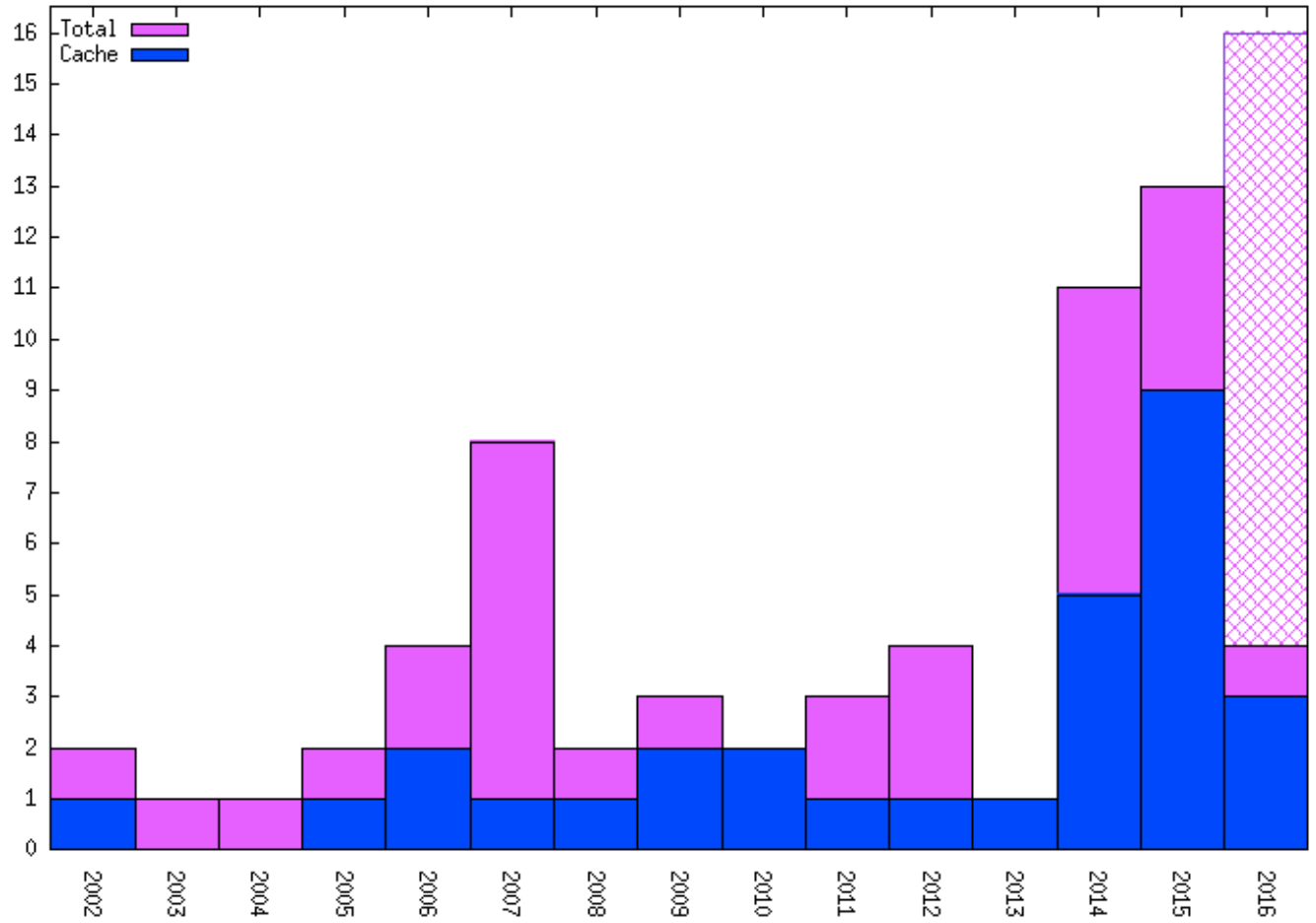
# Microarchitectural Side-Channel Attacks

Yuval Yarom

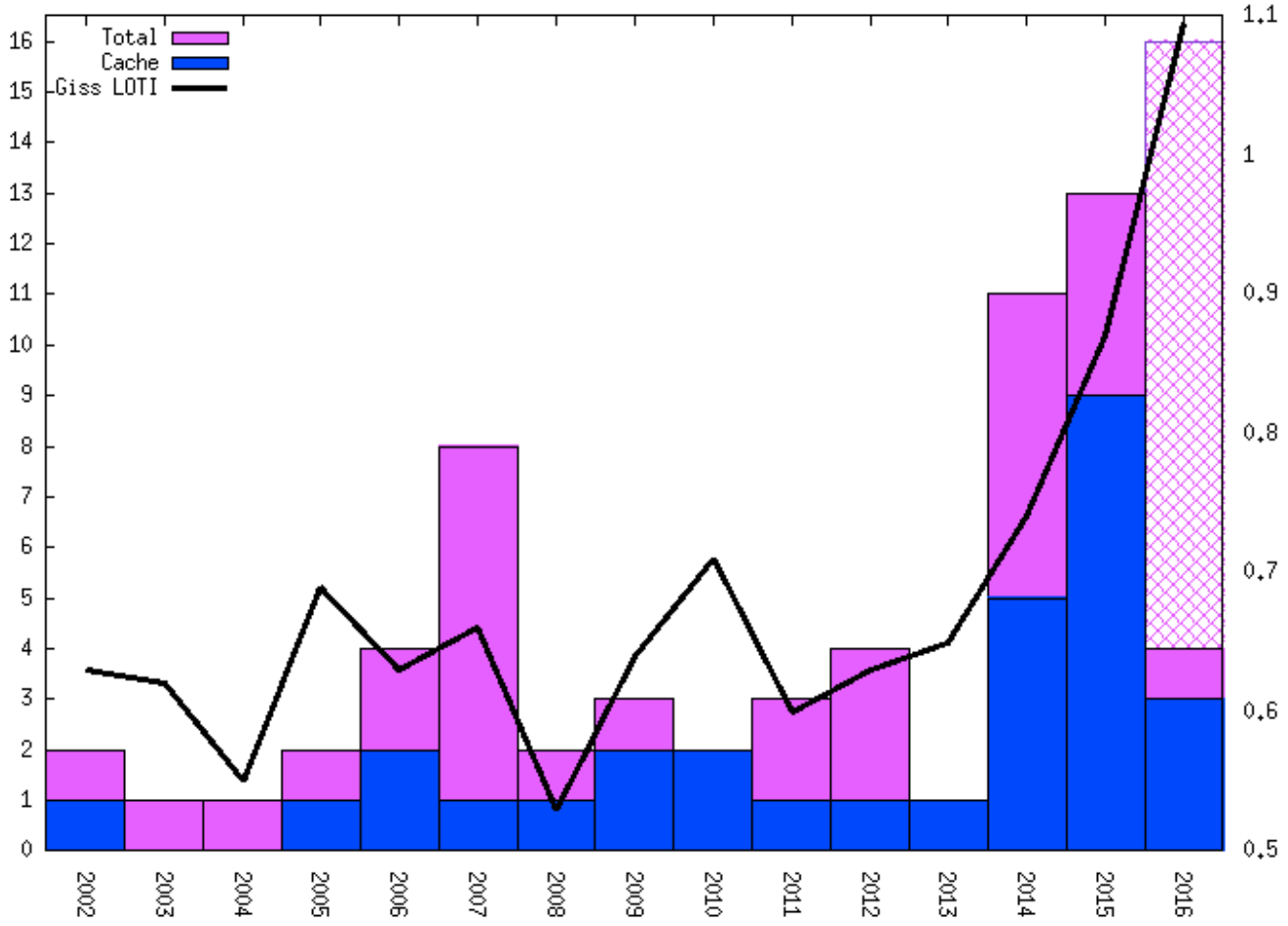
The University of Adelaide and Data61

# Publications on microarchitectural timing attacks




Data from [GYCH16]

# Publications and Global Temperature



# Global Average Temperatures Vs. Number of Pirates

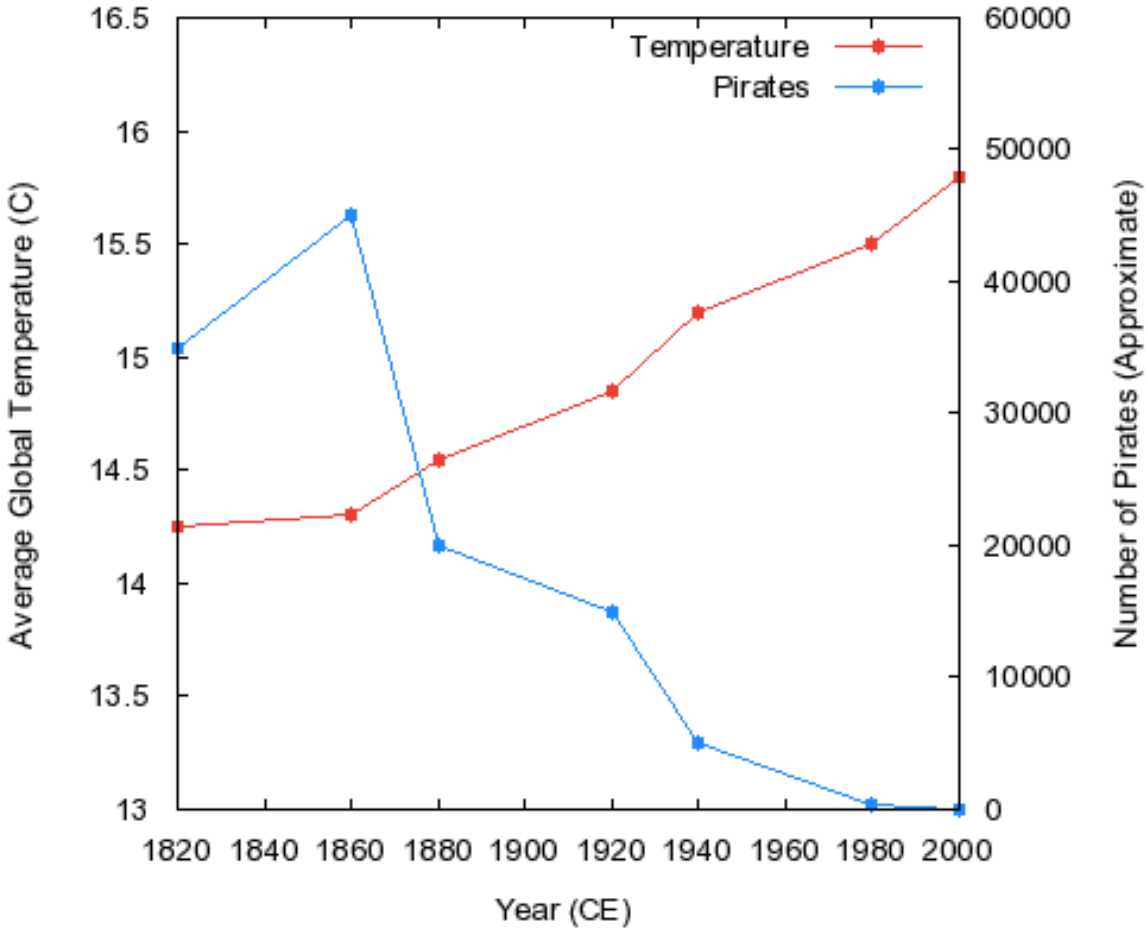
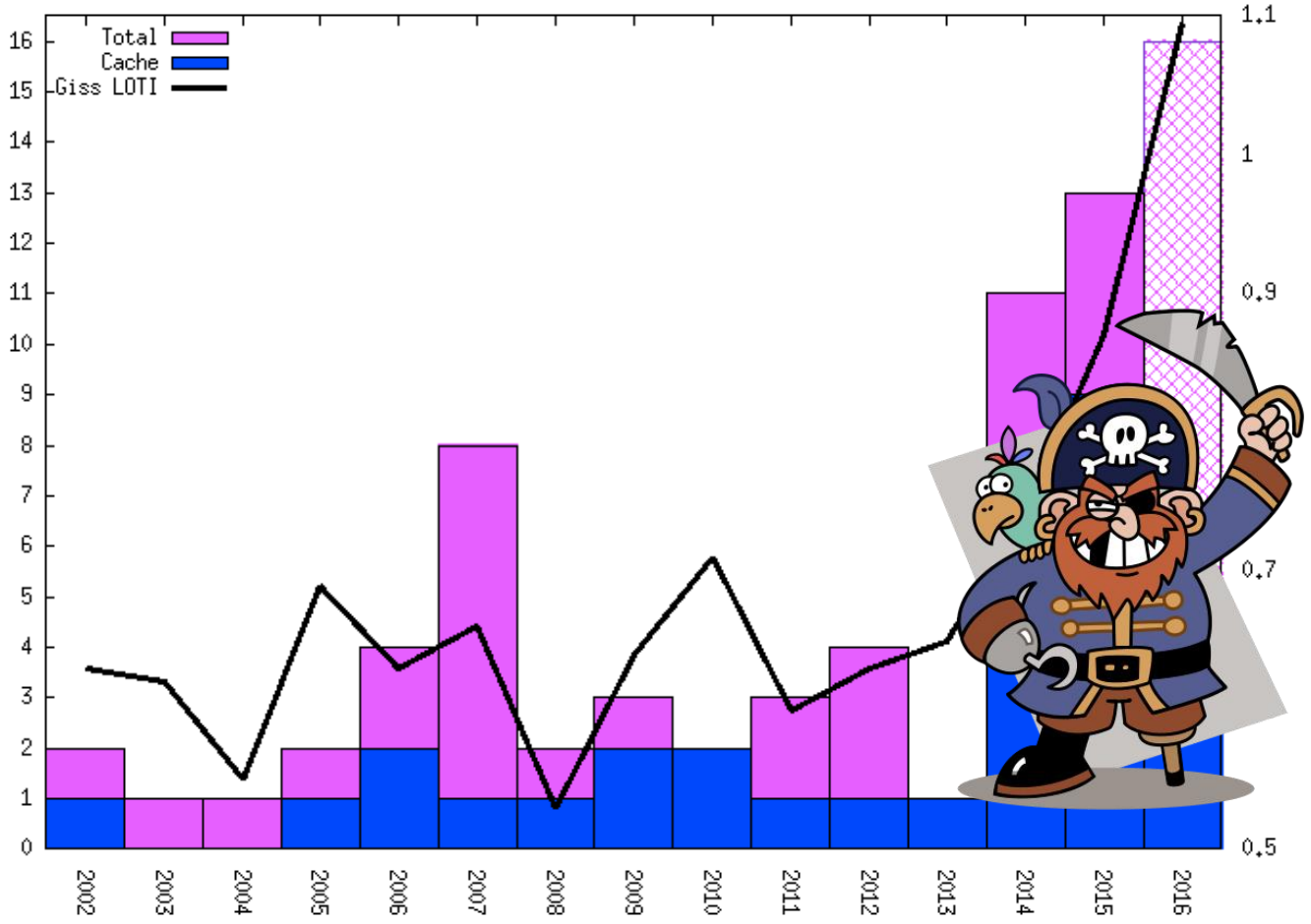



Image from :http://sparrowism.soc.srcf.net/home/pirates.html

# Pirates and publications on microarchitectural timing attacks




Pirate image By J.J. at the English language Wikipedia,
CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=665628

# Still considered hard

- OpenSSL

  *LOW Severity*. This includes issues such as those that … or hard to exploit timing (side channel) attacks.

  https://www.openssl.org/policies/secpolicy.html

# Still considered hard

- OpenSSL

  *LOW Severity*. This includes issues such as those that … or hard to exploit timing (side channel) attacks.

  https://www.openssl.org/policies/secpolicy.html

- Attacks are easy, but at the same time
  - Publications are terse – technical details are often omitted
  - Generic tools do not exist

# Motivation

- Reduce barriers to entry
- Why?
  - Offensive research
    - A potential leak is nice. An exploit is better
  - Cipher development
    - Know your enemy
- How?
  - Education – this tutorial
  - Tools – Mastik

# Mastik

- Extremely bad acronym for
  Micro-Architectural Side-channel ToolKit
- Aims
  - Collate information on SC attacks
    - Improve our understanding of the domain
    - Provide somewhat-robust implementations of all known SC attack techniques for every architecture
    - Implementation of generic analysis techniques
  - Overcome the barrier to entry into the area
  - Shift focus to cryptanalysis

# Mastik - Status

- Reasonably solid implementation of four attacks
  - Prime+Probe on L1-D, L1-I and L3, Flush+Reload
- Only Intel x86-64, on Linux and Mac
  - x86-32 and limited ARM currently working in the lab
- Zero documentation, little testing
- No user feedback

# Outline

- Background on and a taxonomy of microarchitectural side-channel attacks
- The Flush+Reload attack and variants
- The Prime+Probe attack
- Countermeasures

- Slides and sources available at http://cs.adelaide.edu.au/~yval/CHES16/

# The Microarchitecture

- An (*Instruction Set*) *Architecture* (ISA) can have multiple implementations
  - A *microarchitecture* is one such implementation
  - The ISA abstracts the implementation detail

# The Microarchitecture

Software

ISA

Hardware

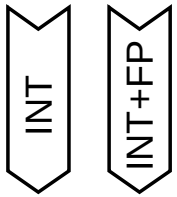





Image adapted from [GYCH16]

# The Microarchitecture

- An (*Instruction Set*) *Architecture* (ISA) can have multiple implementations
  - A *microarchitecture* is one such implementation
  - The ISA abstracts the implementation detail
- The microarchitecture is *functionally* transparent
- But contains hidden state that can be observed through program execution timing

# Microarchitectural attacks

- Create contention on microarchitectural components
- Which results in timing variations
- That are used to expose an intarnal state
- Which depends on secret data
- Allowing the attacker to infer said data

# Attack taxonomy - level

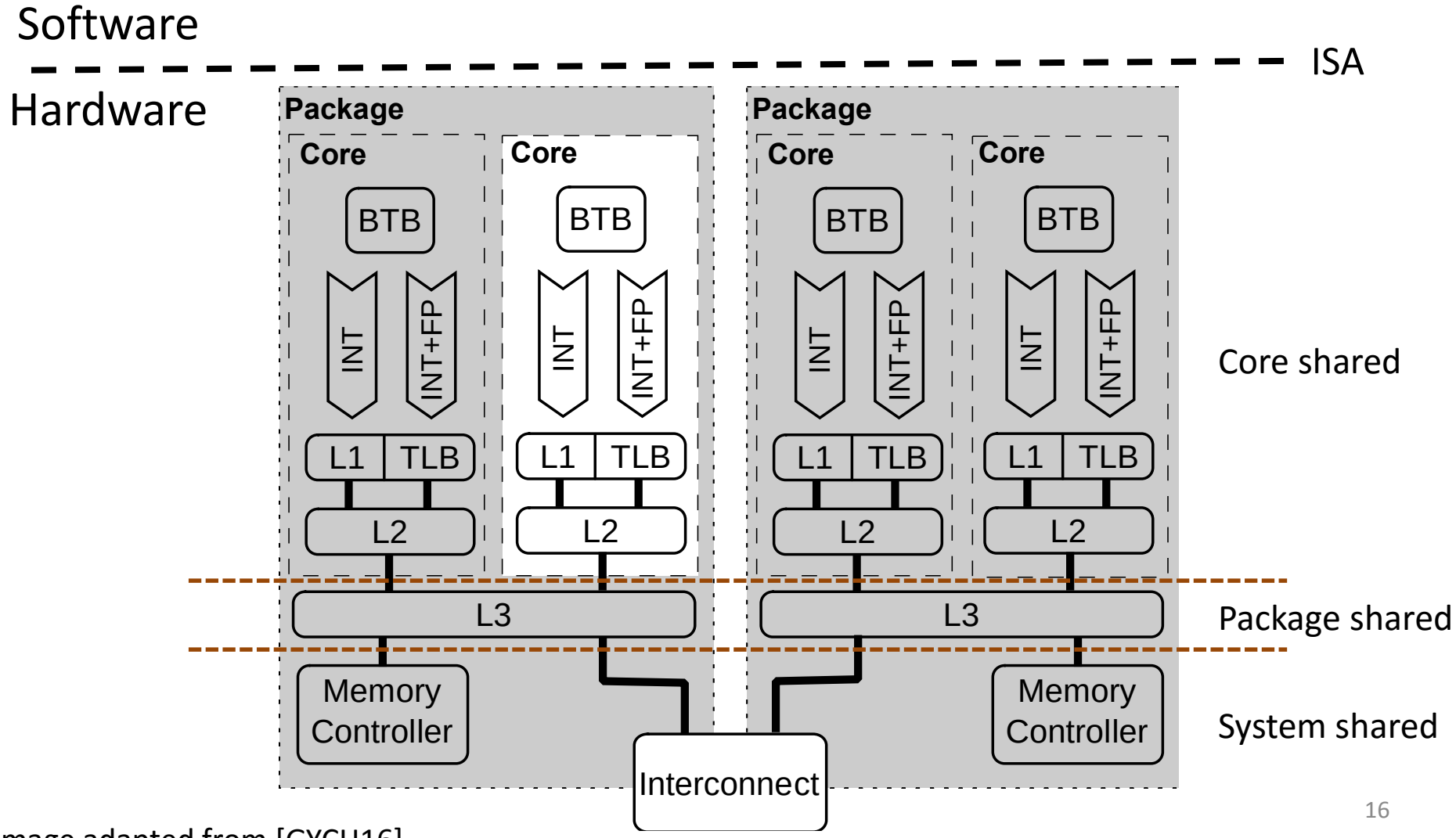



Image adapted from [GYCH16]

# Attack taxonomy – type [AS07]

- Persistent-state attacks
  - Spatial contention on limited storage space
  - Example: most cache attacks
- Transient-state attacks
  - Temporal contention on limited processing speed
  - Example: CacheBleed

# Basic taxonomy

| | Persistent-state | Transient-state |
|---|---|---|
| Core level | [Ber05,Per05, OST06] Others | [Lam73,Ber05, AS07,YGH16] |
| Package | [YF14,LYG+15,IES15] | ? |
| System | [PGM+16,IES16] | [WS12,WXW12] |

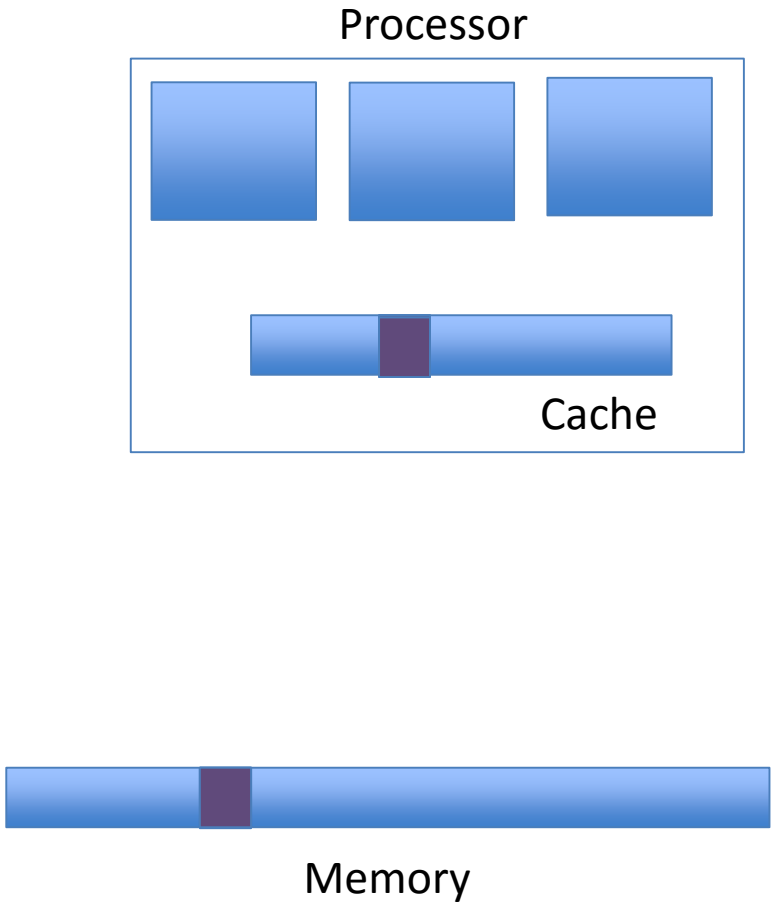
- We focus on persistent-state at the package level (with some core)

# Other classifications

- Classical taxonomy [Pag03], [NS06]
  - Time-driven – measure complete execution time
  - Trace-driven – capture sequence of cache hits/misses
  - Access-driven – obtain some information on accessed memory addresses
- Internal vs. external contention [AK09]
- Degree of concurrency [GYCH16]
  - Multicore
  - Hyperthreading
  - Time slicing

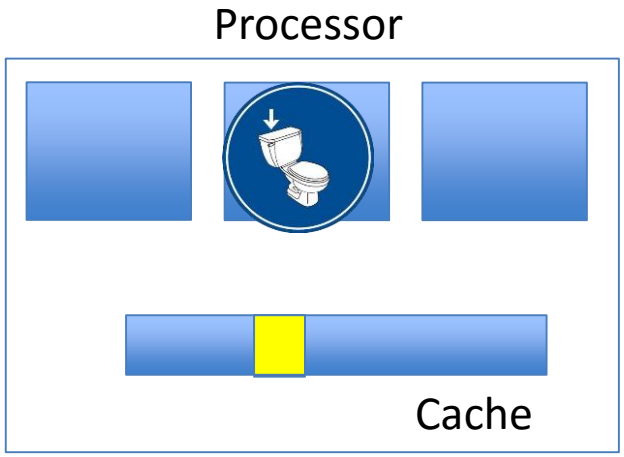
# The (X86) Cache

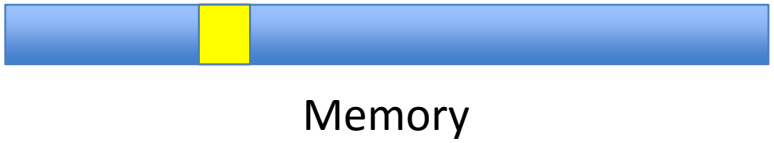
- Memory is slower than the processor
- The cache utilises locality to bridge the gap
  - Divides memory into *lines*
  - Stores recently used lines
- Shared caches improve performance for multi-core processors



# Cache Consistency

- Memory and cache can be in inconsistent states
  - Rare, but possible
- Solution: Flushing the cache contents
  - Ensures that the next load is served from the memory



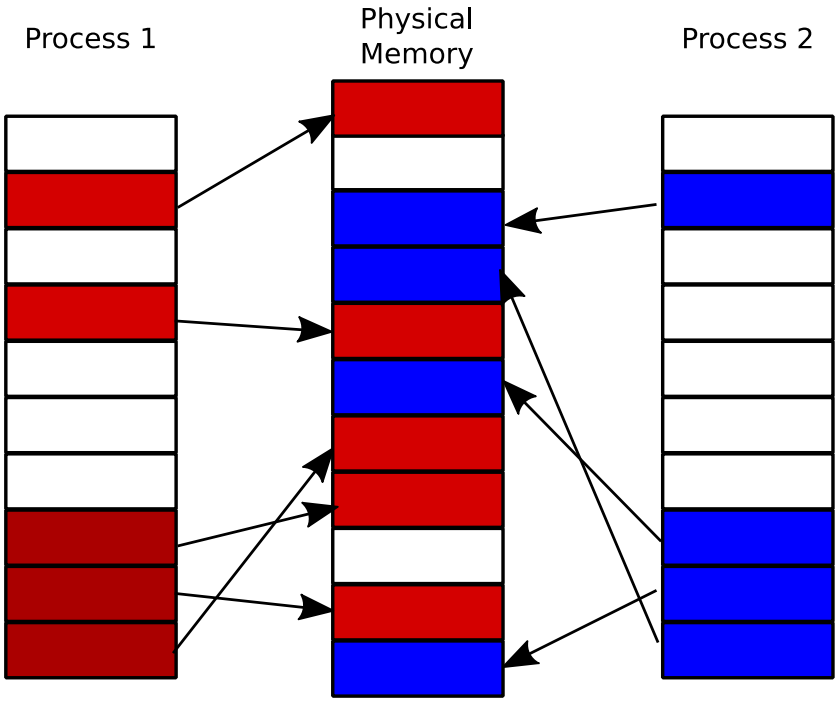



# The FLUSH+RELOAD Technique

- Exploits cache behaviour to leak information on victim access to shared memory.
- Spy monitors victim's access to shared code
  - Spy can determine what victim does
  - Spy can infer the data the victim operates on

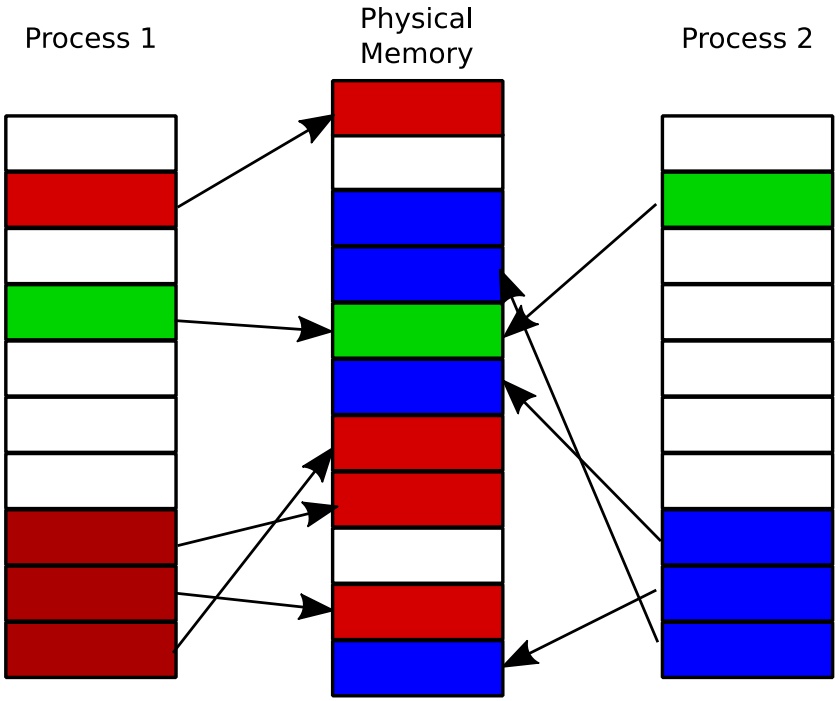
# Detour - Virtual Memory

- Processes execute within a virtual address space
  - Virtual pages map to physical frames



# Sharing

- Frames can be shared by multiple processes
  - Read only sharing maintains *functional* isolation
  - Protection using *Copy-on-write*

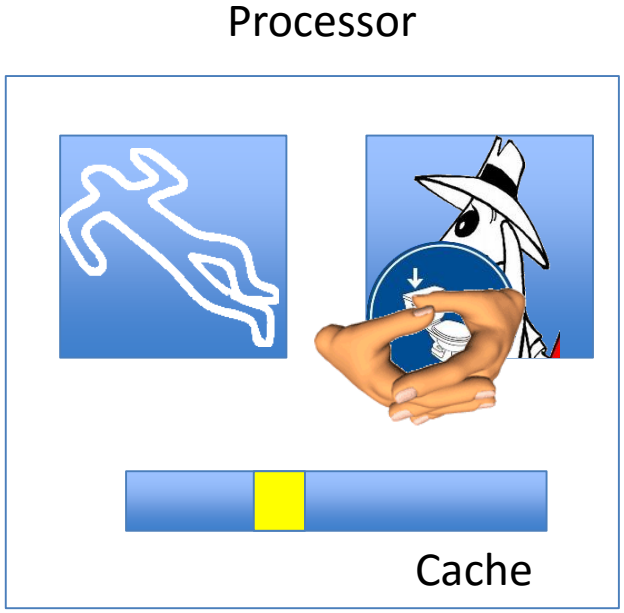


# Causes of sharing

- Content-aware sharing
  - Pages from the same file have identical content
  - Shared program or library code
    - Can also share constant data
  - Shared images in PaaS clouds
- Content-based sharing (a.k.a. page deduplication)
  - The system identifies and coalesces identical pages
  - Implemented in many hypervisors and in most modern operating systems

# FLUSH+RELOAD [GBK11,YF14]

- **FLUSH** memory line
- Wait a bit
- Measure time to **RELOAD** line
  - slow-> no access
  - fast-> access
- Repeat




# Flush+Reload code

```
mfence
rdtscp
mov %eax, %esi
mov (%ebx), %eax
rdtscp
sub %esi, %eax
clflush 0(%ebx)
```

- Also need:
  - Wait
  - Data collection
  - Noise handling
  - Initial parsing

# Demo

- FR-1-file-access
- FR-2-file-access

- FR-threshold

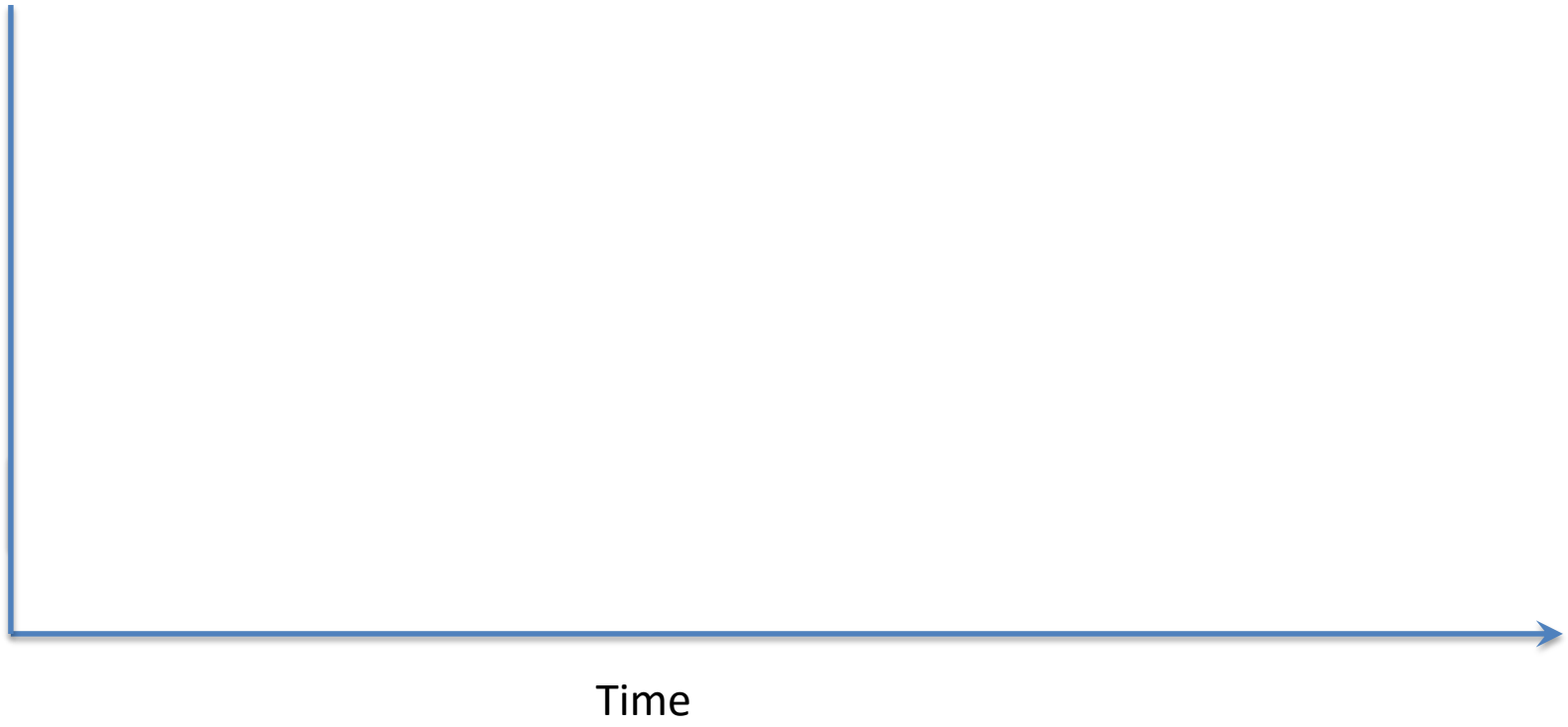
# Finding code

- Use `nm`, `gdb`, `objdump`, etc.
  - Demo scripts `functiondump.sh` and `debuginfo.sh`
- Remember the base address

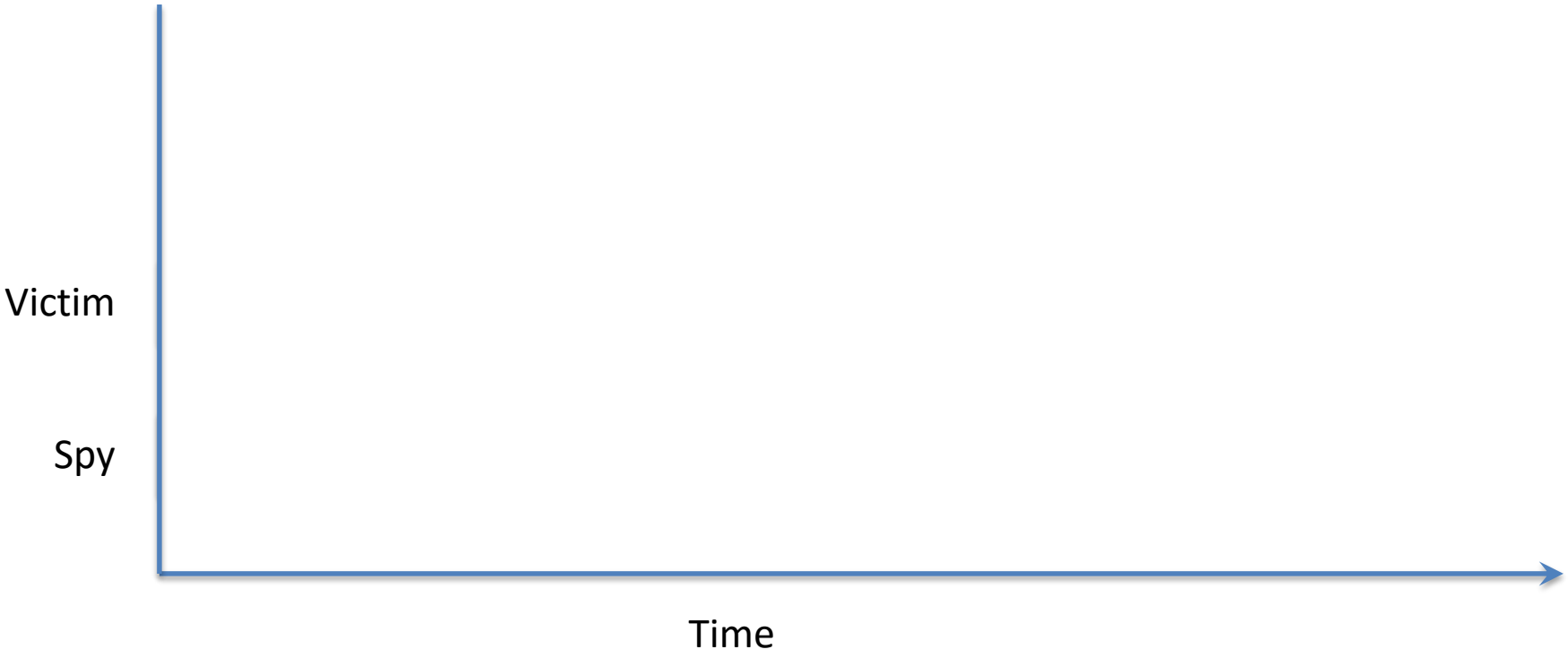
# Demo

- FR-function-call

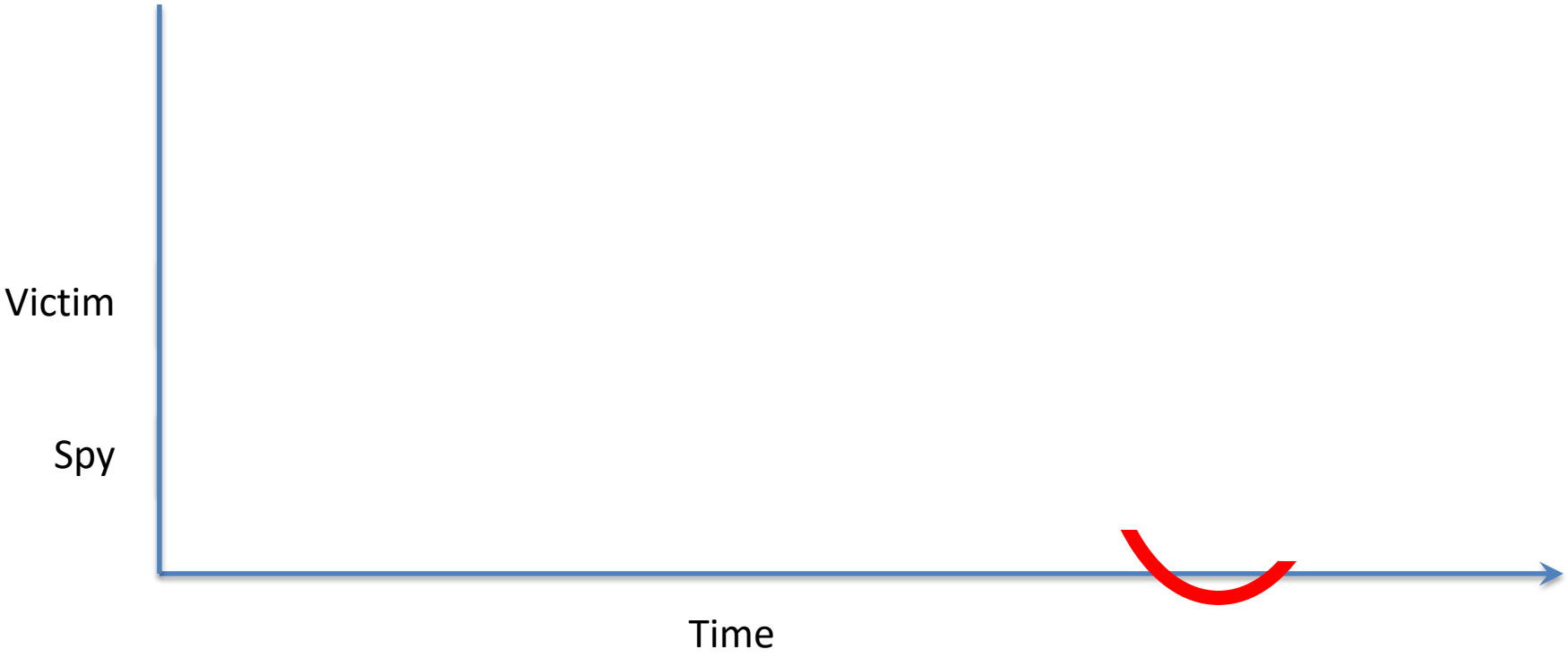
# A closer look at F+R



# A closer look at F+R



# Timing matters



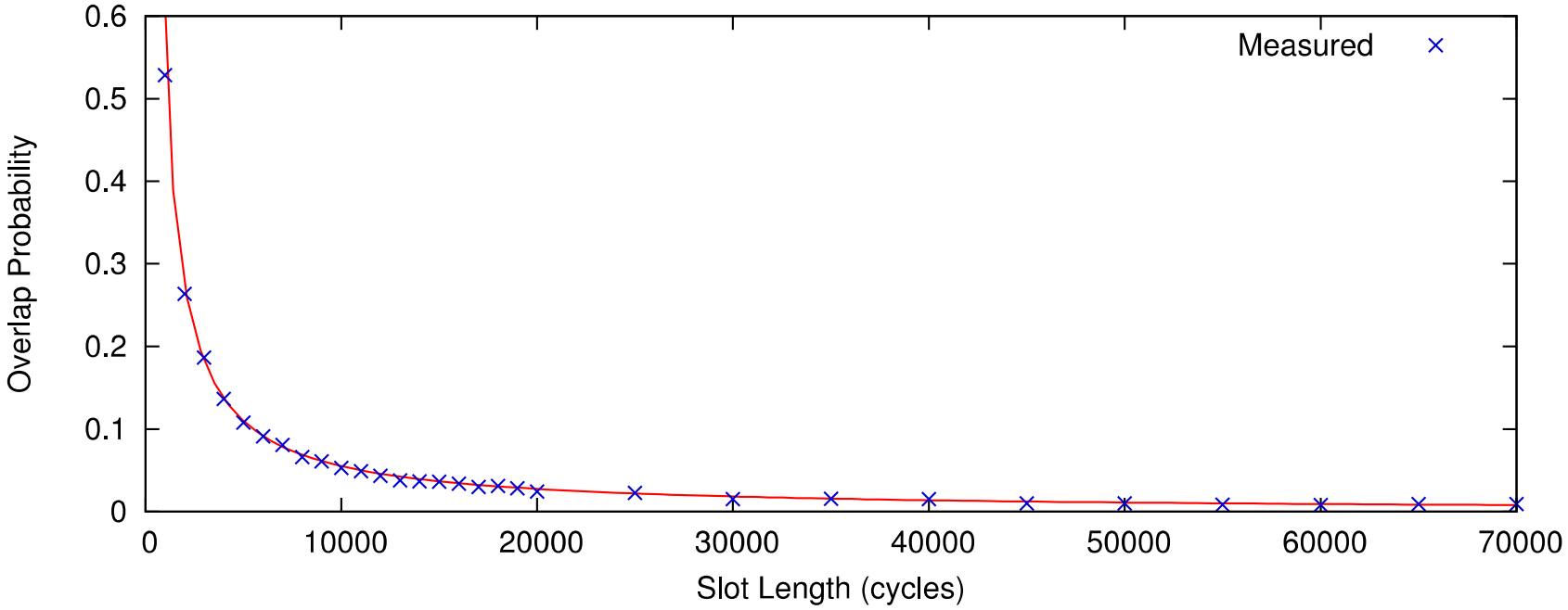

Access missed due to temporal overlap

# Demo

- FR-function-call-nodelay

# Probability of a probe miss [ABF+15]

- Ratio of wait time to slot length



# Handling misses

- Probe function calls [YB14]
  - A cache line containing a function call is accessed once before the call and once on return
    - Except, maybe, when the return address is in the next cache line
  - The timings of the two accesses are not independent
- Probe loops [YF14]

# GnuPG 1.4.13 Modular Exponentiation

```
x ⟵1
for i ⟵|e|-1 downto 0 do
    x ⟵x^2 mod n
    if (e_i =1) then
        x = xb mod n
    endif
done
return x
```

Example:

$11^5 \bmod 100 =$

$161{,}051 \bmod 100 = 51$

| Operation | res | i | $e_i$ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**The secret exponent is encoded in the sequence of operations !!!**

# Demo

- FR-gnupg-1.4.13

# F+R Spatial Resolution

- Cache line
  - Can't have two different probes on the same cache line
- Cache line pairs
  - Probes on paired lines interfere with each other –Don't
- Streaming
  - Use "random" order when probing multiple lines in the same page
  - Don't probe too many of those
- Speculative execution
  - Probe rear end of functions

# Improving temporal resolution

- Scheduler trick [GBK11]
  - Monitoring each and every memory access
  - Requires hyperthreadings
  - Uses hundreds of threads
- Amplification [ABF+15]
  - Flushing commonly used cache lines slows the victim

# Demo

- FR-flush

# Flush+Reload Summary

- Simple attack
  - Even without Mastik
- High temporal resolution
  - Up to sub-micro-second
- High accuracy
  - Few false positives
    - Increases as a function of spatial granularity
  - A bit more false negatives
    - Particularly when exploiting the high temporal resolution

# Variants

- Evict+Reload [GSM15]
  - Uses cache contention instead of `clflush`
- Flush+Flush [GMWM16]
  - Measures variations in `clflush` timing between cached and non-cached data
  - Improved temporal and spatial resolution
  - Increased error rate
- Invalidate+Transfer [IES16]
  - Use Flush+Reload on Intel or Evict+Reload on ARM to implement a cross-package attack
- Flush+Reload on ARM [ZXZ16]
  - ???