What Good are Models and What Models are Good?

Fred B. Schneider

Huygens Systems Research Laboratory

Universiteit Twente

Enschede

Understanding Complex Systems

Understanding complex systems or complex phenomena is hard.

Our intuition does not help when things become too complicated.

The scientific approach to understanding complex phenomena is

- Experimental Observation
- Modelling and Analysis

Computer Science is an Engineering Discipline

Is Computer Science really science?

If science is — as the OED says – ‘the systematic study [· · ·] of substances, animal and vegetable life, and natural laws,’ then computer science isn’t.

Computer science is an engineering discipline and involves the systematic study of building information-processing machines.

Engineering Method

A *specification* describes what has to be built.

The engineering process needs to produce an *implementation* and it needs to *validate* the implementation; i.e., check whether it *conforms* to the specification.

Specification

A *specification* describes what has to be built in terms of functionality — the *functional specification* describes what the system must do.

It can also describe constraints on the process of building the system (e.g., it has to be finished in a week), or on the cost of the system, or on the environment (e.g., total dummies have to be able to operate it).

Specifications

Some specifications are very exact, others are very vague.

Often, exact specifications can only be given for very small systems, such as, for instance, systems that do sorting or solve the travelling salesman problem.

Large problems, such as file servers, may have parts of the specification described in exact terms (e.g., correctness), but other parts can be described very vaguely (e.g., performance or fault tolerance).

Exact Specifications

Systems with an exact specification can be and should be designed using rigourously formal design processes.

In distributed operating systems research, however, such systems are rarely encountered.

Design

A *design* describes how a system will meet its specifications.

An *implementation* is a realization of a *design*.

Experimental Observation

Validation can be done by *testing*, by using *formal-proof methods*, by *code inspection*, etc.

Testing in engineering can be likened to *experimental observation* in science. It is an essential step in validating a design.

Synchronous and Asynchronous Systems

A distributed system is *synchronous* if

- Bounded relative processing speeds
- Bounded message transmission delays

It is *asynchronous* if no bounds exist

In a synchronized system it is possible to synchronize clocks

The Role of Failure

Independent failure ignored:

> "You know you have a distributed system when the crash of a computer you've never heard of stops you from getting any work done." *L. Lamport, 1987*

Independent failure exploited:

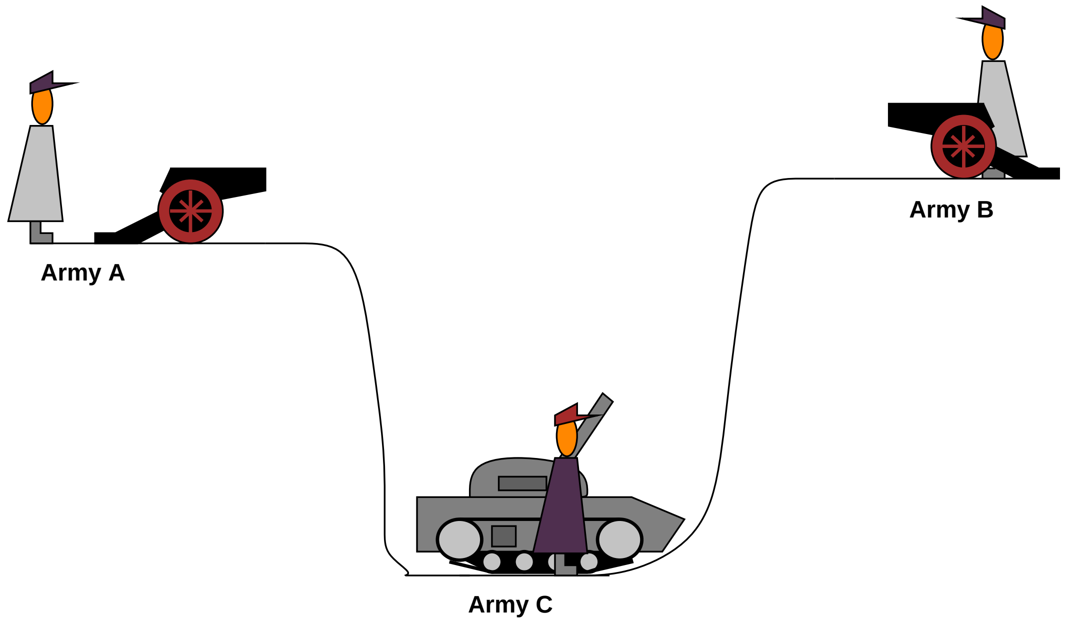
- increased availability
- increased reliability
- increased autonomy

The Role of Failure

Unreliable Connections:

Cannot distinguish *down* from *disconnected*

The Two Army Problem



- Messengers can be intercepted
- Either both armies attack or both armies retreat

Is there a protocol?

A Shortest Protocol

A to B:: “Attack!” ▮

B to A:: “Got Message ‘Attack!’ ” ▮

A to B:: “Got Message ‘Got Message “Attack!” ’ ” ▮

▮

...:

A to B:: Last Message ...

Failure Classification

Crash Failures: Processor crashes (and is rebooted) ▮

Omission Failures: Message loss, bad CRC on disk blocks ▮

Performance Failures: Overloaded operating system, network congestion ▮

Timing Failures: Fast or slow clock, response comes too early or too late ▮

Response Failures: Wrong response $(2 + 2 = 5)$, message altered on the wire

Processor Failure Classification

Byzantine: Arbitrary (or even malicious) failures

Fail Silent: System stops responding

Fail Stop: System stops responding and other systems can tell

Failure masking: If a depends on b and a provides service despite b's failure, then a **masks** b's failure.

Fail-stop failures are easier to mask than *fail-silent* ones and *fail-silent* failures are easier to mask than *Byzantine* failures.

Crash Failure Classification

Amnesia Crash: All state is lost ▮

Partial Amnesia Crash: Some state is lost ▮

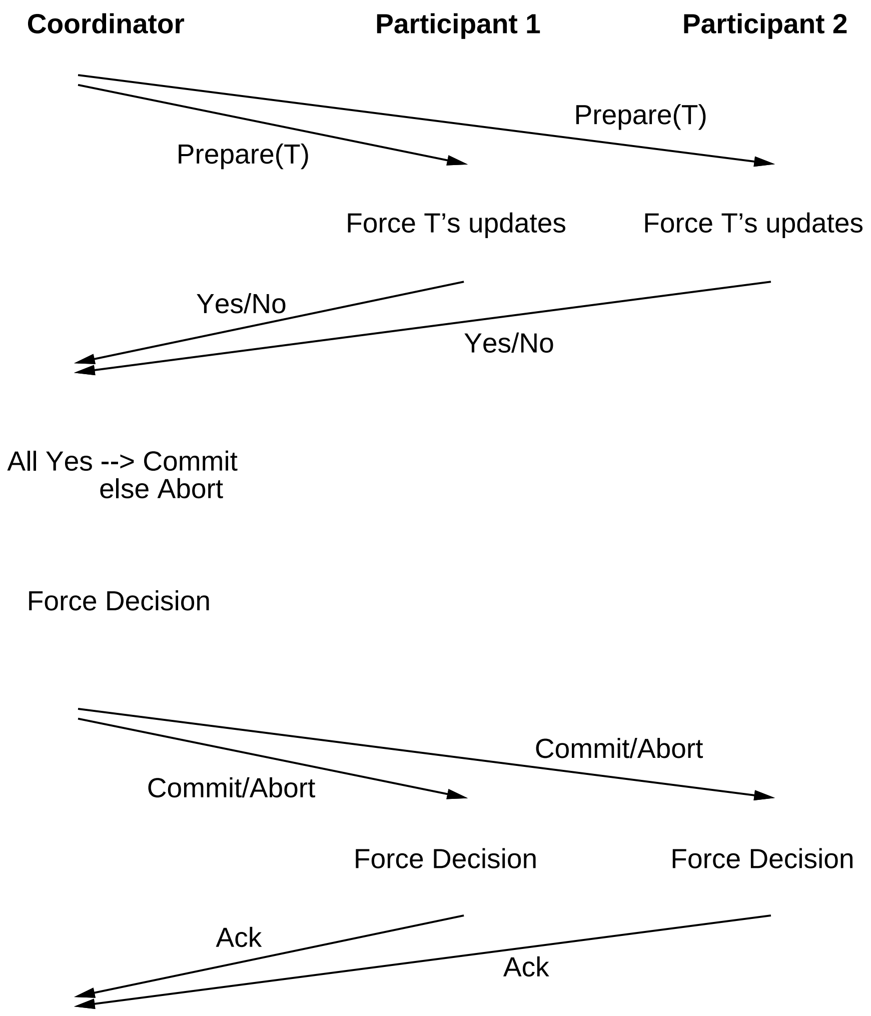
Pause Crash: Finite compact sequence of omissions ▮

Halting Crash: Infinite sequence of omissions

Fault Tolerance

A system is t fault tolerant if it satisfies its specification, provided that no more than t of those components becaome faulty during some interval of interest. *Fred Schneider*

Example: Transactions with Two-Phase Commit



Two Phase Commit

- Participants lose autonomy after prepare — must wait for a decision from coordinator.
- Cannot abort uncommitted transactions after a crash; must restore locks and versions.
- Vulnarable to partitions and crash of coordinator: correct but *blocks*.
- Alternative: three-phase commit (and variations)