



Aspects of and for Software-based Fault Tolerance

SOBRES '12

Olaf Spinczyk

olaf.spinczyk@tu-dortmund.de
<http://ess.cs.tu-dortmund.de/~os>



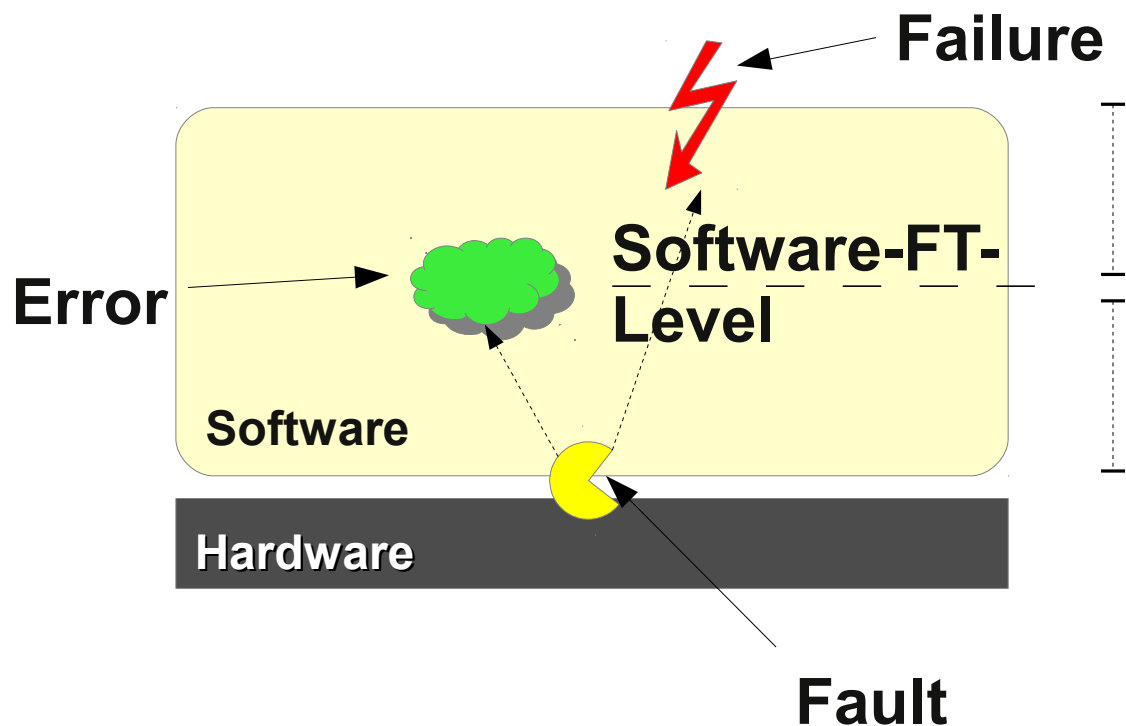
Embedded System Software Group
Computer Science 12, TU Dortmund

Danger of 
lightning talk

Agenda

- Terminology
- Motivation for Aspect-Orientation
- Requirements
- Example: Generic object protection

Faults, Error, Failures, and other Terms



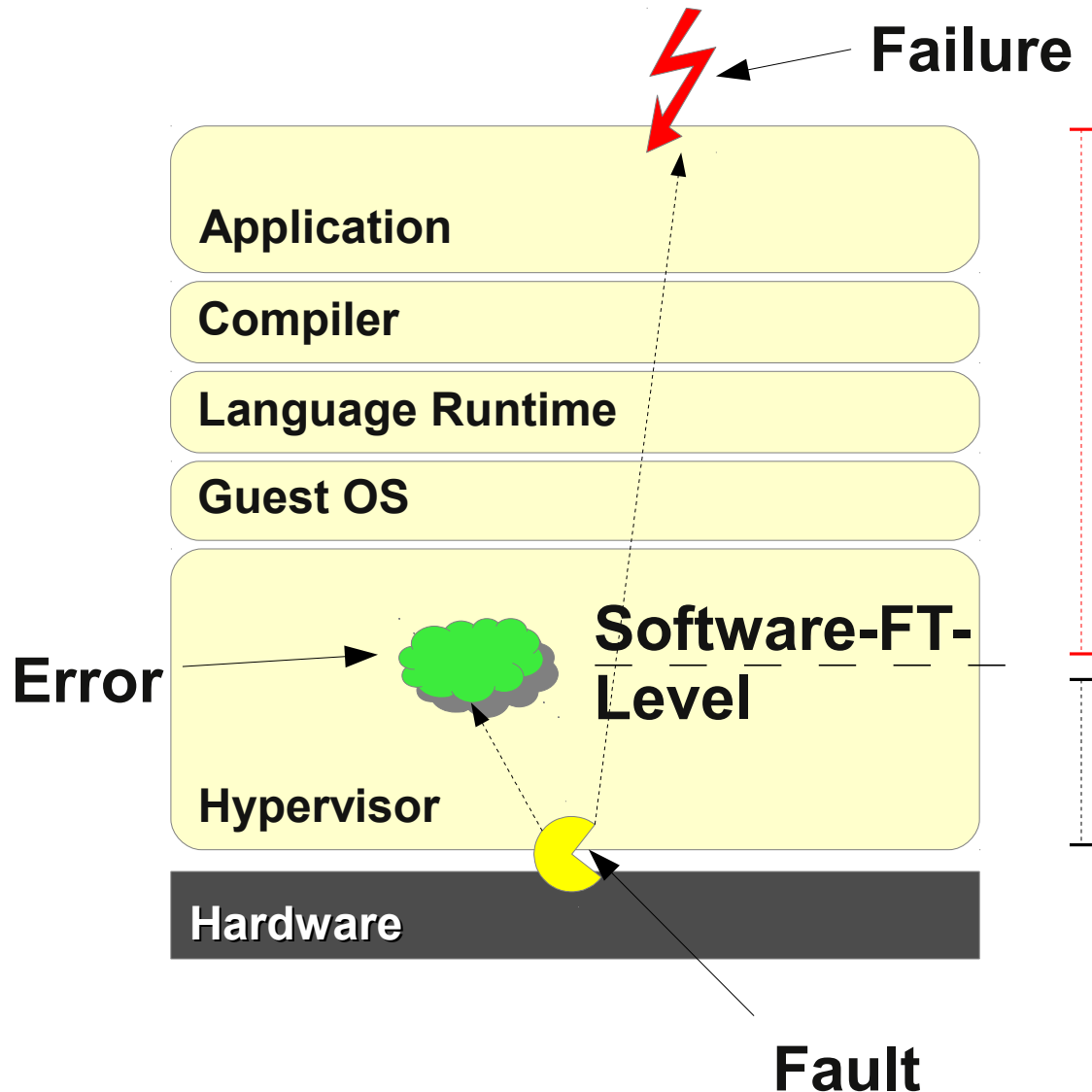
Determines whether and how an error has to be handled to avoid failure.

Error/Failure-Distance

Fault/Error-Distance

Determines whether and which specific kinds of faults can be detected before turning into a failure.

Complex Software Stacks



Determines whether and how an error has to be handled to avoid failure.

Error/Failure-Distance

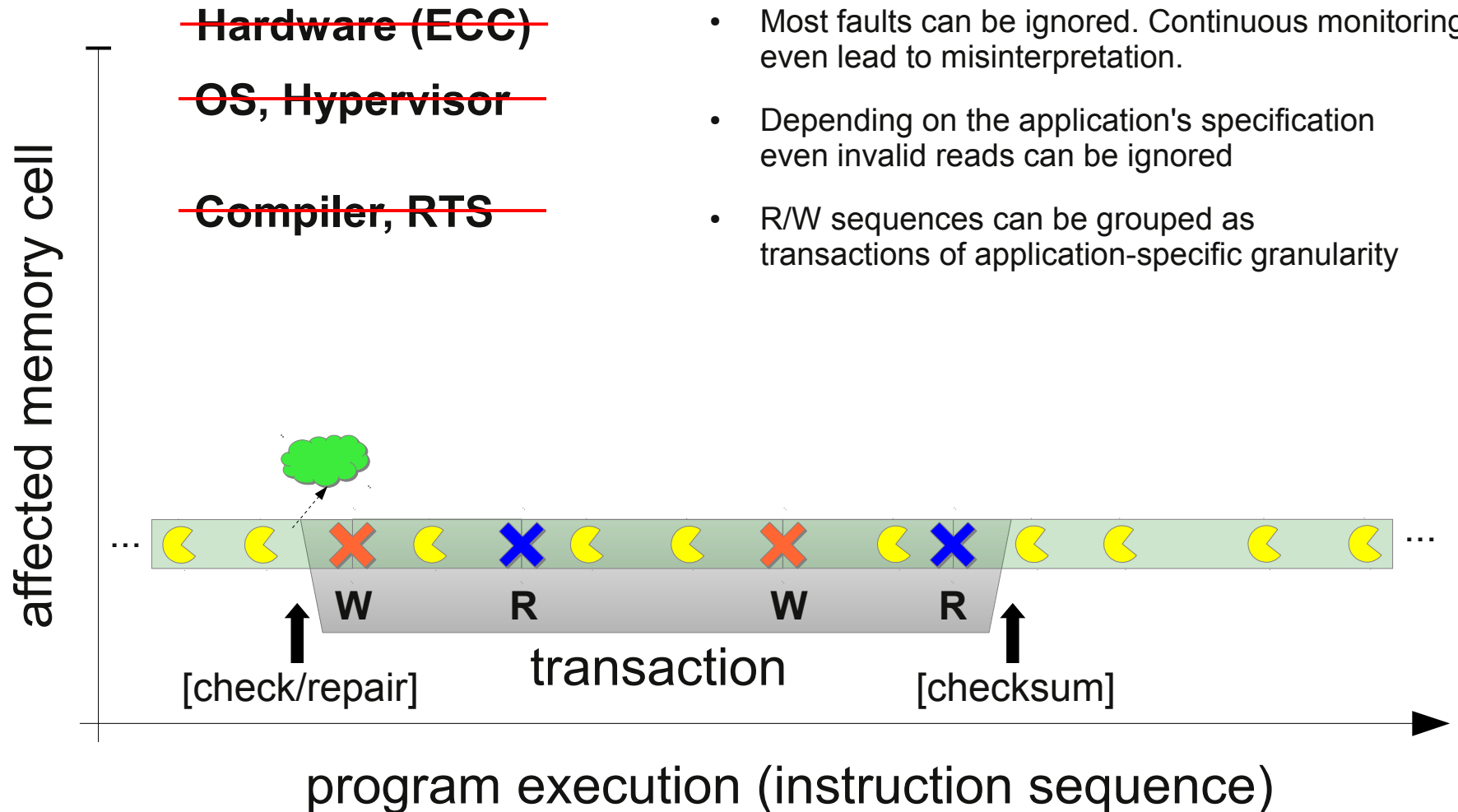
Fault/Error-Distance

Determines whether and which specific kinds of faults can be detected.

Error/Failure-Distance ...

- is highly relevant for efficiency
- is a semantic gap that can be reduced by ...
 - maximizing the Fault/Error-Distance
 - pushing the application's specification (partly) down to the software FT implementation

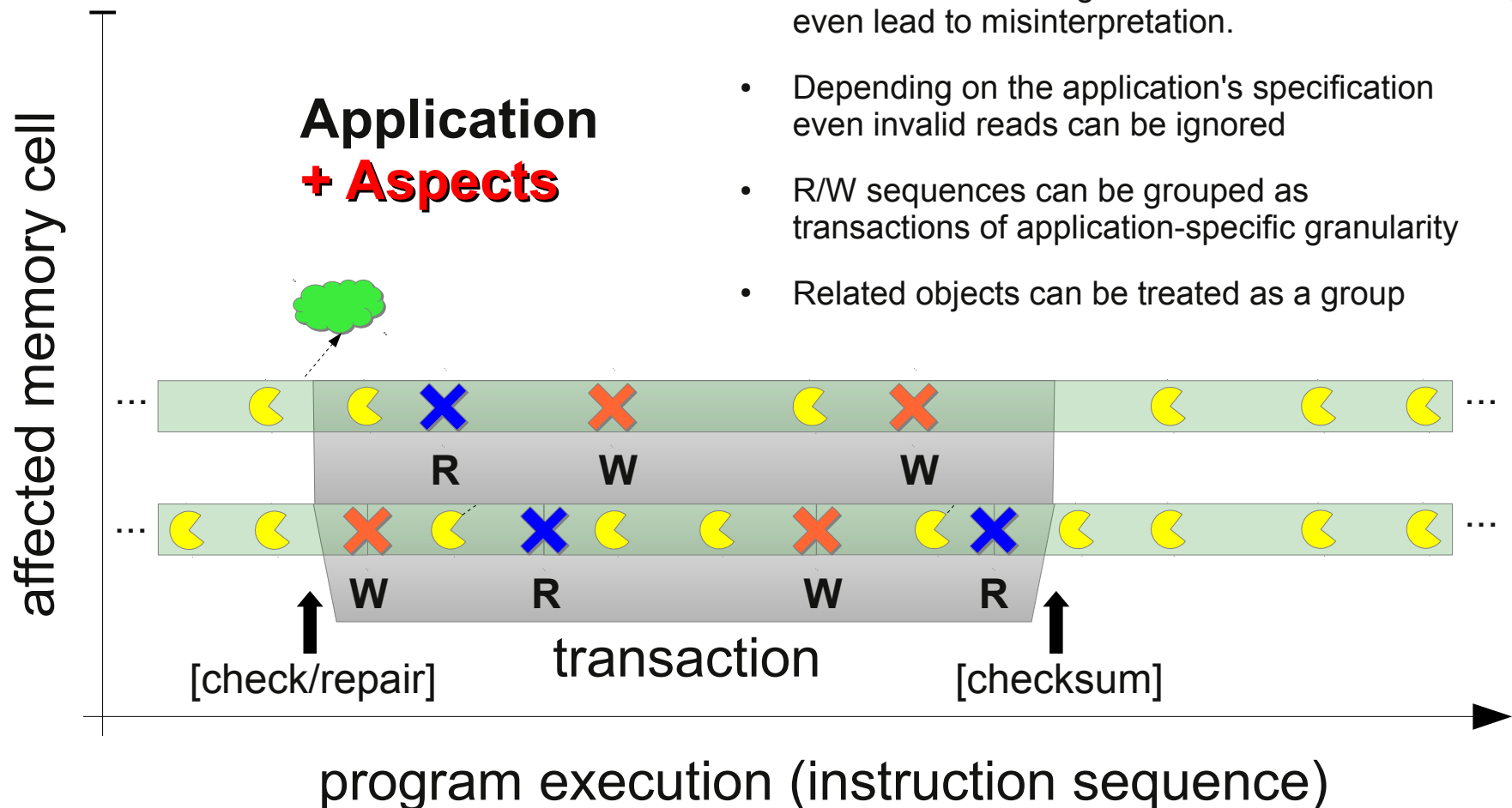
Example: Bit Flips in Variables



- Applications have specific read/write patterns
- Most faults can be ignored. Continuous monitoring might even lead to misinterpretation.
- Depending on the application's specification even invalid reads can be ignored
- R/W sequences can be grouped as transactions of application-specific granularity

Example: Bit Flips in Variables

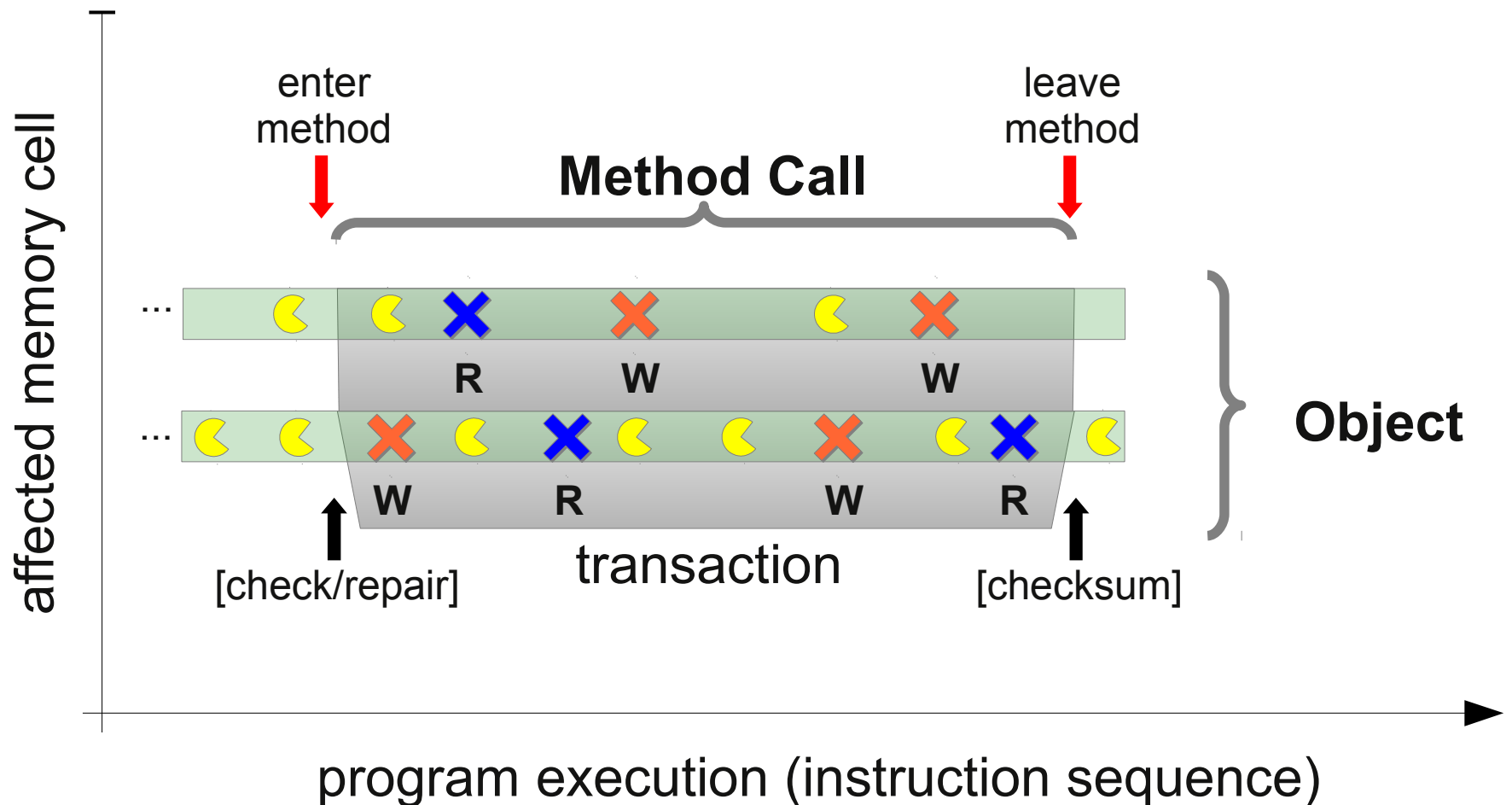
- Applications have specific read/write patterns
- Most faults can be ignored. Continuous monitoring might even lead to misinterpretation.
- Depending on the application's specification even invalid reads can be ignored
- R/W sequences can be grouped as transactions of application-specific granularity
- Related objects can be treated as a group



Requirements

- separation of concerns: avoid scattered FT code
- generic fault-tolerance measures for re-use
- abstraction level of the protected software itself
- exploit programmer knowledge
 - program structure
 - annotations
- do as much as possible at compile time

Example: Generic Object Protection





Example: Generic Object Protection (2)

```

aspect ObjectProtection {

    pointcut virtual protected() = 0;

    advice protected() : slice class {
        Checksum<JoinPoint>::Type chksum;

        void check_and_repair();
        void generate();
    };

    advice call(protected()) : around() {
        tjp->target()->check_and_repair();
        tjp->proceed();
        tjp->target()->generate();
    } };
    
```

```

aspect MyProt : ObjectProtection {
    pointcut virtual protected() = "MyData";
};
    
```

Requirements

- separation of concerns: avoid scattered FT code ✓
- generic fault-tolerance measures for re-use ✓
- exploit programmer knowledge
 - program structure ✓
 - annotations ✓
- do as much as possible at compile time
- abstraction level of the protected software itself ✓



AspectC++ Static JoinPoint-API

JoinPoint::BASECLASSES

Number of base classes

JoinPoint::BaseClass<N>::Type

Type of the N-th base class

JoinPoint::ELEMENTS

Number of member variables (attributes)

JoinPoint::Member<N>::Type

Type of the N-th member variable

JoinPoint::Member<N>::prot

Member access rights (AC::PROT_PRIVATE, ...PUBLIC, ...PROTECTED)

JoinPoint::Member<N>::spec

Member specifiers (AC::SPEC_STATIC, ...MUTABLE)

JoinPoint::Member<N>::pointer(T*obj)

Address of the N-th member in object obj

JoinPoint::member_pointer(int n, T*obj)

Address of the n-th member with a non-constant n

JoinPoint::member_name(int n)

Member name as a string



Thank you
for your attention!