Vienna 1 Oct 2001

# Formal Verification in Support of TTA And Related Applications

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

## Overview

- Why do this?
- Basic algorithms
  - Individual verifications
  - Mutual interdependence of these
  - Other issues
- Application support
  - Programming model
  - Application-independent fault-tolerance
- Related activities elsewhere

## What Is Formal Verification? Engineering Analogy

- Engineers in traditional disciplines build mathematical models of their designs
- And use calculation to establish that the design, in the context of the environment, satisfies its requirements
- Only useful when mechanized (e.g., CFD)
- Used in the design loop (exploration, debugging)
  - Model, calculate, interpret, repeat
- Also used in certification
  - Verify by calculation that the modeled system satisfies certain requirements
- Need to be sure that model faithfully represents the design, and that the design is implemented correctly
  - These concerns are handled separately from verification

## Formal Methods: Analogy with Engineering Math (ctd.)

- Same idea
- The applied math of Computer Science is formal logic
- So the models are formal descriptions in some logical system
  - E.g., a program reinterpreted as a mathematical formula rather than instructions to a machine
- And calculation is mechanized by automated deduction: static analysis, model checking, theorem proving, etc.
- Formal calculations cover all modeled behaviors
- If the model is accurate, this provides verification
- If the model is approximate, still good for debugging (aka. refutation)

## Comparison with Simulation, Testing etc.

- Simulation also considers a model of the system (designed for execution rather than analysis)
- Testing considers the real thing
- Both differ from formal methods in that they examine only some of the possible behaviors
- For continuous systems, verification by extrapolation from partial tests is valid, but for discrete systems, it is not
- Can make only statistical projections, and it's expensive
  - 114,000 years on test for $10^{-9}$

  Limit to evidence provided by testing is about $10^{-4}$
- Heavy duty formal methods can consider all behaviors of an accurate model (but they are expensive)

## Comparison with Simulation, Testing etc. (ctd)

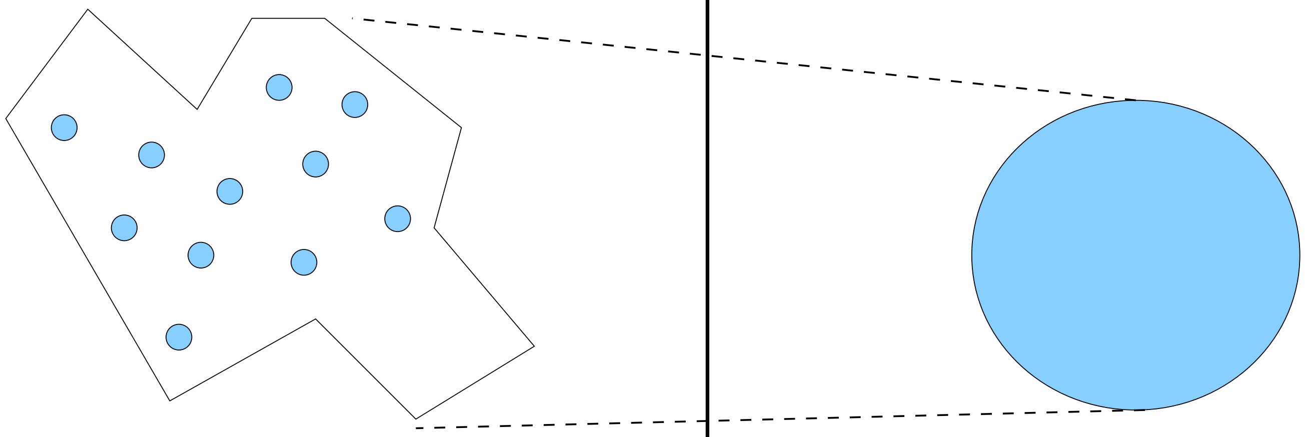
- Debugging by simulation and testing depend on skill in choosing test cases, luck
- Almost impossible when the environment can introduce huge numbers of different behaviors (fault arrivals, real-time, asynchronous interactions)
- Formal methods consider all behaviors, so certain to find the bugs
  - Provided the model and the properties checked are sufficiently accurate to manifest them

  So depends on skill in modeling, luck
- Experience is that you find more bugs (and more high-value bugs) by exploring all the behaviors of an approximate model than by exploring some of the behaviors of a more accurate one

# Formal Methods: In Pictures

## Testing/Simulation

## Formal Analysis



Real System

- Partial coverage

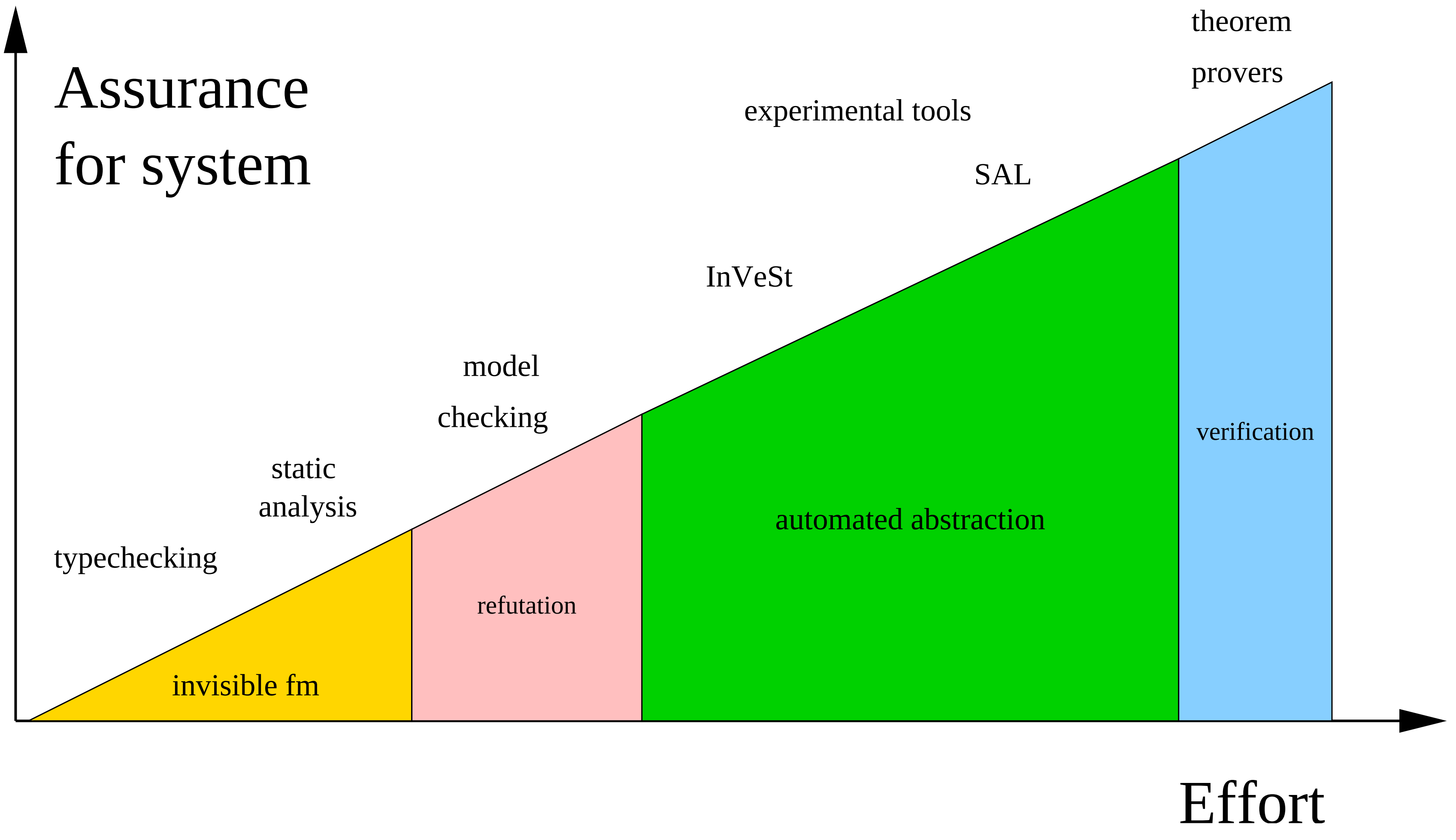
Formal Model

- Complete coverage
  (of the modeled system)

Accurate model: verification

Approximate model: debugging

# Another Picture: A Graduated Slope of Formal Methods



## Why Apply Formal Verification to TTA?

**Formal methods motivation:**

- TTA algorithms are challenging, push the technology of automated verification

**Fault Tolerance motivation:**

- Nowadays, expected to have at least an informal proof
- Formal proof gets into all the corners, may find bugs
- Formal proof exposes assumptions (fault hypotheses)
- Model checking and mechanized proof allow refined design exploration

  Pruning of assumptions, strengthening of claims

**Business case:**

- Useful for certification
- Product differentiator: if others do not do it
- Market necessity: if others do

## Basic Algorithms

- Clock synchronization
- Bus timing
- Group membership
- Clique avoidance
- MEDL properties

## Clock Synchronization

- The foundation of all else
- Basically the Lundelius-Lynch algorithm
  - Each node collects clock differences wrt. other nodes
  - Takes average of 2nd smallest and 2nd largest as its correction
- Formally verified by Shankar and improved by Paul Miner
- Except TTA uses only 4 clock differences
- Need to verify this variation
  - Need to be sure all nodes use the same 4 clocks
  - Which depends on membership

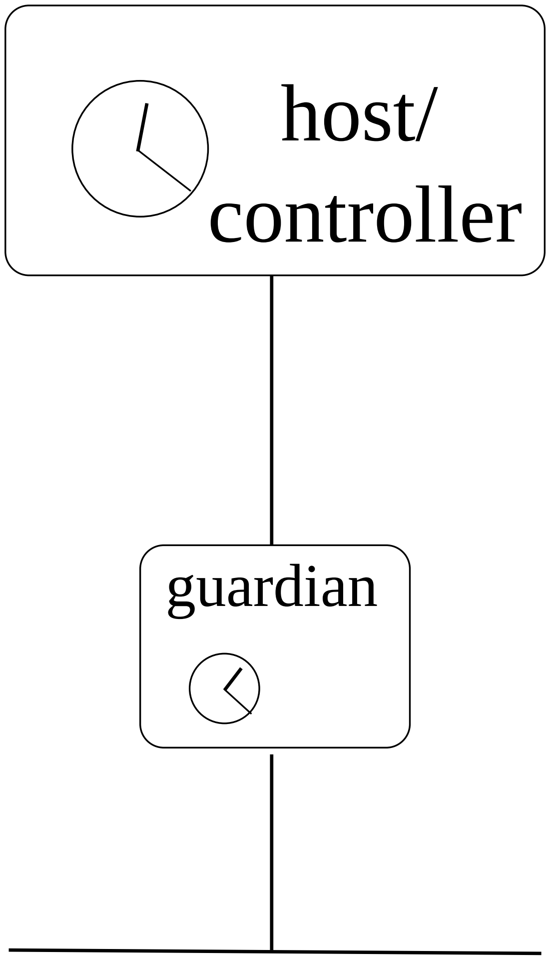
## Clock Synchronization (ctd)

- TTA algorithm was verified by group at Ulm
- But verification was lost in a fire
- A lot of work: don't want to repeat it
- My proposal: verify Ulrich Schmid's treatment of clock synchronization under hybrid fault model with link faults (independently interesting)
- Then interpret TTA algorithm in this model with $n-4$ "permanent" link faults to each node
- This will also allow formulation of properties when less than 4 good clocks remain

## Bus Guardian Window Timing

- Synchronized clocks and global schedule ensure no bus collisions between nonfaulty nodes
- But a faulty node might misbehave (or simply lose clock sync) and transmit out of turn—or all the time!
  - Babbling idiot failure mode
- So interpose a bus guardian that fails independently (has its own clock and copy of the schedule)
- Want the bus guardian window to be as narrow as possible
- But still pass all messages from nonfaulty nodes

## Bus Guardian



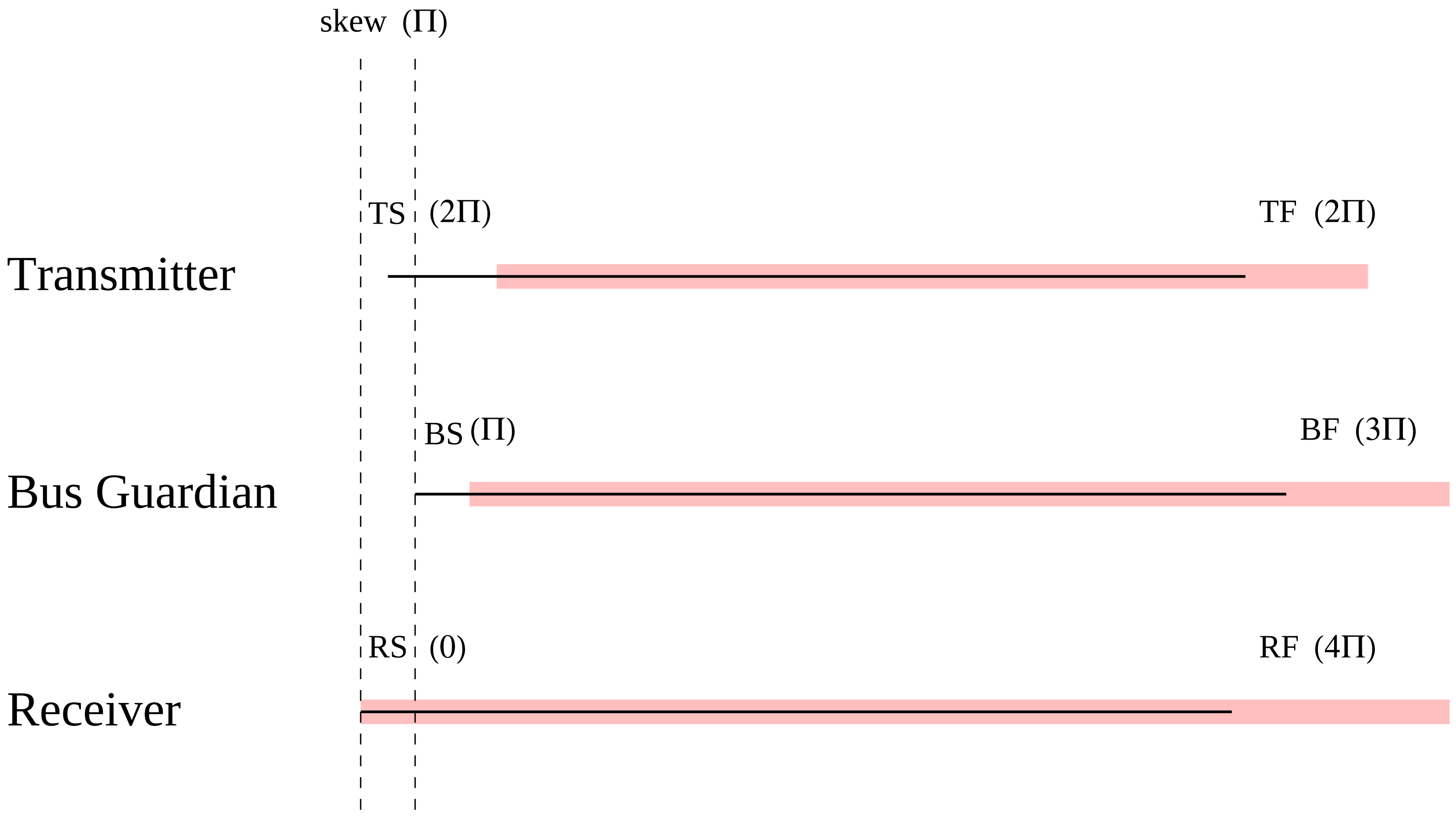
## Window Timing: Requirements

- Need to consider windows of three (classes of) components
  - A transmitter
  - Its bus guardian
  - The receivers
- Requirements

  **Validity**: If any nonfaulty node transmits a message, then all nonfaulty nodes will accept the transmission.

  **Agreement**: If any nonfaulty node accepts a transmission, then all nonfaulty nodes do
- Given that clocks are synchronized only within some parameter $\Pi$

# Window Timing: In Pictures



## Window Timing: Design Rules

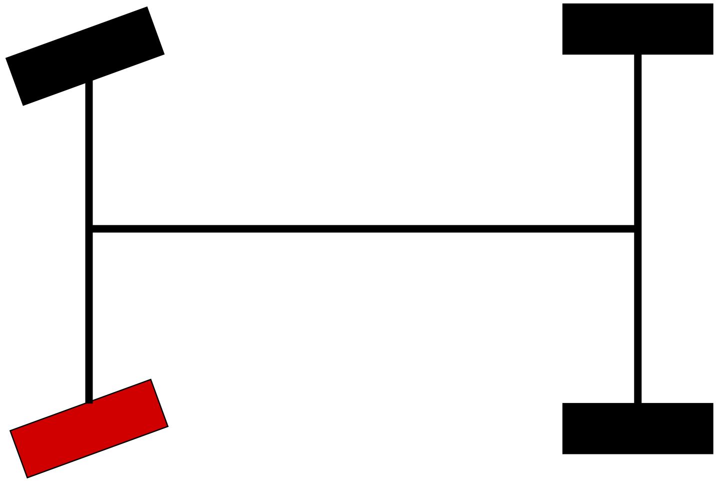
- Each slot has a start time and a maximum duration recorded in the schedule
  1. Transmission begins $2\Pi$ units after the beginning of the slot and should last no longer than the allotted duration.
  2. The bus guardian for a transmitter opens its window $\Pi$ units after the beginning of the slot and closes it $3\Pi$ beyond its allotted duration.
  3. The receive window extends from the beginning of the slot to $4\Pi$ beyond its allotted duration.
- I have formally verified these rules

## Group Membership

- Needed for internal fault-tolerance of TTA
  - TTA is designed to single fault assumption
  - Membership excludes faulty nodes, can then tolerate new fault
  - Therefore its properties are a strong influence on the fault hypothesis and arrival rate
- Also an application-level service

## Application Need for Group Membership

- Consider a brake-by-wire application
- Separate computers at each wheel adjust braking force according to inputs from brake pedal, accelerometers, steering angle, wheel-spin sensors etc.
- Suppose one of these computers fails



- The others need to redistribute the braking force
- So must have consistent opinion about who has failed

## Requirements For Group Membership

Each processor maintains a membership set

**Validity:** the membership sets of nonfaulty processors contain all the nonfaulty processors

- And, ideally, nothing else—but this is not possible because it takes some time to diagnose a faulty processor
- So allow at most one faulty processor in the membership

**Agreement:** all nonfaulty processors have the same membership sets

**Self-Diagnosis:** faulty processors eventually remove themselves from their own membership sets (and fail silently)

**Rejoin:** Repaired processors can get back in

Subject to fault hypothesis about possible fault modes, fault arrival rate, and maximum number of faults

## Verification of Group Membership

- Performed by Holger Pfeifer (Ulm): Forte/PSTV 2000
- Based on an approach developed by me: CAV 2000
- Generates a diagram of possible "configuration" that conveys a lot of insight into the operation of the algorithm
- Proof only does validity and agreement
- Should be extended to self-diagnosis and rejoin

## Clique Avoidance

- Implementation integrated with membership, but logically a separate algorithm
- Forces agreement on membership when outside fault hypothesis of membership algorithm (so part of "never give up" strategy)
- Not yet formally verified
  - The most interesting remaining challenge
  - Need to discover its properties/limits (i.e., what membership properties does it give up, what is its fault-hypothesis)
  - Has implications for application-independent fault-tolerance (see later)
- Some work by Merceron (see later)

## Interaction of Membership and Synchronization

- Each depends on the other
- How to break the circularity?
- There are assume/guarantee methods that do this
- Ken McMillan has a rule that is appropriate here: breaks the dependency by time
  - Membership at round $t$ depends on synchronization up to round $t-1$
  - Synchronization at round $t$ depends on membership up to round $t-1$

## Interaction of Membership and Synchronization (ctd)

- McMillan's rule: $H$ is a "helper" property, $\Box$ is the "always" modality of Linear Temporal Logic (LTL), and $p \triangleright q$ means that if $p$ is always true up to time $t$, then $q$ holds at time $t+1$ (i.e., $p$ fails before $q$)

$$\frac{\begin{array}{c}\langle H\rangle X_1 \langle P_2 \triangleright P_1\rangle \\ \langle H\rangle X_2 \langle P_1 \triangleright P_2\rangle\end{array}}{\langle H\rangle X_1 || X_2 \langle \Box(P_1 \wedge P_2)\rangle}$$

- I have formally verified McMillans's rule
- Now plan to apply it to synchronization/membership
  - Here, $X_1$ is membership, $X_2$ is synchronization
- Holger Pfeifer is working on the same problem from a different direction

## Other Issues Regarding Basic Algorithms

- Requirements on schedule/MEDL design
- Startup
- Restart
- Reintegration of failed nodes
- Mode changes and other reconfigurations
- Integration of all of these

## Top-Level Issues

- Partitioning
  - The main issue for aircraft certification
  - It's what allows several "functions" to be integrated on single platform
  - I don't have a good formalization of this, hence not sure how to formally verify it
  - I do have a report on "modular certification" that explains the issues informally
- The Time-Triggered Model of Computation
  - I would like to have a crisp formalization of this (cf. "synchronous system")
  - Tom Henzinger has Giotto: a time-triggered language, but that's not what I'm after
  - Do temporal firewalls belong here or below?

## Top-Level Issues (ctd)

- Application-independent fault tolerance
  - Services that make it possible/easy to make applications fault tolerant

    **Possible:** must have replica determinism

    **Easy:** automatic distribution and voting

## Replica Determinism

- Aka. Byzantine agreement, consensus, interactive consistency
- Usually requires two-round algorithm to tolerate single arbitrary fault
- In writing my report comparing bus architectures, I realized TTA achieves this in a novel way with just a single round
  - CRCs eliminate some fault modes on broadcasts
  - Need to worry only about received/not received splits
  - But these are eliminated by membership (single fault) or clique avoidance (multiple fault)

  Combination provides "Draconian Agreement"
- Need to formally verify this
- All buses except FlexRay provide some form of replica determinism

## Replica Determinism (Multiple Round Algorithms)

- Ulrich Schmid and Bettina Weiss have extended several consensus algorithms to a realistic hybrid fault model
  - Manifest, symmetric, omission, arbitrary

  Plus link faults
- I have formally verified OMH(r) under this model
- Still negotiating exact formulation of assumptions
- Formalization revealed that all algorithms need to be modified to ensure that "slightly faulty" senders maintain agreement with receivers

## Related Activities Elsewhere

Agathe Merceron (University of Sydney, Australia)

- Has verified "no collisions" and "no cliques"
- Model checking (i.e., refutation, not verification) and I'm skeptical about some of the treatment
- A student has simulated TTA

## Related Activities Elsewhere (ctd)

Daniel Kroening (CMU, ex. Saarbrücken)

- Working for Ed Clarke on GM contract
- Initially TTA: found errors (rqt. for strong fault model) in early membership
- Now FlexRay (GM favor this)
    - They (Motorola) are adding membership
      Though GM say they have to replicate it at application level anyway
    - Have new, patented clocksync algorithm
    - Claim is simple, uniform proofs