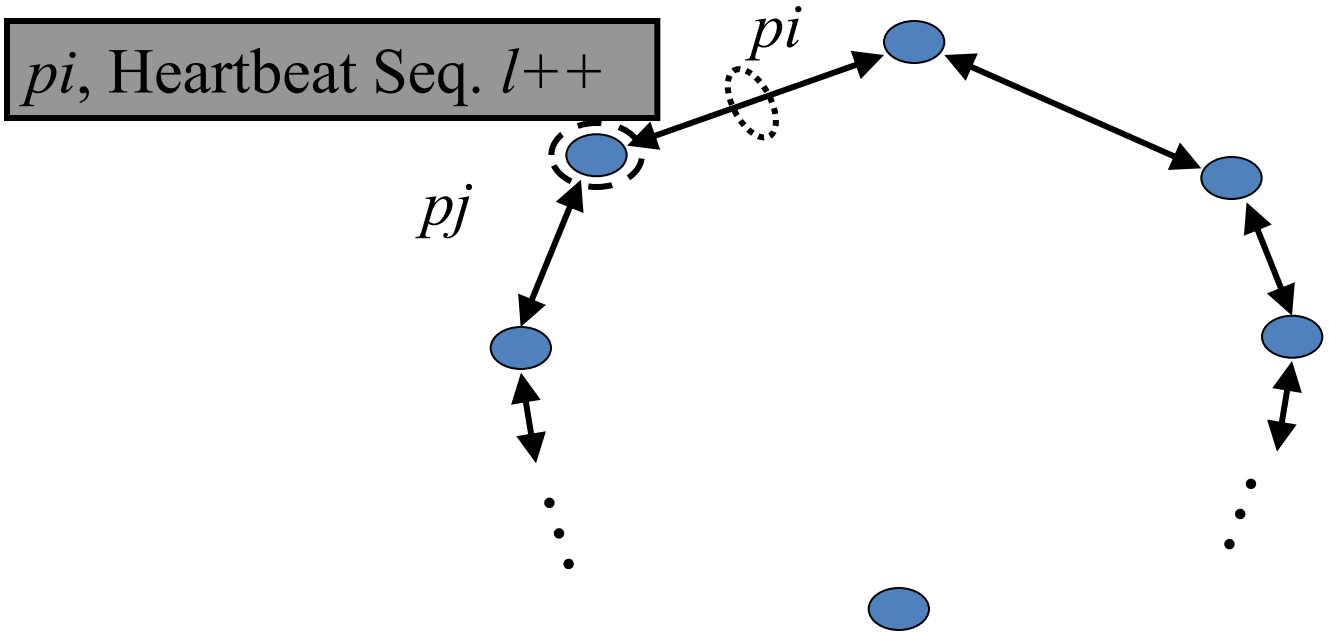
Pj cannot distinguish whether Pi failed or Pi's messages are dropped/delayed.

For completeness, Pj marks Pi as having failed

- Heartbeats sent **periodically**

•If heartbeat not received from *pi* within **timeout**, mark *pi* as failed

Ring Heartbeating



For ring heartbeating, assuming at most one process fails, which of the following best describes the failure detector?

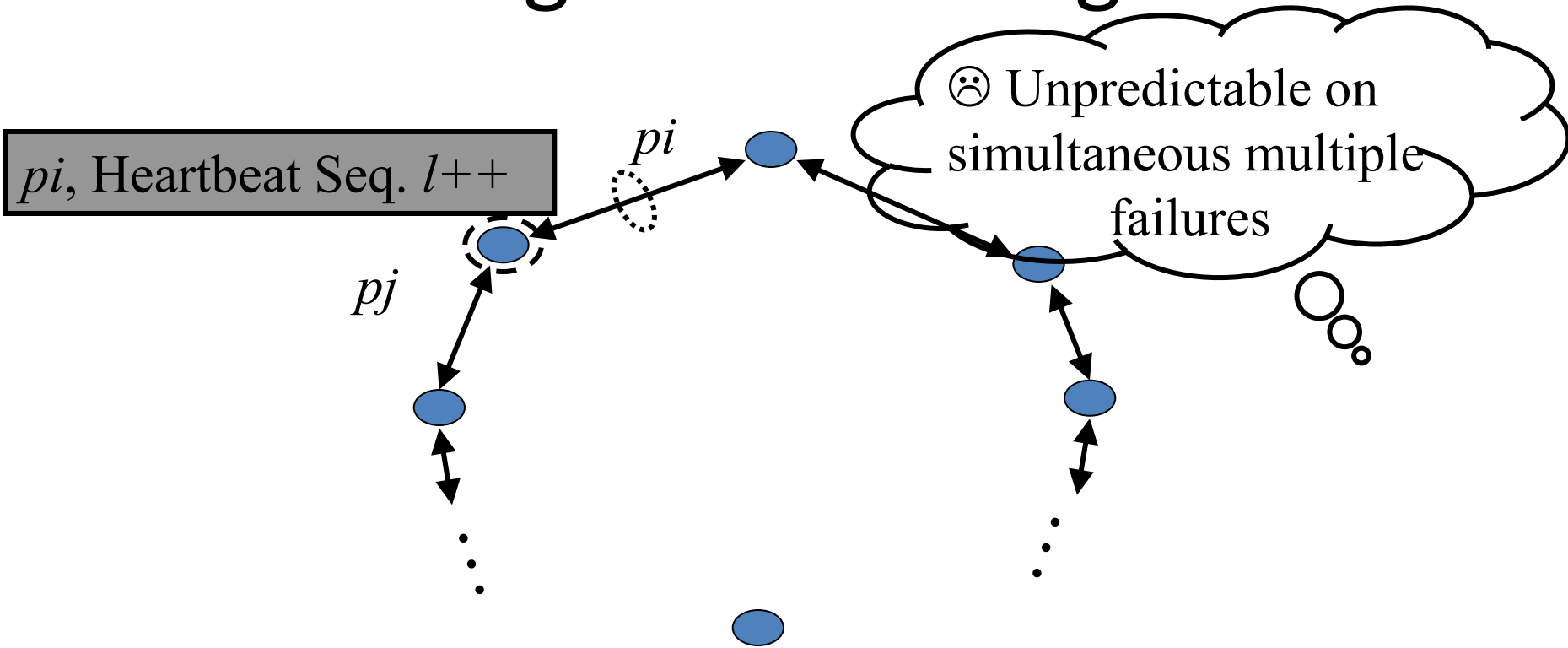
Accurate and complete

Not accurate, but complete

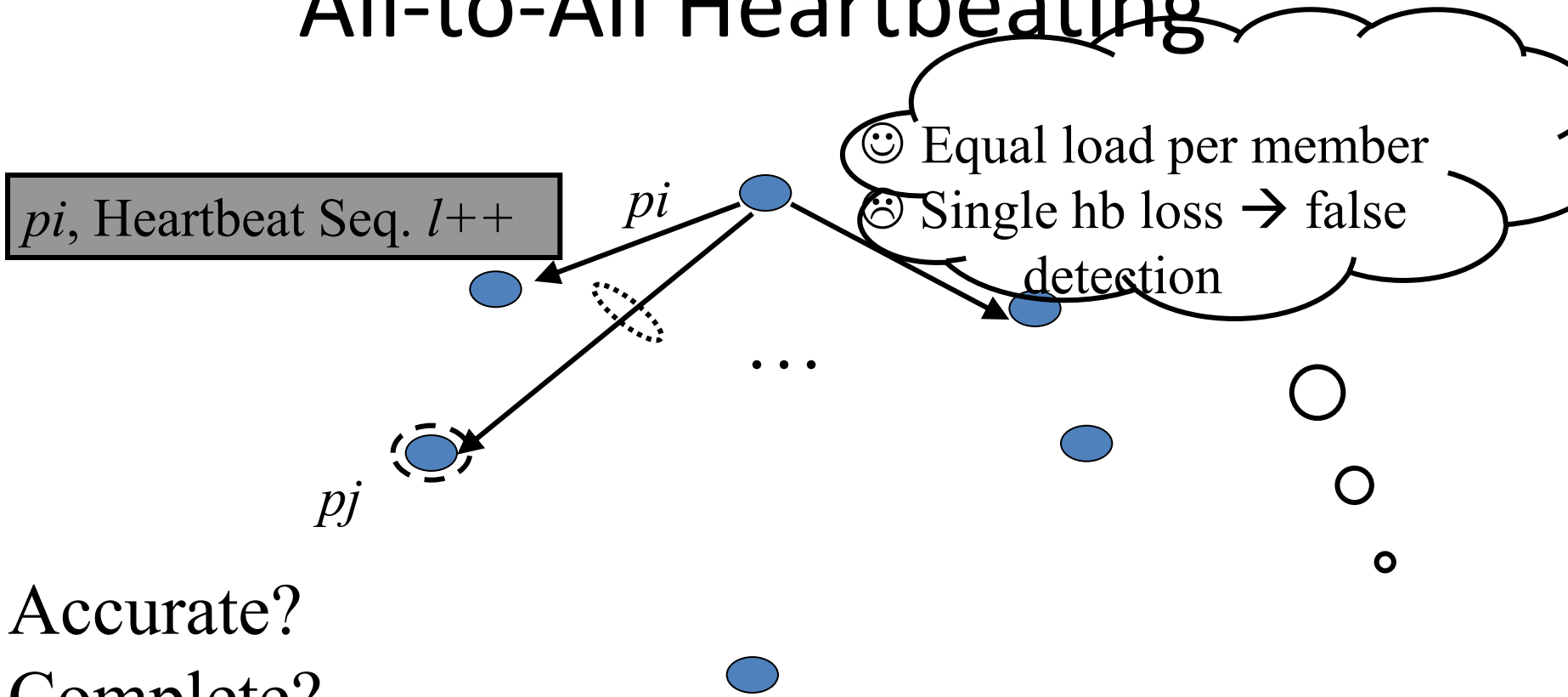
Accurate, but not complete

Neither accurate nor complete

Ring Heartbeating



All-to-All Heartbeating



Accurate?

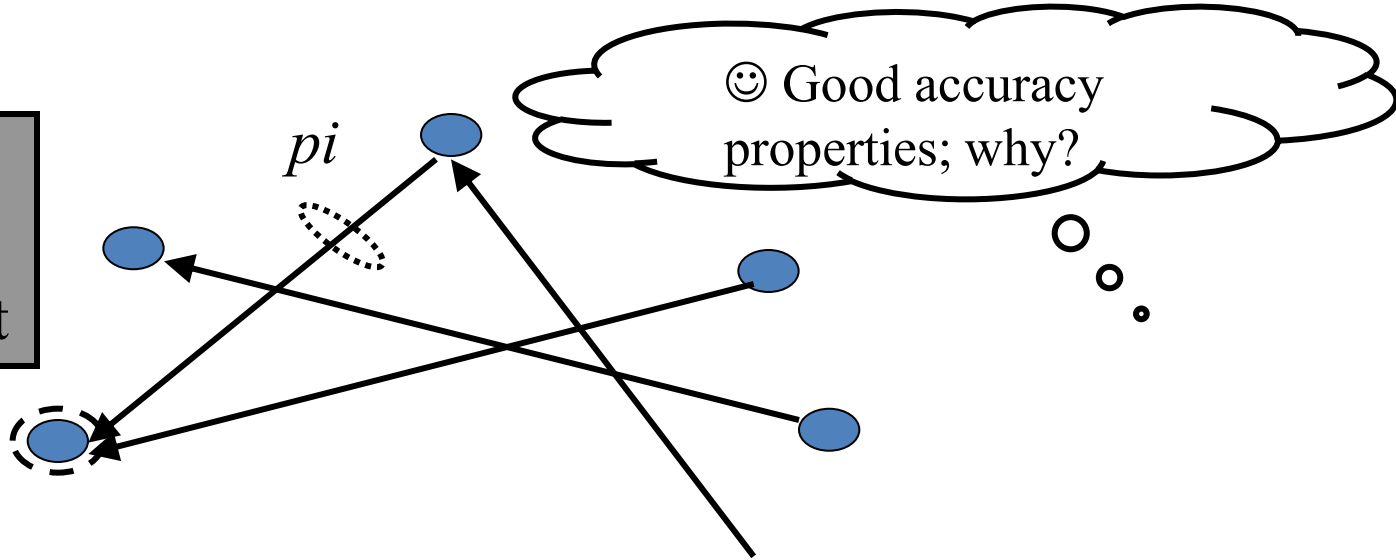
Complete?

Next

- How do we increase the robustness of all-to-all heartbeating?

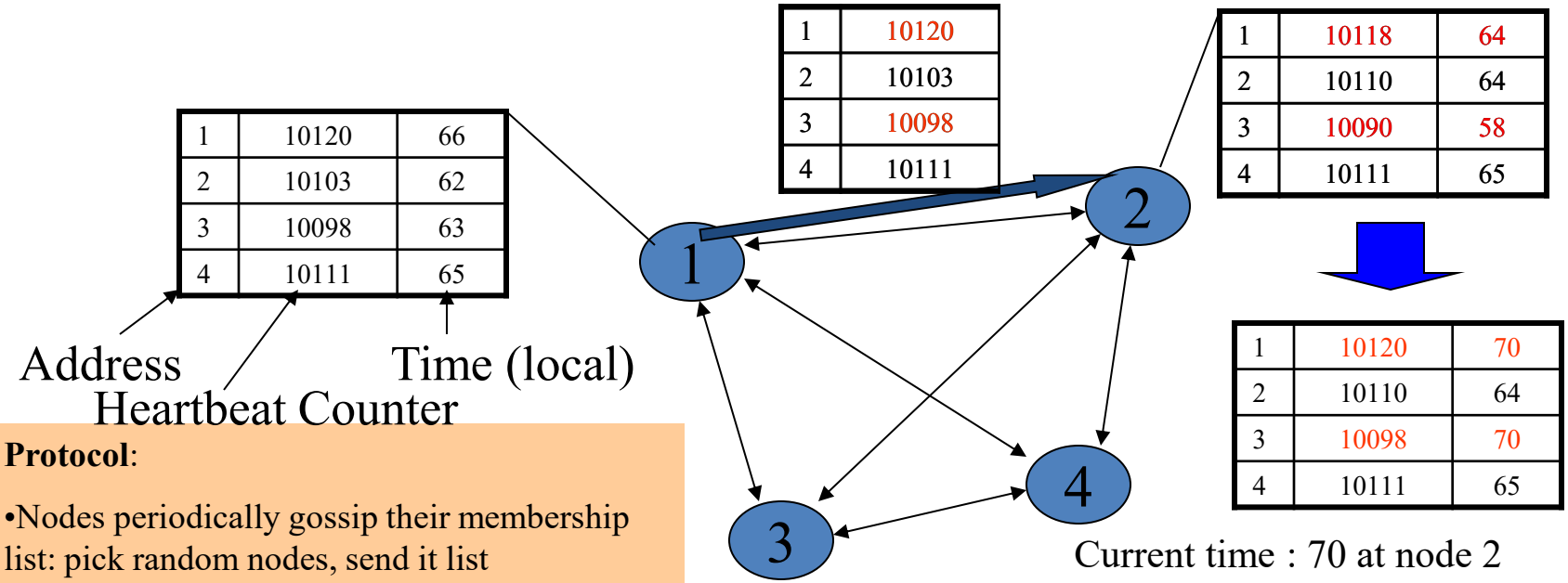
Gossip-style Heartbeating

Array of Heartbeat Seq. *l* for member subset



Gossip about your heartbeat and other processes' heartbeats

Gossip-Style Failure Detection



Protocol:

- Nodes periodically gossip their membership list: pick random nodes, send it list
- On receipt, it is *merged* with local membership list
- When an entry times out locally, member is marked as failed

Suppose current time is 80 at node 2 and timeout is 12 units, which nodes will 2 mark as fail?

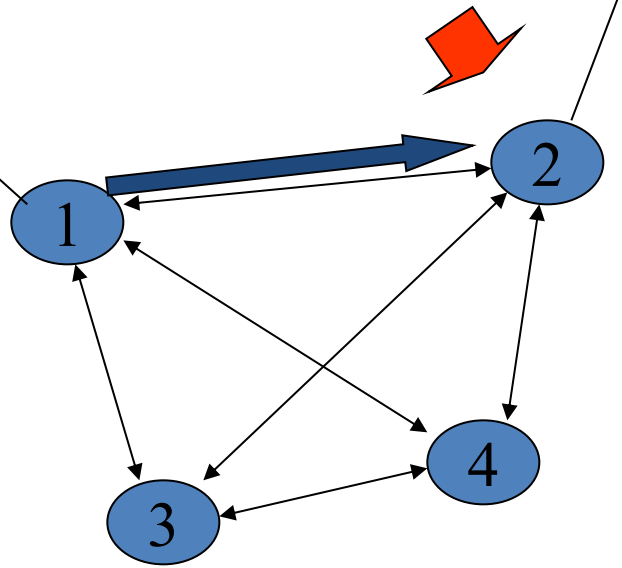
Gossip-Style Failure Detection

- If the heartbeat has not increased for more than T_{fail} seconds,
  the member is considered failed
- And after a further $T_{cleanup}$ seconds, it will delete the member from the list
- Why an additional timeout? Why not delete right away?

Gossip-Style Failure Detection

- What if an entry pointing to a failed node is deleted right after T_{fail} (=24) seconds?

1	10120	66
2	10103	62
3	10098	55
4	10111	65



1	10120	66
2	10110	64
3	10098	75
4	10111	65

Current time : 75 at node 2

How to set T_{fail} and $T_{cleanup}$?

Analysis/Discussion

- Well-known result: a gossip takes O(log(N)) time to propagate.
- So: Given sufficient bandwidth, a single heartbeat takes O(log(N)) time to propagate.
- So: N heartbeats take:
  - O(log(N)) time to propagate, if bandwidth allowed per node is allowed to be O(N)
  - O(N.log(N)) time to propagate, if bandwidth allowed per node is only O(1)
  - What about O(k) bandwidth?

Respond at
pe.app/aishwaryaganesa

Poll Everywhere

What happens if gossip period T_gossip is decreased?

Responses hidden

Anonymous 104

Bandwidth decreases; false-positive rate decreases

Bandwidth increases; false-positive rate decreases

Bandwidth increases; false-positive rate increases

Bandwidth decreases; false-positive rate increases

Edit

Choices | Results | Lock | Correctness

If the failure timeout T_fail is increased, while all other parameters are unchanged, what happens?

Responses hidden

119 1

Bandwidth increases; false-positive rate decreases

Bandwidth stays the same; false-positive rate decreases

Bandwidth decreases; false-positive rate increases

Bandwidth stays the same; false-positive rate increases

Analysis/Discussion

- Well-known result: a gossip takes O(log(N)) time to propagate.
- So: Given sufficient bandwidth, a single heartbeat takes O(log(N)) time to propagate.
- So: N heartbeats take:
  - O(log(N)) time to propagate, if bandwidth allowed per node is allowed to be O(N)
  - O(N.log(N)) time to propagate, if bandwidth allowed per node is only O(1)
  - What about O(k) bandwidth?
- What happens if gossip period T_{gossip} is decreased?
- What happens to $P_{mistake}$ (false positive rate) as T_{fail} and $T_{cleanup}$ is increased?
- Tradeoff: False positive rate vs. detection time vs. bandwidth

CS 425 / ECE 428 Distributed Systems Fall 2026

Aishwarya Ganesan

w/ Ram Kesavan

Ack: Indranil Gupta (Indy)

Lecture 5-6: Failure Detection and Membership

Failure Detector Properties …

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

So far (heartbeating protocols)…

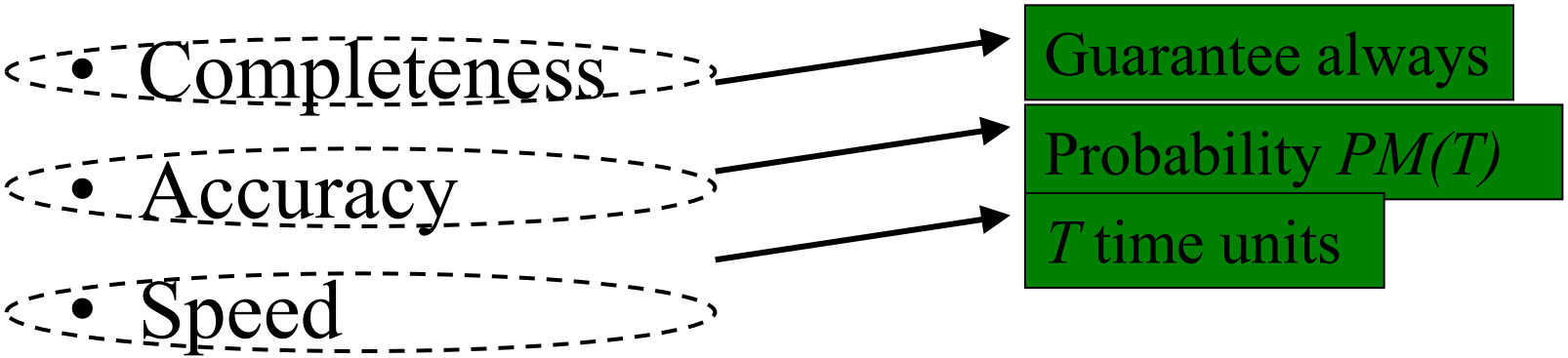
	Accurate?	Complete?
• Centralized	✗	✗
• Ring	✗	✗
• All-to-all	✗	✓
• Gossip	✓	✓

more accurate than all-to-all

Next

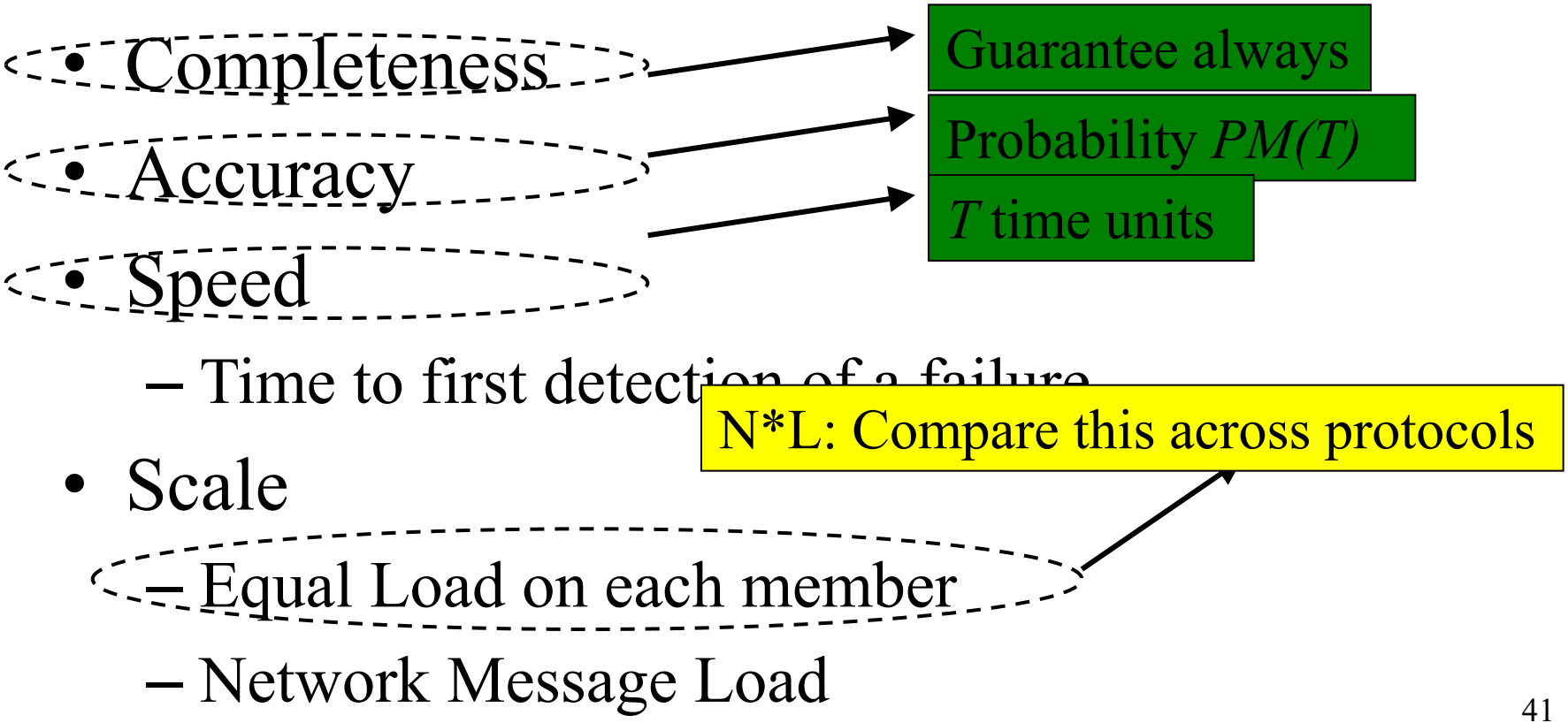
- So, is this the best we can do? What is the best we can do?

…Are application-defined Requirements

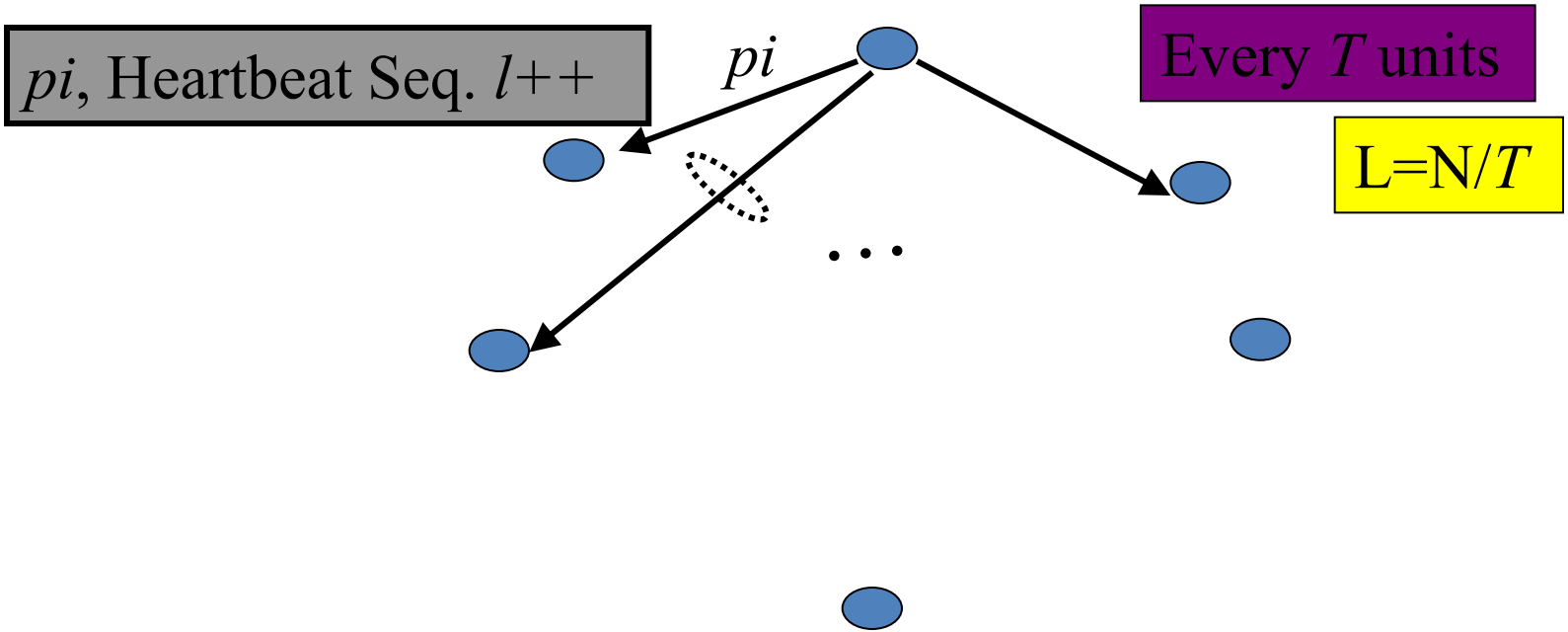


  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

…Are application-defined Requirements



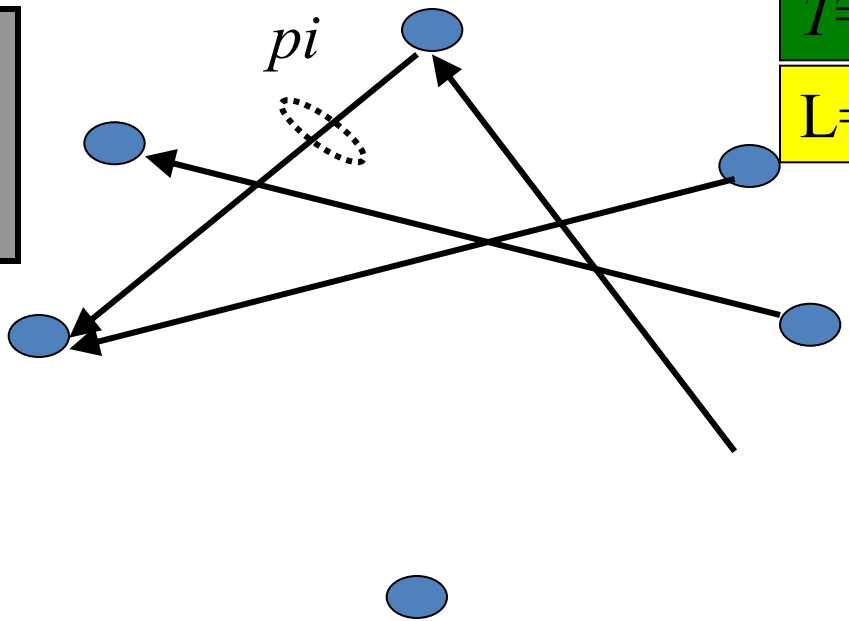
All-to-All Heartbeating



Gossip-style Heartbeating

Array of Heartbeat Seq. *l* for member subset

Every tg units =gossip period, send O(N) gossip message

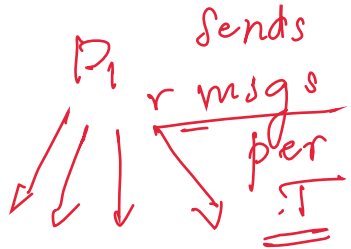


T=logN * tg

L=N/tg=N*logN/*T*

What's the Best/Optimal we can do?

- *Worst case* load L* per member in the group (messages per second)
  - as a function of *T*, *PM(T)*, N
  - Independent Message Loss probability p_{ml}
- $$L^* = \frac{\log(PM(T))}{\log(p_{ml})} \cdot \frac{1}{T}$$



P(all msgs dropped) = $(p_{ml})^r$

To meet app req, $(p_{ml})^r \le PM(T)$

$\log(p_{ml})^r \le \log PM(T)$

$r \log(p_{ml}) \le \log PM(T)$

$r \ge \frac{\log(PM(T))}{\log(p_{ml})}$

r msgs per T period

Heartbeating

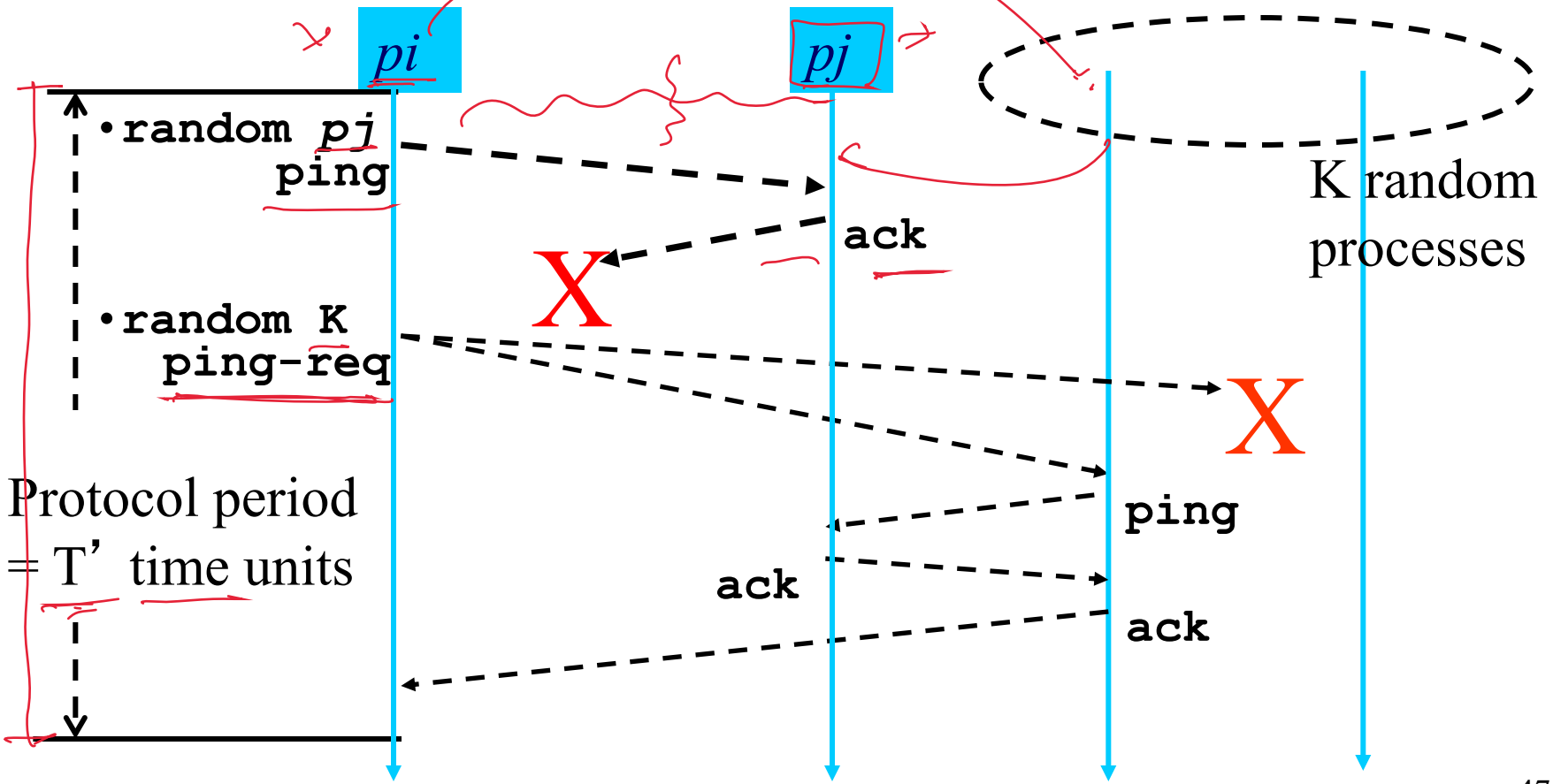
- Optimal L is independent of N (!)
- All-to-all and gossip-based: sub-optimal
  - L=O(N/T)
  - try to achieve simultaneous detection at ***all*** processes
  - fail to distinguish *Failure Detection* and *Dissemination* components

- Can we reach this bound?
- Key:
  - Separate the two components
  - Use a non heartbeat-based Failure Detection Component

Next

- Is there a better failure detector?

SWIM Failure Detector Protocol



Detection Time

Suppose P_j crashed.

$P(\text{some node picked } P_i \text{ picks } P_j) = \frac{1}{N}$

$P(P_i \text{ doesn't pick } P_j) = 1 - \frac{1}{N}$

$P(\text{none of the nodes pick } P_j) = \left(1 - \frac{1}{N}\right)^{N-1}$

- Prob. of being pinged in T' = $1 - (1 - \frac{1}{N})^{N-1} = 1 - e^{-1}$

i.e., $P(\text{at least one node picks } P_j)$

$\to 1/e$ for N large

- $E[T] = T' \cdot \frac{e}{e-1}$

$E[\#\text{ periods until detection}] = \frac{1}{1-e^{-1}} = \frac{e}{e-1}$

expected detection time

- Completeness: *Any* alive member detects failure
  - Eventually
  - By using a trick: within worst case *O(N)* protocol periods

each period is T' time units

Accuracy, Load

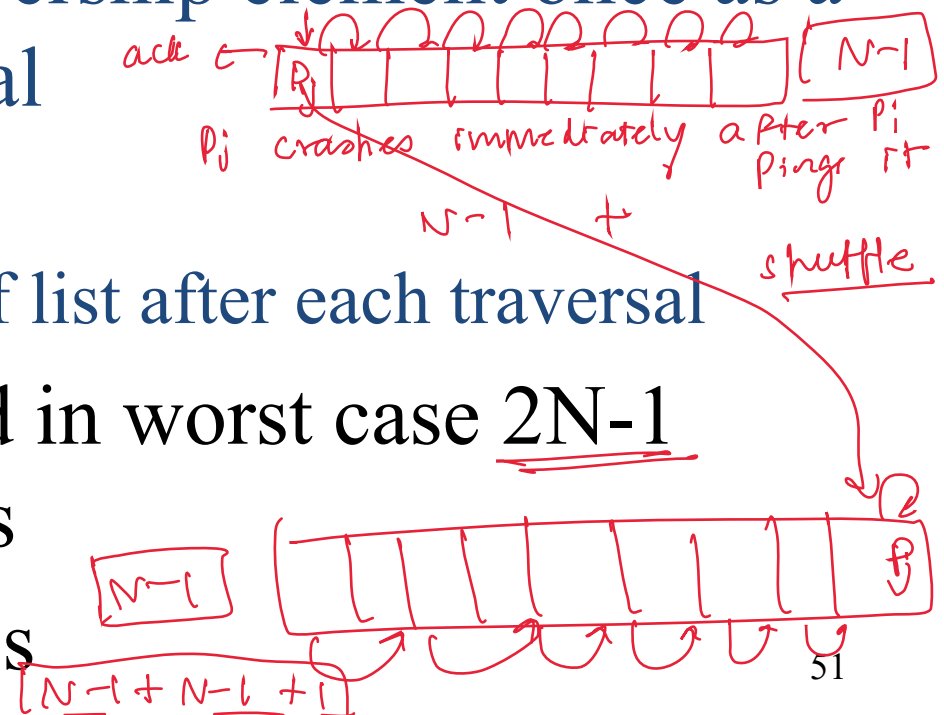
- *PM(T)* is exponential in -*K*. Also depends on *pml* (and *pf*)
  - See paper
- $\frac{L}{L^*} < 28$ $\frac{E[L]}{L^*} < 8$ for up to 15 % loss rates

SWIM Failure Detector

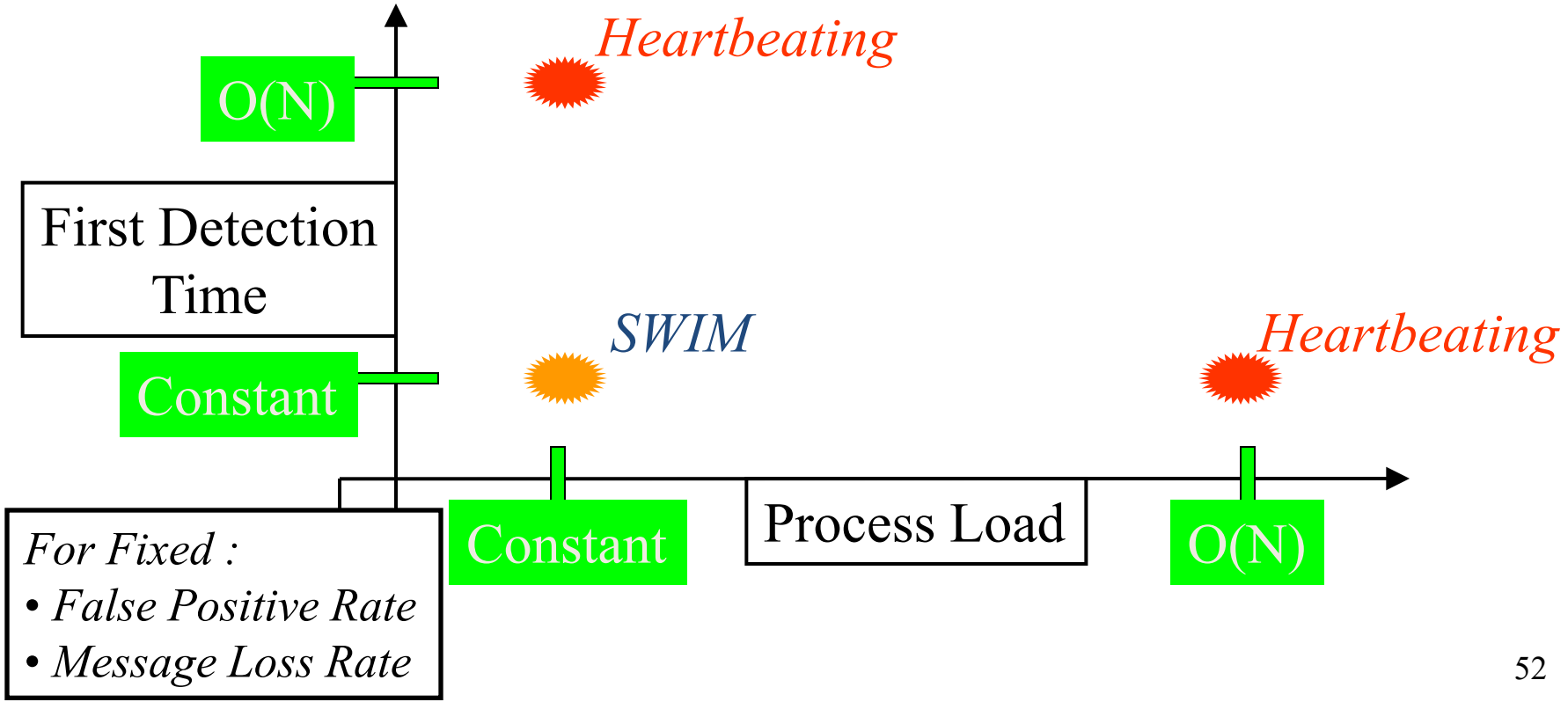
Parameter	SWIM
First Detection Time	• Expected $\left\lceil \frac{e}{e-1} \right\rceil$ periods • Constant (independent of group size)
Process Load	• Constant per period • < 8 L* for 15% loss
False Positive Rate	• Tunable (via K) • Falls exponentially as load is scaled
Completeness	• Deterministic time-bounded • Within O(log(N)) periods w.h.p.

Time-bounded Completeness

- Key: select each membership element once as a ping target in a traversal
  - Round-robin pinging
  - Random permutation of list after each traversal
- Each failure is detected in worst case 2N-1 (local) protocol periods
- Preserves FD properties



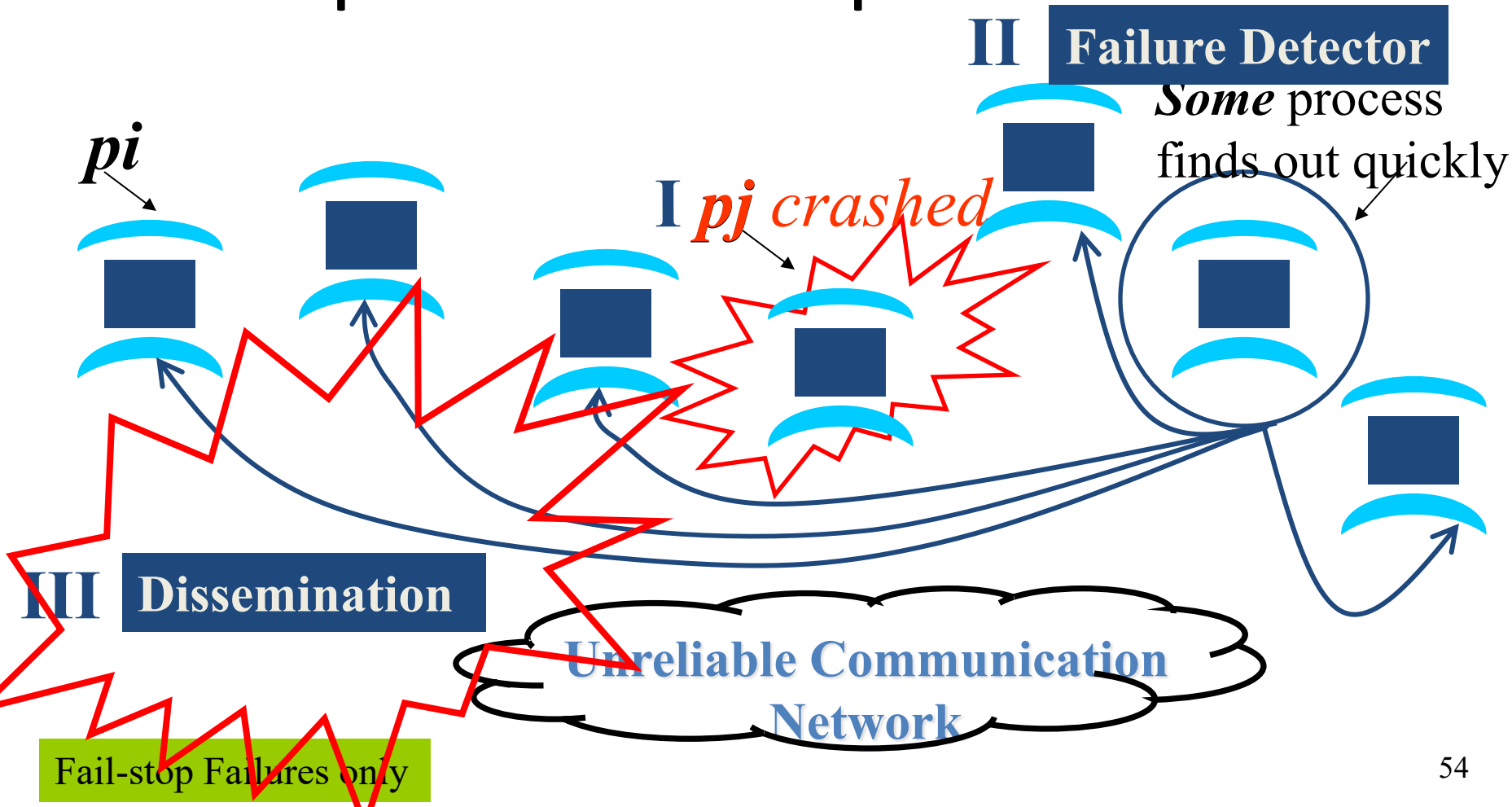
SWIM versus Heartbeating



Next

- How do failure detectors fit into the big picture of a group membership protocol?
- What are the missing blocks?

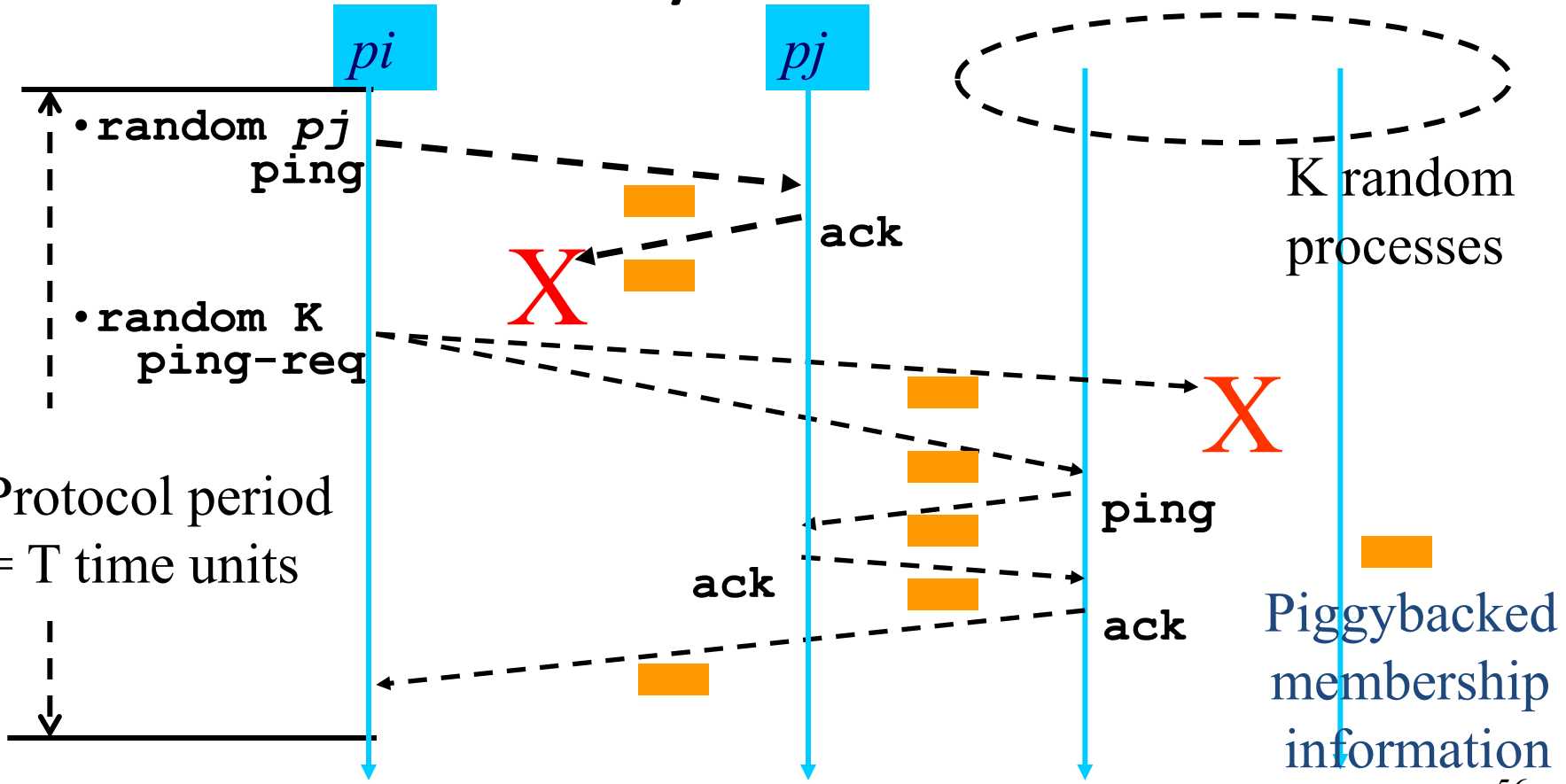
Group Membership Protocol



Dissemination Options

- Multicast (Hardware / IP)
  - unreliable
  - multiple simultaneous multicasts
- Point-to-point (TCP / UDP)
  - expensive
- Zero extra messages: Piggyback on Failure Detector messages
  - Infection-style Dissemination

Infection-style Dissemination



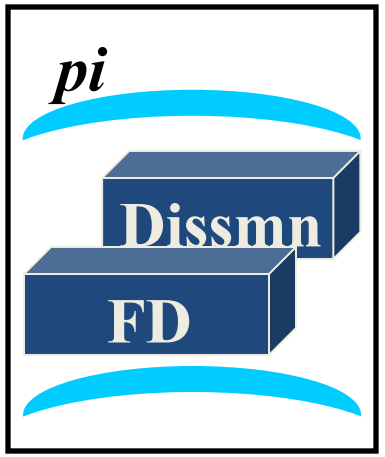
Infection-style Dissemination

- Epidemic/Gossip style dissemination
  - After $\lambda.\log(N)$ protocol periods, $N^{-(2\lambda-2)}$ processes would not have heard about an update
- Maintain a buffer of recently joined/evicted processes
  - Piggyback from this buffer
  - Prefer recent updates
- Buffer elements are garbage collected after a while
  - After $\lambda.\log(N)$ protocol periods, i.e., once they've propagated through the system; this defines weak consistency

Suspicion Mechanism

- False detections, due to
  - Perturbed processes
  - Packet losses, e.g., from congestion
- Indirect pinging may not solve the problem
- Key: *suspect* a process before *declaring* it as failed in the group

Suspicion Mechanism



Dissmn (Suspect *pj*)

Suspected

FD:: *pi* ping failed
Dissmn::(Suspect *pj*)

FD::*pi* ping success
Dissmn::(Alive *pj*)

Time out

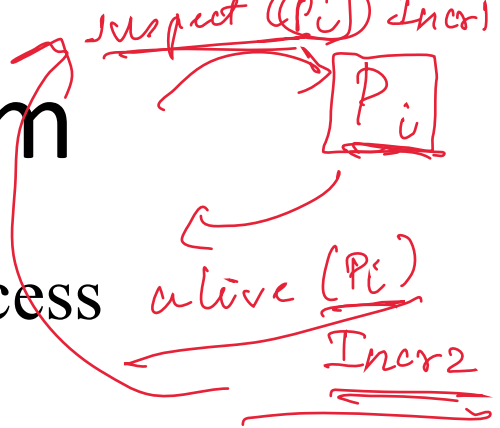
Alive

Failed

Dissmn (Alive *pj*)

Dissmn (Failed *pj*)

Suspicion Mechanism



- Distinguish multiple suspicions of a process
  - Per-process *incarnation number*
  - *Inc #* for *pi* can be incremented only by *pi*
    - e.g., when it receives a (Suspect, *pi*) message
  - Somewhat similar to DSDV (routing protocol in ad-hoc nets)
- Higher inc# notifications over-ride lower inc#'s
- Within an inc#: (Suspect inc #) > (Alive, inc #)
- (Failed, inc #) overrides everything else

SWIM In Industry

- First used in Oasis/CoralCDN
- Implemented open-source by Hashicorp Inc.
  - Called “Serf”
  - Later “Consul”
- Today: Uber implemented it, uses it for failure detection in their infrastructure
  - See “ringpop” system

Wrap Up

- Failures the norm, not the exception in datacenters
- Every distributed system uses a failure detector
- Many distributed systems use a membership service

- Ring failure detection underlies
  - IBM SP2 and many other similar clusters/machines

- Gossip-style failure detection underlies
  - Amazon EC2/S3 (rumored!)

Announcements

- HW1: due 9/20
- Please view Grid Computing Lecture Video from website!
  - (We won't lecture in class)