Modular Certification Research Activities, FAA National Software Conference, 16–19 September 2003, Reno NV

Modular Certification Research Activities

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

Overview

- Modular certification: what is it?
- Research activities and an aside on automated verification
- Partitioning
- Assume/Guarantee analysis and domino failures
- Coupling through the plants
- Summary

Modular Certification: What Is It?

- Currently: certify the whole aircraft
  - Suppose some software was used in another aircraft?
  - Informal reuse of certification argument
  - Now being formalized in AC20-RSC
- ModCert: software comes with qualification data
  - Modules may be bigger, more interdependent than contemplated for RSC
- Full certification examines the qualification data but does not go inside the design
- Requires that
  the qualification data provides everything you need to know
- For further qualification or certification,
  the qualification data is as good as the actual thing

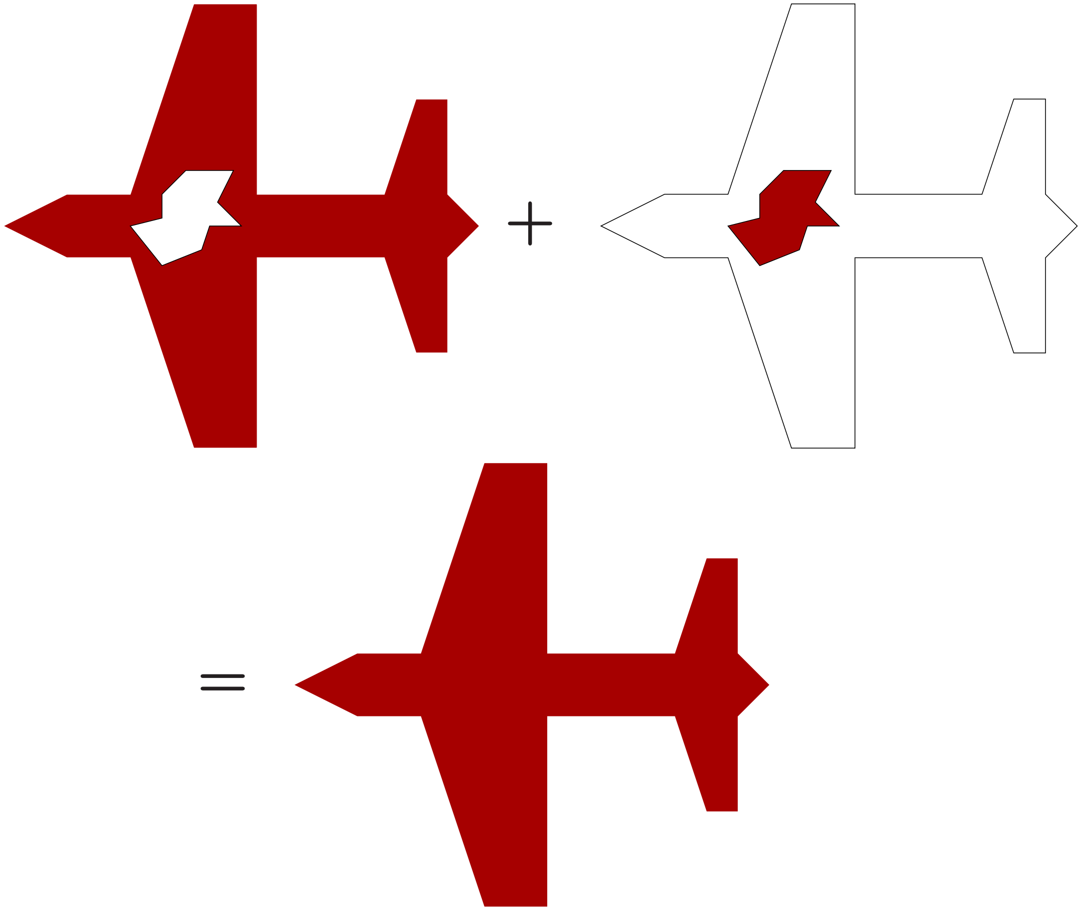
Modular Certification: Deliberations

- RTCA Special Committee 200
- And Eurocae Working Group 60
- Have joined forces to develop guidance for (Integrated) Modular Avionics
- Goal is a document DO-xxx/ED-yyy by March 2004
- "The document should promote aircraft systems design using modular avionics by developing guidance for regulatory approval of the platform and supporting components through the assurance of safety and performance requirements, with attention to means of organizing data developed and used by multiple stakeholders"

Modular Certification: Why Is It Wanted?

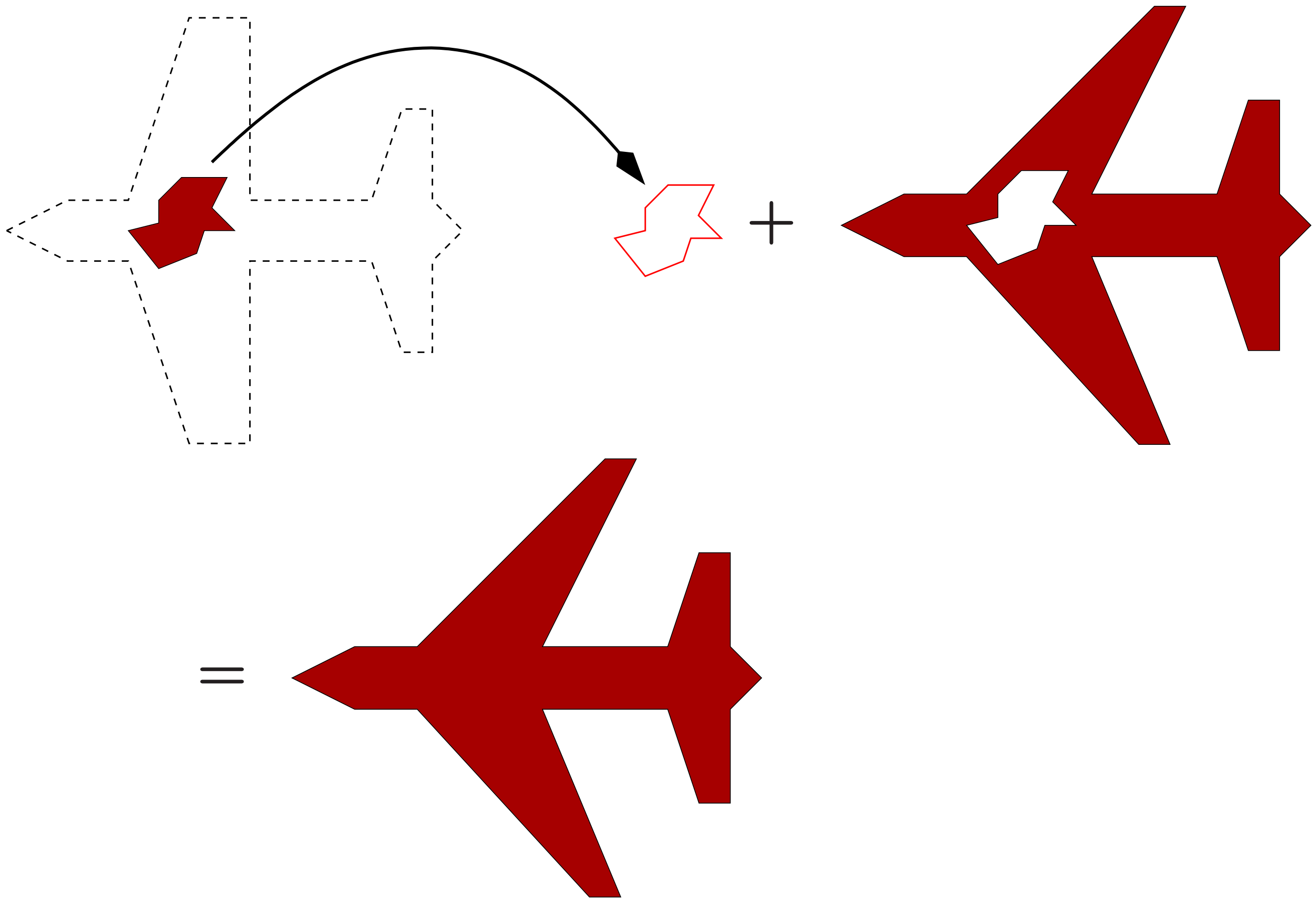
- An enabler for Modular Avionics (MA)
  - And a different business model that might go with it
  - Module supplier owns the qualification data
- And for Integrated Modular Avionics (IMA)
- And for Incremental Certification

Traditional Approach



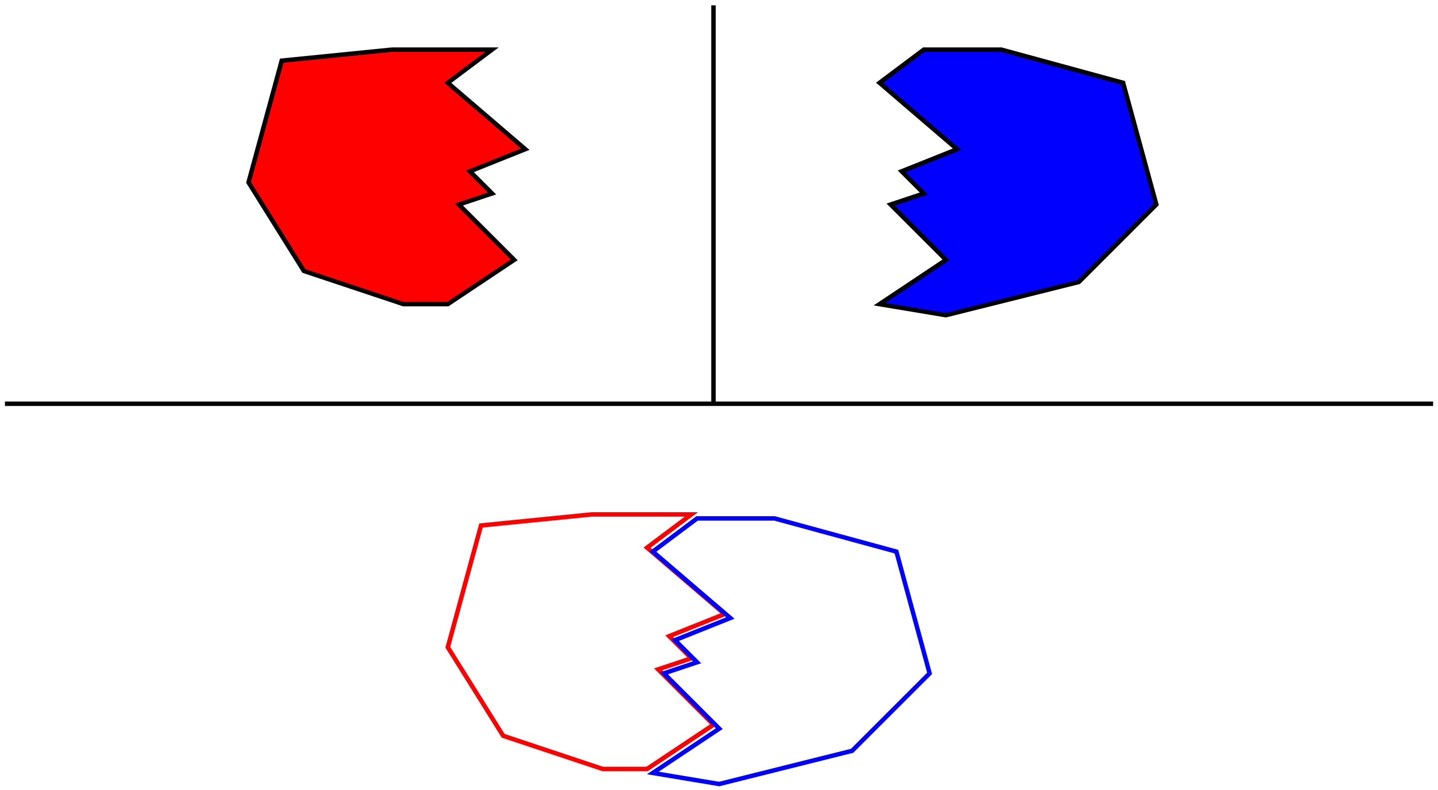
Requirements, analyses, flow down, go inside components

Modular Certification Approach



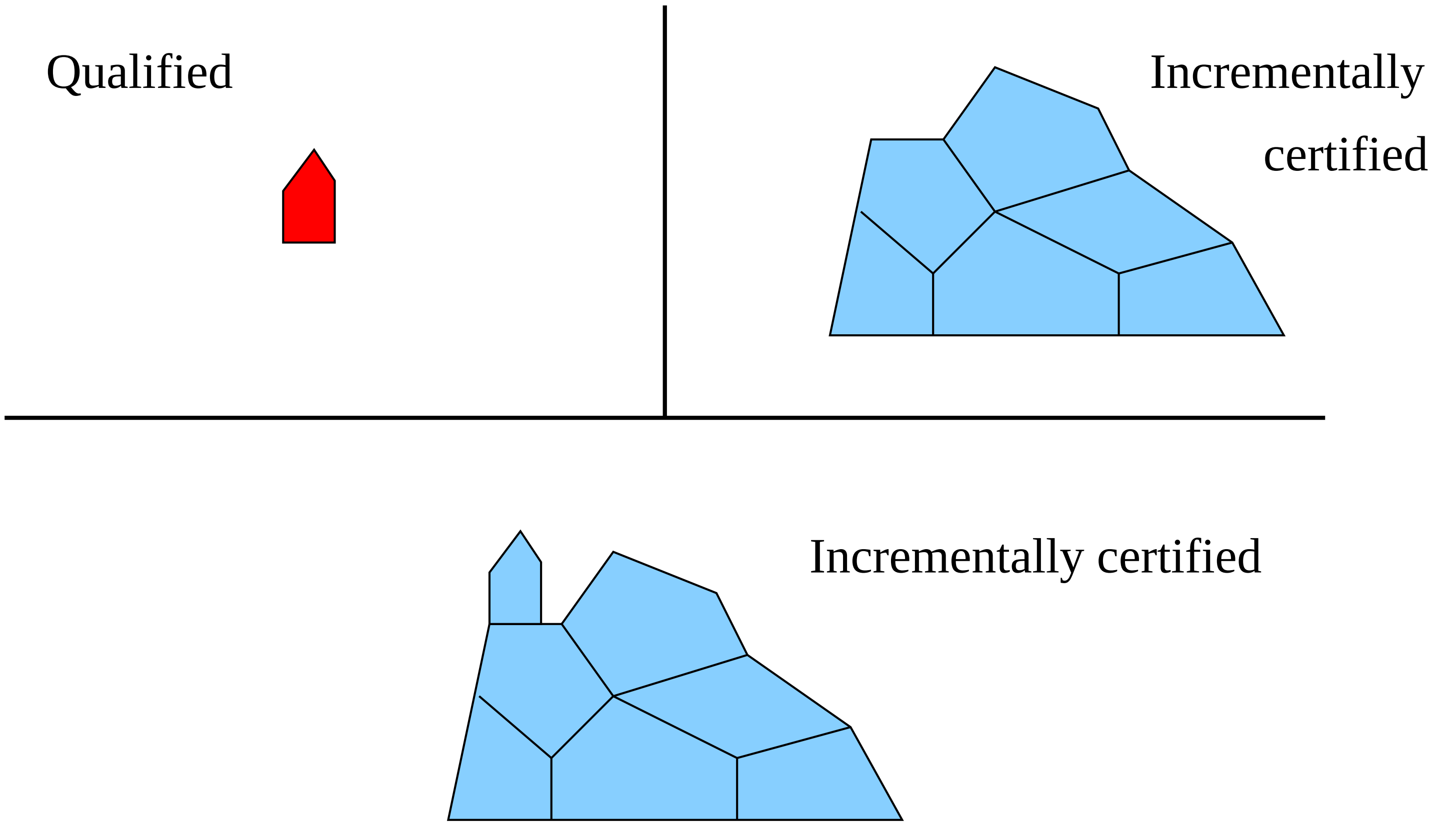
Requirements, analyses stop at component interface and do not go inside: e.g., an RTOS

Integrated Modular Avionics



Separately qualified modules can be combined and used without going inside their designs: e.g., a fadec and a thrust reverser

Incremental Certification



Research Activity in Support of Modular Certification

- Sponsored by NASA
- In-house work at NASA Langley
  - Wednesday: Paul Miner on Spider
  - But focus of that session was DO254
- Honeywell/SRI/TTTech
  - Modular Aerospace Controls (MAC) architecture
  - Engine controller for Aeromachi trainer, F16
  - Based on Time Triggered Architecture (TTA)

  The work I'm reporting is from here

Goals of Research Activity

- Develop computer science basis for modular certification
    - What do I need to know about components
    - And the way they are put together
    - To be sure the whole thing works safely?
    - And how can I verify these things?
    - Want to push the envelope
- This kind of computer science is formal methods
- But before you all leave. . .
- The goal is not to impose formal methods
- But to use it to develop techniques and automated tools that might actually be useful

Aside: Formal Methods vs. Automated Verification

- Formal methods is not a religion
  - But it does have many sects
- All want to treat software as mathematics
- Some because they think that's good for you
- Others because you can calculate with mathematics
  - This is automated verification
  - Just as CFD calculates the lift and drag of a wing
  - So automated verification calculates properties of software

Automated Verification

- Is routine in some areas
  - Processor design, for example
- There's a tradeoff between how automated it is
- And how deep the properties are that you can get to
- But there's tremendous progress year by year

Automated Verification (ctd)

Fully automated: (in principle)

- Could there be a runtime exception?
- Is this piece of code reachable?
- Unit test case generation
  (e.g., MC/DC coverage of a Stateflow diagram)

Largely Automated: (Have to write the spec)

- Can there ever be more than two collisions on the bus during TTA startup with six nodes?

Partially Automated: (Have to guide the analysis)

- Is the TTA group membership always correct for any combination of faults within the fault hypothesis?

Back to Modular Certification

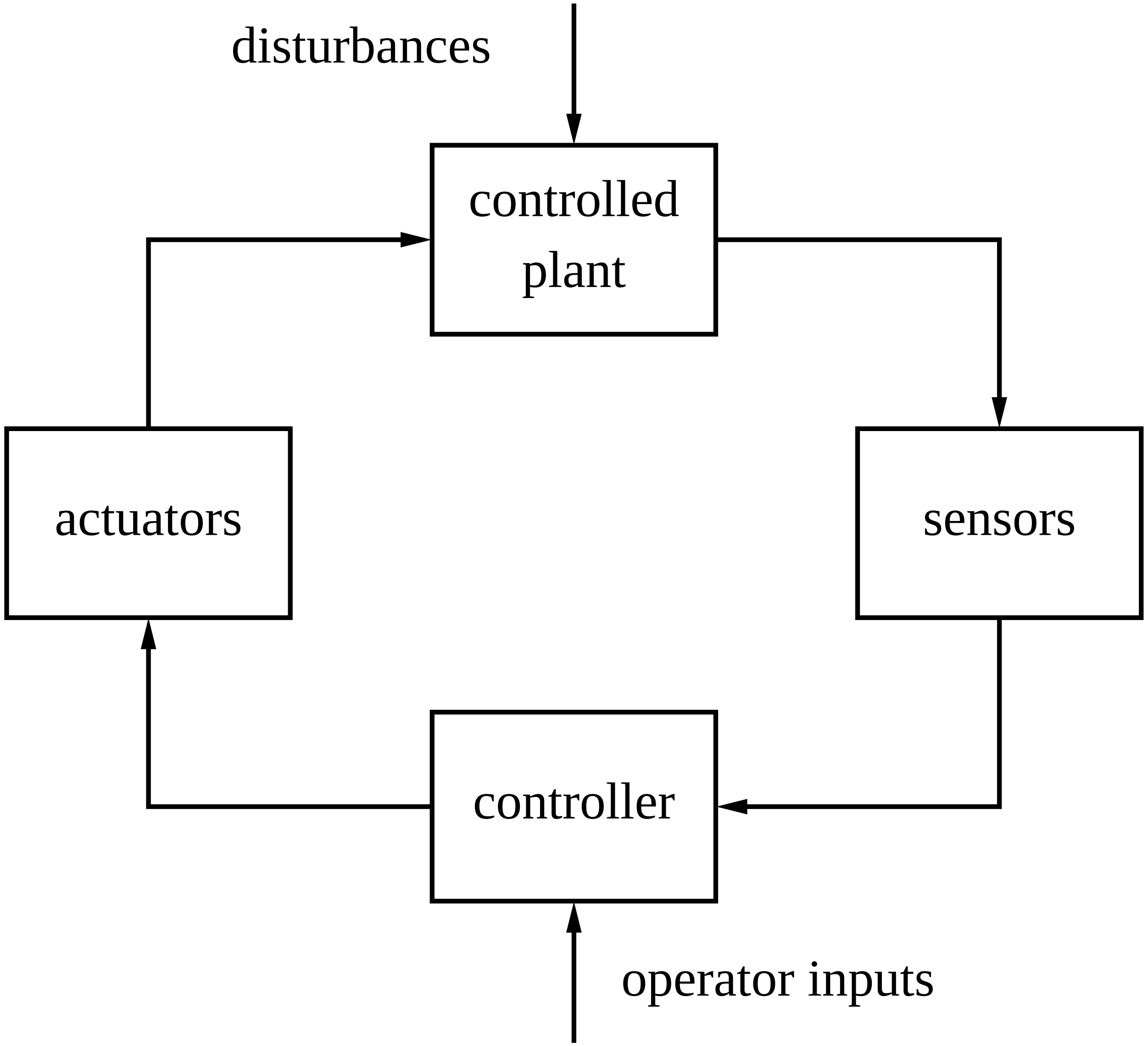
- Idea is to figure out what properties you need
  - Of the components
  - And of the architecture in which they are located
  - And of the way they interact
- To be confident that these combine to guarantee properties of the overall system
- Implicit in this is a shift from process-based to product-based assurance

Old and New Issues

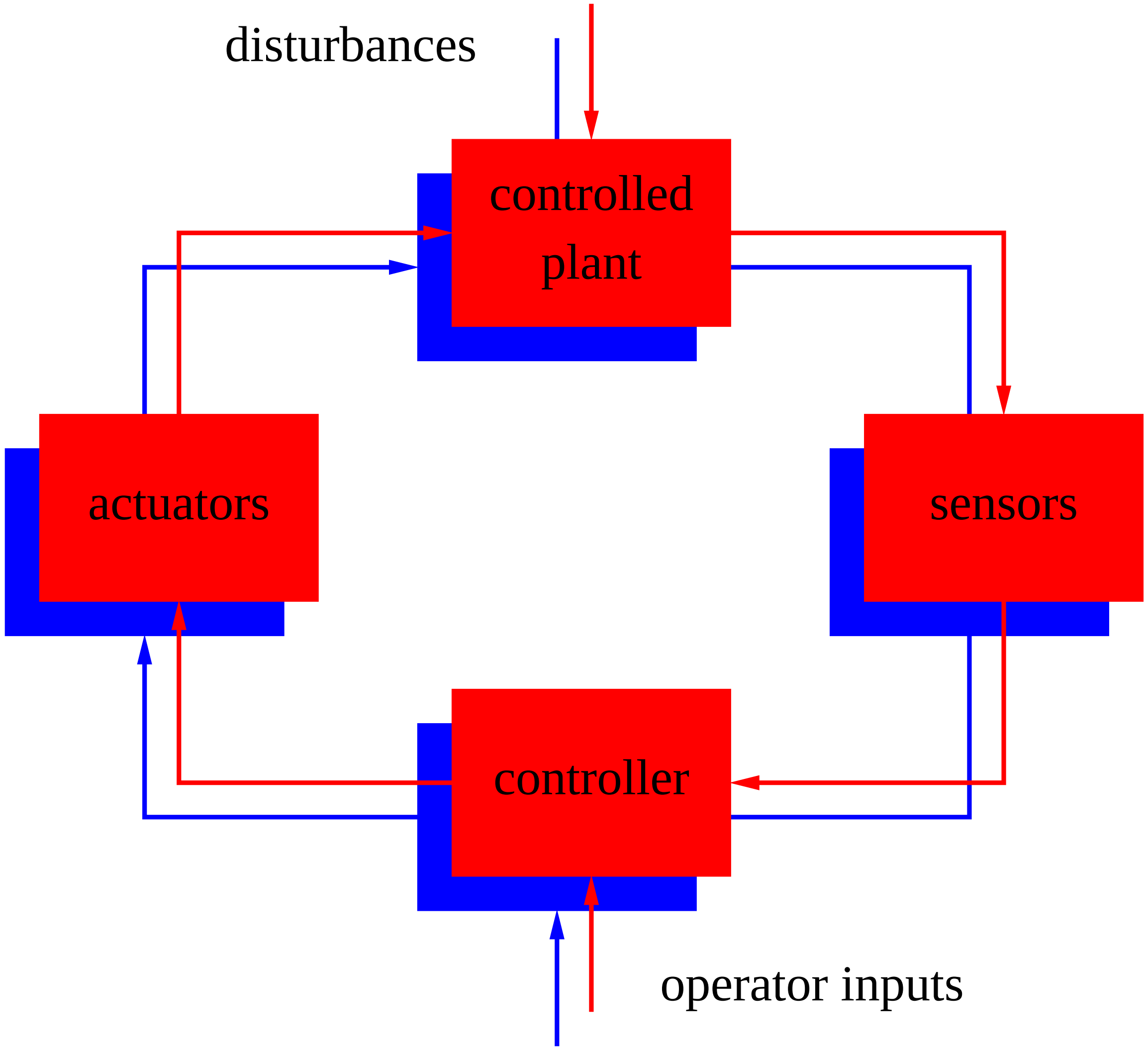
- Does this component do its own thing safely and correctly?
  Standard assurance methods can take care of this
- Could this component (perhaps due to malfunction) stop some other component doing its thing safely and correctly?
  This is the new: you may not yet know what the other components are
  Three 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

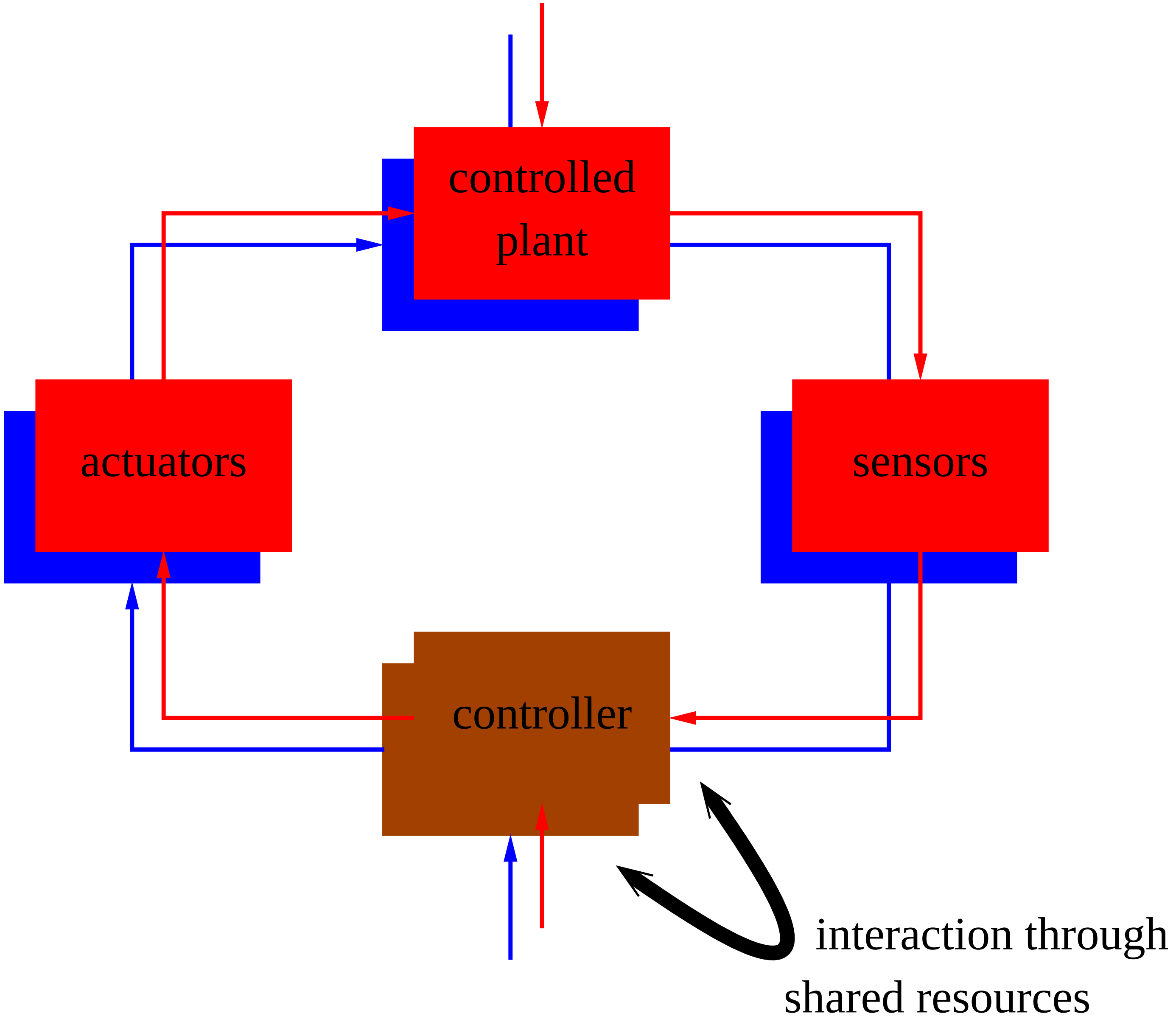
Context: Generic Embedded Control System



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 modular certification is that you can understand interactions of components by considering their specified interfaces
- So partitioning is about enforcing interfaces

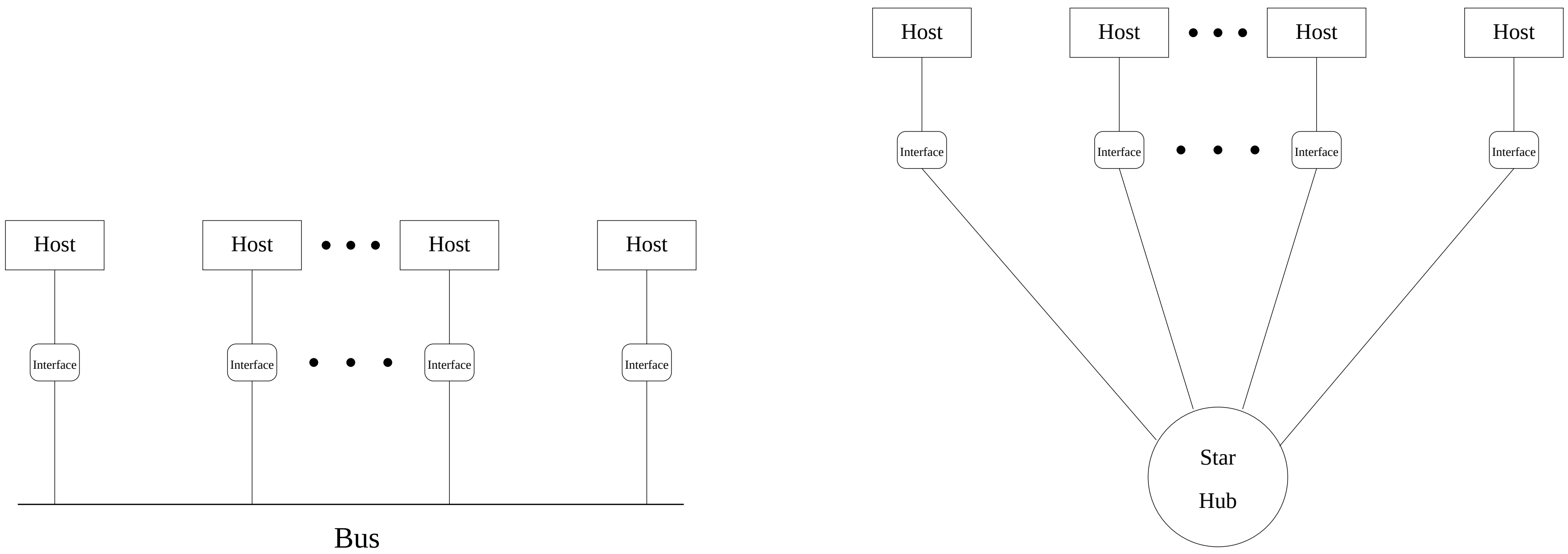
Partitioning

- 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)

Partitioning in Bus Architectures

- Partitioning in distributed systems can be vulnerable to collisions, contention, babbling on the bus, etc.
  - CAN buses, for example, are not suitable for highly critical applications
- Mechanisms to ensure partitioning in bus architectures rely on sophisticated distributed algorithms
  - Fault tolerant clock synchronization
  - Fault tolerant consistent message delivery, etc.
- Algorithms of these kinds are found in SAFEbus, TTA, and SPIDER

TTA Bus Architecure



Bus/hub must be replicated; hub is a logical bus

Assurance For Partitioning

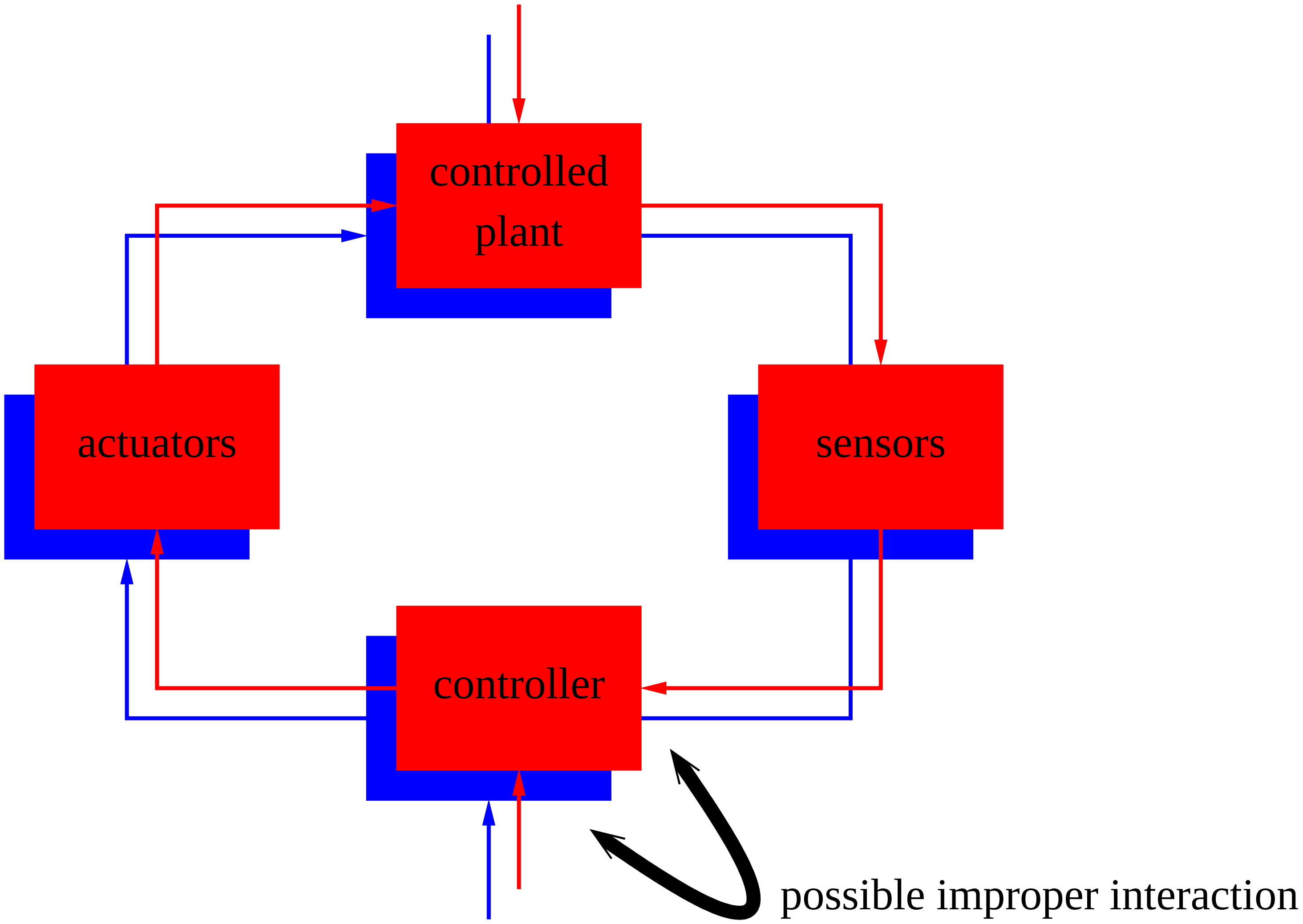
- The subtlety of the algorithms underlying partitioning in bus architectures can make automated verification worthwhile
  - Quite challenging
  - But only needs to be done once
- The benefit over traditional forms of examination is that automated verification considers all possible behaviors
- Can also be used to perform static scheduling
- And to calculate worst case execution time

Partitioning in TTA

- There are five basic algorithms that contribute to partitioning
  - Startup/restart
  - Clock synchronization
  - Bus guardian window timing
  - Group membership
  - Clique avoidance/self stabilization

  We have formally verified some or all of the critical properties of each of these; this also reveals exact fault assumptions
- The algorithms interact: clock synchronization depends on membership and vice versa
- We've figured out how to disentangle these
- Partitioning is an emergent property of all of these (TBD)
- This level of verification goes beyond what has traditionally been performed and is in addition to traditional processes

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 called assume/guarantee reasoning
- Looks circular but computer scientists know how to do it soundly
- In modular 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

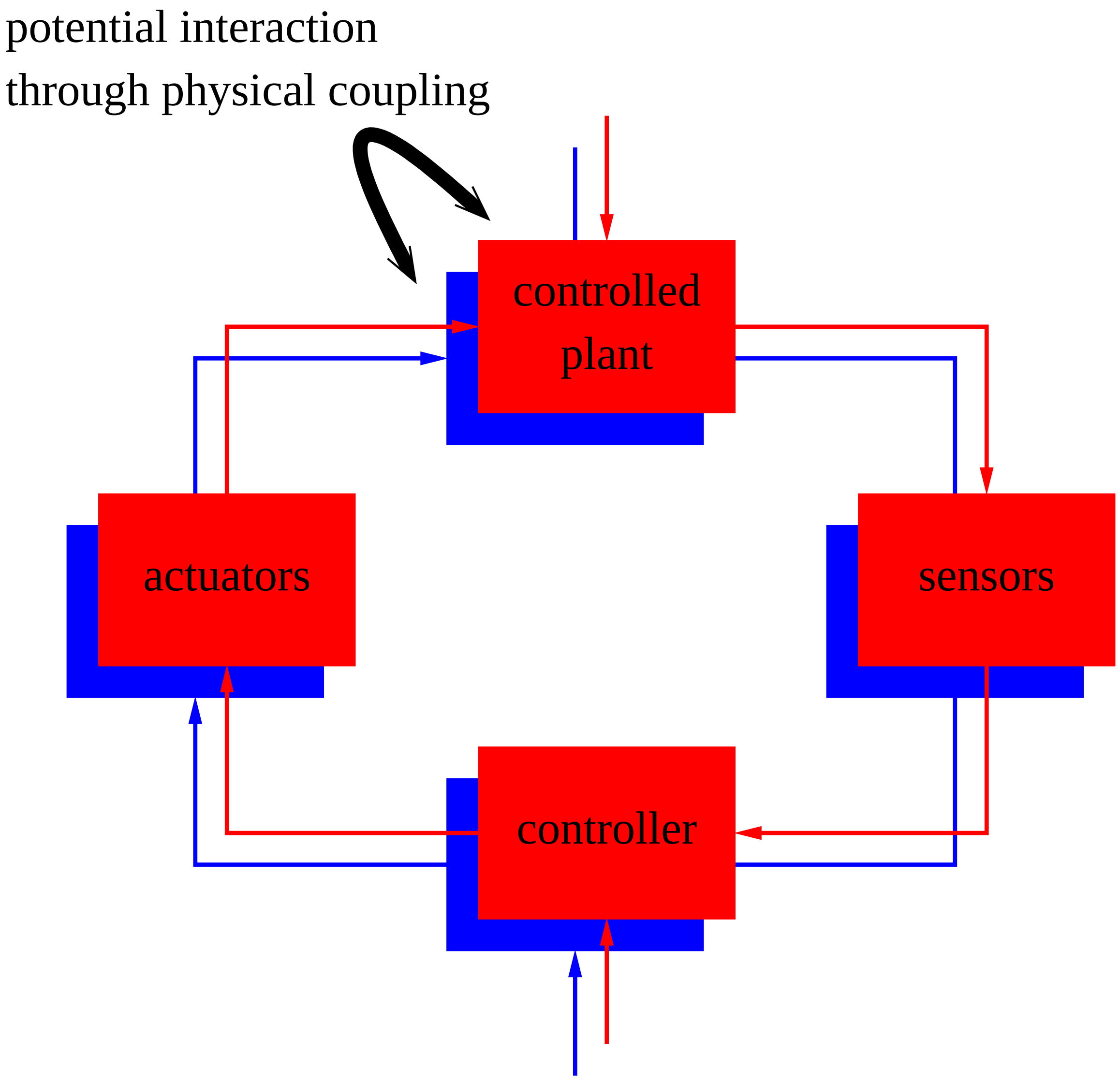
- If component A suffers a failure that causes its behavior to revert from guarantee $G(A)$ to $G'(A)$
- May expect that B's behavior will revert from $G(B)$ to $G'(B)$
- Do not want the lowering of B's guarantee to cause a further regression of A from $G'(A)$ to $G''(A)$ and so on

  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



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

- Modular 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

  Automated verification may be useful here

To Read More

- Partitioning for Avionics Architectures: Requirements, Mechanisms, and Assurance
  http://techreports.larc.nasa.gov/ltrs/PDF/1999/cr/NASA-99-cr209347.pdf
- A Comparison of Bus Architectures for Safety-Critical Embedded Systems,
  http://techreports.larc.nasa.gov/ltrs/PDF/2003/cr/NASA-2003-cr212161.pdf
- Modular Certification,
  http://techreports.larc.nasa.gov/ltrs/PDF/2002/cr/NASA-2002-cr212130.pdf
- Other papers
  http://www.csl.sri.com/users/rushby/biblio.html