Avoiding Leakage and Synchronization Attacks through Enclave-Side Preemption Control

Marcus Völp, Adam Lackorzynski*, Jérémie Decouchant, Vincent Rahli, Francisco Rocha, and Paulo Esteves-Veríssimo

University of Luxembourg SnT
CritiX Lab
Luxembourg
<name>.<surname>@uni.lu

*Kernkonzept GmbH and
TU Dresden – Operating-systems group
Dresden, Germany
adam.lackorzynski@kernkonzept.com

1st Workshop on System Software for Trusted Execution (SysTEX 2016), Dec. 12, 2016, Trento, Italy

The functionality/code size dilemma

- application scenarios require the system to implement a certain set of functionalities
- implementing these functionalities comes at the cost of a certain minimal amount of code
  - even if development time and costs don't matter; and
  - even if you only use high-class developers
- correlation of code size and complexity to vulnerabilities
  - Chou et al., *"An Empirical Study of Operating Systems Errors"*, SOSP 2001
  - Asadollah et al., *"A Study of Concurrency Bugs in an Open Source Software"*, OSS 2016

The functionality/code size dilemma

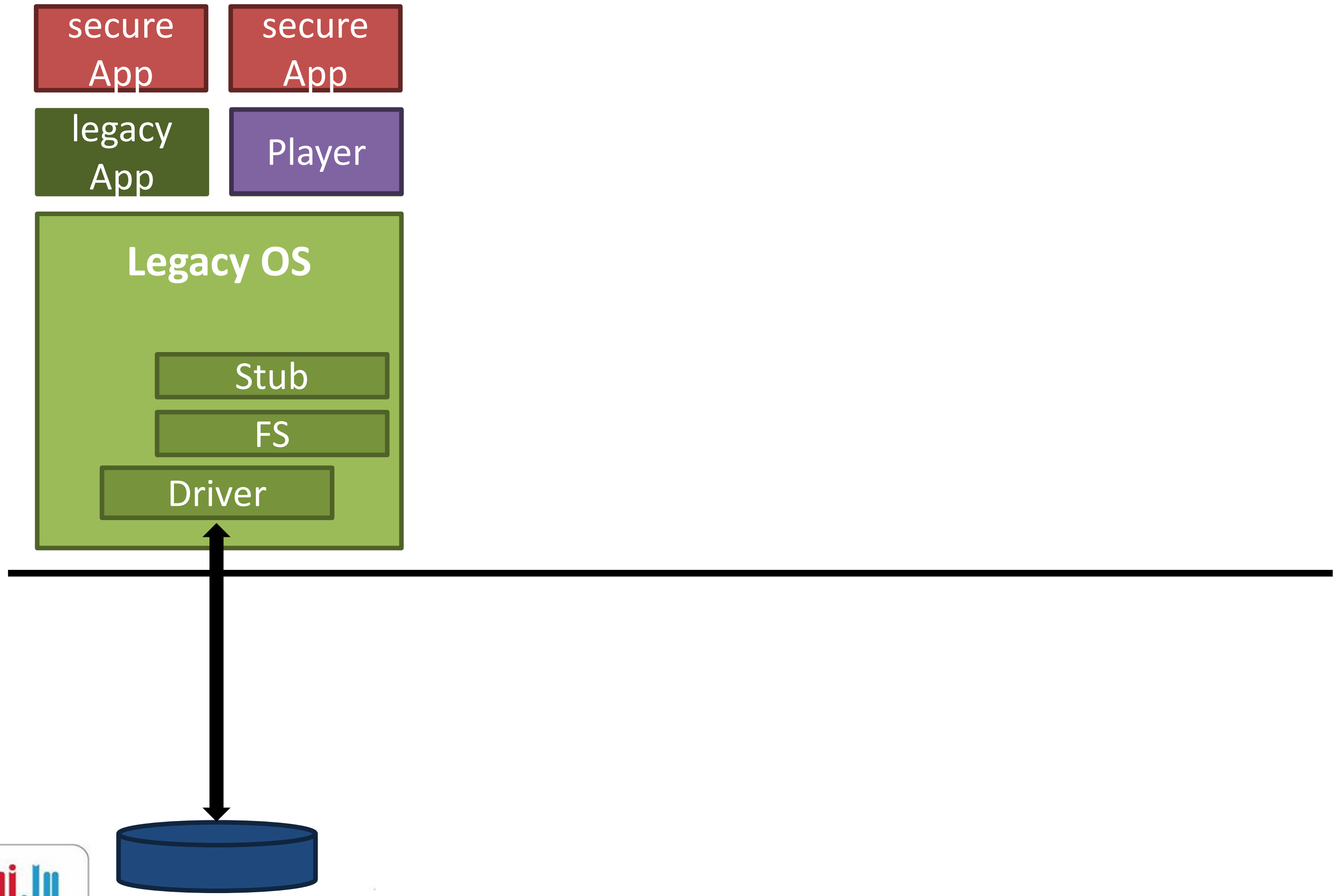
SNT securityandtrust.lu CRITIX

- application scenarios require the system to implement a certain set of functionalities
- implementing these functionalities comes at the cost of a certain minimal amount of code
  - even if d … a
  - even if y
- correlation … nerabilities
  - Chou et al., *"An Empirical Study of Operating Systems Errors"*, SOSP 2001
  - Asadollah et al., *"A Study of Concurrency Bugs in an Open Source Software"*, OSS 2016

• RTOS	ca. 5 KLOC
• Microkernel	10 – 15 KLOC
• Legacy OS	15 – 50 MLOC

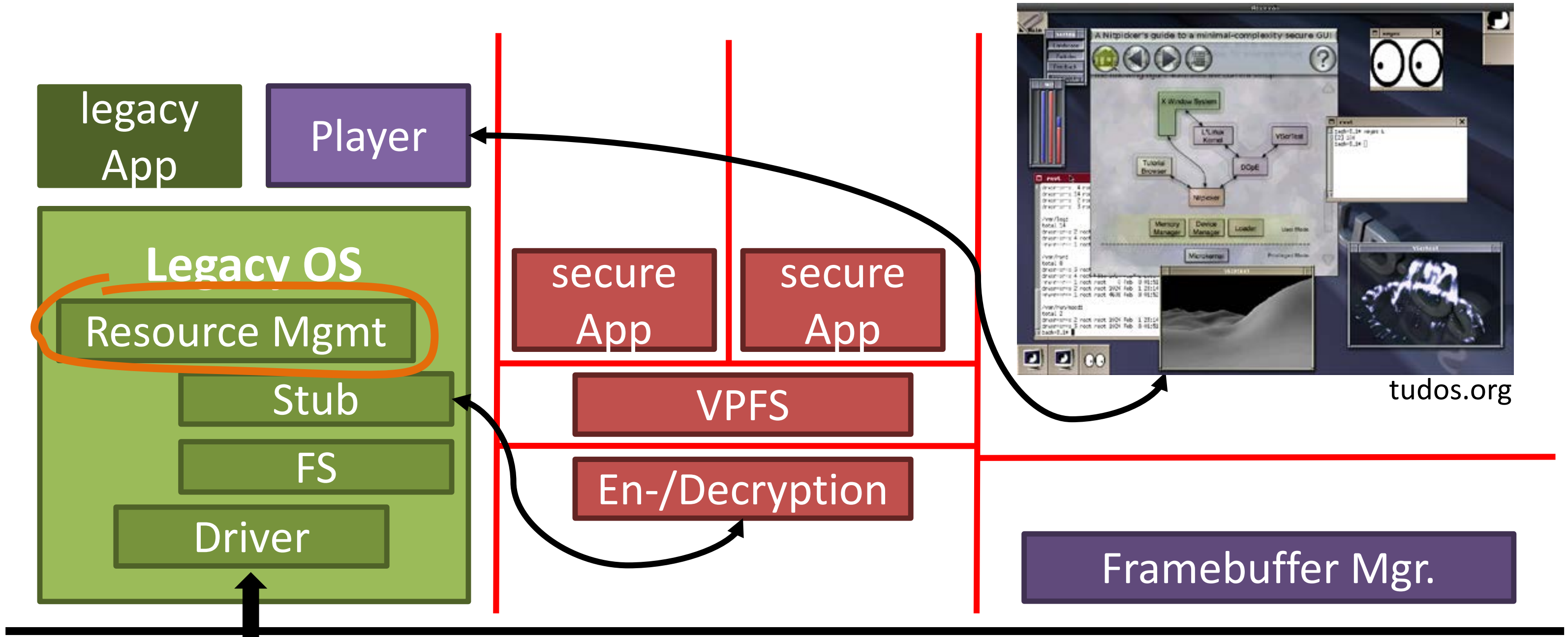
5-13 PY formal verification (RTOS, Microkernel)

Intransitive trust

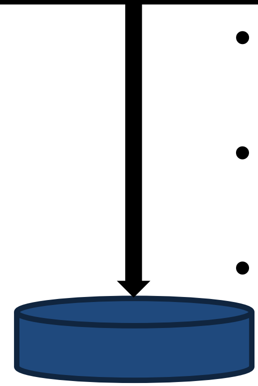


Intransitive trust

SnT
securityandtrust.lu
CRITIX

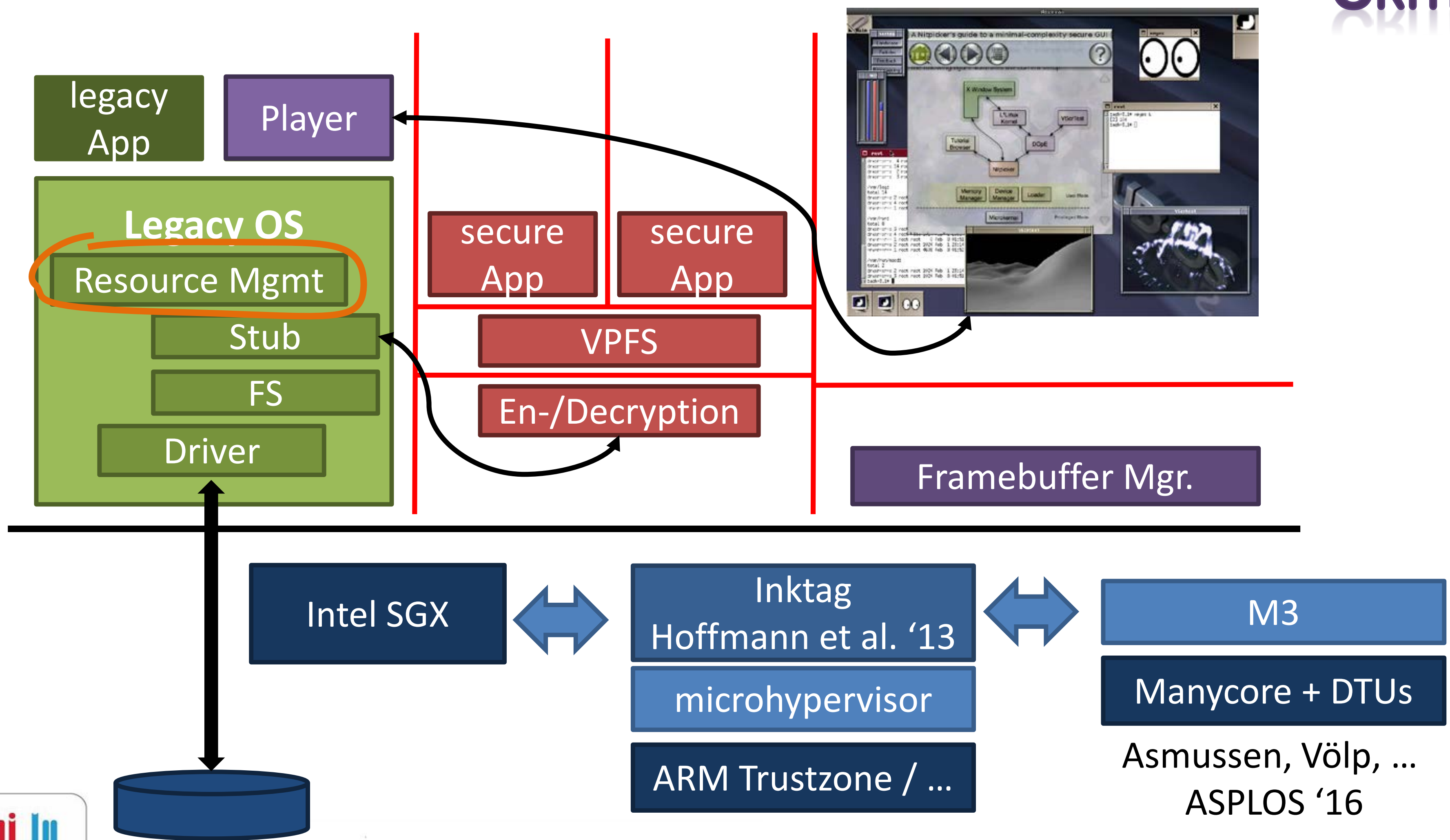


- Weinhold et al., "*jVPFS: Adding Robustness to a Secure Stacked File System with Untrusted Local Storage Components*", USENIX ATC, 2011
- Singaravelu et al., "*Reducing TCB Complexity for Security-Sensitive Applications: Three Case Studies*", Eurosys, 2006
- …



Asmussen, Völp, …
ASPLOS '16

Intransitive trust



SGX Vulnerabilities

AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves

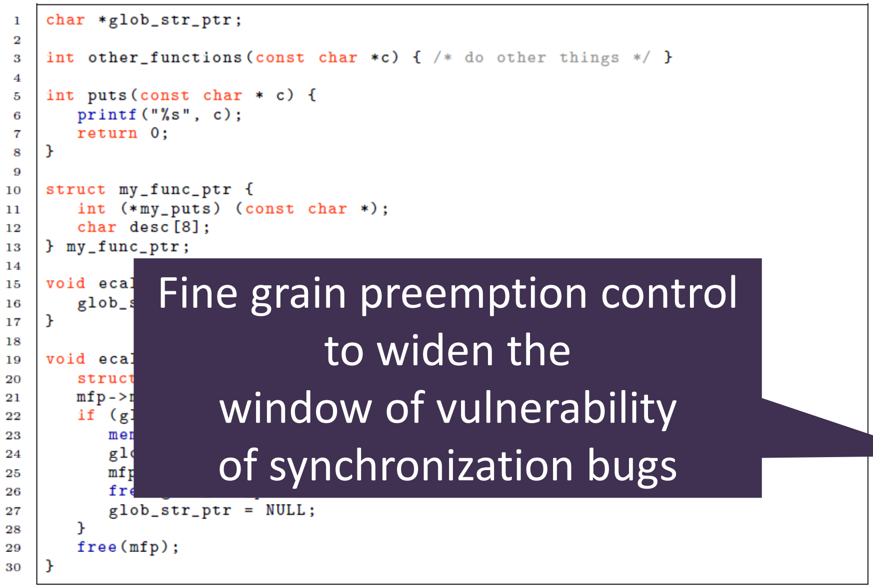
Nico Weichbrodt, Anil Kurmus†, Peter Pietzuch*, and Rüdiger Kapitza

TU Braunschweig, †IBM Research Zurich, *Imperial College London
{weichbr,kapitza}@ibr.cs.tu-bs.de,kur@zurich.ibm.com,prp@imperial.ac.uk

Abstract. Intel's Software Guard Extensions (SGX) provide a new hardware-based trusted execution environment on Intel CPUs using secure enclaves that are resilient to accesses by privileged code and physical attackers. Origi... promise to prot... In this paper, ... sation bugs can... By exploiting t... bugs in enclav... access control... We present A... multithreade... only manipu... enclave code... mentation fa... Skylake CP... and TOCT...

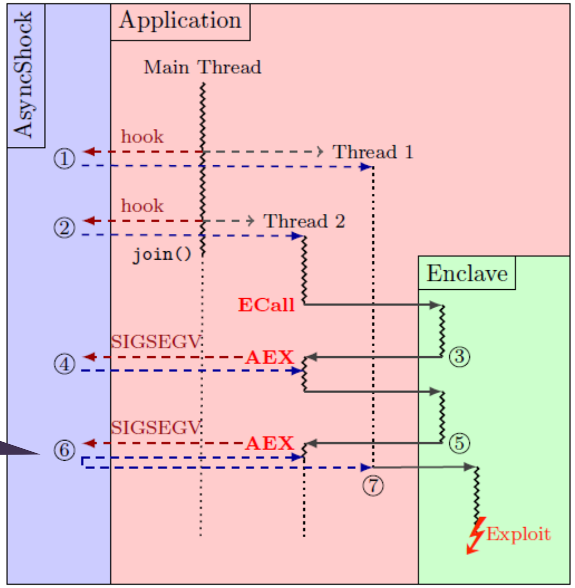
Keywords ... chronisatio...

Source: AsyncShock

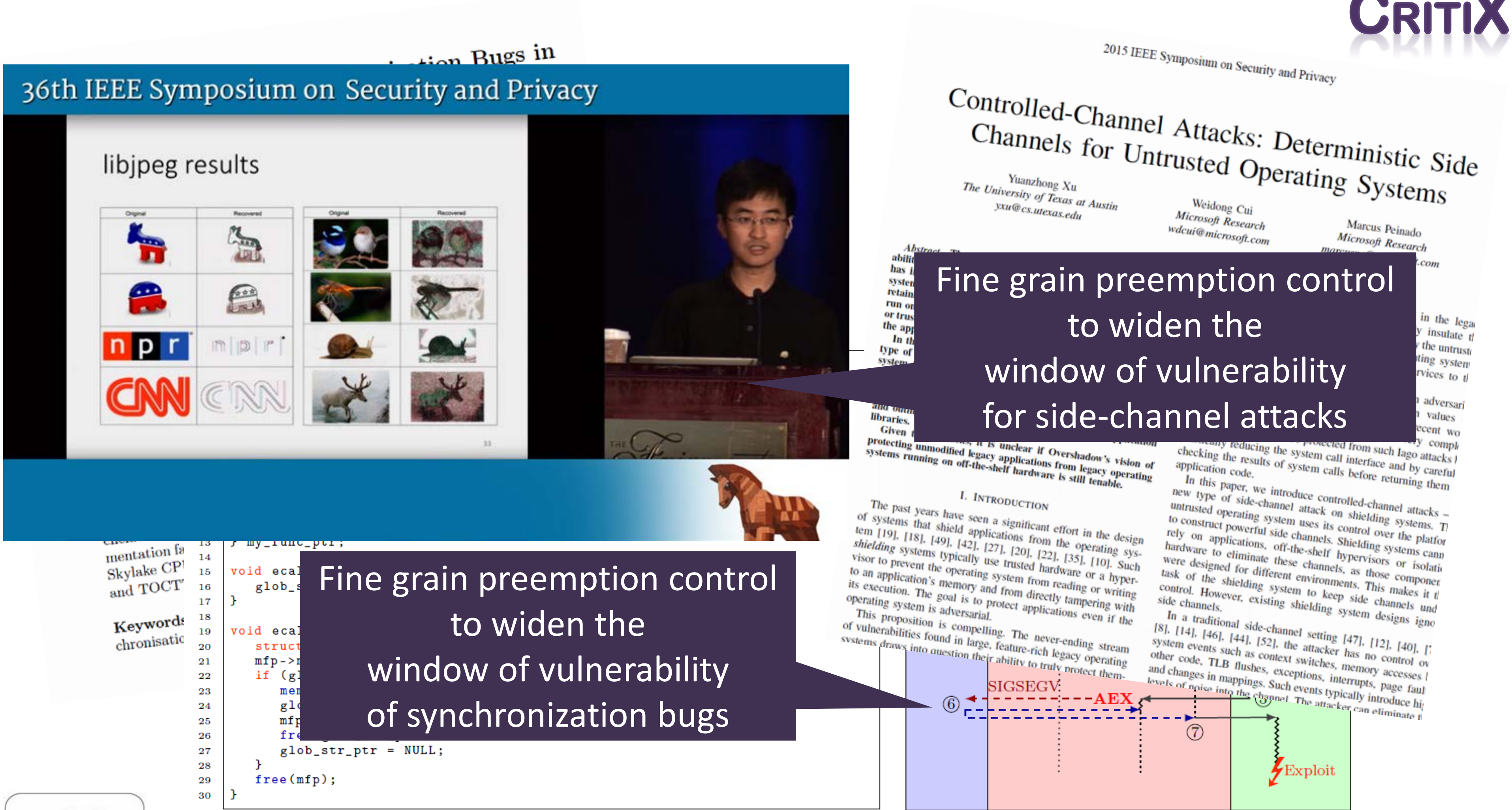


```
char *glob_str_ptr;

int other_functions(const char *c) { /* do other things */ }

int puts(const char * c) {
   printf("%s", c);
   return 0;
}

struct my_func_ptr {
   int (*my_puts) (const char *);
   char desc[8];
} my_func_ptr;

void ecal
   glob_s
}

void ecal
   struct
   mfp->
   if (g
      mem
      gl
      mfp
      fre
      glob_str_ptr = NULL;
   }
   free(mfp);
}
```

Fine grain preemption control to widen the window of vulnerability of synchronization bugs



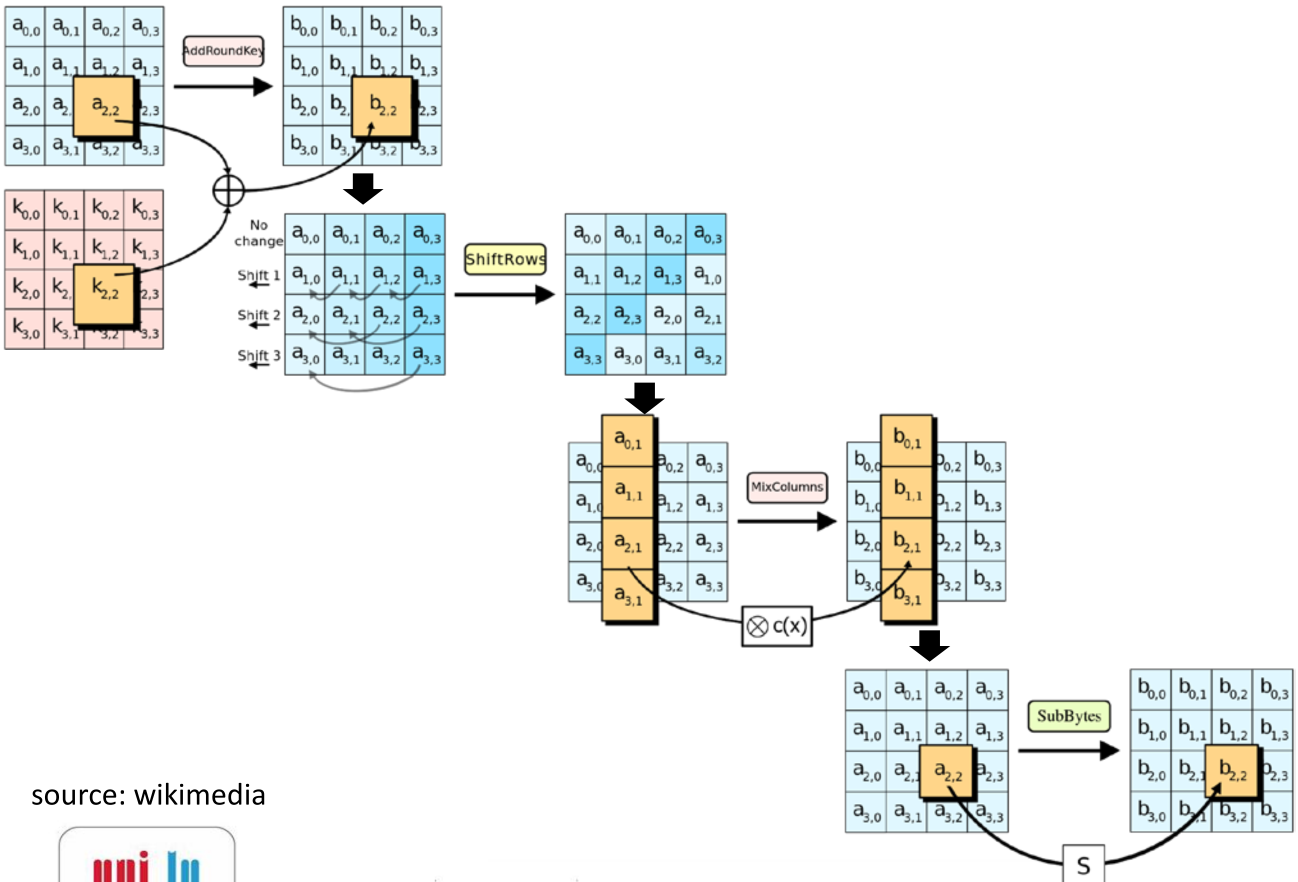
SGX Vulnerabilities



SGX Vulnerabilities

- Running Example:

  Osvik et al., *"Cache Attacks and Countermeasures: the Case of AES"*, CT-RSA 2006



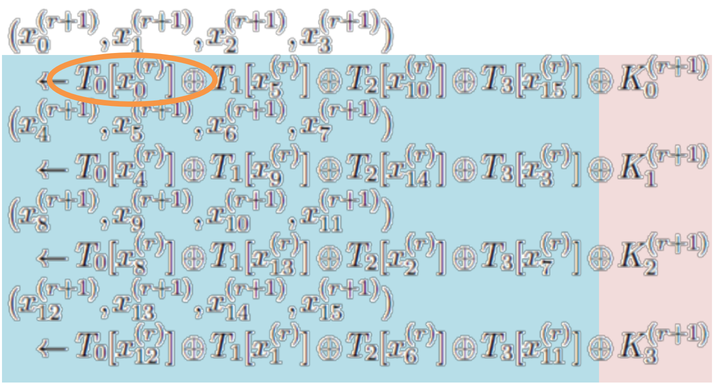
in-memory tables T_i

source: wikimedia

SGX Vulnerabilities

- Running Example:

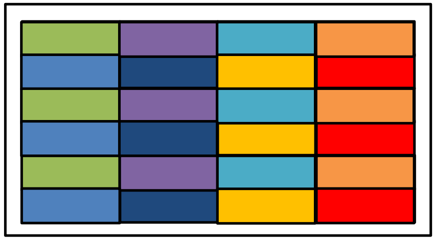
  Osvik et al., *“Cache Attacks and Countermeasures: the Case of AES”*, CT-RSA 2006



R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
…
read $T_i[0]$
…
read $T_i[n]$

T_i

T_i



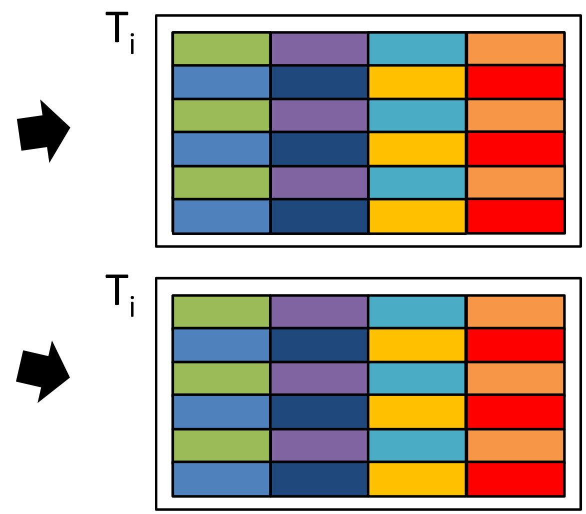
SGX Vulnerabilities

- Running Example:

  Osvik et al., “*Cache Attacks and Countermeasures: the Case of AES*”, CT-RSA 2006

$$
\begin{aligned}
(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}) &\leftarrow T_0[x_0^{(r)}] \oplus T_1[x_5^{(r)}] \oplus T_2[x_{10}^{(r)}] \oplus T_3[x_{15}^{(r)}] \oplus K_0^{(r+1)} \\
(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}) &\leftarrow T_0[x_4^{(r)}] \oplus T_1[x_9^{(r)}] \oplus T_2[x_{14}^{(r)}] \oplus T_3[x_3^{(r)}] \oplus K_1^{(r+1)} \\
(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}) &\leftarrow T_0[x_8^{(r)}] \oplus T_1[x_{13}^{(r)}] \oplus T_2[x_2^{(r)}] \oplus T_3[x_7^{(r)}] \oplus K_2^{(r+1)} \\
(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}) &\leftarrow T_0[x_{12}^{(r)}] \oplus T_1[x_1^{(r)}] \oplus T_2[x_6^{(r)}] \oplus T_3[x_{11}^{(r)}] \oplus K_3^{(r+1)}
\end{aligned}
$$

R_6 = read $T_i[0]$
cmp 0, x_j
R_5 = cmov R_6
R_0 = xor R_0, R_5
…



low indistinguishable data access pattern embedded into low indistinguishable control flow

SGX Vulnerabilities

- Running Example:

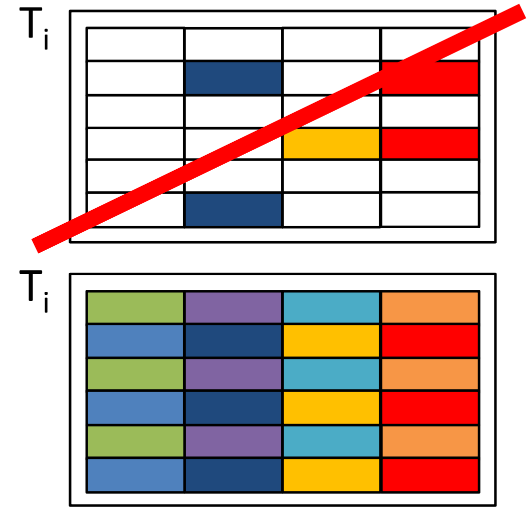
  Osvik et al., *“Cache Attacks and Countermeasures: the Case of AES”*, CT-RSA 2006

$$(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}) \leftarrow T_0[x_0^{(r)}] \oplus T_1[x_5^{(r)}] \oplus T_2[x_{10}^{(r)}] \oplus T_3[x_{15}^{(r)}] \oplus K_0^{(r+1)}$$
$$(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}) \leftarrow T_0[x_4^{(r)}] \oplus T_1[x_9^{(r)}] \oplus T_2[x_{14}^{(r)}] \oplus T_3[x_3^{(r)}] \oplus K_1^{(r+1)}$$
$$(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}) \leftarrow T_0[x_8^{(r)}] \oplus T_1[x_{13}^{(r)}] \oplus T_2[x_2^{(r)}] \oplus T_3[x_7^{(r)}] \oplus K_2^{(r+1)}$$
$$(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}) \leftarrow T_0[x_{12}^{(r)}] \oplus T_1[x_1^{(r)}] \oplus T_2[x_6^{(r)}] \oplus T_3[x_{11}^{(r)}] \oplus K_3^{(r+1)}$$

disable preemptions

R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
…
read $T_i[0]$
…
read $T_i[n]$

enable preemptions



This talk

Re-investigate delayed-preemption:

Towards Scalable Multiprocessor Virtual Machines

Volkmar Uhlig   Joshua LeVasseur   Espen Skoglund   Uwe Dannowski

System Architecture Group
Universität Karlsruhe
contact@l4ka.org

Abstract

A multiprocessor virtual machine benefits its guest operating system in supporting scalable job throughput and request latency—useful properties in server consolidation where servers require several of the system processors for steady state or to handle load bursts.

Typical operating systems, optimized for multiprocessor systems in their use of spin-locks for critical sections, can defeat flexible virtual machine scheduling due to lock-holder preemption and misbalanced load. The virtual machine must assist the guest operating system to avoid lock-holder preemption and to schedule jobs with knowledge of asymmetric processor allocation. We want to support a virtual machine environment with flexible scheduling policies, while maximizing guest performance.

This paper presents solutions to avoid lock-holder preemption for both fully virtualized and paravirtualized environments. Experiments show that we can nearly eliminate the effects of lock-holder preemption. Furthermore, the paper presents a scheduler feedback mechanism that despite the presence of asymmetric processor allocation achieves optimal and fair load balancing in the guest operating system.

1 Introduction

A recent trend in server consolidation has been to provide virtual machines that can be safely multiplexed on

of guests, such that they only ever access a fraction of the physical processors, or alternatively time-multiplex guests across a set of physical processors to, e.g., accommodate for spikes in guest OS workloads. It can also map guest operating systems to virtual processors (which can exceed the number of physical processors), and migrate between physical processors without notifying the guest operating systems. This allows for, e.g., migration to other machine configurations or hot-swapping of CPUs without adequate support from the guest operating system. It is important to recognize that allowing arbitrary allocation policies offers much more flexibility than schemes where one can only configure a virtual machine to either have an arbitrary share of a single processor [7, 24], or have uniform shares over multiple physical processors [10, 24].

Isolating commodity operating systems within virtual machines can defeat the assumptions of the guest operating system. Where the guest operating system expects constant resource configurations, critical timing behavior, and unrestrained access to the platform, the virtual machine provides illusionary access as it sees fit. Several methods exist to attempt to satisfy (a subset of) the assumptions of the guest operating system. The solutions may focus on the issues of instruction set emulation, such as trapping on system instructions [22], or they may focus on the behavior of the guest operating system algorithms, such as dynamic allocation of physical memory [25].

This paper presents solutions to two problems that arise with scheduling of virtual machines which provide

- How can we allow user-level applications (in enclaves) to disable preemptions without being able to monopolizing the system?
- How can we prevent solicited exits through which the management OS could regain control?
- How can we translate delayed-preemption to Intel SGX?

This talk

disable preemptions

R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
...
read $T_i[0]$
...
read $T_i[n]$

enable preemptions

How can we prevent solicited exits in sensitive code?

How can we make sure the enclave enables preemptions again?

This talk

disable preemptions

prepare
if preempted
    goto retry
R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
...
read $T_i[0]$
...
read $T_i[n]$

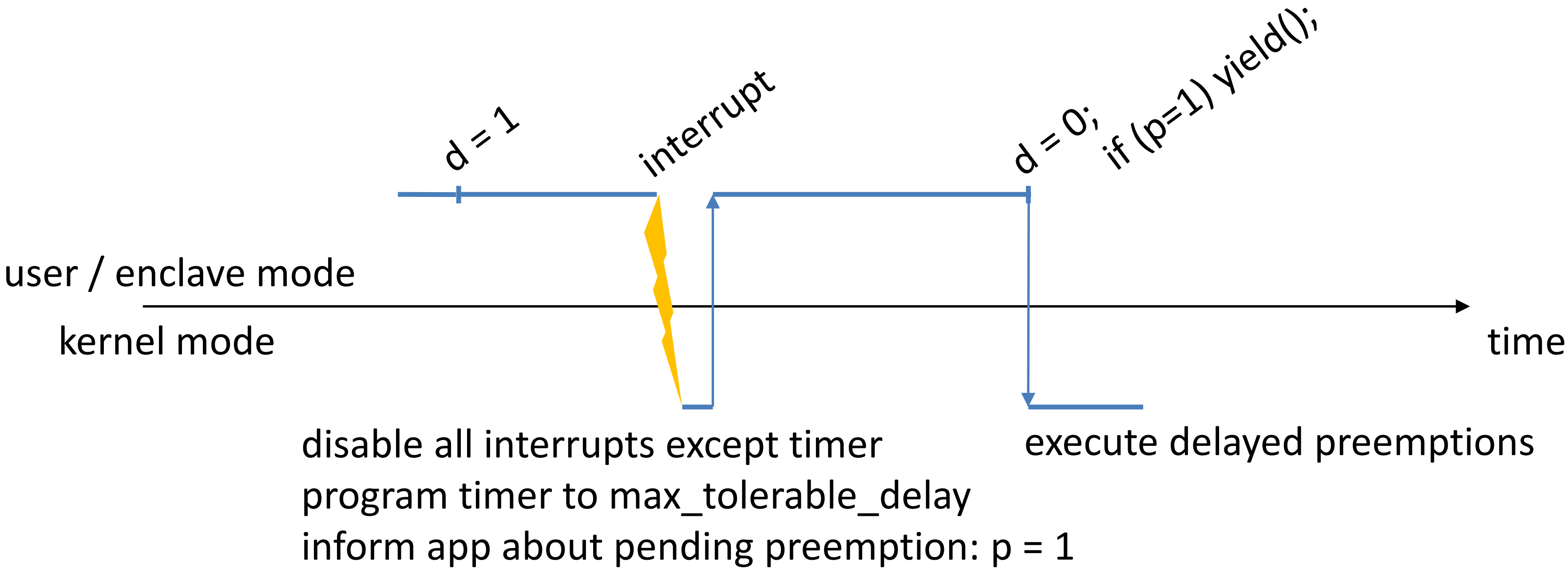
enable preemptions

How can we prevent solicited exits in sensitive code?

How can we make sure the enclave enables preemptions again?

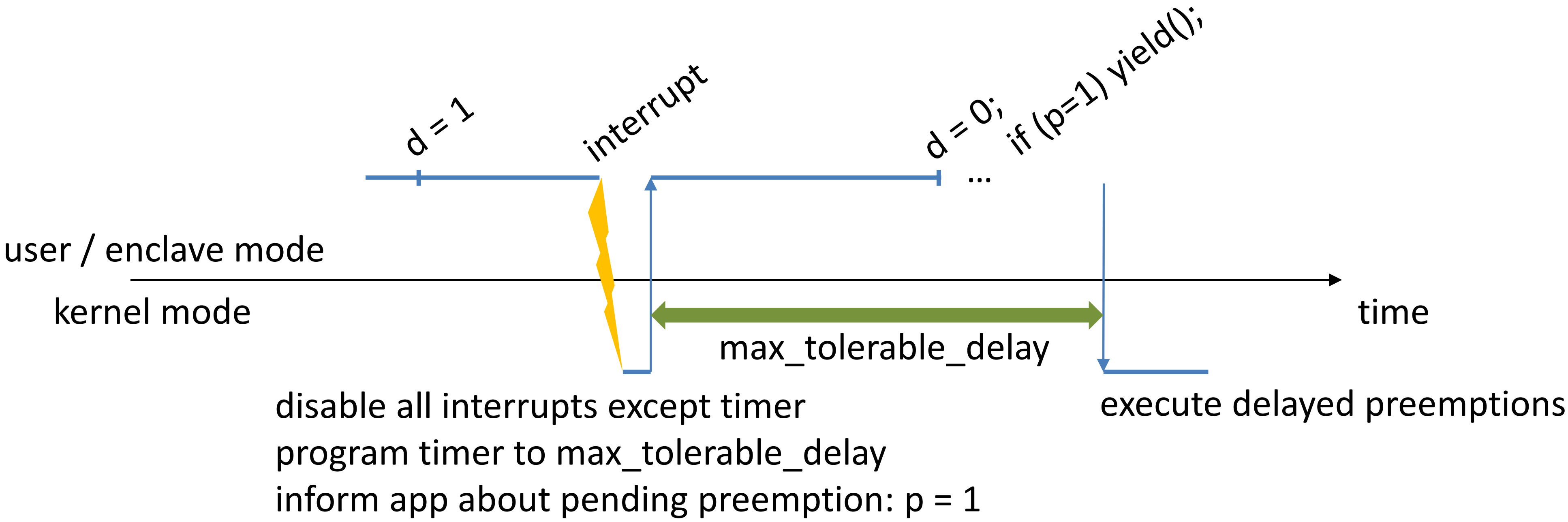
Delayed Preemption

- … in a Trusted-Trustworthy Hypervisor



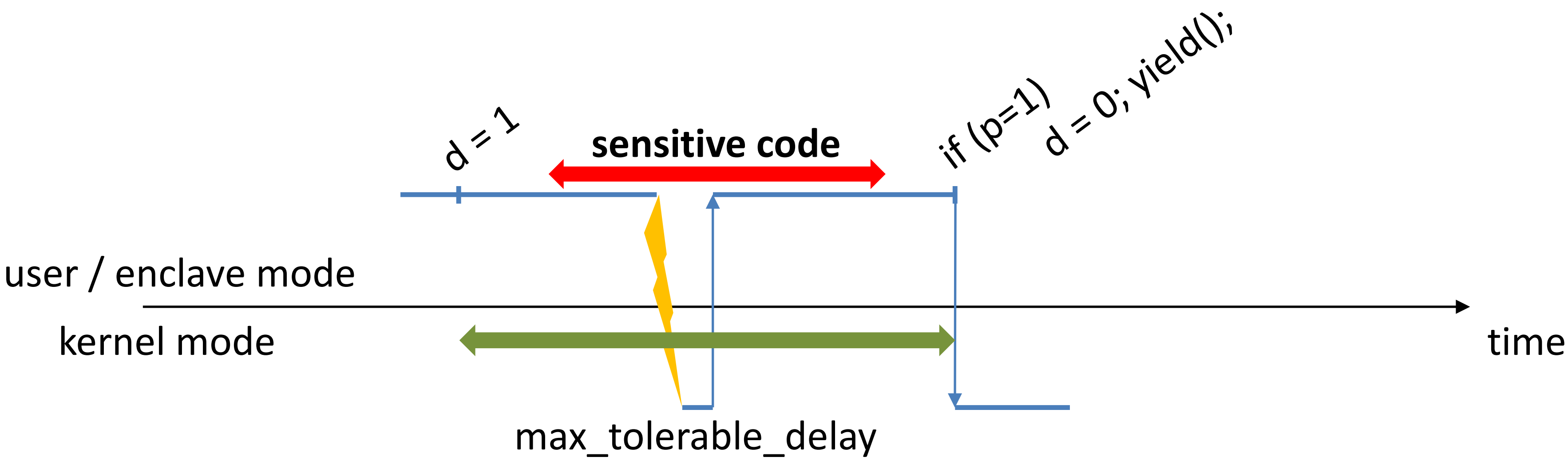
Delayed Preemption

- … in a Trusted-Trustworthy Hypervisor



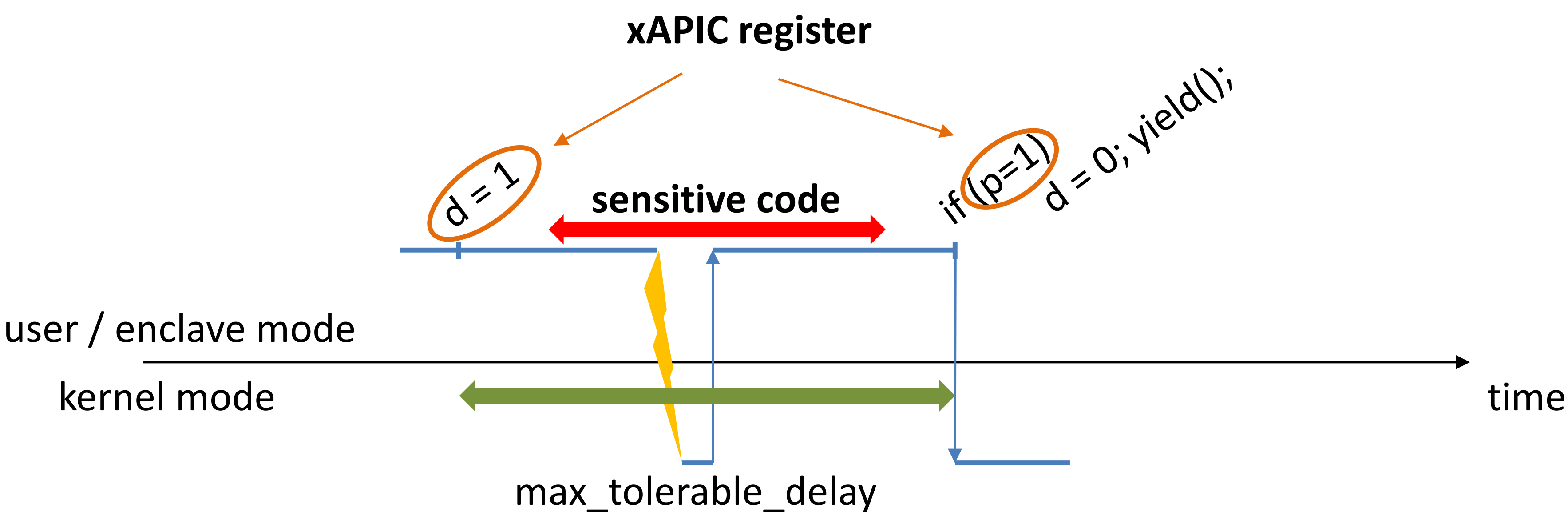
Delayed Preemption

- … in a Trusted-Trustworthy Hypervisor



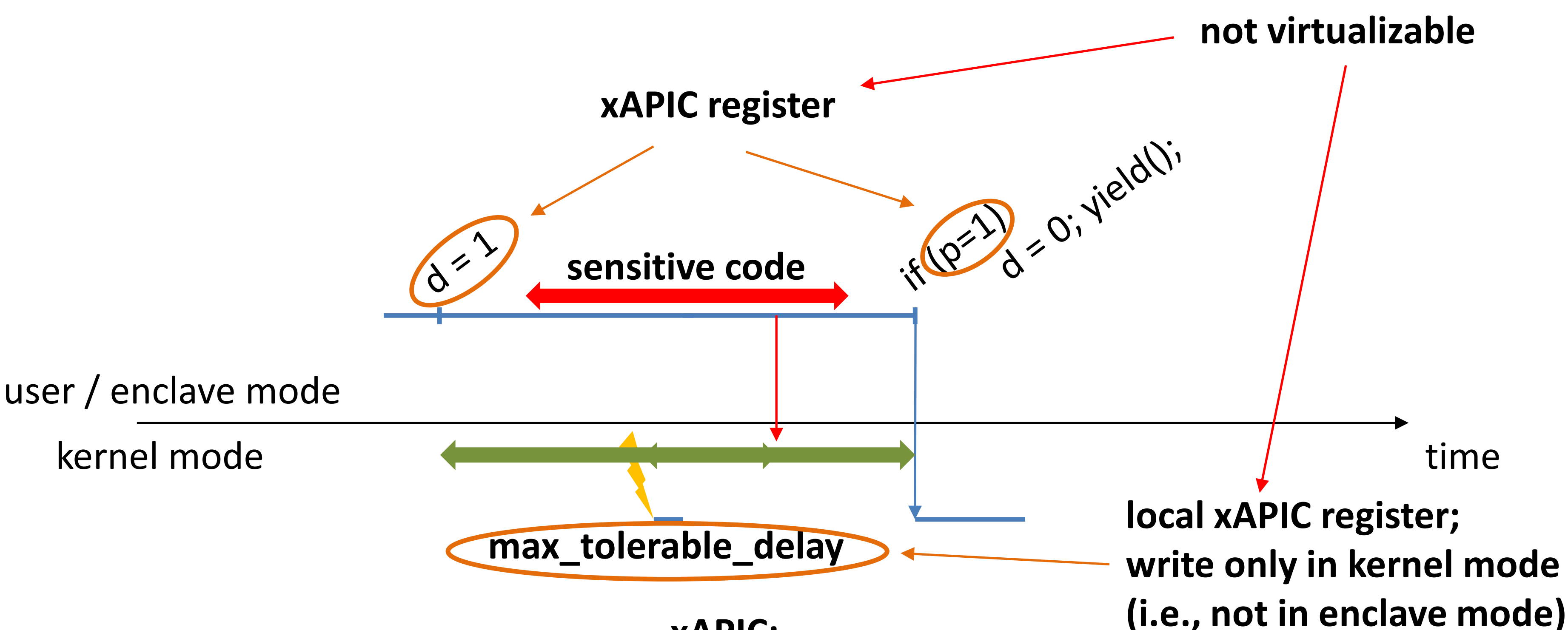
Delayed Preemption

- … in SGX



Delayed Preemption

- … in SGX



xAPIC:
set timer on first preemption;
don't interrupt application

Solicited Exits

disable preemptions

prepare
if preempted
    goto retry
R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
…
read $T_i[0]$
…
read $T_i[n]$

enable preemptions

Solicited Exits

```
retry:
xApic.d = 1;
```

prepare
if (p = 1)
    goto retry
R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
…
read $T_i[0]$
…
read $T_i[n]$

```
xApic.d = 0
// if (xApic.p = 1) -> AEX
```

✓ Trigger all such exits during non-sensitive prepare phase;
Set p flag to make code aware of these exits;
Context switch p flag as part of enclave state

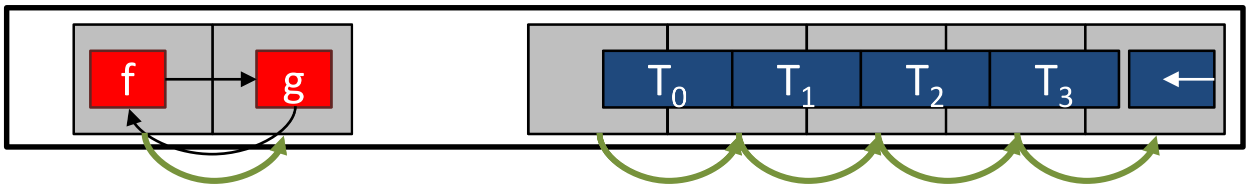
How to prevent solicited exits in sensitive code?

- data / instruction page-faults
- lazy FPU context switch
- power management
- device virtualization

✓ max_tolerable_delay

Solicited Exits

- data / instruction TLB




```
retry:
   d = 1;
   //prepare
   call pg(f)
   call pg(g)
   or $0, [pg(T0)]
   or $0, [pg(T1)]
   or $0, [pg(T2)]
   or $0, [pg(T3)]
   if (p = 1)
         goto retry
```

set p-flag on instruction / data pagefault

Recall: cross-CPU page-table changes require IPIs to shootdown TLBs

Solicited Exits

retry:
xApic.d = 1;

prepare
if (p = 1)
goto retry
R_5 = read $T_i[x_j]$
R_0 = xor R_0, R_5
…
read $T_i[0]$
…
read $T_i[n]$

xApic.d = 0
// if (xApic.p = 1) -> AEX

max_tolerable_delay

How to prevent solicited exits in sensitive code?

- data / instruction page-faults
- lazy FPU context switch
- power management
- device virtualization

access pages / FPU;
check p-flag

report power state

access device MMIO / ports;
check p-flag

check
max_tolerable_delay >
WCET(prepare + sensitive)
of current power state

Concurrency Bugs

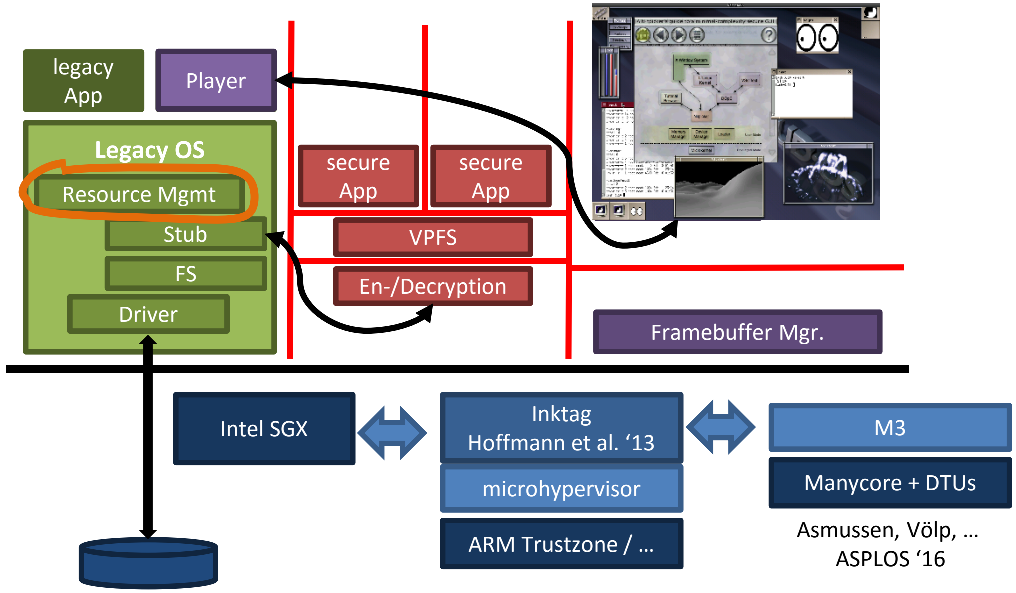
- Cannot fix concurrency bugs by delaying preemptions
- Avoid widening the window of vulnerability

```
disable preemptions              disable preemptions
free object                 ||   if (pointer)
invalidate pointer                   use object
enable preemptions               enable preemptions
```

This talk in one slide…

SnT
securityandtrust.lu
CRITIX

intransitive trust: enabler for TCB reduction



CritiX Lab (Critical and Extreme Security and Dependability)

Interdisciplinary Centre for Security, Reliability and Trust - University of Luxembourg

PEARL Grant FNR/P14/8149128 – Paulo Esteves-Veríssimo

We are hiring bright post-docs and research associates!

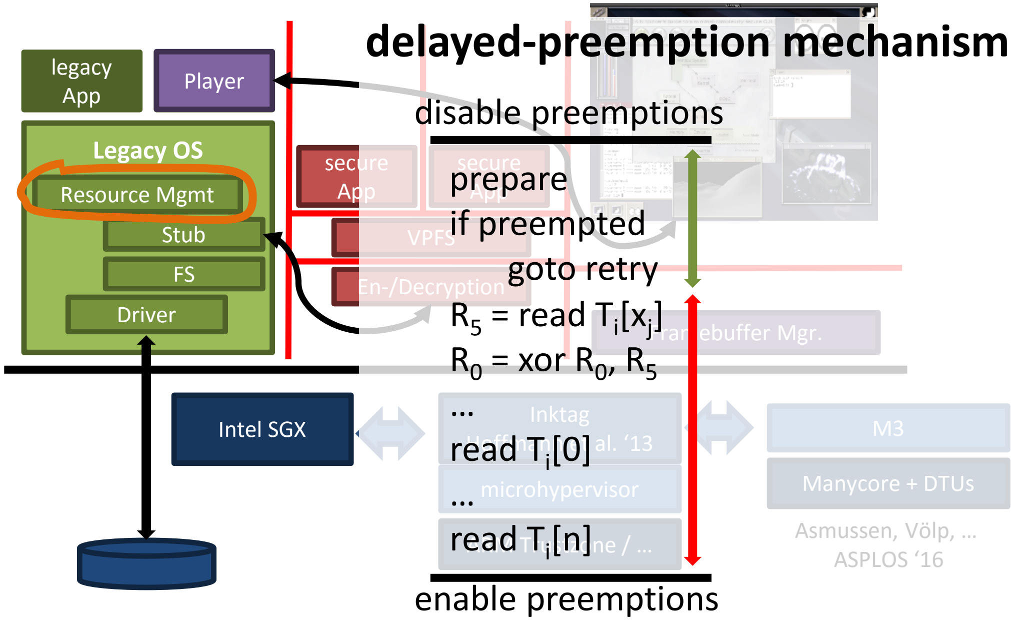
uni.lu UNIVERSITÉ DU LUXEMBOURG

This talk in one slide…

SnT securityandtrust.lu CritiX

intransitive trust: enabler for TCB reduction

delayed-preemption mechanism prevents widening attack windows



CritiX Lab (Critical and Extreme Security and Dependability)

Interdisciplinary Centre for Security, Reliability and Trust - University of Luxembourg

PEARL Grant FNR/P14/8149128 – Paulo Esteves-Veríssimo

We are hiring bright post-docs and research associates!

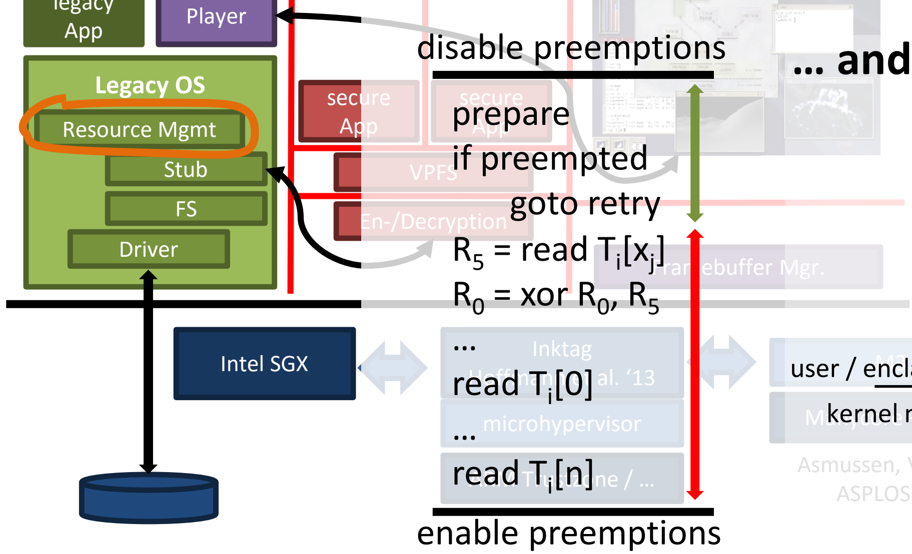
uni.lu UNIVERSITÉ DU LUXEMBOURG

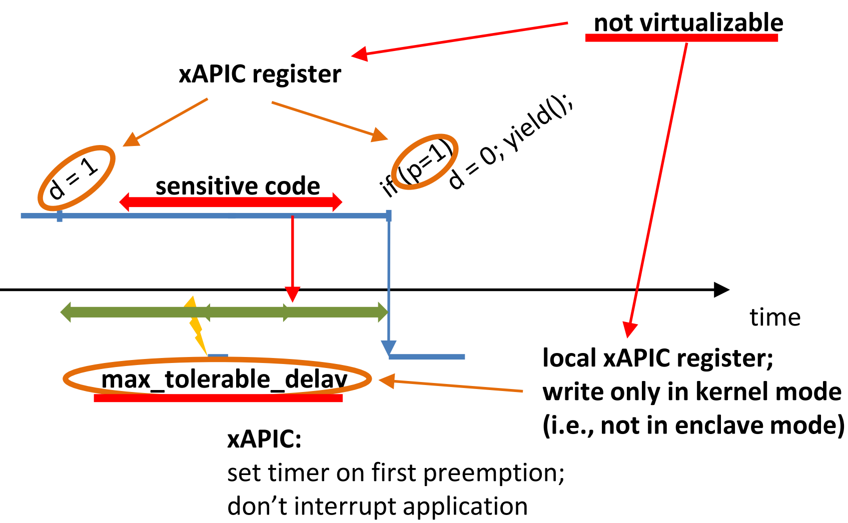
This talk in one slide…

intransitive trust: enabler for TCB reduction

delayed-preemption mechanism prevents widening attack windows

… and it can be integrated in SGX





CritiX Lab (Critical and Extreme Security and Dependability)

Interdisciplinary Centre for Security, Reliability and Trust - University of Luxembourg

PEARL Grant FNR/P14/8149128 – Paulo Esteves-Veríssimo

We are hiring bright post-docs and research associates!