What Might A Science of Certification Look Like?

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

Overview

- Some tutorial introduction
- Implicit vs. explicit approaches to certification
- Making (software) certification "more scientific"
- Compositional certification

Certification

- Judgment that a system is adequately safe/secure/whatever for a given application in a given environment
- Based on a documented body of evidence that provides a convincing and valid argument that it is so
- Some fields separate these two
  - e.g., security: certification vs. evaluation
  - Evaluation may be neutral wrt. application and environment (especially for subsystems)
- Others bind them together
  - e.g., passenger airplane certification builds in assumptions about the application and environment
    - ⋆ Such as, no aerobatics—though Tex Johnston did a barrel roll (twice!) in a 707 at an airshow in 1955

View From Inside Inverted 707

During Tex Johnston's barrel roll

Certification vs. Evaluation

- I'll assume the gap between these is small
- And the evaluation takes the application and environment into account
- Otherwise the problem recurses
  - The system is the whole shebang, and evaluation is just providing evidence about a subsystem
- And I'll use the terms interchangeably

"System is Safe for Given Application and Environment"

- So it's a system property
  - e.g., the FAA certifies only airplanes and engines (and propellers)
- Can substitute secure, or whatever, for safe
  - Invariably these are about absence of harm
- So, generically, certification is about controlling the downsides of system deployment
- Which means that you know what the downsides are
  - And how they could come about
  - And you have controlled them in some way
  - And you have credible evidence that you've done so

Knowing What the Downsides Are And How They Could Come About

- The problem of "unbounded relevance" (Anthony Hall)
- There are systematic ways for trying to bound and explore the space of relevant possibilities
  - Hazard analysis
  - Fault tree analysis
  - Failure modes and effects (and criticality) analysis: FMEA (FMECA)
  - HAZOP (use of guidewords)
- These are described in industry-specific documents
  - e.g., SAE ARP 4761, ARP 4754 for aerospace

Controlling The Downsides

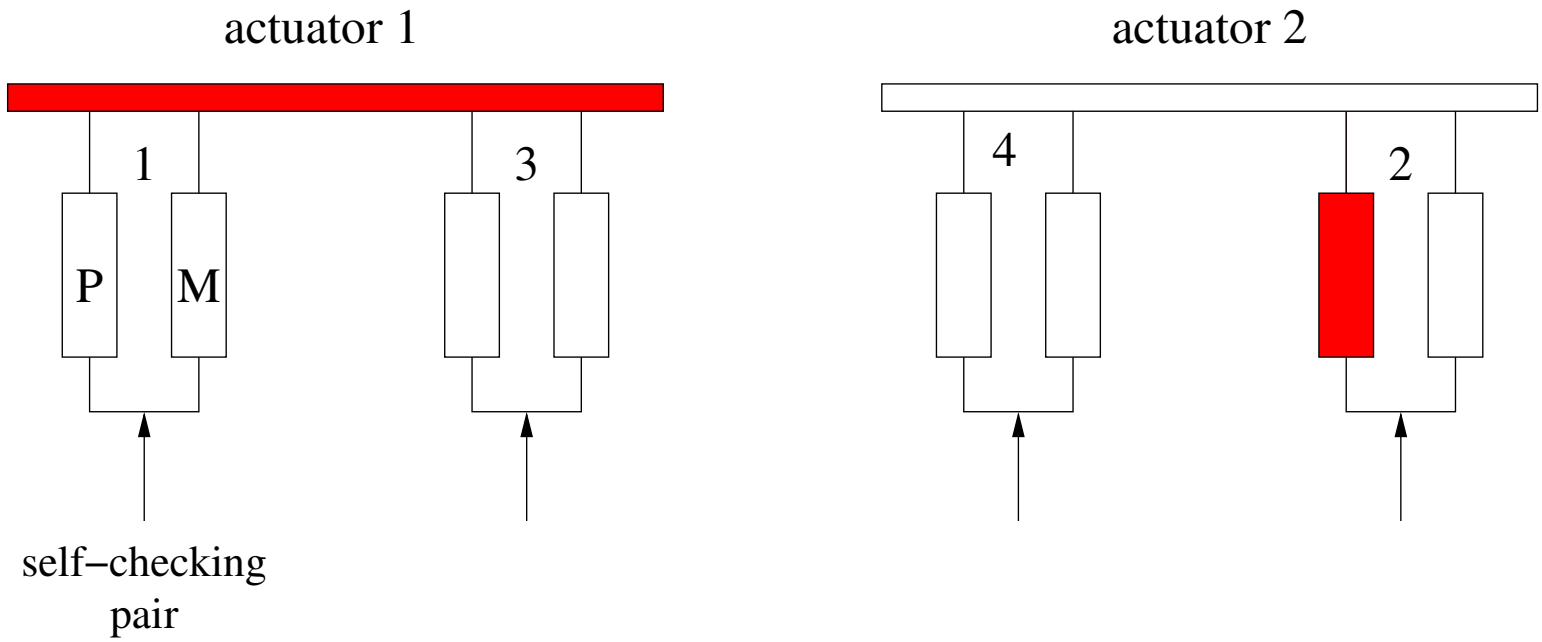
- Downsides are usually ranked by severity
  - e.g. catastrophic failure conditions for aircraft are "those which would prevent continued safe flight and landing"
- And an inverse relationship is required between severity and frequency
  - Catastrophic failures must be "so unlikely that they are not anticipated to occur during the entire operational life of all airplanes of the type"

Subsystems

- Hazards, their severities, and their required (im)probability of occurrence flow down through a design into its subsystems
- The design process iterates to best manage these
- And allocates hazard "budgets" to subsystems
  - e.g., no hull loss in lifetime of fleet, 10^7 hours for fleet lifetime, 10 possible catastrophic failure conditions in each of 10 subsystems, yields allocated failure probability of 10^{-9} per hour for each
- Another approach could require the new system to do no worse than the one it's replacing
  - e.g., in 1960, big jets averaged 2 fatal accidents per 10^6 hours; this improved to 0.5 by 1980 and was projected to reach 0.3 by 1990; so set the target at 0.1 (10^{-7}), then subsystem calculation as above yields 10^{-9} per hour again

Design Iteration

- Might choose to use self-checking pairs to mask both computer and actuator faults
- Must tolerate one actuator fault and one computer fault simultaneously



- Can take up to four frames to recover control

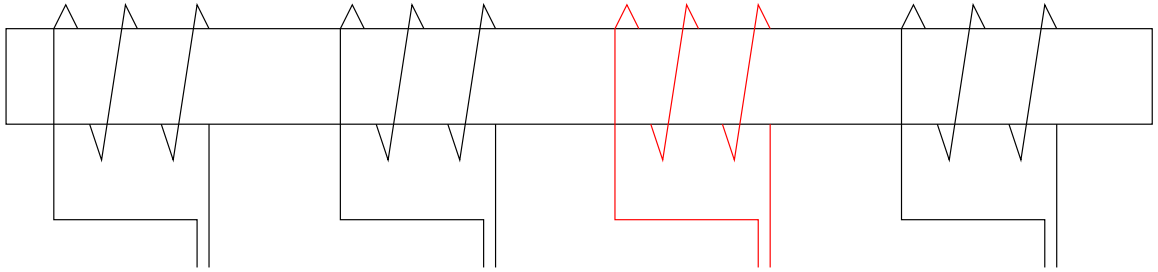
Consequences of Slow Recovery

- Use large, slow moving ailerons rather than small, fast ones
- As a result, wing is structurally inferior
- Holds less fuel
- And plane has inferior flying qualities
- All from a choice about how to do fault tolerance

Design Iteration: Physical Averaging At The Actuators

An alternative design uses averaging at the actuators

- e.g., multiple coils on a single solenoid



- Or multiple pistons in a single hydraulic pot

Design Margin and Redundancy

- Can often calculate the stresses on physical components
- May then sometimes be able to build in a safety margin
  - e.g., airplane wing must take 1.5 times maximum expected load
- In other cases, historical experience yields failure rates
- Can tolerate these through redundancy
  - e.g., multiple hydraulic systems on an aircraft
- And can calculate probabilities
  - Assuming no common mode failures
  - i.e., no overlooked design flaws

Design Failure

- Possibility of residual design faults is seldom considered for physical systems
  - Relatively simple designs, much experience, accurate models, massive testing of the actual product
- But it still can happen
  - e.g., 737 rudder actuator

  Especially when redundancy adds complexity
- But software is nothing but design
- And it is often complex
- So, can we tolerate software design faults, or must we eliminate them?

Diversity As Defense For Design Faults?

- Use of redundancy to tolerate faults rests on the assumption of independent failures
- Achievable when physical failures only are considered
- To control common mode failures, may sometimes use diverse mechanisms
  - e.g., ram air turbine for emergency hydraulic power
- And some advocate software redundancy with design diversity to counter software flaws
- Many arguments against this
  - Need diversity all the way up the design hierarchy
  - Diverse designs often have correlated failures
  - Better to spend three times as much on one good design
- So usually must show that software is free of design faults

Software Certification

- Software is usually certified only in a systems context
- Hazards flow down to establish properties that must be guaranteed, and their criticalities
  - Unrequested function
  - And malfunction
  - Are generally more serious than loss of function
- How to establish satisfaction of such requirements?
- Generally try to show that software is free of design faults
- Try harder for more software critical components
  - i.e., for higher software integrity levels (SILs)

Approaches to System and Software Certification

The implicit standards-based approach

- e.g., airborne s/w (DO-178B), security (Common Criteria)
- Follow a prescribed method
- Deliver prescribed outputs
  - e.g., documented requirements, designs, analyses, tests and outcomes, traceability among these
- Internal (DERs) and/or external (NIAP) review

Works well in fields that are stable or change slowly

- Can institutionalize lessons learned, best practice
  - e.g. evolution of DO-178 from A to B to C (in progress)

But less suitable when novelty in problems, solutions, methods

Implicit that the prescribed processes achieve the safety goals

Does The Implicit Approach Work?

- Fuel emergency on Airbus A340-642, G-VATL, on 8 February 2005 (AAIB SPECIAL Bulletin S1/2005)
- Two Fuel Control Monitoring Computers (FCMCs) on this type of airplane; they cross-compare and the "healthiest" one drives the outputs to the data bus
- Both FCMCs had fault indications, and one of them was unable to drive the data bus
- Unfortunately, this one was judged the healthiest and was given control of the bus even though it could not exercise it
- Further backup systems were not invoked because the FCMCs indicated they were not both failed

Approaches to System and Software Certification (ctd.)

The explicit goal based approach

- e.g., aircraft, air traffic management (CAP670 SW01), ships

Applicant develops an assurance case

- Whose outline form may be specified by standards or regulation (e.g., MOD DefStan 00-56)
- The case is evaluated by independent assessors

An assurance case

- Makes an explicit set of goals or claims
- Provides supporting evidence for the claims
- And arguments that link the evidence to the claims
  - Make clear the underlying assumptions and judgments
- Should allow different viewpoints and levels of detail

Evidence and Arguments

Evidence can be facts, assumptions, or sub-claims
(from a lower level argument)

Arguments can be

Analytic: can be repeated and checked by others, and potentially by machine

- e.g., logical proofs, calculations, tests
- Probabilistic (quantitative statistical) reasoning is a special case

Reviews: based on human judgment and consensus

- e.g., code walkthroughs

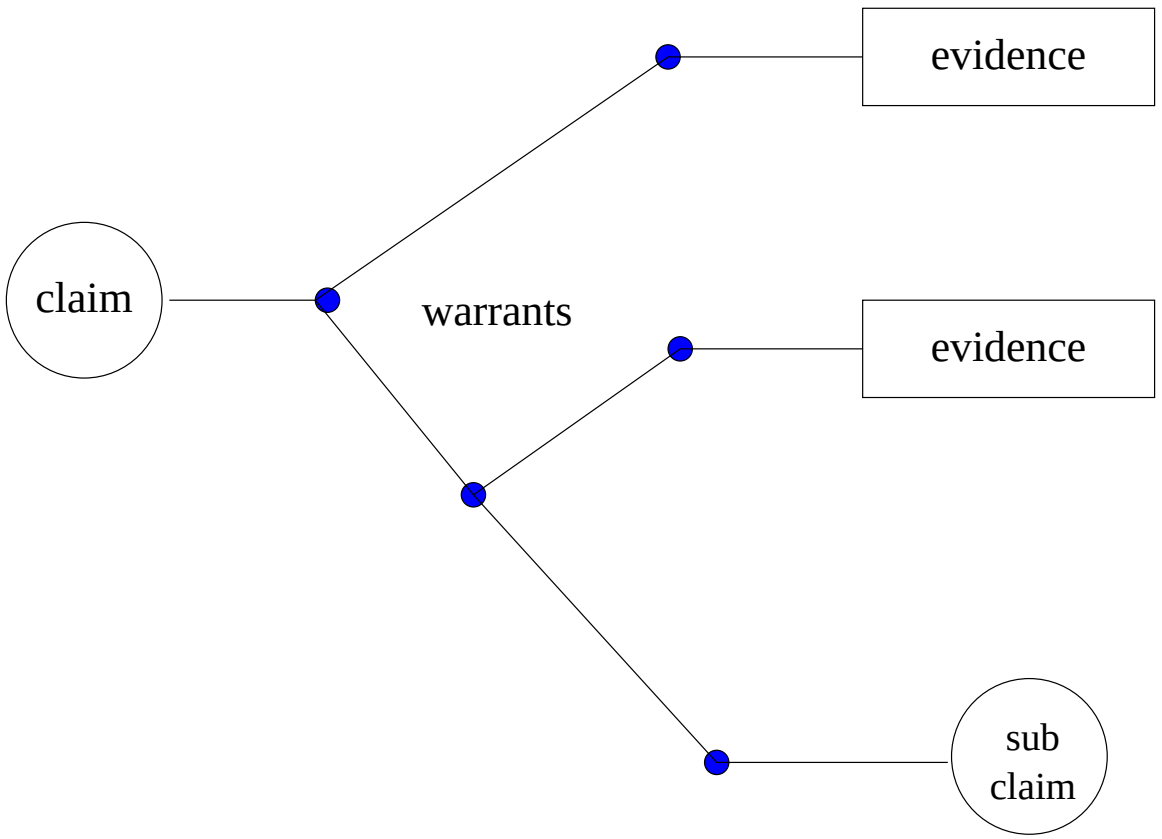
Qualitative/Indirect: establish only indirect links from evidence to desired attributes

- e.g., CMI levels, staff skills and experience

Toulmin Arguments

- Not all the arguments in an assurance case are of the strictly logical kind
  - The local argument that links some evidence into the claim is called a warrant
- And the overall argument uses warrants of several kinds
- So this style of argument is not of the kind considered in classical (formalized) logic
  - Though I suspect it can be formalized using additional inference rules (e.g., "because experience says so")
- Advocates of assurance cases generally look to Toulmin for guidance on argument structure
  - "The Uses of Argument" (1958)
  - Stresses justification rather than inference

Argument Structure for an Assurance Case



Making Certification "More Scientific"

- We could start by favoring explicit over implicit approaches
- At the very least, expose and examine the arguments and assumptions implicit in the standards-based approaches
- Many of them turn out to be indirect
  - Requirements for "safe subsets" of C, C++ and other coding standards (JSF standard is a 1 mbyte Word file)
  - Follow certain design practices
- No evidence many are effective, some contrary evidence
- Others impose qualitative selections on analysis and reviews to be performed, or on the degree of their performance
  - Formal specifications at higher EAL levels
  - MC/DC tests for DO-178B Level A

  Little evidence which are effective, nor that more is better

Critique of Standards-Based Approaches

- Too much focus on the process, not enough on the product
  - "Because we cannot demonstrate how well we've done, we'll show how hard we've tried"
- Some explicit processes are required to establish traceability
  - So we can be sure that it was this version of the code that passed those tests, and they were derived from that set of requirements which were partly derived from that fault tree analysis of this subsystem architecture

Making Certification "More Scientific" (ctd.)

Replace indirect warrants and qualitative selections of reviews and analyses by analytic warrants that support sub-claims of a form that can feed into a largely analytic argument structure a higher levels

- Statistically valid testing (delivers about 10^{-4} max)
- Static analysis for absence of runtime errors
- Automated formal code and design verification
  - Exposes assumptions that feed upper levels of analysis
- Automated formal support for FMEA, human interaction errors, and other aspects of hazard analysis
- Automated formal support for hierarchical arguments (replace Toulmin?)

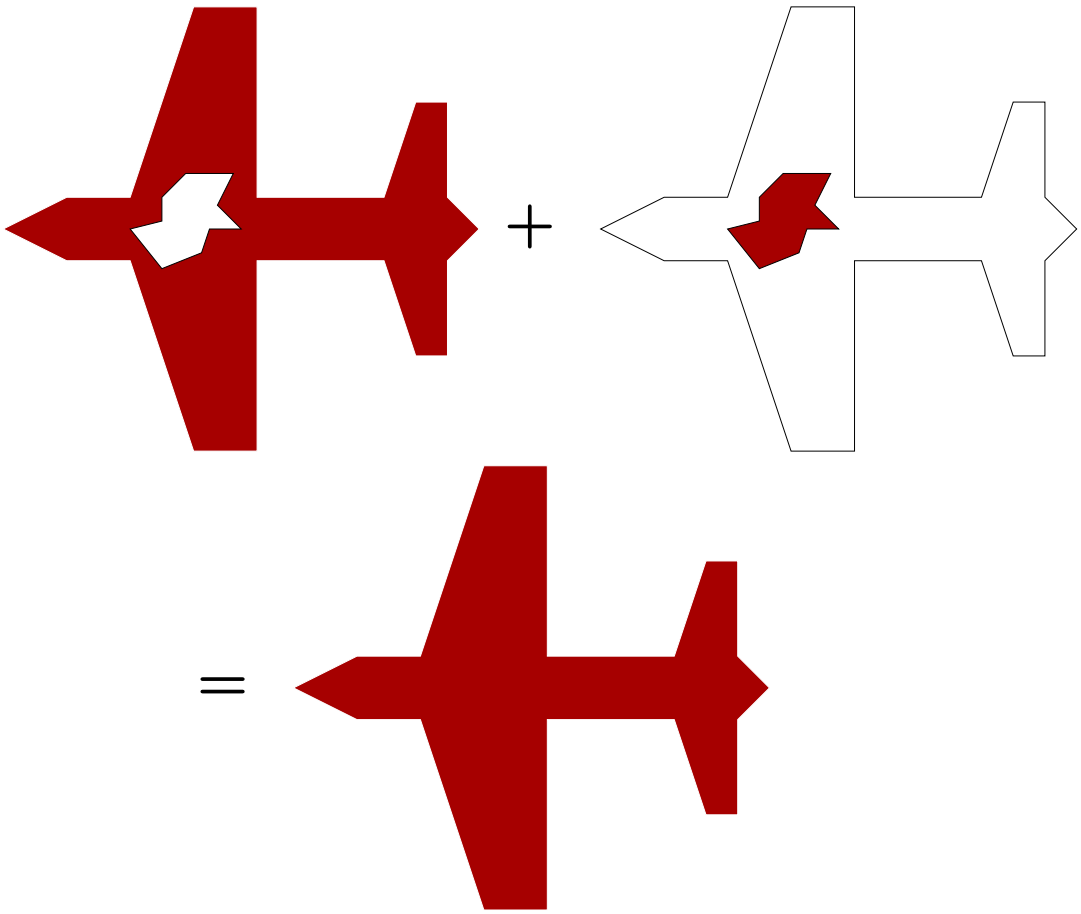
Automated Formal Methods

- I've referred to formal methods simply because that is the applied math of computer systems
- Just as PDEs are the applied math of aerodynamics
- Automated formal methods perform logical calculations, just as CFD performs numerical calculations
  - Soundness is important, and so are performance and capacity: cannot sacrifice one for another
- Approach FV as engineering calculations and focus on delivering maximum value to the overall certification process
  - It's not necessary to prove everything
  - It's not necessary for proofs to be reduced to trivial steps ('cos that's not where the problems are)
  - Though it's ok to do these things if the economics work

Scientific Certification

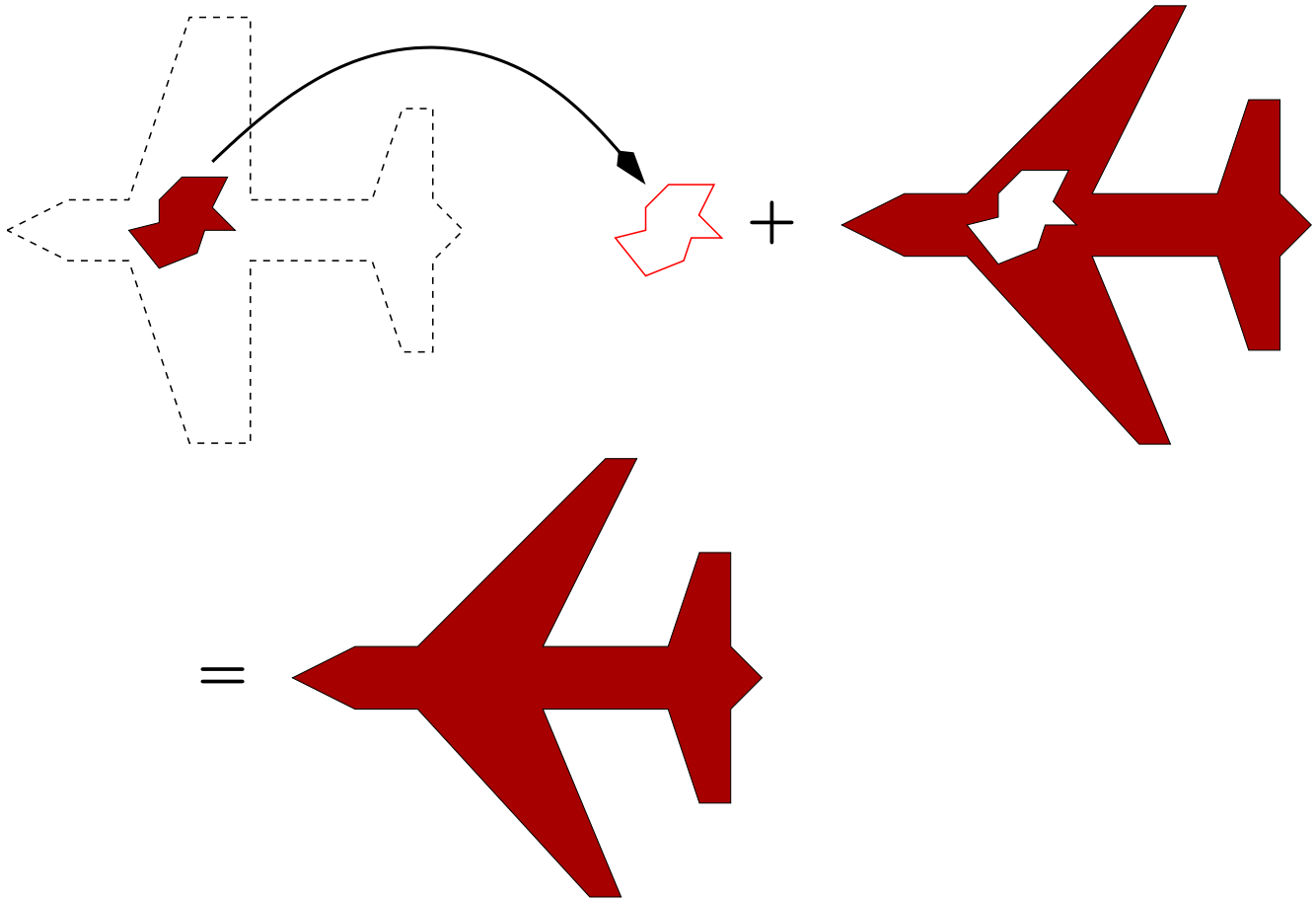
- I've outlined a proposal for a "science of certification"
- A lot of work will be needed to fully develop this
- In particular, a full science of certification:
  - Will need to be hierarchical
  - And incremental
    - ⋆ An issue here is that certification may be nonmonotonic: an analytic warrant that system S satisfies property P in environment E may be invalidated when a new element X is incrementally added to S
  - And compositional
- Let's look at the last of these

Traditional Approach is Noncompositional



Requirements, analyses, flow down, go inside components

A Compositional Certification Approach



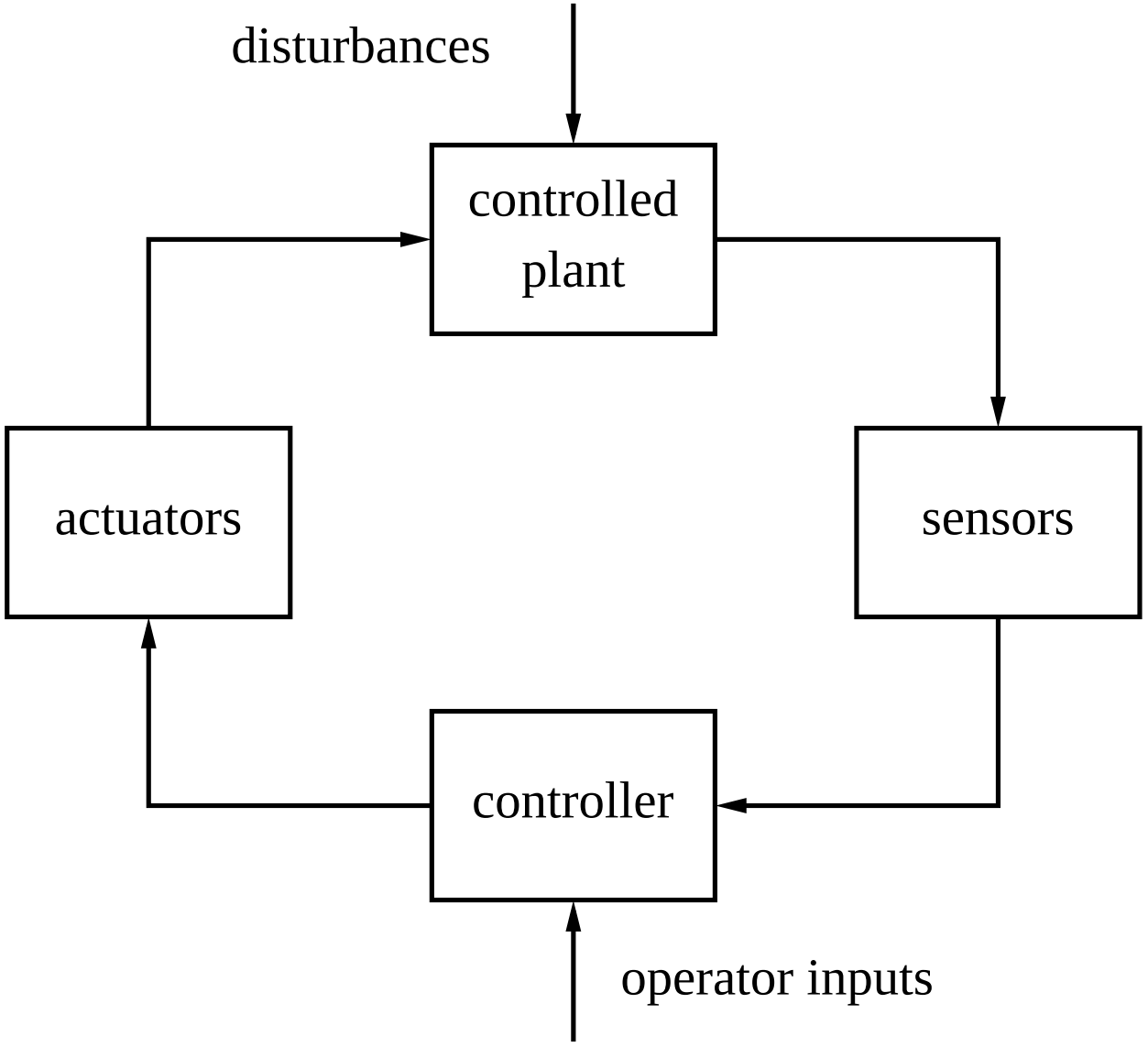
Requirements, analyses stop at component interface and do not go inside: e.g., an RTOS

Old and New Issues

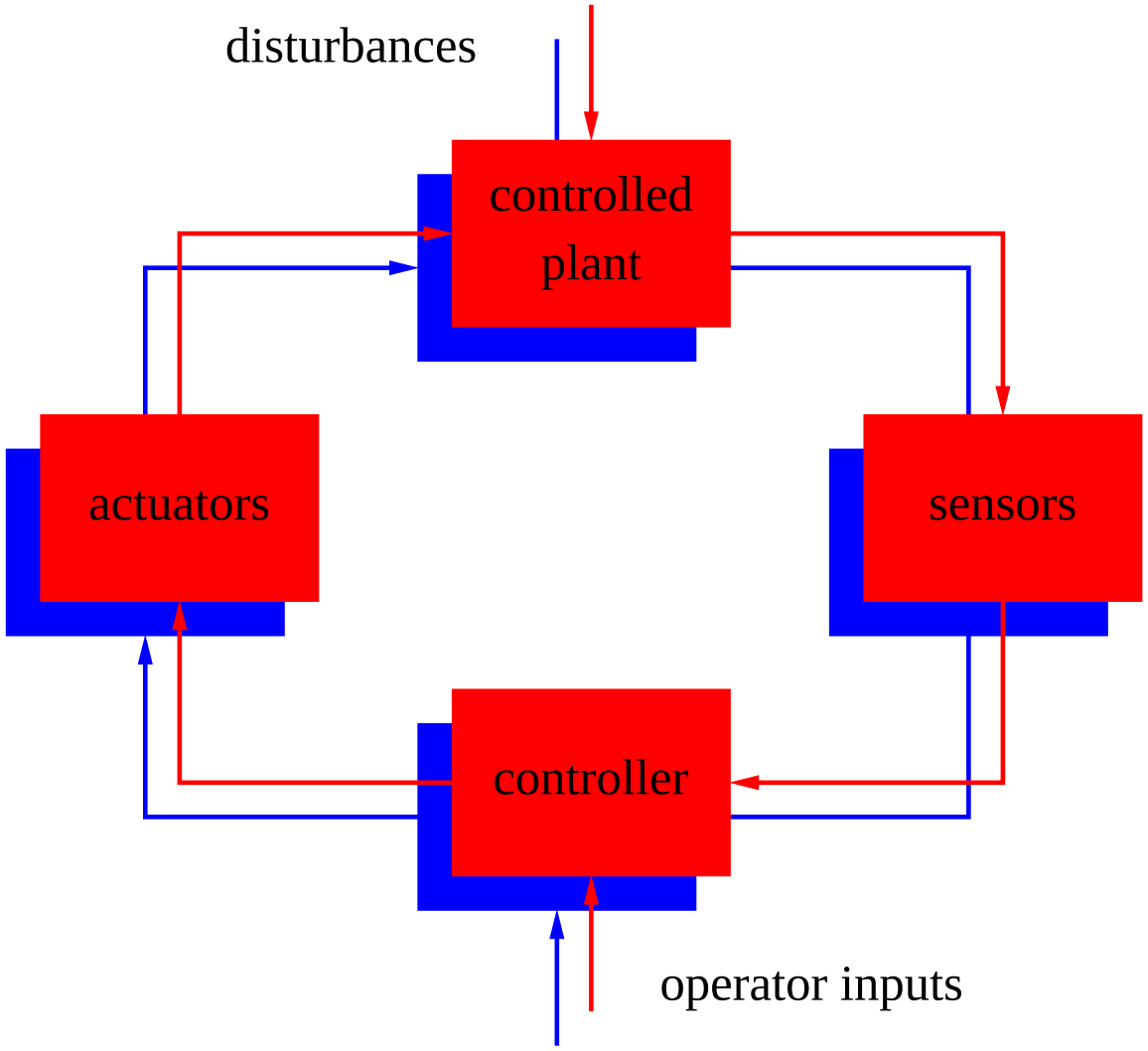
- Does this component do its own thing safely and correctly?
  Standard assurance methods can take care of this
- Could this component (perhaps due to malfunction) stop some other component doing its thing safely and correctly?
  This is the new: you may not yet know what the other components are
  Three ways one component can adversely affect another
  - Monopolize or corrupt shared resources
  - Interact improperly (send bad data, fail to follow protocol)
  - Generate a hazard through coupling of the plants (e.g., thrust reverser moves when engine thrust is above idle)

  Consider each of these in turn

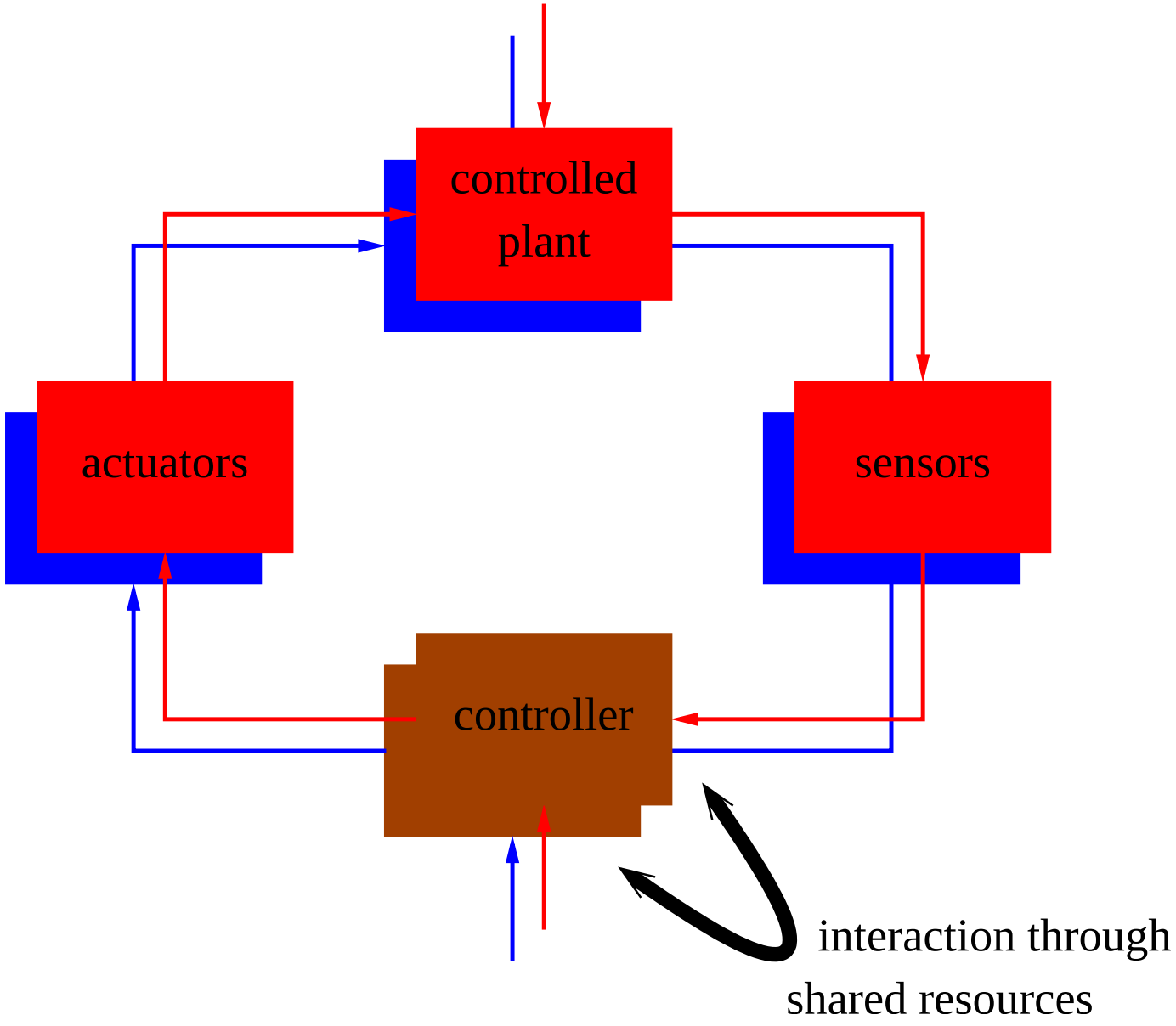
Context: Generic Embedded Control System



Now Suppose We have Two Of Them



Unintended Interaction Through Shared Resources



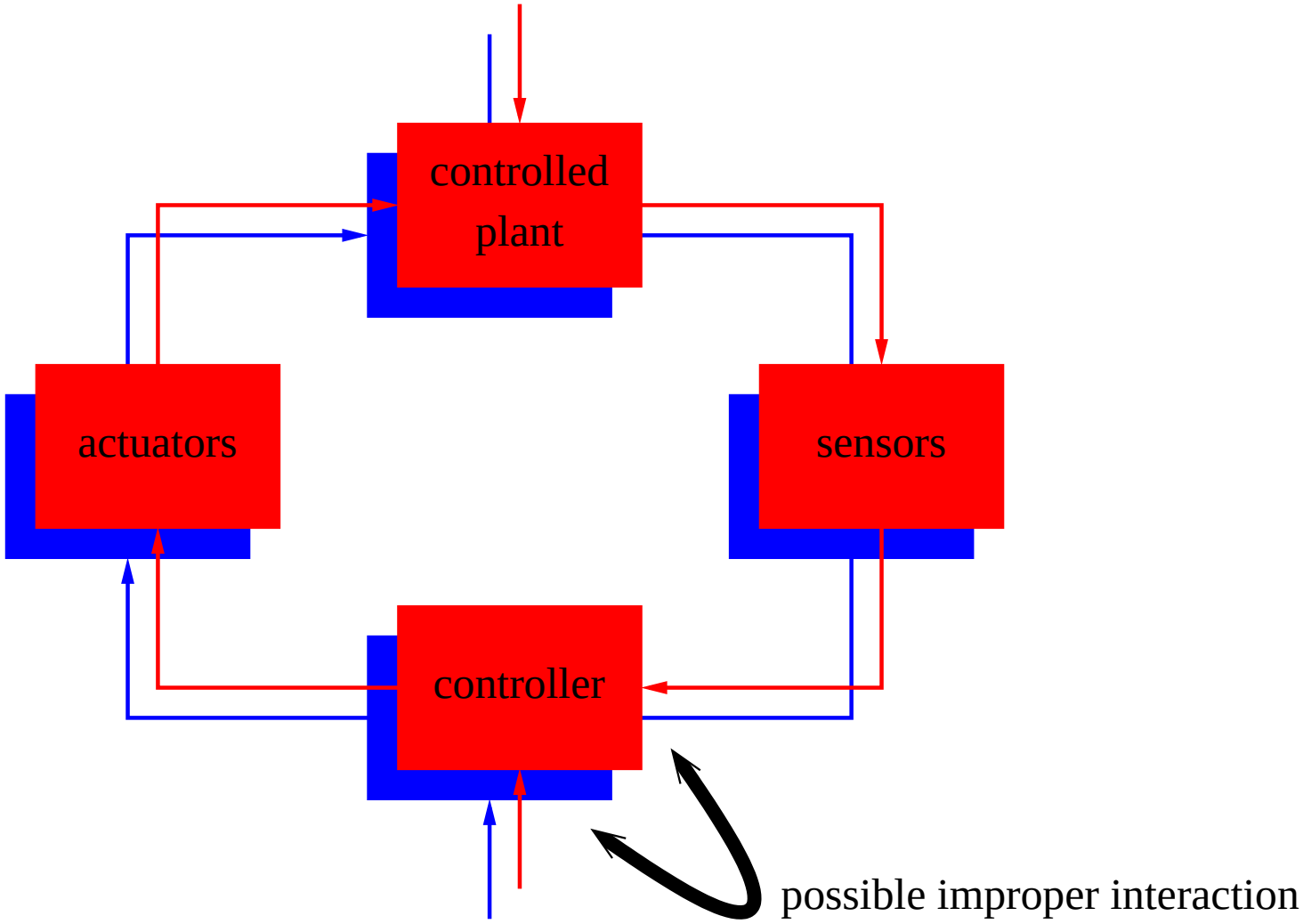
The Architecture Must Enforce Partitioning

- One component (even if faulty) cannot be allowed to affect the operation of another through shared resources
  - Write to its memory, devices
  - Grab locks, CPU time
  - Collide on access to shared bus, devices
- Components cannot guarantee this themselves—it's a property of the architecture in which they operate
- This is partitioning
- Whole idea of compositional certification is that you can understand interactions of components by considering their specified interfaces
- So partitioning is about enforcing interfaces

Partitioning

- Top-level requirement specification for partitioning:
  - Behavior perceived by nonfaulty components must be consistent with some behavior of faulty components interacting with it through specified interfaces
- Federated architecture ensures this by physical means
- IMA or MAC architectures such as Primus Epic or TTA must ensure this by logical means
- For single processors, space partitioning is standard O/S technology: memory management, virtual machines, etc.
- Time partitioning can get tricky if using dynamic scheduling with locks, budgets, slack time, etc.
  - Recall failures of Mars Pathfinder (priority inversions)

Improper Interaction Through Intended Channels



Assumptions Underlie Interactions

- Some components are intended to interact
- E.g., One may pass data to another
- Implicitly, each assumes something about the behavior of the other
- If those assumptions are violated (or not explicitly recorded and managed), component may fail
- Violation is most likely when other component is in failure condition
- Can then get uncontrolled fault propagation
- Recall loss of Ariane 501

Ariane 501

- Inertial reference systems reused from Ariane 4, where they had worked well
- Greater horizontal velocity of Ariane 5 led to arithmetic overflow in alignment function
- Flight control system switched to the other inertial system
- But that had failed for the same reason
- No provision for second failure (assumed only random faults)
- So flight control system interpreted diagnostic output as flight data
- Led to full nozzle deflections of solid boosters
- And destruction of vehicle and payload

Ariane 501 (continued)

- Everyone sees Ariane 501 as justifying their own favorite issue/technique/tool
- However, it was really a failure to properly record interface assumptions
- Assumption in IRS about max horizontal velocity during alignment
- Assumption at system level that failure indication from IRS could only be due to random hardware faults
  - And double failure was therefore improbable

Assumptions and Guarantees

- Components make assumptions about other components
- And guarantee what other components can assume about them
- This is assume/guarantee reasoning
- Looks circular but computer scientists know how to do it soundly
- In compositional certification, we need to extend this to failure conditions

Normal and Abnormal Assumptions and Guarantees

- In most concurrent programs one component cannot work without the other
  - e.g., in a communications protocol, what can the sender do without a receiver?
- But the software of different aircraft functions should not be so interdependent
  - In the limit should not depend on others at all
  - Must provide safe operation of its function in the absence of any guarantees from others
  - Though may need to assume some properties of the function controlled by others (e.g., thrust reverser may not depend on the software in the engine controller, but may depend on engine remaining under control)

Normal and Abnormal Assumptions and Guarantees (ctd)

- Component should provide a graduated series of guarantees, contingent on a similar series of assumptions about others
  - These can be considered its normal behavior and one or more abnormal behaviors
- Component may be subjected to external failures of one or more of the components with which it interacts
  - Recorded in its abnormal assumptions on those components
- Component may also suffer internal failures
  - Documented as its internal fault hypothesis
- Hypotheses must encompass all possible faults
  - Including arbitrary or Byzantine faults
  - Unless these can be shown infeasible (e.g., masked by partitioning architecture)

Components Must Meet Their Guarantees

True guarantees: under all combinations of failures consistent with its internal fault hypothesis and abnormal assumptions, the component must be shown to satisfy one or more of its normal or abnormal guarantees.

Safe function: under all combinations of faults consistent with its internal fault hypothesis and abnormal assumptions, the component must be shown to perform its function safely

- e.g., if it is an engine controller, it must control the engine safely
- Where "safely" means behavior consistent with the safety case assumption about the function concerned

Avoiding Domino Failures

- If component A suffers a failure that causes its behavior to revert from guarantee $G(A)$ to $G'(A)$
- May expect that B's behavior will revert from $G(B)$ to $G'(B)$
- Do not want the lowering of B's guarantee to cause a further regression of A from $G'(A)$ to $G''(A)$ and so on

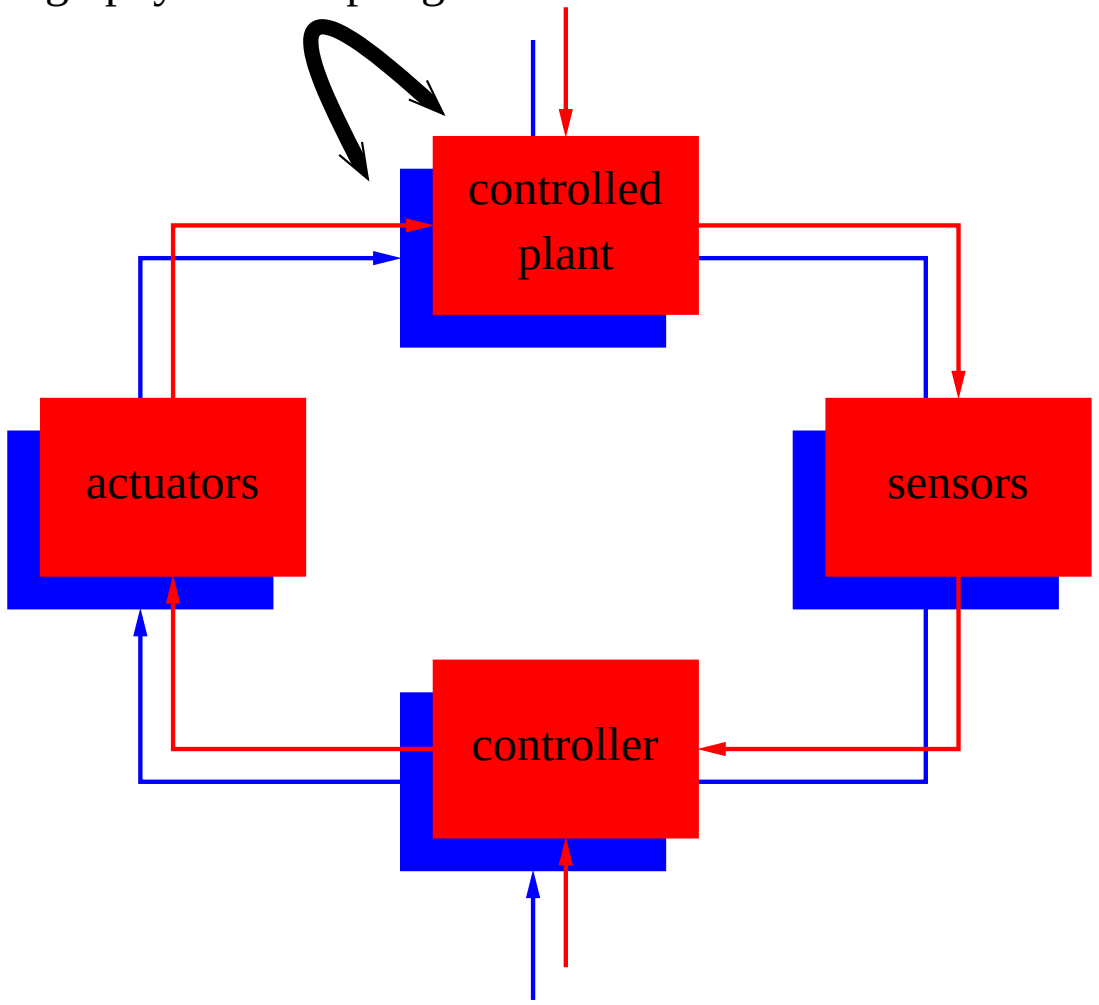
  Controlled failure: there should be no domino effect.

  > Arrange assumptions and guarantees in a hierarchy from 0 (no failure) to j (rock bottom). If all internal faults and all external guarantees are at level i or better, component should deliver its guarantees at level i or better

  This subsumes true guarantees

Coupling Through The Plants

potential interaction
through physical coupling



Interaction Through Physical Coupling Of The Plants

- Must be examined by the techniques of hazard analysis
  - FHA (Functional hazard analysis), HAZOP, etc.
  - Cf. ARP 4754, 4761
- E.g., engine controller and thrust reverser
  - No reverse thrust when in flight (system-component)
  - No movement of the reverser doors when thrust above flight idle (component-component)
- No avoiding the need for holistic analysis here
  - Though some component-system and component-component analyses may be routine
- Results feed back into requirements on safe function for the controllers concerned

Summary of Compositional Certification

- Compositional certification depends on controlling and understanding interactions among components
- Interactions must be restricted to known interfaces through partitioning
- Interactions through those interfaces should be documented as assumptions and guarantees
- These must be extended to failure (abnormal) conditions
- Coupling through the plants must be analyzed and added to the requirements on safe function
- Then provide assurance for
  - Partitioning
  - True guarantees/controlled failure
  - Safe function

Overall Summary

- Explicit goal-based assurance cases seem to offer the best foundation for a science of certification
- Scientific certification will stress analytic warrants
- Which, for software, will use automated formal methods
- To learn more about assurance cases, visit www.adelard.co.uk
- Plenty of opportunities for research
- And for constructive dialog with standards and formal methods communities
  - e.g., SC205, VSR