Guest lecture, UC Berkeley EECS 149, 13 April 2009

# Safety, Fault-tolerance, Verification, and Certification for Embedded Systems

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

## Overview

- It's pretty hard to get embedded systems working at all
- But many embedded systems are used in contexts where failures are really bad news

  **Expensive:** e.g., Prius recalls

  **Catastrophic (to the mission):** e.g., crash of Mars Polar Lander, several others

  **Dangerous/Deadly:** e.g., violent pitching of VH-QPA
- Because hardware can fail, critical systems often must be fault tolerant
- This adds complexity, and the mechanisms for fault tolerance often become the leading cause of failures
- We'll look at some of these issues, starting with sensors, then computation, then actuators

## Sensors: Violent Pitching of VH-QPA

- An Airbus A330 en-route from Singapore to Perth on 7 October 2008
- Started pitching violently, unrestrained passengers hit the ceiling, 12 serious injuries, so counts as an accident
- Three Angle Of Attack (AOA) sensors, one on left (#1), two on right (#2, #3) of airplane nose
- Want to get a consensus good value
- Have to deal with inaccuracies, different positions, gusts/spikes, failures

## A330 AOA Sensor Processing

- Sampled at 20Hz
- Compare each sensor to the median of the three
- If difference is larger than some threshold for more than 1 second, flag as faulty and ignore for remainder of flight
- Assuming all three are OK, use mean of #1 and #2 (because they are on different sides)
- If the difference between #1 or #2 and the median is larger than some (presumably smaller)threshold, use previous average value for 1.2 seconds
- Failure scenario: two spikes, first shorter than 1 second, second still present 1.2 seconds after detection of first
- Spike gets passed though rate limiter, flight envelope protections activate inappropriately

## Another Example: X29

- Three sources of air data: a nose probe and two side probes
- Selection algorithm used the data from the nose probe, provided it was within some threshold of the data from both side probes
- The threshold was large to accommodate position errors in certain flight modes
- If the nose probe failed to zero at low speed, it would still be within the threshold of correct readings, causing the aircraft to become unstable and "depart"
- Found in simulation
- 162 flights had been at risk

## Sensor Processing: Analysis

- This is a difficult issue and there's no completely satisfactory solution known (good research problem)
- Most algorithms are complex and homespun
- My hunch is that it could be better to deal separately with inaccuracies, position errors, gusts/spikes, failures
- Possible approach: intelligent sensor communicates an interval, not a point value
- Width of interval indicates confidence, health

## Sensor Fusion: Marzullo's Algorithm

**Axiom:** if sensor is nonfaulty, its interval contains the true value

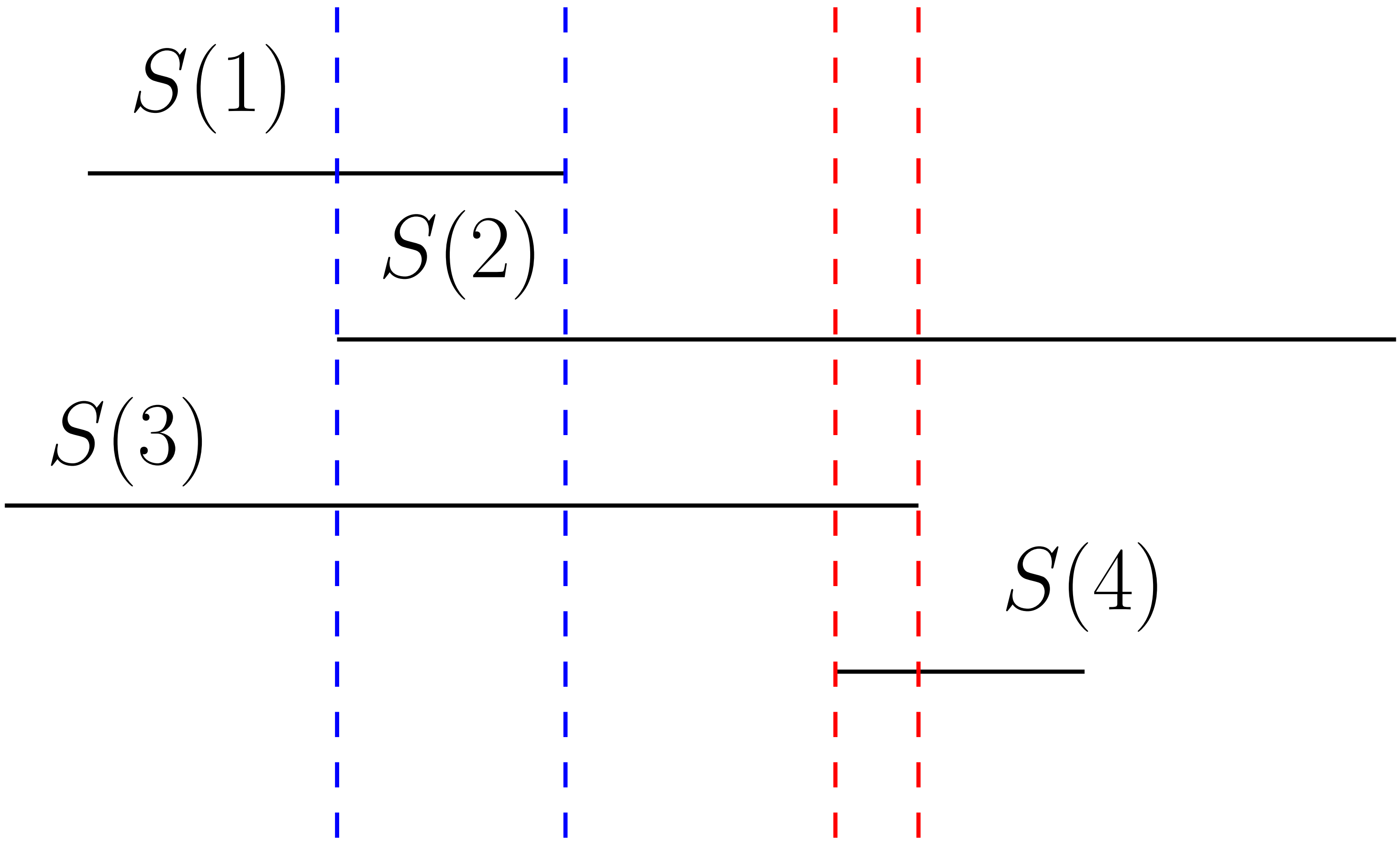
**Observation:** true value must be in overlap of nonfaulty intervals

**Consensus (fused) Interval** to tolerate $f$ faults in $n$, choose interval that contains all overlaps of $n - f$;

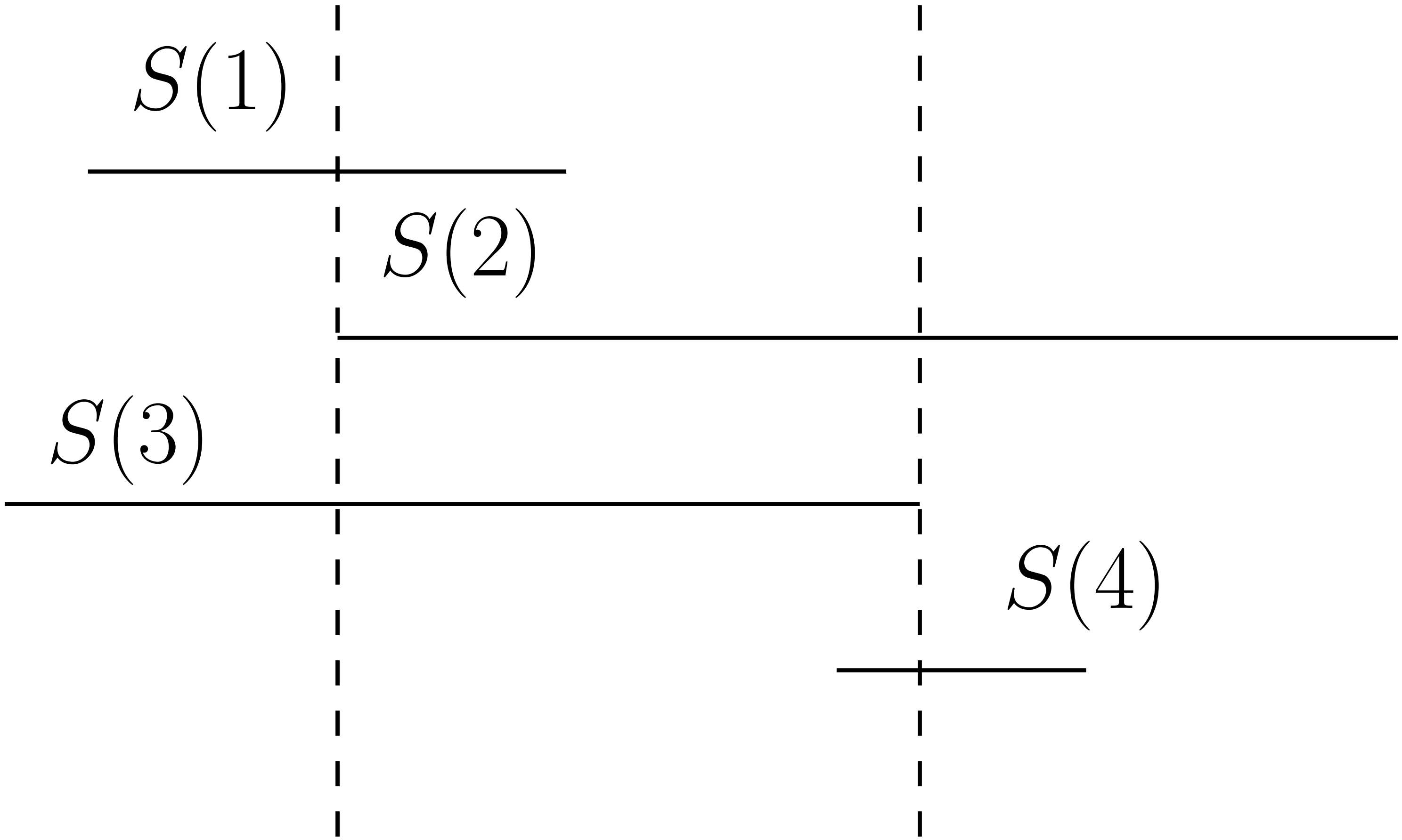
i.e., from least value contained in $n - f$ intervals to largest value contained in $n - f$

**Eliminating faulty samples:** separate problem, not needed for fusing, but any sample disjoint from the fused interval must be faulty

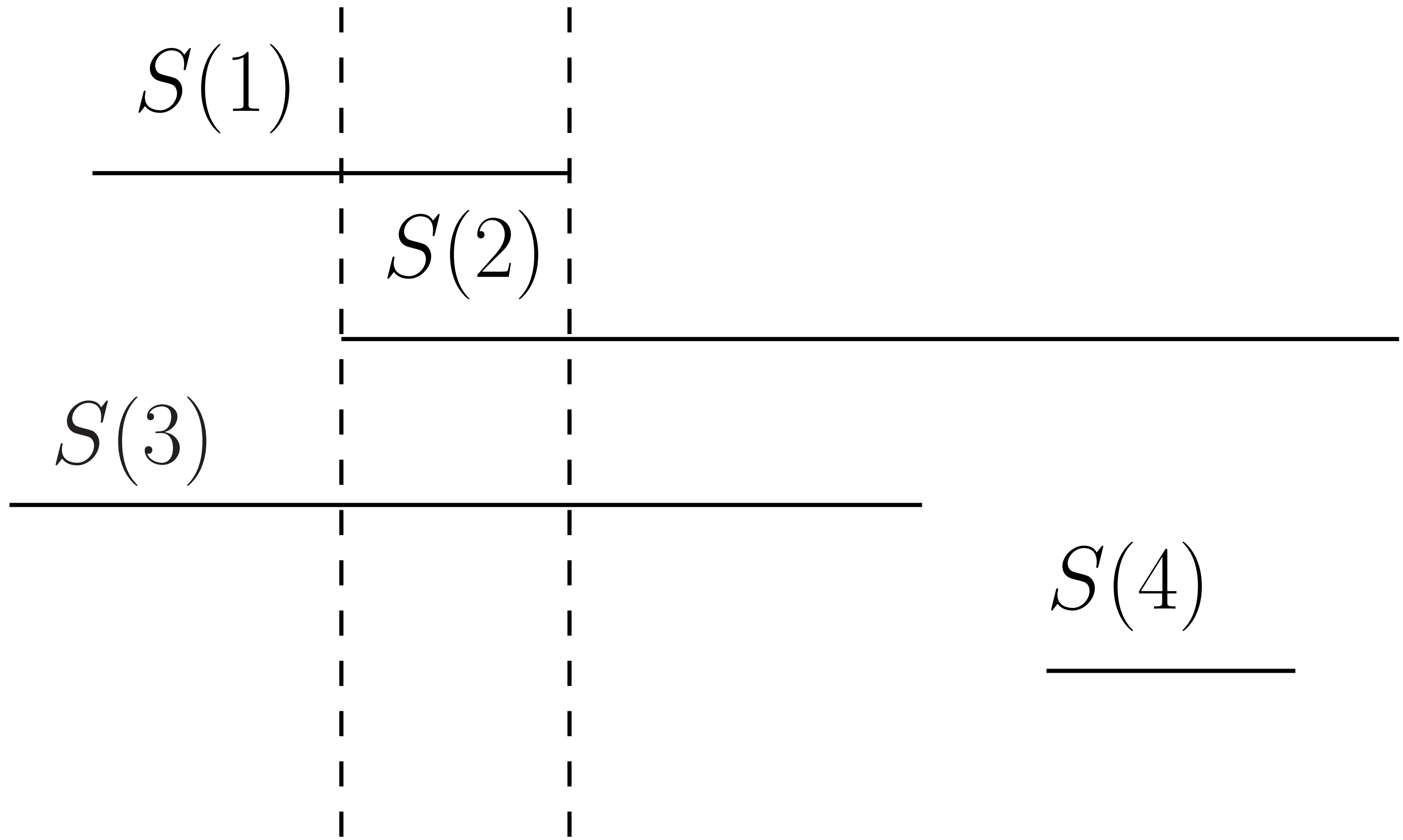
# True Value In Overlap of Nonfaulty Intervals



# Marzullo's Fusion Interval



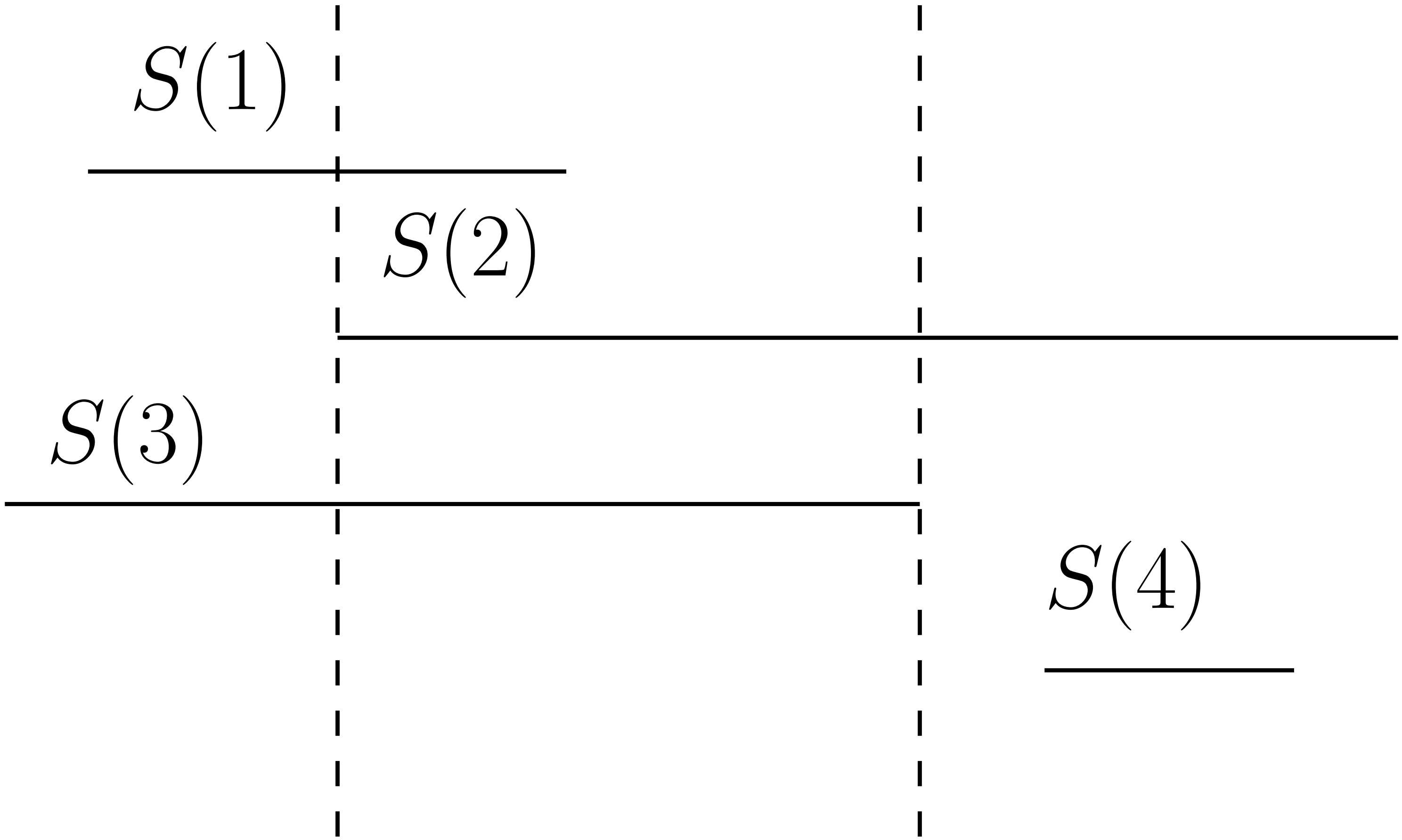
# Marzullo's Fusion Interval: Fails Lipschitz Condition



## Schmid's Fusion Interval

- Choose interval from $f + 1$'st largest lower bound to $f + 1$'st smallest upper bound
- Optimal among selections that satisfy Lipschitz Condition

# Schmid's Fusion Interval



# Compute: Fuel Emergency on G-VATL

- An Airbus A340 en-route from Hong Kong to London on 8 February 2005
- Toward the end of the flight, two engines flamed out, crew found certain tanks were critically low on fuel, declared an emergency, landed at Amsterdam
- 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

## Computational Redundancy: Analysis

- This is big topic, several approaches

  **Self-checking pairs:** two computers cross-compare, shutdown on disagreement, then another pair takes over (more later)

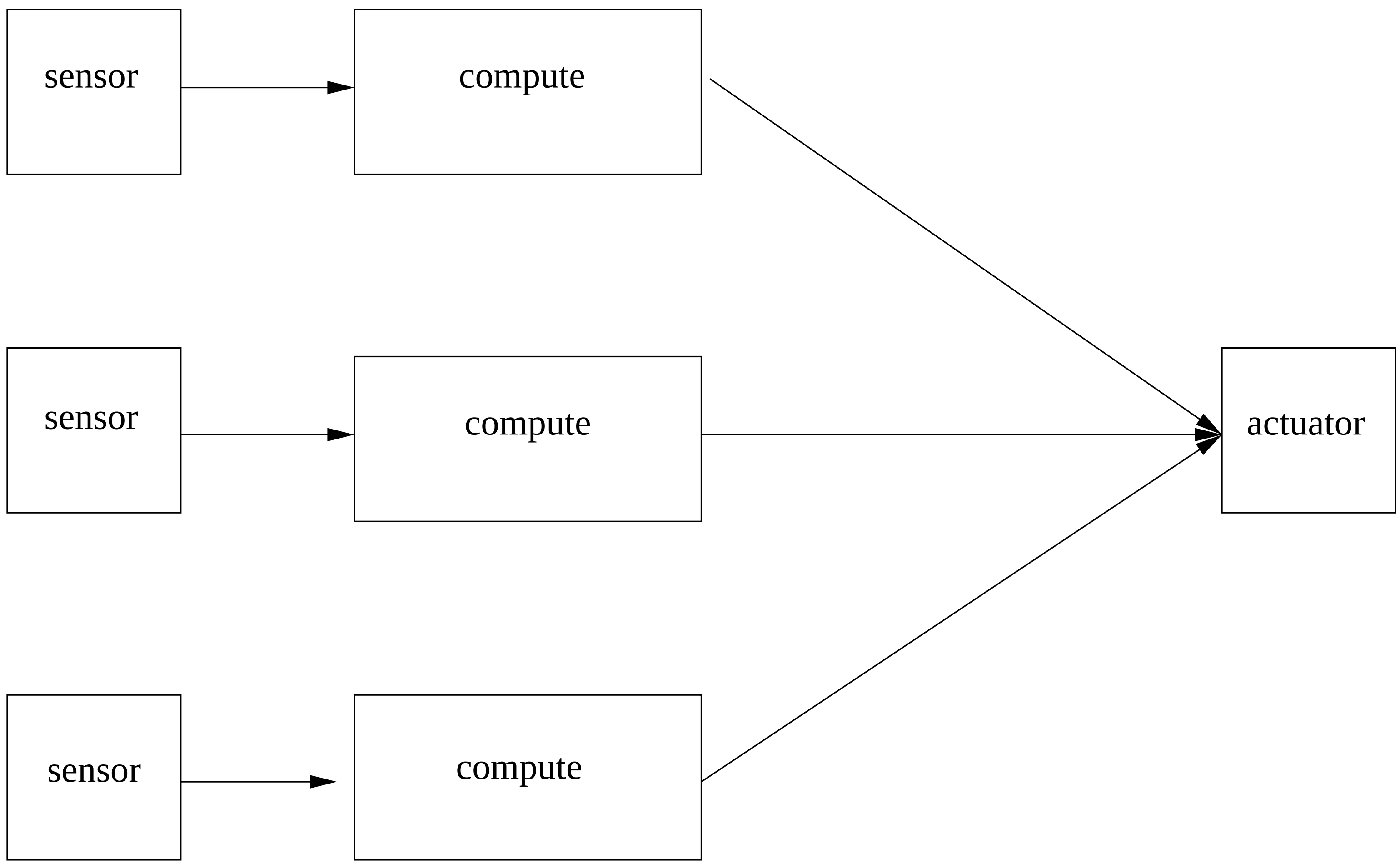
  **N-modular redundancy:** N computers vote on a consensus
  - Exact-match voting, or averaging?
  - Synchronized or unsynchronized?

- The separate computers are generally called channels
- Axiom: failures are independent
- Requires they are separate Fault Containment Units (FCUs)
  - Physically separate
  - Separate power, cooling, etc.

## Unsynchronized Designs (e.g., F16)

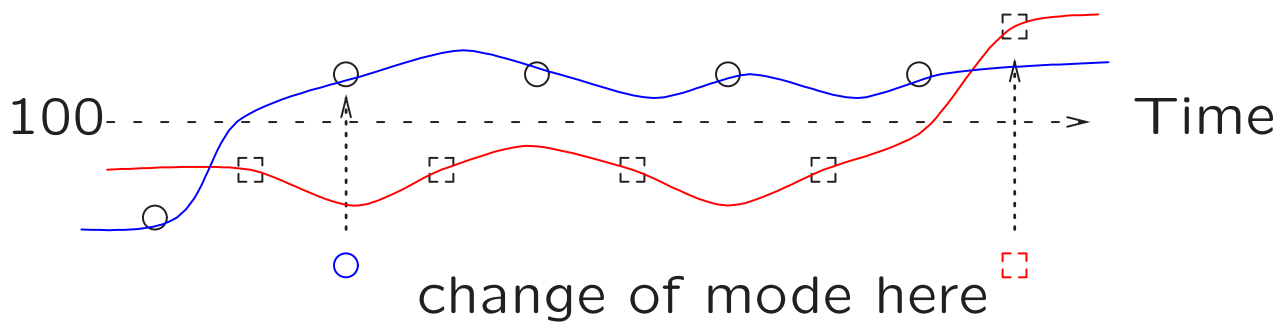
- Channels sample sensors independently, compute independently
- Intuitively maximizes diversity, independence
- But cannot expect outputs to match exactly, so need selection, or averaging, as with sensors
- Tends to produce homespun solutions
- Outputs depend on time integrated values (e.g., velocity, position)
    - Accumulated errors are compounded by clock drift
    - So must exchange and vote integrator values
    - Requires ad-hoc synchronization in the applications code
- Redundancy management pervades applications code (as much as 70% of the code)

# Unsynchronized Designs (e.g., F16)



## Problems with Unsynchronized Designs

- Output selection can induce large transients (cf. Lipschitz)
  - Averaging functions dragged along by faulty values
  - Exclusion on fault detection causes drastic change
- Mode switches can cause channel divergence
  - IF $x > 100$ THEN ... ELSE ...




  - Output very sensitive to sample when near decision point
- Have to modify control laws to ramp changes in and out smoothly, or use ad hoc synchronization and voting
- So computational redundancy interacts with control

## Historical Experience of DFCS (early 1980s)

- Advanced Fighter Technology Integration (AFTI) F16
- Digital Flight Control System (DFCS) to investigate "decoupled" control modes
- Triplex DFCS to provide two-fail operative design
- Analog backup
- Digital computers not synchronized
- "General Dynamics believed synchronization would introduce a single-point failure caused by EMI and lightning effects"

## AFTI F16 Flight Test, Flight 36

- Control law problem led to "departure" of three seconds duration
- Sideslip exceeded $20^\circ$, normal acceleration exceeded $-4$g, then $+7$g, angle of attack went to $-10^\circ$, then $+20^\circ$, aircraft rolled $360^\circ$, vertical tail exceeded design load, failure indications from canard hydraulics, and air data sensor
- Side air data probe blanked by canard at high AOA
- Wide threshold passed error, different channels took different paths through control laws
- Analysis showed this would cause complete failure of DFCS for several areas of flight envelope

## AFTI F16 Flight Test, Flight 44

- Unsynchronized operation, skew, and sensor noise led each channel to declare the others failed
- Simultaneous failure of two channels not anticipated
  So analog backup not selected
- Aircraft flown home on a single digital channel
  (not designed for this)
- No hardware failures had occurred

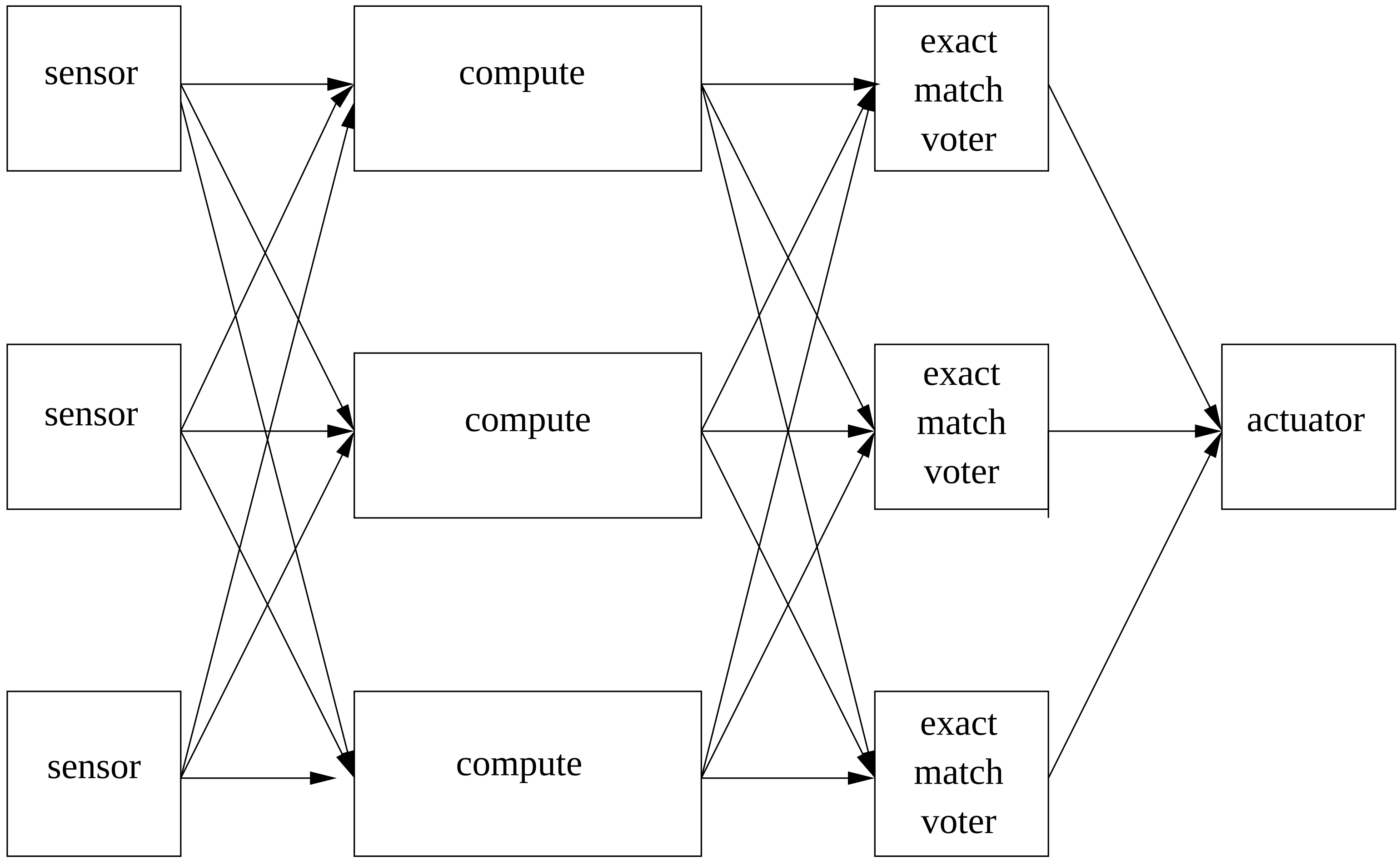
## Other AFTI F16 Flight Tests

- Repeated channel failure indication in flight was traced to roll-axis software switch
- Sensor noise and unsynchronized operation caused one channel to take a different path through the control laws
- Decided to vote the software switch
- Extensive simulation and testing performed
- Next flight, same problem still there
- Found that although switch value was voted, the unvoted value was used

## Analysis: Dale Mackall, NASA Engineer
## AFTI F16 Flight Test

- Nearly all failure indications were not due to actual hardware failures, but to design oversights concerning unsynchronized computer operation
- Failures due to lack of understanding of interactions among
  - Air data system
  - Redundancy management software
  - Flight control laws (decision points, thumps, ramp-in/out)

# Synchronized Designs

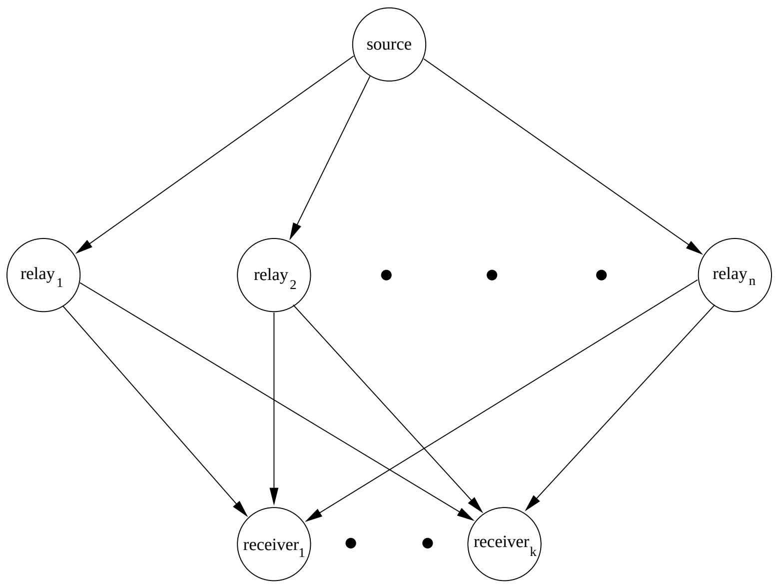


## Synchronized Fault-Tolerant Systems (e.g., 777 AIMS)

- Synchronized systems can use exact-match voting for fault-masking and transient recovery—potentially simpler and more predictable
- It's easier to maintain order than to establish order (Kopetz)
  - Synchronized designs solve the hard problems once
  - Unsynchronized designs must solve them on every frame
- Need fault-tolerant clock synchronization
- And fault-tolerant distribution of sensor values so that each channel works on the same data: interactive consistency (aka. source congruence, Byzantine agreement)
- Both these need to deal with asymmetric or Byzantine faults

## Interactive Consistency

- Needed whenever a single source (e.g., sensor) is distributed to multiple channels (e.g., redundancy for fault tolerance)
  - Faulty source could otherwise drive the channels apart
- A solution is to pass through $n$ intermediate relays in parallel and vote the results (OM(1) algorithm)




Can tolerate certain numbers and kinds of faults: e.g.,
$n \geq 3a + 2s + m + 1$

## SOS Interpretation of Byzantine Faults

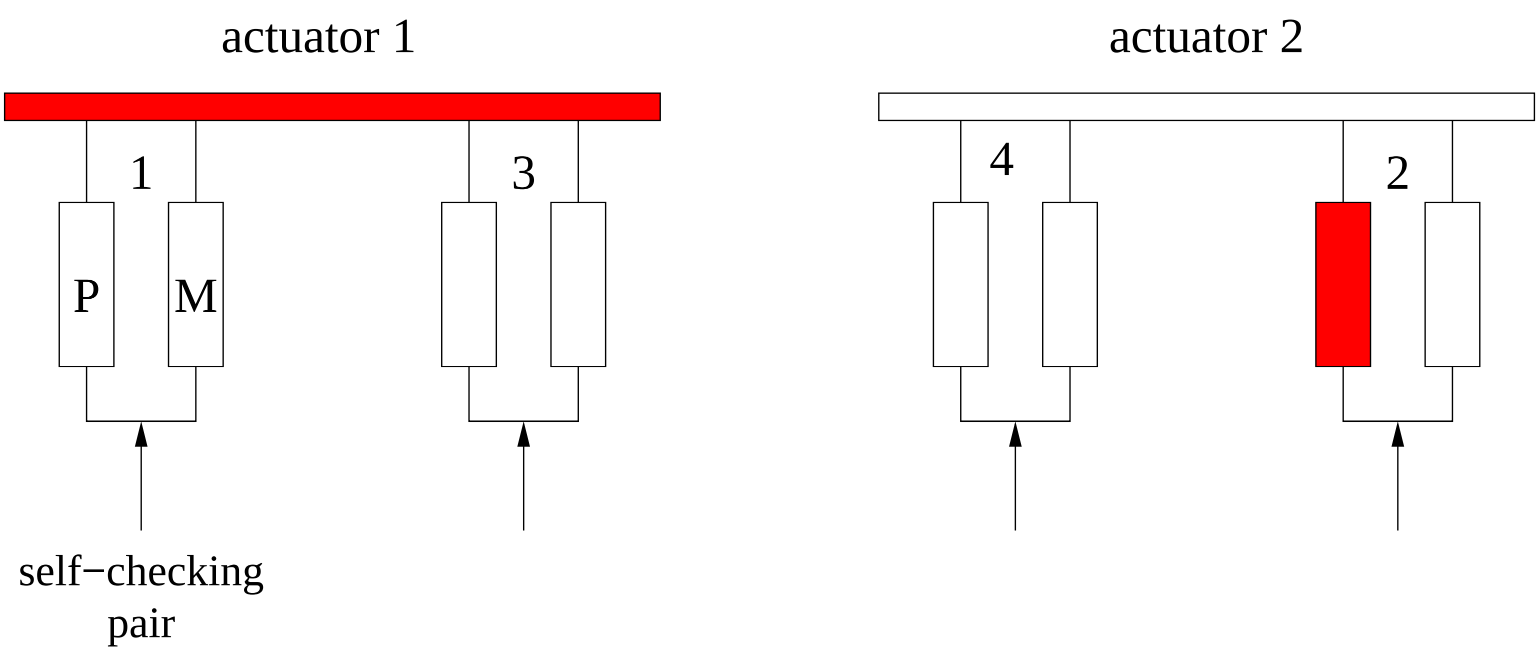
- The "loyal" and "traitorous" Byzantine Generals metaphor is unfortunate
  - Also academic focus on asymptotic issues rather than maximum fault tolerance from given resources
- Leads most homespun designers to reject the problem
  - Also, $10^{-9}$ per hour is beyond casual human experience
  - Actual frequency of rare faults is underestimated
- Slightly Out of Specification (SOS) faults can exhibit Byzantine behavior
  - Weak voltages (digital 1/2)
    - ⋆ One receiver may interpret 2.5 volts as 0, another as 1
  - Edges of clock regions
    - ⋆ One receiver may get the message, another may not

## A Real SOS Fault

- Massively redundant aircraft system
- Theoretically enough redundancy to withstand 2 Byzantine faults
- But homespun design did not consider such possibility
- Several failures in 2 out of 3 "independent" units
- Entire fleet within days of being grounded
- Adequate fix developed by engineer who had designed a Byzantine-resilient system for same aircraft

# Actuators: Airbus Aileron Design

- One approach, based on self-checking pairs does not attempt to distinguish computer from actuator faults
- Must tolerate one actuator fault and one computer fault simultaneously



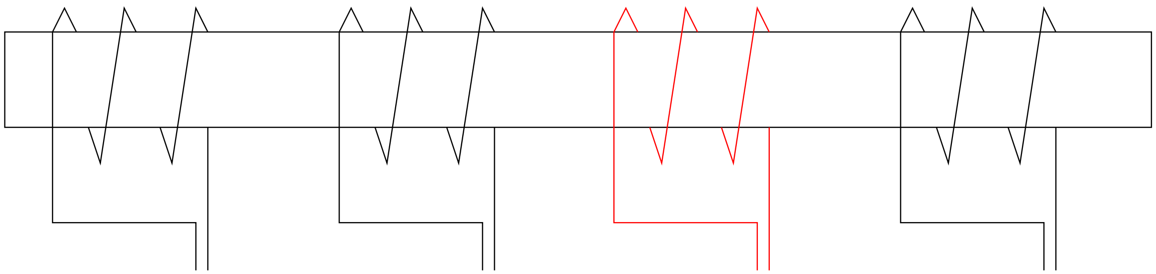

- Can take up to four frames to recover control

## Consequences of Slow Recovery

- Use large, slow moving ailerons rather than small, fast ones
    - Hybrid systems question: why?
- So the ailerons take up a larger part of the wing
- 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

# Actuators: Physical Averaging

- Alternative uses averaging at the actuators
  - E.g., multiple coils on a single solenoid



  - Or multiple pistons in a single hydraulic pot
- Hybrid systems question: how well does this work?

## Human Interaction

- Sophisticated control laws can leave the operator (pilot) out of phase, get Pilot Induced Oscillations (PIOs): first Shuttle drop test, F22 crash (Google for the video)
- Human error is the dominant cause of aircraft incidents and accidents (70% of accidents)
- Actually, the error is usually bad and complex interface design, which provokes automation surprise, of which mode confusion is a special case
- Pilots are surprised by the behavior of the automation
  - Or confused about what "mode" it is in
  - "Why did it do that?"
  - "What is it doing now?"
  - "What will it do next?"

## Human Factors Example: MD-88 Altitude Bust

- The pitch modes determine how the plane climbs
  - VSPD: climb at so many feet per minute
  - IAS: climb while maintaining set airspeed
  - ALT HLD: hold current altitude
- The altitude capture mode determines whether there is a limit to the climb
  - If altitude capture is armed
    - ⋆ Plane will climb to set altitude and hold it
    - ⋆ There is also an ALT CAP pitch mode that is used to end the climb smoothly
  - Otherwise
    - ⋆ Plane will keep climbing until pilot stops it

## Altitude Bust Scenario—I

Crew had just made a missed approach

Climbed and leveled at 2,100 feet

Color code: done by pilot, done by others or by automation

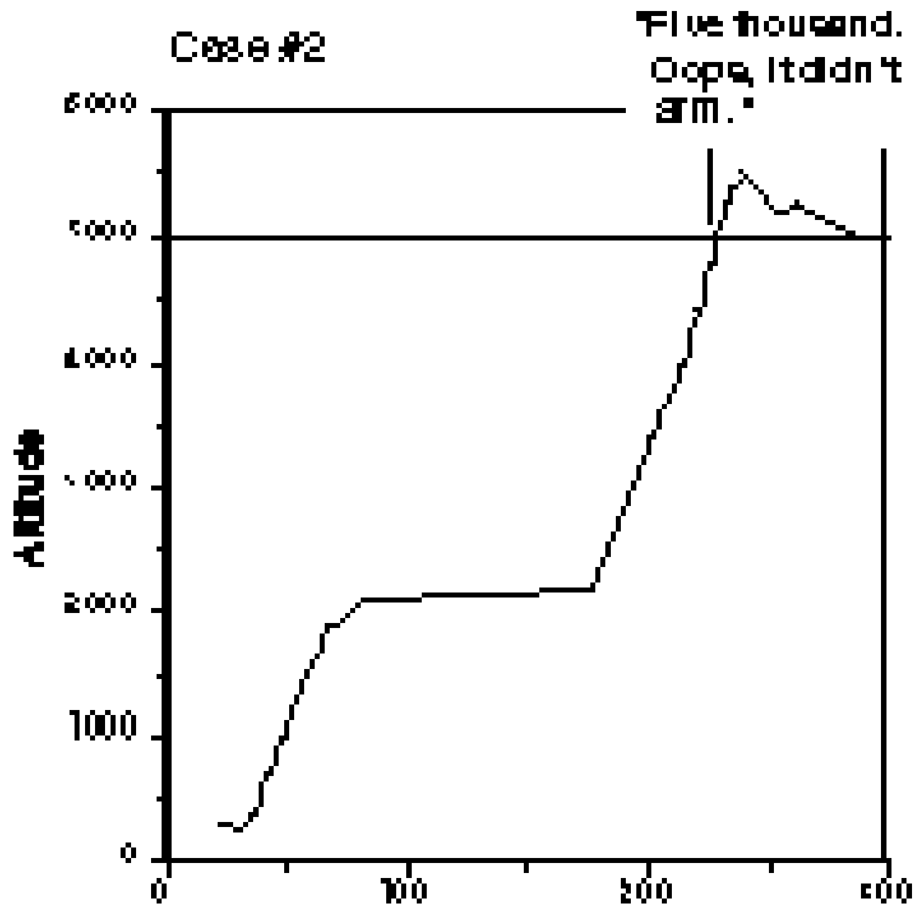
- Air traffic Control: "Climb and maintain 5,000 feet"
- Captain set MCP altitude window to 5,000 feet
  - Causes ALT capture to arm
- Also set pitch mode to VSPD with a value of 2,000 fpm
- And autothrottle (thrust) to SPD mode at 255 knots

## Altitude Bust Scenario—II

- Climbing through 3,500 feet, flaps up, slats retract
- Captain changed pitch mode to IAS
  - Causes autothrottle (thrust) to go to CLMP
- Three seconds later, nearing 5,000 feet, autopilot automatically changed pitch mode to ALT CAP
  - Which disarmed ALT capture
- 1/10 second later, Captain changed VSPD dial to 4,000 fpm

# Altitude Bust Scenario: Outcome

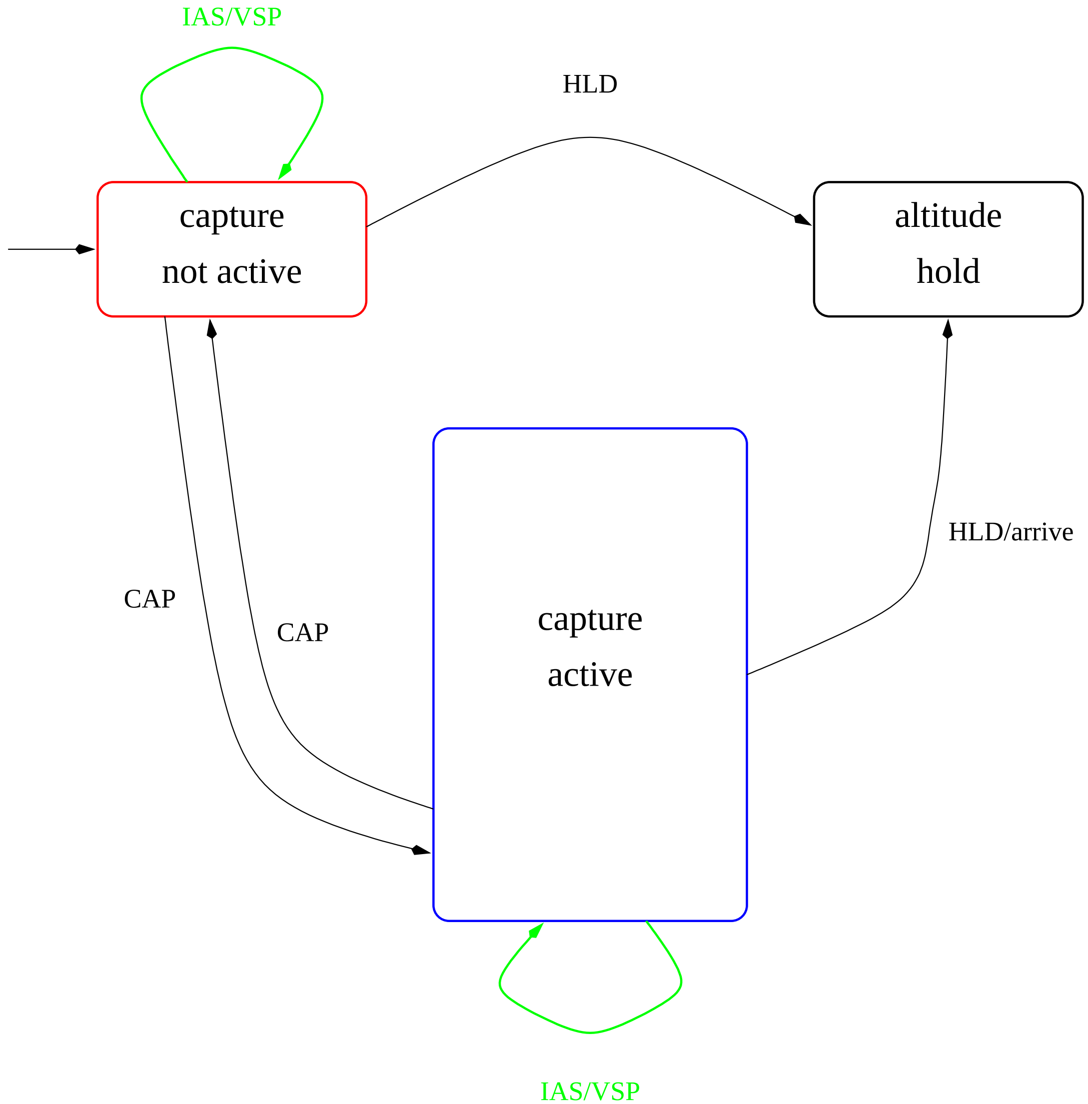
- Plane passed through 5,000 feet at vertical velocity of 4,000 fpm
- “Oops: It didn’t arm”
- Captain took manual control, halted climb at 5,500 with the “*altitude—altitude*” voice warning sounding repeatedly



## Human Factors: Analysis

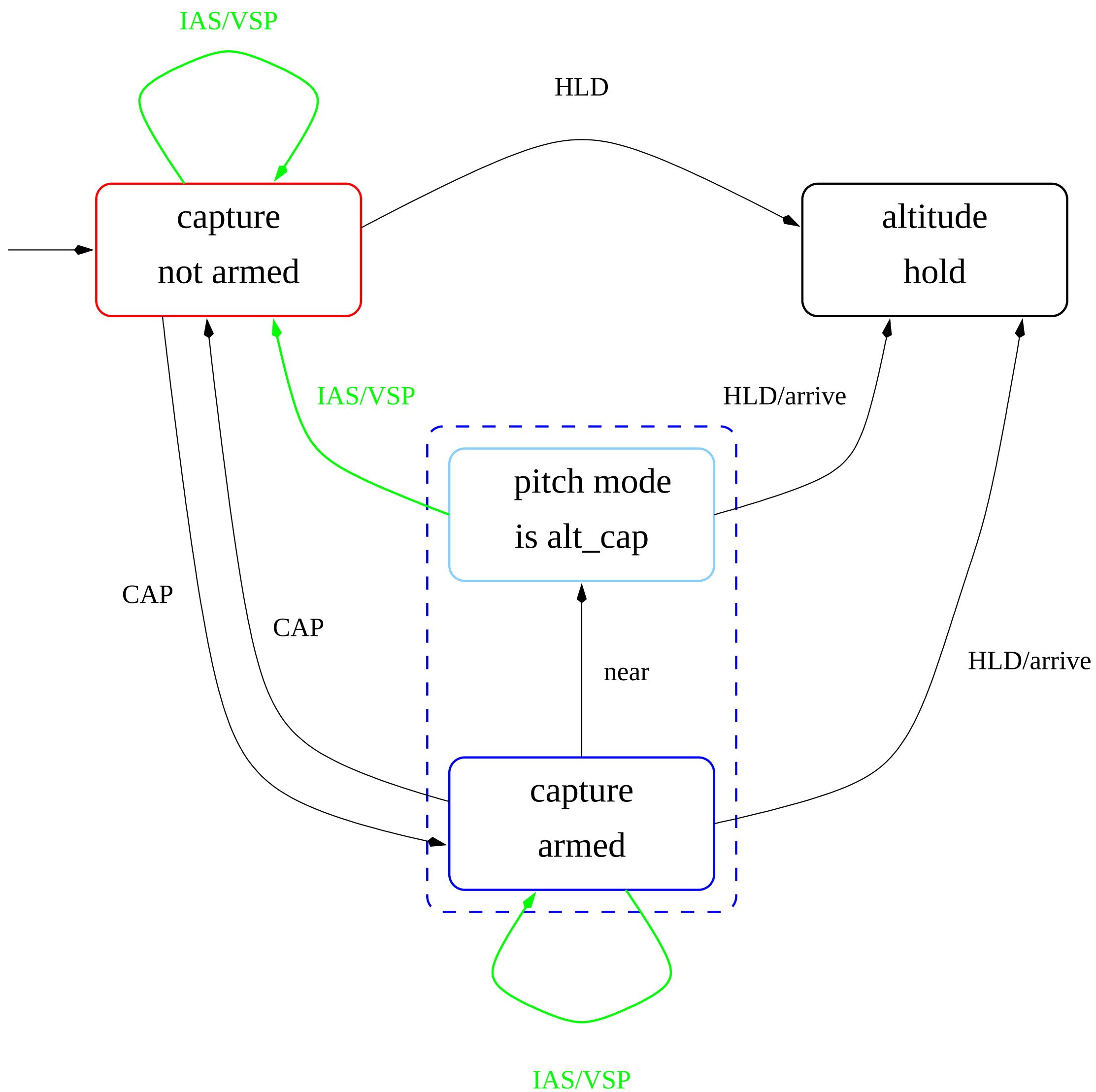
- Operators use "mental models" to guide their interaction with automated systems
- Automation surprises arise when the operator's mental model does not accurately reflect the behavior of the actual system
- Mode confusion is a just a special case: the mental model is not an accurate reflection of the actual mode structure
  - Or loses sync with it
- Mental models can be explicitly formulated as state machines
  - And we can "capture" them through observation, interviews, and introspection
  - Or by studying training manuals
    (which are intended to induce specific models)

# Mental Model for Pitch Modes in MD88




Whether capture is active is independent of the pitch mode

# Actual System, Pitch Modes in MD88




There is an alt_cap pitch mode that flies the final capture

## Reliability and Safety

- These are not the same
- Different techniques are needed to ensure them
- Often require both simultaneously
  - Nuclear: shutdown on problems, reliability affects efficiency, not safety
  - Airplane: have to keep flying
- Both can be specified probabilistically
  - Typically probability of (safety) failure on demand
  - Or probability of (safety) failure per hour

## Nine Nines

- Requirement for civil aircraft is no catastrophic failure condition (one which could prevent continued safe flight and landing) in the entire life of the fleet concerned
- Say 1,000 aircraft in fleet, 40 years life, 5,000 hours/year, 10 embedded systems, each with 10 catastrophic failure conditions
- That's $2 \times 10^9$ hours exposure for each
- So need probability of failure less than $10^{-9}$ per hour, sustained for 20 hours
- Also known as nine nines (reliability 0.999999999)

## Assurance for Nine Nines

- Hardware reliability is about six nines
  - Small transistors of modern processors increasingly vulnerable to single event upsets (SEU)s, aging effects
- Can test systems to about three nines (maybe four)
- Nine nines would require 114,000 years on test
- So most of the assurance has to come from analysis
- With proper fault-tolerant design, channel failures are independent
- So can multiply probabilities: two-channel system with three nines per channel gives six nines
- Use Markov and similar models to model reliabilities of more complex architectures

## Design Errors

- All software errors are design errors
- FPGAs, ASICS, etc. are the same as software
- Failure is certain, given a scenario that activates the bug
- But scenarios are a stochastic process
- So can speak of software reliability
  - Three nines means probability of encountering a scenario that activates a bug is 1 in 1,000
- $n$-version software: develop $n$ different versions of the software, deliberately diverse, and vote them
- Experiments and theory cast doubt on the approach
  - Failures not independent: difficulty varies over input space
- Seems to work in practice (Airbus fly-by-wire)
- But difficult to quantify benefits, costs

## Certification

- Have to convince a regulator that you've thought of everything
- Your design deals safely with every contingency
- And your implementation is correct
- Can choose where design (analyzed for safety) ends and implementation (analyzed for correctness) begins
- Have thought of everything: means you have considered all possible behaviors of your design in interaction with its environment
- Conceptually, this is what model checking is about
    - Build models of the design, and of the environment
    - Explore reachable states of their composition
- Except it's traditionally done by hand, with very informal and abstract models

## Hazard Analysis

- First, identify the hazards (e.g, fire in airplane hold)
- Then figure out how to eliminate, control or mitigate them
  - e.g., if hazard is fire, can eliminate by having no combustible material or no oxygen, control by fire extinguishing system, mitigate by preventing spread
  - cf. ETOPS planes
- Iterate as the design evolves, and new hazards emerge
- Formulate safety claims
  - e.g., reliability of fire extinguishing system
- Then analyze those

## Safety Analysis

- Can think of it as model checking by hand
- Can only explore a few paths
- So focus on those likely to harbor safety violation
- Explore backward from hypothesized system failure
  - Fault Tree Analysis (FTA)
- And forward from hypothesized component failures
  - Failure Modes and Effects Analysis (FMEA, FMECA)
- And along control, data, other flows
  - HAZOP guidewords
  - e.g., late, missing, wrong, too little, too much

## Certification Processes

- These differ considerably across industries
- As do the power of the regulatory authorities
- Most are based on standards or guidelines
- FDA 510(k) process is an exception
  - Argue that your device is equivalent to something prior
  - e.g., Da Vinci surgical system (a robot) was certified under 510(k) as equivalent to a clamp

# Standards-Based Assurance

Commercial airplanes, for example

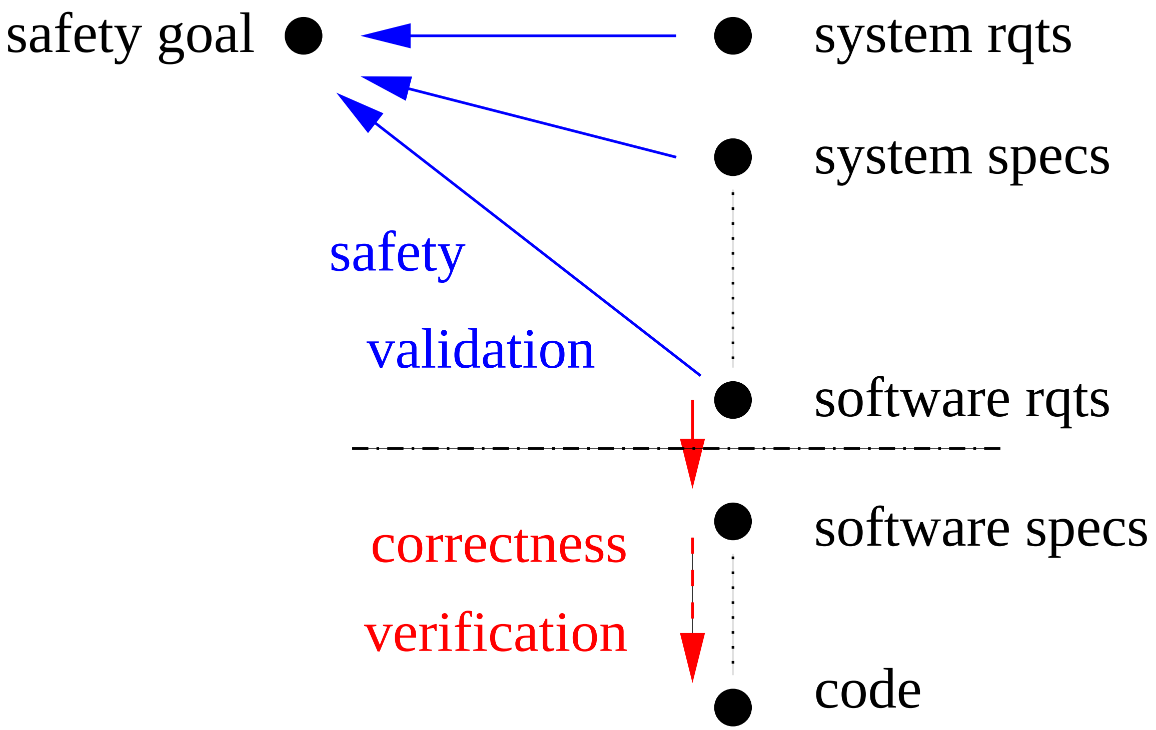
- ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment
- ARP 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems
- DO-297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations
- DO-254: Design Assurance Guidelines for Airborne Electronic Hardware
- DO-178B: Software Considerations in Airborne Systems and Equipment Certification

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

But less suitable with novel problems, solutions, methods

## Software Standards Focus on Correctness Rather than Safety




- Premature focus on correctness is hugely expensive

## Standards and Argument-Based Assurance

- All assurance is based on **arguments** that purport to justify certain **claims**, based on documented **evidence**
- Standards usually define only the evidence to be produced
- The claims and arguments are implicit
- Hence, hard to tell whether given evidence meets the intent
- E.g., is MC/DC coverage evidence for good testing or good requirements?
- Recently, argument-based assurance methods have been gaining favor: these make the elements explicit

## The Argument-Based Approach to Software Certification

- E.g., UK air traffic management (CAP670 SW01), UK defence (DefStan 00-56), growing interest elsewhere
- Applicant develops a safety case
  - Whose outline form may be specified by standards or regulation (e.g., 00-56)
  - 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
- Generalized to security, dependability, assurance cases
- The case is evaluated by independent assessors
  - Explicit claims, evidence, argument

# Looking Forward

- Systems are becoming massively more complex
- And more integrated
- cf. Integrated Modular Avionics (IMA)
- OTOH. sophisticated COTS components (e.g., TT-Ethernet) replace homespun designs
- "Thinking of everything" becomes a lot harder: emergent behaviors
- Need compositional methods of assurance and certification
- Need much more automation in the assurance process
  - Consider more scenarios, more reliably
- Adaptive systems move design to runtime
  - Assurance must go there, too

## A Hint of the Future

- Recall the A340 FCMC fault
- Monitoring for reasonable fuel distribution would have caught this
- Software requirements were the source of the bug
    - So monitor the safety case instead
- Given a formal safety case, could generate a monitor
- It would be possibly perfect
- At the aleatory level, failures of a reliable channel and a possibly perfect one are conditionally independent
- Can multiply their probabilities

$$\text{risk} \leq f \times c_1 \times (C + P_{A1} \times P_{B1}) + (1 - f) \times c_2 \times P_{B2}$$

- Epistemic estimation of the parameters is feasible