Automated Application of Fault Tolerance Mechanisms in a Component-Based System

Isabella Stilkerich

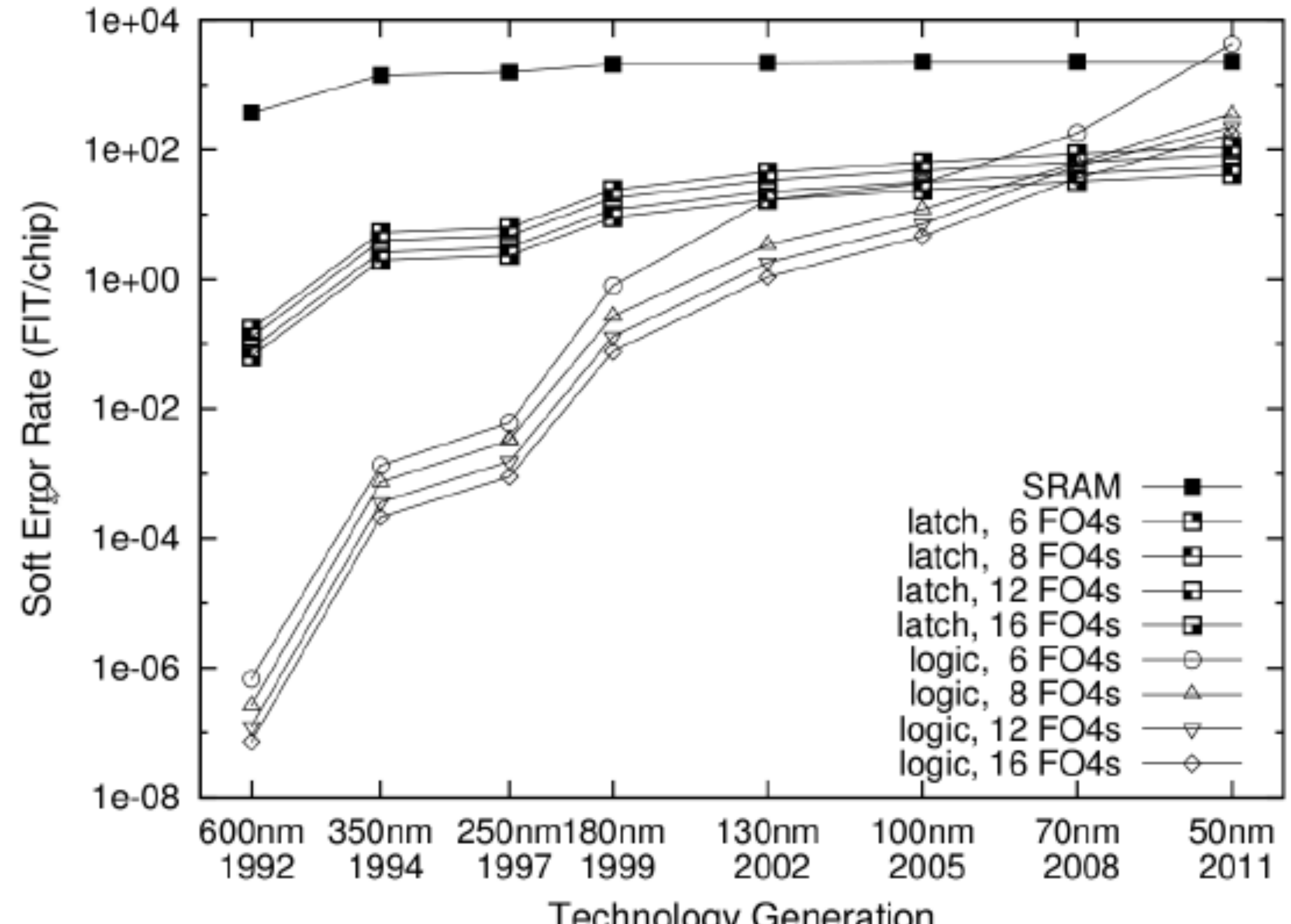
Friedrich-Alexander-Universität
Erlangen-Nürnberg

Department of Computer Science 4
Distributed Systems and Operating Systems
Friedrich-Alexander University Erlangen-Nuremberg

http://www4.cs.fau.de/~isa
isabella.stilkerich@cs.fau.de

Motivation

- Transient hardware faults become more likely
  - soft error rate in logic has increased by 9 orders of magnitude
  - soft error rate in SRAM is constantly high
  - soft errors cannot be ignored anymore

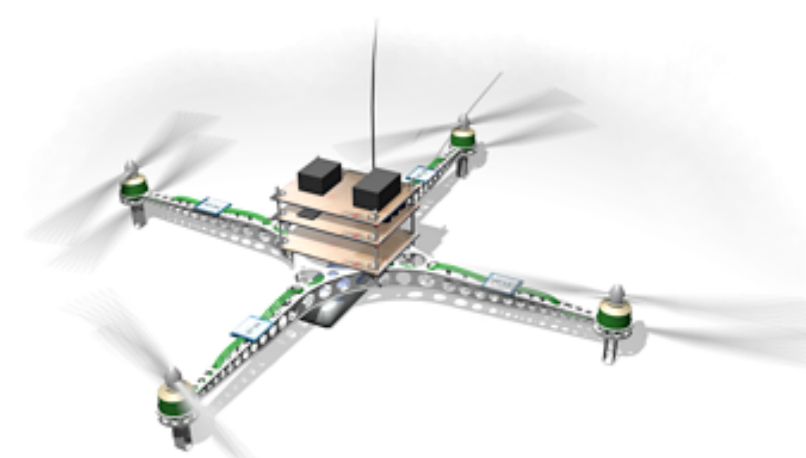


Motivation

- Transient hardware faults become more likely
  - soft error rate in logic has increased by 9 orders of magnitude
  - soft error rate in SRAM is constantly high
  - soft errors cannot be ignored anymore
- Hardware-based fault tolerance (FT) techniques
  - expensive: size, weight and power

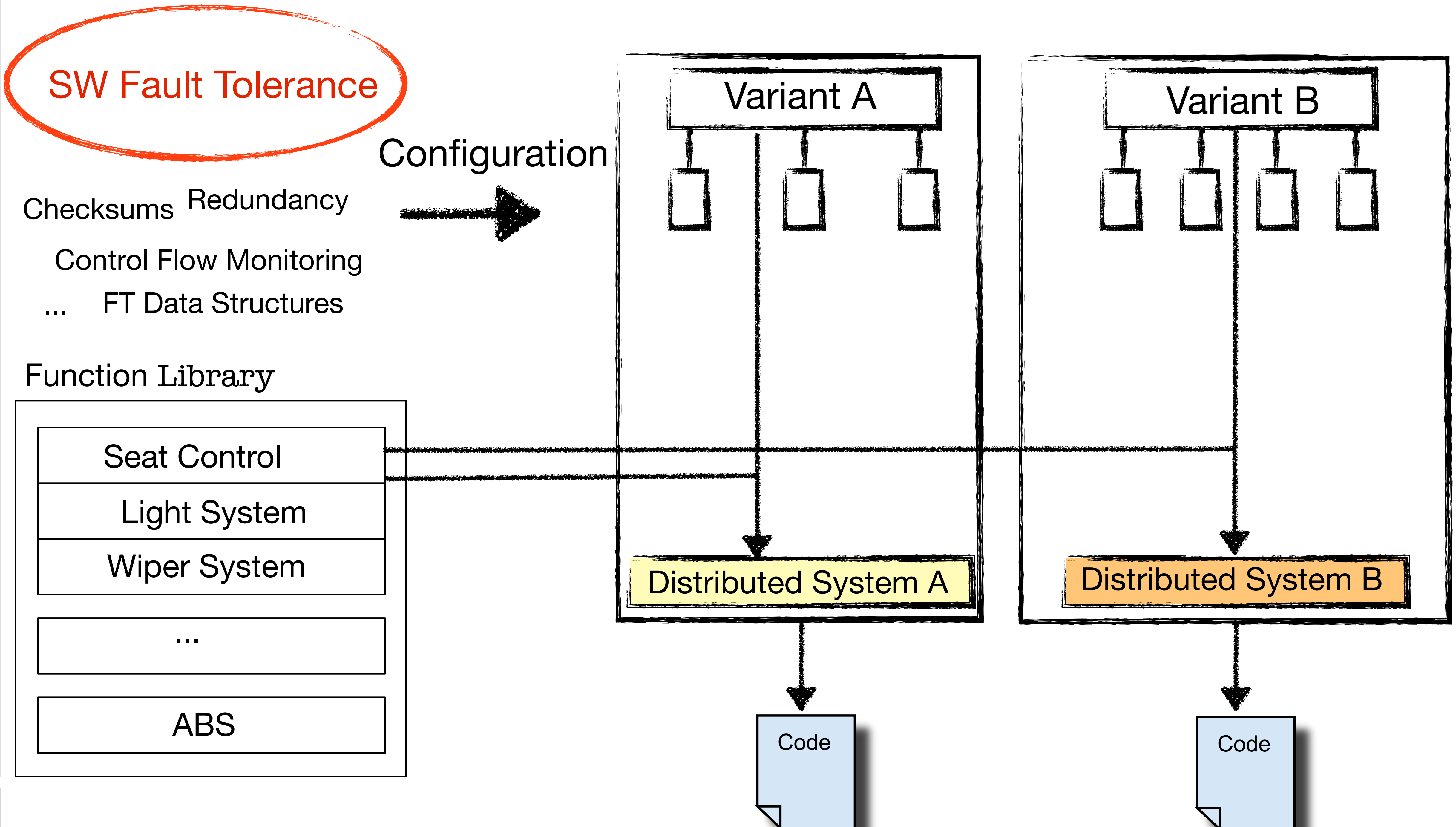
Motivation

- Transient hardware faults become more likely
  - soft error rate in logic has increased by 9 orders of magnitude
  - soft error rate in SRAM is constantly high
  - soft errors cannot be ignored anymore
- Hardware-based fault tolerance (FT) techniques
  - expensive: size, weight and power
- Software-based fault tolerance (FT) techniques



Fault Tolerance - A Non-functional Property

- Techniques differ in protection properties and cost
- Measures are application specific

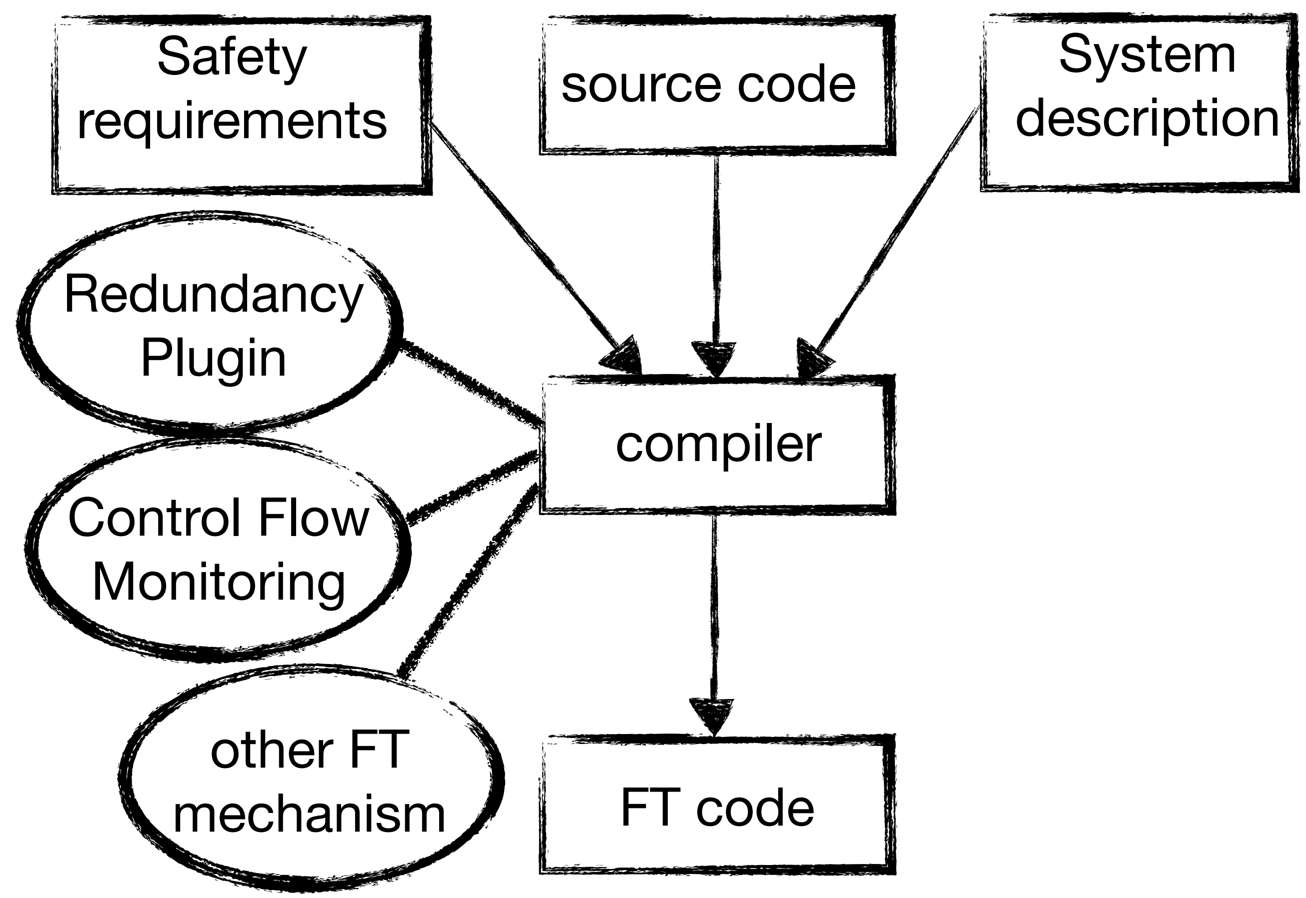


Outline

- Motivation
- General Approach
- Prototype: Module Redundancy in KESO
- Conclusion and Future Work

Configurable Fault Tolerance

- Compiler-based approach
  - Separation functional code and fault tolerance
  - Plugins provide implementations of different FT techniques
  - Techniques can be combined
  - FT tailored to app, HW and safety requirements
- Automated FT application
  - Static system
  - Use of type-safe language



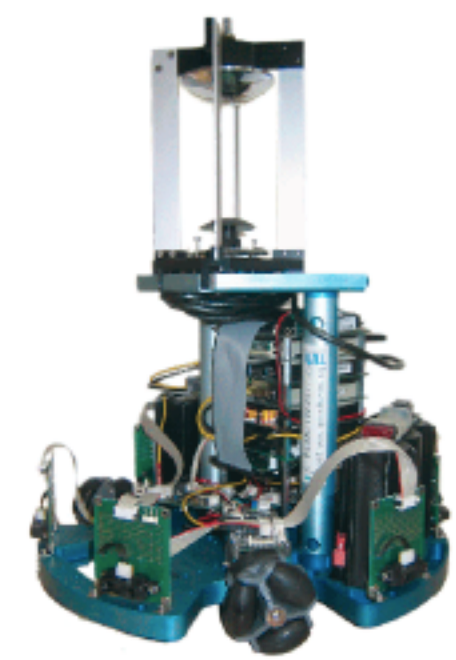
Static Embedded Systems and Java

- comprehensive a priori knowledge
  - code
  - system objects (tasks/threads, locks, events)
  - system object relationships (e.g., which tasks access which locks)

- benefits of Java
  - more robust software (cf., MISRA C)
  - software-based spatial isolation
- problems of Java
  - dynamic code loading
    - fully-featured Java runtimes (e.g., J2ME configurations)
  - overhead
    - code is interpreted or JIT compiled (execution time)
    - dynamic linking (footprint)

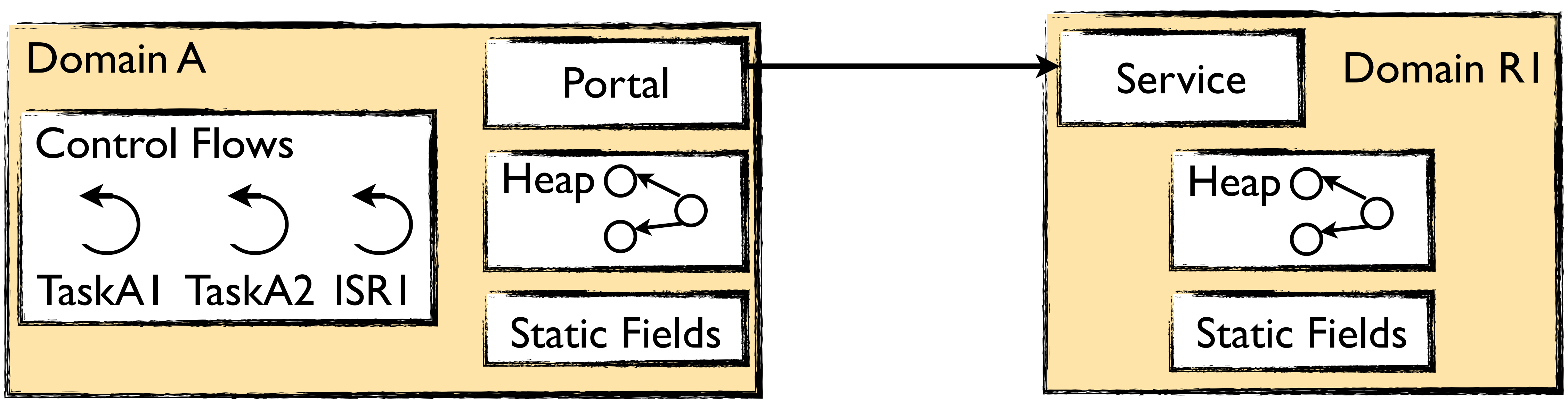
KESO: Overview

- JVM tailoring (instead of fixed configurations)
  - static applications, no dynamic class loading
  - no Java reflection
  - ahead-of-time compilation to Ansi C, VM bundled with application
- scheduling/synchronization provided by underlying OS
  - currently AUTOSAR/OSEK OS
  - accustomed programming model remains
- Our smallest system to date: Robertino
  - Autonomous robot navigating around obstacles
  - Control software running on ATmega8535
  - 8-bit AVR, 8 KiB Flash, 512 B SRAM



The KESO Multi-JVM

- Domain as realm of protection
  - Spatially isolated
  - Logical separation of heap, separate static fields
- Portals for inter-domain communication
  - remote procedure call mechanism
  - parameters are deep-copied



Application Model

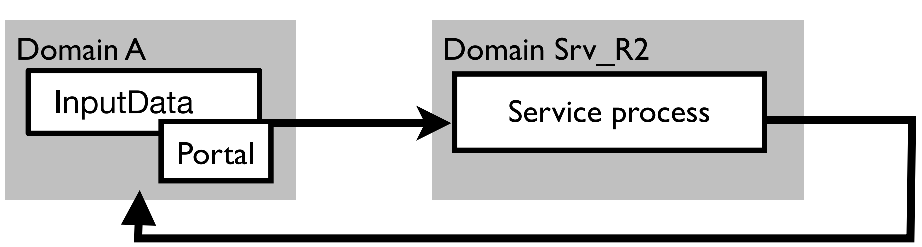
- Fault Model: Transient hardware faults
- Mixed-criticality application
  - Safety-critical control application
  - Mapped to a periodic task
  - Sensible FT measure: module redundancy
- Requirements on application for module redundancy
  - Spatial and temporal isolation
  - Run-to-completion semantics, does not block
  - No side effects, e.g. no read from indeterministic sources

FT Plugin Module Redundancy

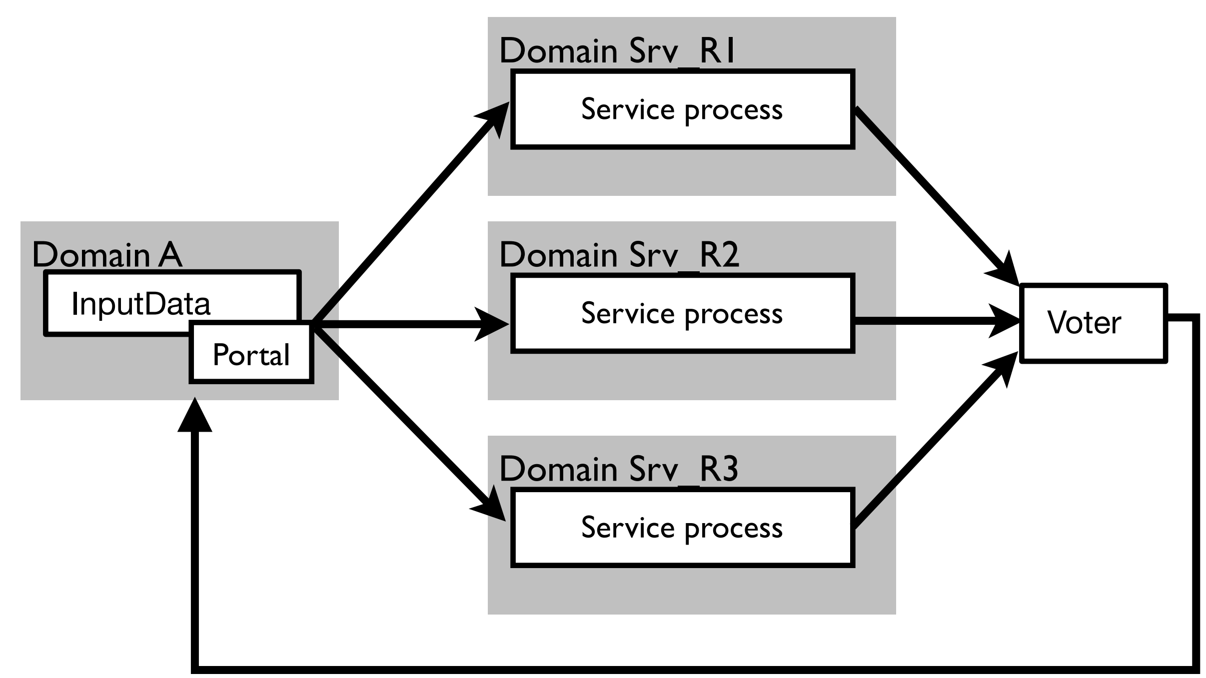
- *Example* for *one* FT technique
  - Sensible in mixed-criticality systems
- Requirements
  - Configurability
  - Module Isolation
  - Replica Determinism
  - Automated Replication

Configurability

- Level of redundancy derived from safety requirements
- Sphere of replication is the domain
  - Replicated via configuration file
- Portal as transition point between single and replicated execution
  - Interface is identical for single and replicated mode
  - Parameters are deep-copied: own copy in each replica



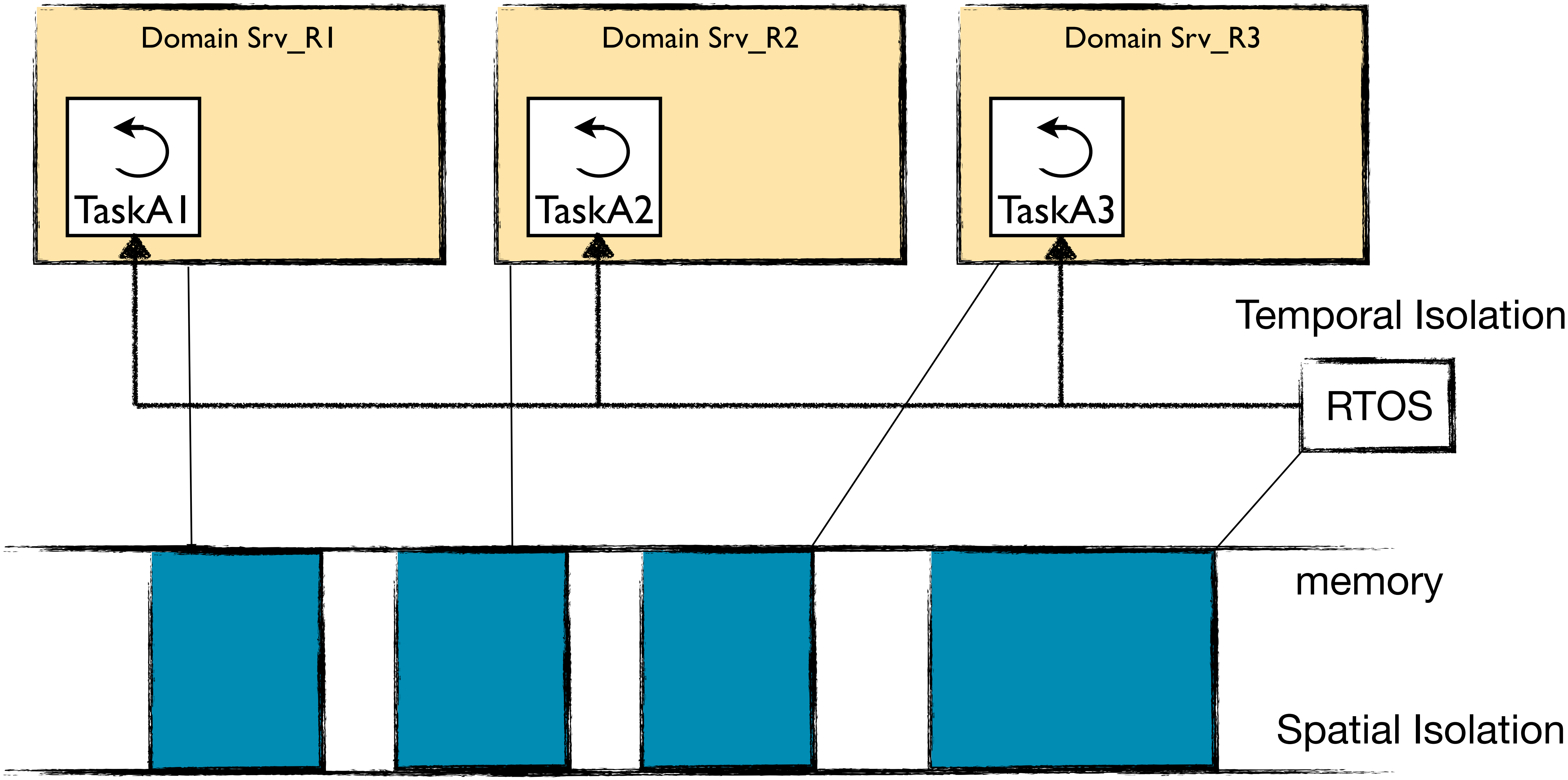
Single execution



Replicated execution

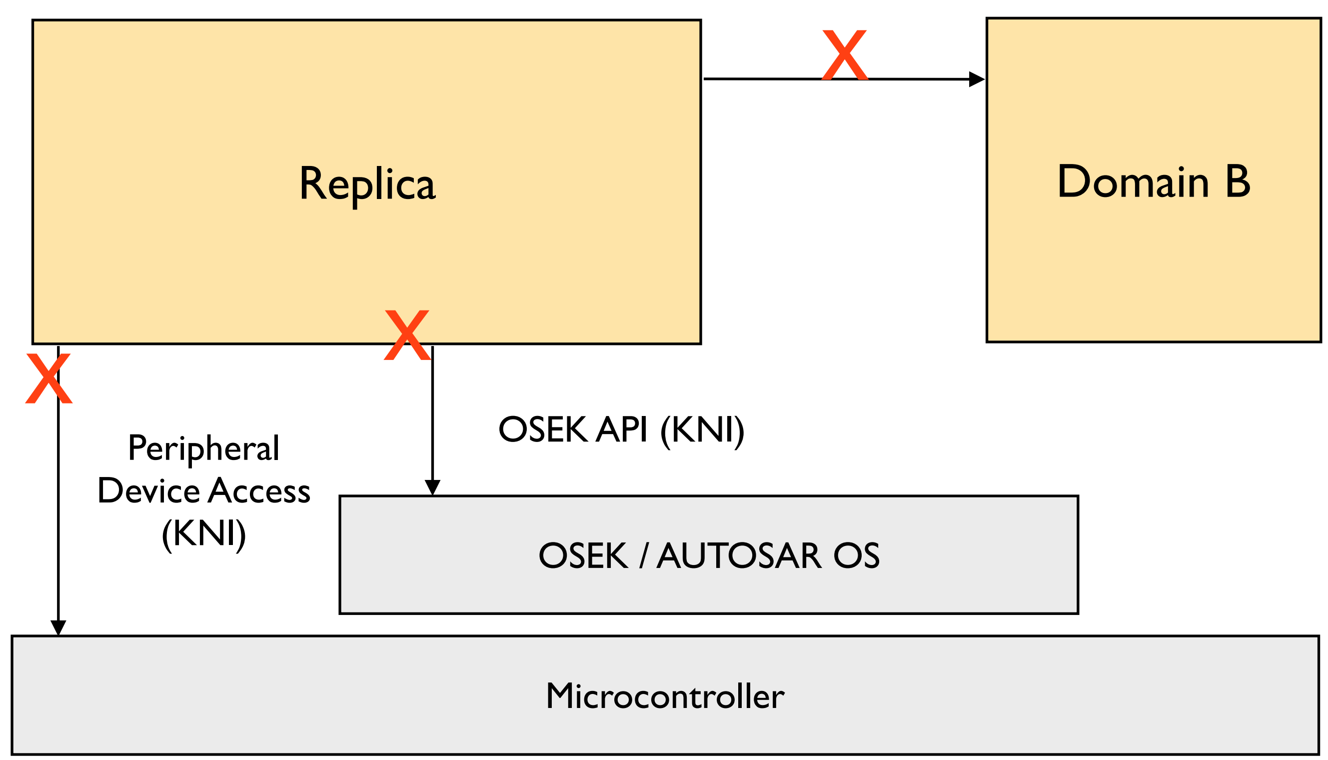
Module Isolation

- Spatial: SW (domains) *and* HW-based (MPU) protection
  - physical separation of the domain heaps
- Temporal: Execution time budgets for task



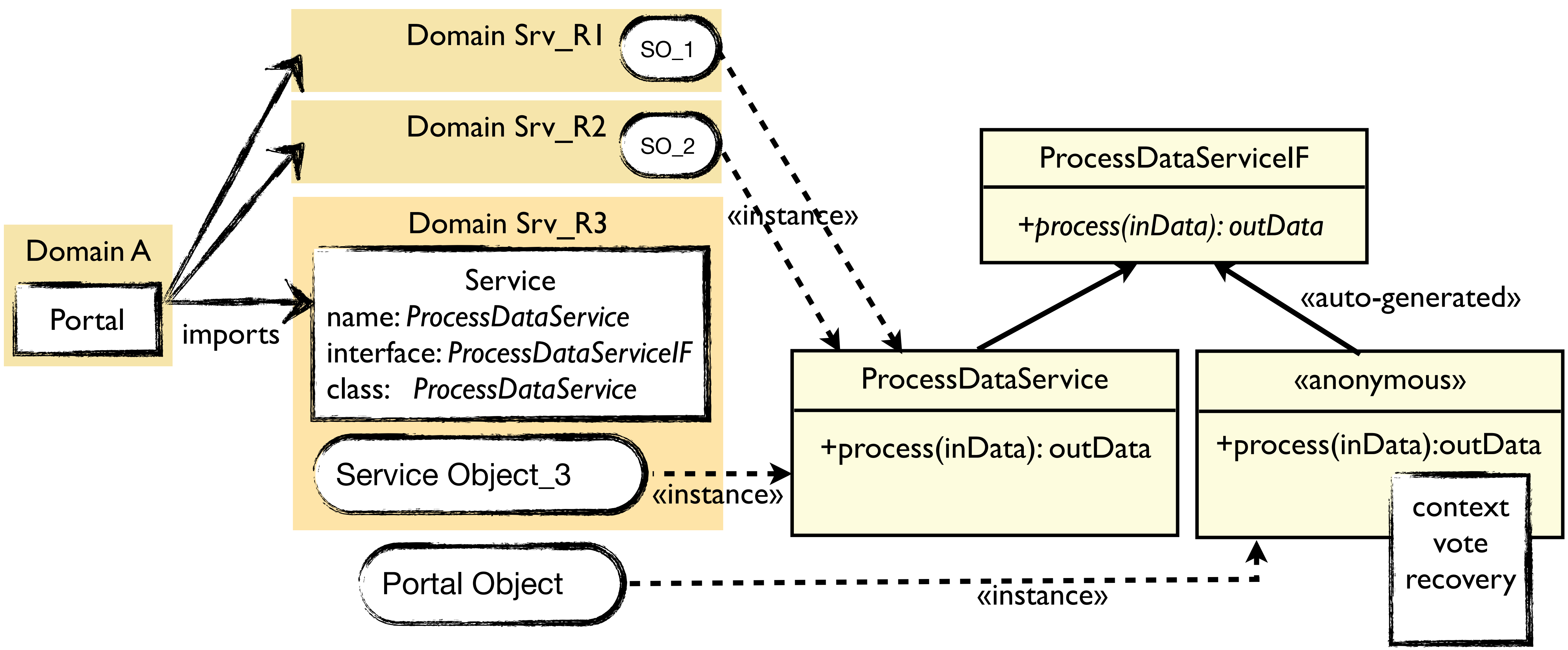
Replica Determinism

- Replica invocations must be ordered
- Critical locations *within* the replica executions
  - Communication (error spreading, indeterministic read of values)
  - Usually not in the scope of control applications
  - Problematic mechanisms are under control of the runtime system
- Alternative:
  - Implicit read and write access for external communication



Automated Replication

- Service: Java interface and implementation
- Replication plugin hooks into the proxy method
  - context switch
  - voter suitable for the return type and the number of replicas
  - recovery of a faulty domain from a healthy one



Conclusion

- Fault tolerance as a configurable property
  - FT is application specific
  - Balance protection and costs: Tailored runtime environment
  - Separation of FT measure from functional code
  - Compiler support
- Automated FT application dependent on
  - Application (written in type-safe language)
  - Hardware (Transient fault characteristics)
  - Safety requirements (amount of FT needed)
- KESO generates FT measure domain redundancy
  - Application does not have to be changed
  - Automated replication, generation of voting and recovery functions for a certain application

Future Work

- Additional FT techniques
  - Other replica and voting variants
  - Control flow information
  - Checksums to ensure data integrity, FT data structures, ...
- Hardening of runtime system
- Finer-grained hardening