# Failure Models

Dominic Duggan
Stevens Institute of Technology
Including materials by K. Birman



## SOURCES OF FAILURE

## 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*."

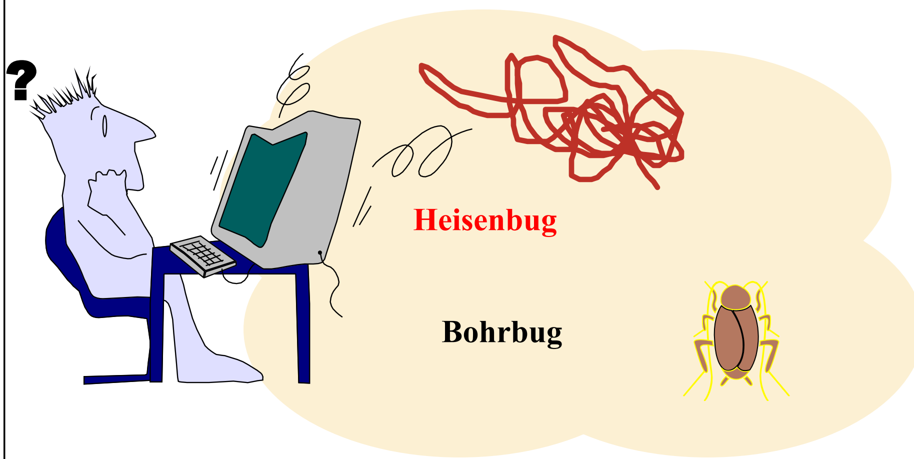
3

## 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

## Sources of Software Faults




5

## Sources of Heisenbugs

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

6

## Bugs in a typical distributed system

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

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

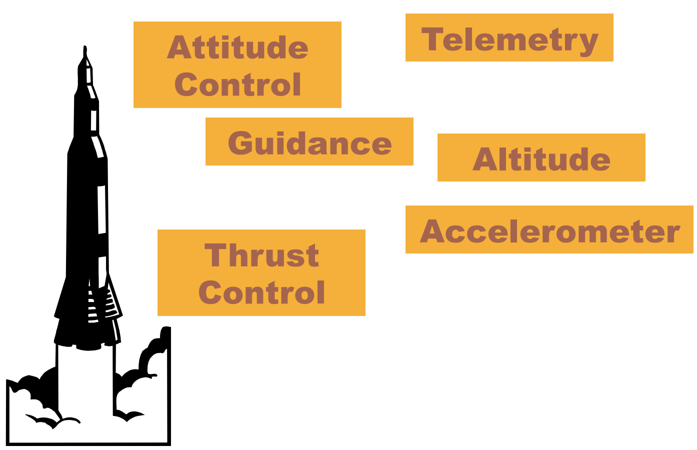
8

# Example

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

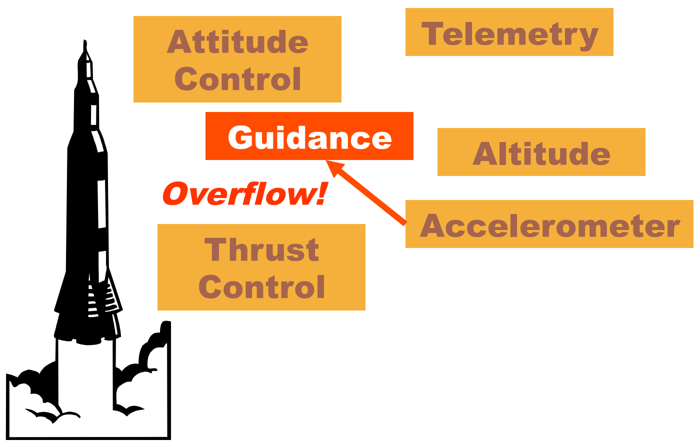
9

# Arianne Rocket



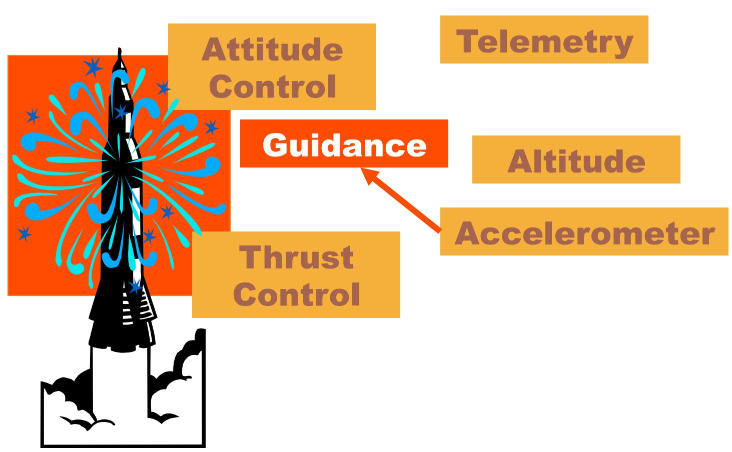

10

# Arianne Rocket




11

# Arianne Rocket




12

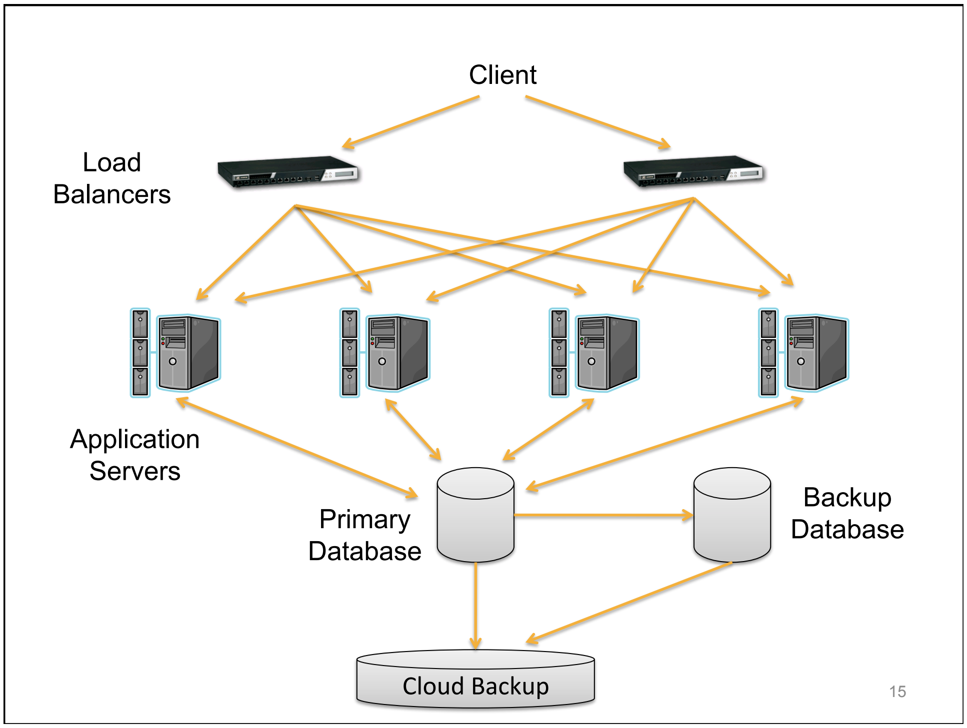
## Insights?

- Correctness depends on the environment
- Components make hidden assumptions

13

## FAILURE MODELS

14

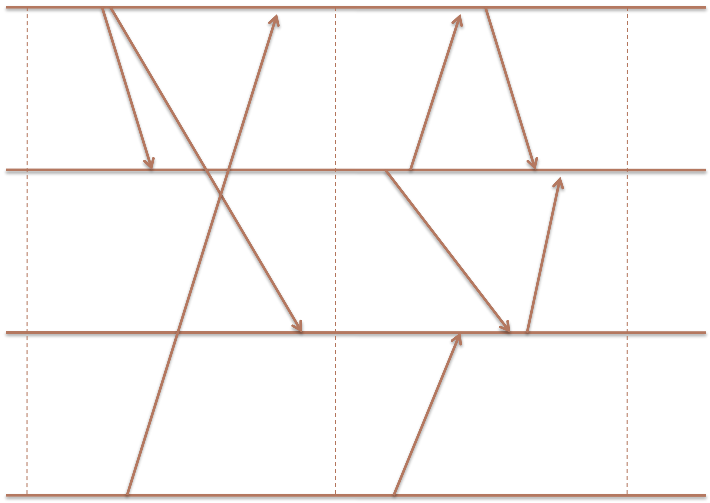



# System Models

- 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

## Synchronous System

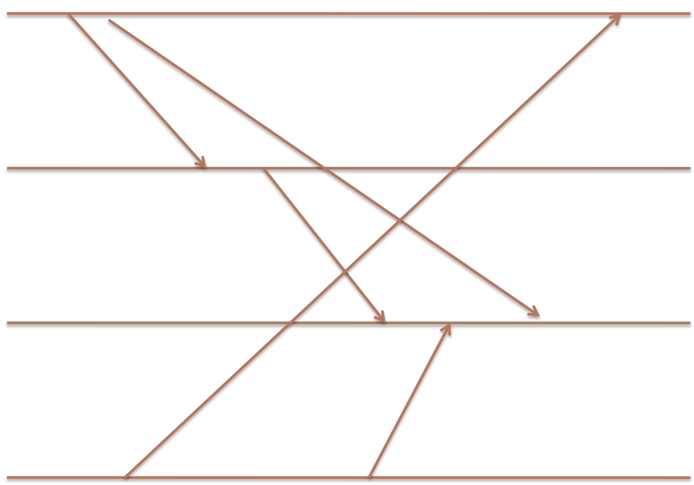




## System Models

- 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

# Asynchronous System





# 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

## Categories of failures

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



## Categories of failures

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

## Categories of failures

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



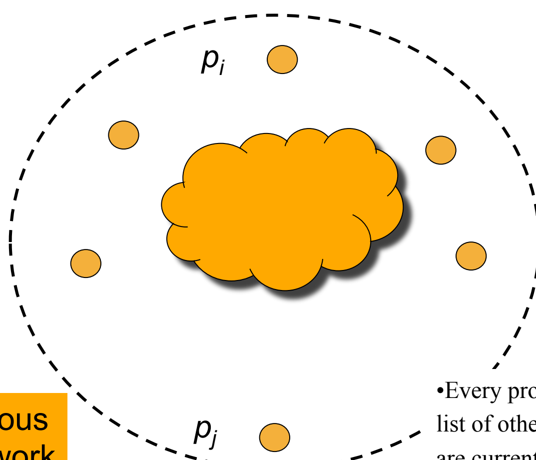
## Categories of failure

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

## FAILURE DETECTION

25

## Failure Detection

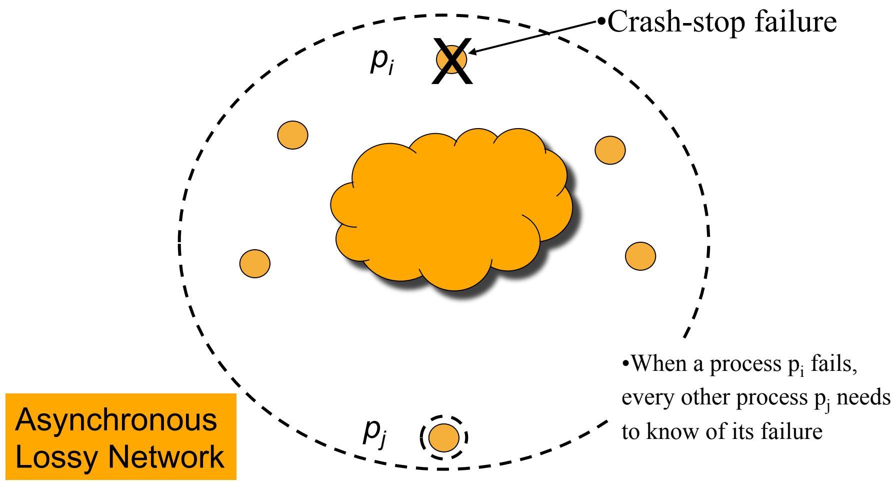



Asynchronous Lossy Network

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

26

## Failure Detection




27

## How is it Useful?

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

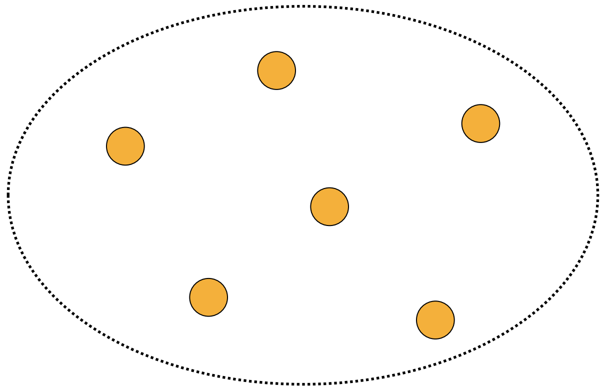
29

## 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

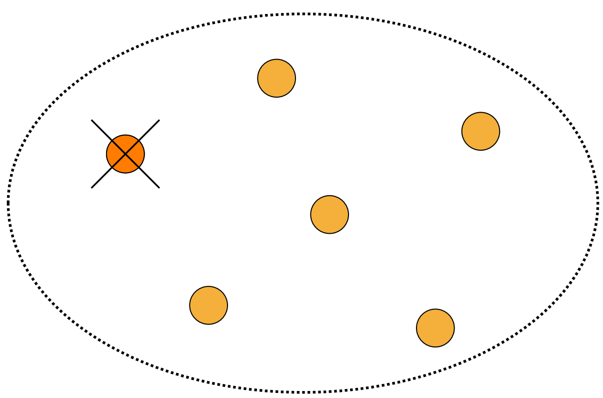
30

## Completeness & Accuracy



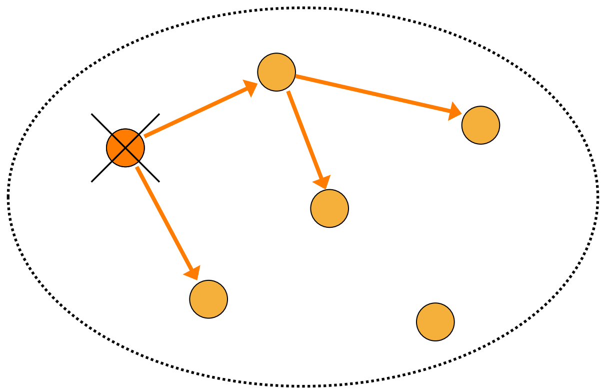
31

## Completeness & Accuracy



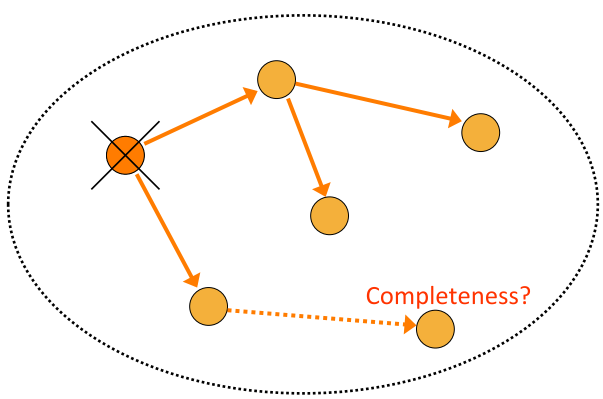
32

# Completeness & Accuracy



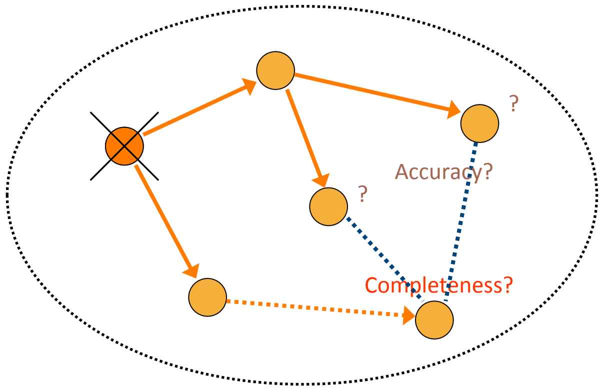
33

# Completeness & Accuracy




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]

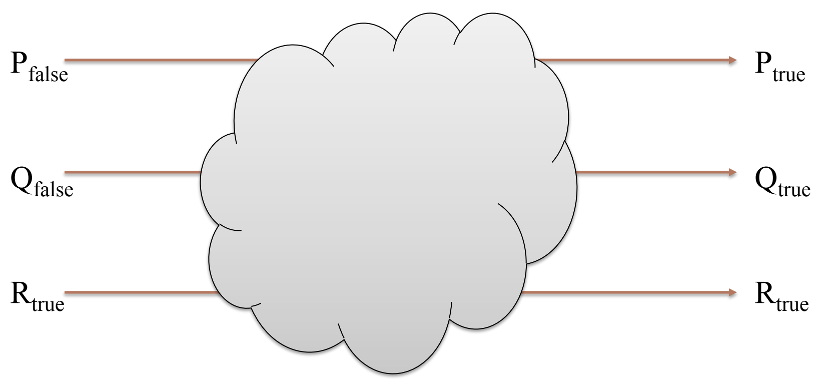
35

**DISTRIBUTED PROBLEMS**

36

# Distributed Problems

- Global Consensus



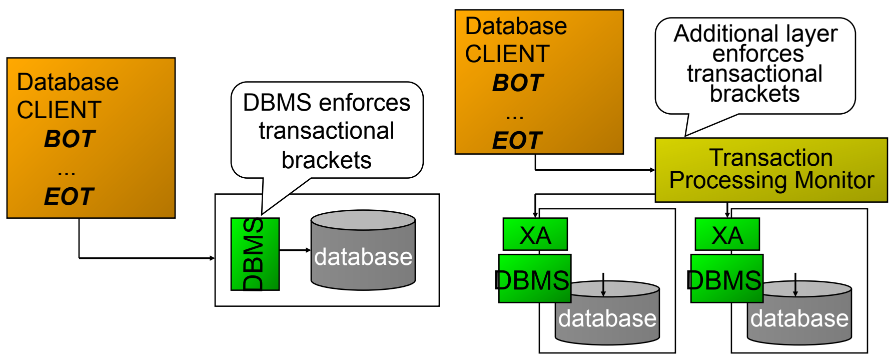



# Distributed Problems

- 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

## Distributed Problems

- *Non-blocking* atomic commitment




39

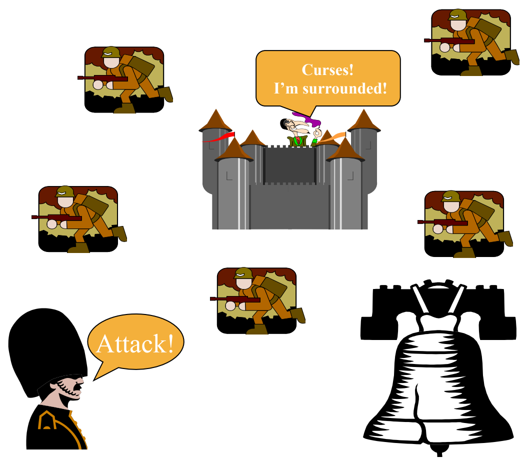
## Distributed Problems

- *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

40

## Distributed Problems

- Byzantine Agreement ("Byzantine Generals")






## 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

## Other Problems

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

43

## 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

44

## CONSENSUS AND FAILURE DETECTION

45

## 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

46

## 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*

47

## 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

48

# Properties of Failure Detectors

- Completeness: detection of every crash
  - **Strong completeness:** Eventually, every process that crashes is permanently suspected by every correct process
  - **Weak completeness:** Eventually, every process that crashes is permanently suspected by some correct process



# Properties of Failure Detectors

- Accuracy: does it make mistakes?
  - **Strong accuracy:** No process suspected before it crashes.
  - **Weak accuracy:** Some correct process is never suspected
  - **Eventual strong accuracy:** there is a time after which correct processes are not suspected by any correct process
  - **Eventual weak accuracy:** there is a time after which some correct process is not suspected by any correct process

# A sampling of failure detectors

| Completeness | Accuracy | | | |
|---|---|---|---|---|
| | Strong | Weak | Eventually Strong | Eventually Weak |
| Strong | *Perfect* P | *Strong* S | *Eventually Perfect* ◊P | *Eventually Strong* ◊ S |
| Weak | D | *Weak* W | ◊ D | *Eventually Weak* ◊ W |



# Perfect Detector

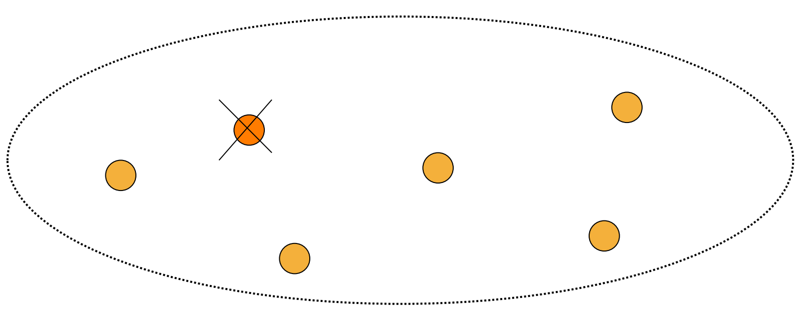
- Named *Perfect,* written *P*
- Strong completeness and strong accuracy
- Immediately detects all failures
- Never makes mistakes

# Eventually Weak Detector

- Eventually Weak: ◊W: “diamond-W”
- Weak Completeness: There is a time after which every process that crashes is suspected by *some* correct process
  - If it crashes, “we eventually, accurately detect the crash”
- Eventually Weak Accuracy: There is a time after which *some* correct process is never suspected by any correct process
  - Think: “we can eventually agree upon a leader.”
  - Failure detectors are unreliable, but mistakes are recognized

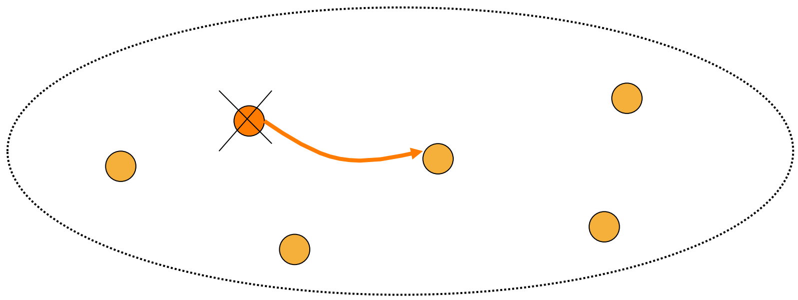
53

# From Weak Completeness to Strong Completeness



54

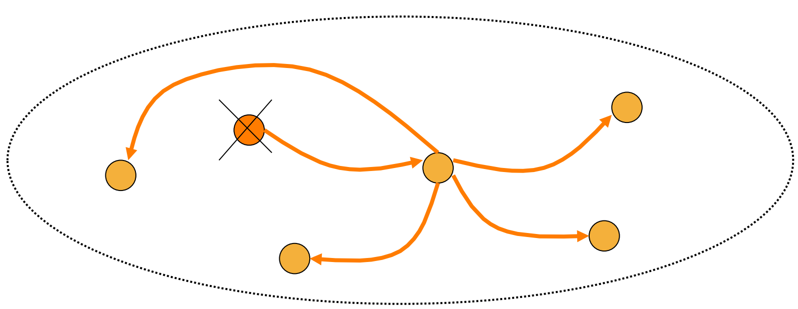
## From Weak Completeness to Strong Completeness



**Weak** Completeness: Failed node is detected by **some** correct process

55

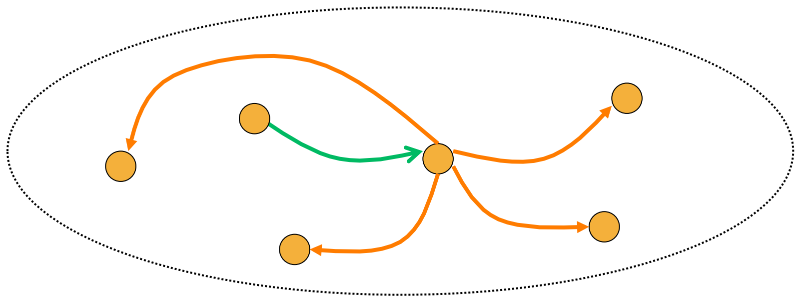
## From Weak Completeness to Strong Completeness



**Strong** Completeness: Initial detector notifies the other correct nodes

56

## From Weak Completeness to Strong Completeness

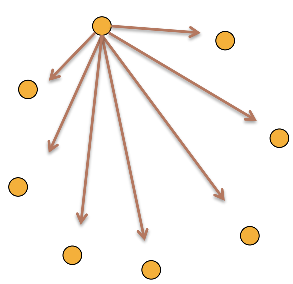


**Accuracy**: “Failed” node eventually notifies correct processes of their mistake

57

## Consensus with Eventually Strong Detector

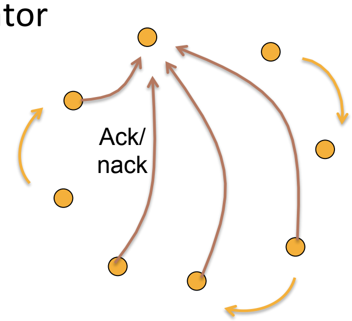
- Round i (repeat until final value):
  - Coordinator is process (i mod N)
  - Broadcast to all processes for their value
  - Wait for majority to respond (assume < N/2 fails)



58

## Consensus with Eventually Strong Detector

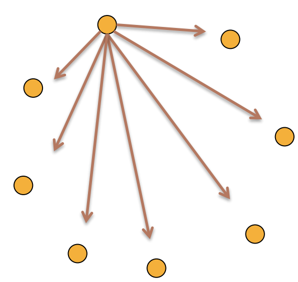
- Round i:
  - Each correct process may ack with its value…
  - …or believe coordinator has failed, i += 1
  - …must still send nack for termination of coordinator



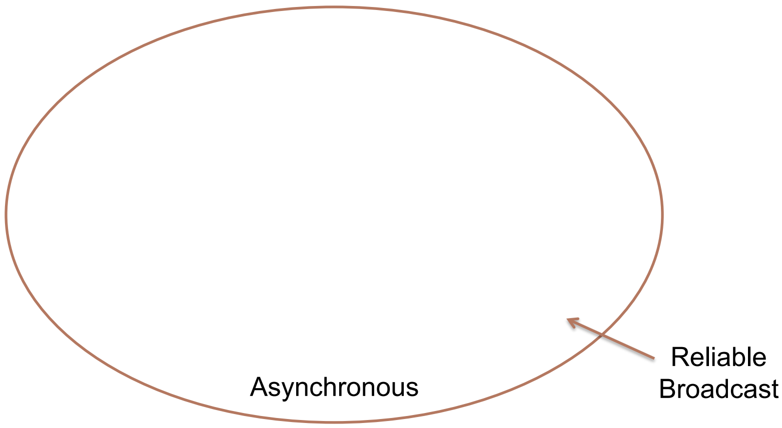



## Consensus with Eventually Strong Detector

- Termination:
  - Eventual weak accuracy: *Some* coordinator will *eventually* be seen correct by all correct processes
  - With majority vote, broadcast final value

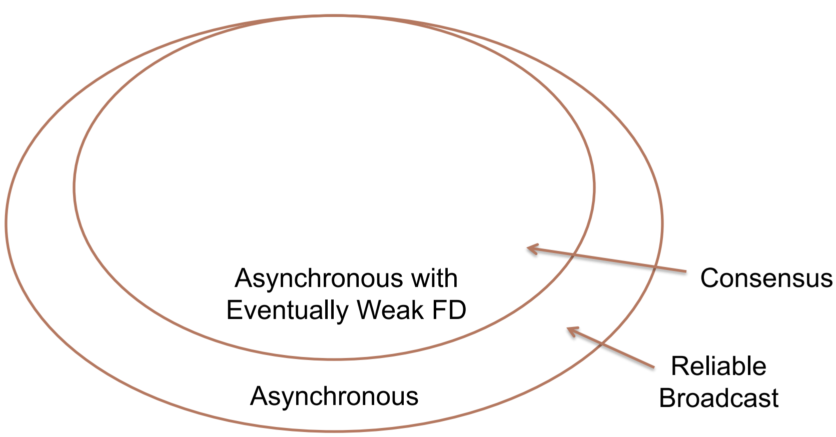


# Power of Failure Detectors



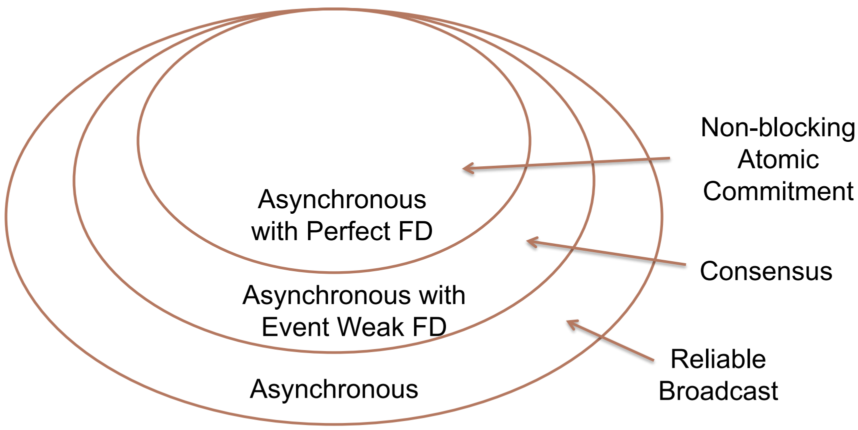

61

# Power of Failure Detectors



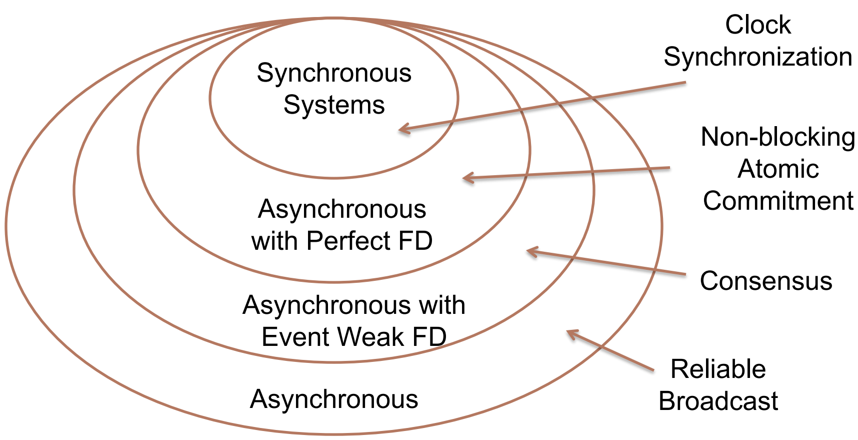

62

## Power of Failure Detectors




63

## Power of Failure Detectors




64

## How to Proceed?

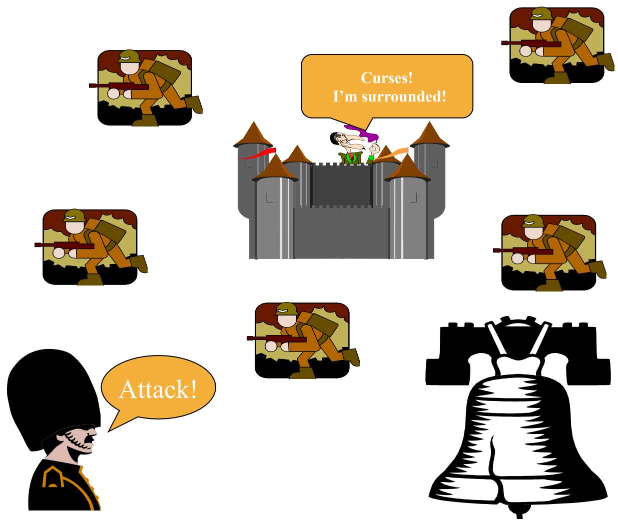
- Approximate ◊W with sufficiently long timeouts
  - Problem: latency
- Use probabilistic protocols
  - Solve consensus with high probability
- Change problem e.g. to group membership
  - Process group approach, false positives ok
- Accept consensus protocol that terminates with high probability
  - Paxos algorithm

65

## BYZANTINE AGREEMENT

66

# Byzantine Agreement






# Byzantine Agreement

- Suppose 3 generals (A,B,C), one of whom may be traitor
- General A knows he's loyal
- Take majority vote?
- But traitor may be saying different things to A and other loyal general
- Lower bound: Need at least 4 generals if 1 traitor
- Generally: Need 3f+1 processors if f are faulty

## Byzantine Agreement

- Assume wlog general sending orders to lieutenants
- Give commanders ability to sign their messages
- Assume no more than f failures, and f+2 commanders



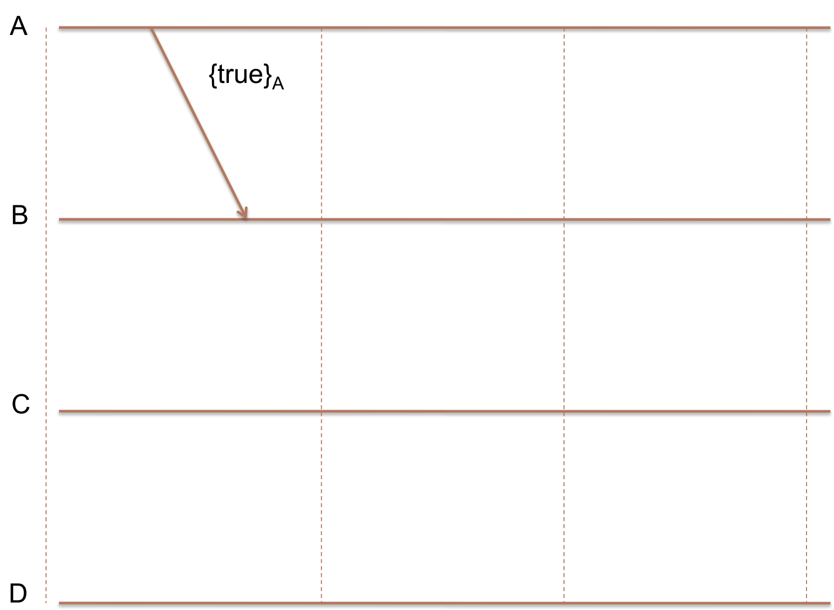
## Protocol

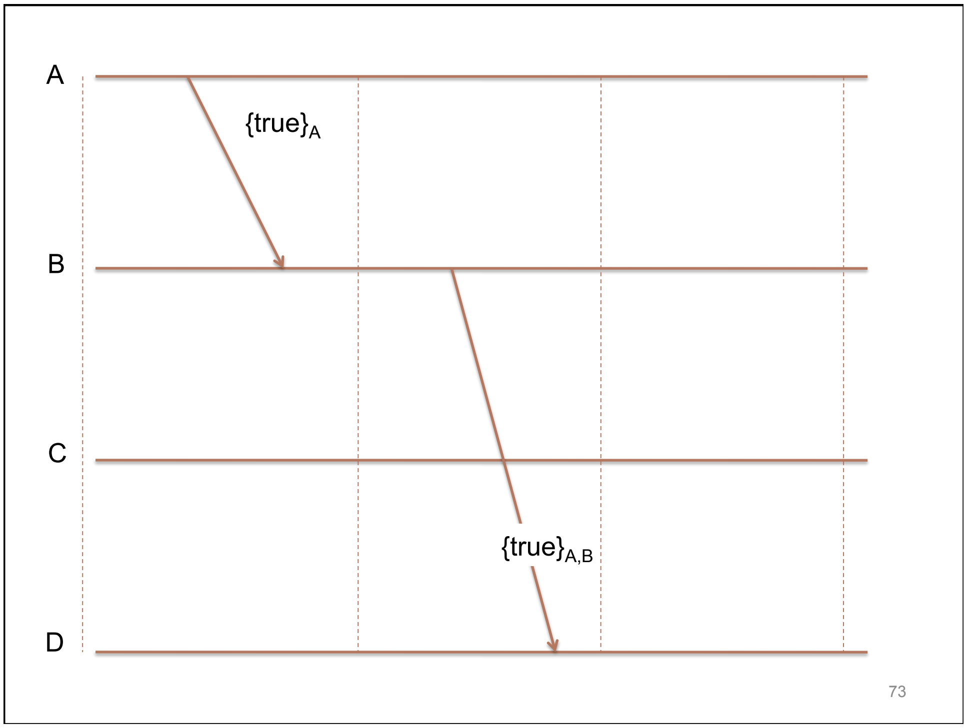
- Round 1:
  - General broadcasts his order (true or false) to all lieutenants
- Round *i*, for loyal commander:
  - Consider any *messages* with i-1 signatures received in previous round
  - Record any *orders* signed by the general
  - Commander adds his signature to each *message*, and broadcasts result to all other processes
  - Repeat this round f times for total f+1 rounds

# Protocol

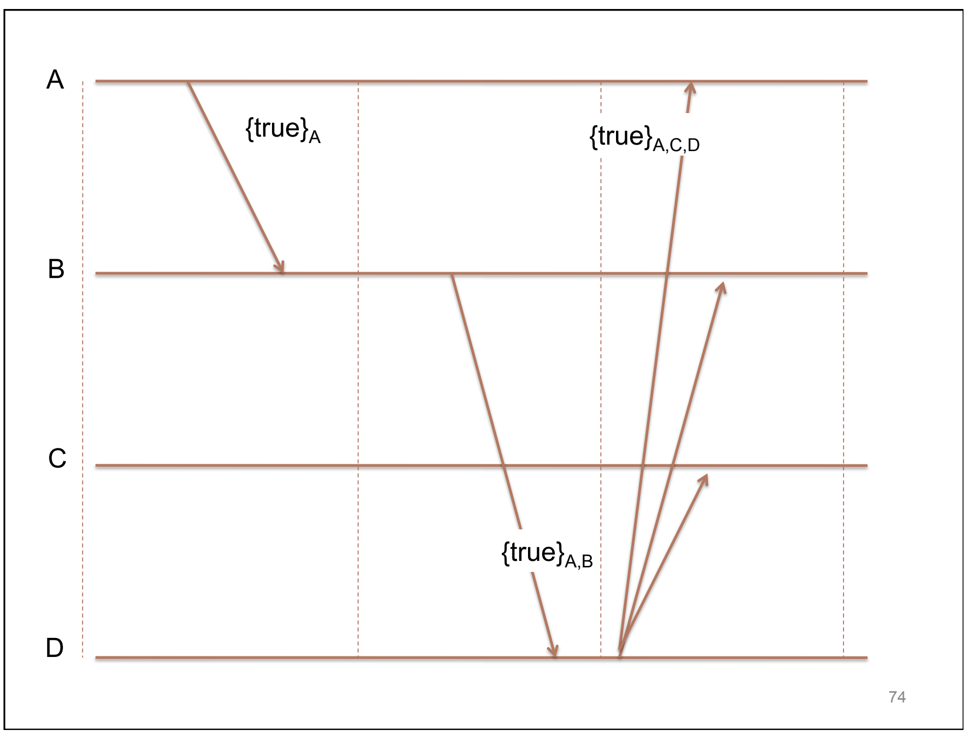
- After f+1 rounds, each loyal commander considers the orders he has recorded:
  - If empty, or conflicting orders, then choose default decision
  - If exactly one order, then execute that order



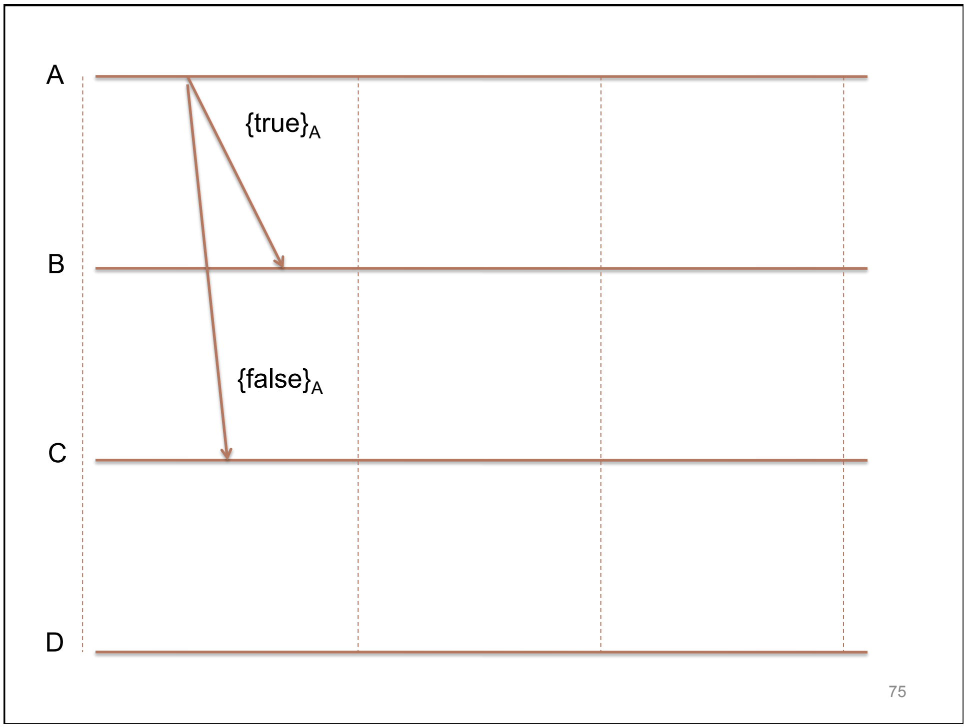


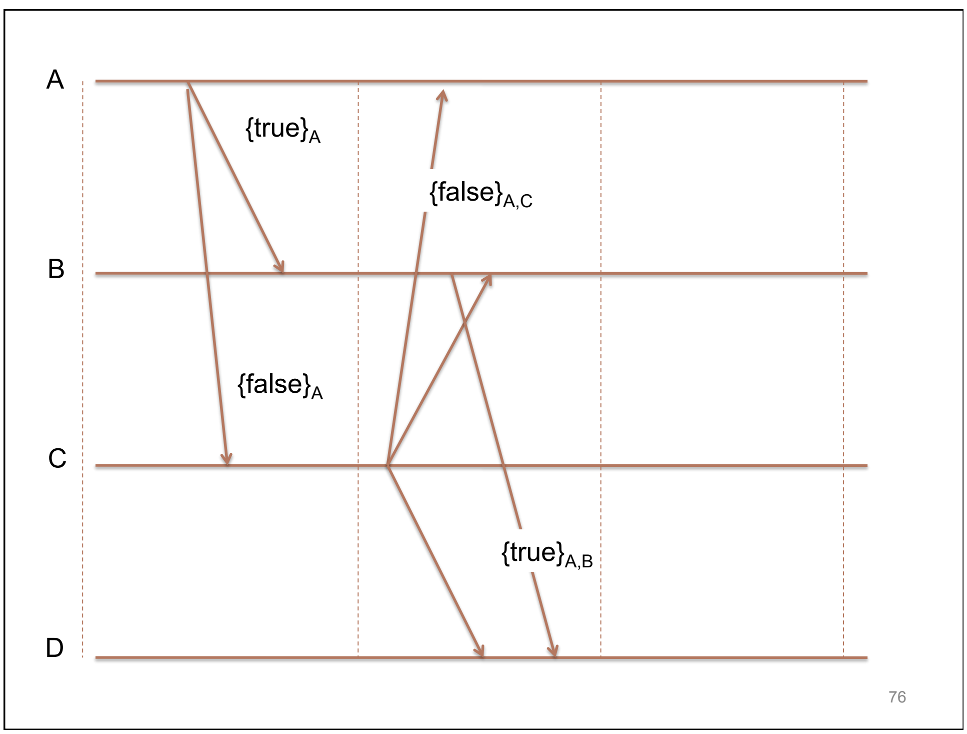


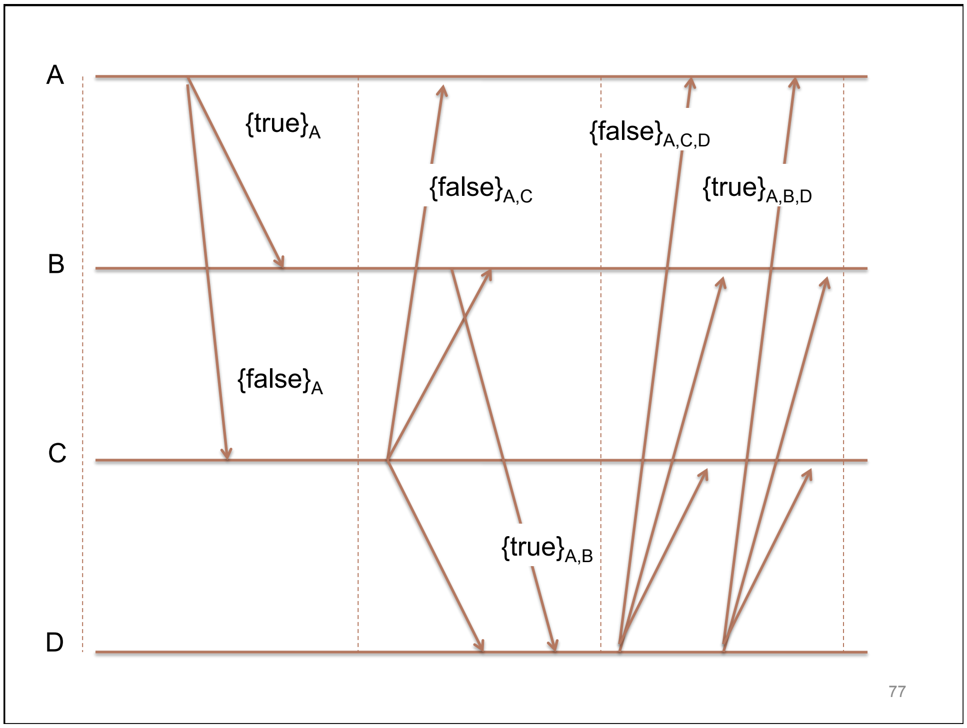
A
B
C
D
{true}$_A$
{true}$_{A,B}$
73

A
B
C
D
{true}$_A$
{true}$_{A,C,D}$
{true}$_{A,B}$
74


A
B
C
D
{true}A
{false}A
75


A
B
C
D
{true}A
{false}A,C
{false}A
{true}A,B
76




# Why does this work?

- Suppose general is loyal
  - Broadcasts order in first round
  - But lieutenants do not know if he is loyal
  - Therefore run for f more rounds
- Disloyal general would:
  - Relay conflicting orders via disloyal lieutenants
  - Orders delivered to loyal lieutenants in last round
  - But protocol requires f+1 rounds, f+1 signatures
  - So orders relayed through at least one loyal lieutenant

78

## Observations

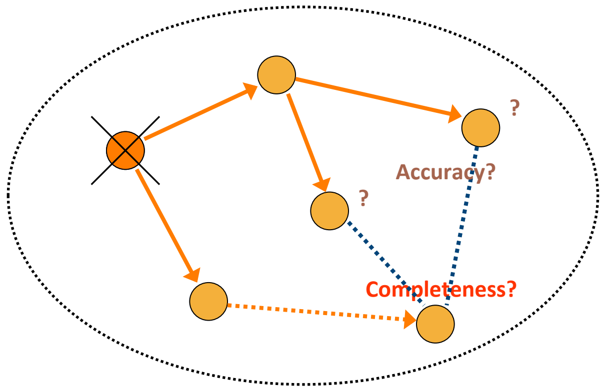
- Complexity of protocol:
  - $O(N^2)$ messages on each round!
  - All Byzantine protocols are expensive
- Rabin: randomized protocols
  - Each process has a form of coin available to it
  - Can flip coin in each round
  - With randomness, very rapid agreement "with high probability" in very little time

79

## GROUP MEMBERSHIP (1/3)

80

# Failure Detection




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]

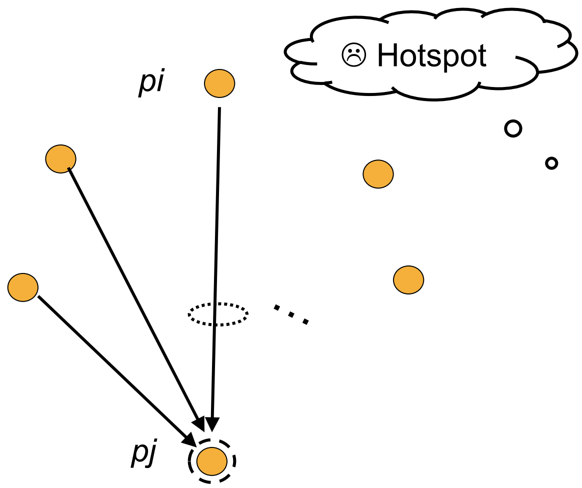
81

# How to Proceed?

- Approximate ◊W with sufficiently long timeouts
  - Problem: latency
- Use probabilistic protocols
  - Solve consensus with high probability
- Change problem e.g. to group membership
  - Process group approach
- Accept consensus protocol that terminates with high probability
  - Paxos algorithm

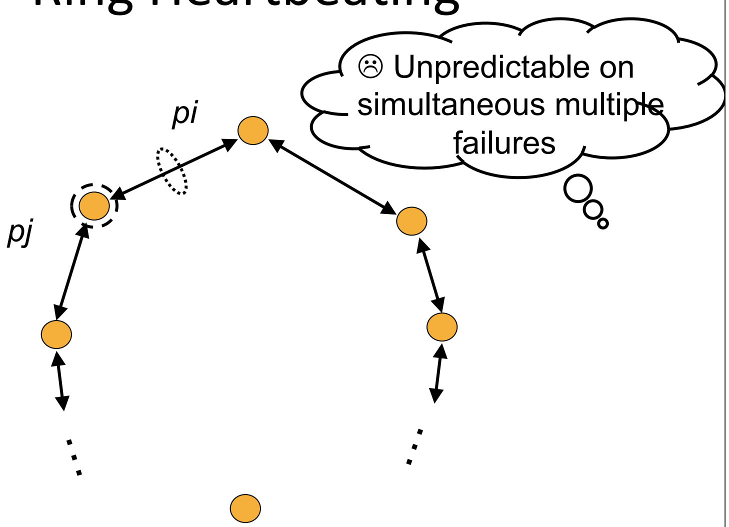
82

# Centralized Heartbeating



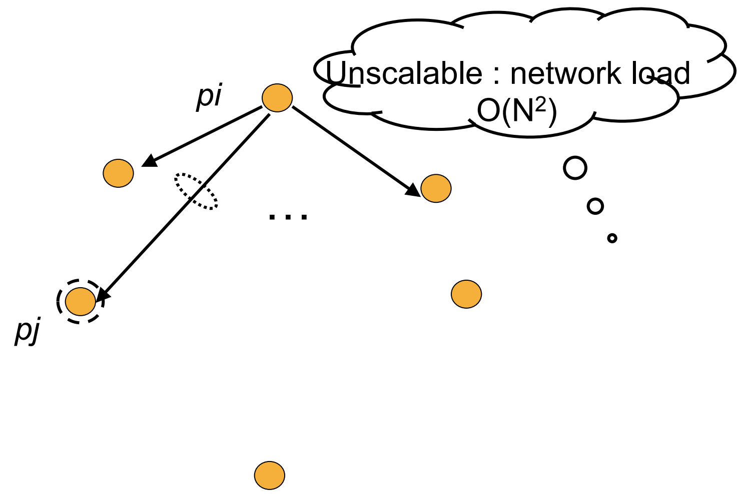

83

# Ring Heartbeating




84

## All-to-All Heartbeating



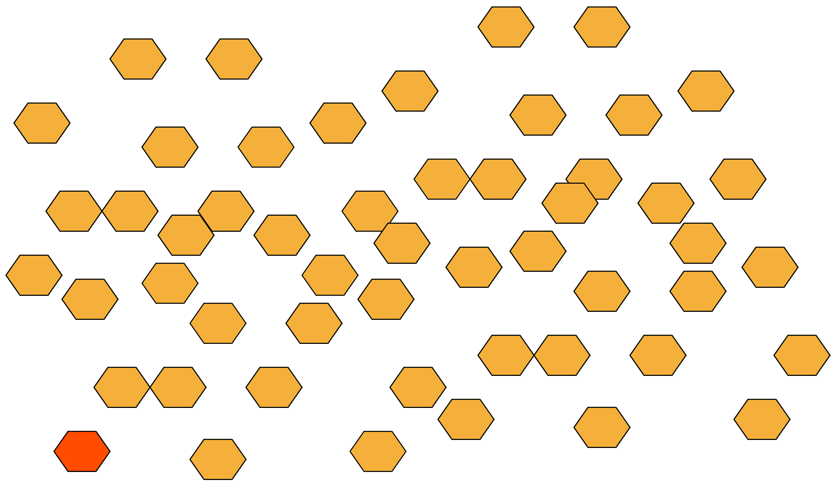



## Gossip "epidemics"

- [t=0] Suppose that I know something
- [t=1] I pick you… Now two of us know it.
- [t=2] We each pick … now 4 know it…

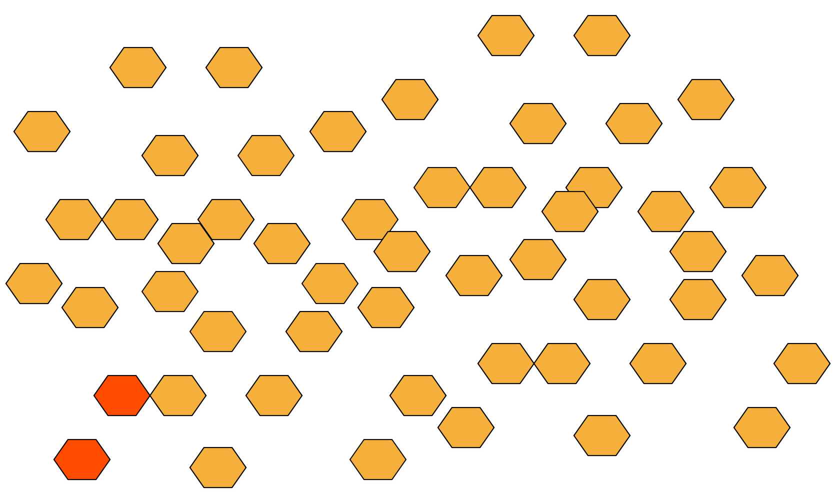
• Information spread: exponential rate.
  - Due to re-infection (gossip to an infected node) spreads as $1.8^k$ after $k$ rounds
  - But in O(log(N)) time, N nodes are infected

## Gossip epidemics

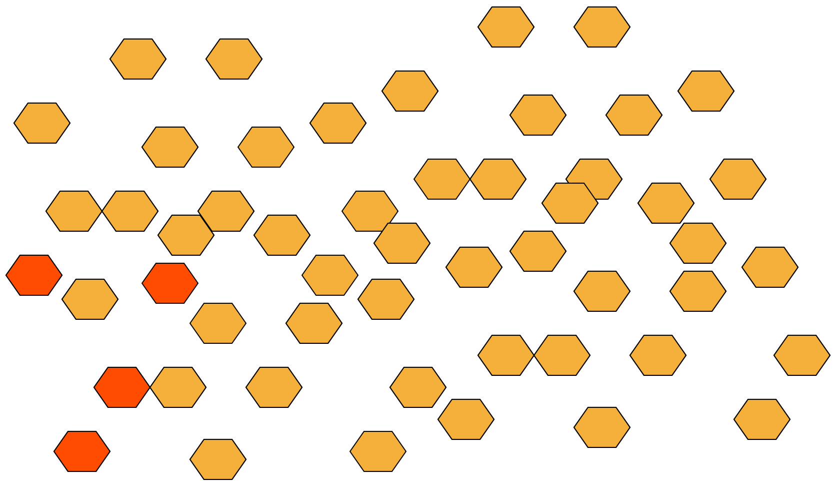




## Gossip epidemics

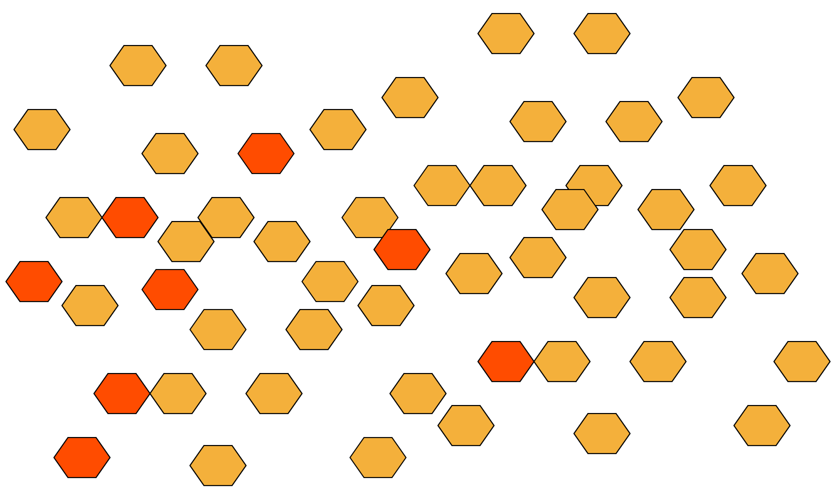


## Gossip epidemics

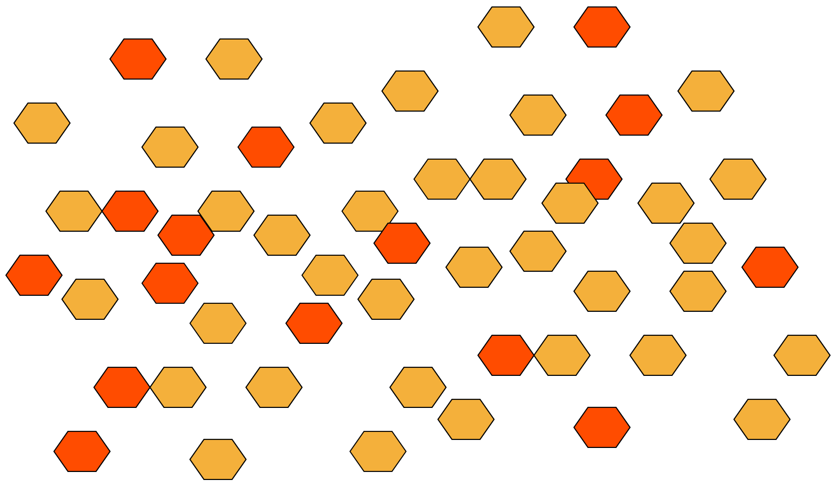




## Gossip epidemics

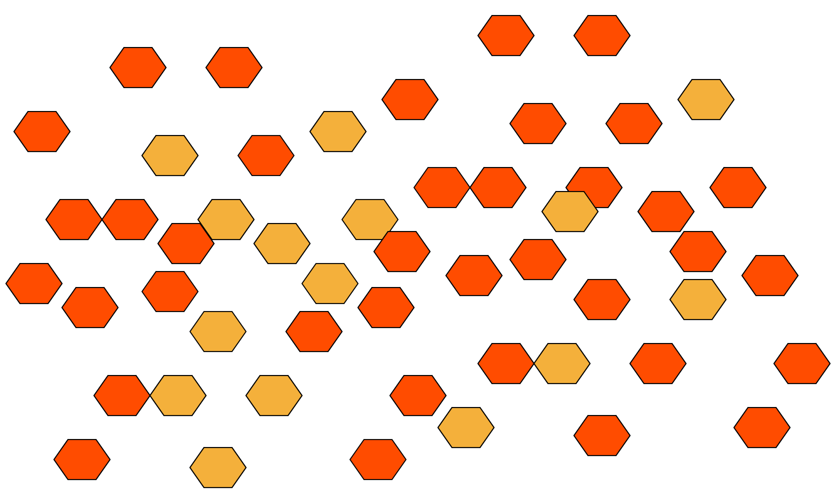


# Gossip epidemics



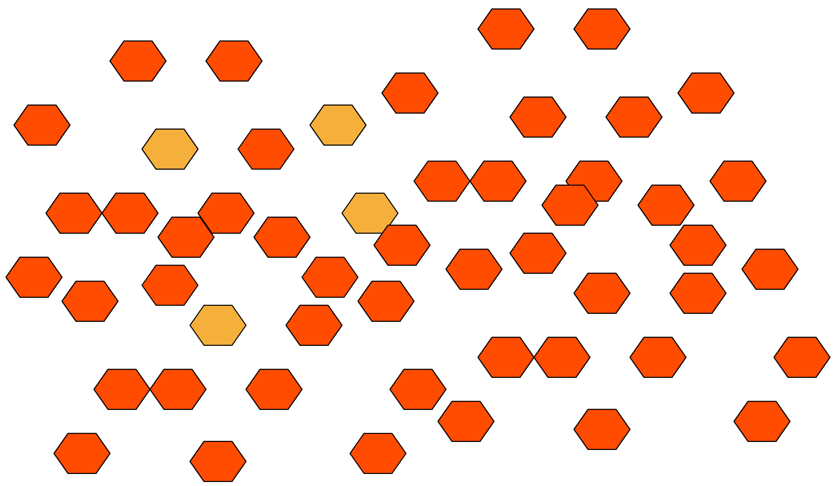
91

# Gossip epidemics



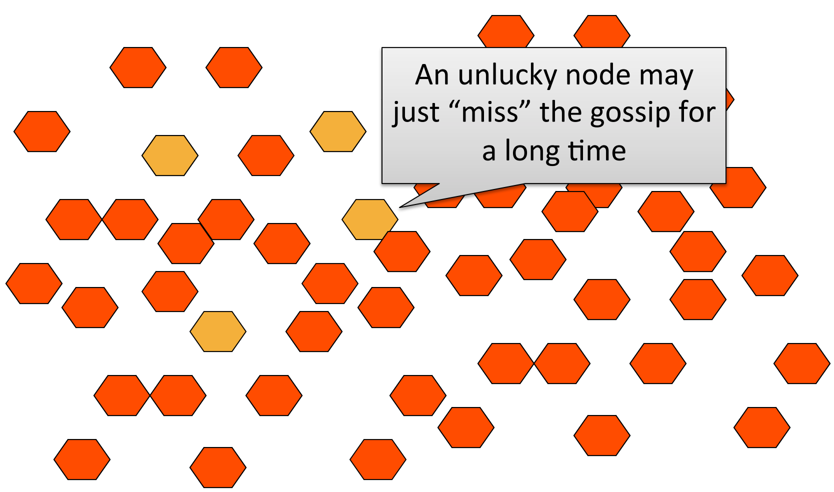
92

## Gossip epidemics



93

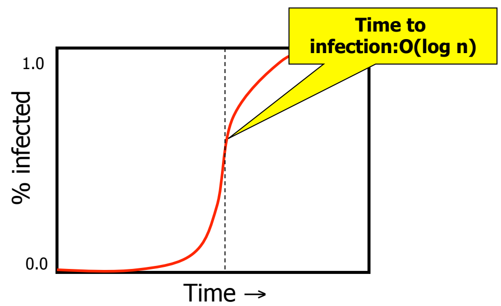
## Gossip epidemics




94

# Gossip: scales nicely

- Participants' loads independent of size
- Network load linear in system size
- Data spreads in log(system size) time




95

# Facts about gossip epidemics

- Extremely robust
  - Data travels on exponentially many paths!
  - Hard to even slow it down…
    - Suppose 50% of packets are lost…
    - … 1 additional round!
  - Push-pull works best.
  - Many optimizations are needed in practice…

96

## GROUP MEMBERSHIP (2/3)

97

## Completeness & Accuracy

- Trivial algorithms
- Completeness :
  - declare all as failed (always)
- Accuracy :
  - declare all as alive (always)

98

## Completeness & Accuracy

- In practice, most applications require
  - Completeness to <u>always</u> be guaranteed
    - *Eventual* consistency absolutely required
  - Accuracy guaranteed <u>most of the time</u> (probabilistically)
    - Performance degradation can be tolerated

99

## Gossip-Based Failure Detection

- Scalable failure detection
  - Detection time : O(N log(N))
  - Network load per node : O(1)
- Detects all faulty nodes within a time bound
  - Time-bounded completeness
- Has a rate of false positives (probabilistic)

100

# Failure Detection Protocol

- System Assumptions
  - No bound on message delivery
    - Most messages delivered in reasonable time
  - Failure model: Crash stop
  - Low clock drift
- Bird's Eye Protocol:
  - each member $M_i$ sends out a heartbeat
  - heartbeat is disseminated using gossip
  - failure detection when time out waiting for $M_i$'s next heartbeat

101

# Basic Protocol

- Each member maintains a list (O(N)) of
  - $< M_i, H_i, T_{last,i} >$
  - $M_i$ : member address
  - $H_i$ : heartbeat count
  - $T_{last,i}$ : last time of heartbeat increase
- Every $T_{gossip}$, each member
  - Increments its heartbeat
  - Selects a random target member (from its list) and sends to it a <u>constant</u> number of $<M_i, H_i>$ entries

102

# Basic Protocol

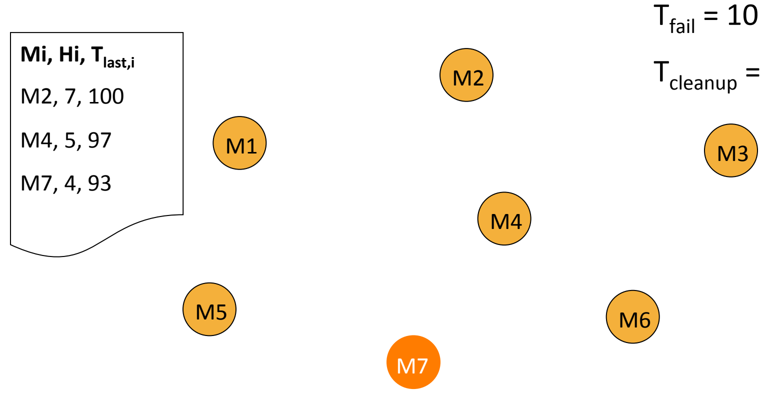
- A member, upon receiving gossip message,
  - Merges the list (maximum heartbeat)
- If $T_{last,i} + T_{fail} < T_{now}$,
  - Member $M_i$ is considered failed
  - But remember $M_i$ for $T_{cleanup}$ (~ $2*T_{fail}$), to prevent <u>resurrection</u>

103

# Basic Protocol

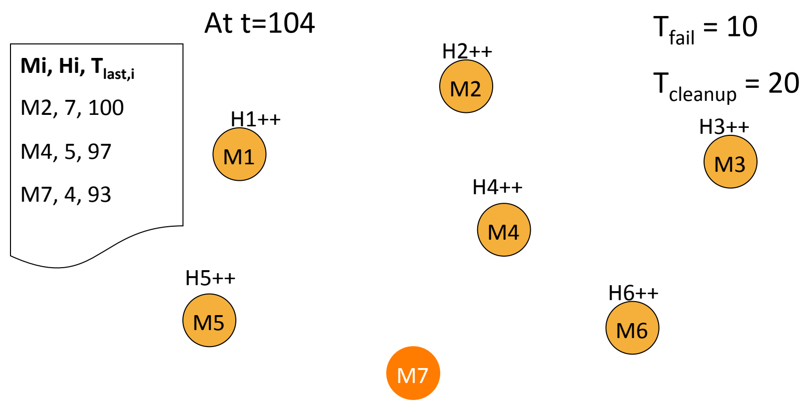
$T_{fail} = 10$

$T_{cleanup} = 20$



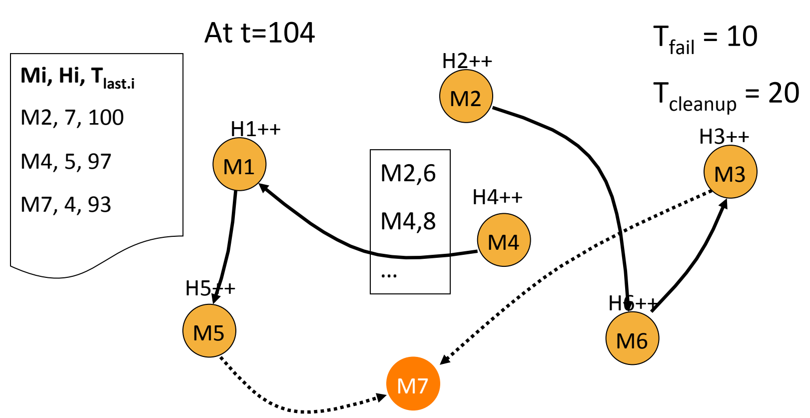

104

# Basic Protocol



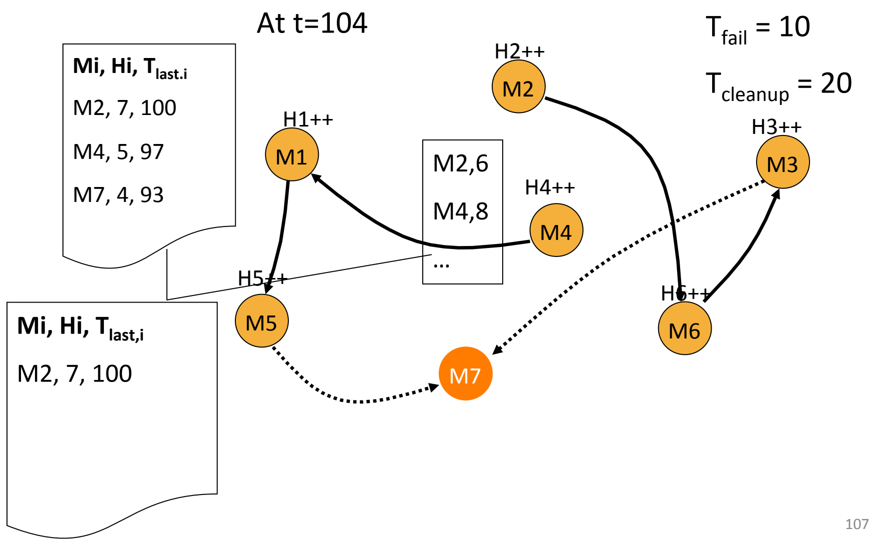

105

# Basic Protocol



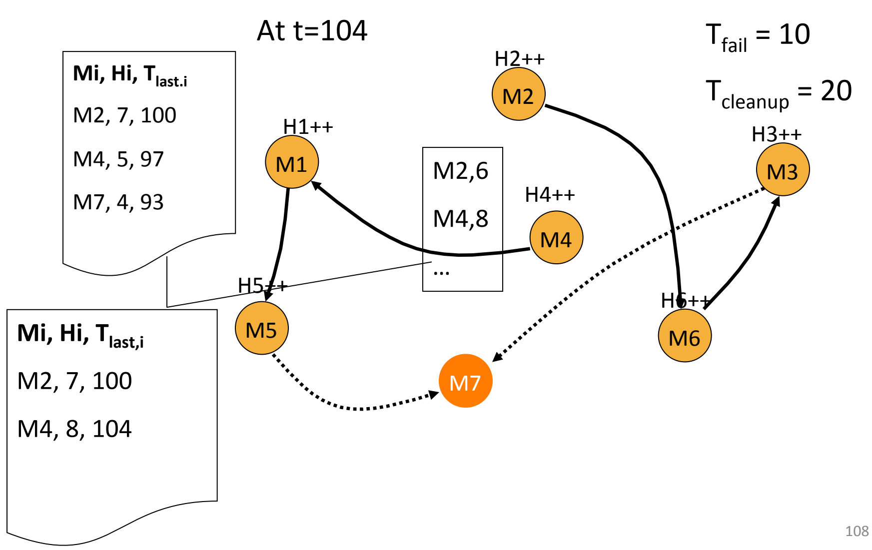

106

# Basic Protocol



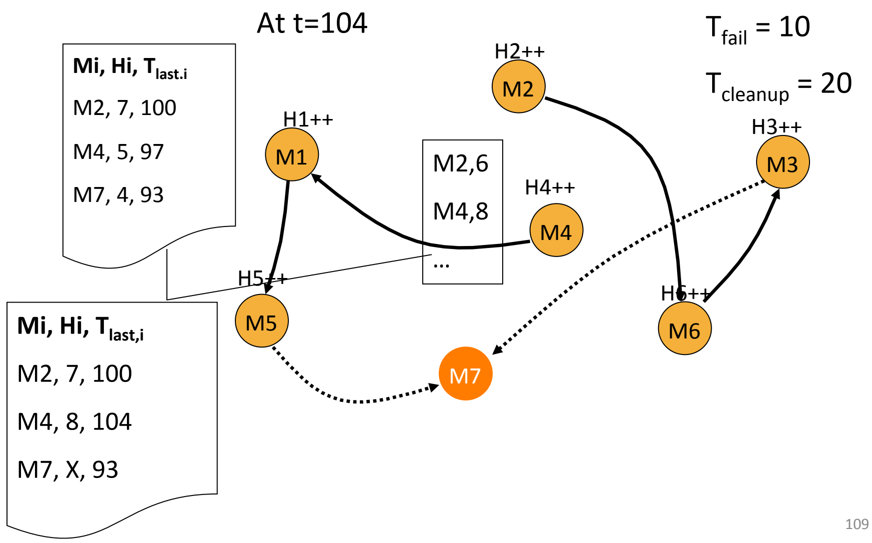

107

# Basic Protocol




108

# Basic Protocol




109

# Analysis

- Detection Time = time to spread a gossip in a group of N nodes
  - O(log(N)) for a single gossip
  - But N such gossips being multicast
    - one heartbeat from each node
  - Since the actual message can carry only a constant number of heartbeats, the total dissemination is O(N log(N)).

110

## Summary

| | |
|---|---|
| Completeness | Eventual detection<br>Expected detection time with known mistake |
| Accuracy | Probabilistic |
| Speed | Detection time : O(N log(N)) |
| Scalability | Detection time : O(N log(N))<br>Network load : O(N)<br>Per node overhead : O(1) |
| Resilience | Basic : resilient to message loss, # of failures<br>Hierarchical : resilient to network partitions, large # of failures |

111

# GROUP MEMBERSHIP (3/3)

112

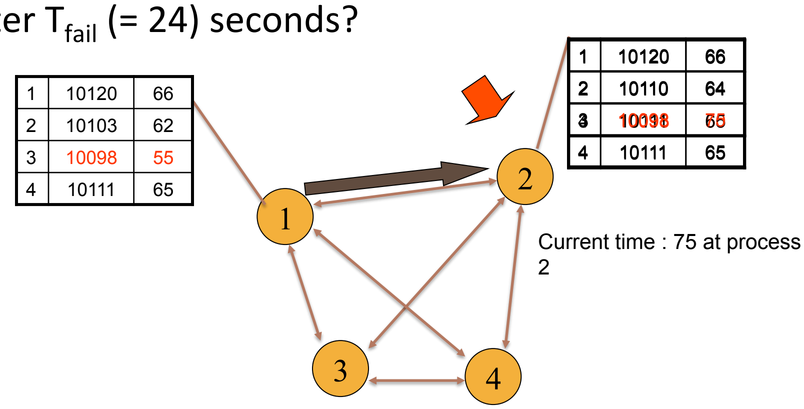
# Gossip Protocol

- A member, upon receiving gossip message,
  - Merges the list (maximum heartbeat)
- If $T_{last,i} + T_{fail} < T_{now}$,
  - Member $M_i$ is considered failed
  - But remember $M_i$ for $T_{cleanup}$ (~ $2*T_{fail}$), to prevent <u>resurrection</u>

113

# Gossip Protocol

- What if an entry for failed process is deleted right after $T_{fail}$ (= 24) seconds?




- Fix: remember for another $T_{fail}$

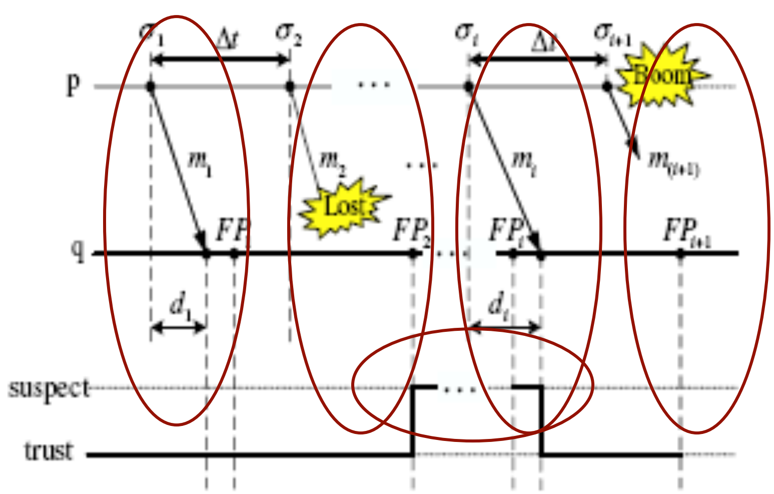
114

# Suspicion Mechanism

- Goal: Reduce the frequency of false positives that might occur due to:
  - Network packet losses
  - Slow and unresponsive processes
- Key:
  - When a process is first detected as having failed, do not declare it as having failed
  - Instead, suspect the process first
  - Allow time to fix mistake



# Suspicion Mechanism



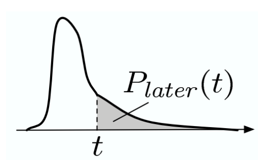
# Accrual Failure Detector

- Accrual Failure Detector
  - $\varphi(t)$: suspicion level at time t (for a node)
- Application sets a max suspicion level
  - Node declared failed otherwise
- Example: Cassandra/Dynamo
  - Set $\varphi(t) = 5 \Rightarrow$ 10-15 sec detection time
- Calculate $\varphi(t)$
  - Consider historical inter-arrival time of heartbeats

117

# Accrual Failure Detector

- $\varphi(t)$: suspicion level at time t



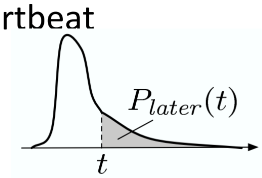

- $P_{later}(t)$: probability of heartbeat after $t$ seconds
- $P_{later}(t_{now} - t_{last})$: probability after "*now*"

- $\varphi(t) = -\log_{10}(P_{later}(t_{now} - t_{last}))$
  - Threshold = 1 $\Rightarrow$ 10% chance of mistake
  - Threshold =2 $\Rightarrow$ 1% chance of mistake
  - Threshold = 3 $\Rightarrow$ 0.1% chance of mistake

118

# Accrual Failure Detector

- φ(t): suspicion level at time t
- $P_{later}(t)$: probability of heartbeat after *t* secs
  - Based on sampling of previous heartbeat timestamps
  - Assume normally distributed

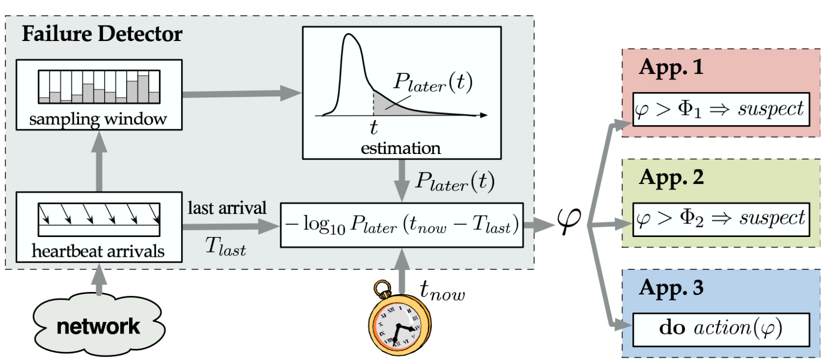



$$\varphi(t_{now}) = -log_{10}(P_{later}(t_{now} - T_{last}))$$

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_t^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1 - F(t)$$

119

# Implementing Failure Detector




120