Distributed system models

Refining intuition

- Good intuition is indispensable

Refining intuition

- Developing an intuition
  - Experimental observation: *build and observe*. Even if we may not understand why it works, this experience enables us to build things in similar settings.
  - Modeling and analysis: *simplify and analyze*. Provides a deep understanding of part of the system, and hopefully the simplified model still matches reality.

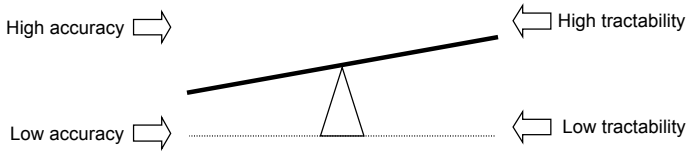
Good models

- But what's a model?
  - Collection of attributes and a set of rules that govern how these attributes interact
- Can a model be wrong?

A theory has only the alternative of being right or wrong. A model has a third possibility: it may be right, but irrelevant.
Manfred Eigen

Good models

- Accurate models
  - Yield truth about the object of interest
- Tractable models
  - Analyzing them is actually possible



Good models

- What to expect from a model
  - Feasibility
    What classes of problems can be solved?
    (in a given model)
  - Cost
    How expensive is the solution?
    (for solvable problems)

Good models

- A coordination problem
  - Processes A and B communicate by sending and receiving messages on a bidirectional channel. Neither process can fail, but the channel may lose messages.
  - A and B can execute two actions, α and β. Devise a protocol in which both processes take the same action, and neither takes both actions.

Good models

- There is no solution to the problem!!!
  (in the given model)
- Proof (by contradiction)
  - Any protocol executes in rounds of message exchanges: first (say) A sends a message to B, then B sends a message to A, and so on.
  - Let P be the protocol that solves the problem using the fewest rounds. Assume that the last message is sent by A, and let it be *m*.

Good models

- Proof (cont'd)
  - Observation #1: the action taken by A can't depend on *m*, because its receipt could never be learned by A (it's the last message).
  - Observation #2: the action taken by B can't depend on *m*, because B must make the same choice of action even if *m* is lost (due to a channel failure).

Good models

- Proof (cont'd)
  - Since the action chosen by A and B does not depend on *m*, it follows that *m* is not needed and so we can construct a new protocol in which one fewer message is sent…

…a contradiction! □

Good models

- What have we learned?
  - All protocols between two processes in this model are equivalent to a series of message exchanges
  - Actions taken by a process depend only on the sequence of messages it has received

Good models

- Going farther (or… scientific curiosity)
  - What if channels never fail?
  - What if channels only lose k messages?
  - What if processes know when a message is lost?
  - What about the cases for more than 2 processes?
  - …

Synchronous *vs.* asynchronous systems

- Asynchronous system
  - No assumptions about process execution speeds and/or message delivery delays
  - Weak assumption: any system is asynchronous
- Synchronous system
  - Relative process speeds and …
  - …Message delays are bounded

Synchronous *vs.* asynchronous systems

- How to choose?
  - Since any system is asynchronous, a protocol designed for such a system will work in any other system
  - So, why not develop all protocols for asynchronous systems?
  - So, why not develop all protocols for synchronous systems?

Synchronous *vs.* asynchronous systems

- Election problem
  - A set of processes P_1, P_2, …, P_n must select a leader. Each P_i has a unique identifier uid(i). Devise an asynchronous and a synchronous protocol so that processes learn the identifier of the leader. Processes start at the same time and communicate using broadcast.

Synchronous *vs.* asynchronous systems

  - Asynchronous system: each P_i broadcasts (i, uid(i)), and waits for the receipt of every broadcast message. P_i elects the process with the smallest uid(i).

Synchronous *vs.* asynchronous systems

– Synchronous system: let τ be a constant bigger than the largest message delivery delay; for simplicity assume local execution takes no time.

Each process P_i waits until either:

- P_i receives a broadcast, or
- τ * uid(i) time units elapse on P_i's clock, at which time P_i broadcasts (uid(i)).

The first process to broadcast is elected.

Failure models

- Assigning faulty behavior to components
  - Processes
  - Communication channels
- How it works
  - Faulty components
  - Not occurrences of faulty behavior
  - *t*-fault tolerant system

Failure models

- Examples
  - **Failstop**: a process fails by halting. Once it halts, the process remains in that state. Other processes can detect the failed process.
  - **Crash**: a process fails by halting. Once it halts, the process remains in that state. A failure may or not be detected by other processes.
  - **Byzantine failures**: a process fails by exhibiting arbitrary behavior.

Failure models

- Examples
  - **Crash+Link**: a link fails by losing some messages, but does not delay, duplicate, or corrupt messages.
  - **Receive-omission**: a process fails by receiving only a subset of the messages sent to it, or by halting (i.e., crashing).
  - …

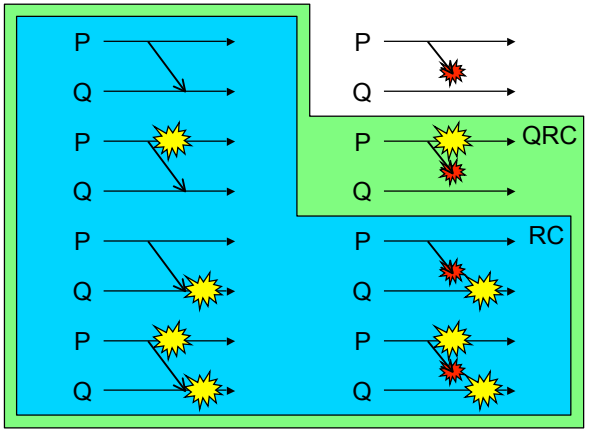
Which model when?

- A theoretical perspective
  - Models help understand how attributes affect the feasibility and cost of solving a problem
- A practical perspective
  - How to choose a model?
  - The real environment (e.g., databases)
  - Risk of failure (e.g., Byzantines and security)

Which model when?

- A communication example
  - Reliable channels
    If P sends a message m to Q and Q does not crash, then Q receives m.
  - Quasi-reliable channels
    If P sends a message m to Q and both P and Q do not crash, then Q receives m.

Which model when?



Which model when?

- Quasi-reliable channels:
  better match to real channels
  (e.g., what if a process fails before completing the send)
- Why bother about reliable channels?
  - Lower bounds