CS 425 / ECE 428
Distributed Systems
Fall 2017

Indranil Gupta (Indy)

Sep 14, 2017

Lecture 6: Failure Detection and Membership, Grids

A Challenge

- You’ve been put in charge of a datacenter, and your manager has told you, “Oh no! We don’t have any failures in our datacenter!”

- Do you believe him/her?

- What would be your first responsibility?
- Build a failure detector
- What are some things that could go wrong if you didn’t do this?

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!

To build a failure detector

- You have a few options

    1. Hire 1000 people, each to monitor one machine in the datacenter and report to you when it fails.
    2. Write a failure detector program (distributed) that automatically detects failures and reports to your workstation.

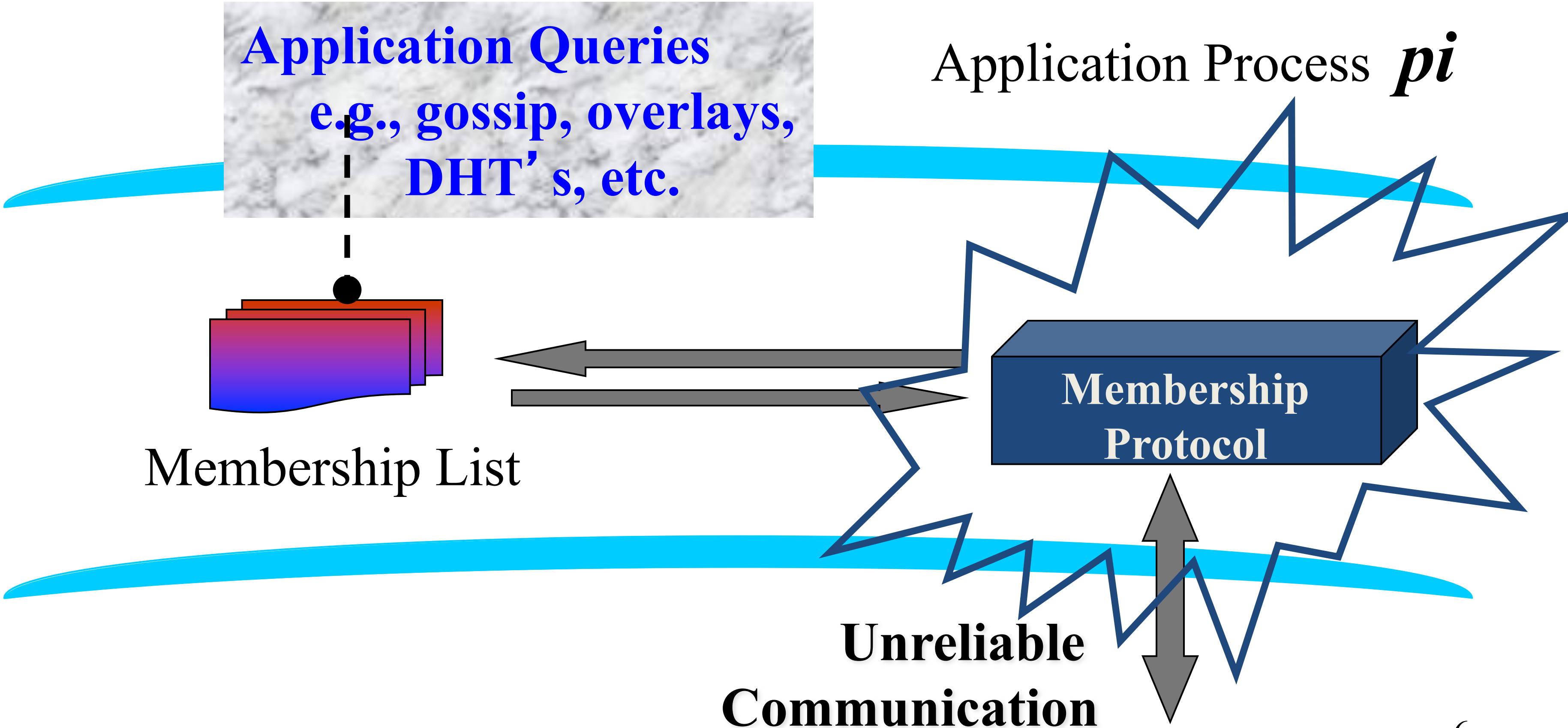
    Which is more preferable, and why?

Target Settings

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

- Fail-stop (crash) process failures

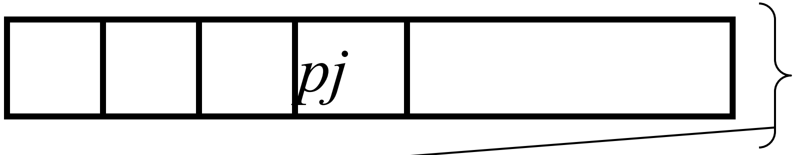
Group Membership Service



Two sub-protocols

Application Process ***pi***

Group Membership List

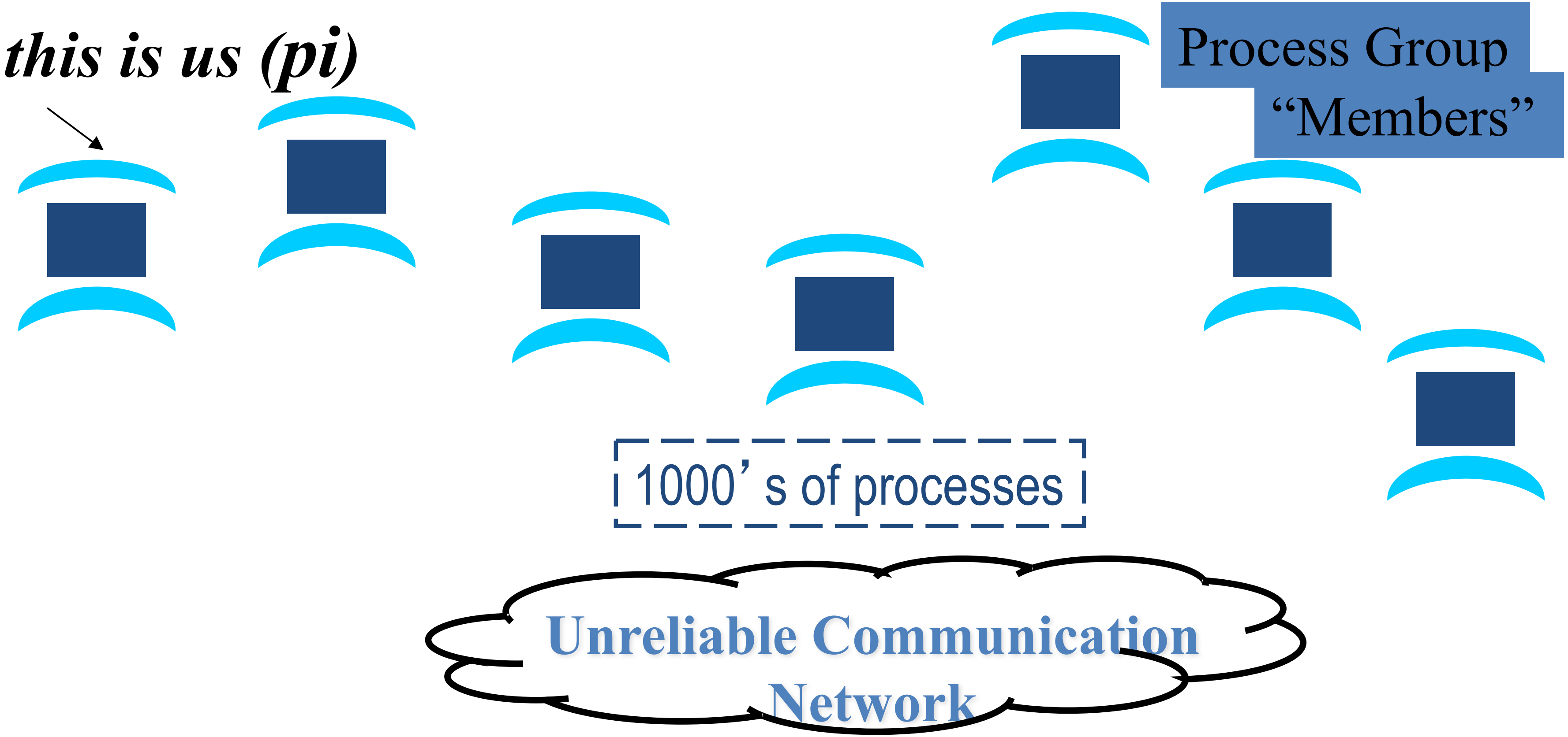


- ***Complete list all the time*** **(Strongly consistent)**
    - Virtual synchrony
- ***Almost-Complete*** **list (Weakly consistent)**
    - Gossip-style, SWIM, …
- **Or *Partial-random* list (other systems)**
    - SCAMP, T-MAN, Cyclon,…

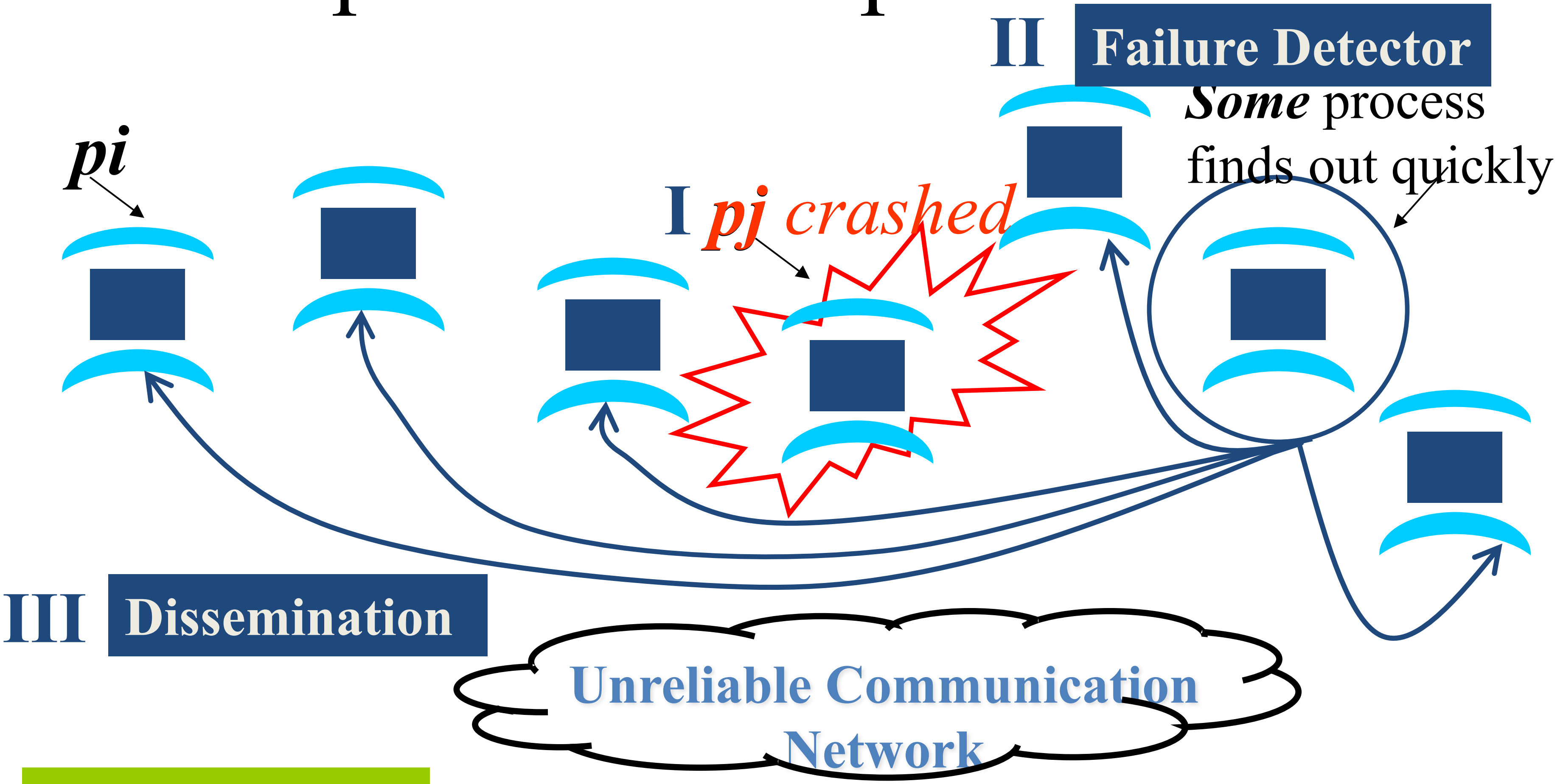
Unreliable Communication

Focus of this series of lecture

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

Impossible together in lossy networks [Chandra and Toueg]

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

What Real Failure Detectors Prefer

- Completeness → Guaranteed
- Accuracy → Partial/Probabilistic guarantee
- Speed
  - – Time to first detection of a failure
- Scale
  - – Equal Load on each member
  - – Network Message Load

What Real Failure Detectors Prefer

- Completeness → Guaranteed
- Accuracy → Partial/Probabilistic guarantee
- Speed
  - Time to first detection of a failure → Time until ***some*** process detects the failure
- Scale
  - Equal Load on each member
  - Network Message Load

What Real Failure Detectors Prefer

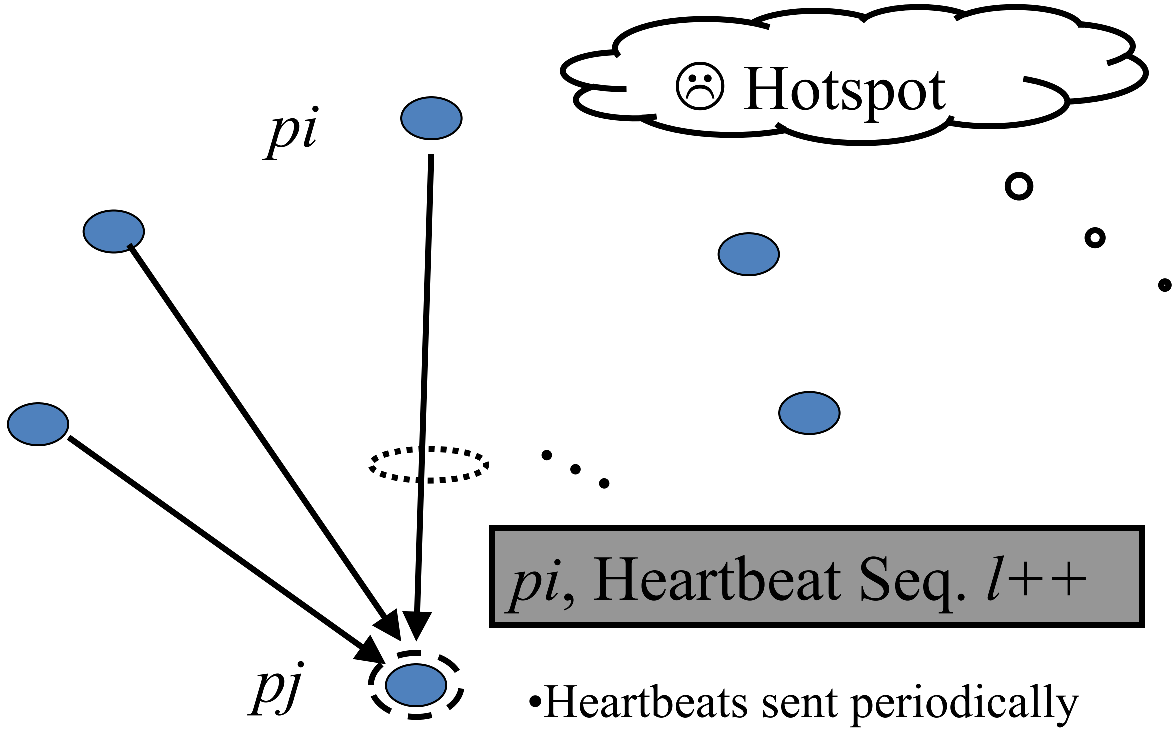
- Completeness → Guaranteed
- Accuracy → Partial/Probabilistic guarantee
- Speed
  - Time to first detection of a failure → Time until ***some*** process detects the failure
- Scale → No bottlenecks/single failure point
  - Equal Load on each member
  - Network Message Load

Failure Detector Properties

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

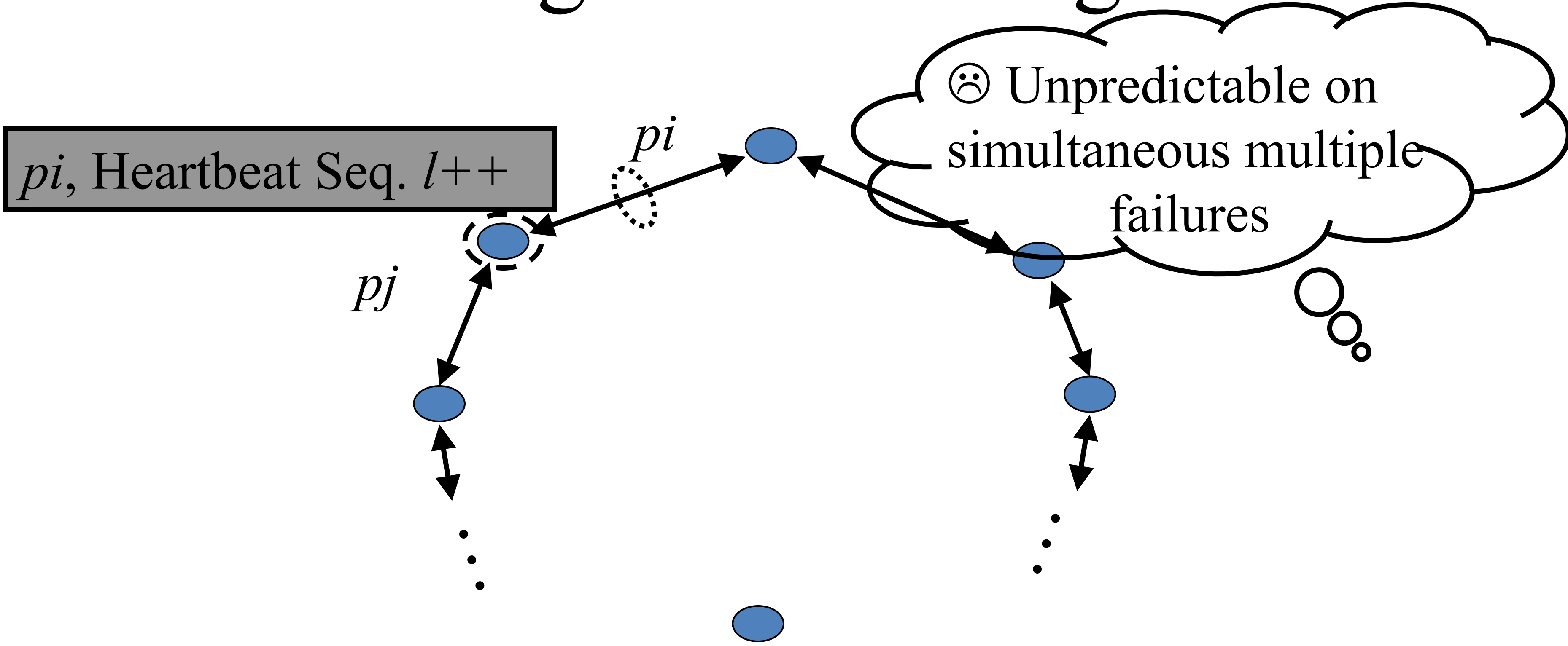
In spite of arbitrary simultaneous process failures

Centralized Heartbeating

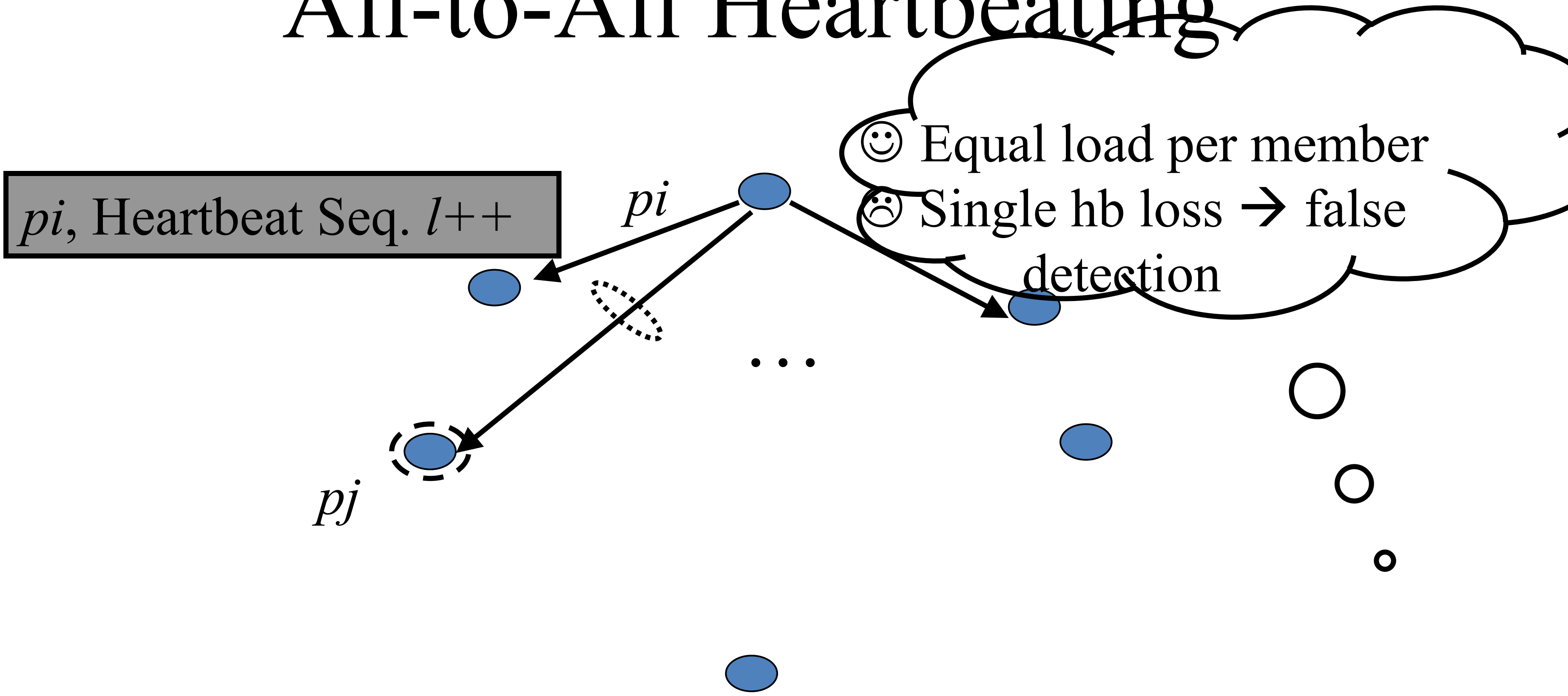


- Heartbeats sent periodically
- If heartbeat not received from *pi* within timeout, mark *pi* as failed

Ring Heartbeating



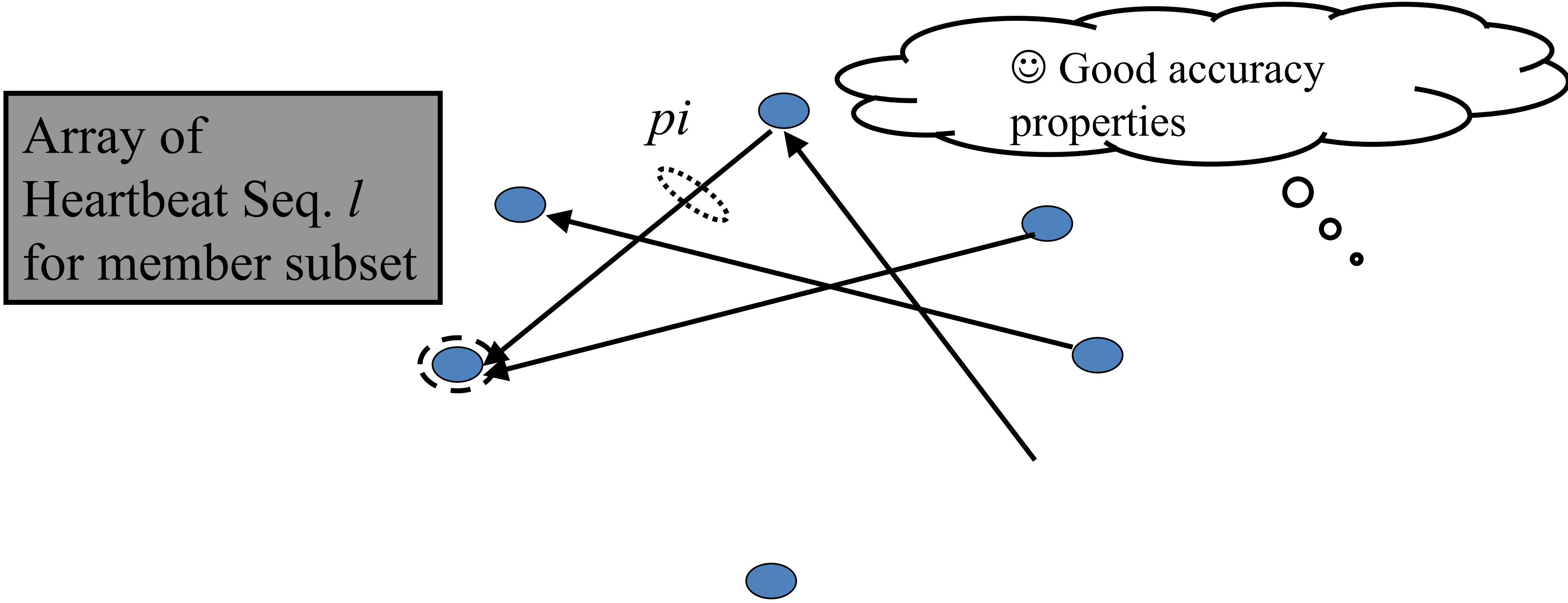
All-to-All Heartbeating



Next

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

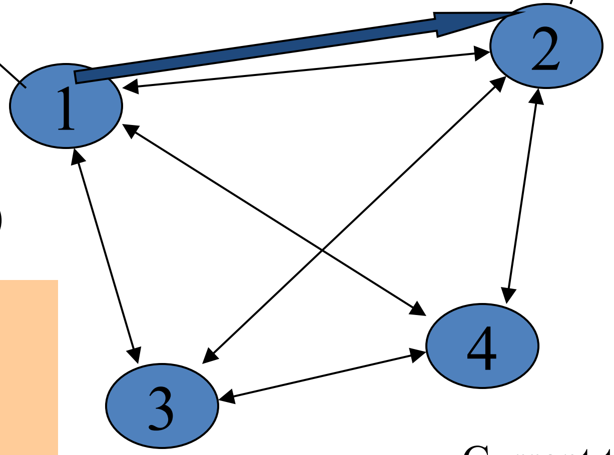
Gossip-style Heartbeating



Gossip-Style Failure Detection

1	10120	66
2	10103	62
3	10098	63
4	10111	65

Address — Heartbeat Counter — Time (local)



1	10118	64
2	10110	64
3	10090	58
4	10111	65

1	10120	70
2	10110	64
3	10098	70
4	10111	65

Current time : 70 at node 2

(asynchronous clocks)

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, member is marked as failed

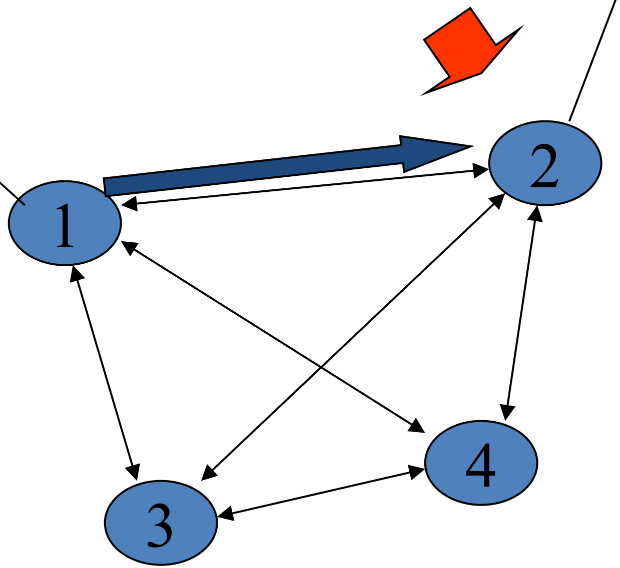
Gossip-Style Failure Detection

- 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 / 4	10098 / 10111	75 / 65
4	10111	65

Current time : 75 at node 2

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} ,$T_{cleanup}$ is increased?
- Tradeoff: False positive rate vs. detection time vs. bandwidth

Next

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

Failure Detector Properties …

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

Are application-defined Requirements

- Completeness → Guarantee always
- Accuracy → Probability *PM(T)*
- Speed → *T* time units
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Are application-defined Requirements

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

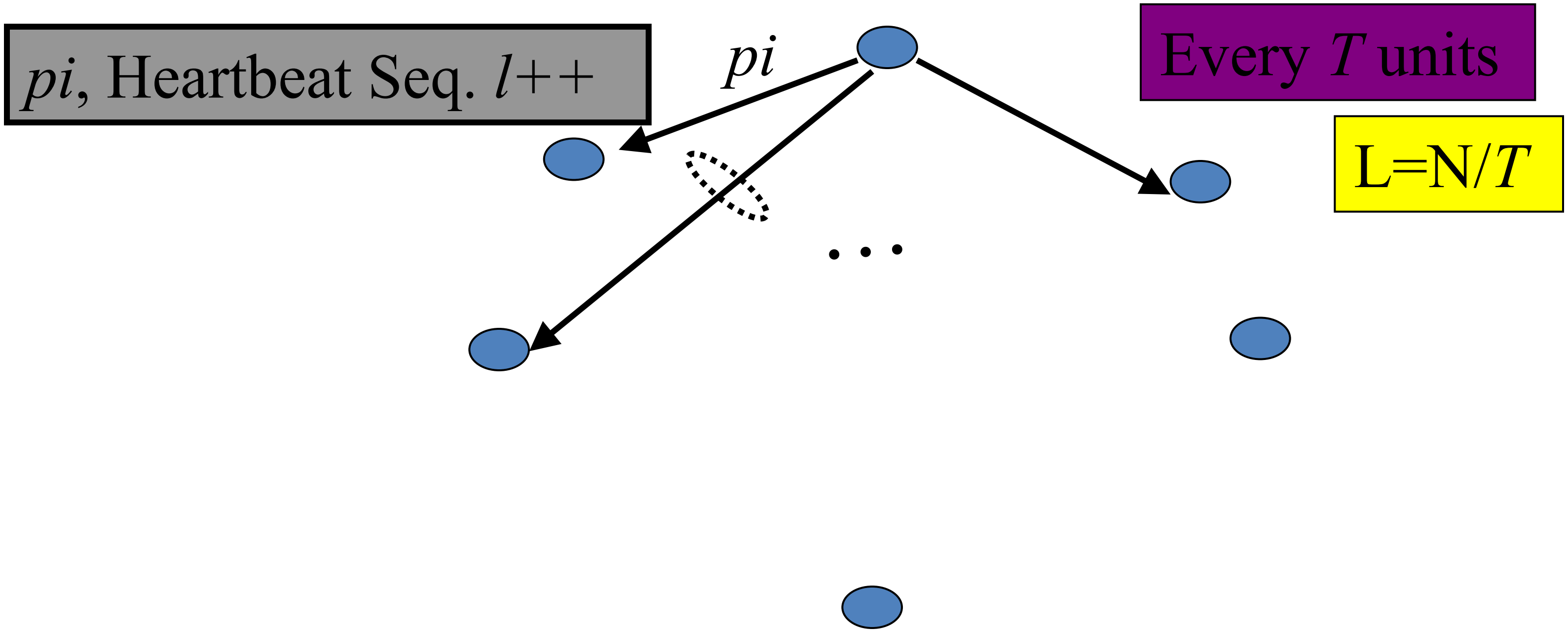
Guarantee always

Probability *PM(T)*

T time units

N*L: Compare this across protocols

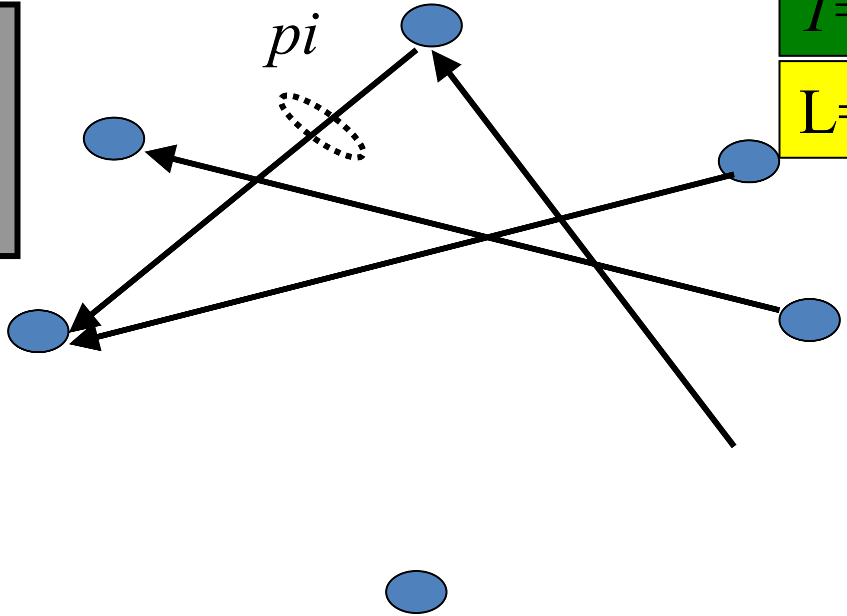
All-to-All Heartbeating



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}
- $$L^* = \frac{\log(PM(T))}{\log(p_{ml})} \cdot \frac{1}{T}$$

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?

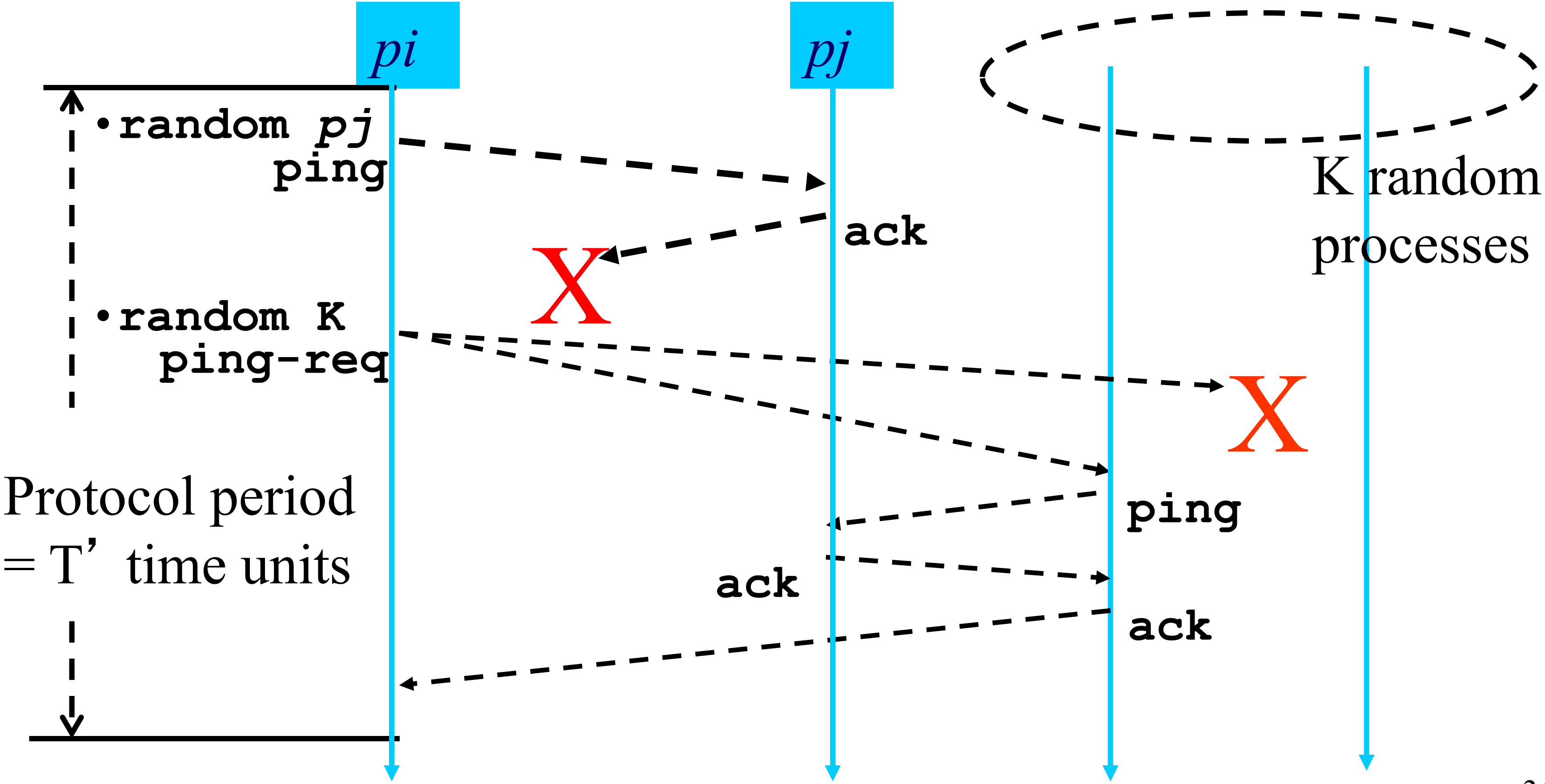
➲ Key:

- 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

- Prob. of being pinged in T'= $1-(1-\frac{1}{N})^{N-1}=1-e^{-1}$
- E[T] = $T'.\frac{e}{e-1}$
- Completeness: *Any* alive member detects failure
  - Eventually
  - By using a trick: within worst case *O(N)* protocol periods

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

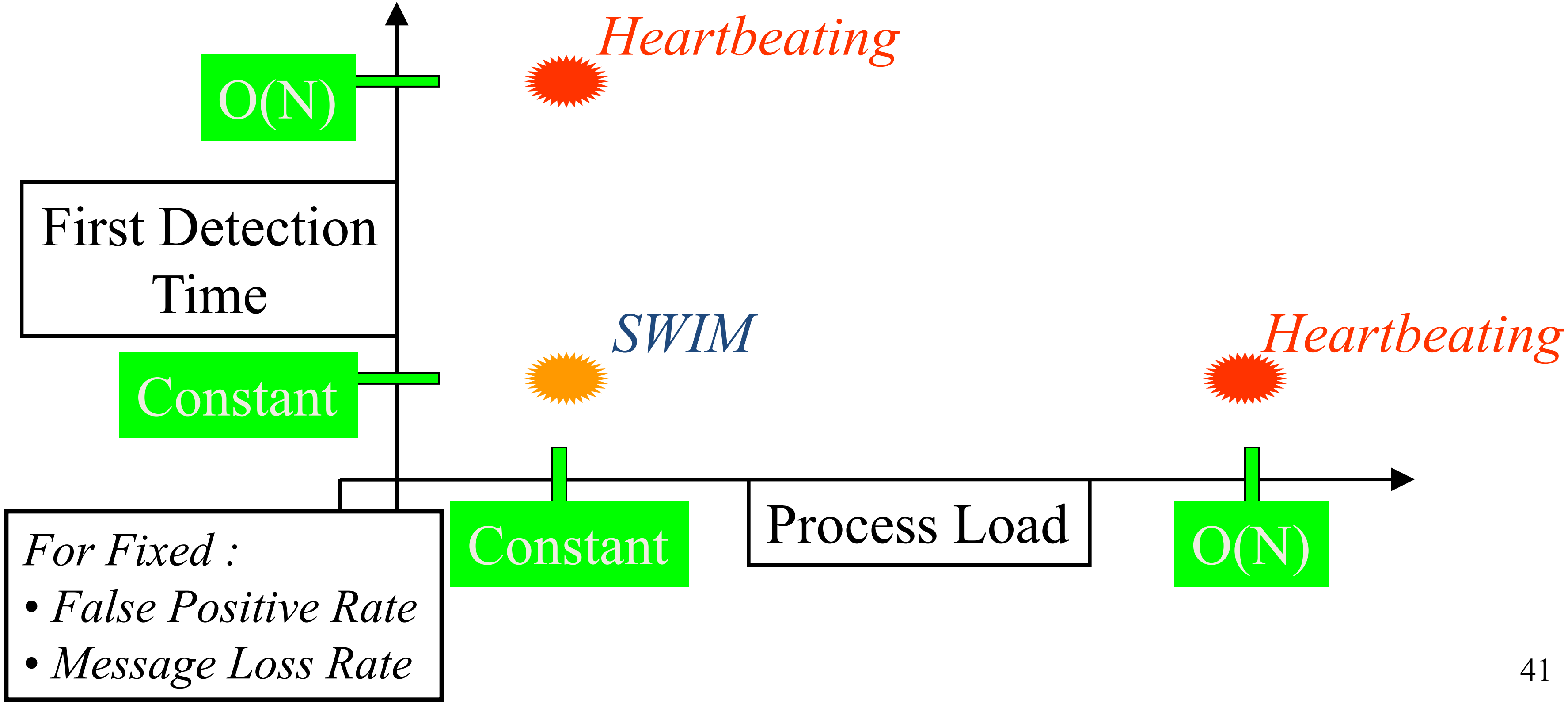
SWIM Failure Detector

Parameter	SWIM
First Detection Time	• Expected $\left[\frac{e}{e-1}\right]$ 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

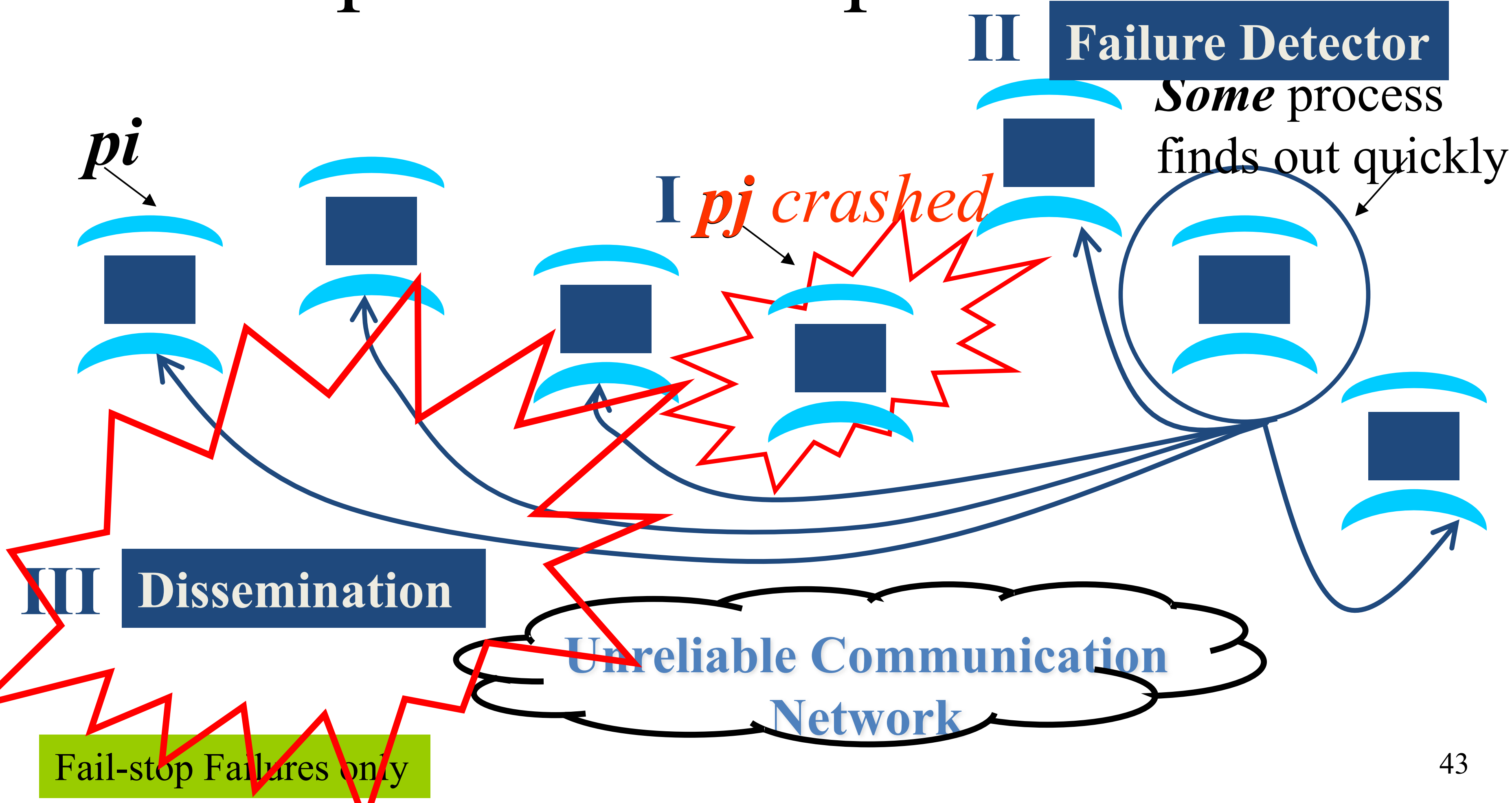
SWIM versus Heartbeating



Next

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

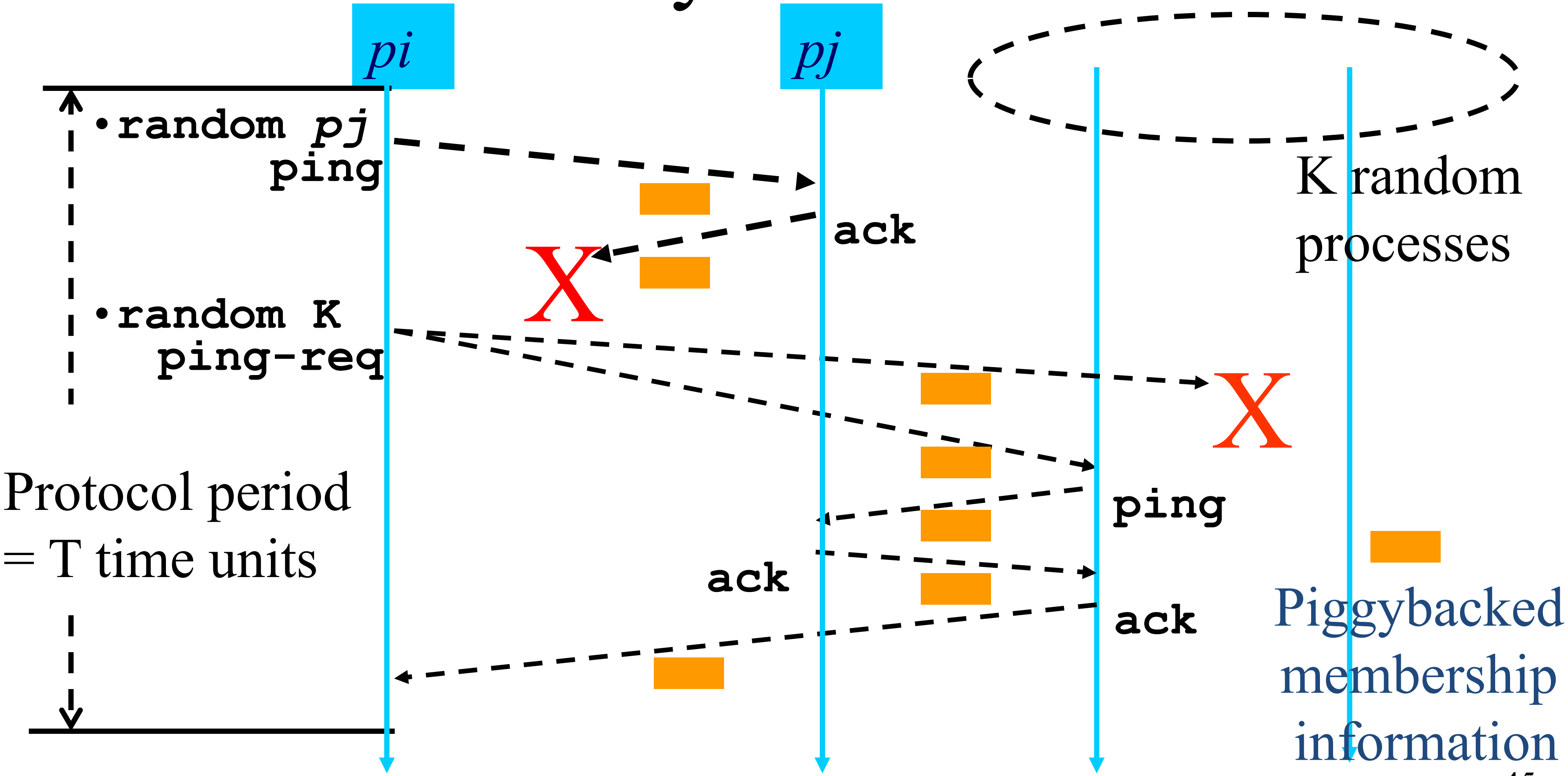
Group Membership Protocol



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



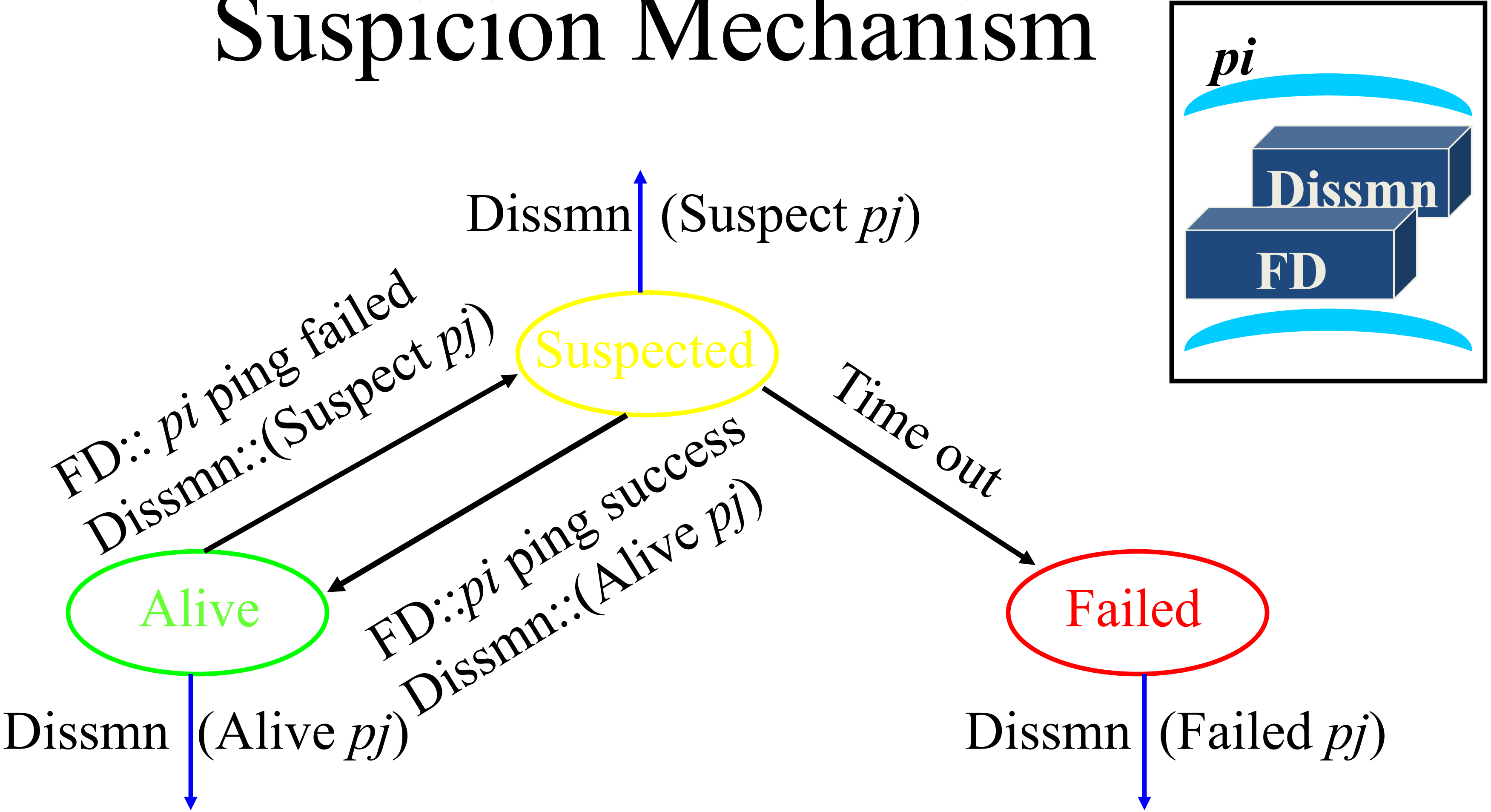
Infection-style Dissemination

- 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



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"
- 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!)

Grid Computing

- Please view two video lectures linked from Lectures Schedule page
  - Part of syllabus! (will appear on exams)
  - Slides also on webpage