On Crash Failures and Self-Stabilization (in Rings)

Felix Gärtner

(joint work with Ted Herman)

In memory of Synnöve Kekkonen-Moneta

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

École Polytechnique Fédérale de Lausanne (EPFL)
I& C, LPD, CH-1015 Lausanne, Switzerland
fgaertner@lpdmail.epfl.ch

Summary

- Revisit the area of "FTSS" (fault-tolerant and self-stabilizing systems).

Executive summary:

Positive and negative results
about mixing self-stabilization with silent crash failures

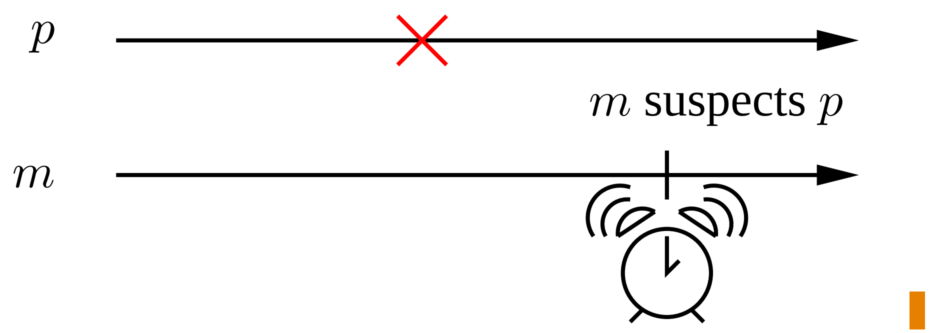
- Outline:
  - Recall major setting and system model (ring of processes).
  - Recall previous work.
  - Some positive and negative conditions for FTSS solvability:
    * General conditions in the spirit of *failure sensitivity* [Anagnostou and Hadzilacos 1993].
    * Characterization in terms of failure detectors [Chandra and Toueg 1996].

General System Model

- n asynchronous processes, at most $t < n$ can crash.
- Crash $=$ process stops making steps.
- Communication only by link registers [Dolev et al. 1993] (not message passing!).
- Initial state of registers arbitrary.
- Initial processor states arbitrary.
- Processes
  - can be uniform/non-uniform.
  - can be anonymous/have unique identifierss.
  - can have (no) common sense of direction.
  - may have access to failure detectors.

Failure Detectors [Chandra and Toueg 1996]

- Devices that can be queries and tell the operational state (up/down) of a remote process.



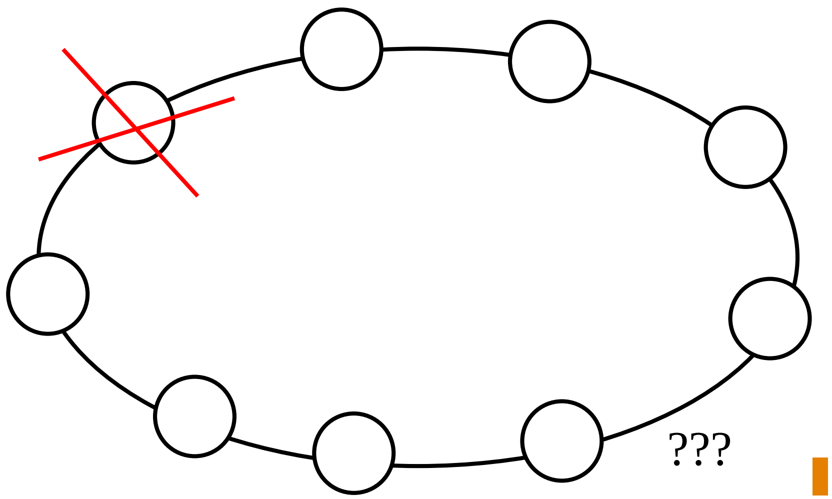
- Crash faults are undetectable in asynchronous systems: Failure detectors can be viewed as synchrony abstractions.

- Example: Class of perfect failure detectors $\mathcal{P}$ satisfies:
  - Process p is not suspected before it crashes.
  - If p crashes, it will eventually be suspected.

- Weaker notions possible (e.g., $\Diamond\mathcal{P}$).

Specific System Model [Anagnostou and Hadzilacos 1993]

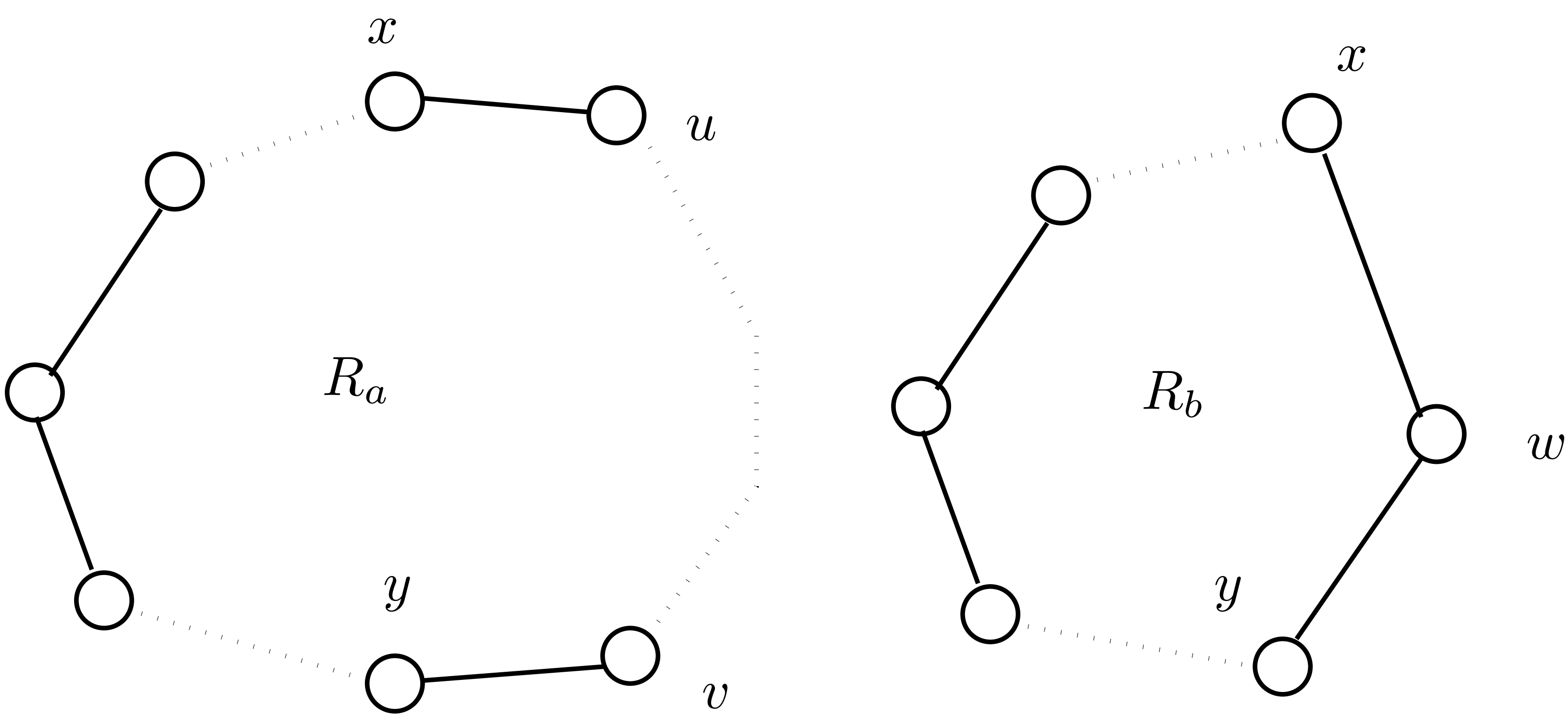
- Asynchronous ring of processes, $t = 1$.



- Processes should determine the size of the ring.

- Theorem by Anagnostou and Hadzilacos [1993]: There is no FTSS protocol for ring size.

Impossibility of ftss Ring Size Counting



- Proof idea: algorithm cannot distinguish between R_a and R_b (cannot even "lock" into R_b because of possibly corrupt inputs).

Other (Im)Possibilities on Rings

- Ring counting is impossible even if randomization is added, the ring is oriented, and processors have unique identifiers.

- It becomes solvable if $\diamond\mathcal{P}$ is added to the system [Beauquier and Kekkonen-Moneta 1997a].

- Also impossible:
  - deterministically assigning unique identifiers [Anagnostou and Hadzilacos 1993].
  - deterministic orientation [Beauquier et al. 1996].

- Both problems are solvable if randomization is added [Anagnostou and Hadzilacos 1993; Beauquier et al. 1996].

- Other related work omitted for brevity (additional slides on request).

Failure Sensitivity

- Generalized condition: failure sensitivity [Anagnostou and Hadzilacos 1993].
- A problem is failure sensitive if
  - for any state C which is legitimate if all are up there exists a process u and a state C' such that
    - C' is indistinguishable from C for all processes apart from u, and
    - C' is illegitimate if u has crashed, and
    - for all states C'' reached from C' (which are legitimate if u has crashed) are illegitimate if u has not crashed.
- Theorem: A failure sensitive problem has no FTSS solution.
- Intuition: Problem depends on the operational state of processes.
  - Example: leader election
    - If the leader has crashed, the system must eventually elect a new leader; danger of having two leaders.

Conditions for Solvability

- Failure sensitivity is a negative criterion (if a problem is failure sensitive, it is impossible).

- Failure sensitivity is also a condition very close to the impossibility proof.

- Two ways to study solvability:
  - strengthen the model (by adding failure detectors).
  - find structural conditions associated with problem specifications that enable FTSS solutions.

- We present two structural conditions . . .

Condition 1

- Assume: Solution is a function on the initial system state (a fixed point, not a non-terminating behavior).

- Condition 1:
  - any local state of a process could be part of a legitimate global state, and
  - for any process p, if the local states of p's neighbors could be part of a legitimate global state, then the local state of p can be changed (locally) to be legitimate without having to change the local state of its neighbors.

- Lemma: If a problem satisfies Condition 1 then there exists a randomized FTSS protocol to solve it.

- Example: 3-coloring the ring is FTSS-solvable.

- Proof assumes unique totally ordered identifiers (this is where randomization is necessary). Use identifiers to break symmetry in the questions "who follows who".

Condition 2

- Assume: (1) Problem is a function on the initial system state, and (2) there is a total order between all solutions.

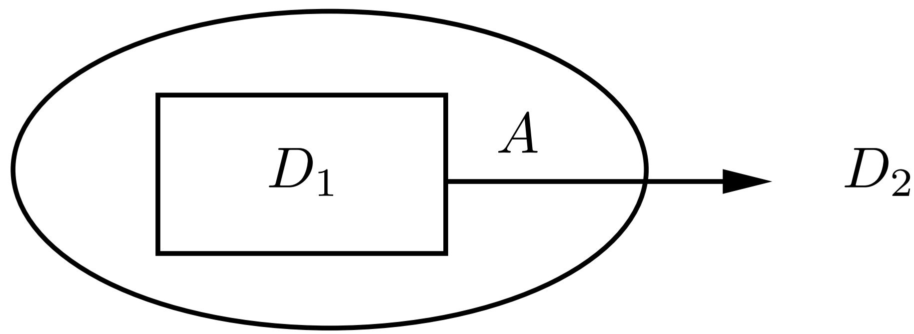
- Condition 2: If a state is legitimate for a given ring, then any ring segment also has a legitimate state.

- Example: Finding upper bound on ring size is FTSS solvable.

- Proof assumes unique and totally ordered identifiers (randomization needed here again):
  - Everybody takes periodic snapshots.
  - Eventually, snapshot image at every process will converge, and solution function will be calculated in the same manner at all processes.
  - A crashed processes may "make the ring look larger", but any solution for the larger ring is also a solution for any ring segment.

- Now turn to failure detectors.

Comparing Failure Detectors



- Set $\mathcal{D}$ of all failure detectors, take $D_1, D_2 \in \mathcal{D}$.

- D_1 weaker than D_2 ($D_1 \leq D_2$) if there exists an algorithm A which transforms output of D_1 into output of D_2.

- Failure detector D is weakest to solve a problem P:

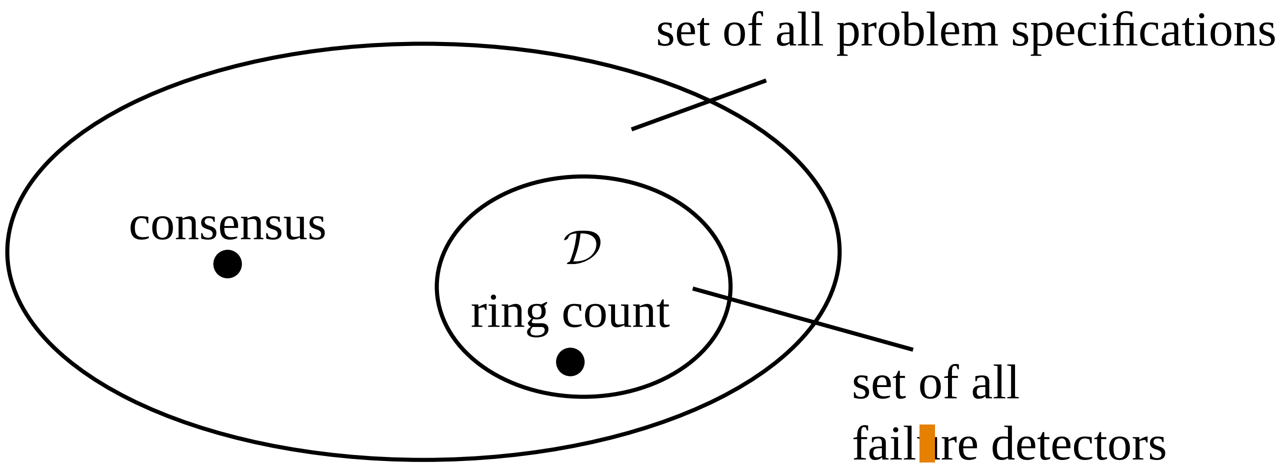
  1. D allows to solve P
  2. Every failure detector D' which allows to solve P is at least as strong as D ($D' \geq D$).

Failure Detectors for Ring Counting

- $\Diamond\mathcal{P}$ allows to solve ring counting, but can we find weaker failure detectors that also do the job?
- This is pretty hard. Two first attempts:
  - boundedly inaccurate failure detector:
    - a crashed process is eventually permanently suspected by both neighbours
    - there exists a constant k such that for a non-crashed process, eventually, in any sequence of k queries there is at least one correct response.
  - the anonymous failure detector:
    - tells whether or not at least one neighbor has crashed.
- Are they really weaker?
- No, both can be transformed into $\Diamond\mathcal{P}$.

A Note on Weakest Failure Detectors

- Failure detectors have a formal definition [Chandra and Toueg 1996]: class of programs defined as a function of failures (and nothing else).
  - Example: “respond 42 to every query” is a failure detector.
  - Rephrase question about sufficient failure detector: Find a program in the set of all failure detectors $\mathcal{D}$ that allows to solve the problem.

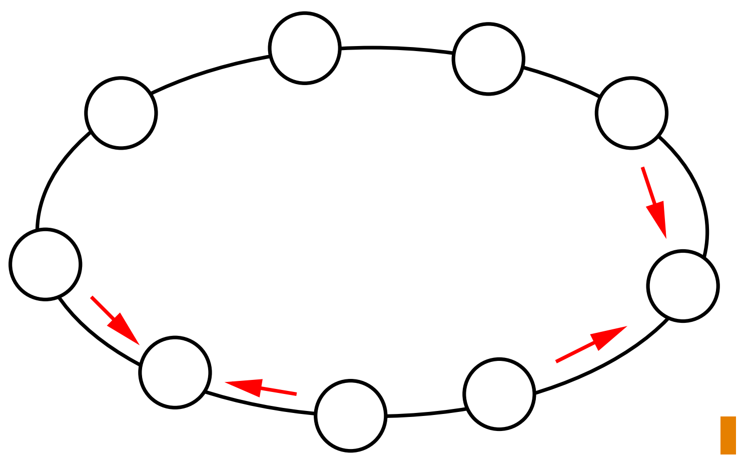


- Ring counting itself is the weakest failure detector.

- Want to find a problem equivalent to ring counting which “looks” like a failure detector.

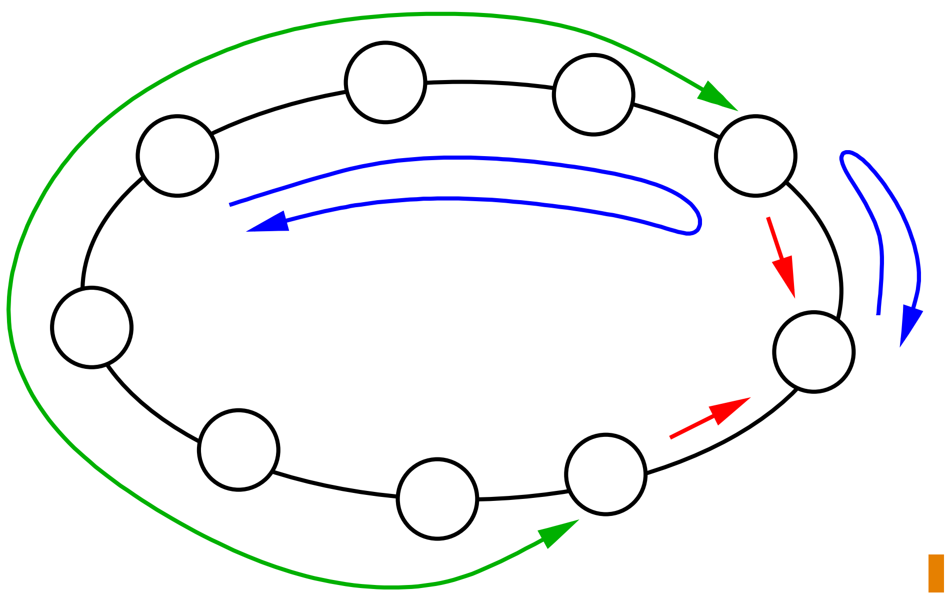
A Strange Failure Detector

- Bounded suspicion failure detector (obviously weaker than $\Diamond\mathcal{P}$):
  - if a process has crashed both neighbors will eventually permanently suspect that process.
  - eventually there will be at most one suspected process in the system (and this process does not change "too often").



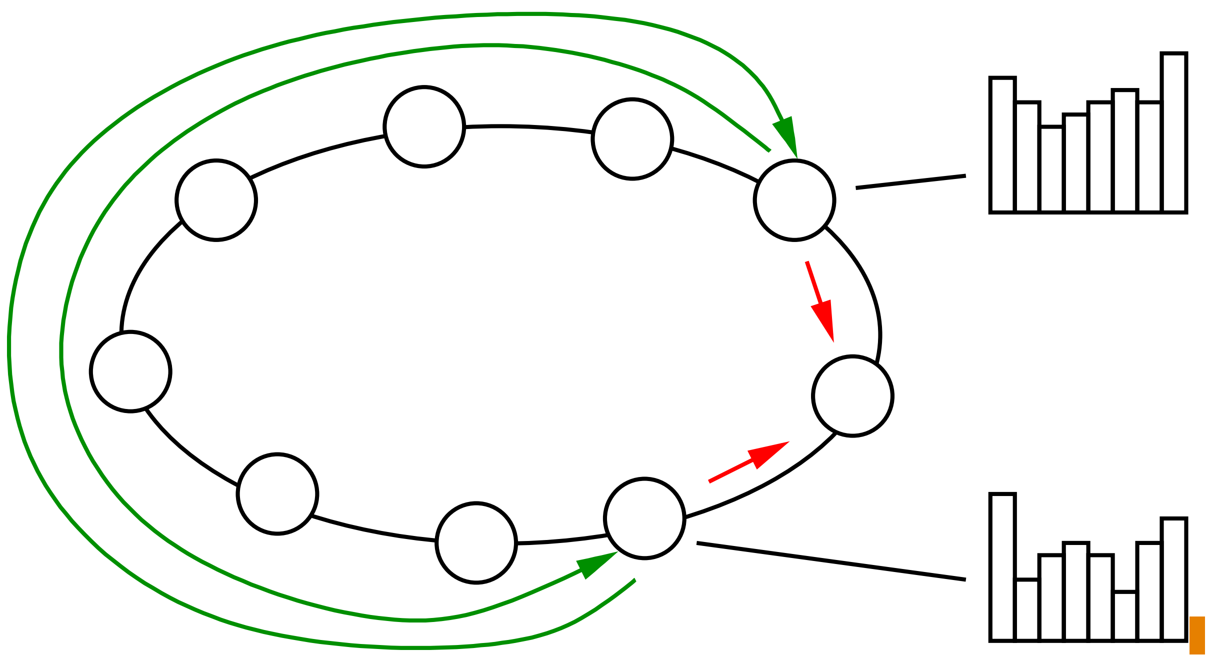
Sufficient to Solve Ring Counting

- Failure detector "narrows down" uncertainty about size of the ring (allows to distinguish cases of impossibility proof).
- Can use the ring size counting protocol of Beauquier and Kekkonen-Moneta [1997a] without modification.



Necessary to Solve Ring Counting

- From knowledge of the ring size, build the bounded suspicion failure detector. (Hard part is accuracy.)
  - Keep on sending explorer messages with time-to-live $n-2$. Upon receipt, increase counter.
  - Detect a certain pattern on all the counters of the ring.
  - Lock in to a suspicion for increasingly longer periods of time.



Summary

- Explore positive results about mixing self-stabilization with silent crash failures.
- Given two new structural conditions for solvability of problems on a ring.
- Investigated the weakest failure detector for determining the ring size.
  - Formalizing the failure detector is not easy.
  - Locking into a process with increasingly longer periods of time needs synchronization between failure detector and upper layer protocol.
- Some possible future work questions:
  - Is there anything special about failure detectors for FTSS in contrast to "normal" failure detectors?
  - Do there exist problems for which no "traditional" failure detector (with up/down interface) exists?

Acknowledgments

• Slides produced using pdfLaTeX and Klaus Guntermann's PPower4.

References

Anagnostou, E. and Hadzilacos, V. 1993. Tolerating transient and permanent failures. In *WDAG93 Distributed Algorithms 7th International Workshop Proceedings, Springer LNCS:725* (1993), pp. 174–188.

Arora, A. and Gouda, M. 1993. Closure and convergence: a foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering 19*, 1015–1027.

Beauquier, J., Debas, O., and Kekkonen, S. 1996. Fault-tolerant and self-stabilizing ring orientation. In *Structure, Information and Communication Complexity (SIROCCO96)* (1996), pp. 59–72. Carleton University Press.

Beauquier, J. and Kekkonen, S. 1996. Making FTSS is hard. In *International Conference on Software Engineering (ICSE'96)* (1996), pp. 91–96.

Beauquier, J. and Kekkonen-Moneta, S. 1997a. Fault-tolerance and self-stabilization: impossibility results and solutions using self-stabilizing failure detectors. *International Journal of Systems Science 28*, 11, 1177–1187.

Beauquier, J. and Kekkonen-Moneta, S. 1997b. On ftss-solvable distributed problems. In *Proceedings of the Third Workshop on Self-Stabilizing Systems* (1997), pp. 64–79. Carleton University Press.

Chandra, T. D. and Toueg, S. 1996. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM 43*, 2 (March), 225–267.

Dolev, S., Israeli, A., and Moran, S. 1993. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing 7*, 3–16.

Gopal, A. and Perry, K. 1993. Unifying self-stabilization and fault-tolerance. In *PODC93 Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing* (1993), pp. 195–206.

Masuzawa, T. 1995. A fault-tolerant and self-stabilizing protocol for the topology problem. In *Proceedings of the Second Workshop on Self-Stabilizing Systems* (1995), pp. 1.1–1.15.

Matsui, M., Inoue, M., Masuzawa, T., and Fujiwara, H. 2000. Fault-tolerant and self-stabilizing protocols using an unreliable failure detector. *IEICE Transactions on Fundamentals of Electronic Communications and Computer Sciences E83D*, 10, 1831–1840.

Nesterenko, M. and Arora, A. 2002. Stabilizing dining philosophers with optimal crash failure. In *ICDCS02 The 22nd IEEE International Conference on Distributed Computing Systems* (2002), pp. ??–??

Additional Emergency Slides

Previous Work in More Detail

- Usually Gopal and Perry [1993], Anagnostou and Hadzilacos [1993] and Arora and Gouda [1993] are referenced as starting points for explicit consideration of crash failures in the context of self-stabilization.

- Note that detectable permanent faults can be treated as transient faults in the self-stabilization methodology.

- We will start off from Anagnostou and Hadzilacos [1993]; four lines of follow-up work:
  - Masuzawa [1995]: FTSS for the topology problem.
  - Beauquier, Debas, and Kekkonen [1996, Beauquier and Kekkonen-Moneta [1997b, Beauquier and Kekkonen-Moneta [1997a, Beauquier and Kekkonen [1996]: solving FTSS with failure detectors.
  - Matsui, Inoue, Masuzawa, and Fujiwara [2000]: FTSS using unreliable failure detector.
  - Nesterenko and Arora [2002]: stabilizing dining philosophers with optimal crash failure.

Previous Work (cont.)

- Results from Anagnostou and Hadzilacos [1993]:
  - FTSS impossible for counting ring size.
  - definition of "failure sensitive"
  - FTSS possible for establishing unique IDs
- Results from Masuzawa [1995]
  - FTSS topology problem solvable when neighbor IDs are known (k crashes in $(k+1)$-connected networks)
  - FTSS not solvable using only connectivity or only neighbor IDs
- Results from Beauquier and Kekkonen-Moneta [1997a]
  - FTSS impossible for round synchronization
  - $(k+1)$-FTSS counting impossible for k-centered ring
  - $\diamond\mathcal{P}$ solves FTSS