SOURCES OF FAILURE

2

2

Why do Systems Fail?

- Gray: “Conventional well-managed transaction processing systems fail about once every two weeks. The ninety minute outage outlined above translates to 99.6% availability for such systems. 99.6% availability sounds wonderful, but hospital patients, steel mills, and electronic mail users do not share this view – a 1.5 hour outage every ten days is unacceptable. Especially since outages usually come at times of *peak demand*.”

3

3

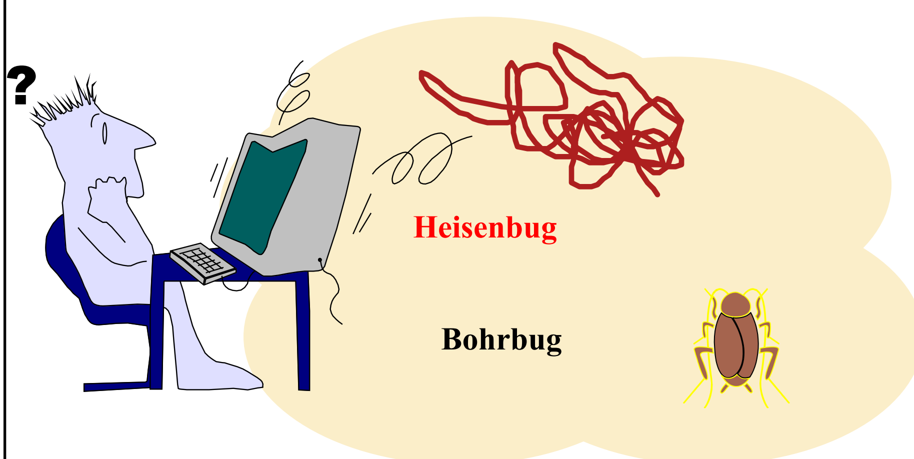
Why do Systems Fail?

- Operator Errors
  - Gray: 42% in a transaction processing system
  - Patterson: 59% among 3 anonymous Web sites
  - Three Mile Island
  - Fly-by-wire (Airbus)
- Autonomic computing
- Automation irony

4

4

Sources of Software Faults



5

5

Sources of Heisenbugs

- Poor Algorithms
- Missing Deadlines
- Race Conditions
- Roundoff Error Build Up
- Memory Leaks
- Broken Pointers
- Register Misuse (embedded software)

6

6

Bugs in a typical distributed system

- Component crash or network partition
- Other components depend on it
- Chain of dependencies
  – Gradual failover

7

7

Leslie Lamport

- “A distributed system is one in which the failure of a machine you have never heard of can cause your own machine to become unusable.”

- Dependency on critical components

8

8

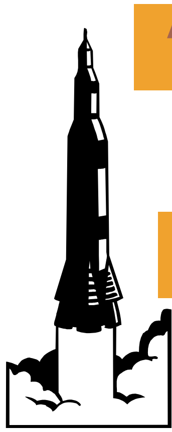
Example

- Arianne rocket: modular design
- Guidance system
  - Flight telemetry
  - Rocket engine control
  - …. Etc
- Upgraded some rocket components
- Hidden assumptions invalidated

9

9

Arianne Rocket



Attitude Control

Telemetry

Guidance

Altitude

Accelerometer

Thrust Control

10

10

Arianne Rocket

Attitude Control

Telemetry

Guidance

Altitude

Overflow!

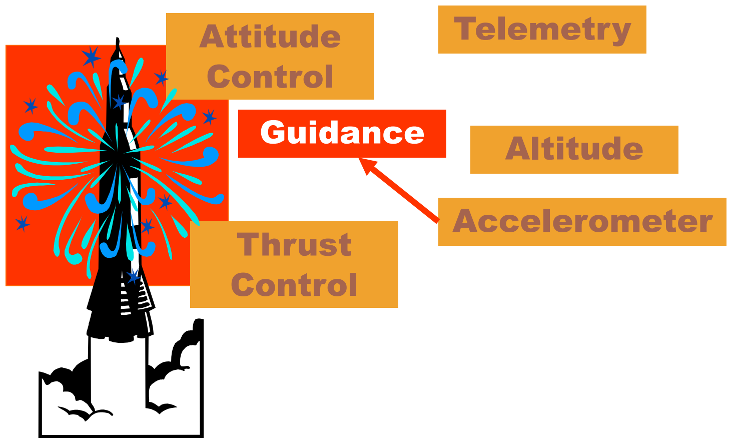
Accelerometer

Thrust Control

11

11

Arianne Rocket



12

12

Insights?

- Correctness depends on the environment
- Components make hidden assumptions

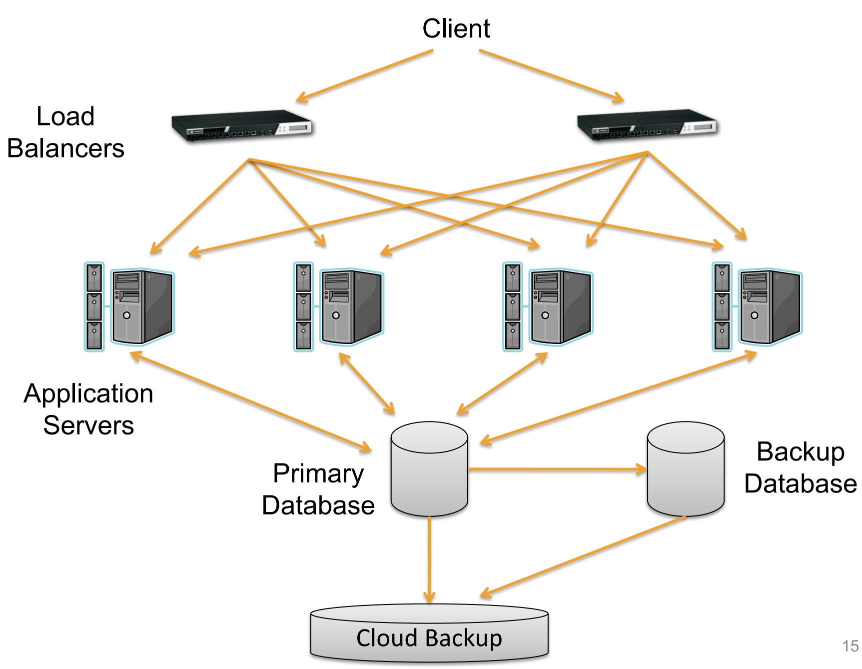
13

13

FAILURE MODELS

14

14



15

15

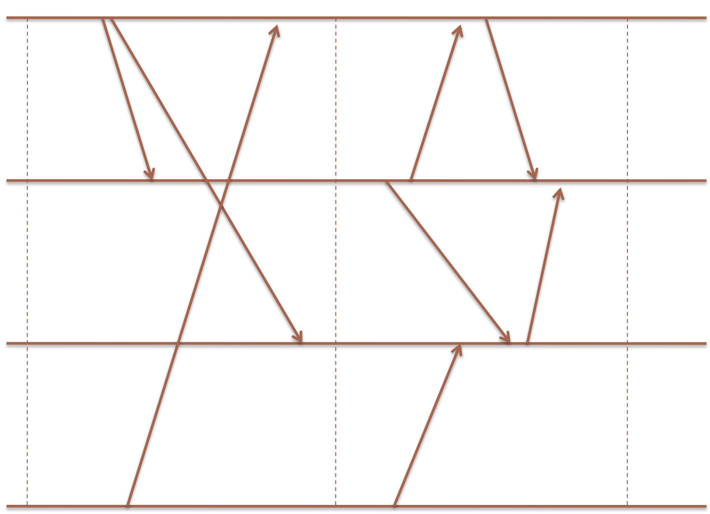
System Models

- Synchronous System
  - Bounded message delivery time
  - Bound on clock drift
  - Bound on computing time
  - Strong assumptions (too strong?)
- Asynchronous System
  - No bounds
  - Very weak model
- Partial Synchrony
  - Approximate bounds on delays, but unknown

16

16

Synchronous System



17

17

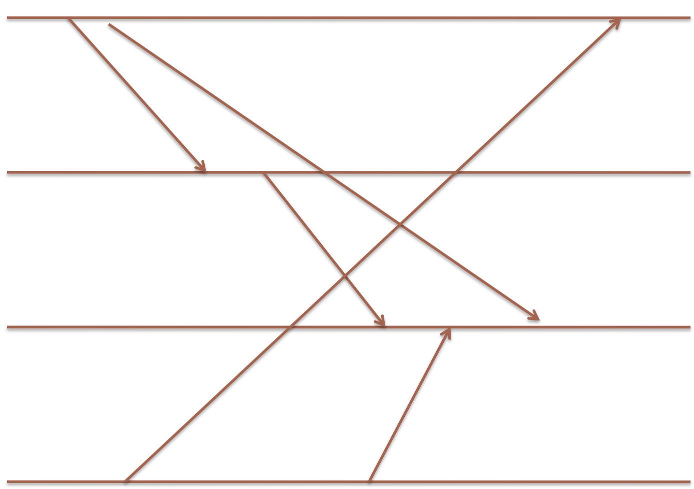
System Models

- Synchronous System
  - Bounded message delivery time
  - Bound on clock drift
  - Bound on computing time
  - Strong assumptions (too strong?)
- Asynchronous System
  - No bounds
  - Very weak model
- Partial Synchrony
  - Approximate bounds on delays, but unknown

18

18

Asynchronous System



19

19

Categories of failures

- Fail-stop failures
  - System support
  - Overcome message loss by resending packets
    - must be uniquely numbered
  - Easy to work with… but rarely supported

20

20

Categories of failures

- Network Partition
  - Failure of router isolates subnet
  - Danger: Processes in subnet continue
  - Result: inconsistency
  - Solutions: Quorum consensus, etc.

21

21

Categories of failures

- Crash faults, message loss
  – Common in real systems
  – Cannot be directly detected
  – Classic impossibility results!

22

22

Categories of failures

- Non-malicious Byzantine failures
  – Pretty much anything
  – Random failure, not coordinated
  – Common mode of failure

23

23

Categories of failure

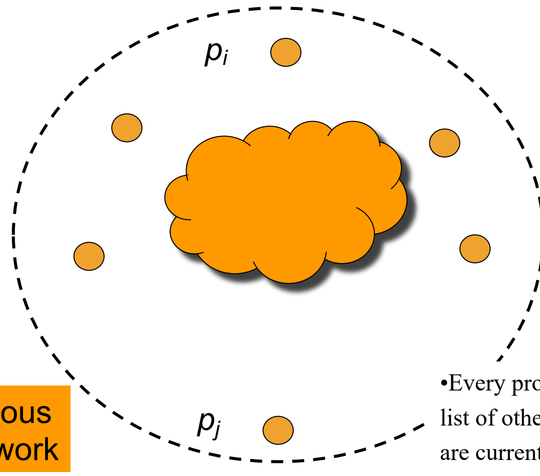
- Malicious (Byzantine?) failures
  - Very costly to defend against
  - Typically used in very limited ways
  - e.g. key mgt. server

24

FAILURE DETECTION

25

Failure Detection



Asynchronous Lossy Network

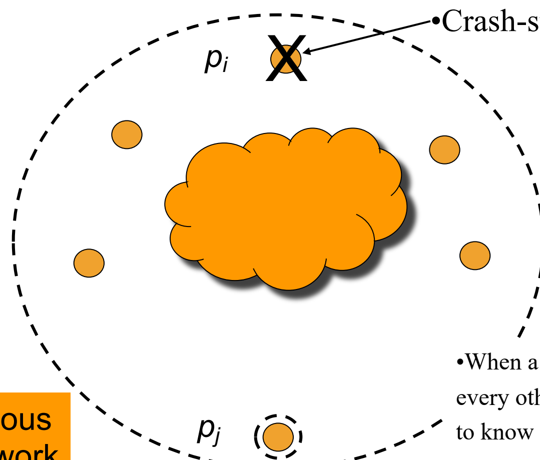
- Every process p_j maintains a list of other processes p_i that are currently alive (non-faulty)

26

26

Failure Detection

- Crash-stop failure



Asynchronous Lossy Network

- When a process p_i fails, every other process p_j needs to know of its failure

27

27

How is it Useful?

28

28

Metrics for Protocols

- Completeness
- Accuracy

Correctness

- Speed
  - First detection time
  - Dissemination time
- Scalability
  - Load : network load, per node overhead
  - How above metrics change with N
- Resilience
  - Performance under many failures

Performance

29

29

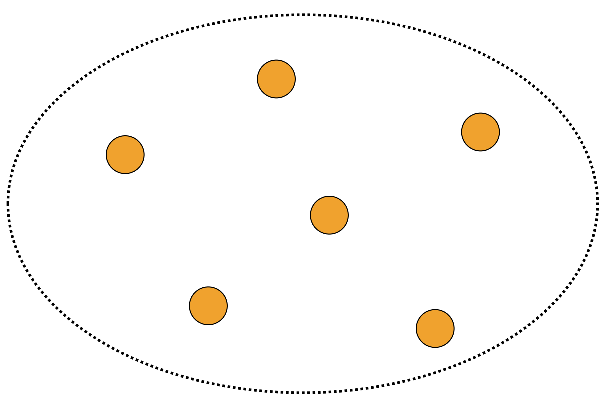
Metrics for Protocols

- Completeness
  - Failure eventually detected by every non-faulty node
- Accuracy
  - No mistake in detection: no alive (non-faulty) node detected as failed

30

30

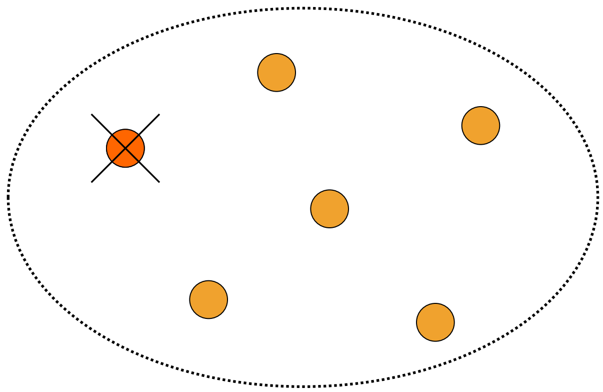
Completeness & Accuracy



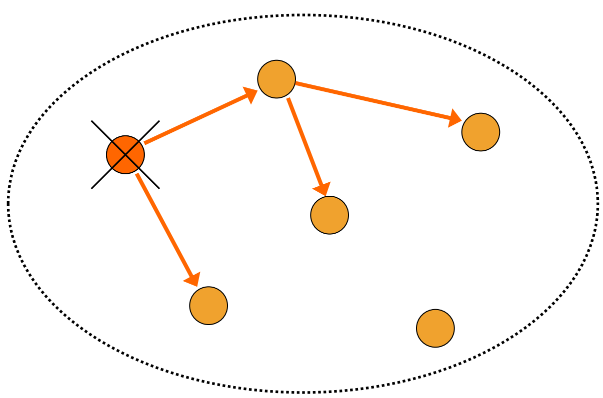
31

31

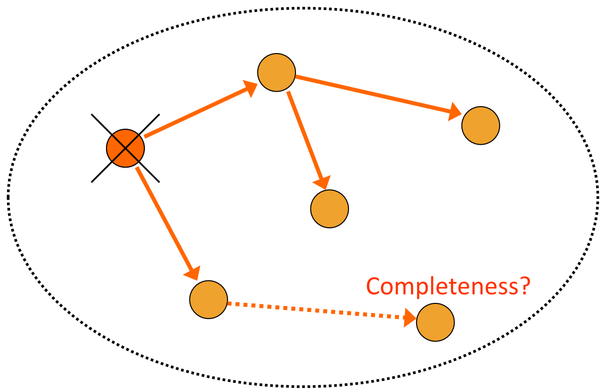
Completeness & Accuracy



Completeness & Accuracy



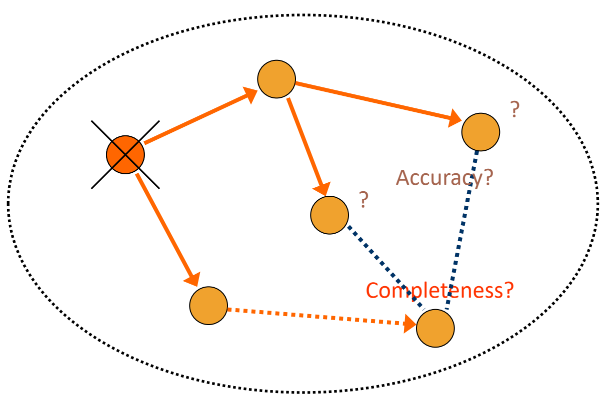
Completeness & Accuracy



34

34

Completeness & Accuracy



FLP Impossibility result: It is impossible to design a failure detector that is both complete and accurate in an asynchronous network [Chandra and Toueg 1990]

35

35

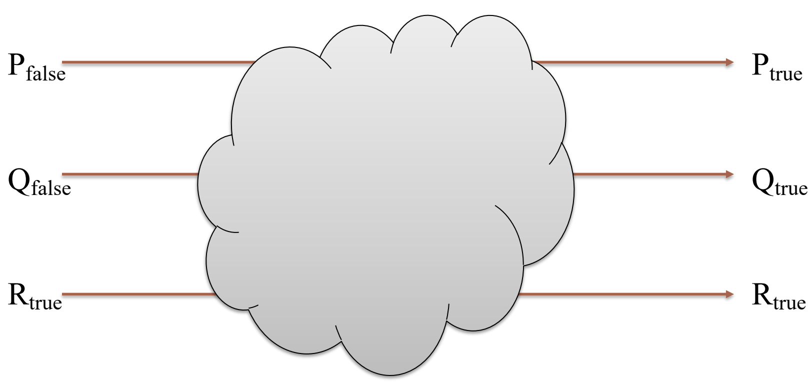
DISTRIBUTED PROBLEMS

36

36

Distributed Problems

- Global Consensus



37

37

Distributed Problems

- Global Consensus
  - N peer processes, each have input true or false
  - System model: Asynchronous
  - Failure model: Crash stop
  - **Agreement:** Everyone agrees to output same value
  - **Termination:** Protocol must finish

38

38

Questions

- What is an easy solution to consensus, as stated so far?
- How can we refine the problem to rule out such solutions?
- What is a solution, if we know that there is **exactly one** failed node in the network?

39

39

Distributed Problems

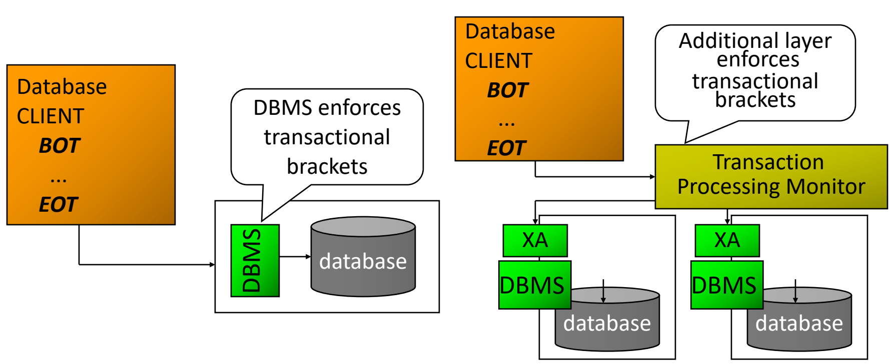
- Global Consensus
  - N peer processes, each have input true or false
  - System model: Asynchronous
  - Failure model: Crash stop
  - **Agreement:** Everyone agrees to output same value
  - **Validity:** Output value is one of the input values
  - **Termination:** Protocol must finish

40

40

Distributed Problems

- *Non-blocking* atomic commitment



41

41

Distributed Problems

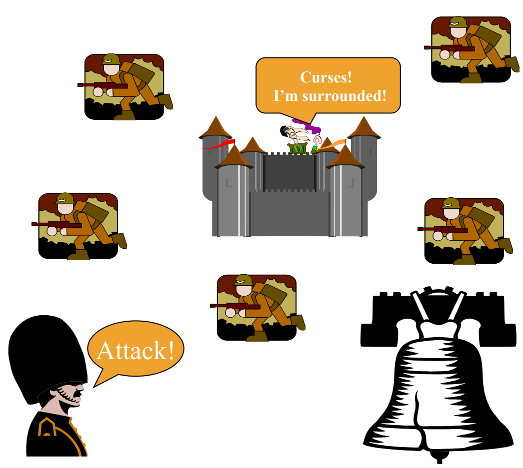
- *Non-blocking* atomic commitment
  - N databases, each involved in a transaction
  - Commit all updates or roll back (abort) all updates
  - System model: Asynchronous system
  - Failure model: Crash stop
  - **Agreement:** No two DBs can make different decisions
  - **Commit Validity:** Only commit if all DBs commit
  - **Abort Validity:** Only abort if one DB aborts
  - **Termination:** Every non-crashed DB must decide

42

42

Distributed Problems

- Byzantine Agreement ("Byzantine Generals")



43

43

Distributed Problems

- Byzantine Agreement (“Byzantine Generals”)
  - N generals, coordinating attack
  - Variant: One general issues orders to lieutenants
  - System model: Synchronous system
  - Failure model: Byzantine
  - **Agreement:** Loyal lieutenants obey same order
  - **Validity:** If general is loyal, lieutenants obey his order
  - **Termination:** Protocol must finish

44

44

Other Problems

- Leader Election
  - A leader must eventually be chosen
  - Other processes must learn of decision
- Deadlock Detection
- Termination Detection
- Garbage Collection
- …

45

45

Properties of Solutions

- **Safety:** Algorithm is guaranteed to leave an incorrect state
  - Violation: If property is violated in execution E, then there is another execution E'
    same as E up until property violation
    and property continues to be violated in E'
- **Liveness:** Algorithm must make progress
  - 2PC for atomic commitment

46

46

CONSENSUS AND FAILURE DETECTION

47

47

FLP: Impossibility of Consensus

- Consensus is impossible
  - … in asynchronous system
  - … with crash-stop failures
- Adversary argument:
  - Any protocol cannot block
  - Delay delivery of critical message
  - Force system to reconfigure
  - Deliver message now it’s no longer critical
  - Continue ad infinitum

48

48

FLP: Impossibility of Consensus

- Consensus is impossible
  - … in asynchronous system
  - … with crash-stop failures
- Adversary argument:
  - Relies on only one failure (message loss)
    - …which never actually happens!
  - *Key point: protocol cannot distinguish failure from delay*

49

49

FLP: Impossibility of Consensus

- Suppose we knew *exactly* one failure
- If N processes, then every process broadcasts its input (true or false) to every other process
- Each process: Make decision after receiving N-1 broadcasts

50