

Club SEE « Systèmes Informatiques de Confiance »
Cercle « Conception et validation pour la sûreté de fonctionnement »

Réunion du 9 mars 1999

Thème : « Conception et validation des mécanismes de tolérance aux fautes »

Validation et encapsulation de micronoyaux du commerce

Jean Arlat, Jean-Charles Fabre, Manuel Rodríguez et Frédéric Salles

LAAS-CNRS et *LIS*

7, Avenue du Colonel Roche

31077 Toulouse cedex 4

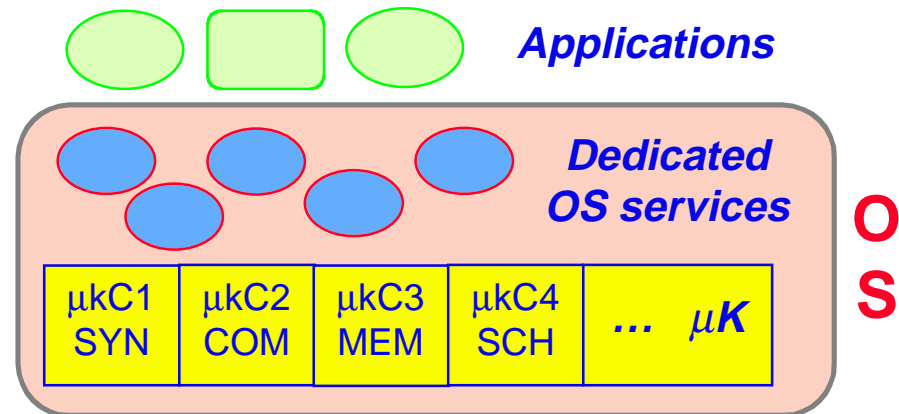
COTS Microkernels

■ Microkernels

- ◆ Set of basic OS functions
 - synchronization
 - task management
 - memory management
 - scheduling
 - communication
 - IT management
- ◆ Properties
 - componentized
 - Configurability

■ Microkernel-based OS

- ◆ Set of services on the μk
- ◆ A given instance of the μk



■ Motivations

- ◆ Cost reduction
- ◆ Flexibility



Development of

- general-purpose
- or dedicated (embedded)

operating systems

Application Fields

■ Telecoms

- ◆ Mobile phones,
- ◆ Switching systems

■ Critical systems

- ◆ Railway signaling
- ◆ Avionics
- ◆ Space

■ Fault-tolerant systems

- ◆ Middleware handling replication
- ◆ Cold-warm restart

■ Supercomputing

- ◆ Parallel systems
- ◆ Multiprocessing architectures

■ General-purpose systems

- ◆ Unix-compliant
- ◆ Others

■ Others

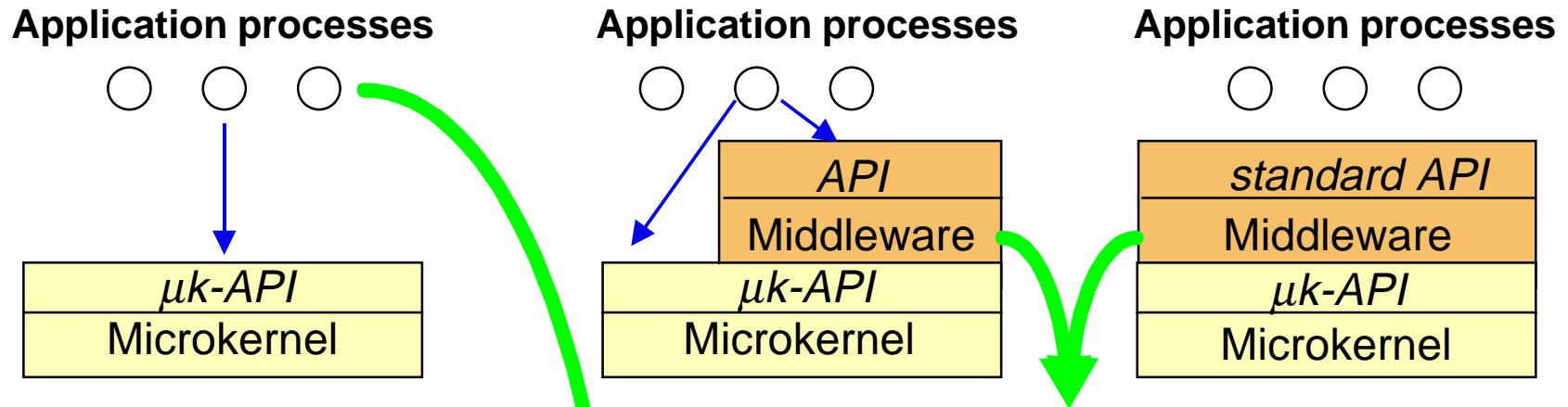
- ◆ “Karaoke-on-Demand” machines
- ◆ ...



Examples of microkernels

- ◆ Chorus, VxWorks, Qnx, Psos, VRTX, etc.

Microkernel-based Systems

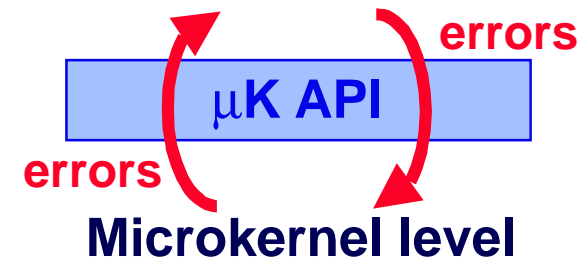


■ Collection of services

- ◆ Specific/standard API
- ◆ Dependability services

■ Assessment of μK behavior in the presence of faults

Application/service level



Dependability Concerns

■ Impairments to dependability:

◆ SW & HW fault \rightsquigarrow **error** \rightsquigarrow **failure**

■ Main concerns with COTS microkernels

◆ COTS development cycle \neq standards for developing applications

◆ Short lifetime and customization

✦ very little statistics from field experience to be expected

◆ Some crucial issues

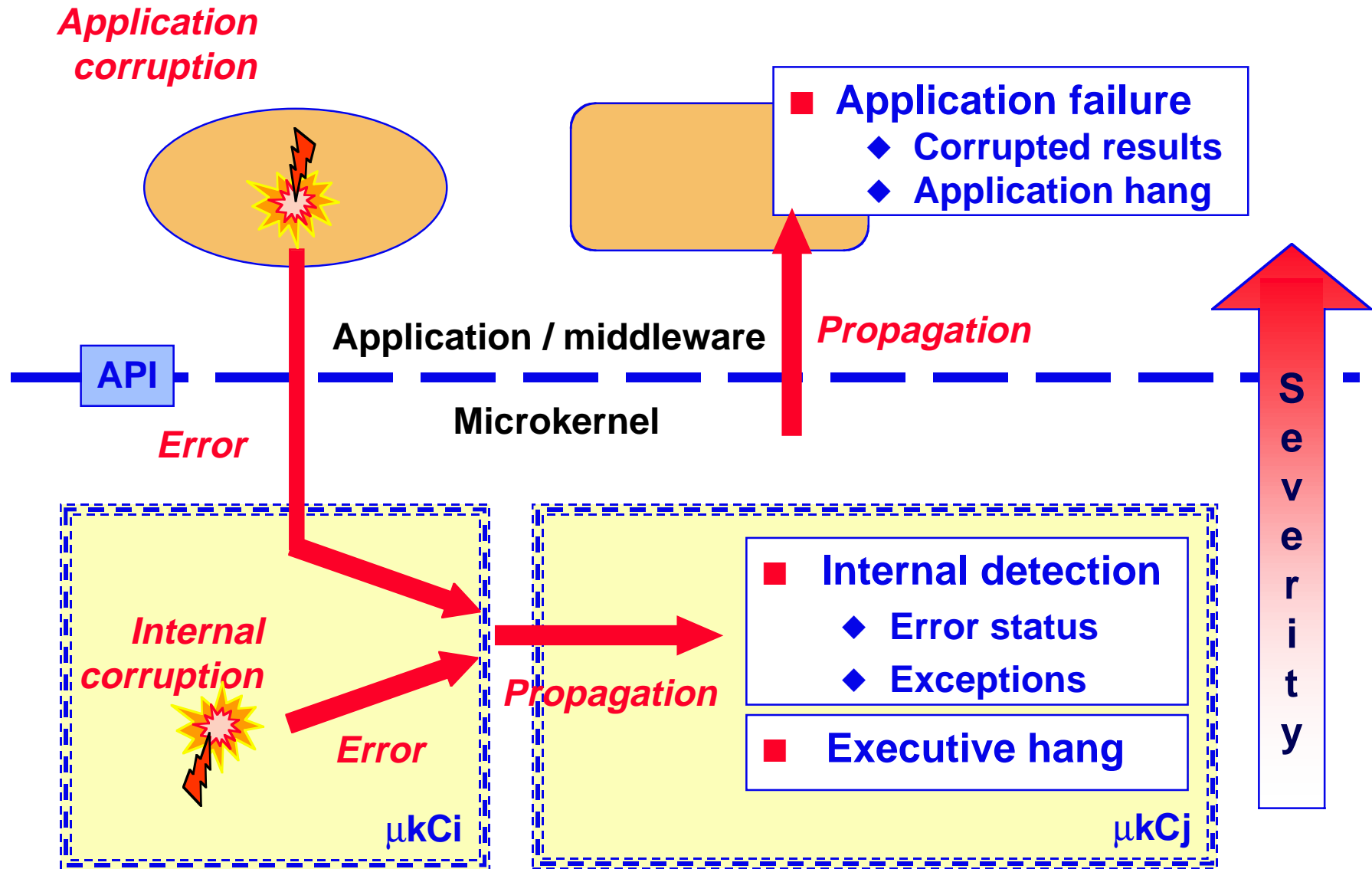
✦ robustness of the microkernel interface?

✦ coverage of error detection mechanisms?

✦ failure modes and error propagation channels?

✦ impact of faulty behavior on application/middleware layer?

Behavior in the Presence of Faults



Assessment by Fault Injection:

MAFALDA



■ Evaluation

- Error detection mech.
- Interface robustness

■ Injection target

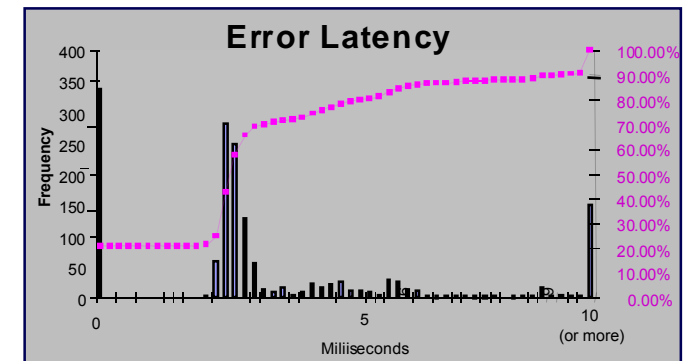
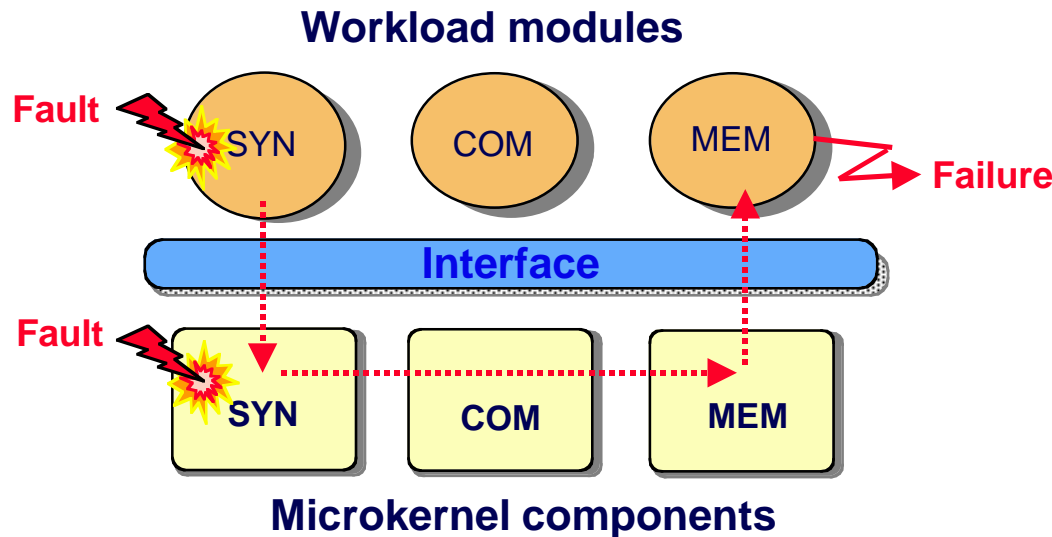
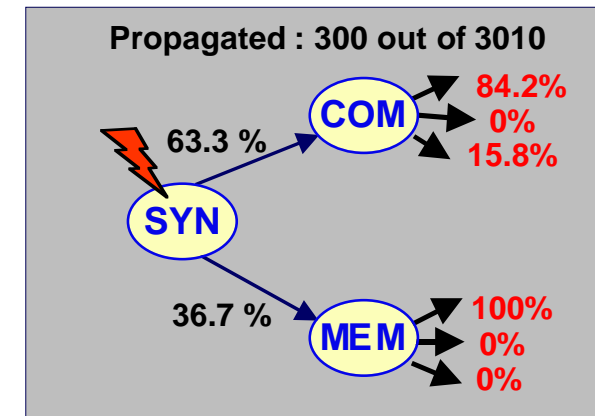
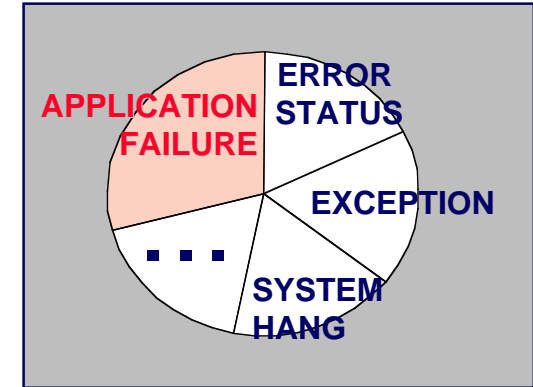
- μ kernel component
- Workload module

■ Fault types

- Microkernel corruption
- Input parameter corr.

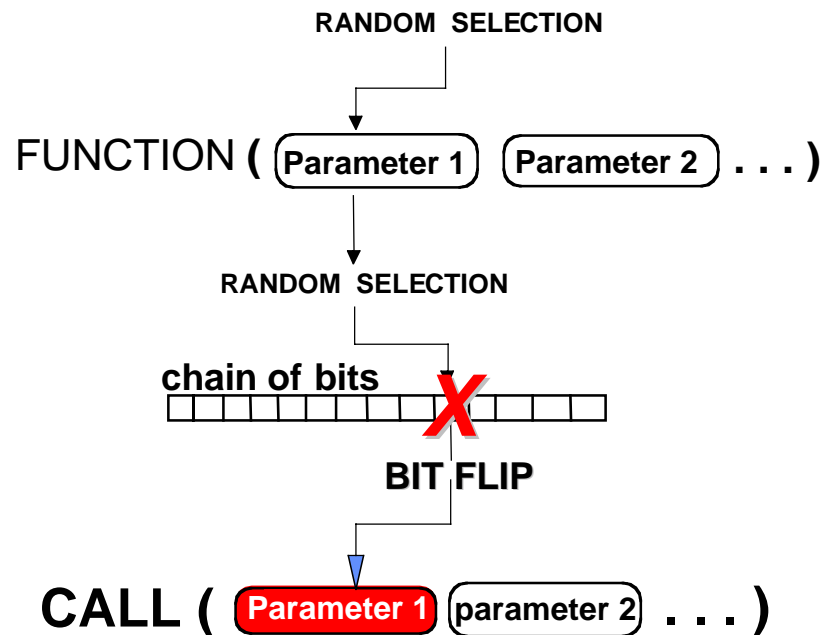
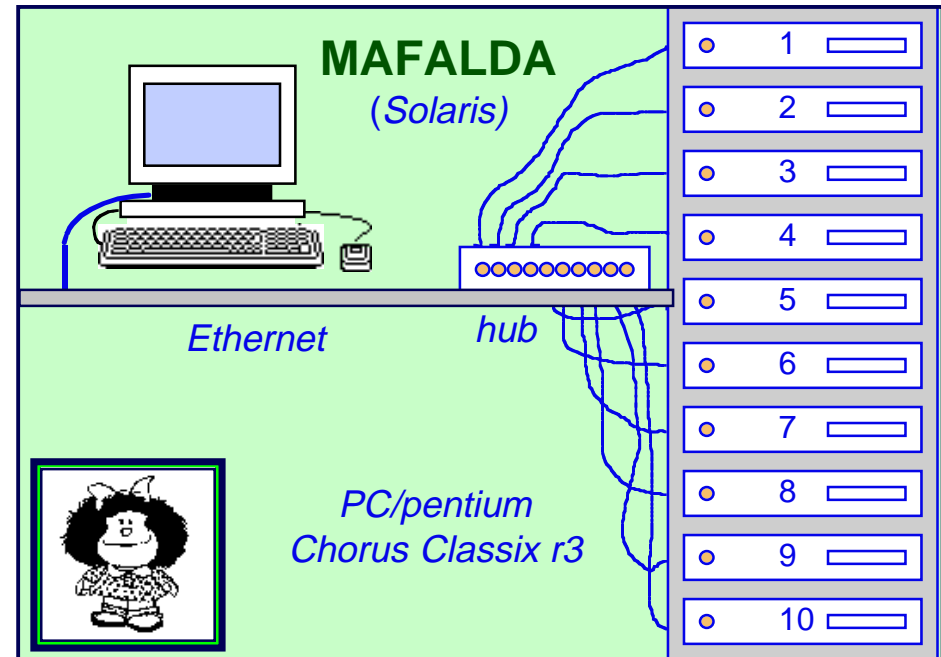
■ Observation

- Single module behavior
- Inter-module err. Propag
- Error latency

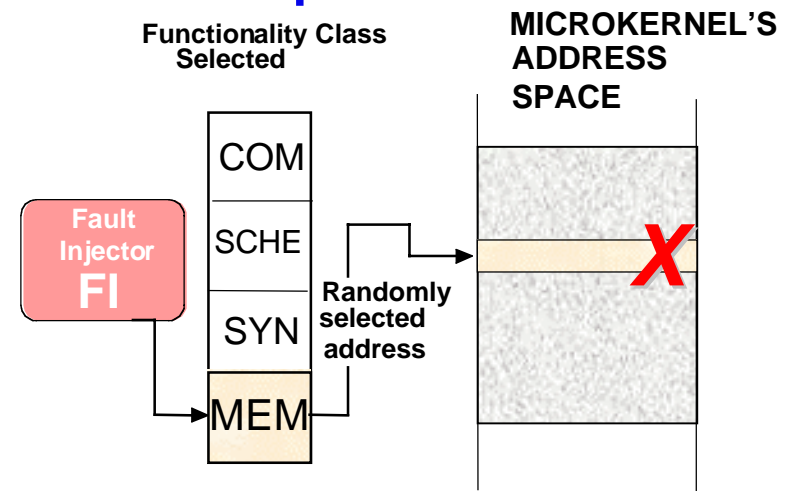


MAFALDA: Architecture and Fault Models

- Corruption of kernel service calls parameters



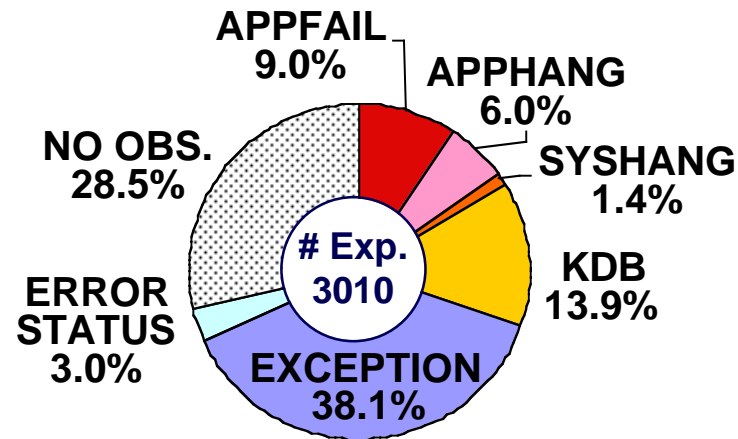
- Fault injection into the μ K address space



Example: Semaphore Module of Chorus ClassiX r3

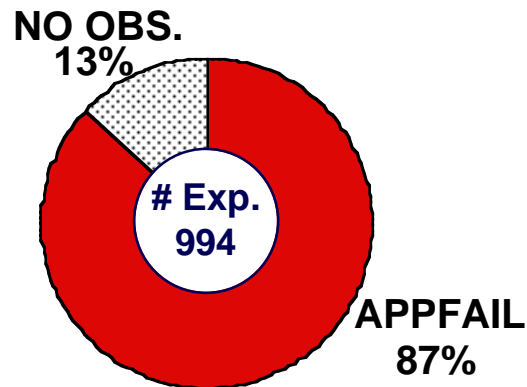
■ Microkernel FI

- Code segment

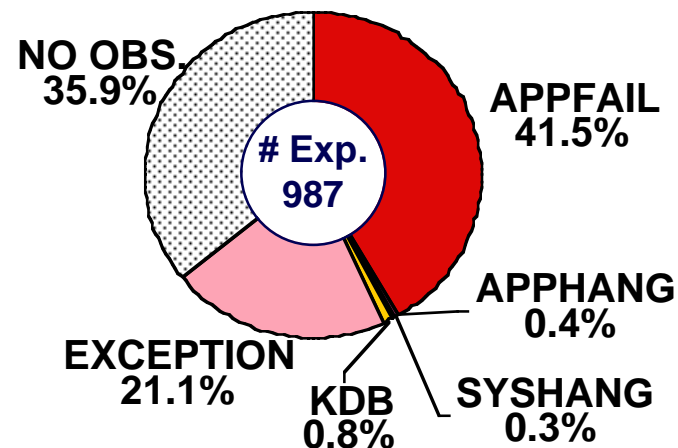


■ Parameter FI

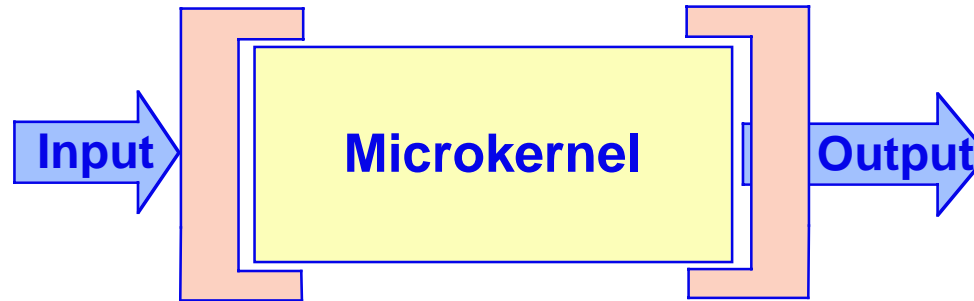
- user space



- supervisor space



Fault Containment Wrappers



■ Principle of wrapping:

- ◆ Filtering microkernel system calls only wrt external information
- ◆ Analysis of service to system calls according to expected behavior

■ Fault containment wrappers

- ◆ Modeling the behavior of the microkernel component in the absence of faults \Rightarrow notion of **runtime predicates**
- ◆ **Observability** of some internal features to implement runtime predicates
- ◆ **Controllability** to act on the microkernel when necessary

A Simple Example: Synchronization by Semaphores

Semaphores:

- integers **s.val**
- process queue **s.queue**

Main operations:

- Init: **I(s)**
- Get: **P(s)**
- Release: **V(s)**

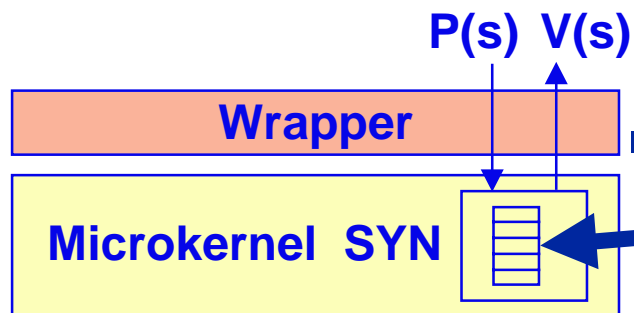
Initial value: **init_value**

Let **#P(s)** and **#V(s)** be the total number of calls to **P(s)** and **V(s)**

$$s.val = \text{init_value} - \#P(s) + \#V(s)$$

Let **#Suspended(s)** be the number of threads in s.queue

$$\#Suspended(s) = \max(0, -s.val)$$



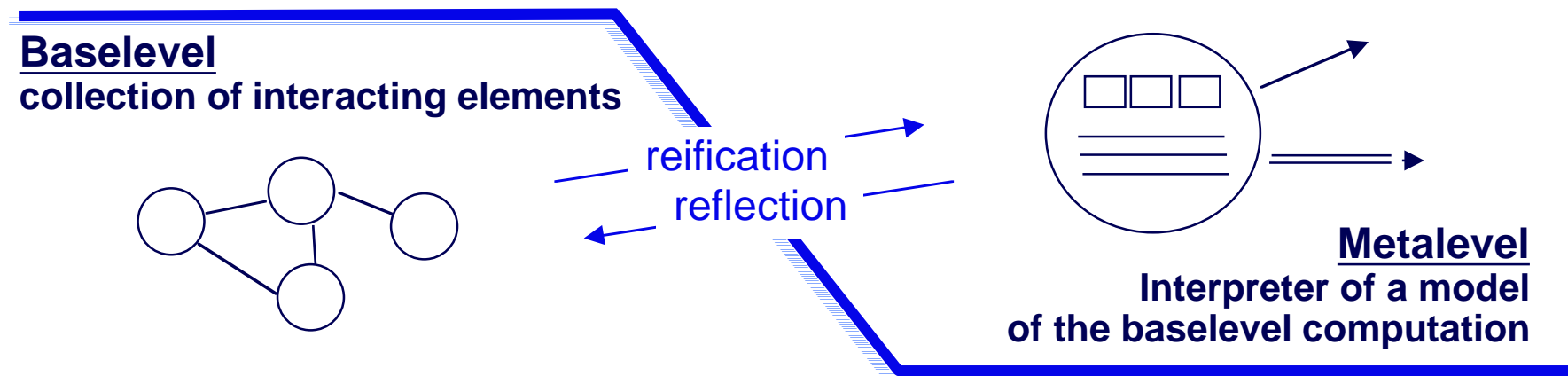
Accordingly, the predicate can be defined as:

$$[s.val = \text{init_value} - \#P(s) + \#V(s)] \wedge$$

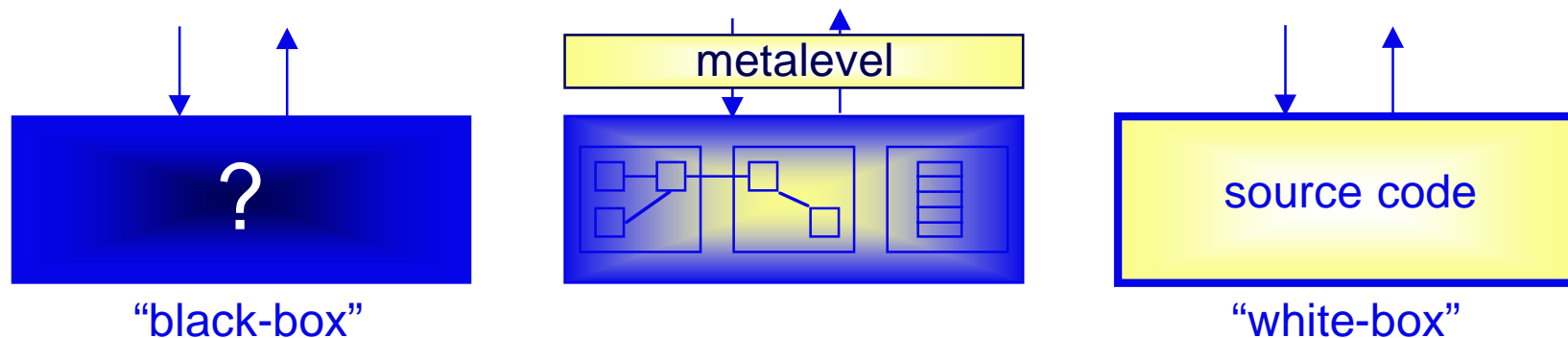
$$[\#Suspended(s) = \max(0, -s.val)]$$

Notion of Reflective Component

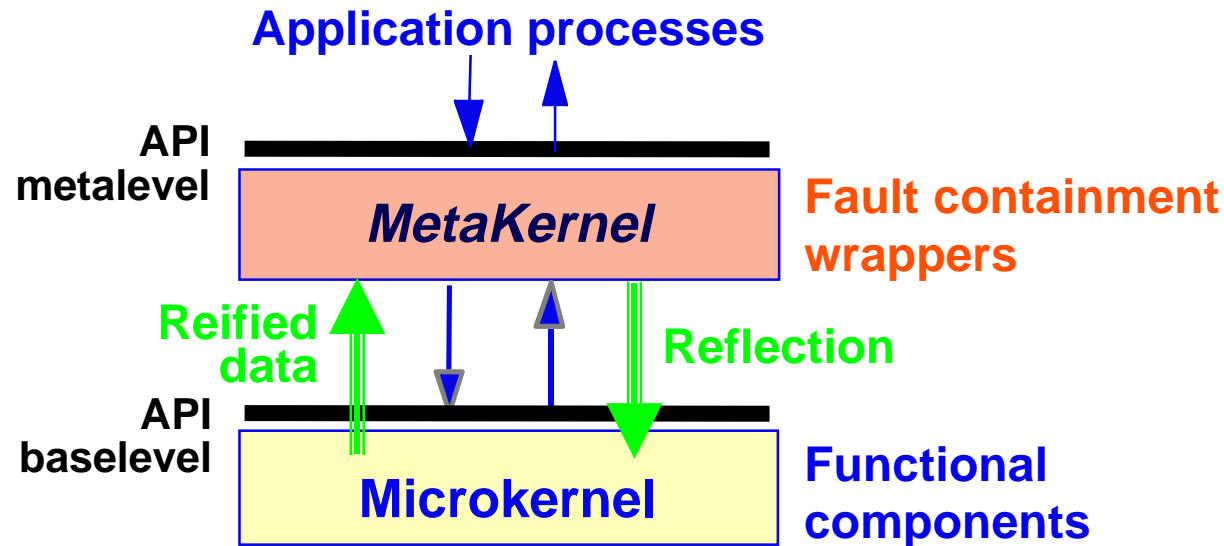
- **Reflection:** process by which a system can reason and act upon itself
 - ◆ provides means to access and update internal aspects
 - *structure and behavior* – of a system / language / component



Notion of reflective microkernel



MetaKernel-based Wrappers



■ Fault containment wrappers

- ◆ Set of predicates (executable assertions) for the internal components

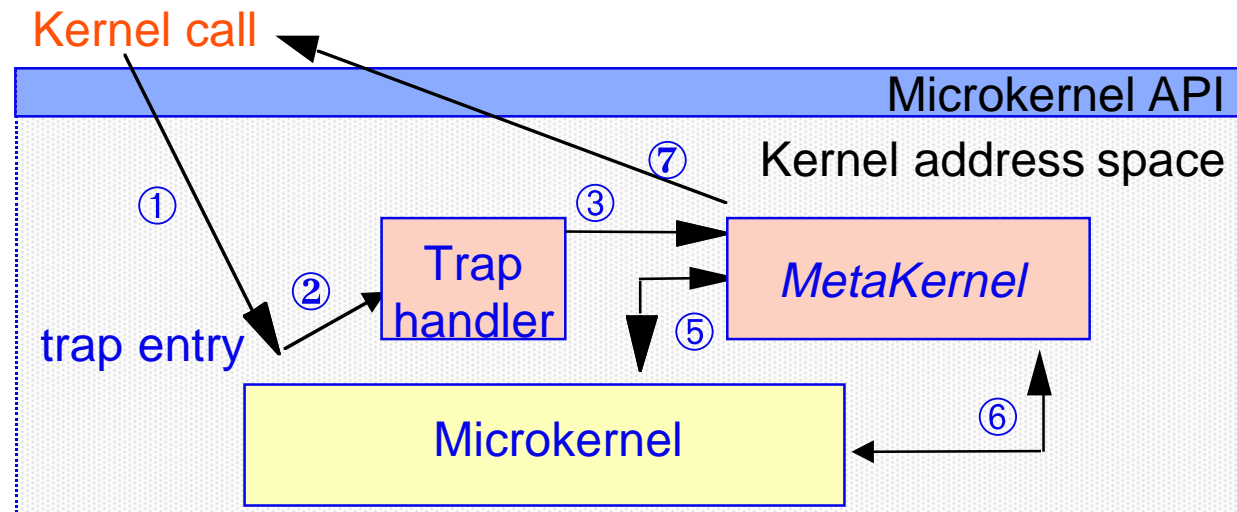
■ MetaKernel key issues

- ◆ Interception of the microkernel service calls
- ◆ Definition of the internal information needed for the implementation of the predicates (microkernel introspection - **reified data**)
- ◆ Definition of possible actions on the microkernel (e.g. corrective action or crash - **reflection**)

Implementation Issues

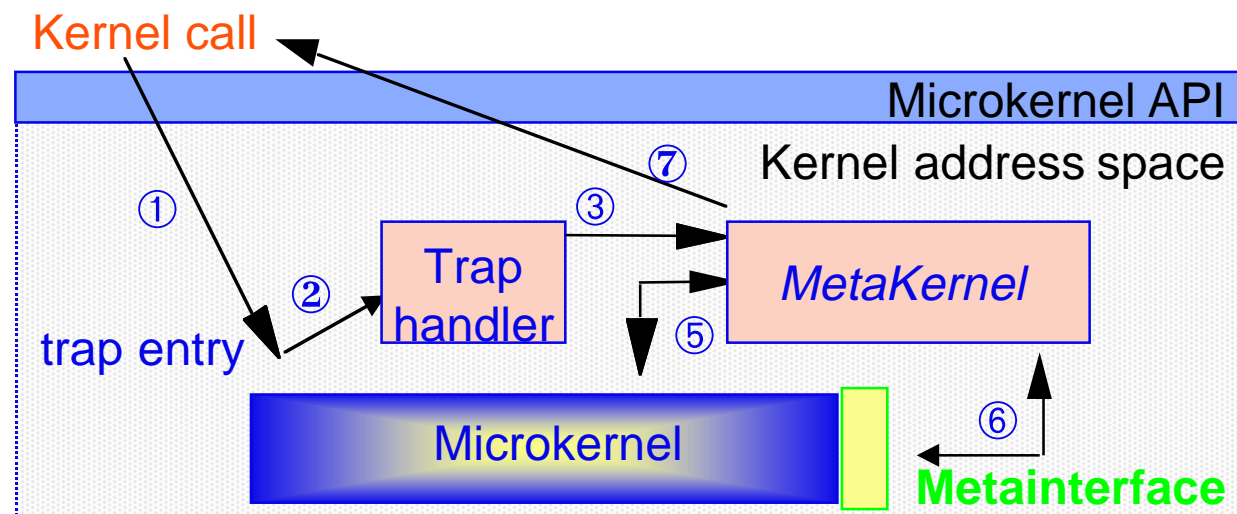
■ Direct access

↓
source code
available



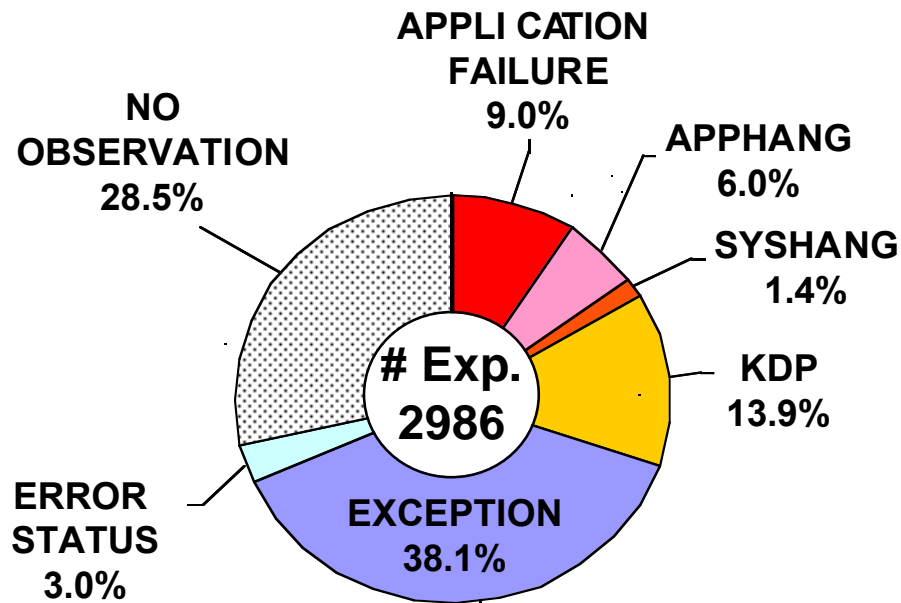
■ Metainterface

↓
implemented
by the μ K
provider

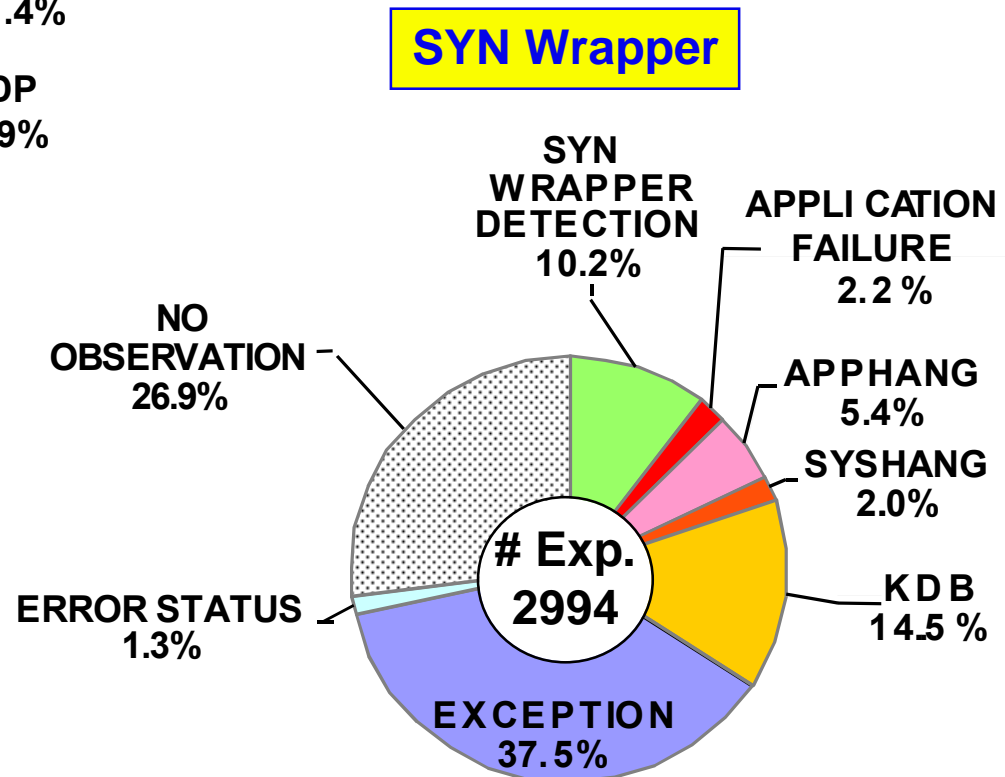


Impact of the SYN Wrapper

Injection of transient faults in the code of the Synchronization μ K component



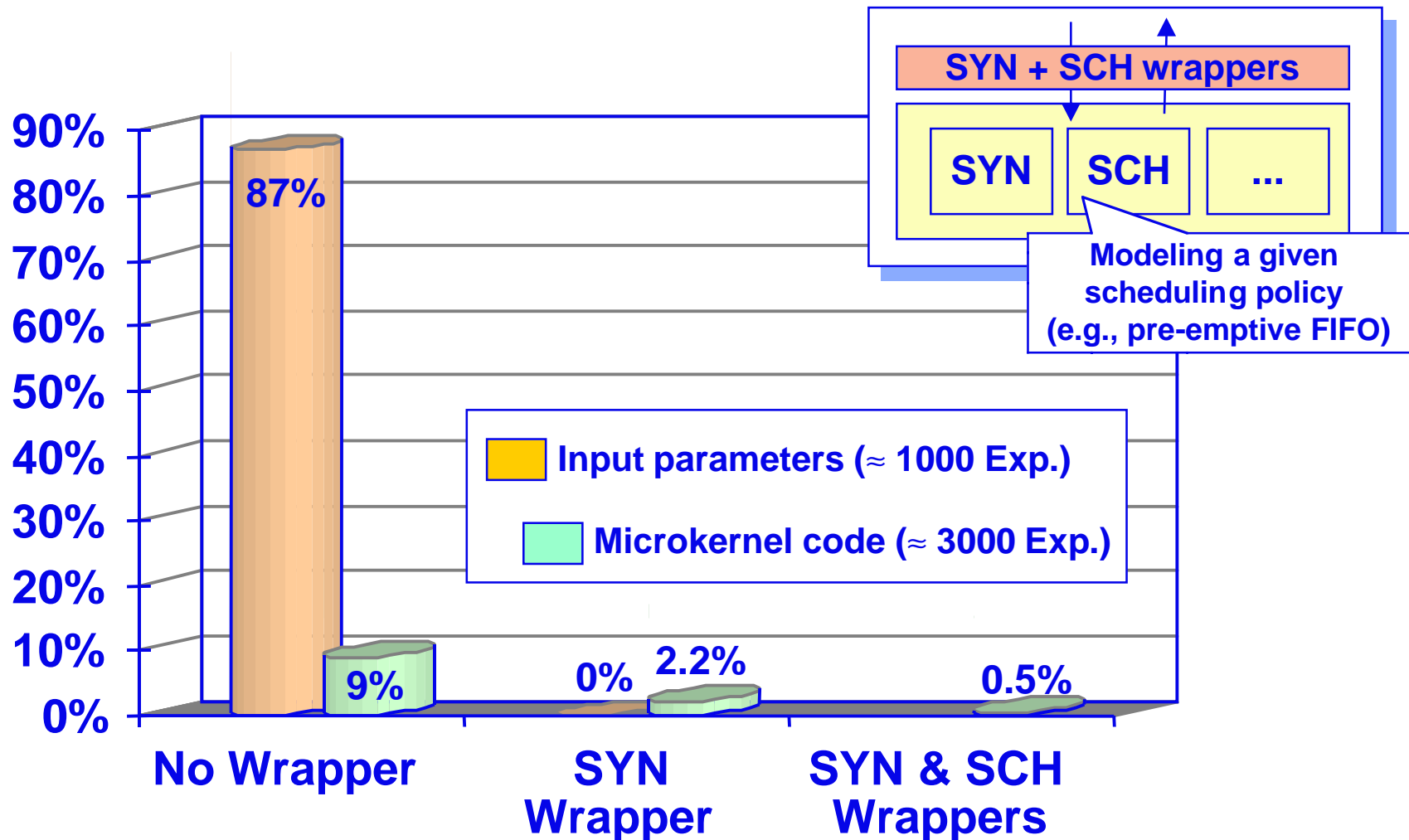
No Wrapper



SYN Wrapper

Combined Impact of the Wrappers

Fault Injection Experiments Focusing on the Synchronization Component and Module



Complementary Viewpoints

■ *System integrator viewpoint*


- ◆ Analysis of a given customized instance of a COTS microkernel
- ◆ Definition of wrappers
 - ✦ finding the appropriate abstraction level for modeling
 - ✦ alternatively, definition of acceptance expressions
 - ✦ identification of reified data
 - ✦ definition of corrective actions or actions for improving the fail-silent behavior of the microkernel candidate
- ◆ Definition of application-dependent or generic fault tolerance strategies as upper software layers

■ *Microkernel provider viewpoint*

- ◆ Analysis of the traces provided by MAFALDA (APPFAIL, KDB, SYSHANG)
- ◆ Provision of fault containment wrappers according to well established predicates \Rightarrow implementation of the metainterface

Conclusion and Future Work

■ Dependability requirements for COTS microkernel-based systems



Assessment by Fault Injection

- ✦ identification of the failure modes
- ✦ quantitative measurements

Design Aid

- ✦ coverage of underlying assumptions regarding the implementation of upper layers fault tolerance strategies
- ✦ analysis of a given customization of a COTS microkernel
- ✦ framework for the implementation of fault containment wrappers

■ Current work and prospective

- ◆ Development of wrappers for scheduling including real-time aspects
- ◆ Comparison of several microkernel candidates
- ◆ Improvement of the tool