FTRTFT Oldenburg Sep 12, 2002

An Overview of Formal Verification For The Time Triggered Architecture

John Rushby

Computer Science Laboratory

SRI International

Menlo Park, California, USA

The Time-Triggered Architecture: What Is It?

- The Time-Triggered Architecture (TTA) is a platform for safety-critical embedded systems
  - E.g., aircraft and engine flight control, and “by wire” cars
- Functionally, it is a TDMA (time-triggered) serial bus
- “Bus” understates its criticality and sophistication
  - It is the safety-critical core of the systems built above it
- Must achieve failure probability below 10^{-10}/hour for 10 hours, maximum outage 10ms

TTA: Where Did It Come From?

- Developed by the group of Hermann Kopetz, TU Vienna
- Commercialized by TTTech
- Builds on a lineage of research architectures that developed principled solutions to the challenges of concurrent, real-time, distributed, fault-tolerant systems design
  - SIFT (SRI), FTP, FTPP (Draper), MAFT (Allied Signal), MARS (TU Vienna)
- TTA is unique in being developed for mass-market for automobile applications (Audi, PSA etc.) but also used for aircraft applications (Honeywell)
  - "Aircraft safety at automobile cost"

Similar Systems

- There are other safety-critical buses
- Avionics: SAFEbus (Honeywell 777 AIMS), SPIDER (NASA)
- Automotive: TTA, FlexRay (Daimler/Chrysler et al)
- I've written a NASA Tech Report and a paper presented at EMSOFT '01 that compare them
- Use Google to find my home page, follow link to my papers

Applications of TTA and Similar Buses

- Safety-critical embedded systems

  Avionics "functions": flight control, autopilot, autoland, flight management, displays. . .

  Aircraft "controls": engine controls, thrust reversers, cabin pressurization, brakes, doors and slides, public address,. . .

  Automotive: "by wire" brakes, suspension, steering,. . .

- TTA specifically
  - Engine controller for an Italian fighter (Honeywell Tucson)
  - Engine controller for F16 (Honeywell Tucson)
  - Environmental control for A380 (Hamilton Sundstrand)
  - GenAv cockpits (Honeywell Olathe)
  - By wire applications in next generation cars (Audi, PSA. . .), Snowcats, . . .

Fault Tolerant Architectures

- Provide basic services to a collection of host computers
  - Timing, communication

  These services must not fail, despite failure of components

- Support fault tolerant applications in the hosts
  - E.g., through state machine replication

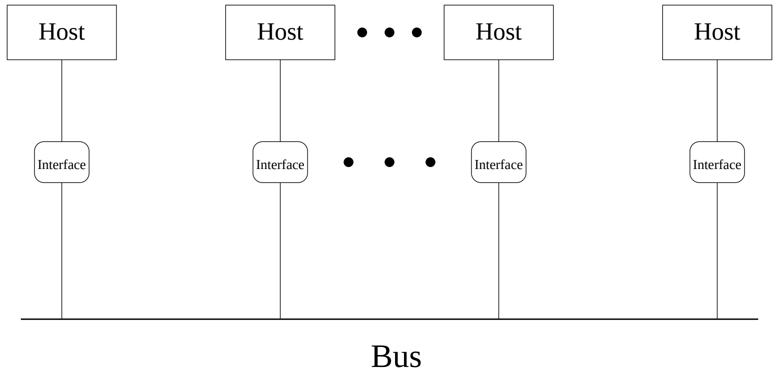
  Consistent message delivery, failure notification, partitioning

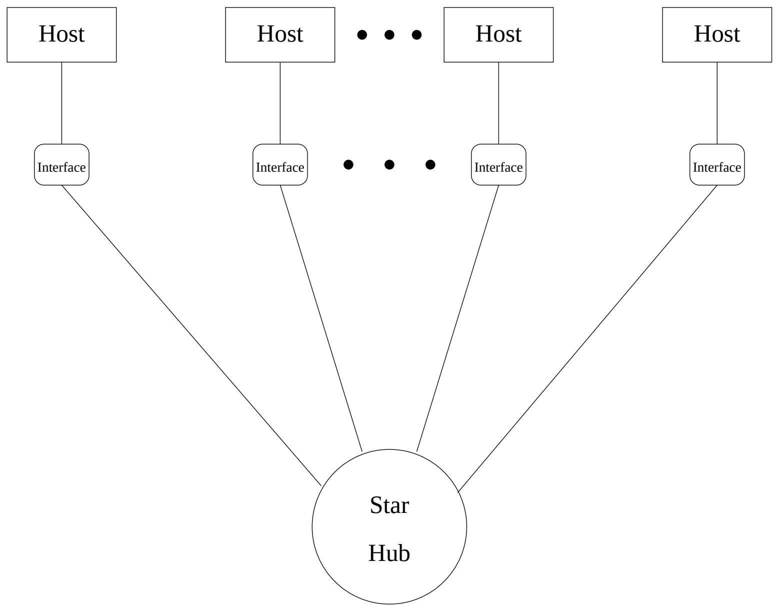
The Rôle of Buses

- There must be some communication system for exchanging sensor samples, state data, control signals, actuator outputs
- Many possible topologies, but only a serial bus is economically viable
- The bus is then a critical shared resource
  - Communication must be assured with guaranteed bandwidth, low jitter, low end-to-end latency
  - In the presence of faults
- Bus embodies the fault tolerant architecture

Basic Characteristics of TTA

- Exists in both bus and star topologies (logically still a bus)



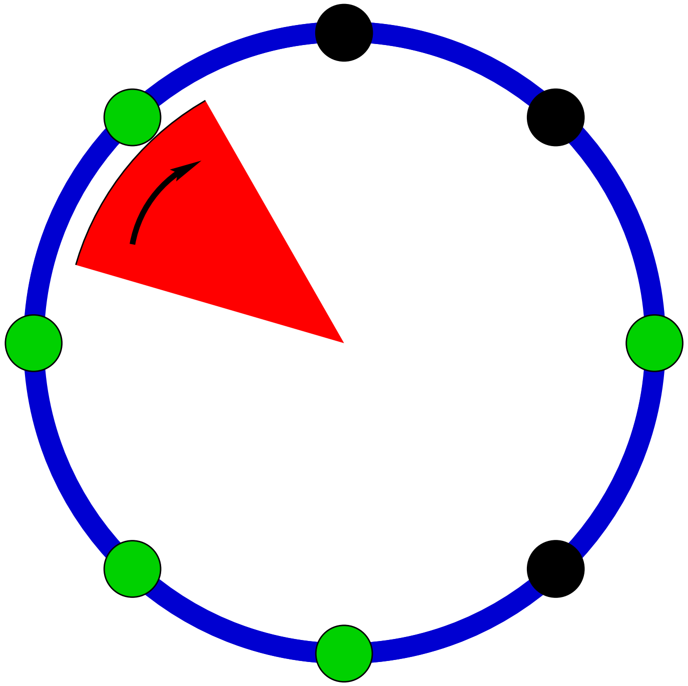


Bus/hub are replicated

- All functionality implemented in the distributed interfaces (called TTP/C controllers)
- And in the hub of the star topology (a modified controller)

Basic Characteristics of TTA (ctd.)

- Creates a synchronous, TDMA ring on a broadcast bus



- Global clock (achieved by synchronizing local clocks)
- Global schedule known at all nodes

Why Formal Verification?

Safety motivation:

- Need all the assurance possible
- Help move certification from process- to product-basis
- Help develop approach to modular certification

Developer (TTTech) 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

Formal methods motivation:

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

The TTA Algorithms are Challenging...

- TTA comprises several algorithms
- That are individually challenging for formal verification
- Even in their "academic" form
  - Hard to do at all
  - Really hard to automate

  Further complicated by practical details
- The algorithms interact in interesting ways
- And some of the most important properties are emergent
  - Consistent message delivery is achieved indirectly, not by an agreement algorithm
  - Partitioning is not ensured by any individual algorithm

The TTA Algorithms are Challenging To...

- I'll sketch formal analyses by several projects and groups
- Projects
  - SRI, with Honeywell Tucson and NASA
  - NextTTA: TU Vienna, VERIMAG, Ulm, ...
  - RISE: Esterel, Verimag, ...
- Groups
  - Liafa, Paris 7
  - PAX, Kiel
- But I'll focus on what remains to be done

Aside: Formal Verification of Fault Tolerant Algorithms

Fault Hypothesis and Fault Containment Units

- Must identify the fault containment units (FCUs) that faults can afflict
  - Faults at different FCUs must be independent
  - Need design evidence for this
    (separate power, physically apart)
- Must state an explicit fault hypothesis
  - The modes (kinds), number, and arrival rate of faults that can afflict FCUs
  - Must be validated by experiment, experience
- Redundancy and suitable algorithms then provide fault tolerance: this is what we verify
- And should have a never give up (NGU) strategy in case the fault hypothesis is violated

Formal Verification and Stochastic Modeling

- Architecture must be shown to satisfy the mission requirements under its fault hypotheses
- Formal verification establishes theorems of the form

  fault hypothesis satisfied $\vdash$ architecture works correctly

- Stochastic modeling establishes probability of the hypothesis (hence, ability to satisfy the mission requirement)

  System failures that could lead to a catastrophic failure condition must be "extremely improbable," which means that they must be "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of one type" . . . "When using quantitative analyses. . . numerical probabilities. . . on the order of 10^{-9} per flight-hour

  [FAA Advisory Circular 25.1309-1A]

Specific, Arbitrary, and Hybrid Fault Models

Specific: enumerate the possible fault modes, provide defense for each one

- Need to show no other kind of fault can occur

Arbitrary (aka. Byzantine): no assumptions at all on behavior of faulty elements

- Requires a lot of redundancy
- Could fail under lots of simple faults

Hybrid: combination of the above

- Originally: arbitrary, symmetric, and manifest node faults
- Improvement: adds omission node fault, plus link faults
- Just right

Formal Verification With Hybrid Fault Models

- Establish theorems such as
- ICAH (a clock synchronization algorithm) maintains synchronization provided

$$n > 3a + 2s + c$$

Where

- n is total number of clocks
- a is number that are arbitrary faulty
- s is number that are symmetric faulty
- s is number that are manifest faulty

Return from Aside

Basic Algorithms of TTA

- Clock synchronization
- Bus guardian window timing
- Group membership
- Clique avoidance
- Nonblocking write
- Startup/restart

TTA Clock Synchronization

- Keeps good clocks close together, in presence of faulty clocks
- Based on the Lundelius-Lynch algorithm
  - Each node collects clock differences wrt. other nodes
  - Takes average of 2nd smallest and 2nd largest as its correction
- Restrict to nodes that have accurate oscillators
- But TTA uses only 4 clock differences
- Tolerates a single arbitrary fault

Clock Synchronization: Previous Verifications

- Byzantine fault-tolerant clock synchronization algorithms are a major challenge for formal verification systems
  - Intricate combination of arithmetic and combinatorial reasoning
- Friedrich von Henke and I were the first to verify one (called interactive convergence) using EHDM (TSE '93)
  - Subsequently repeated by Bill Young using Nqthm
- Schneider's general treatment and Lundelius-Lynch instantiation formally verified by Shankar (FTRTFT 92) and improved by Paul Miner (MS Thesis) using EHDM
- Verification of interactive convergence extended to hybrid fault model by me (PODC 94)

Clock Synchronization: TTA Case

- TTA uses only 4 clock differences
- Miner's treatment was converted to PVS, generalized, and applied to TTA variant by group at Ulm (DCCA '97)
- But then lost in a fire
- Need to recreate this, but don't want merely to repeat the lost Ulm treatment
- Satisfaction of mission requirements requires a hybrid fault model
  - This will allow formulation of properties when less than 4 good clocks remain, or more than a single fault arrives

Clock Synchronization: Full TTA Case

- Proposal: verify Ulrich Schmid's treatment of clock synch. under hybrid fault model with link faults (DSN '01)
  - Independently interesting
- Then interpret TTA algorithm in this model with $n - 4$ "permanent" link faults to each node
- Will be interesting to compare gain in efficiency of PVS over EHDM (hope for an order of magnitude)
- But real desire is for fully automated proofs
  - Feasible with timed/hybrid automata?

Bus Guardians

- A faulty node could broadcast at the wrong time
- Or all the time (babbling fault mode)
  - Destroys all good communications
- Must introduce a separate FCU with own clock and knowledge of schedule that mediates access to the bus
- This is a (logical) bus guardian
- Several design choices

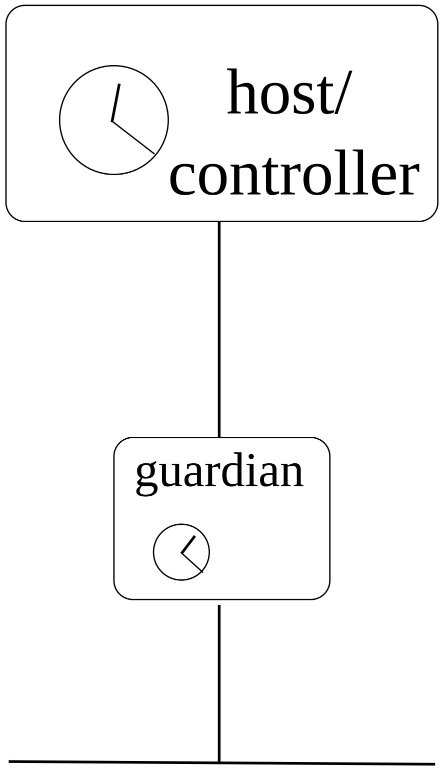
  SAFEbus: paired interfaces (and buses): each is a guardian for the other

  TTA-bus, FlexRay: explicit guardians

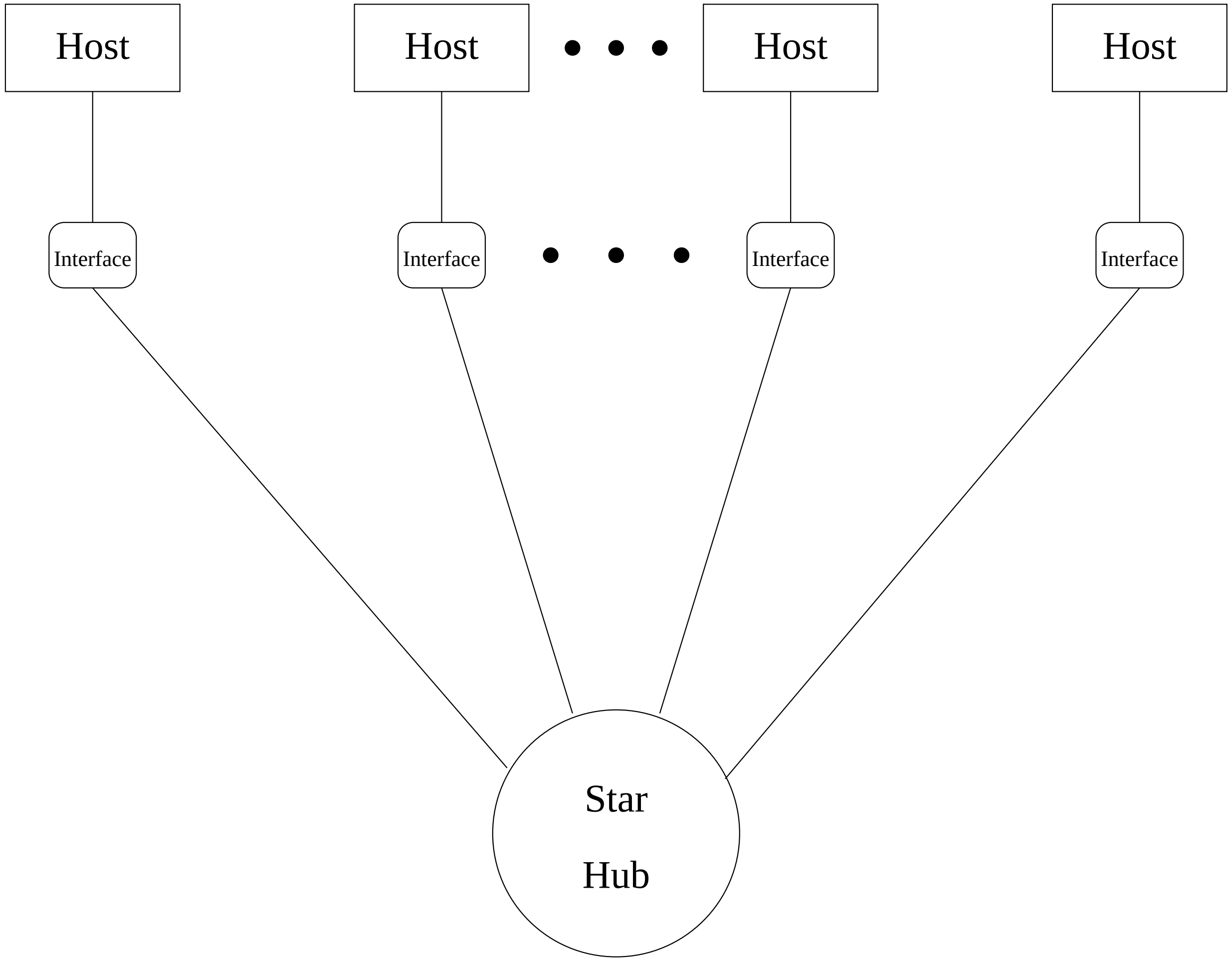
  TTA-star: guardian functionality in central hub

Explicit Guardian

- One per bus, or shared?
- Fully independent clock synchronization?



Guardian in Central Hub



Bus Window Timing

- Bus guardian allows its node to write to the bus only during a limited window
- Want the bus guardian window to be as narrow as possible
- But still pass all messages from nonfaulty nodes
- Despite the fact that clocks are only loosely synchronized
- Also, no source or destination addresses are sent with messages
  - These are determined by time message sent
  - Eliminates masquerading, greatly increases bandwidth
- So receivers also maintain a narrow reception window

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 Π

Window Timing: Design Rules

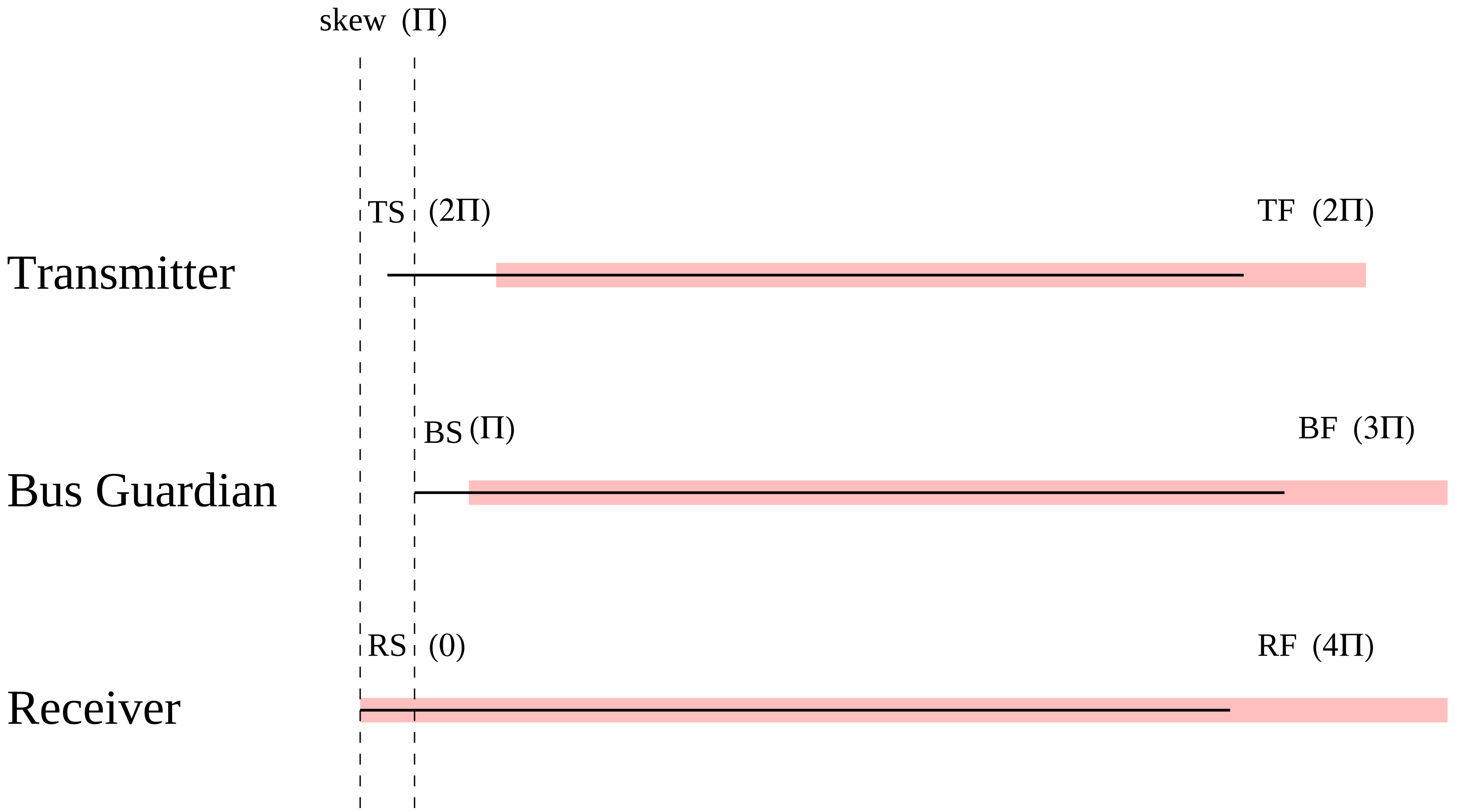
Each slot has a start time and a maximum duration recorded in the schedule

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 Π 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.

Window Timing: In Pictures



Verification of Window Timing

- Done by me (Tech Report)
- Straightforward and largely automatic (used as tutorial)

Asynchronous Communication

- An important element in Kopetz' conception of time-triggered systems is the distinction between elementary and composite interfaces
- Control flow must be unidirectional for elementary interfaces
- At the TTA the controller/host interface, we need reliable, timely communication across an asynchronous interface with no handshakes or blocking
- In computer science, this is called a wait-free, lock-free, atomic register construction
- TTA uses algorithm called NBW (nonblocking write)
  - A combination of Lamport's lock-free construction
  - And ideas from Simpson's wait-free construction

Safe, Regular, Atomic Registers

What happens when we read memory at the same time it is being written?

Consider a read that overlaps possibly many writes

Safe: can get any value

regular: gets one of the values written

atomic: a series of reads behaves in a manner that is consistent with the reads and writes interleaving in some order (reads never return older values than previous reads)

For atomic registers, want mutual-exclusion on access to the register

- Lock-free: no blocking
- Wait-free: always get the most recent

Simpson's 4-Slot Algorithm

- Patented by BAe in the 80s
- Widely used
- Uses 4 safe slots (buffers)
- And 4 Boolean control registers
- To construct a wait-free, lock-free atomic register
- What are the assumptions on the control registers?

Analyzing 4-Slot

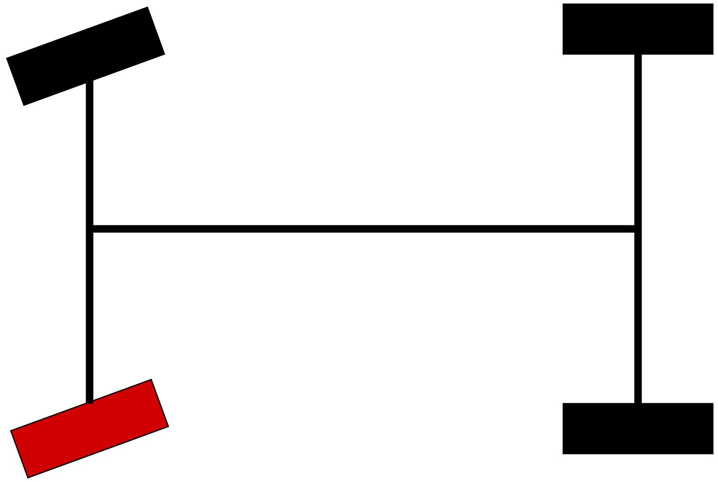
- I did it by model checking with SALenv
  - Its first road test
- Found that it achieves mutual exclusion even when the control registers are merely safe
  - Finite state, so model checking provides verification
- But does not achieve atomicity
- Even if control registers are written only when changed
  - This makes them regular, not atomic
- Requires atomic control registers!
- This is discussed in Simpson's papers, but was interesting to independently discover it by model checking
- There is a large activity on these algorithms in UK, and interesting work by Hesselink (ACTA '02)

Group Membership

- Similar to fault diagnosis
- Informs good nodes which other nodes are good
- Needed for internal fault-tolerance of TTA
  - TTA is designed to single fault assumption
  - Membership excludes faulty nodes, can then tolerate new faults
  - Therefore its properties are a strong influence on the fault hypothesis and arrival rate
- Is also an application-level service

Applications Need Consistent Knowledge

- 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

TTA Group Membership Algorithm

- Each broadcaster acknowledges the previous two
  - Requires only two bits per message (encoded in CRC)
- Works only under symmetric fault model
- And no more than one fault per two rounds

Verification of Group Membership

- Lincoln verified MAFT diagnosis algorithms (TSE 95)
- We became interested in verifying membership, which is a similar problem
- But TTA algorithm was not published at that time
- So Katz, Lincoln, and I invented our own (WDAG ’97)
- Needs only one bit per message
- Verified by hand

The Published WDAG Proof

- Was a conventional inductive invariance proof
- It is incorrect (incomplete)
  - And the algorithm has a bug
    - ⋆ Found independently by Shankar (inspection), and
    - ⋆ Sadie Creese and Bill Roscoe (model checking)
- But is fairly easy to correct
- However, it defeated attempts by Pat Lincoln, Shmuel Katz, and me to formally verify it in PVS
  - Because of its horrible complexity

The Published WDAG Invariant

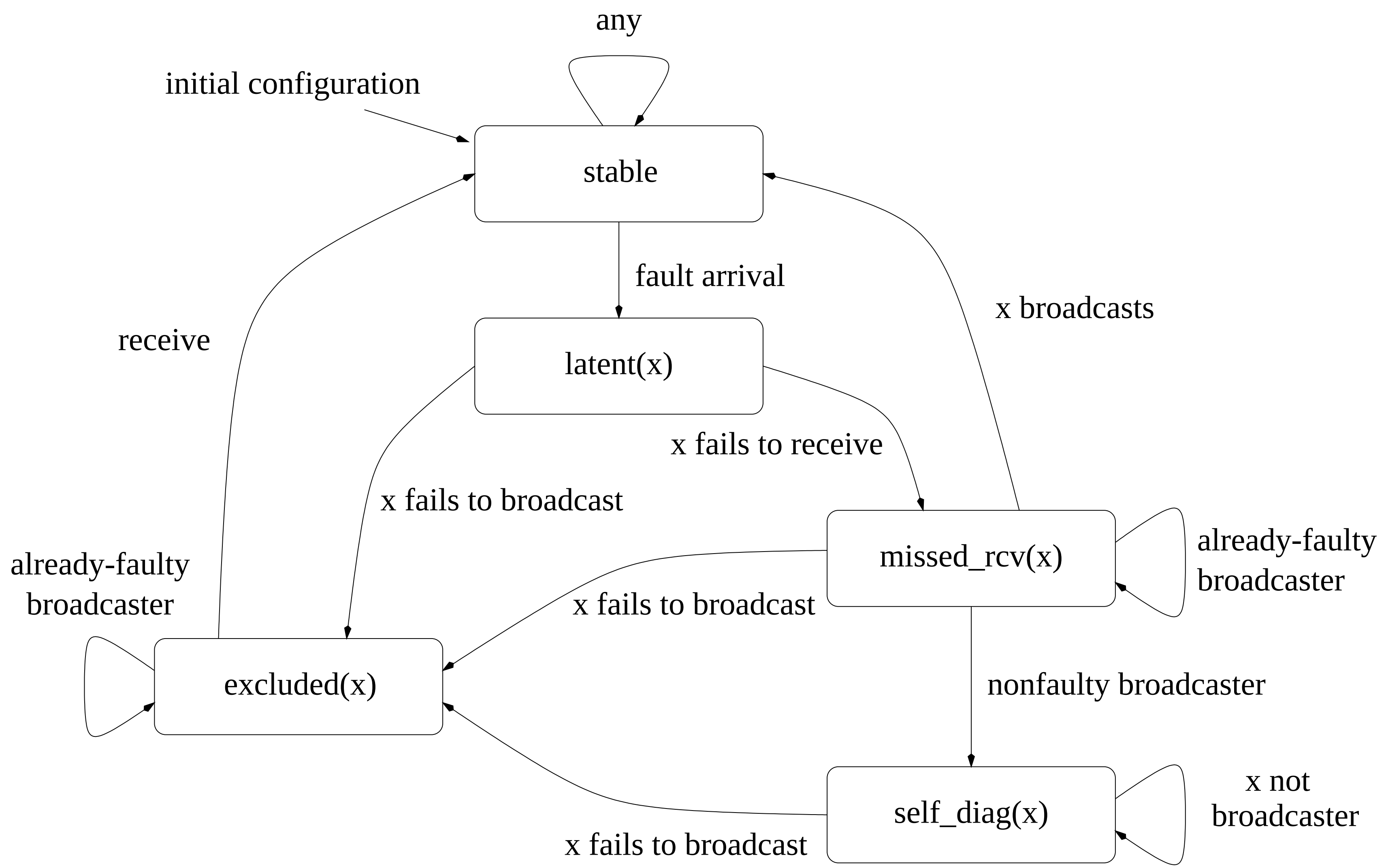
The invariant has the following conjuncts.

1. All nonfaulty processors have the same membership sets.
2. All nonfaulty processors are in their own membership sets.
3. All nonfaulty processors have the same value for $\mathtt{ack}$.
4. For each processor p, $\mathtt{ack}(p)$ is true iff in the most recent previous step in which p expected a broadcast from a processor b, either p was b, or $arrived(b, p) \wedge (\mathtt{ack}(b) \vee \neg\mathtt{ack}(p))$ in that step.
5. If a processor p became faulty less than n steps ago and q is a nonfaulty processor, either p is the present broadcaster or the present broadcaster is in p's local membership set iff it is in q's.
6. If a receive fault occurred to processor p less than n steps ago, then either p is not the broadcaster or $\mathtt{ack}(p)$ is $false$ while all nonfaulty q have $\mathtt{ack}(q) = true$, or p is not in its local membership set.
7. If in the previous step b is broadcaster, p is a nonfaulty processor, and $arrived(b, p)$ does not hold, then b is faulty in the current step.
8. If the broadcaster b is expected by a nonfaulty processor, then b is either nonfaulty, or became faulty less than n steps ago.

Successful Verification of Membership

- I found a method to verify the WDAG algorithm
- Uses disjunctive invariants
- Proof has a natural diagrammatic representation
- And can be constructed systematically
- I described the method using a simplified version of the WDAG algorithm (CAV ’00)

There is a Natural Diagrammatic Representation



Verification of TTA Group Membership

- Performed by Holger Pfeifer (Forte/PSTV 00)
- Based on disjunctive invariants method (CAV 00)
- Generates a diagram of possible “configuration” that conveys a lot of insight into the operation of the algorithm
- Proof is completely systematic, but not highly automated
  - Well. . . try it in your prover

Other Verifications of Membership

- Creese and Roscoe verified the WDAG algorithm by manually abstracting it to a finite configuration, then model checking
- Problem with such approaches is that formal verification of the abstraction is hard
- An alternative uses theorem proving to construct the abstraction
  - E.g., predicate abstraction
  - Creates the context for failure-tolerant theorem proving
  - Precision of the abstraction depends on the theorem proving power deployed
- PAX group at Kiel use WS1S and Mona to perform automated abstraction
- Handles the CAV algorithm automatically

Clique Avoidance

- Membership is verified under benign fault hypothesis:
  at most one symmetric fault every two rounds
- Beyond this fault hypothesis lie
  - Asymmetric faults
  - Multiple faults
  - Node faults
  - Arbitrary faults
- Clique avoidance (elimination) algorithm forces agreement on membership when outside fault hypothesis of membership algorithm
  - So part of “never give up” strategy
- May sacrifice validity

SOS and Asymmetric (Byzantine) Faults

- SOS = slightly out of specification
- Weak power supply or faulty line driver may send intermediate voltages
  - Neither digital 0 nor 1

  Some receivers may see 0, others 1, and others may reject
- Or may send weak (slow rise) edges
  - May look like 0 or 1, depending when sampled

  Some receivers may see 0, others 1, and others may reject
- Or clock drift may put edges at edge of sampling interval
- Or could go metastable
- All these can give rise to asymmetric reception
- Can reduce incidence of these with central hub
- But cannot eliminate at 10^{-10}

Group Membership and Clique Avoidance

- Group membership and clique avoidance are not separate algorithms, but intertwined
- Can start from a basic group membership algorithm that works on the basis of implicit acks from successor and next-successor
- Then add accept and reject counter
- Replace some of the fault detection by comparison between these counters
- Still have membership, but also ability to tolerate wider class of faults—this is clique avoidance
- Can then consider clique avoidance as a self stabilizing extension to group membership

Self Stabilization

- Given a network of processes in arbitrary initial states, prove they converge to some good state
- A good model for recovery from transient faults
  - Components do silly things, then the faults go away
  - Leaving just the contaminated state

  (Combination with permanent faults is a research topic)
- Previous verifications were tours-de-force
- Detectors and correctors theory of Kulkarni and Arora provides tractable treatment (formalized in PVS by Kulkarni)

Detectors and Correctors

- Stripped down version of the theory, with only correctors
- "Base" algorithm B whose purpose is to maintain invariant S in presence of fault class F (e.g., group membership)

$$\{S\}\ B||F\ \{S\}$$

- Transients take system outside S, "corrector" C brings it back

$$C \models \Diamond S$$

- But B and C actually run concurrently and must not interfere with each other, so really need

$$\{S\}\ C||B||F\ \{S\}, \qquad C||B||F \models \Diamond S$$

- If C is part of B, only need prove B doesn't interfere with C
- Small complication that C only corrects to S'

Weakened Detectors and Correctors

- $\{S\}\, C||B||F\, \{S\}$
- $\{S'\}\, C||B||F\, \{S' \vee S\}$, and $S \supset S'$
- $C||B||F \models \Diamond S'$

Interpretation for TTA

- The base algorithm is group membership
- The corrector is clique avoidance
- The benign fault model is at most one symmetric fault every two rounds
- S is validity
  - All and only nonfaulty nodes in membership, except one faulty one is allowed during recovery

  and agreement
- S' sacrifices validity to ensure agreement
  - May exclude some nonfaulty nodes

Verification of Clique Avoidance

- Bauer and Paulitsch verify (by hand)

$$\{S'\}\ C||B||F\ \{S' \vee S\}$$

where F is a single asymmetric fault (SRDS '00)

- Bouajjani and Merceron verify (automatically)

$$C \models \Diamond S'$$

where the system starts from a state caused by a single asymmetric fault (multiple faults verified by hand)

- Challenge is to combine and extend these results

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 $p \rhd q$ means that if p is always true up to time t, then q holds at time $t+1$ (i.e., p fails before q)

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

- 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 (Ulm) is working on the same problem from a different direction

Startup/Restart

- When a node has heard nothing for a while, sends a wakeup message
- Other nodes may do same thing at the same time
- Collision detection is unreliable
- Needs self stabilization
- Should get clean wakeup after some small interval
- Need to prove this is achieved, in the presence of faults
  - Previous membership information is lost
- Involves transition between two models of computation (asynchronous to synchronous)
- Awaits formalization (some work at TU Vienna)

Replica Determinism as a System Service

- Strategies for fault-tolerant applications require that all nonfaulty replicas have the same state
- That is, have received same sequence of messages
- So need more than "best efforts" message delivery
- Need consensus (aka. Byzantine agreement, interactive consistency)
- Under weakest fault hypothesis (Byzantine) this sets lower bounds (to tolerate t simultaneous faults):
  - $3t + 1$ FCUs
  - $2t + 1$ disjoint comms paths, or $t + 1$ broadcast channels
  - $t + 1$ rounds of information exchange

Consensus

SAFEbus: Honeywell implementation has an extra communication channel; uses method of Davies and Wakerly

SPIDER: Has redundancy inside central hub; uses variation on Draper FTP algorithm

TTA:

- Provides Group Membership as basic service (assumes benign fault modes)
- With Clique Avoidance as NGU backup (on asymmetric faults)
- Provides Draconian Consensus (resembles Crusader Agreement) by eliminating receivers that disagree

Need to verify Draconian consensus and explain how it (apparently) violates known lower bounds

Top-Level Issues

- The individual algorithms are useful and interesting, but the real value of TTA is in the top-level properties that it provides
  - Partitioning
  - Time-triggered model of computation
- These are emergent: not found in any single algorithm

Partitioning

- The main issue for aircraft certification
- It's what allows several "functions" to be integrated on single platform (IMA and MAC architectures)
- Important dual attribute: strong composability
- Putative requirement specification for partitioning:
  - Behavior perceived by nonfaulty components must be consistent with some behavior of faulty components interacting with it through specified interfaces
- Need to formalize this
- And verify it for TTA
- The most difficult outstanding challenge?

The Time-Triggered Model of Computation

- Hermann Kopetz has a whole philosophy for this
  - Includes Temporal firewalls, composability arguments, elementary vs. compound interfaces. . .
- Tom Henzinger has Giotto: a time-triggered language, that provides some additional ideas
- Would like to give a formal account for this
  (cf. Paul Caspi's rational reconstruction for CriSys)
- I have verified that TTA supports the abstraction of synchronous system (TSE '99) but more is needed

Modular Certification

- How to certify components separately?
- And glue the arguments together?
- Certification differs from verification in that you have to take faults (hazards) seriously
- Trying assume-guarantee approach, based on normal and (multiple) abnormal assumptions and guarantees
- May help explain Perrow's concerns, and Kopetz' recommendation for elementary interfaces

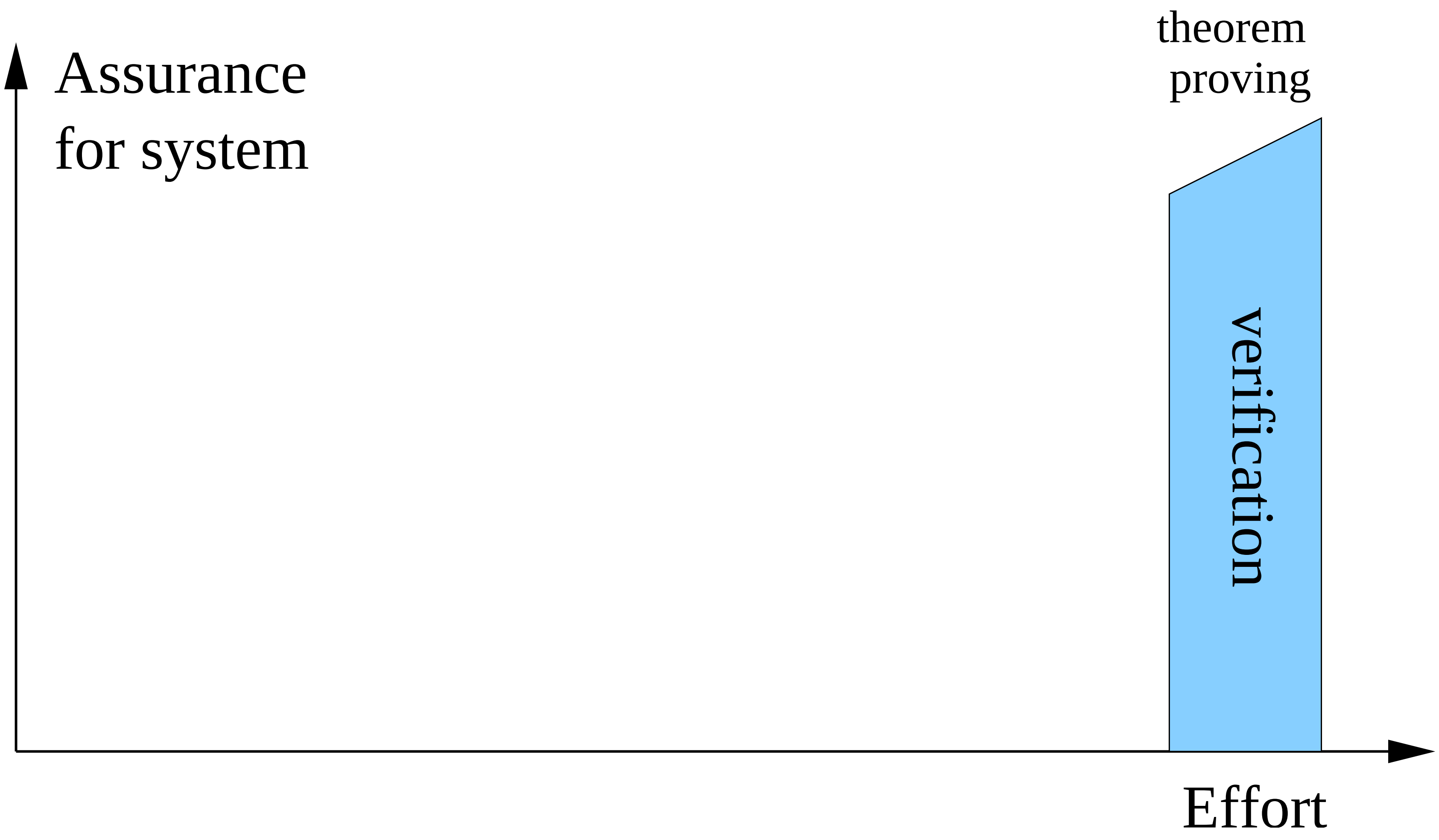
Utility of These Verifications?

- The completed verifications will have obvious utility in certification
- But the main benefits are sharpened statements of assumptions and properties
- And clarification of interactions and interdependencies among the algorithms
- Stimulates useful dialog with the designers of TTA
- And provides education for potential users of TTA
- Severe test of verification methods and automation

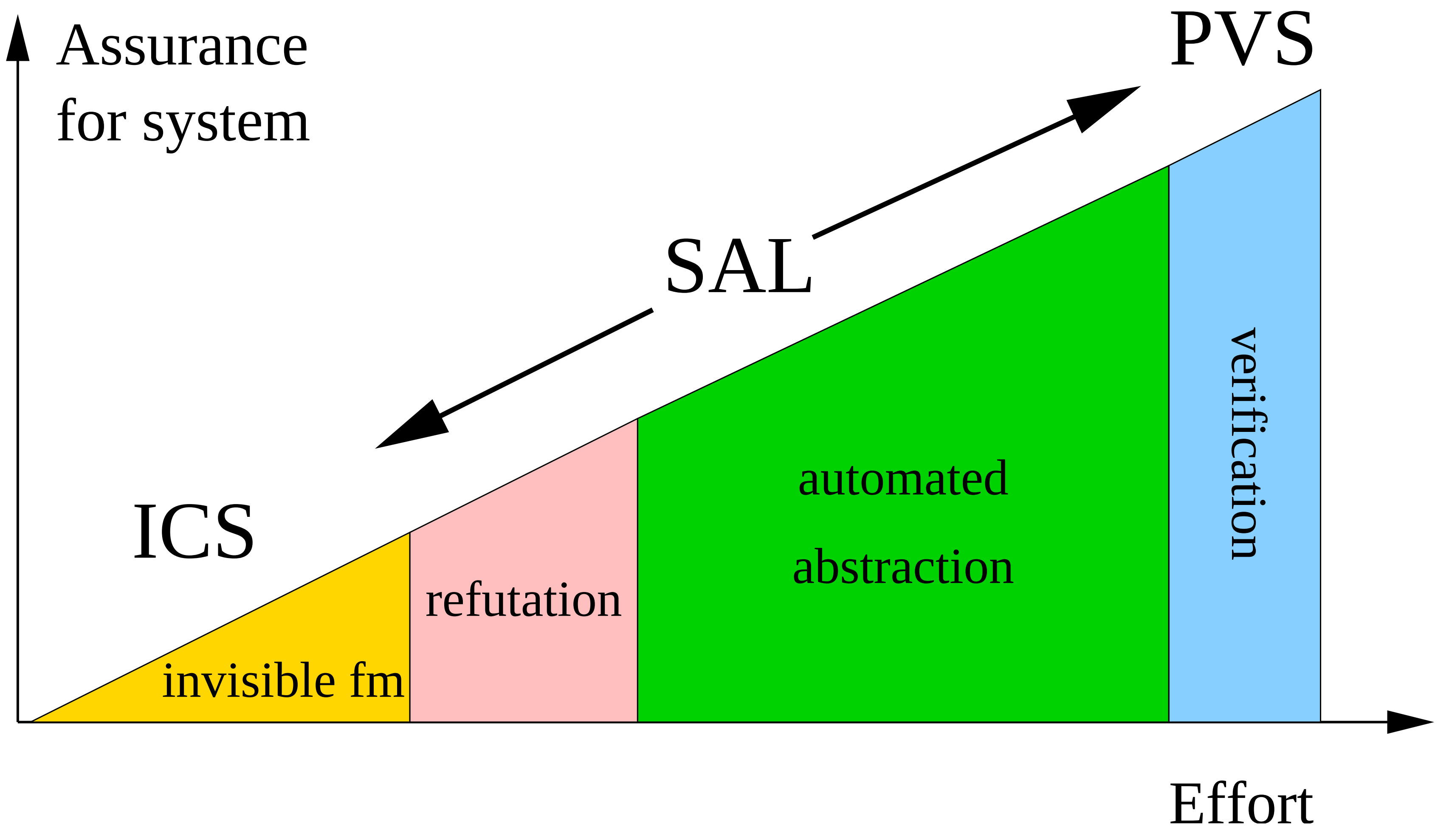
Next Steps

- Want to support developers of applications to run on TTA
- Should be able to verify their designs
  - Expressed in e.g., Lustre or Simulink
- And their transformation into fault-tolerant implementations running on TTA
- Formalization needs to be largely transparent
- And verification must be largely automatic
  - Need test vectors as well as formal proofs
- We cannot do all of this: concentrate on providing basic toolkits for others

The Wall of Formal Verification



A Smooth Slope of Formal Methods



Summary

- TTA is the last best hope for introducing rational fault-tolerance to distributed embedded systems
  - Displacing homespun solutions
- Analysis of its algorithms is a challenging and interesting problem for formal verification
  - But only needs to be done once
- Formalizing the computational model and properties presented to its client applications is crucial
- Can then bring formalization and verification to those clients
- In the form of “disappearing formal methods”