

TDDDD07 Real-time Systems

Lecture 7

Dependability & Fault tolerance

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and Information Science
Linköping University



Dependability and real-time

- If a system is to produce results within time constraints, it needs to produce results at all!
- Dependable computer systems *justify* and *measure* how well systems meet their requirements *in presence of faults*.

Predictability & faults?

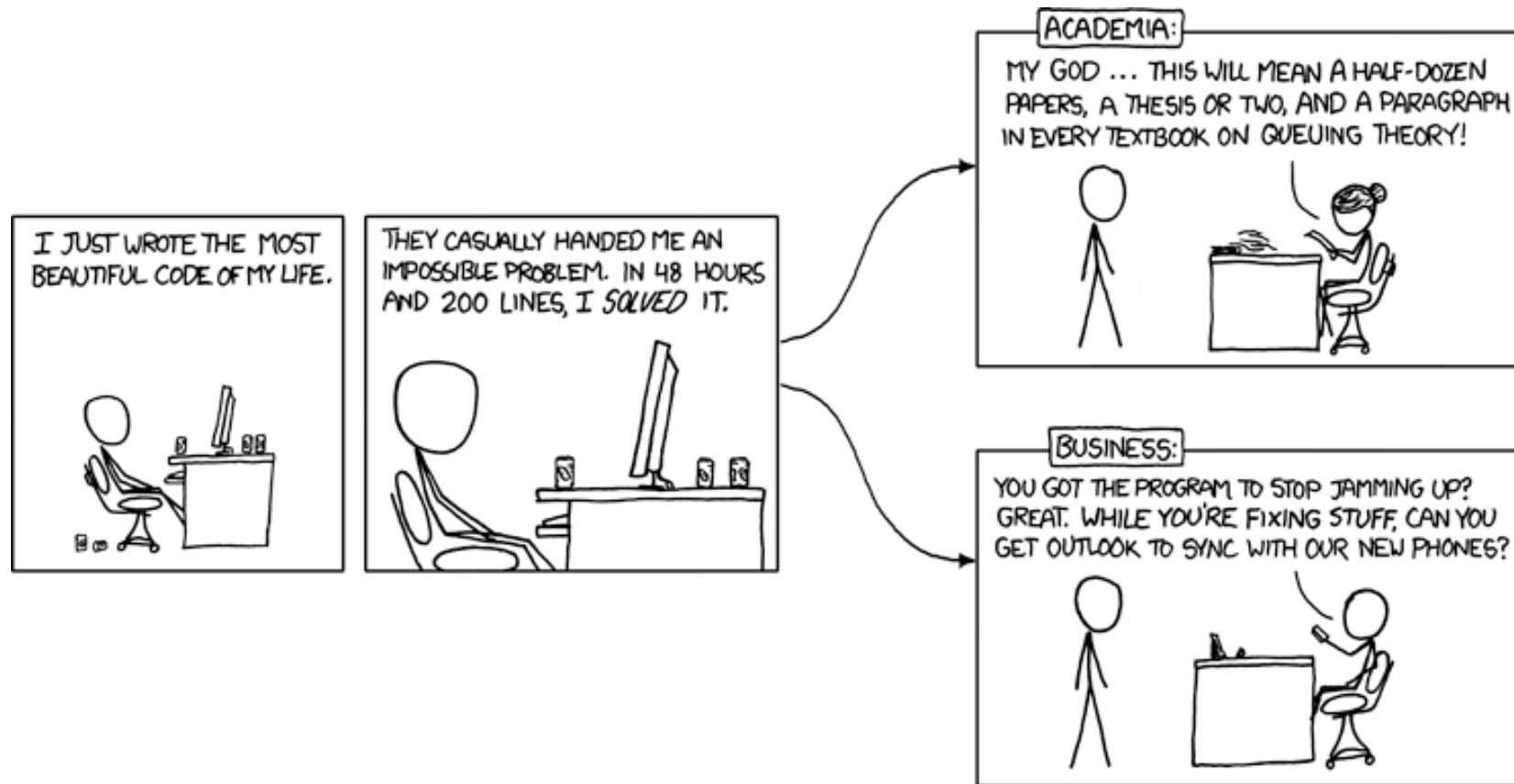
The film...

Where to start?

- How do things go wrong and why?
- What can we do about it?



Building dependable systems



http://imgs.xkcd.com/comics/academia_vs_business.png



The next three lectures

This lecture and part of lecture 9 covers theory

- Basic notions of dependability and redundancy in fault-tolerant systems
- Fault tolerance:
 - Relating faults/redundancy to distributed systems from lectures 4-6
 - Relating timing and fault tolerance

Lecture 8: Industrial perspective

Lecture 9: Fault prevention and design aspects

February 2, 2016

- 32 year old Kaushal Gandhi driving a Skoda Octavia on the M40 motorway, found that his cruise control was stuck at over 110 mph. His conversation in the emergency call for 8.5 minutes recorded the chain of events before a fatal accident where the car crashed into a parking lorry.

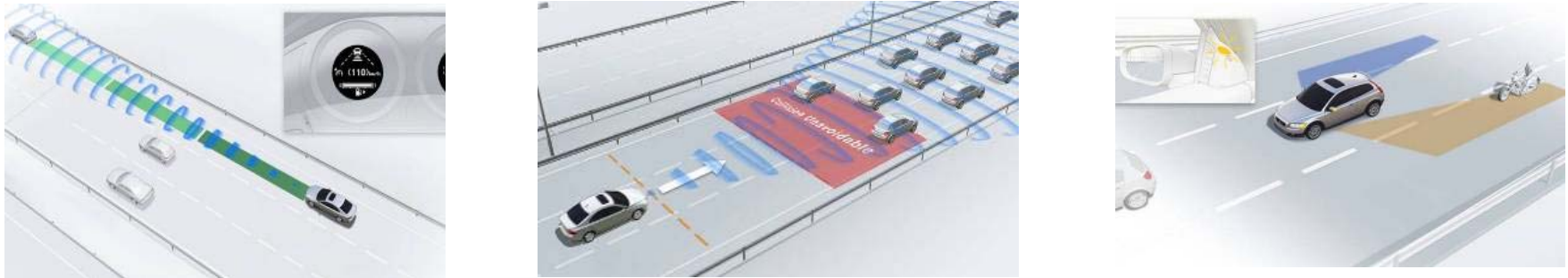
<https://www.theguardian.com/business/2016/nov/24/skoda-driver-decapitated-in-stuck-cruise-control-mystery>

- “Automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly.”

Wired 2005-11-08

- The problem was found to be an embedded software bug

Trends: software in cars



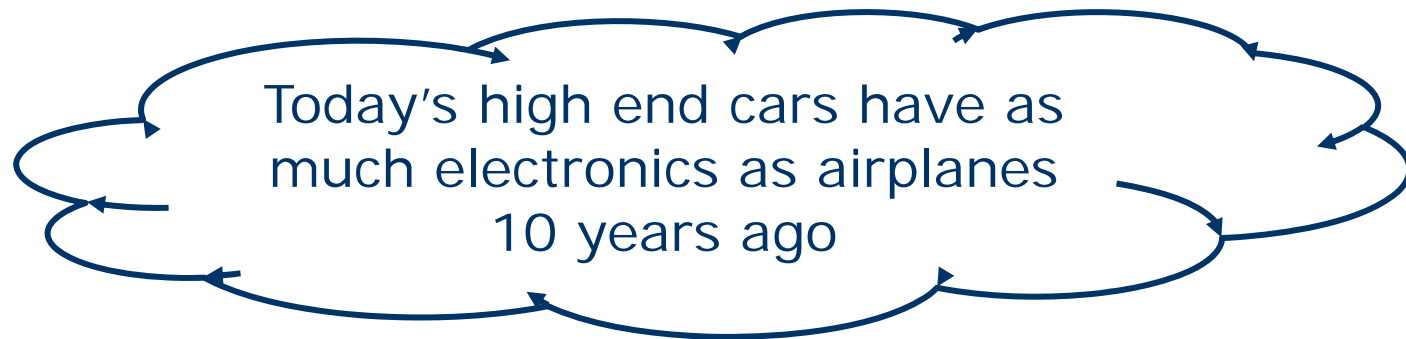
1984	ABS Anti-lock Braking System	2004	Blind Spot Information system (BLIS)
1998	Dynamic Stability and Traction Control (DSTC)	2006	Active Bi-Xenon lights
2002	Roll Stability Control (RSC)	2006	Adaptive Cruise Control (ACC)
2003	Intelligent Driver Information System (IDIS)	2006	Collision warning system with brake support

Source: Volvo Cars

October 24, 2013

- Toyota in trouble ...

www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences



- Volvo recalls 59,000 cars over software fault
- The glitch can shut down the engine and electrical system while the car is in motion
- Stefan Elfström told AP:
 - they would then both restart immediately

Early space and avionics

- During 1955, 18 air carrier accidents in the USA (when only 20% of the public was willing to fly!)
- Today's complexity many times higher

Airbus 380

- Integrated modular avionics (IMA), with safety-critical digital components, e.g.
 - Power-by-wire: complementing the hydraulic powered flight control surfaces
 - Cabin pressure control (implemented with a TTP operated bus)



What is dependability?

Property of a **computing system** which allows reliance to be *justifiably* placed on the service it delivers.

[Avizienis et al. 2004]

The ability to avoid service failures that are *more frequent* or *more severe* than is acceptable.

(sv. Pålitliga datorsystem)

Attributes of dependability

IFIP WG 10.4 definitions:

- **Safety**: absence of harm to people and environment
- **Availability**: the readiness for correct service
- **Integrity**: absence of improper system alterations
- **Reliability**: continuity of correct service
- **Maintainability**: ability to undergo modifications and repairs

[sv. Tillförlitlighet]

- Means that the system (functionally) behaves as specified, and does it *continually* over measured intervals of time
- Measured through relating to probability of failure, e.g. 10^{-9}
- Another way of putting it: MTTF
 - In commercial flight systems - One failure in 10^9 flight hours

Safety and security interplay

- Today we have software embedded in many networked systems that control critical infrastructures
- Water, electricity, transport are indeed safety-critical
- But also subject to severe security threats...

Faults, Errors & Failures

- **Fault:** a defect within the system or a situation that can lead to failure
- **Error:** manifestation (symptom) of the fault - an unexpected behaviour
- **Failure:** system not performing its intended function



Examples

- Year 2000 bug
- Bit flips in hardware due to cosmic radiation in space
- Loose wire
- Air craft retracting its landing gear while on ground

Effects in time:

Transient/ Intermittent / Permanent


Dependability techniques

Four approaches
[IFIP 10.4]:

1. Fault prevention
2. Fault removal
3. Fault tolerance
4. Fault forecasting



Next
lecture



Let's look at
an example!

Google's 100 min outage

September 2, 2009:

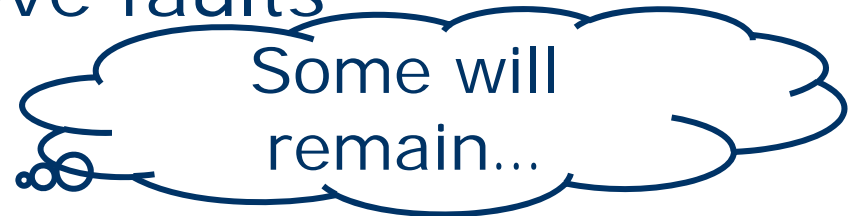
A small fraction of Gmail's servers were taken offline to perform routine upgrades.

"We had slightly underestimated the load which some recent changes (ironically, some designed to improve service availability) placed on the request routers."



Fault \Rightarrow Error \Rightarrow Failure

- Goal of system verification and validation is to remove faults



- Goal of hazard/risk analysis is to focus on more important faults
- Goal of fault tolerance is to reduce effects of errors if they appear - *eliminate or delay failures*

Dependability techniques

Four approaches [IFIP 10.4]:

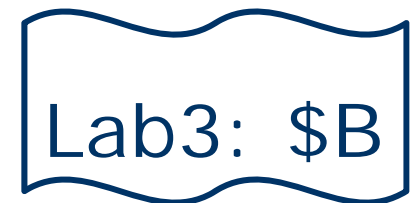
1. Fault prevention
2. Fault removal
3. Fault tolerance
4. Fault forecasting

Fault tolerance

- Means that a system provides a degraded (but acceptable) function
 - Even in presence of faults
 - During a period defined by certain **fault model** (i.e. assumptions)
- Foreseen or unforeseen?
 - Fault model describes the foreseen faults

Fault models

- Leading to **Node** failures
 - Crash
 - Omission
 - Timing
 - Byzantine
- Distributed systems: can also have **Channel** failures
 - Crash (and potential partitions)
 - Message loss
 - Message delay
 - Erroneous/arbitrary messages



On-line error management

- Detection: By program or its environment
- Mitigation:
 - Fault containment by architectural choices
 - Fault tolerance using redundancy
 - in software (redundancy in space or time)
 - in hardware
 - in data

Redundancy

From D. Lardner: Edinburgh Review, year 1824:

”The most certain and effectual check upon errors which arise in the process of computation is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if their computations are carried out by different methods.”*

** people who compute*

Static Redundancy

Used all the time (whether an error has appeared or not), just in case...

- SW: N-version programming
- HW: Voting systems
- Data: Parity bits, checksums

Dynamic Redundancy

Used when error appears and specifically aids the treatment

- SW:
 - Space: Exceptions, Rollback recovery
 - Time: Re-computing a result
- HW: Switching to back-up module
- Data: Self-correcting codes

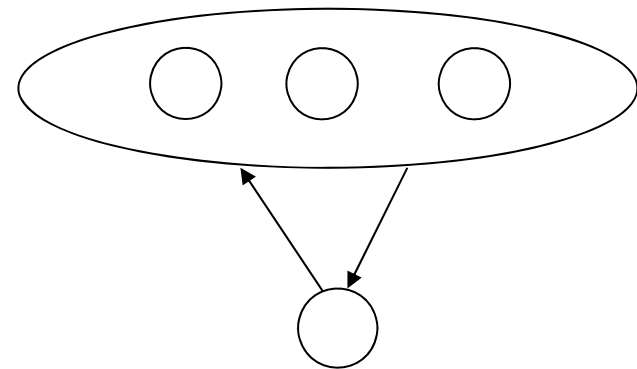
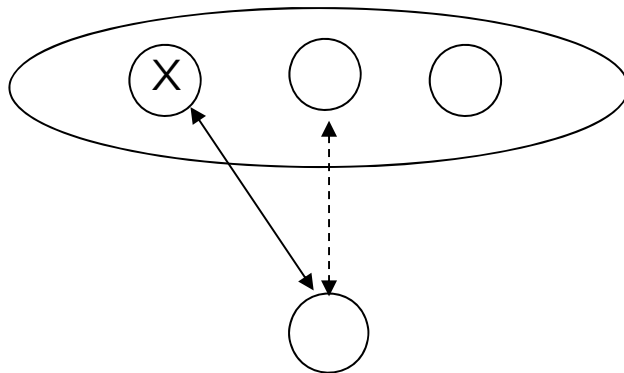
App crashes

- May 2014: Study of 1800 Android apps showed that 19% of crashes were caused by not handling exceptions

<http://dl.acm.org/citation.cfm?id=2597089>

Server replication models

- Passive replication
- Active replication



: Denotes a replica group



The next three lectures

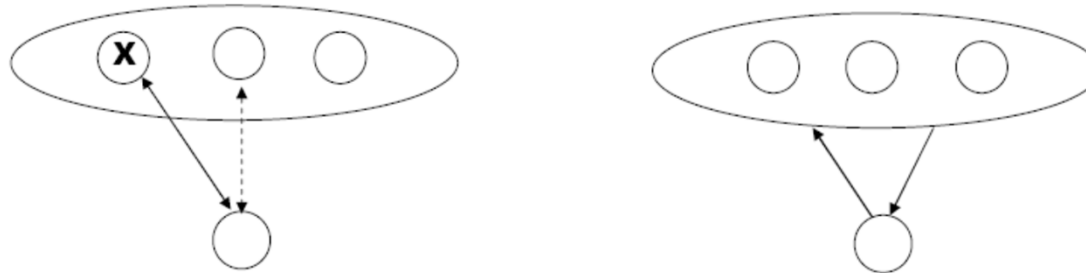
This lecture and part of lecture 9 covers theory

- Basic notions of dependability and redundancy in fault-tolerant systems
- Fault tolerance:
 - Relating faults/redundancy to distributed systems from lectures 4-6
 - Relating timing and fault tolerance

Lecture 8: industrial perspective

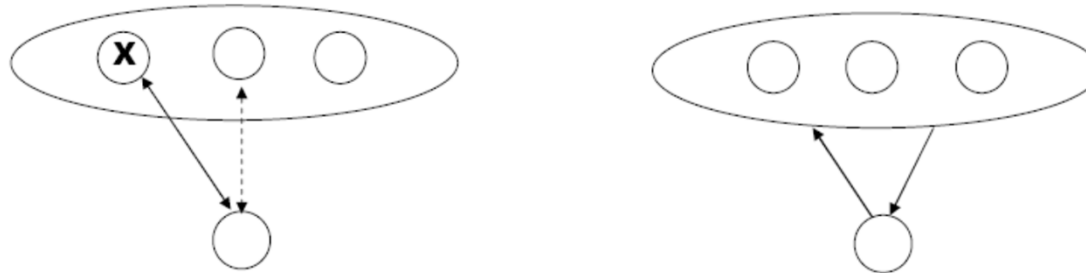
Lecture 9: Fault prevention and design aspects

Underlying mechanisms



- Active replication
 - Replicas must be deterministic in their computations (same output for same sequence of inputs)
 - Relies on group membership protocol: who is up, who is down? Only servers that are up need to have same state
- Passive replication
 - Primary – backup: brings the secondary server up to date when the primary fails
 - After each “write”, or periodically?

For high availability



- In both cases, servers need to
 - Respect (some) message ordering
- Implicit **agreement** among replicas

Recall: Interaction models

... in distributed systems from lecture 5

- Sharing state information at two servers needs a notion of time or event ordering
- The two possible models
 - Synchronous: related process/clock rates at different nodes, and bounded message delay
 - Asynchronous: related events and their partial order

The consensus problem

Assume that we have a
reliable broadcast

- Processes p_1, \dots, p_n take part in a decision
- Each p_i *proposes* (and broadcasts) a value v_i
 - e.g. application state info
- All non-faulty processes *decide* on a common value v that is equal to one of the proposed values

Important property

- No two non-faulty processes decide differently (Agreement)

Algorithms for consensus have to be proven to have this property



Basic impossibility result

[Fischer, Lynch and Paterson 1985]

There is no deterministic algorithm solving the consensus problem in an asynchronous distributed system with a single *crash* fault



On the other hand...

Byzantine Agreement Problem:

- Nodes need to agree on a decision but some nodes are faulty
- Faulty nodes can act in an arbitrary way (can be malicious)



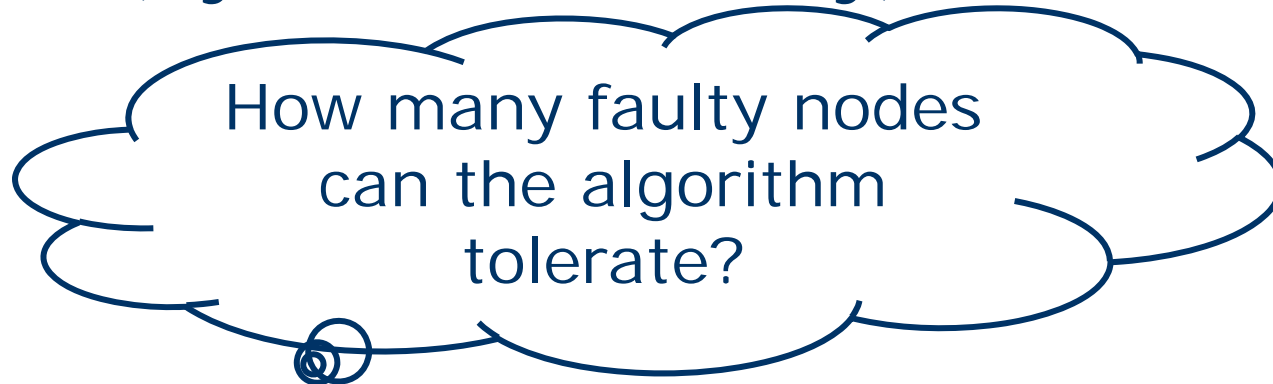
Byzantine agreement protocol

- Solves consensus in presence of synchrony



[Pease, Shostak and Lamport 1980]

- Agreement in presence of a harder fault model (Byzantine/arbitrary) is solvable!

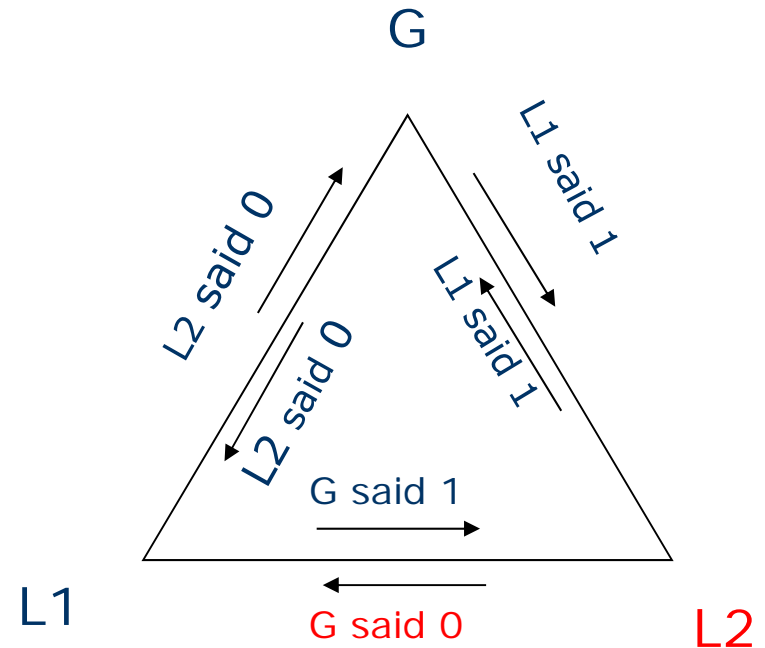
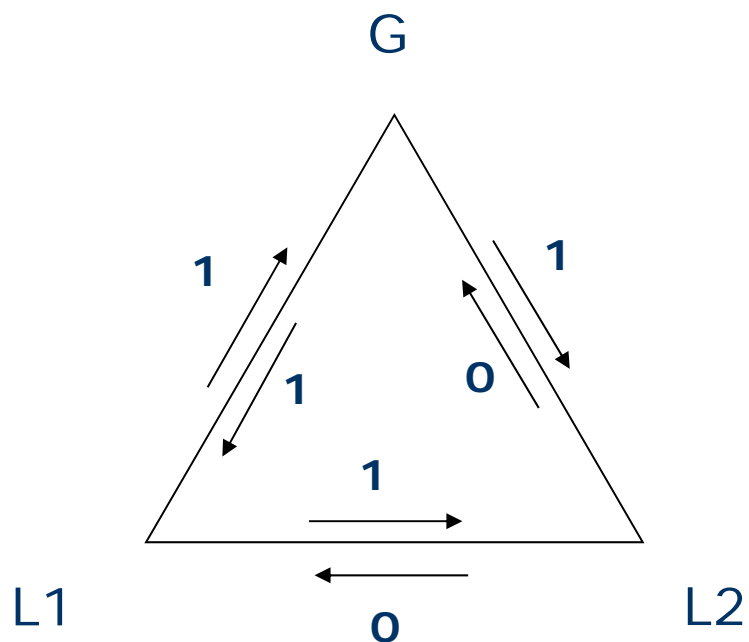


Result from 1980

- Theorem: There is an upper bound f for the number of byzantine node failures compared to the size of the network N , $N \geq 3f + 1$
- Gives a $f + 1$ round algorithm for solving consensus in a synchronous network
- Here:
 - We just demonstrate that $3f$ nodes would not be enough!

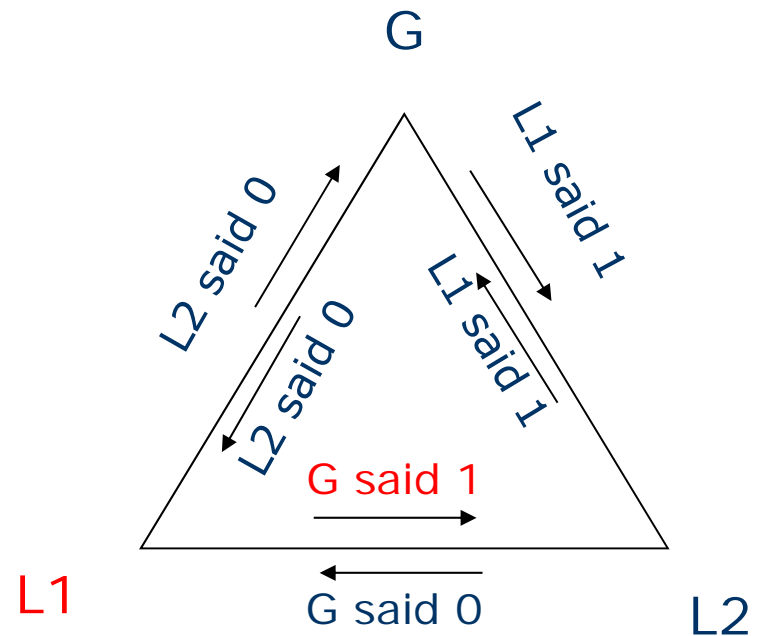
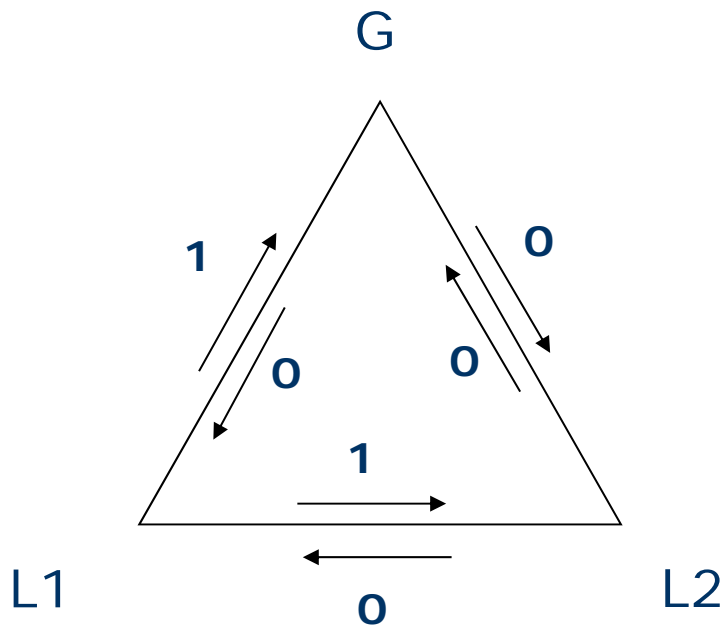
Scenario 1

- G and L1 are correct, L2 is faulty



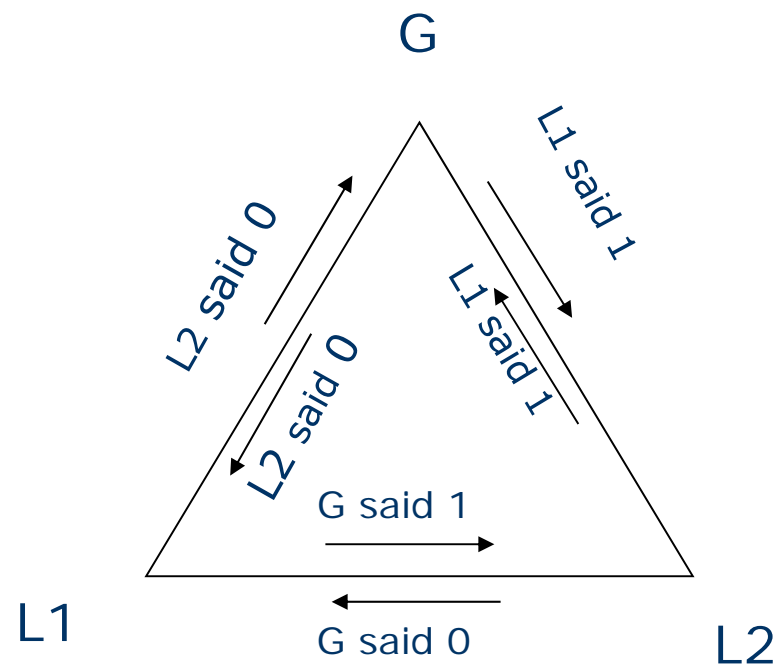
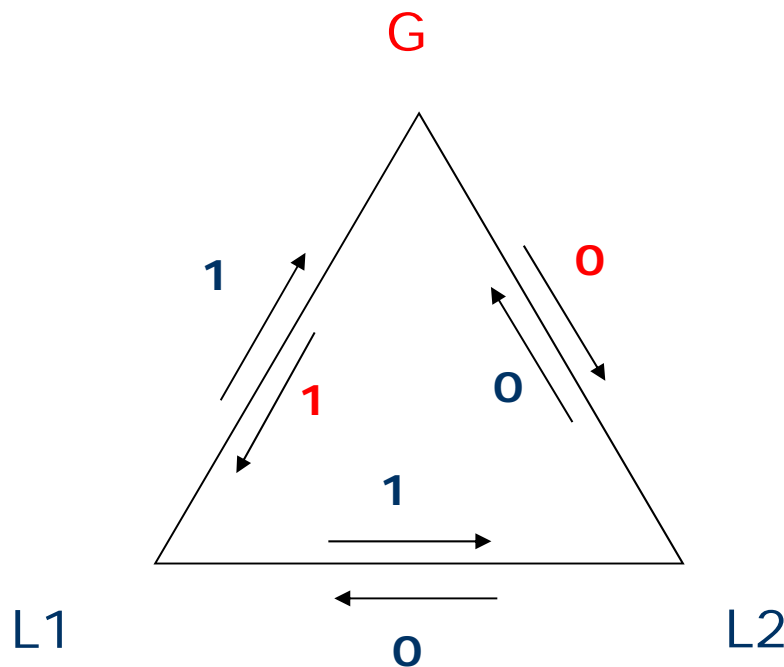
Scenario 2

- G and L2 are correct, L1 is faulty



Scenario 3

- The general is faulty!



2-round algorithm

... does not work with $f=1$, $N=3$!

- Seen from L1, scenario 1 and 3 are identical, so if L1 decides 1 in scenario 1 it will decide 1 in scenario 3
- Similarly for L2, if it decides 0 in scenario 2 it decides 0 in scenario 3
- L1 and L2 do not agree in scenario 3!

Summary

- Dependable systems need to *justify* why services can be relied upon
- To tolerate faults we normally deploy replication
- Replication needs (some kind of) agreement on state
- To prove correctness of agreement protocols in distributed systems we require explicit assumptions on faults (fault model), and (some) synchrony



The next three lectures

This lecture and part of lecture 9 covers theory

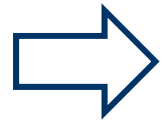
- Basic notions of dependability and redundancy in fault-tolerant systems
- Fault tolerance:
 - Relating faults/redundancy to distributed systems from lectures 4-6
 - Relating timing and fault tolerance

Lecture 8: industrial perspective

Lecture 9: Fault prevention and design aspects

Timing and fault tolerance

- We saw that support for timing guarantees helps fault tolerance



- We know that consensus is solvable under the synchronous model

- How does fault tolerance affect timing?
 - Implementing fault tolerance mechanisms (in turn) affects application timeliness

Recall in a TTP bus

- Nodes have synchronised clocks
- The communication infrastructure (collection of CCs and the TTP bus communication) guarantees a bounded time for a new data appearing on the communication interface of a (receiving) node (its CNI)
- That's why membership protocol could be implemented: Agreeing on who has crashed!

Other fault models:

- What are the faults that you can think of in a system connected by CAN or TTP?
 - Node related faults
 - Channel related faults
- How does CAN and TTP respectively provide help to detect the errors?

Timing and fault tolerance

- We saw that support for timing guarantees helps fault tolerance
 - We know that consensus is solvable under the synchronous model
- How does fault tolerance affect timing?
 - Implementing fault tolerance mechanisms (in turn) affects application timeliness

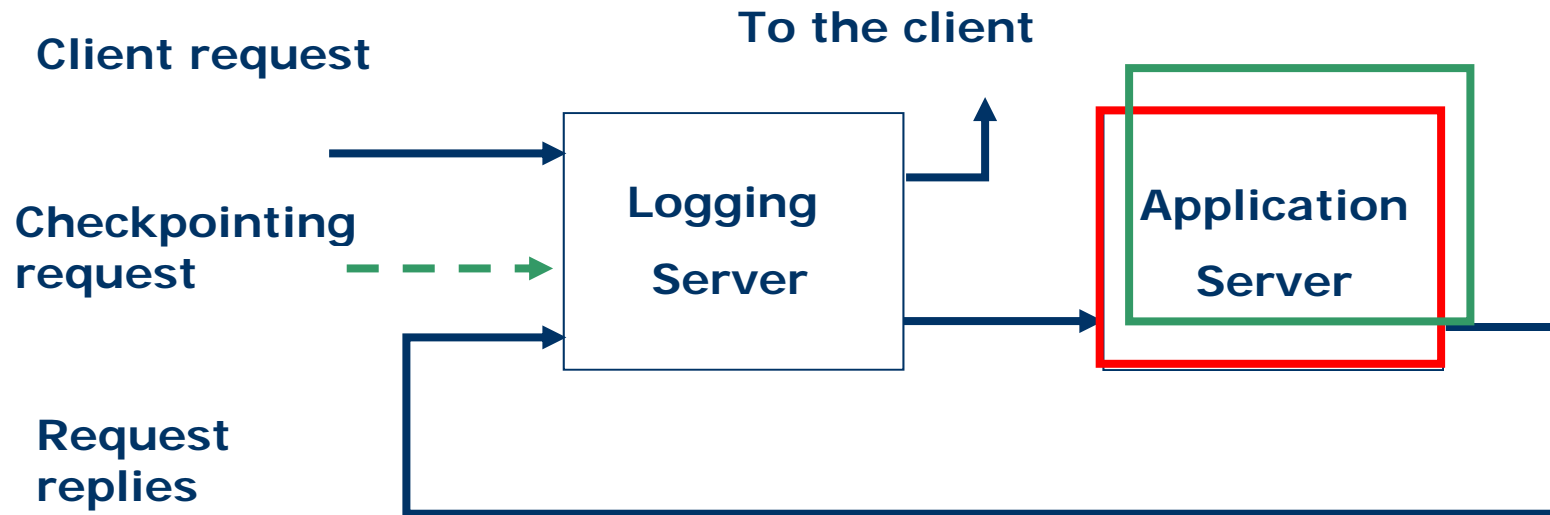
Timing overhead

- Does fault tolerance cost time? Why?
- Two examples:
 - Redundancy in time: Fault-tolerant scheduling
 - Redundancy in space: Replicated server, checkpointing

Fault-tolerant scheduling

- Assume we are running RMS
- How can we deal with *transient* faults that lead to a *process* crash?
- We can reschedule the process before its deadline!
 - Need to allow more than the (non-faulty) C_i for each process during analysis

Passive replication: checkpointing



- Optimal checkpointing interval for achieving highest availability

Fault prevention & timing

- Note that fault prevention has also an impact on timing!
 - Access control to stop unauthorised access/alteration
 - Authentication
 - Encryption, and so on...
- For some power control systems the response times are in the order of ms!