Fault Tree Generation and Augmentation

Ann T. Tai, Chris J. Walter, and Robert Insley
WW Technology Group, Maryland, USA

The 5th annual Safe and Secure Systems and Software Symposium (S5)

Fault-Tree Generator: Current State of the Art

- **An important merit in common:**

  Source Information concerning faults are extracted from a design model specified in standardized design language, enabling efficient fault tree model updates and evolution.

- **A shared critical shortcoming:**

  Relationships between fault events are derived primarily from data/architectural dependencies specified in design models, neglecting fault types and the respective fault-management (FM) coverage.

A Lesson Motivated Our Effort: Ariane-5 Failure

- An inappropriate reuse of the alignment software from Ariane-4 in the Inertial Reference System (SRI): Ariane-5 had a higher initial acceleration and had a trajectory which led to a build-up of horizontal velocity which was five times more rapid than for Ariane-4.
- The higher horizontal velocity of Ariane-5 generated an out-of-range value, triggering an exception. Furthermore, replication-based dual redundancy was the sole FM technique applied to the system, causing the alignment software to be unprotected and eventually leading to rocket self-destruction.
- With pure architectural decomposition, a misleading fault tree which is unable to reveal the FM inadequacy is likely to be built. Consequently, that misleading fault tree would make us have a false sense of safety and overlook the FM inadequacy.

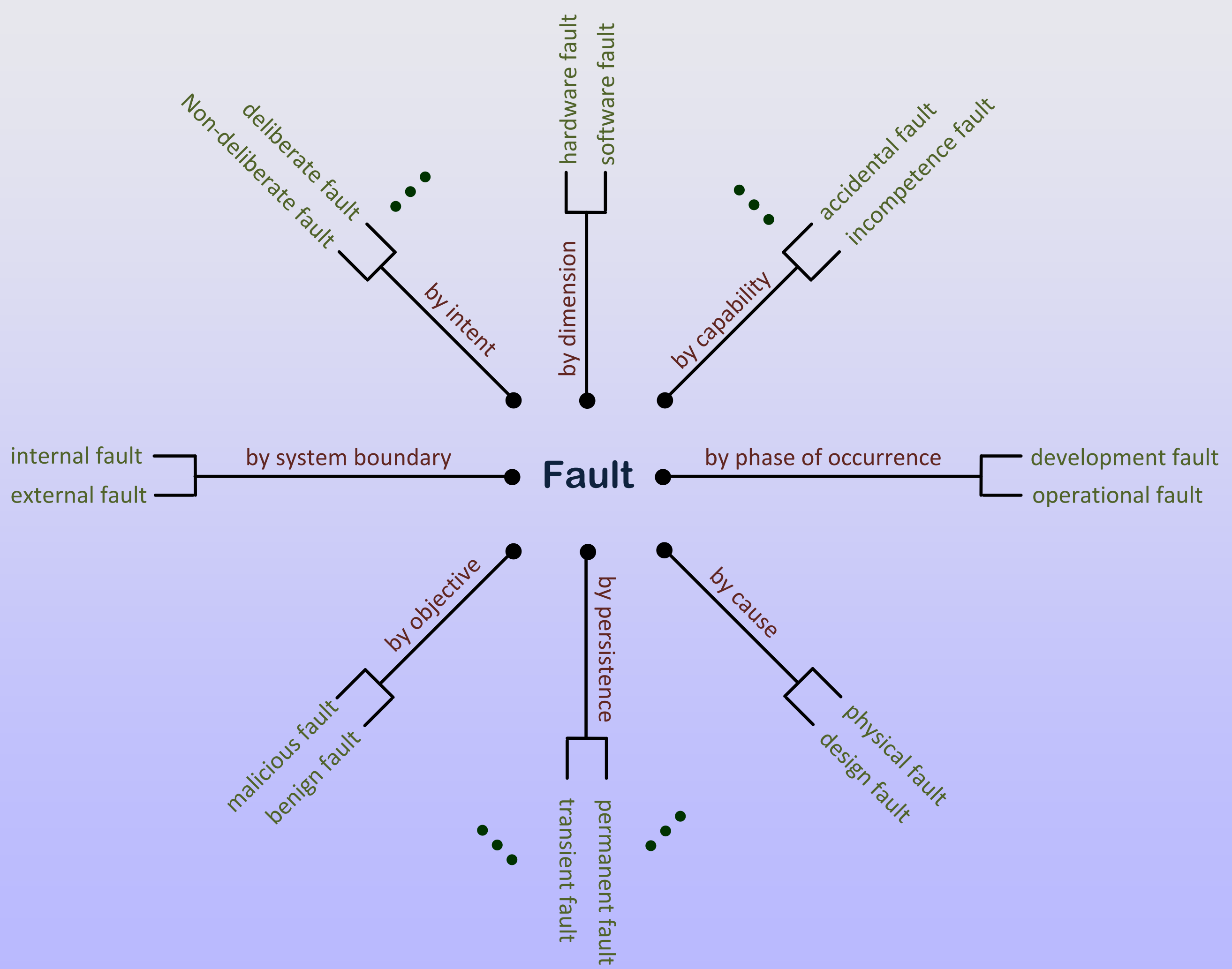
What we Learned from Ariane 5 Failure

- Replication enables a system to tolerate random physical faults in devices but not software design faults (e.g., arithmetic overflow caused by “mis-reuse”), implying awareness of conflict of FM applicability is important.
- Fault-class-oriented decomposition is beneficial in fault tree generation to expose otherwise hidden failure vulnerability, especially for mission-critical systems equipped by FM mechanisms.

GOAL: FAULT-CLASS-AWARE FAULT-TREE GENERATION & ANALYSIS

- Go beyond mechanical translation
  - Awareness of fault class and FM coverage limitation during tree generation.
  - Prioritize fault-class-oriented decomposition over pure architectural decomposition.
- Go beyond faults in application systems
  - Model-based FM scheme applicability checking.
  - Vigilant about critical faults in the use of FM schemes.
  - Enabling the exposure of the faults that are not covered due to inappropriate FM application.

Multifaceted Fault Reference Model



Fault Class Matrix

	Physical fault	Design fault	Permanent fault	Transient fault	Hardware fault	Software fault
Physical fault	●		■	●	●	
Design fault		■	●		●	●
Permanent fault	■	●	●		●	●
Transient fault	●			■	●	
Hardware fault	●	●	●	●	●	
Software fault		●	●			●

- Classification by cause
- Classification by persistence
- Classification by dimension

■ Anchor fault class (simple or composite)

● Non-anchor fault class

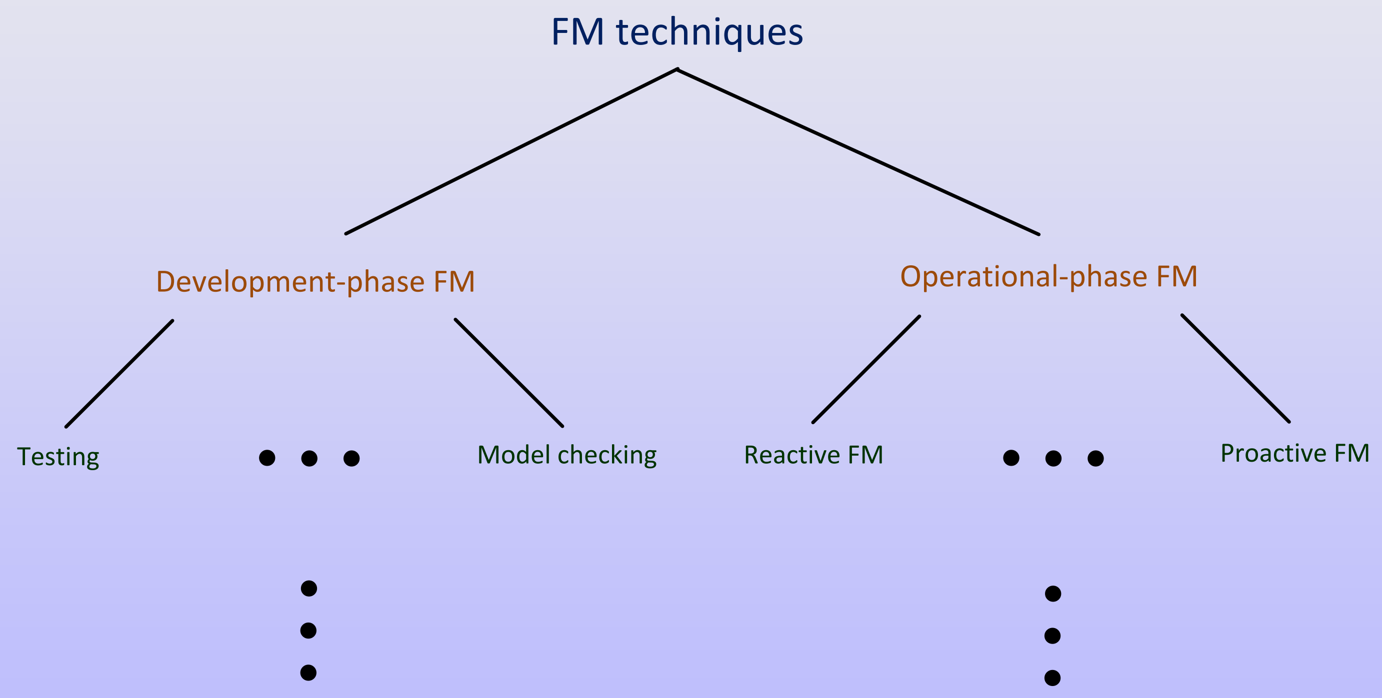
ANCHOR FAULT CLASS: AN INFORMAL DEFINITION

An anchor class is a fault class *FC* whose characteristics specification is adequate for us to identify a set of feasible FM methods *S* which satisfies the following conditions:

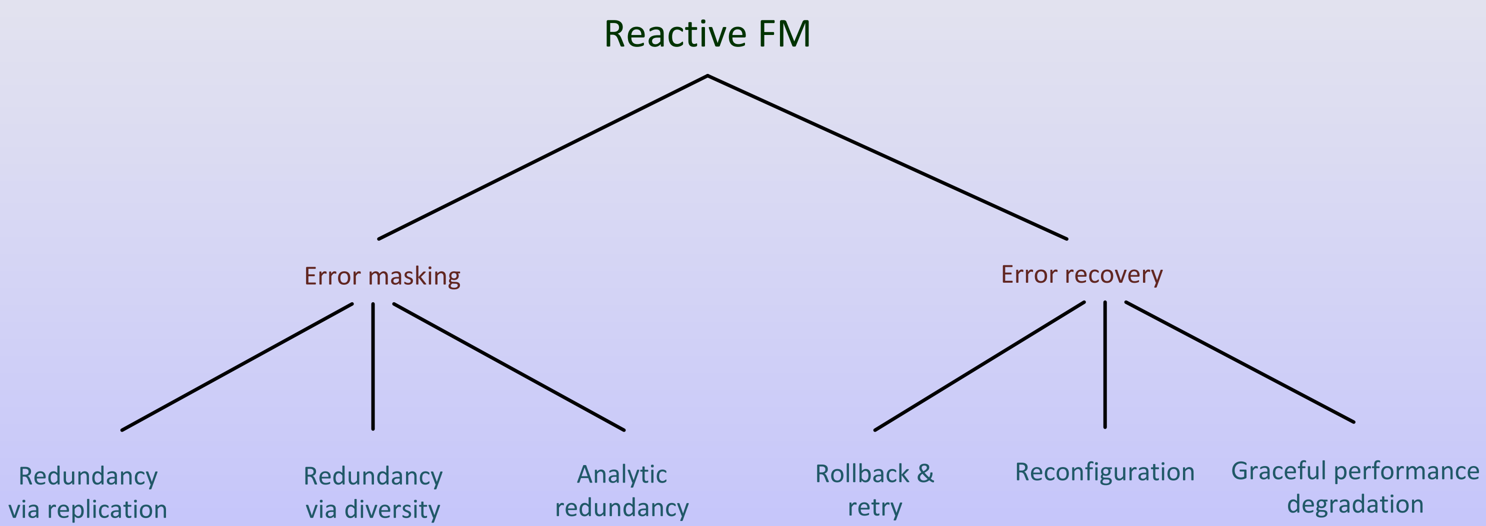
1) Set *S* is exhaustive with respect to *FC* (i.e., including all the methods feasible to *FC)*, and
2) Each method in *S* is feasible not only to *FC* itself but also to all the composite fault classes involving *FC**.

* Can be also viewed as sub-fault-classes of FC when one of the classification criteria is considered as the primary criterion.

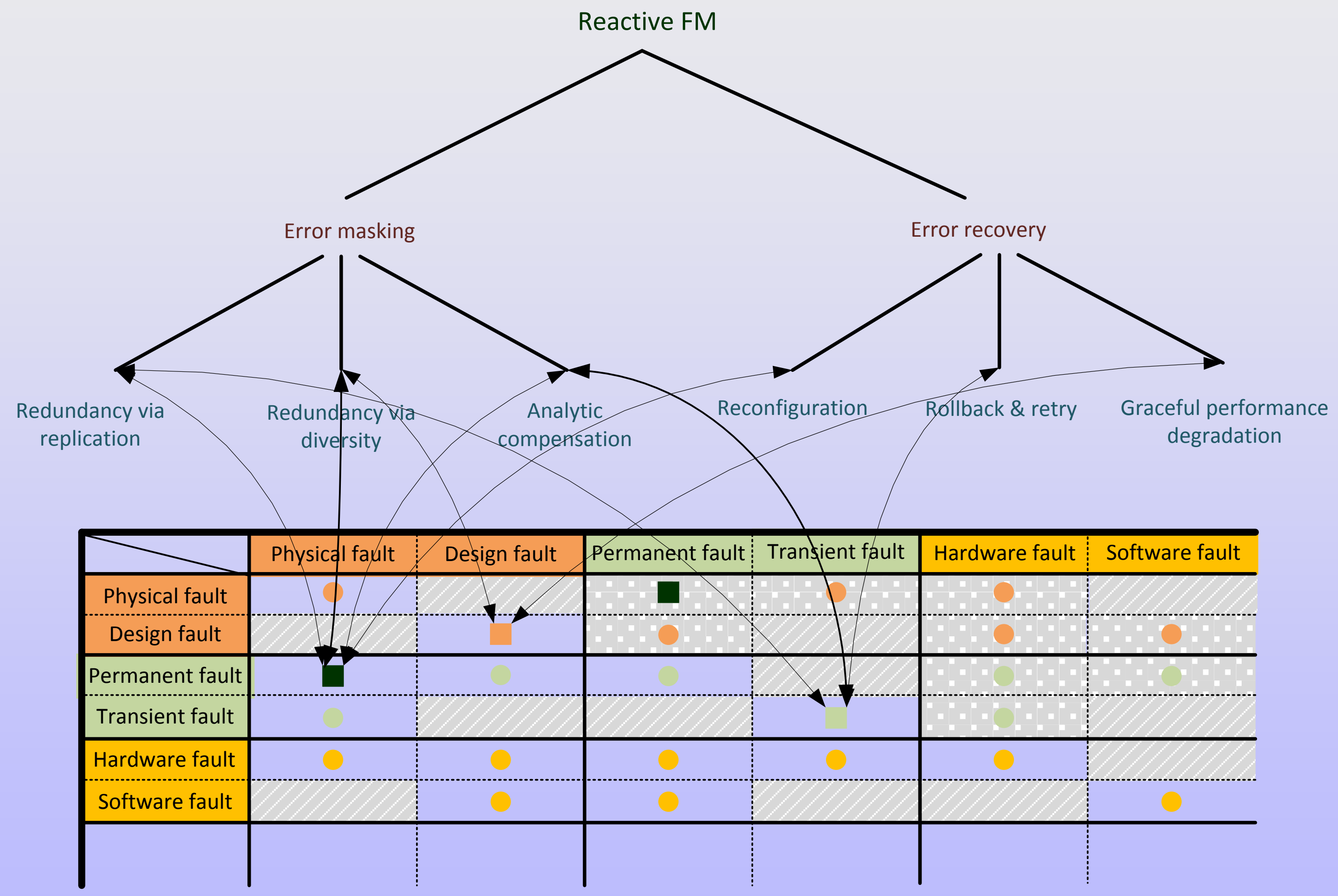
FM Technique Hierarchy (I)



FM Technique Hierarchy (II)



Fault Classes & FM Techniques



Classification by cause

Classification by persistence

Classification by dimension

Anchor fault class (simple or composite)

Non-anchor fault class

AADL Design Model: Ariane-5 Example (I)

```
package systemET
public
annex EMV2 {**
error types
  physicalFault: type;
  designFault: type;
  hardwareFault: type extends physicalFault;
  softwareFault: type extends designFault;
end types
**}
end systemET
package systemEM
public
annex EMV2 {**
error behavior embeddedSystem
use types systemET;
events
deviceError: error event {hardwareFault};
softwareError: error event {softwareFault};
states
operational: initial state;
deviceFail: state;
softwareFail: state;
systemFail: state;
transitions
operational -[deviceError]-> deviceFail;
operational -[softwareError]-> softwareFail;
end behavior;
**}
end systemEM;
```

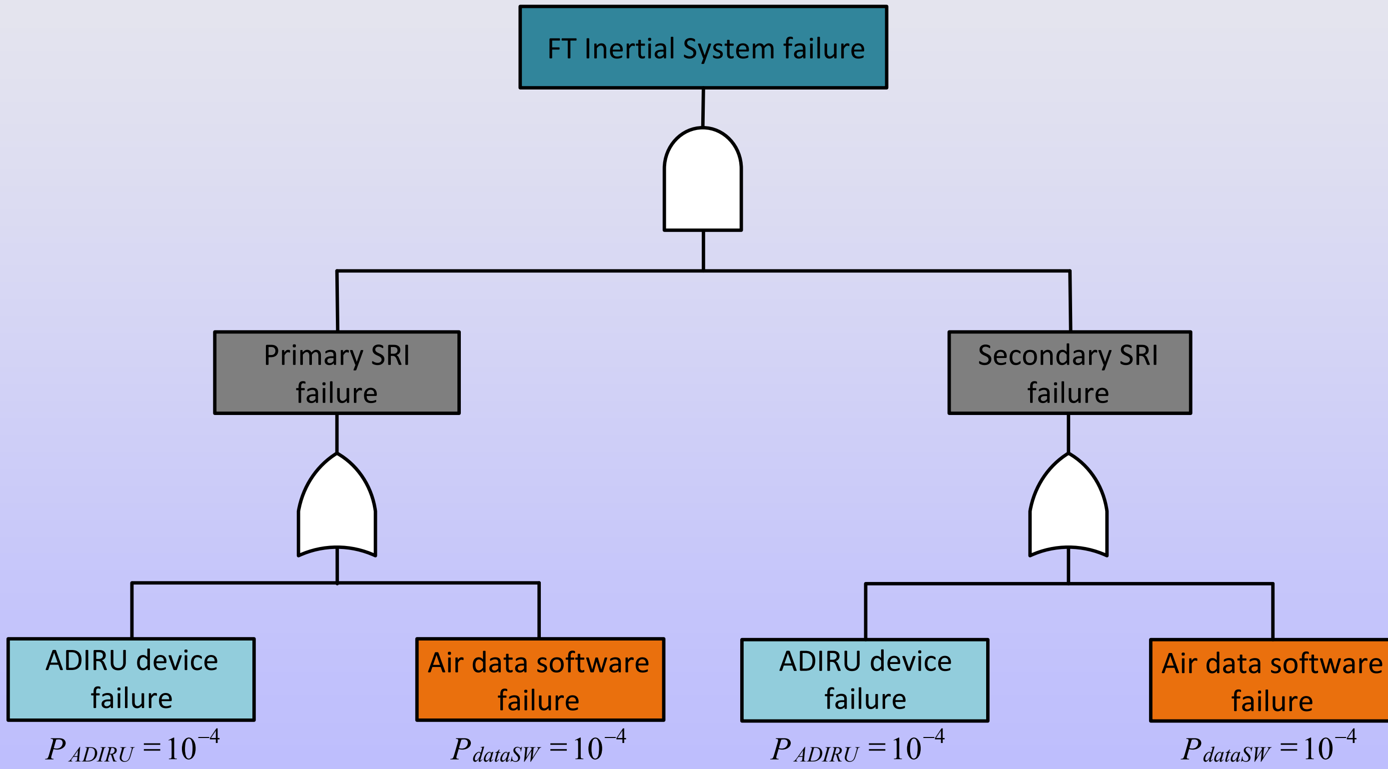
AADL Design Model: Ariane-5 Example (II)

```
system SRI
subcomponents
  D: ADIRU;
  A: airDataSW;
end SRI;
system implementation SRI.impl;
  annex EMV2 {**
  composite error behavior
  use types systemET;
  use behavior systemEM::embeddedSystem;
  composite states
  [D.operational and A.operational] -> operational;
  [D.deviceFail or A.softwareFail] -> systemFail;
  end composite;
  **}
end SRI.impl;
system ADIRU
end ADIRU;
system implementation ADIRU.impl
annex EMV2 {**
component error behavior
use types systemET;
use behavior systemEM::embeddedSystem;
transitions
operational -[deviceError]-> deviceFail;
**};
end ADIRU.impl;
```

AADL Design Model: Ariane-5 Example (III)

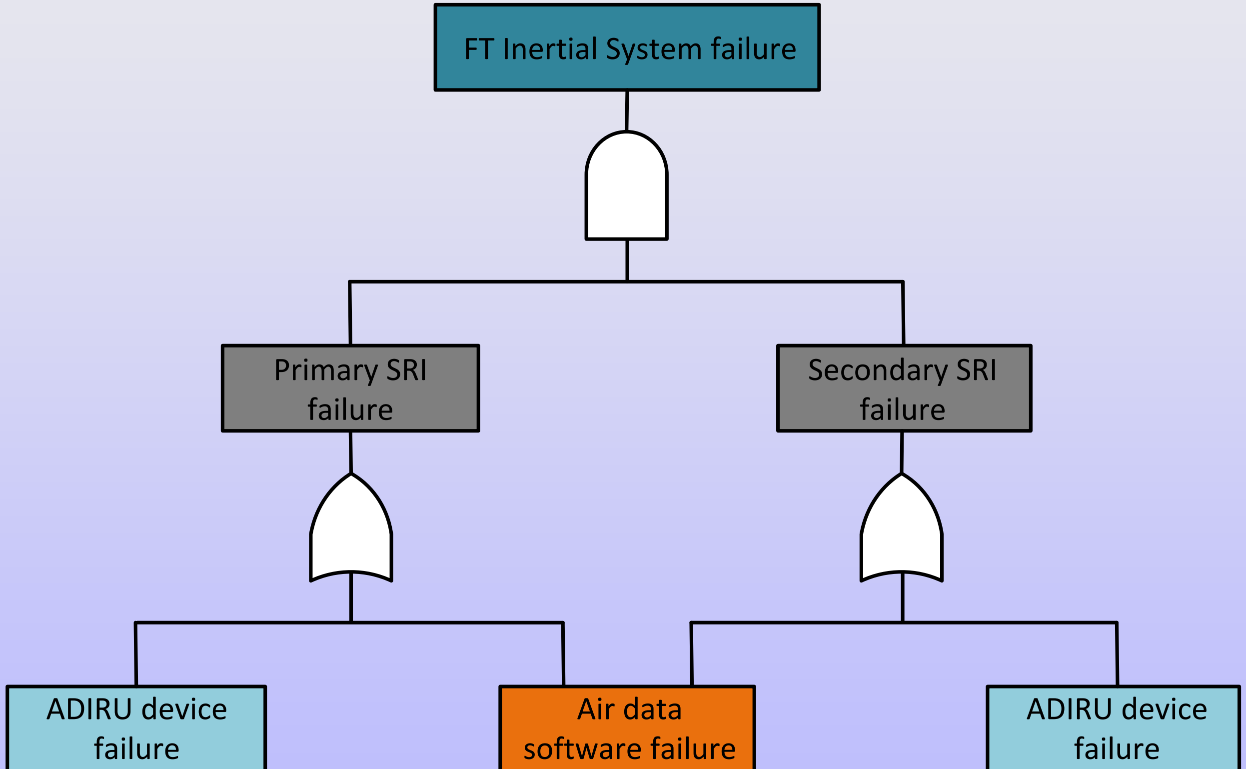
```
system airDataSW
end airDataSW;
system implementation airDataSW.impl
annex EMV2 {**
component error behavior
use types systemET;
use behavior systemEM::embeddedSystem;
transitions
operational -[softwareError]-> softwareFail;
**};
end airDataSW.impl;
system ftInertialSystem
end ftInertialSystem;
system implementation ftInertialSystem.impl
subcomponents
  U1: system SRI;
  U2: system SRI;
annex EMV2 {**
composite error behavior
use types systemET;
use behavior systemEM::embeddedSystem;
composite states
    [U1.operational and U2.operational
    or U1.operational and U2.systemFail
    or U1.systemFail and U2.operational] -> operational;
    [U1.systeFail and U2.systemFail] -> systemFail;
end composite;
**}
end ftInertialSystem.impl;
```

Without FC/FMC Awareness



$$P_{inertialSys} = (1-(1-P_{ADIRU})(1-P_{dataSW}))^2 = 4\times10^{-8}$$

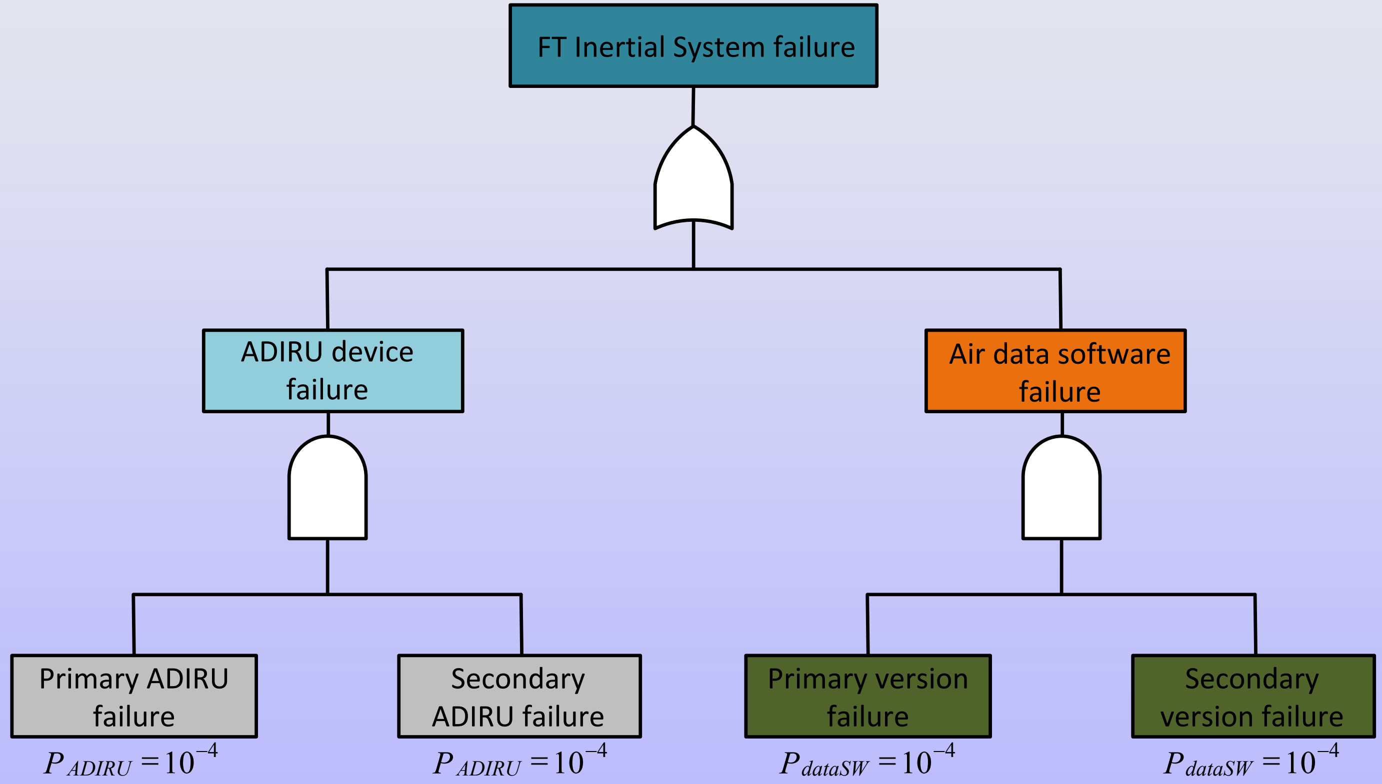
With FC/FMC Awareness



$P_{ADIRU} = 10^{-4}$ $P_{dataSW} = 10^{-4}$ $P_{ADIRU} = 10^{-4}$

$$P_{inertialSys} = 1 - (1 - P_{ADIRU}{}^2)(1 - P_{dataSW}) = 1 \times 10^{-4}$$

Fault Tree with Augmentation



$$P_{inertialSys} = 1 - (1 - P_{ADIRU}{}^2)(1 - P_{dataSW}{}^2) = 2\times10^{-8}$$

Summary of FTGA

- FTGA generates fault trees from standardized design models: Basic fault trees or advanced fault trees is user's choice.
- When an FM application is recognized from a design model, the system entity to which FM is applied will undergo a model checking and possibly a fault-class-based decomposition.
- By doing so, FM inadequacy or misuse would have a direct exposure in the resulting fault tree, raising a qualitative and quantitative alarm to the design team.
- The mapping between the fault class matrix and FM-method hierarchy enables fault tree augmentation (FM method insertion), allowing FTGA to play an active role in FM architecture rather than just a passive evaluation tool.