




Fault Tolerant Computing for Exploration

May 2009

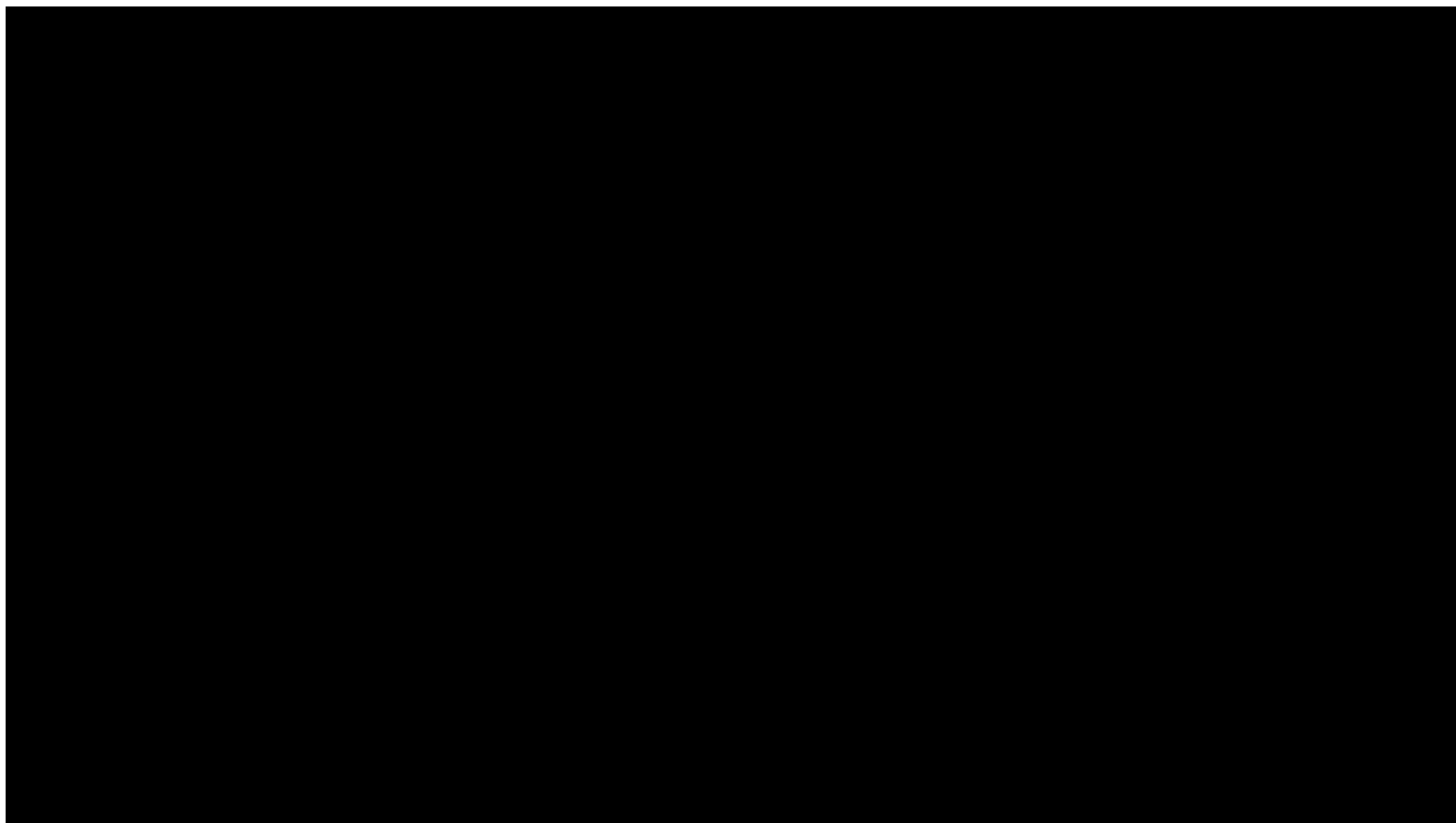
A composite image of celestial bodies in space. From left to right, it shows the Earth with blue oceans and brown continents, the Moon with its grey surface, and Mars with its reddish-brown surface. A bright sun or star is visible in the upper right, creating a lens flare effect. Two thick red lines originate from the sun and extend downwards and to the left, crossing over the Earth and Moon.

Dr. Robert F. Hodson
Avionics Lead
Software & Avionics Integration Office
NASA Constellation Program

CONSTELLATION

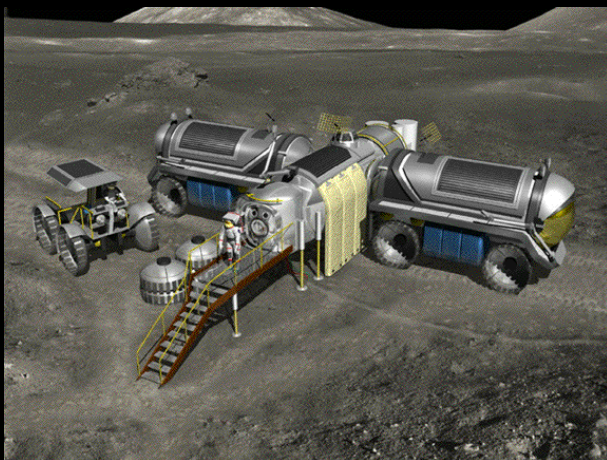
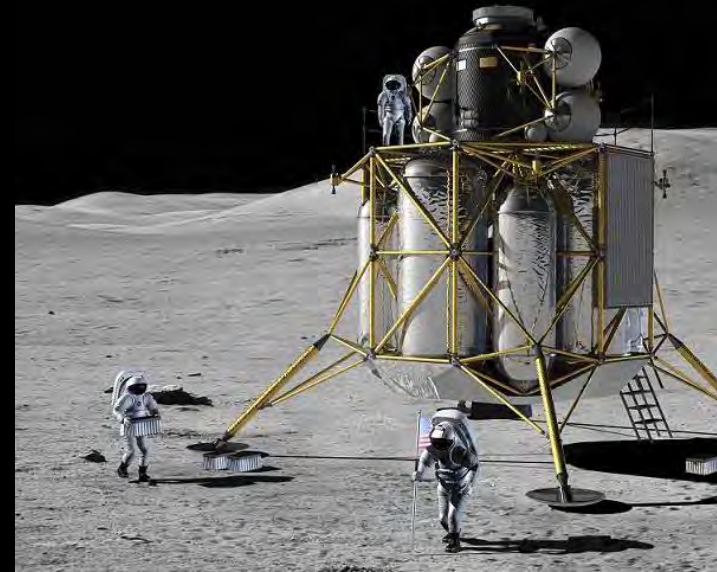
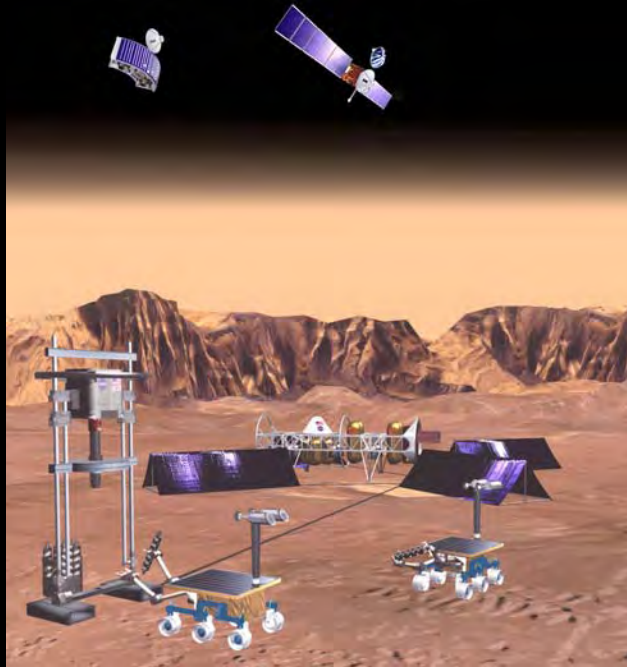


NASA's Constellation Mission Overview

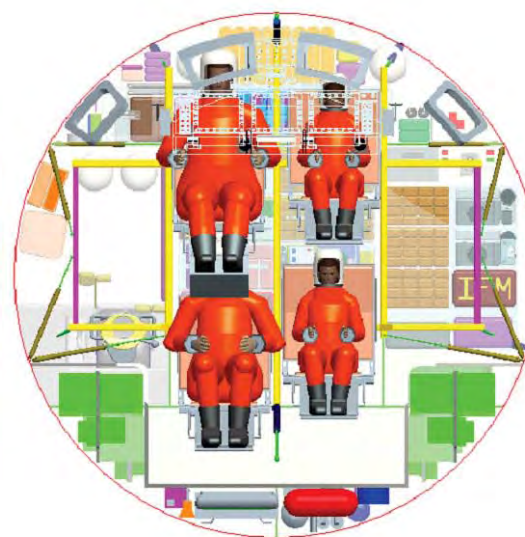
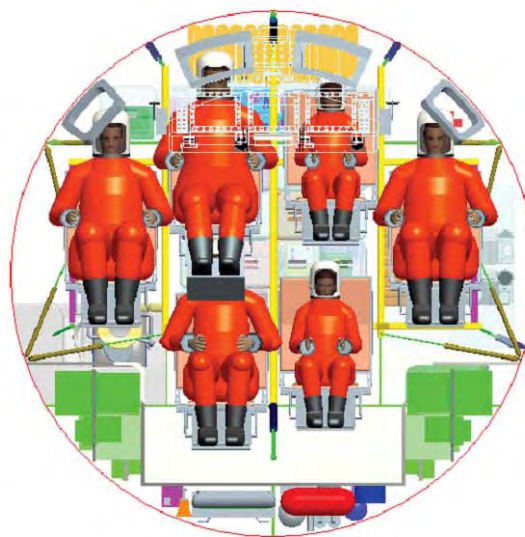


Video: [CX_Mission_11_08.wmv](#)

Future Missions

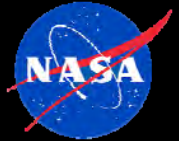


- ◆ **Safety**
 - Reliability, Fault Tolerance, Human Rating
- ◆ **Environments/Survivability**
 - Ascent, Descent, Landing (Water), Radiation, Thermal, Vacuum
 - Long Duration Exposure
- ◆ **Resource limitations**
 - Mass, Power, Volume
- ◆ **Performance**
 - Video Rates, Autonomy, Docking, Landing,
 - Determinism/Non-Determinism, Latency
- ◆ **System of systems**
 - Interoperability, Managing complexity, Commonality





Types of Faults



◆ **Faults classified by behavior**

- Fail silent - component stops producing outputs when it fails
- Fail symmetric - the fault results in the same erroneous value being sent to all other redundant lanes
- Fail asymmetric (Byzantine) - the fault results in different erroneous values being sent to parts of the system

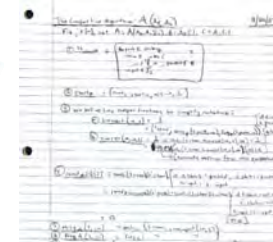
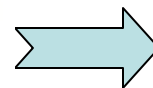
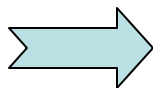
◆ **Faults classified by temporal characteristics**

- Permanent faults - once they occur they do not disappear (i.e. power converter failure)
- Transient faults - appear for a short time and then disappear (i.e. radiation single event upset or transient)
- Intermittent faults - they appear, disappear and then reappear (i.e. loose cable connection)

◆ **Common cause (Single mechanism triggering multiple faults)**

- Logic errors in application software (i.e. mixed English and metric units in the software)
- Hardware errors (i.e. common bug in a processor)
- Operating system or compiler errors
- Manufacturing defects
- Environmental induced effects (i.e. MMOD, Lightning, EMI, total dose radiation)

Fault Tolerance



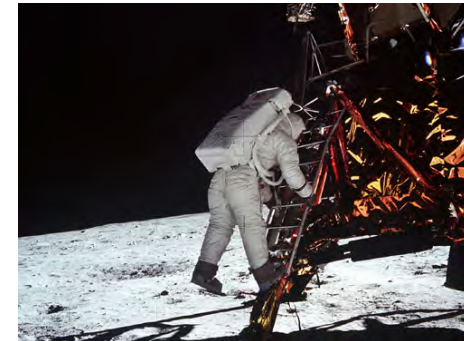
- ◆ **Degradable Redundant Dissimilar Hardware**
 - LCD Projector → transparency overhead → notes/chalk
- ◆ **Low Power/High reliability/Low Tech Backup**
 - Notes/chalk
- ◆ **Alternate Power Sources**
 - Electric vs Chalk
- ◆ **Standby Redundancy**
 - Light bulb, extra chalk
- ◆ **Methods not employed**
 - Test like you fly
 - Fault masking (hot backup)
 - Repair methodology
 - No software backup
 - Physical separation



What is the right approach to FT?

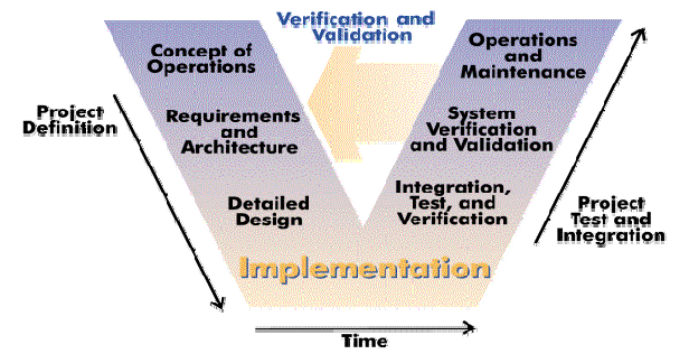
◆ **What are the ConOps? What are your system requirements?**

- Human-rated system?
- Real-time fault masking required?
- System reliability?
- Mission duration?
- Can the system be serviced?
- Constraints (SWaP – Size, Weight & Power)?



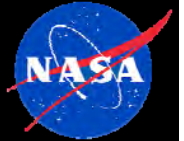
◆ **These are important questions to get answer to up front (if possible)**

- The appropriate fault-tolerance approach should be part of the systems engineering process trade space and may often involve system trades beyond avionics.





Constellation Specifics (Ares, Orion mostly)



- ◆ **Human-rated systems**
- ◆ **Designing to PLOC and PLOM numbers for various missions**
- ◆ **Real-time fault masking needed for dynamic phases of flight**
 - Both random and common cause faults
- ◆ **Mass & Power are a system drivers**
 - Support for low power operation (both crewed and un-crewed)
 - Hard choices mass vs redundancy are being made
- ◆ **Mission duration/characteristics**
 - Ares: ~ 12 minutes (highly dynamic)
 - Orion: ~ 6 months for outpost mission (variety of quiescent and dynamic phases, crewed/un-crewed operations)
- ◆ **No in-flight servicing/replacement/sparing**
 - This will not be the case for long duration lunar surface systems

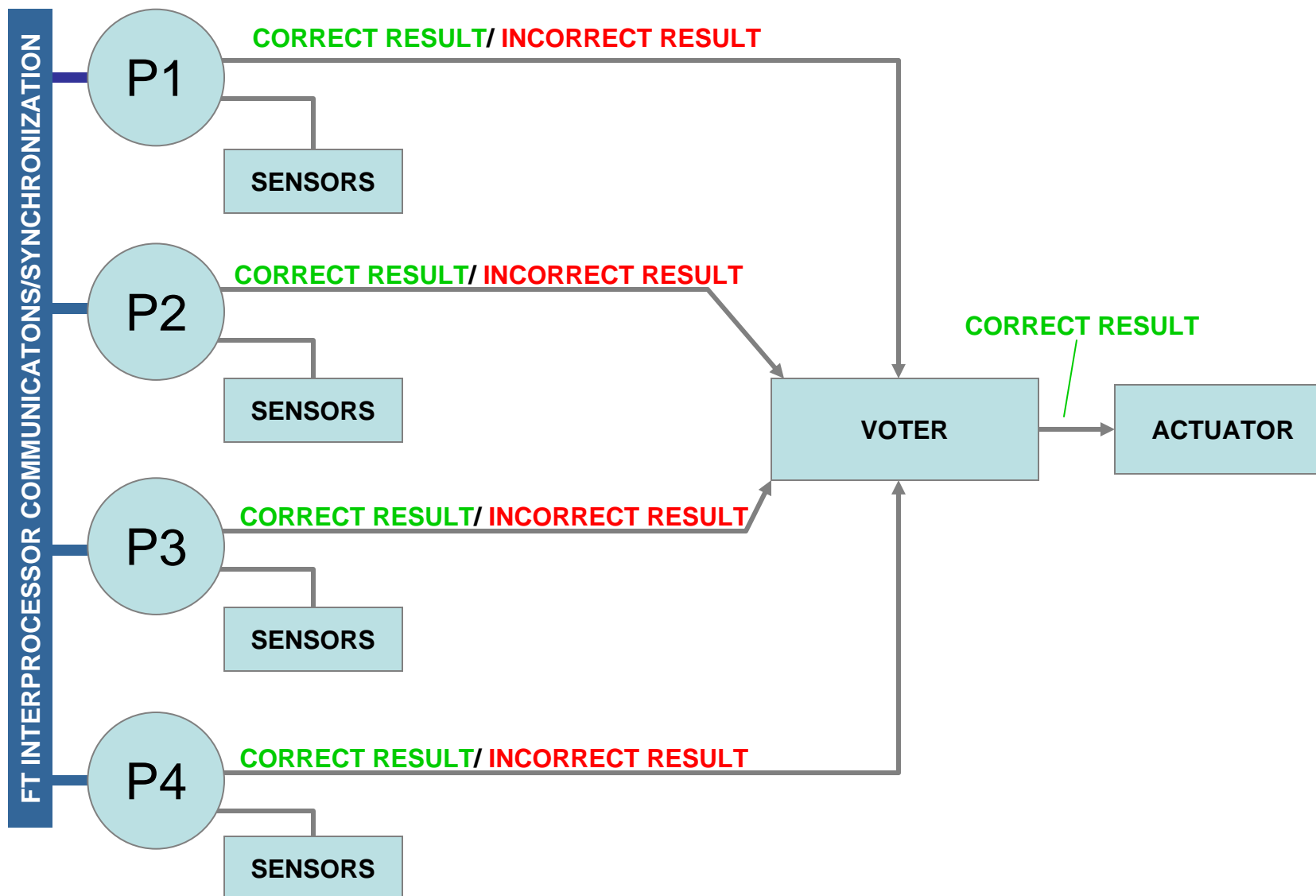


Fault Mitigation Techniques

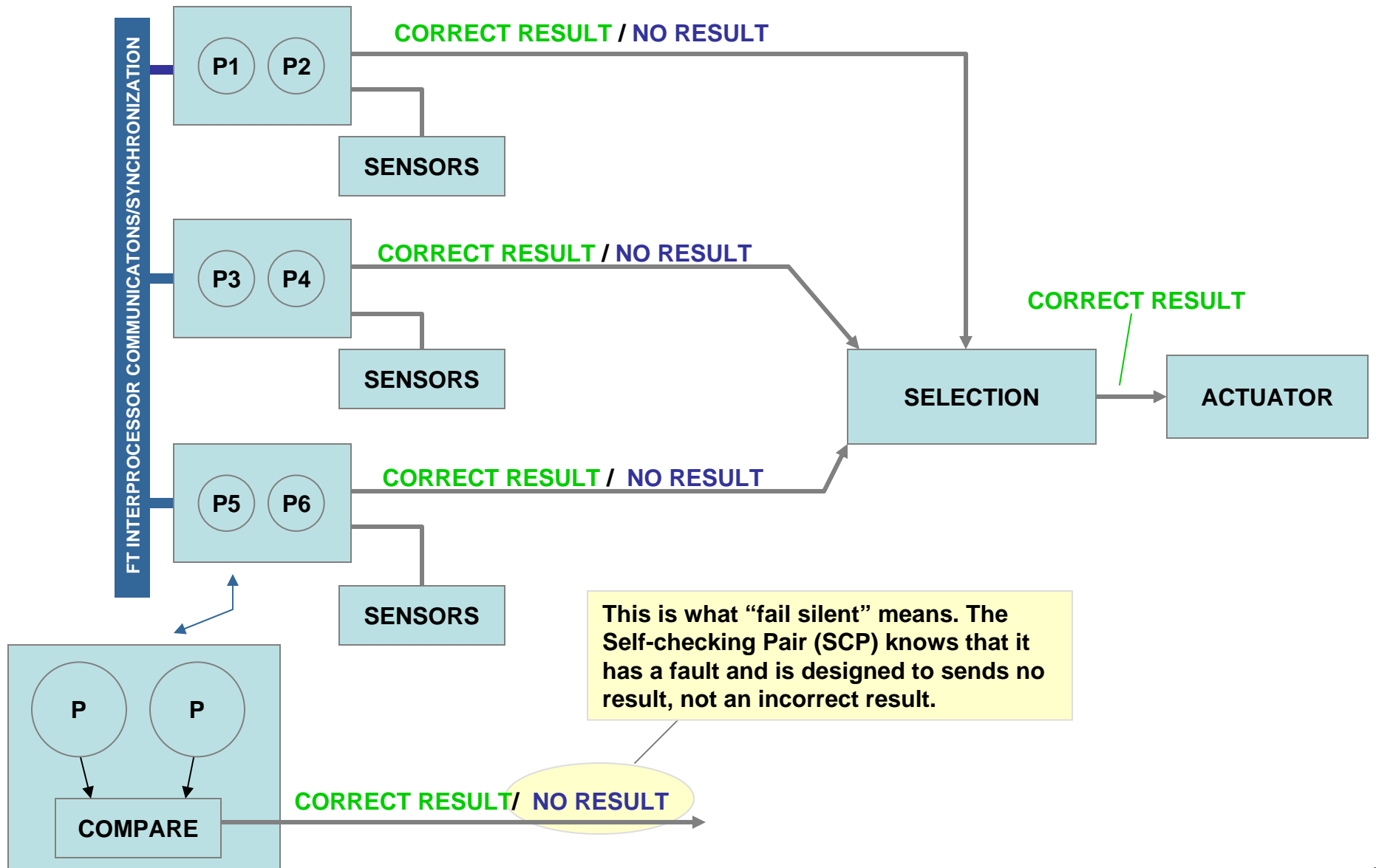


- ◆ **Systems-level similar redundancy**
 - Real-time voting
 - Fail-silent with selection
- ◆ **Dissimilar redundancy**
 - Hardware (computers, interconnects, sensors, etc.) and software (application layer)
- ◆ **Temporal redundancy**
 - Multi-phase commands (arm, fire), multiple samples of inputs
- ◆ **Analysis: FMEA/FMECA, Formal modeling/validation**
- ◆ **Physical fault containment**
- ◆ **Software fault containment**
 - ARINC 653 Time/Memory partitioning
- ◆ **EDAC, Scrubbing, CRCs, device level TMR, self-checking, shielding**
- ◆ **Extensive TV&V and part of the system engineering and certification processes**
 - Device through System-level, incremental verification
 - Simulation/modeling, fault injection
 - “Test as you Fly” (Actually test much more than that because most of the contingency cases you hope to never fly)

Voting Example



Fail-Silent Example





Dissimilarity & Common Cause



- ◆ **Often debated**
- ◆ **Will add cost**
- ◆ **May not work if not done right**
 - Dissimilar code will still have logical errors if derived from incorrect requirements
 - Requirement-level dissimilarity needed
 - Can a single string dissimilar backup over-ride multiple primary strings?
 - Dilemma case – two disagree – who's right
 - Who decides? Automated switch over? Human in the loop?
 - On automated switch-over, is that a single point system failure?
- ◆ **Where is the common cause risk?**
 - New hardware → high risk?
 - New code/algorithm → high risk?
 - Complexity of the system – goes to the ability to test effectively
- ◆ **Common cause failure is not a simple issue**
 - SAE ARP 4761 outlines guidelines for conducting safety assessment for civil airborne systems. This standard enumerates a total of 8 common mode types, 22 common mode subtypes with between 2 and 9 example sources for each subtype.



A few parting thoughts



- ◆ **Pay particular attention to interface/boundaries**
 - Both system and programmatic
- ◆ **Pay attention to corner cases (timing, environments, startup)**
- ◆ **It is never too late to get it right**
 - Speak up, ask questions
 - It will cost more later
- ◆ **Plan for the unexpected, build in robustness & capability when possible**
- ◆ **Don't believe absolute reliability numbers but use them as a tool to identify areas to improve**
 - Understand reliability assumptions
- ◆ **The only guarantee when adding redundancy to a system is an increase in the fault arrival rate**

– J. H. Lala et. al., *A Design Approach for Ultrareliable Real-Time Systems*