# *Cheap Hardware Parallelism Implies Cheap Security*

Onur Acıiçmez and Jean-Pierre Seifert

- Introduction

- Related and Previous Work

- Multi-threading Basics

- Exploiting Shared Functional Units

- Comparison to Other MicroArchitectural Cryptanalysis Techniques

- Conclusions

Side channel attacks are methods by which an attacker can extract secret information by exploiting some real-world's implementation issues of a specific cryptographic algorithm.

Recently, the Side-Channel attack arena hit the PC as a new victim platform:

- Especially interesting since maturing Trusted Computing efforts promise a "trusted environment" with isolated execution for applications, etc.

This new Side-Channel attack arena is different from embedded security market

- due to the PC platform environment, which is quite different from the embedded security market.
- Only pure "unpriviliged" software-based attacks are really interesting.

MicroArchitectural Side-Channel attacks are a special new class of attacks that exploit the microarchitectural and throughput-oriented internal functionality of modern processor components.

These attacks capitalize on the situations where several applications share the same processor resources, and the shared usage between spy and crypto process allows a spy process

- **running in parallel to the victim process**

to extract critical information like secret keys.

On powerful PC-platforms many applications can run in para

- Either quasi-parallel enabled by OS scheduling, or
- More or less explicitly parallel depending on the degree of additional hardwa

Thus, several applications share the same processor and its resources, and also at more or less the same time.

Therefore, when a highly critical crypto algorithm is executed there is the potential threat that a malicious or so called spy process is executed in parallel with the crypto process which might try to extract critical or secret information by "spying" on the crypto process during its execution.

…

Hu 1992] – Covert channels by caches

Trostle 1998] – Cache attack against trusted keyboard input

…

Page 2002] – Theoretical cache attacks via power trace

Tsunoo Tsujihara Minematsu Miyuachi 2002],
Tsunoo Saito Suzaki Shigeri Miyauchi 2003] – Timing attacks via internal collisio

…

Bernstein 2004] – Pure timing attack on AES

Percival 2005] – Cache attack on RSA

Tromer Shamir Osvik 2005/2006] – First work which fully demonstrated an effici
cache attack on AES in a real-life setting

Neve Seifert 2006] – Improvements of Tromer Shamir Osvik AES cache attack

Acıiçmez Koç Schindler 2007] – Remote cache attack on AES

Acıiçmez Koç Seifert 2006/2007] – Branch Prediction Attacks

Acıiçmez 2007] – Instruction Cache Attack

Modern superscalar processors can issue several instructions to independent functional units each cycle

The benefit of such superscalar architectures is ultimately limited by the parallelism available in a single thread, i.e., instruction level parallelism (ILP), even in the presence of Out-of-Order execution engines.
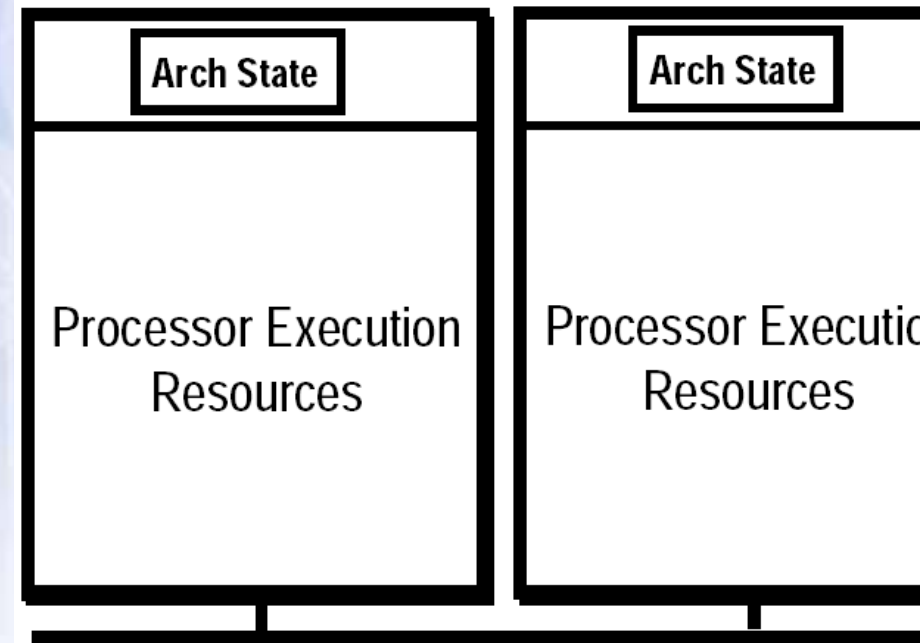


Multi-processing
System Using Two
Superscalar Processors
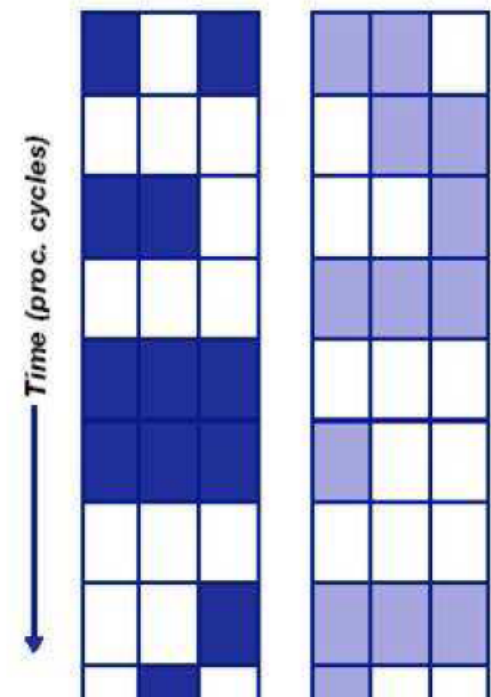
Simultaneous Multi Threading
is a processor design that
combines hardware
multithreading with superscalar
processor technology to allow
multiple threads to issue
instructions each cycle
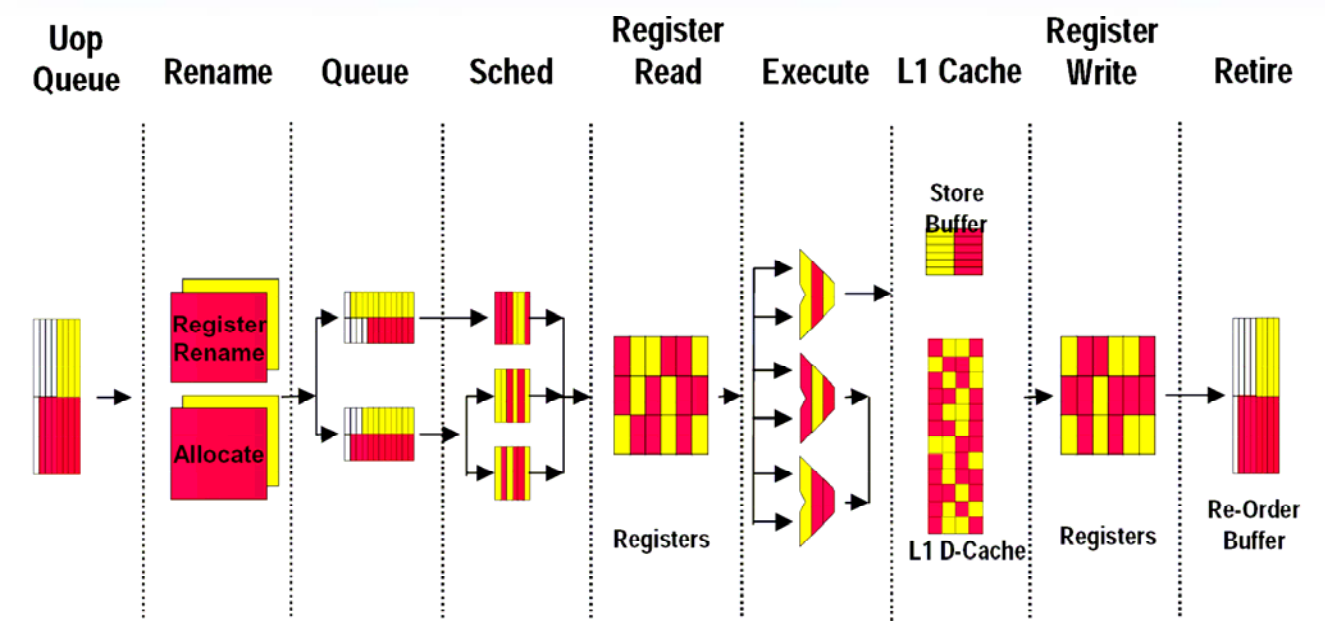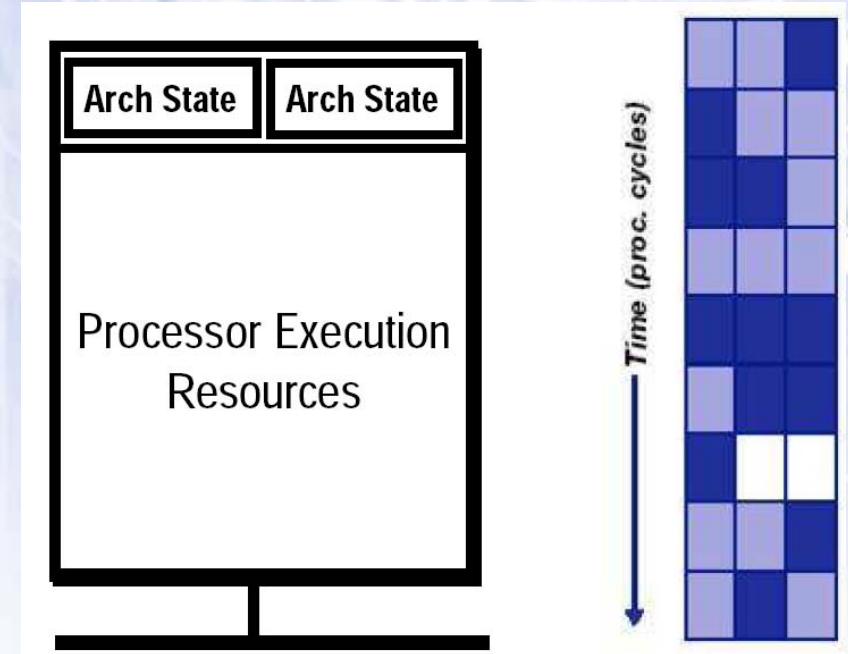




•Simultaneous Multi
Threading interleave
instructions from
different threads in
the CPU pipeline

# Simultaneous Multi Threading

- Two threads are running simultaneously
- They share the CPU resources "*on-the-fly*"
- Instructions from one thread affect the execution of the other thread
  - The execution time statistics of a process heavily depend on the execution of the other thread

# Our observation:

- Pentium-4 HT has a single integer multiplier shared between logical processors
- If both threads issue several multiplication instructions at the same time, they will suffer from a race condition

**A spy process can detect when a cipher process executes multiplications**

# OpenSSL's RSA implementation:

- First performs multi-precision multiplication, then reduces the result
- Has different functions for multi-precision multiplication and square operations (Karatsuba vs. normal multiplication)
- Modular square operation takes less time than multiplication

# The goal of shared FU:

- Execute several multiplication instructions in the spy
- Detect when the cipher executes multi-precision functions and how long these functions take (allows us to distinguish multiplication from square)
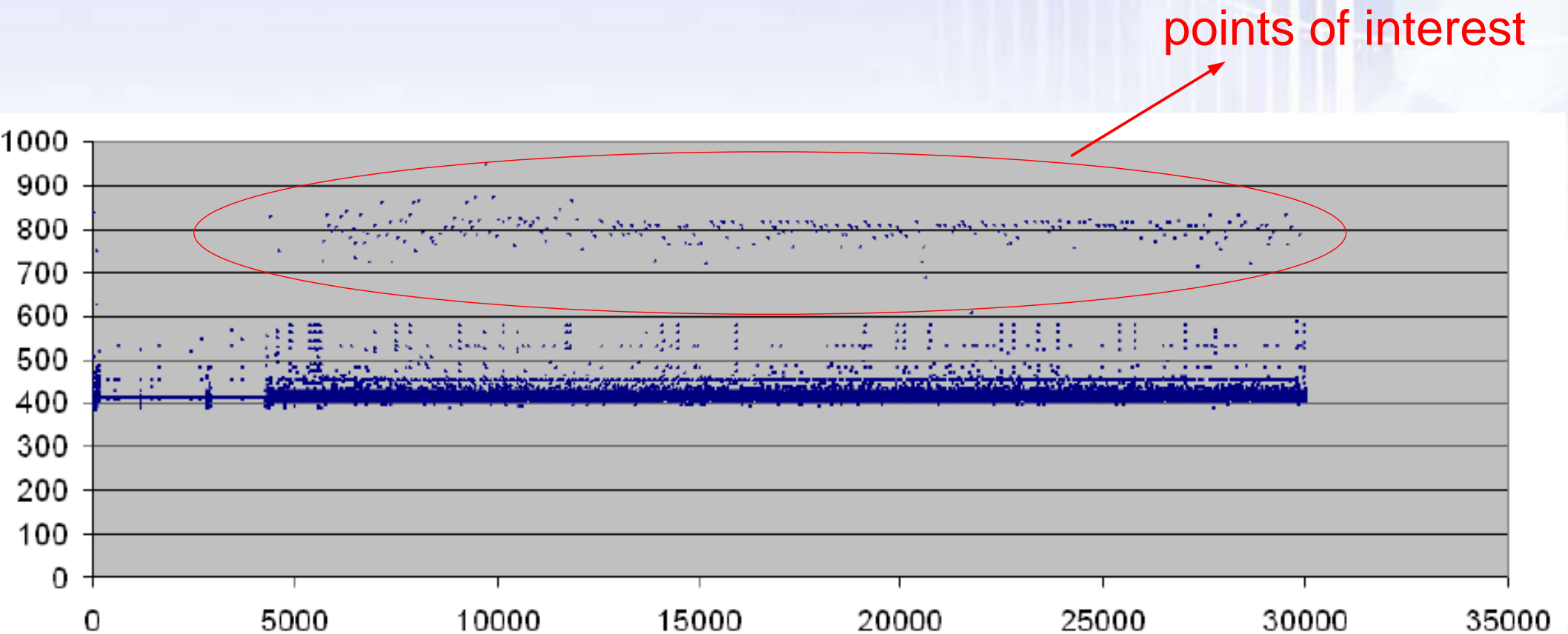
```
crypto thread:                          spy thread:

S = M                                   for many j do
for i from 1 to n − 1 do
    S = S * S (mod N)

                                        start some long integer multiplication

    if d_i = 1 then
        S = S * M (mod N)

                                        t_j ← #clock cycles(long integer multiplication)
```

We tested our idea on Pentium-4 HT by running our spy simultaneously with an RSA process (OpenSSL)

Results:

points of interest

# Now, lets zoom in
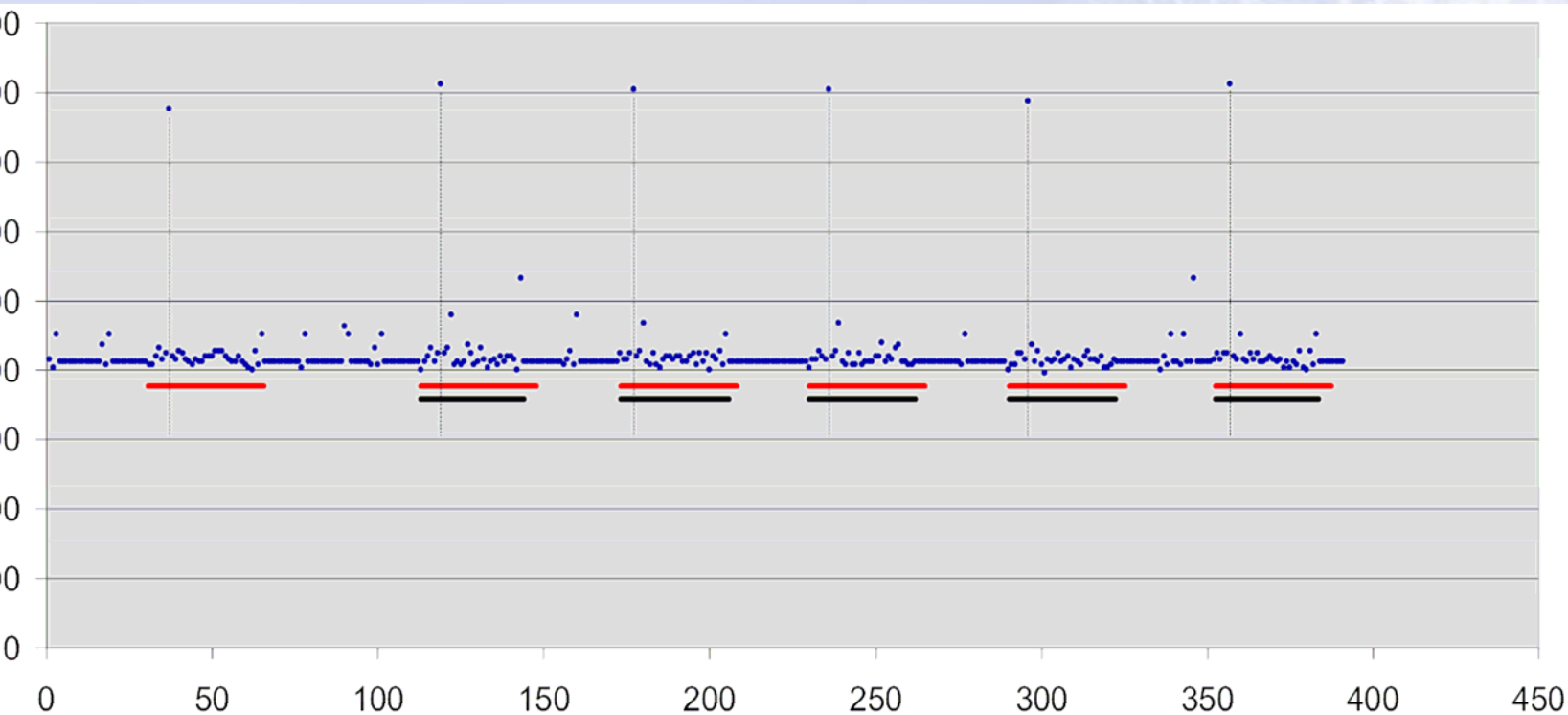


•We inserted artificial dummy for loops after each operation. A long loop after each mult. ar a shorter loop after eac sq.

The high values appear almost all the time at the 7th or 8th measurements after th starting point of each multiplication and square operation within the crypto proces

The **red bar** shows the length of a **multiplication**

The **black bars** shows the length of **square** operations

Conclusion:

–   We could pinpoint the beginning of each RSA operation (mult./sq.)

–   *We could distinguish multiplications from squares due to the timing differenc*

Known MA types:

- Cache Attacks  (Page, Tsunoo et. al., Bernstein, Osvik et. al., Percival, Neve et. al., Acıiçmez et. al. )

- Branch Prediction Analysis (Acıiçmez et. al.)

- Instruction Cache Analysis (Acıiçmez)

All of these attacks exploit **persistent states** of buffers (cache BTB, etc.) and rely on

- either h/w assisted parallelism (e.g., SMT, multi-processors, etc.)

- or s/w assisted parallelism (e.g., OS scheduling on uniprocessors)

*SMT is not a requirement for these attacks*


Shared FU attack does not depend on persistent states

Functional units must be shared between *running* threads

- It can only work on SMT
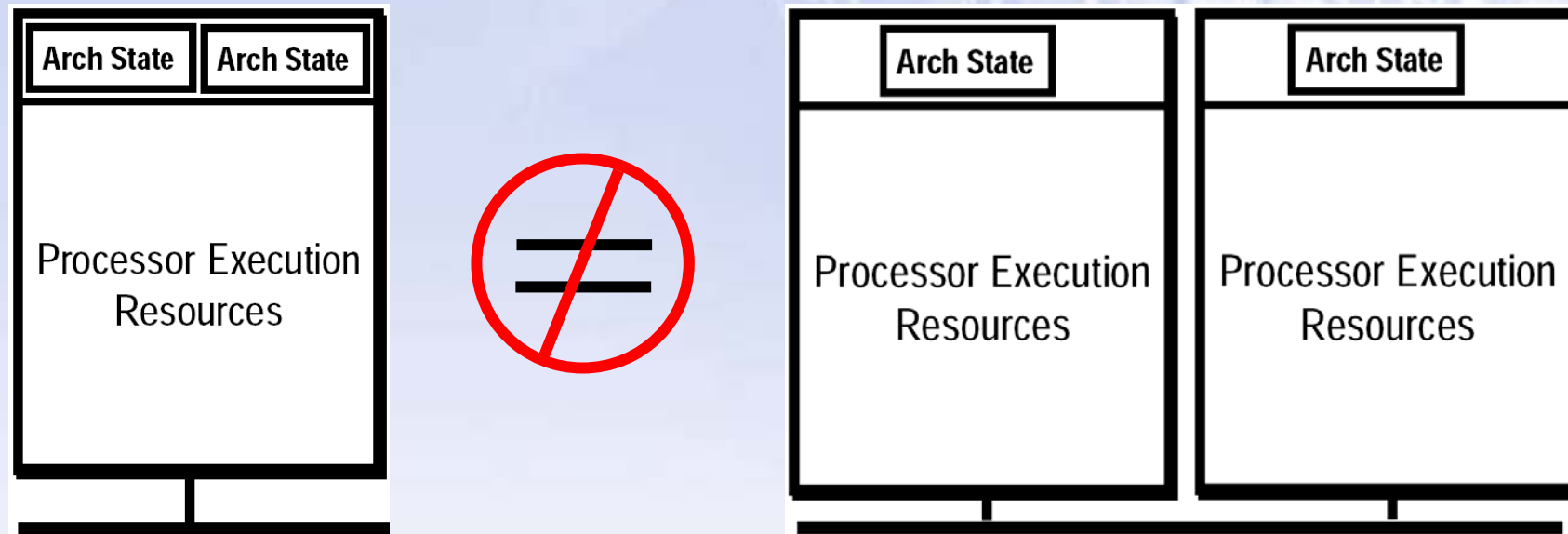
We have introduced a new SMT based attack type that cann
work on non-SMT systems

Our results indicate that:

- Simultaneous Multi-threading has intrinsic and unique security weaknesse
  that do not exist in other designs

- In other words:

# Thank You!

Questions?