Architectural Approaches for using COTS Components in Critical Applications

David Powell[1] Jean-Paul Blanquart[2] Yves Crouzet[1] Jean-Charles Fabre[1]

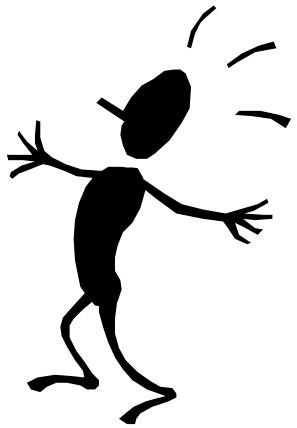
[1] LAAS-CNRS/LIS [2] Matra Marconi Space France/LIS

7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France

Pros and Cons of COTS Components [Arlat *et al.* 2000]

Pros and Cons of COTS Components [Arlat *et al.* 2000]

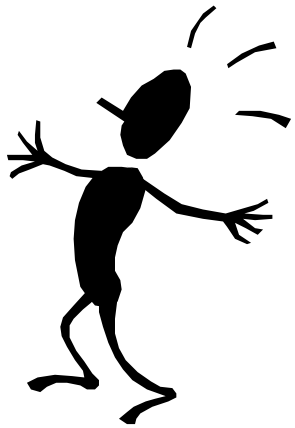
- **Pros**
  - Often de facto standards
  - Lower development cost
  - Shorter development time
  - Rapid incorporation of technological progress
  - Reliability data from extensive previous use



Pros and Cons of COTS Components [Arlat *et al.* 2000]

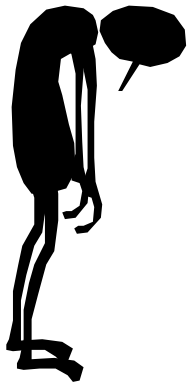
Pros

- Often de facto standards
- Lower development cost
- Shorter development time
- Rapid incorporation of technological progress
- Reliability data from extensive previous use



Cons

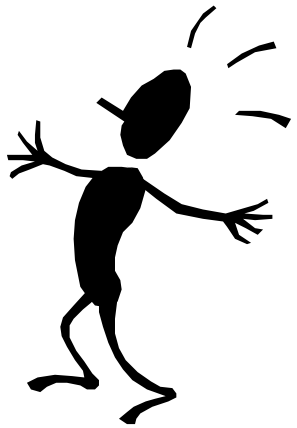
- No guarantee of continued support from supplier
- No guarantee of compatibility of successive versions
- Little objective information about the component itself or its development process
- Component may not be well-adapted to requirements
- Inheritance of unwanted or even unknown functions
- Uncertain quality



Pros and Cons of COTS Components [Arlat *et al.* 2000]

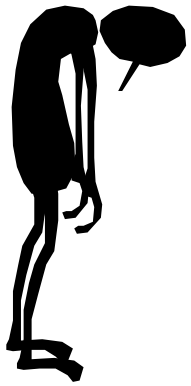
Pros

- Often de facto standards
- Lower development cost
- Shorter development time
- Rapid incorporation of technological progress
- Reliability data from extensive previous use

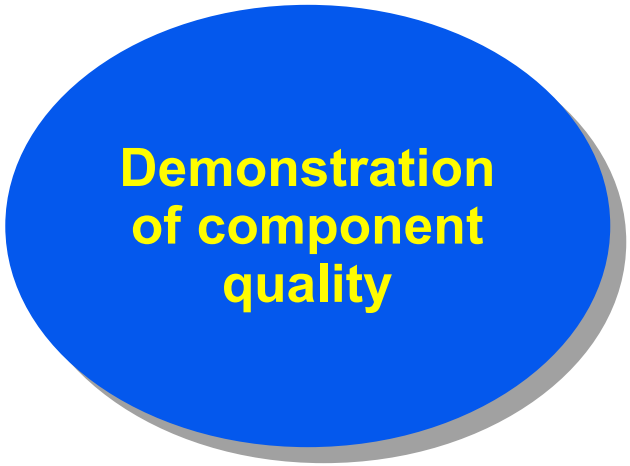


Cons

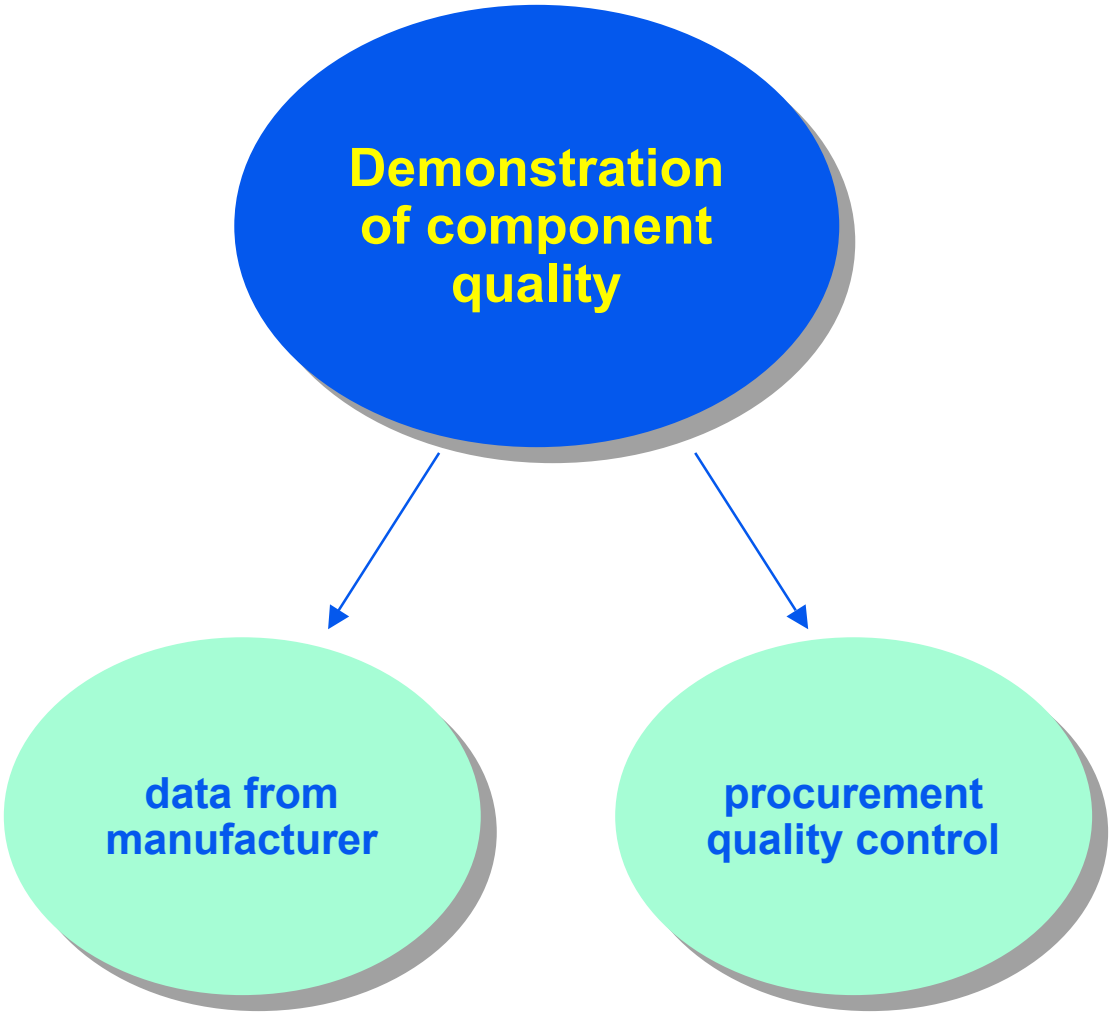
- No guarantee of continued support from supplier
- No guarantee of compatibility of successive versions
- Little objective information about the component itself or its development process
- Component may not be well-adapted to requirements
- Inheritance of unwanted or even unknown functions
- Uncertain quality



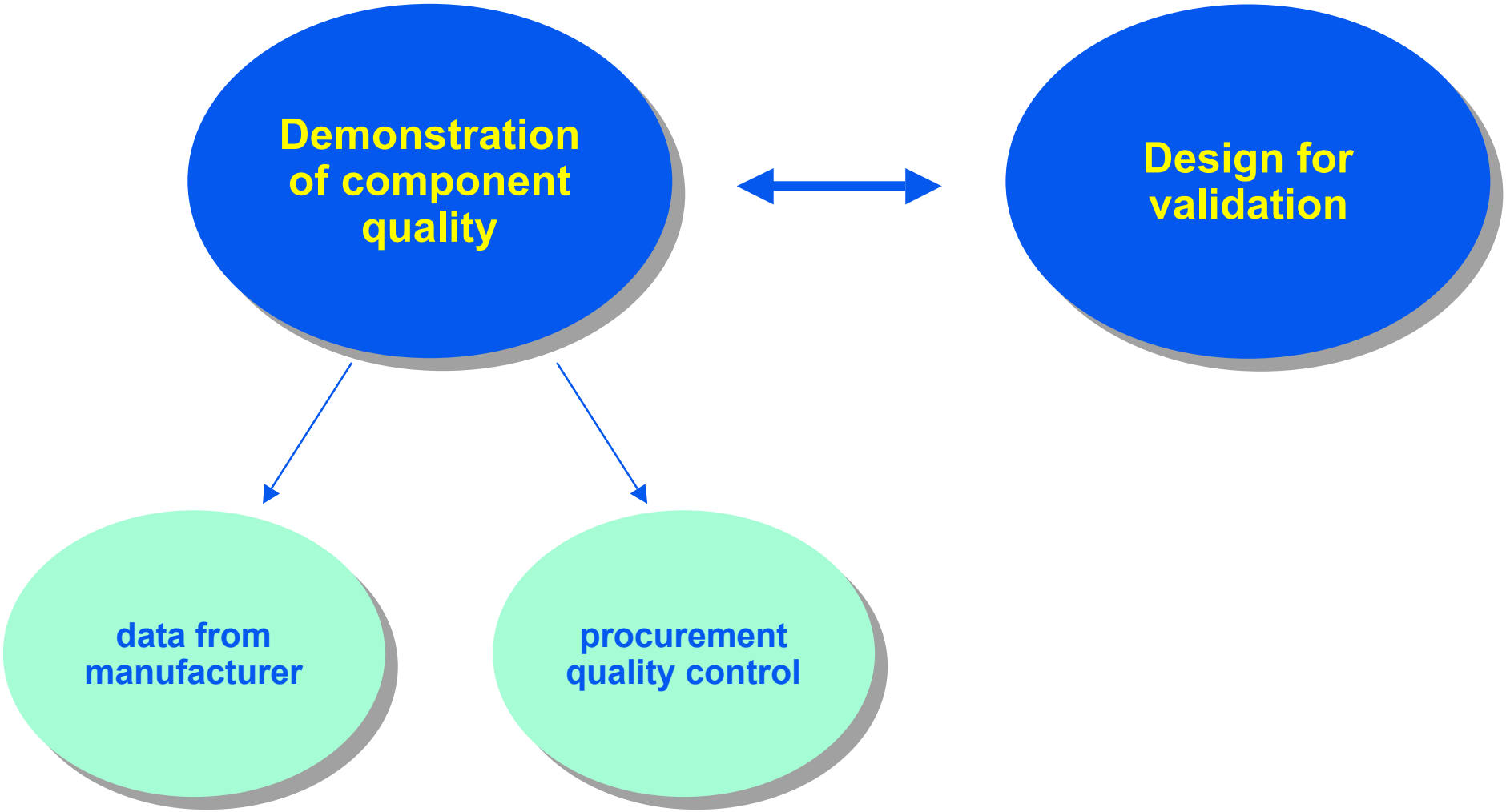
Qualification of Critical Systems



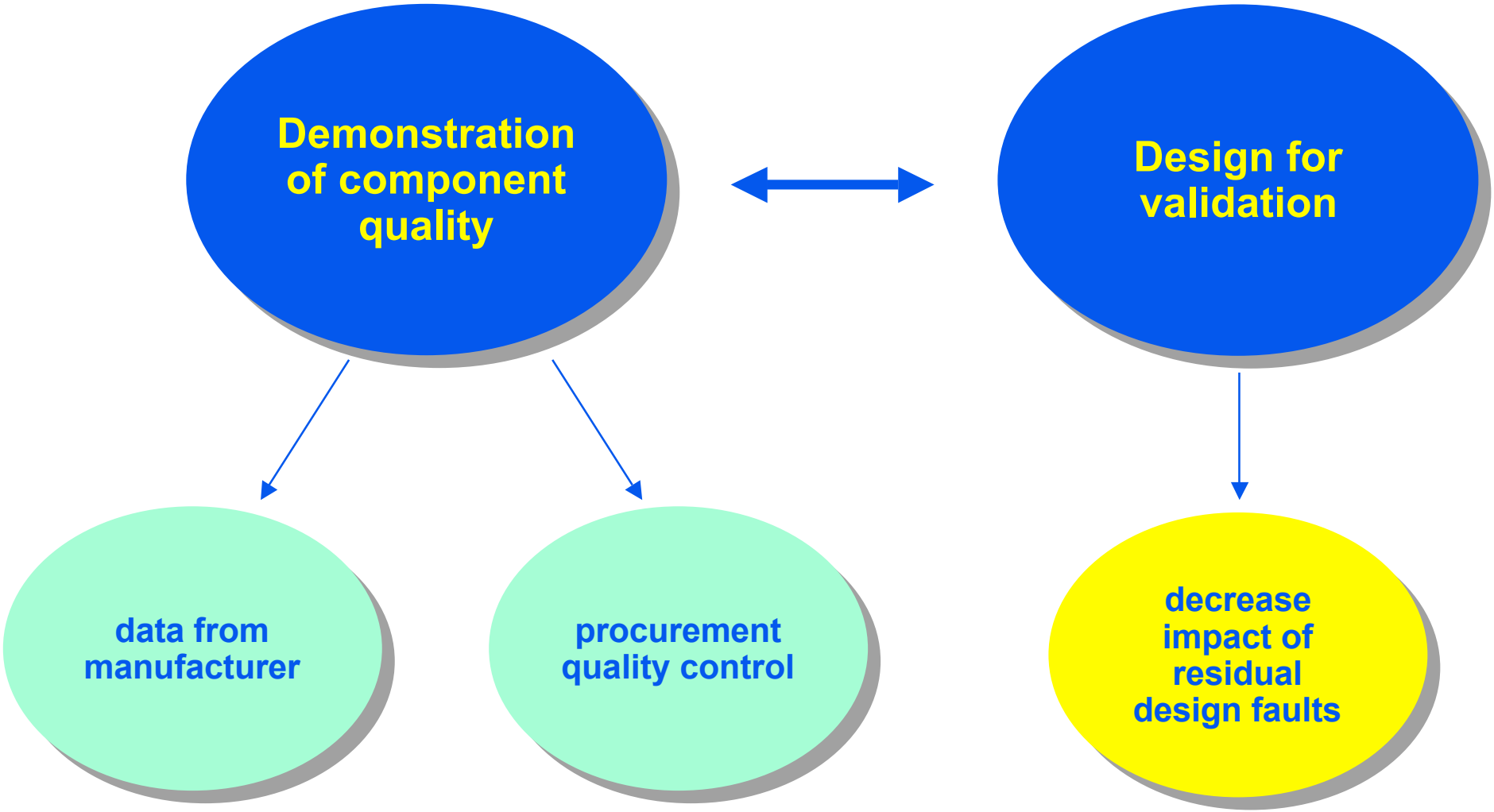
Qualification of Critical Systems



Qualification of Critical Systems

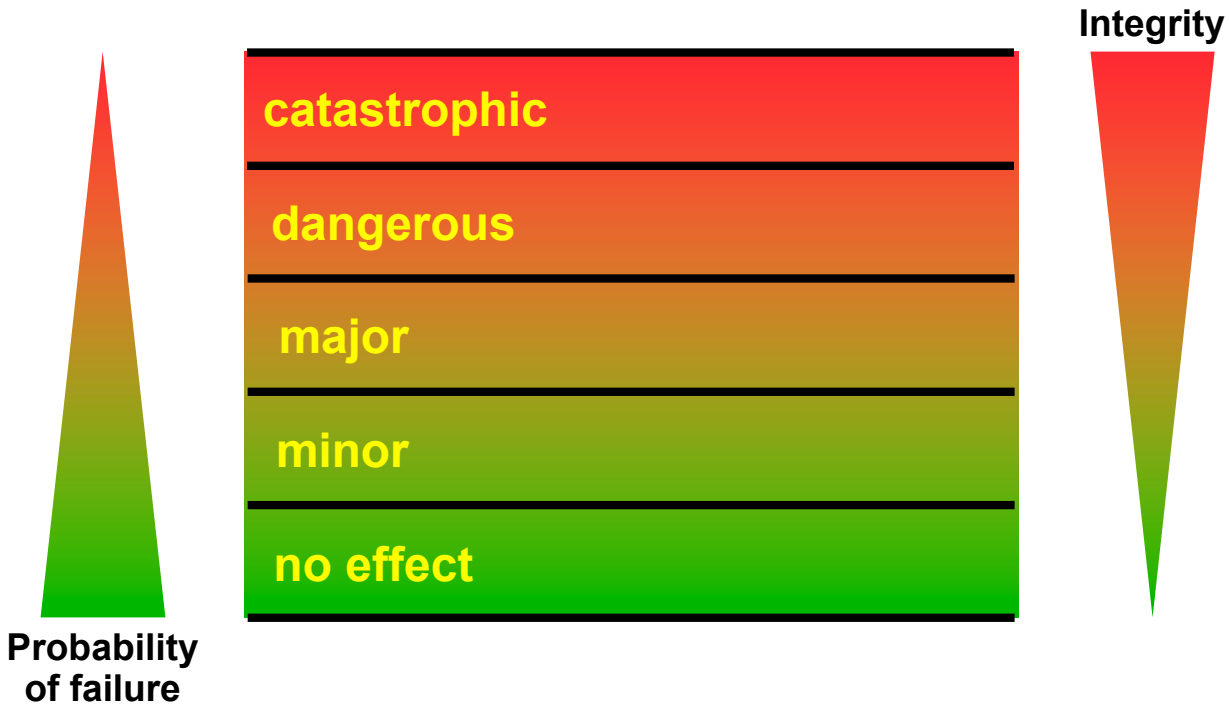


Qualification of Critical Systems



Designing Critical Systems

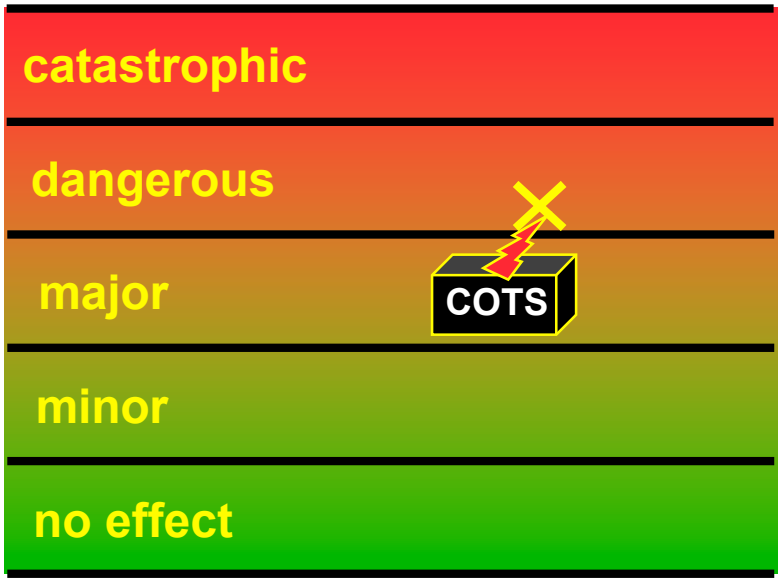
- Functional decomposition
- Categorise functions according to their criticality, i.e., in terms of the consequences of function failure
  - ➥ e.g. [DO-178B] — catastrophic, dangerous, major, minor, no effect
- Validate components to a confidence degree commensurate with function criticality



Designing COTS-Based Critical Systems

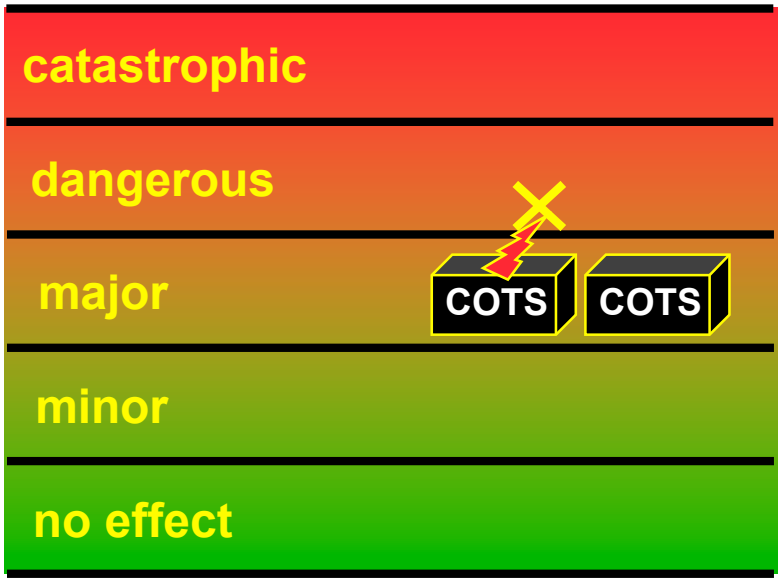
- Functional decomposition
- Categorise functions according to their criticality, i.e., in terms of the consequences of function failure
  - ➥ e.g. [DO-178B] — catastrophic, dangerous, major, minor, no effect
- Validate components to a confidence degree commensurate with function criticality

- Confidence in COTS appropriate for criticality of considered function
  - ➥ Still need to ensure that residual design faults do not affect functions of higher criticality
- Degraded service + error confinement
  - ➥ Replication & activation decorrelation
  - ➥ Functionally-independent error detection schemes (inc. diversification)
  - ➥ Partitioning
  - ➥ Wrapping



Designing COTS-Based Critical Systems

- Functional decomposition
- Categorise functions according to their criticality, i.e., in terms of the consequences of function failure
  - ➥ e.g. [DO-178B] — catastrophic, dangerous, major, minor, no effect
- Validate components to a confidence degree commensurate with function criticality



- Confidence in COTS not appropriate for criticality of considered function
  - ➥ Attempt to ensure continued operation of function despite residual design faults

    (and ensure that functions of higher criticality are not affected when continued operation is not possible)
- Continued service
  - ➥ Replication & activation decorrelation
  - ➥ Diversification

Designing COTS-Based Critical Systems

- Functional decomposition
- Categorise functions according to their criticality, i.e., in terms of the consequences of function failure
  - ➥ e.g. [DO-178B] — catastrophic, dangerous, major, minor, no effect
- Validate components to a confidence degree commensurate with function criticality

- Confidence in COTS appropriate for criticality of considered function
  - ➥ Still need to ensure that residual design faults do not affect functions of higher criticality
- Degraded service + error confinement
  - ➥ Replication & activation decorrelation
  - ➥ Functionally-independent error detection schemes (inc. diversification)
  - ➥ Partitioning
  - ➥ Wrapping

- Confidence in COTS not appropriate for criticality of considered function
  - ➥ Attempt to ensure continued operation of function despite residual design faults
  - ➥ and ensure that functions of higher criticality are not affected when continued operation is not possible
- Continued service
  - ➥ Replication & activation decorrelation
  - ➥ Diversification

Replication & Activation Decorrelation

- **Simplest (but least effective) protection against design faults**
- **Principle**
  - ➥ ***Residual* design faults have activation conditions depending on subtle combinations of internal system states**
    - ➢ ***popularised through the term* Heisenbugs**
  - ➥ **Can use identical redundant components if activation conditions are sufficiently decorrelated**
- **Examples:**
  - ➥ **Re-execution of identical tasks in different execution contexts (Tandem)** [Gray 1986]
  - ➥ **Parallel execution of identical tasks in different execution contexts (Elektra)** [Erb 1989, Kantz & Koza 1995]
  - ➥ **Parallel execution of diversified tasks on identical processors (Airbus)** [Brière & Traverse 1993]

Functionally-Independent Error Detection

Functionally-Independent Error Detection

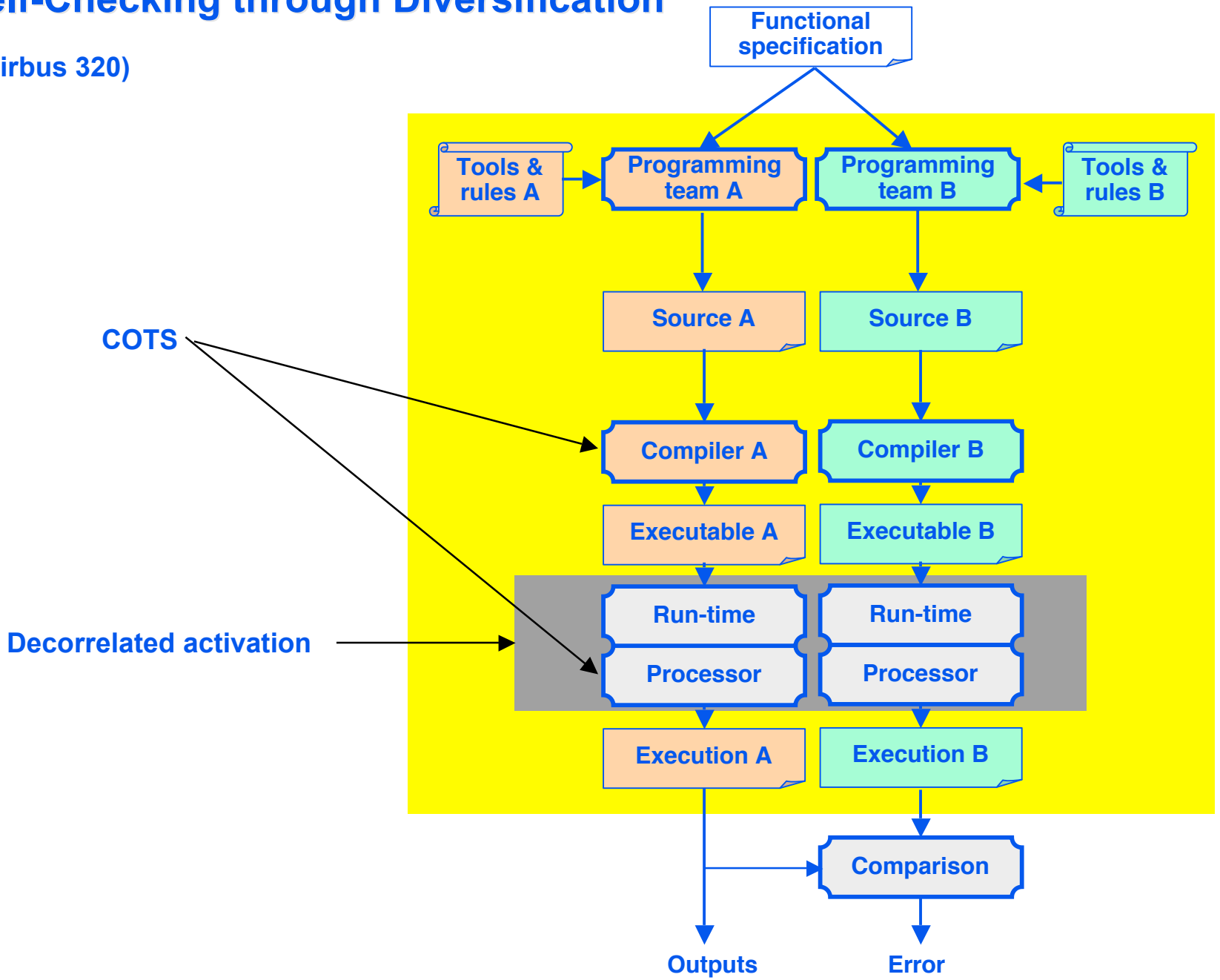
- Any error detection scheme other than duplication and comparison!

Functionally-Independent Error Detection

- Any error detection scheme other than duplication and comparison!
- **Self-checking through diversification** [Brière & Traverse 1993]
  - ➥ **At least two *variants* of a component are executed in parallel together with a comparison algorithm**

Self-Checking through Diversification

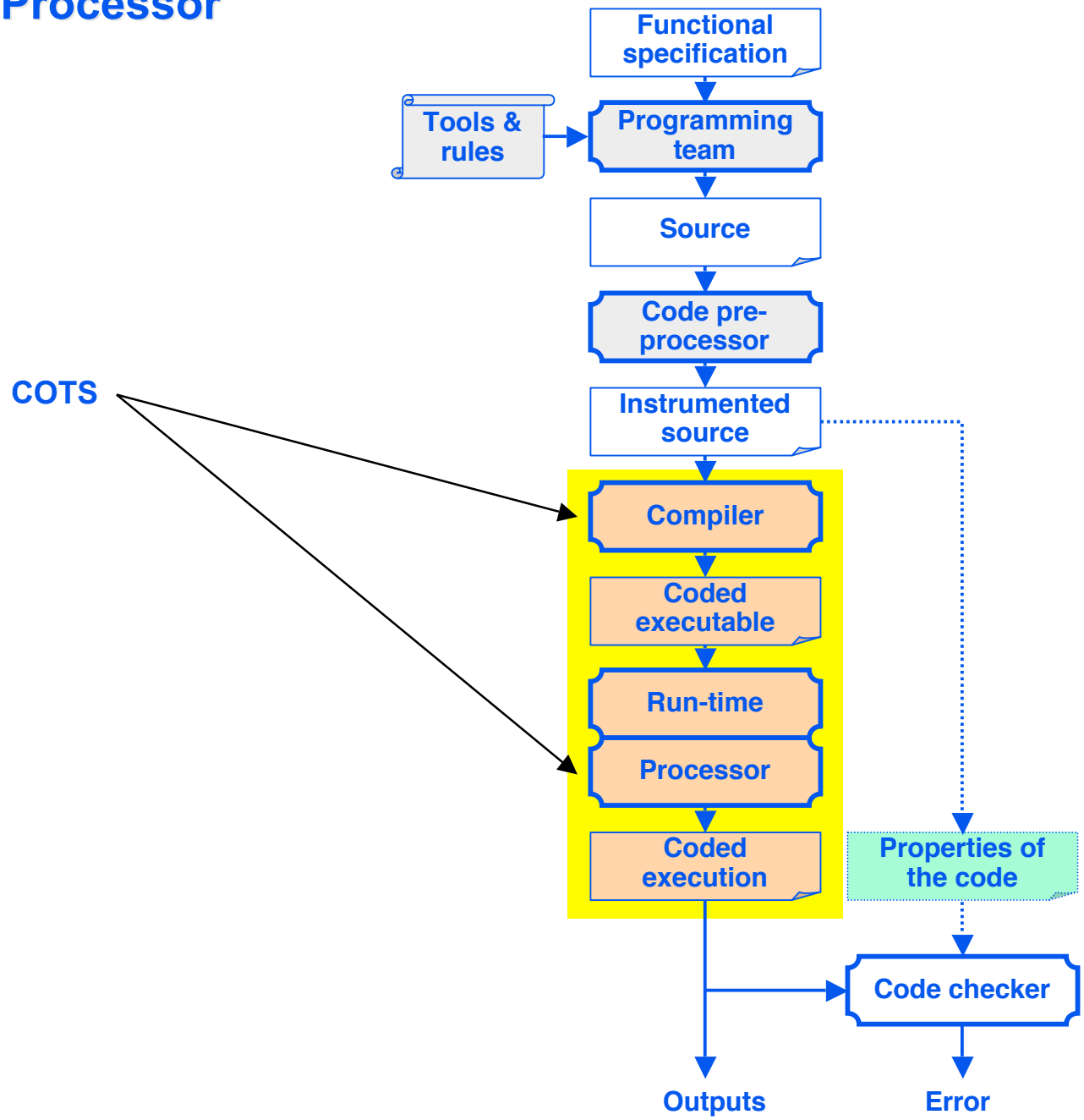
(Airbus 320)



Functionally-Independent Error Detection

- Any error detection scheme other than duplication and comparison!
- Self-checking through diversification [Brière & Traverse 1993]
  - ➥ At least two *variants* of a component are executed in parallel together with a comparison algorithm
- **Coded processor** [Forin 1989, Profeta *et al.* 1996]
  - ➥ **Arithmetic code for data storage, transfer and transformation errors**
  - ➥ **Signatures for sequencing and addressing errors**

Coded Processor

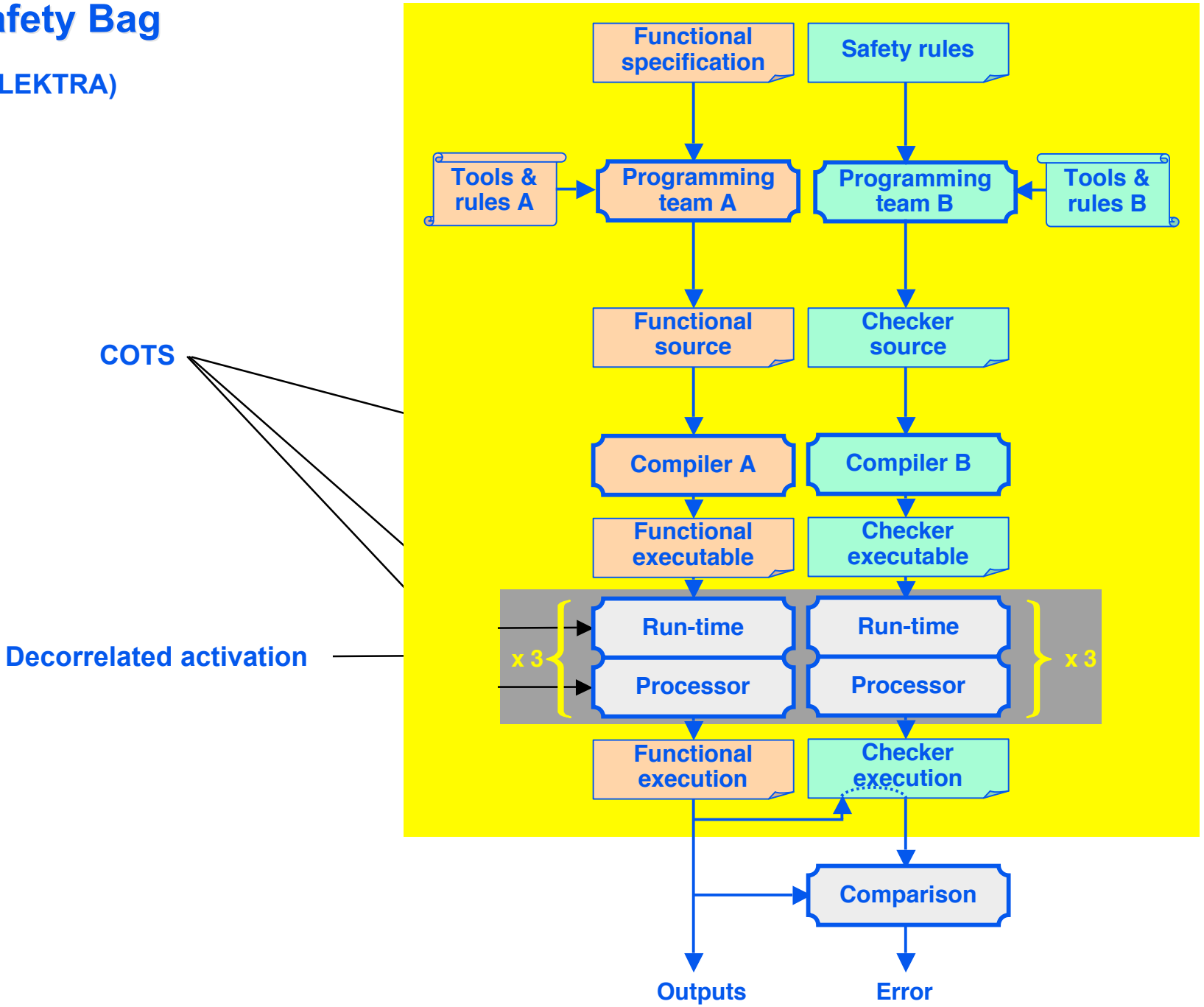


Functionally-Independent Error Detection

- Any error detection scheme other than duplication and comparison!
- Self-checking through diversification [Brière & Traverse 1993]
  - At least two *variants* of a component are executed in parallel together with a comparison algorithm
- Coded processor [Forin 1989, Profeta *et al.* 1996]
  - Arithmetic code for data storage, transfer and transformation errors
  - Signatures for sequencing and addressing errors
- **Safety bag (Elektra)** [Erb 1989, Kantz & Koza 1995]
  - **Independent channels for functional computation and checking computation**
  - **Checking computation (acceptance test) consists of executable assertions**

Safety Bag

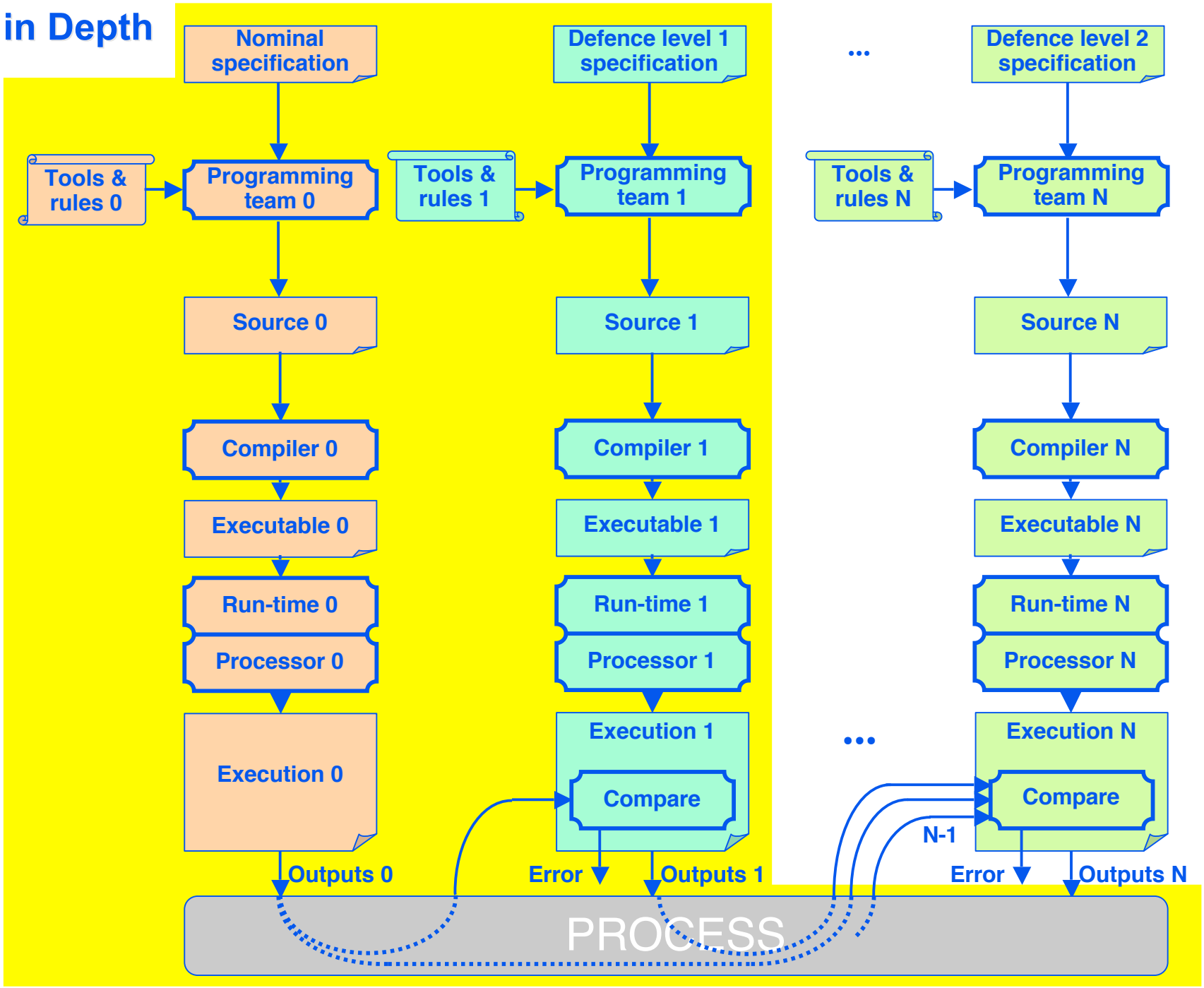
(ELEKTRA)



Functionally-Independent Error Detection

- Any error detection scheme other than duplication and comparison!
- Self-checking through diversification [Brière & Traverse 1993]
  - At least two *variants* of a component are executed in parallel together with a comparison algorithm
- Coded processor [Forin 1989, Profeta *et al.* 1996]
  - Arithmetic code for data storage, transfer and transformation errors
  - Signatures for sequencing and addressing errors
- Safety bag (Elektra) [Erb 1989, Kantz & Koza 1995]
  - Independent channels for functional computation and checking computation
  - Checking computation (acceptance test) consists of executable assertions
- **Defence in depth**
  - **Hierarchy of control channels with different levels of integrity and service (simpler service => higher integrity)**
  - **Each channel monitors the operation of lower integrity channels**

Defence in Depth



Partitioning

Partitioning

- **Aims to prevent propagation of errors due to design faults**
  - ➥ **from components ensuring low criticality functions (thus possibly low integrity)**
  - ➥ **to those ensuring high criticality functions (thus hopefully high integrity)**

Partitioning

- **Aims to prevent propagation of errors due to design faults**
  - from components ensuring low criticality functions (thus possibly low integrity)
  - to those ensuring high criticality functions (thus hopefully high integrity)
- **Requires**
  - Segregation of different levels of integrity
    - *physical isolation*
    - *logical isolation (in space and time)*
  - Strict rules regarding interaction between different levels of integrity
    - *no interaction*
    - *mediated interaction through enforcement of an appropriate integrity policy*

Partitioning

Aims to prevent propagation of errors due to design faults

- from components ensuring low criticality functions (thus possibly low integrity)
- to those ensuring high criticality functions (thus hopefully high integrity)

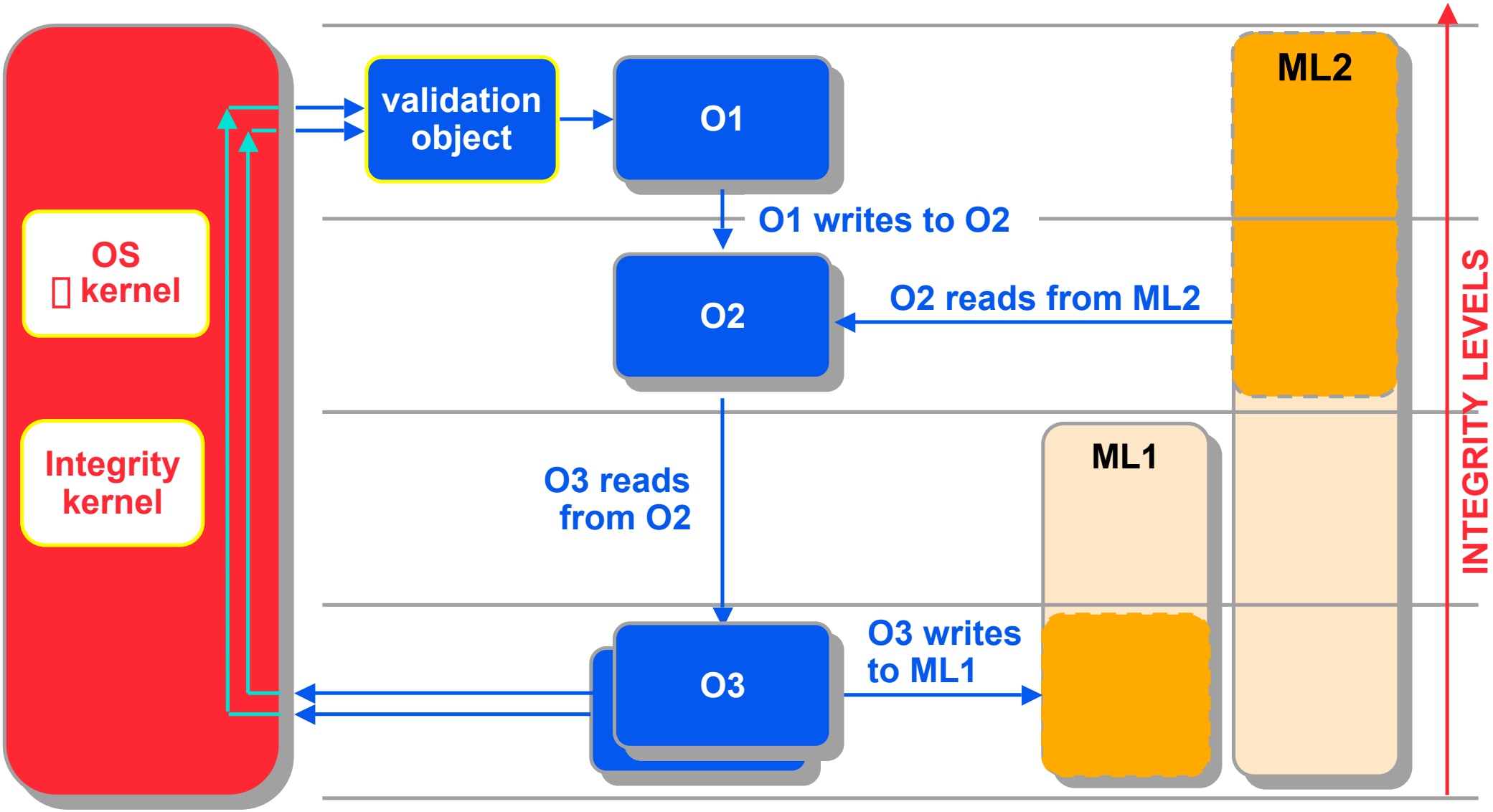
Requires

- Segregation of different levels of integrity
  - *physical isolation*
  - *logical isolation (in space and time)*
- Strict rules regarding interaction between different levels of integrity
  - *no interaction*
  - *mediated interaction through enforcement of an appropriate integrity policy*

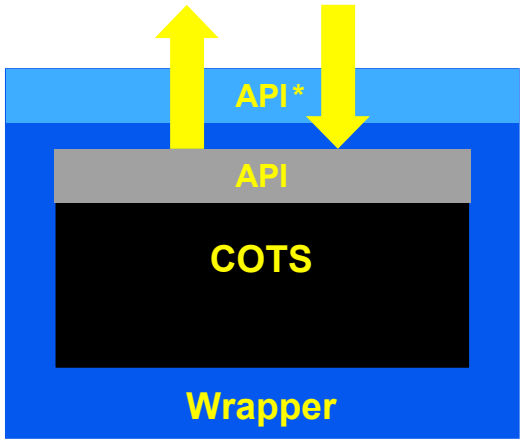
Examples

- Integrated Modular Avionics (IMA)
  - *write-protected memory allocated to processes of a partition*
  - *SAFEbus for time-partitioning access to hardware resources* [Hoyme & Driscol 1992]
- **GUARDS integrity domains** [Totel *et al.* 1998]

GUARDS Integrity Domain Structuring



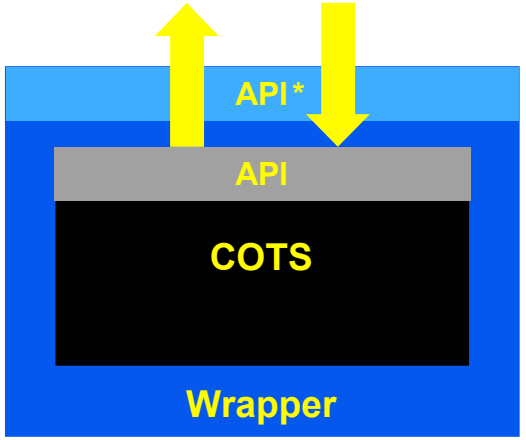
Wrapping



[Salles *et al.* 1999]

Wrapping

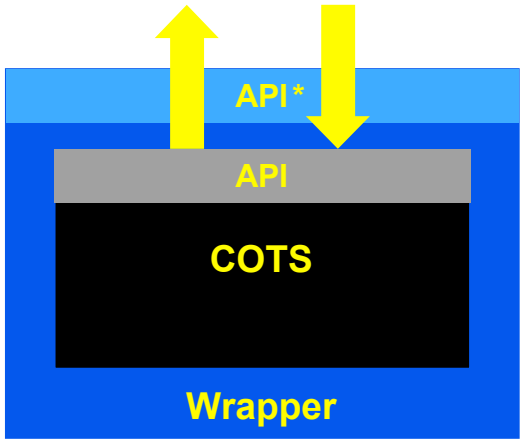
- **Objectives:**



[Salles *et al.* 1999]

Wrapping

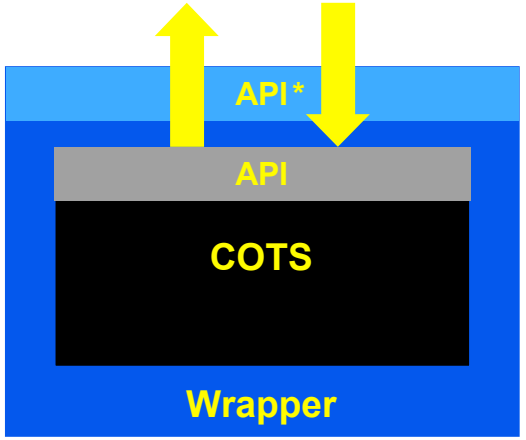
- **Objectives:**
  - **Extension of functionality (e.g., a POSIX interface to a COTS RTOS)**



[Salles *et al.* 1999]

Wrapping

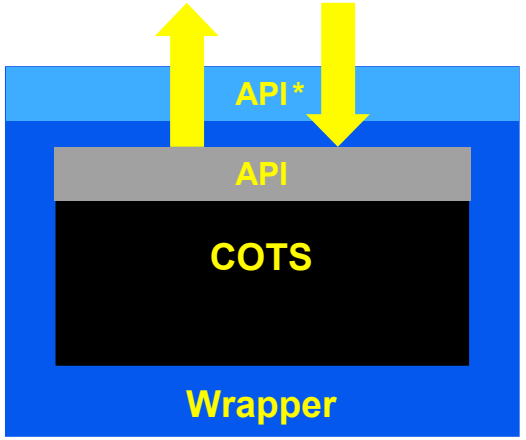
- **Objectives:**
  - **Extension of functionality (e.g., a POSIX interface to a COTS RTOS)**
  - **Restriction of functionality — limit invocation possibilities to those validated**



[Salles *et al.* 1999]

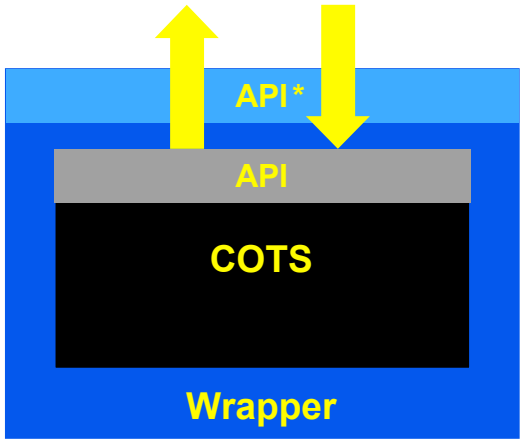
Wrapping

- **Objectives:**
  - Extension of functionality (e.g., a POSIX interface to a COTS RTOS)
  - Restriction of functionality — limit invocation possibilities to those validated
  - Filtering of input and output parameters to acceptable domains



[Salles *et al.* 1999]

Wrapping



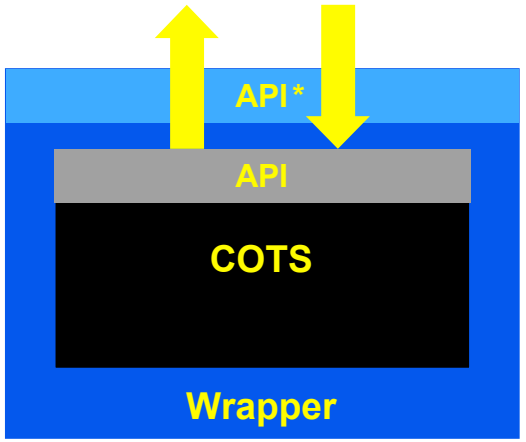
Objectives:

- Extension of functionality (e.g., a POSIX interface to a COTS RTOS)
- Restriction of functionality — limit invocation possibilities to those validated
- Filtering of input and output parameters to acceptable domains
- Implementation of executable assertions to detect, confine and possibly recover errors due to residual design faults — efficiency strongly dependent on:
  - *degree of specification formality, and knowledge thereof*
  - *observability of internal operation*

[Salles *et al.* 1999]

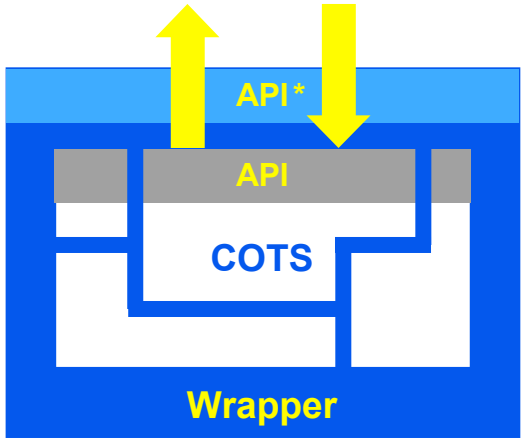
Wrapping

Objectives:



- Extension of functionality (e.g., a POSIX interface to a COTS RTOS)
- Restriction of functionality — limit invocation possibilities to those validated
- Filtering of input and output parameters to acceptable domains
- Implementation of executable assertions to detect, confine and possibly recover errors due to residual design faults — efficiency strongly dependent on:
  - *degree of specification formality, and knowledge thereof*
  - *observability of internal operation*

White-box COTS allows internal hooks to improve observability

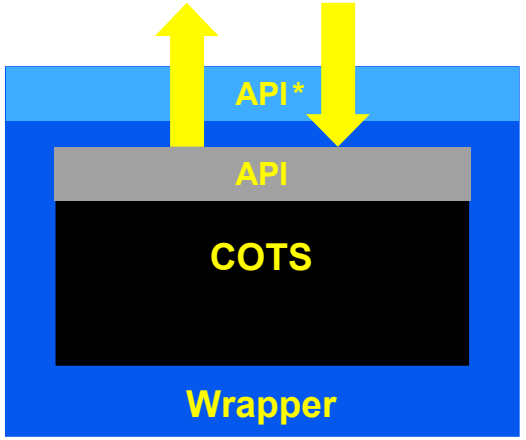


[Salles *et al.* 1999]

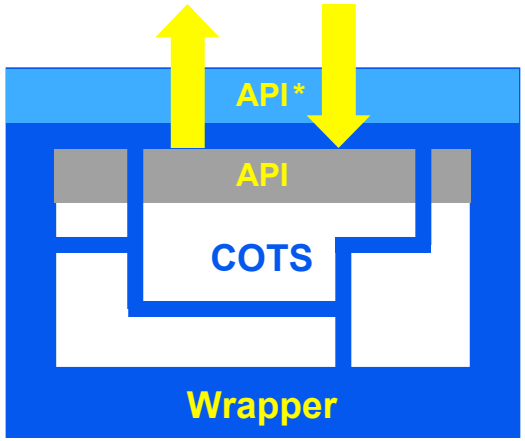
Wrapping

Objectives:

- Extension of functionality (e.g., a POSIX interface to a COTS RTOS)
- Restriction of functionality — limit invocation possibilities to those validated
- Filtering of input and output parameters to acceptable domains
- Implementation of executable assertions to detect, confine and possibly recover errors due to residual design faults — efficiency strongly dependent on:
  - *degree of specification formality, and knowledge thereof*
  - *observability of internal operation*

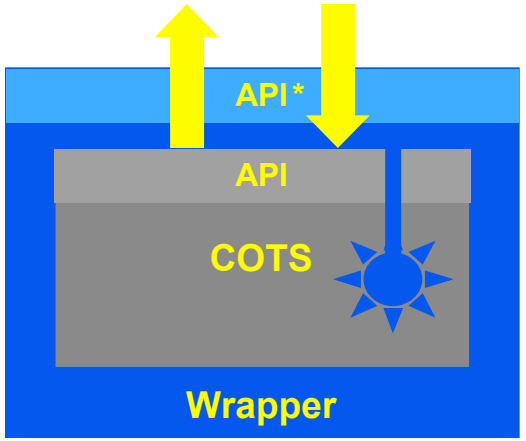


White-box COTS
allows internal hooks to improve observability



Grey-box COTS
a reflective interface supplied by COTS manufacturer allows *introspection*

[Salles *et al.* 1999]



Designing COTS-Based Critical Systems

- Functional decomposition
- Categorise functions according to their criticality, i.e., in terms of the consequences of function failure
  - ➥ e.g. [DO-178B] — catastrophic, dangerous, major, minor, no effect
- Validate components to a confidence degree commensurate with function criticality

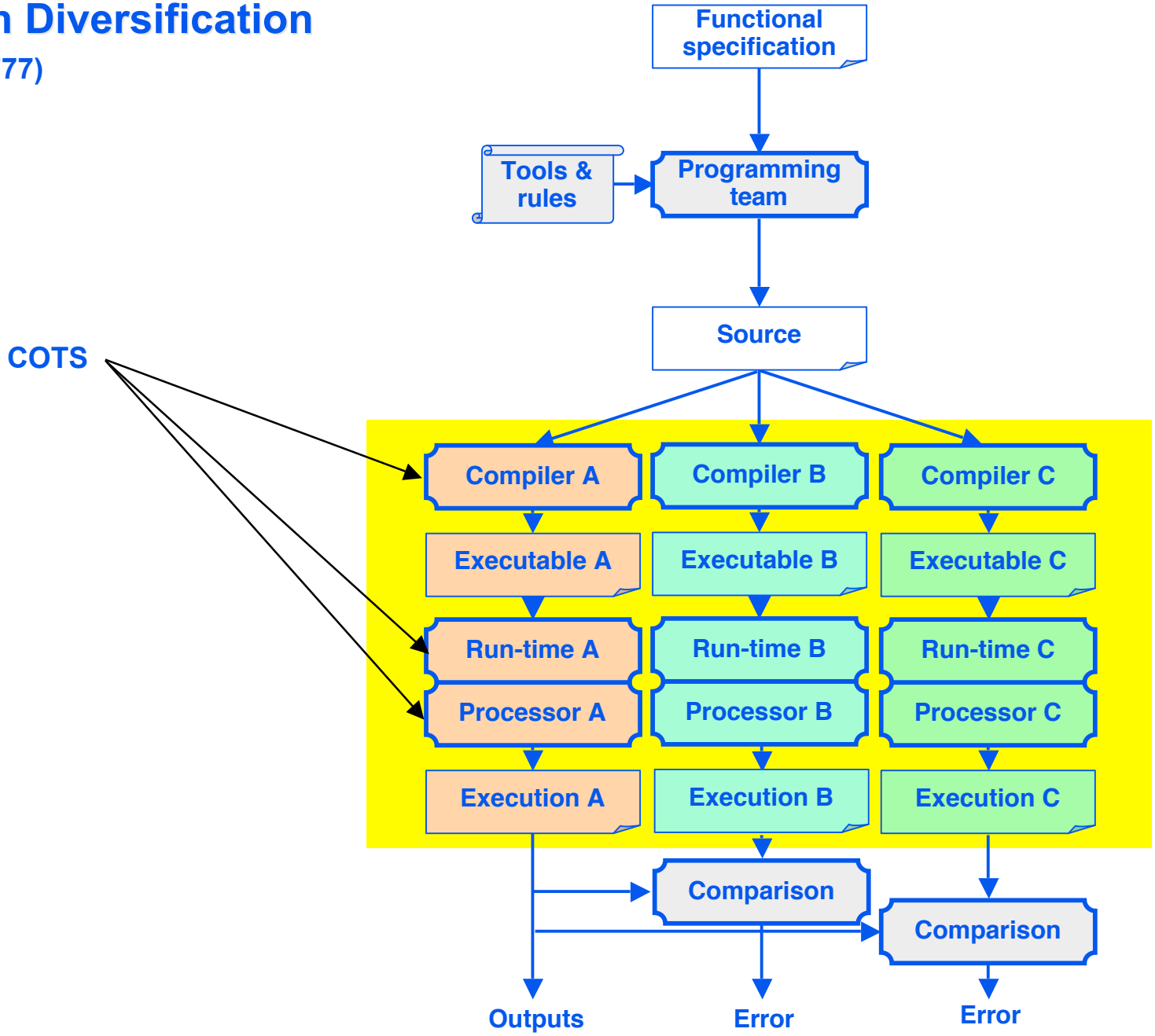
- Confidence in COTS appropriate for criticality of considered function
  - ➥ Still need to ensure that residual design faults do not affect functions of higher criticality
- Degraded service + error confinement
  - ➥ Replication & activation decorrelation
  - ➥ Functionally-independent error detection schemes (inc. diversification)
  - ➥ Partitioning
  - ➥ Wrapping

- Confidence in COTS not appropriate for criticality of considered function
  - ➥ Attempt to ensure continued operation of function despite residual design faults
  - ➥ and ensure that functions of higher criticality are not affected when continued operation is not possible
- Continued service
  - ➥ Replication & activation decorrelation
  - ➥ Diversification

Continued Service through Diversification

(Boeing 777)

[Yeh 1998]



Continued Service through Diversification

(Airbus 320)

Slats & flaps

SFCC1

SFCC2

Pilots

Spoilers

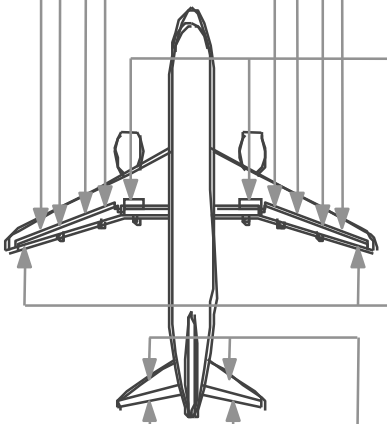
SEC1

Spoilers

SEC2

Spoilers

SEC3



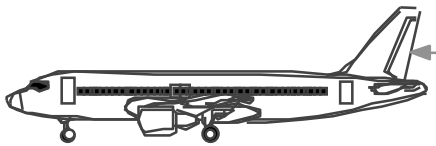
Ailerons

ELAC1

ELAC2

Trim

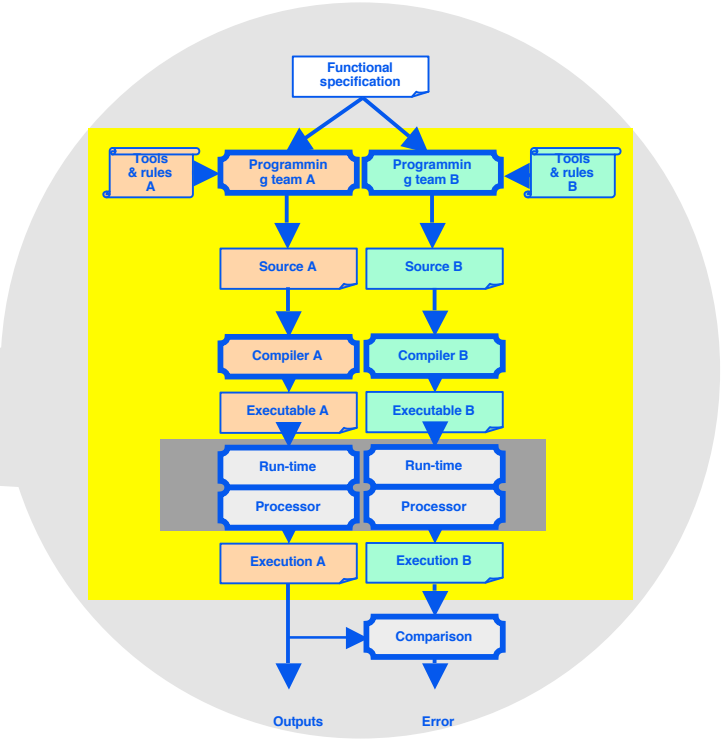
Elevator



Vertical rudder

FAC1

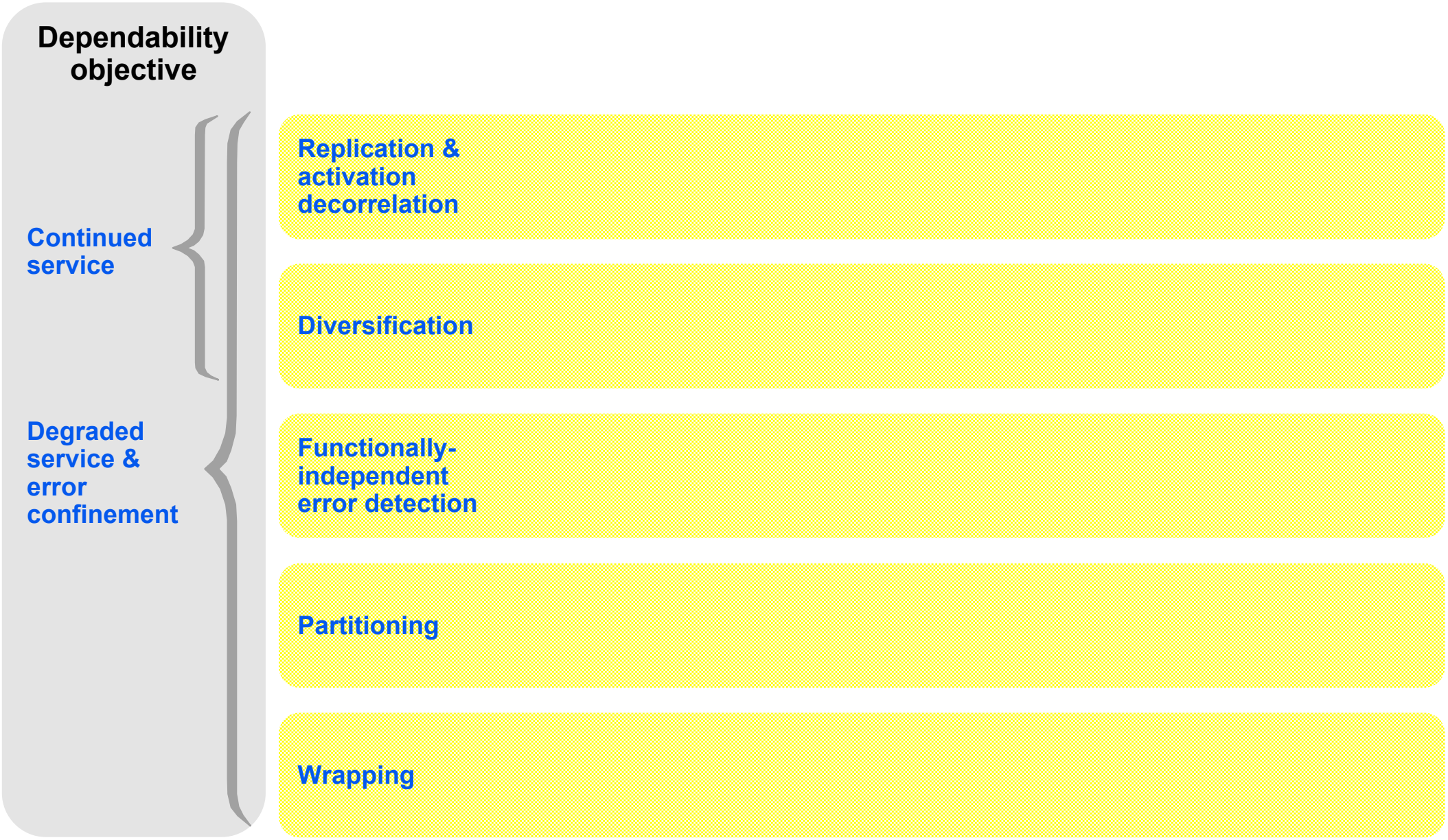
FAC2



Spoiler and Elevator Computer
SFENA/Aerospatiale
80186

ELevator and Aileron Computer
Thomson CSF
68010

Summary


Dependability objective
Continued service
Degraded service & error confinement
Replication & activation decorrelation
Diversification
Functionally-independent error detection
Partitioning
Wrapping

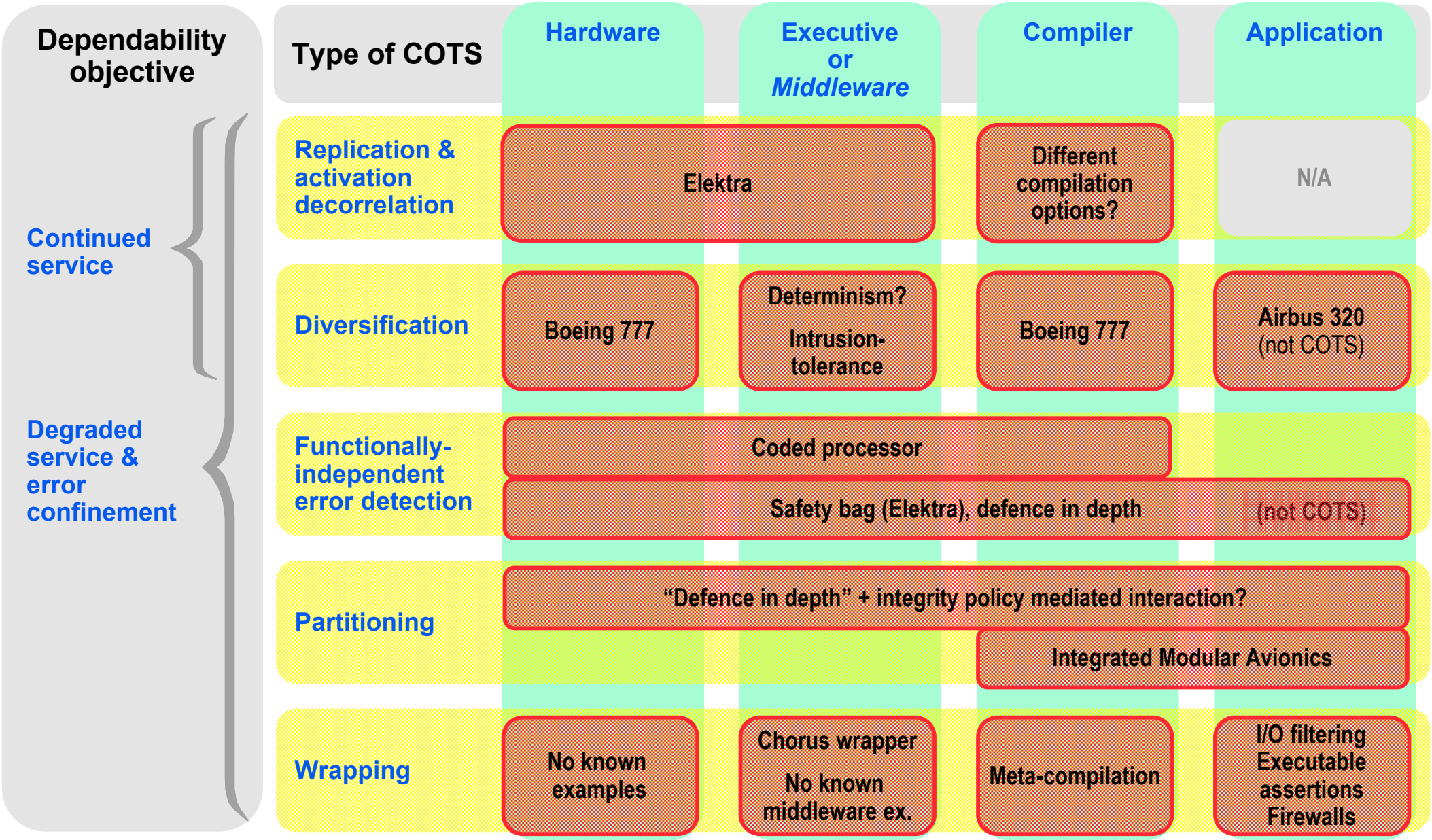
Summary

Dependability objective	Type of COTS	Hardware	Executive or *Middleware*	Compiler	Application
Continued service	Replication & activation decorrelation				
	Diversification				
Degraded service & error confinement	Functionally-independent error detection				
	Partitioning				
	Wrapping				

Summary

Dependability objective	Type of COTS	Hardware	Executive or *Middleware*	Compiler	Application
Continued service	Replication & activation decorrelation				N/A
	Diversification				
Degraded service & error confinement	Functionally-independent error detection				
	Partitioning				
	Wrapping				

Summary


Dependability objective
Type of COTS
Hardware
Executive or Middleware
Compiler
Application
Continued service
Degraded service & error confinement
Replication & activation decorrelation
Elektra
Different compilation options?
N/A
Diversification
Boeing 777
Determinism?
Intrusion-tolerance
Boeing 777
Airbus 320 (not COTS)
Functionally-independent error detection
Coded processor
Safety bag (Elektra), defence in depth
(not COTS)
Partitioning
"Defence in depth" + integrity policy mediated interaction?
Integrated Modular Avionics
Wrapping
No known examples
Chorus wrapper
No known middleware ex.
Meta-compilation
I/O filtering
Executable assertions
Firewalls

References

[Arlat et al. 2000] J. Arlat, J.-P. Blanquart, T. Boyer, Y. Crouzet, M.-H. Durand, J.-C. Fabre, M. Founau, M. Kaâniche, K. Kanoun, C. Mazet, P. Le Meure, D. Powell, F. Scheerens, P. Thévenod and H. Waeselynck, Conception et qualification de systèmes critiques intégrant des COTS, LAAS-CNRS, Rapport, N°00112, février 2000.

[Brière & Traverse 1993] D. Brière and P. Traverse, "AIRBUS A320/A330/A340 Electrical Flight Controls - A Family of Fault-Tolerant Systems", in 23rd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-23), (Toulouse, France), pp.616-23, IEEE Computer Society Press, 1993.

[D0-178B] Software Considerations in Airborne Systems and Equipment Certification, Advisory Circular D0-178B, RTCA, Inc., Washington D.C.

[Erb 1989] A. Erb, "Safety Measures of the Electronic Interlocking System 'Elektra'", in 8th IFAC Workshop on the Safety of Computer Control Systems (SAFECOMP 89), (Vienna, Austria), pp.49-52, 1989.

[Forin 1989] P. Forin, "Vital Coded Microprocessor Principles and Application for Various Transit Systems", in *IFAC Conf. on Control, Computers, Communications in Transportation (CCCT'89)*, (Paris, France), pp.137-142, AFCET, 1989.

[Gray 1986] J. Gray, "Why do Computers Stop and What can be done about it?", in *5th Symp. on Reliability in Distributed Software and Database Systems*, (Los Angeles, CA, USA), pp.3-12, IEEE Computer Society Press, 1986.

[Hoyme & Driscol 1992] K. Hoyme and K. Driscoll, "SAFEbus®", in *11th Digital Avionics Systems Conference*, (Seattle, WA, USA), pp.68-73, AIAA/IEEE, October 1992.

[Kantz & Koza 1995] H. Kantz and C. Koza, "The ELEKTRA Railway Signalling System: Field Experience with an Actively Replicated System with Diversity", in *25th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-25)*, (Pasadena, CA, USA), pp.453-458, IEEE Computer Society Press, June 1995.

[Profeta et al. 1996] J. A. Profeta, N. K. Andrianos, B. Yu, B. W. Johnson, T. A. DeLong, D. Guaspari and D. Jamsek, "Safety-Critical Systems Built with COTS", *IEEE Computer*, 29 (11), pp.54-60, November 1996.

[Salles *et al.* 1999] F. Salles, M. Rodriguez-Moreno, J.-C. Fabre and J. Arlat, "Metakernels and Fault Containment Wrappers", in *29th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-29),* (Madison, WI, USA), pp.22-9, IEEE CS Press, 1999.

[Totel et al. 1998] E. Totel, J.-P. Blanquart, Y. Deswarte and D. Powell, "Supporting Multiple Levels of Criticality", in *28th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.70-79, IEEE Computer Society Press, 23-25 June 1998.

[Yeh 1998] Y. C. B. Yeh, "Dependability of the 777 Primary Flight Control System", in *Dependable Computing for Critical Applications (DCCA-5),* (R. K. Iyer, M. Morganti, W. K. Fuchs and V. Gligor, Eds.), Dependable Computing and Fault-Tolerant Systems, 10, pp.3-17, IEEE Computer Society Press, 1998.