Cache Storage Channels: Alias-Driven Attacks

2016-03-30

Roberto Guanciale (KTH)
Mads Dam Hamed Nemati Christoph Dam

2016-03-30

Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses

Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?

Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels

Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels
- precise analysis of timing/power consumption exceedingly difficult
- channels counteracted by model-external means

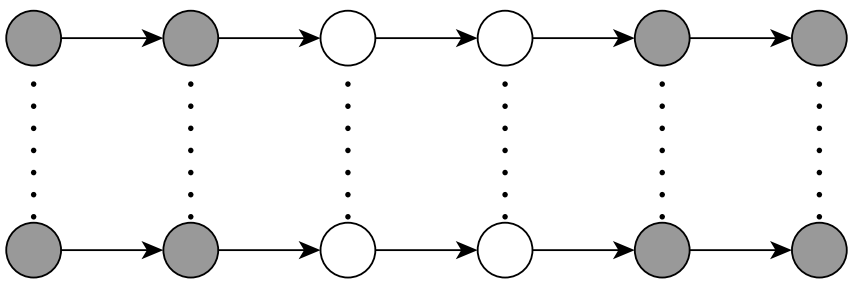
Introduction

- huge strides in formally verified execution platforms
  seL4 - Hyper-V - Integrity - Prosper
- caches are mostly excluded from these analyses
- how much is this a problem?
- timing/power consumption can be exploited to mount side channels
- precise analysis of timing/power consumption exceedingly difficult
- channels counteracted by model-external means
- models should preferably be sound with respect to the features that are reflected

Common verification approach

- If models are sound (wrt the selected features)
- Verification by refinements

Abstract model

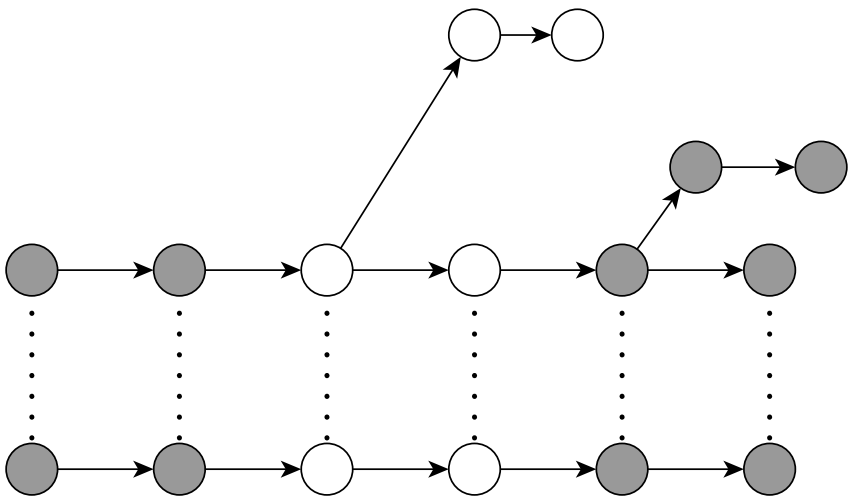


Concrete model

Common verification approach

- If the hidden features (e.g. cache state) affect the model variables (e.g. memory) $\Rightarrow$ Overapproximation

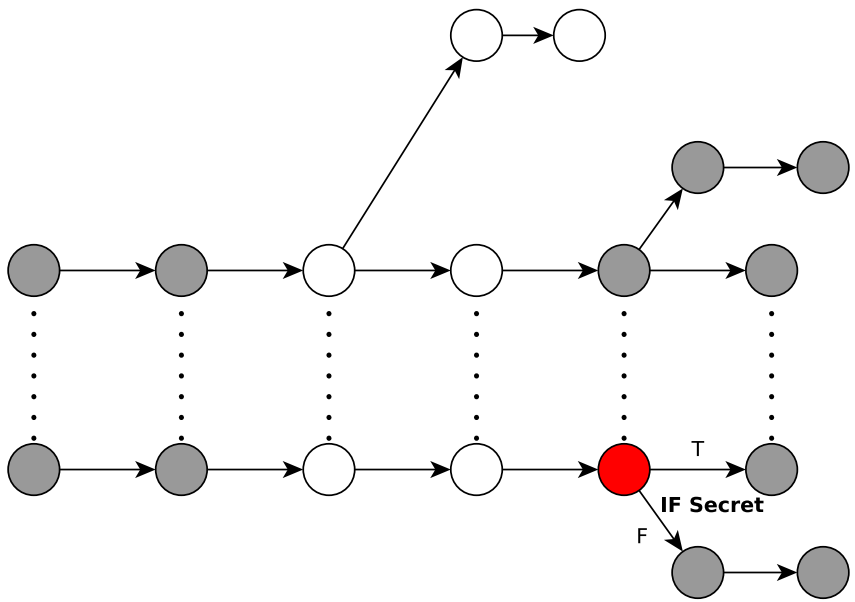
Abstract model



Concrete model

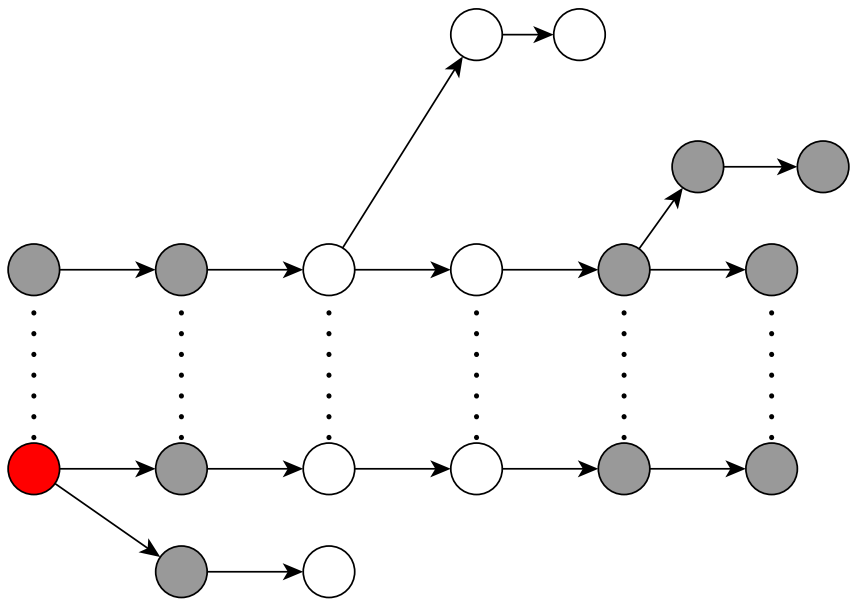
Challenges of Storage channel

- If the hidden features (e.g. cache state) affect the model variables (e.g. memory) $\Rightarrow$ Overapproximation
- Problems for information-flow properties
- The actual HW is deterministic

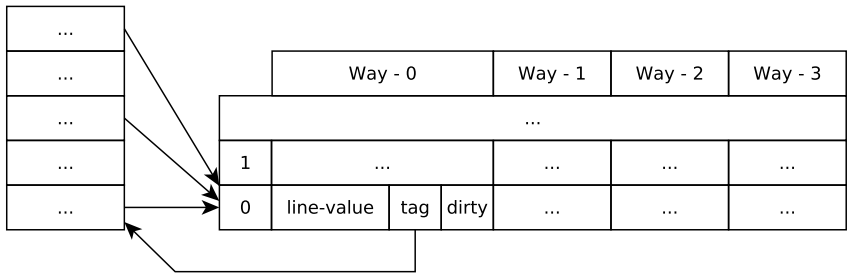


Challenges of Storage channel

- The HW respect the abstract specification is some architectural constraints are met
- What about unknown colocated SW?



Cache today



- MMU allows to configure (via the page tables) the caching policy on a per-page basis
- A processor can use the Harvard arrangement

Incoherent behaviors

- caches should be transparent to program behaviour
- this is usually not the case unless the system configuration satisfies some architecture-specific constraints

Incoherent behaviors

- caches should be transparent to program behaviour
- this is usually not the case unless the system configuration satisfies some architecture-specific constraints
- e.g. memory mapped UART

Incoherent behaviors

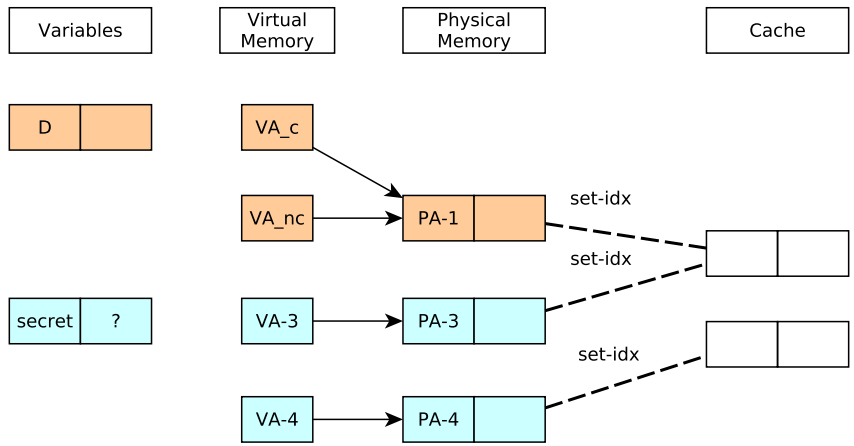
- caches should be transparent to program behaviour
- this is usually not the case unless the system configuration satisfies some architecture-specific constraints
- e.g. memory mapped UART
- Mismatched cacheability attributes
  (ARM-terminology: "unexpected cache hit")
  if the data cache reports a hit on a memory location that is marked as non-cacheable, the cache might access the memory disregarding such hit.

Incoherent behaviors

- caches should be transparent to program behaviour
- this is usually not the case unless the system configuration satisfies some architecture-specific constraints
- e.g. memory mapped UART
- Mismatched cacheability attributes
  (ARM-terminology: "unexpected cache hit")
  if the data cache reports a hit on a memory location that is marked as non-cacheable, the cache might access the memory disregarding such hit.
- Self-modifying code;
  even if the executable code is updated, the processor might execute the old version of it if this has been stored in the instruction cache.

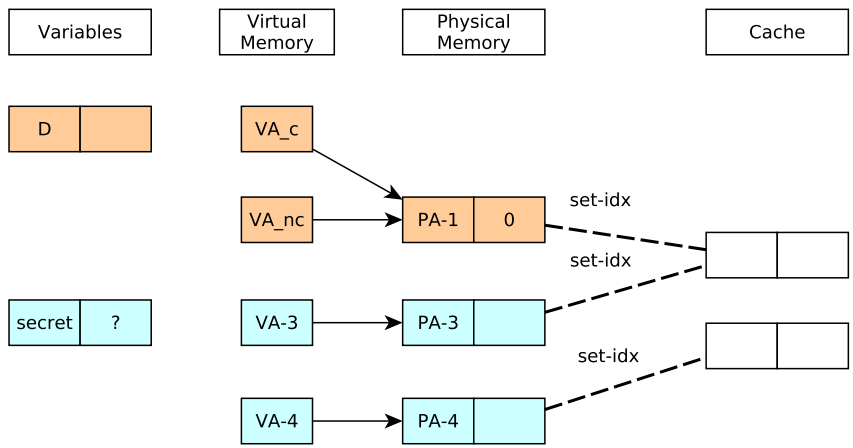
Attacking confidentiality using data-caches

```
* A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



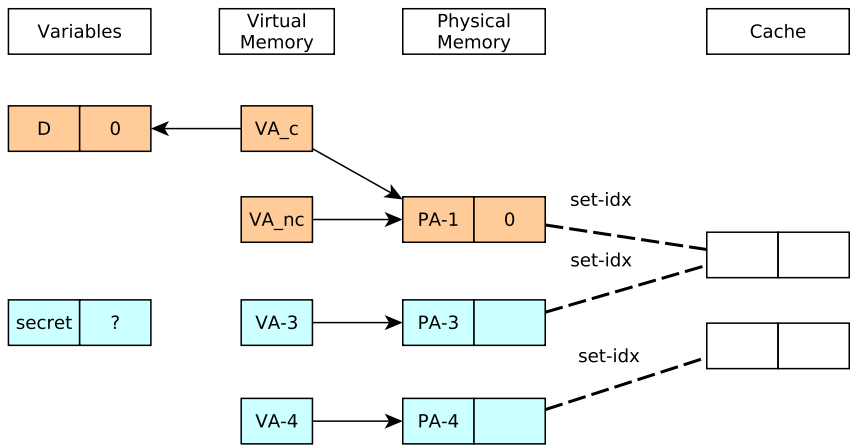
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
* A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



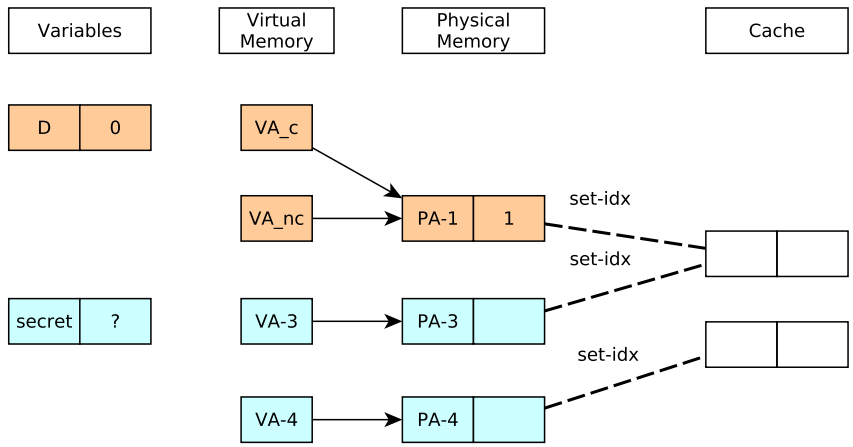
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
* A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



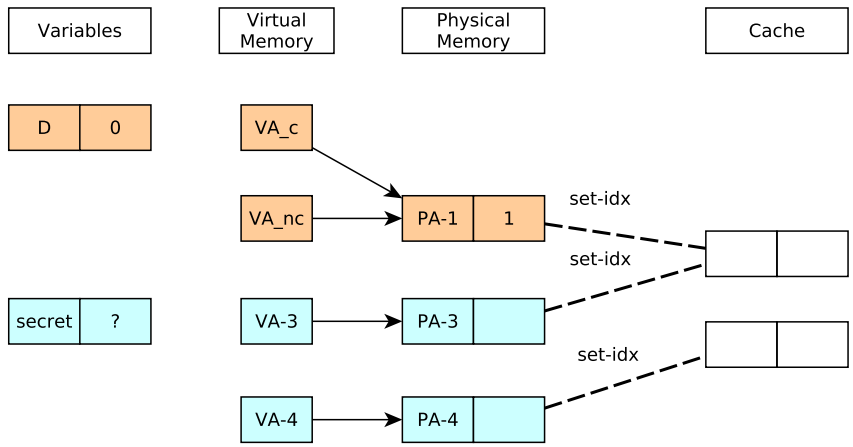
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
* A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



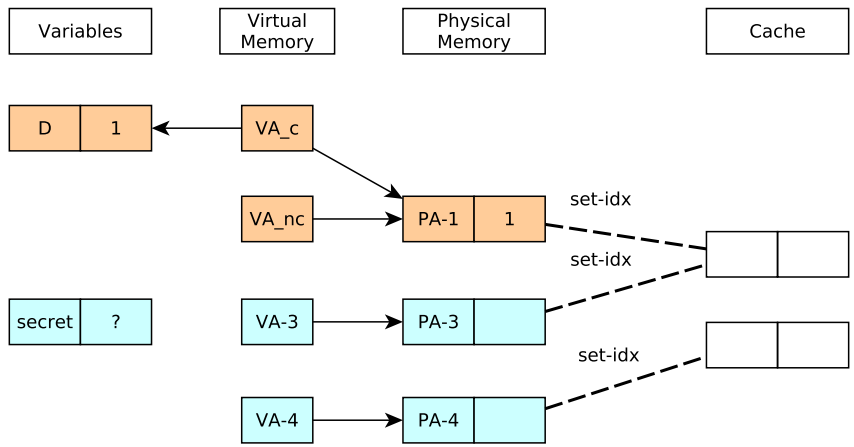
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
* A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



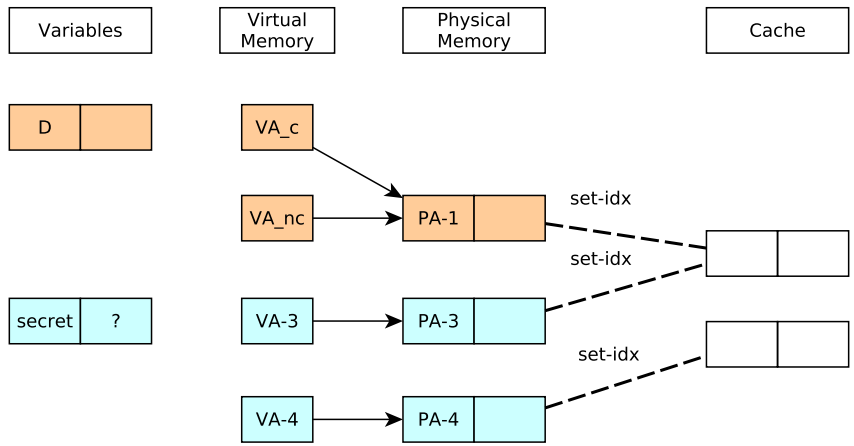
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
* A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



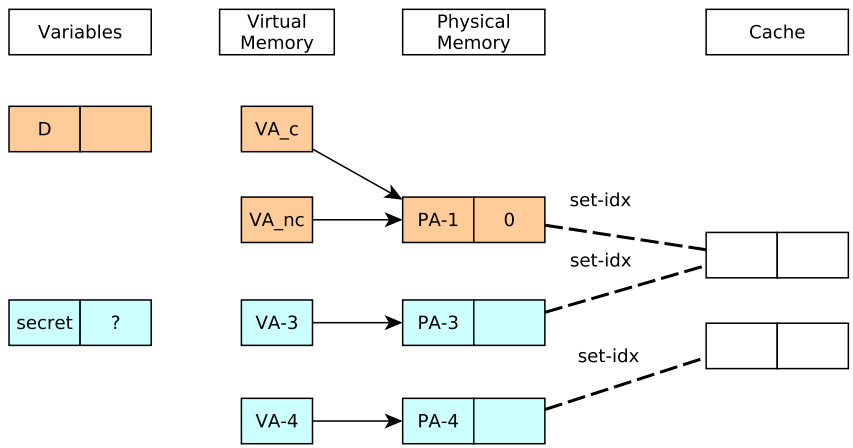
Attacking confidentiality using data-caches

```
* A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



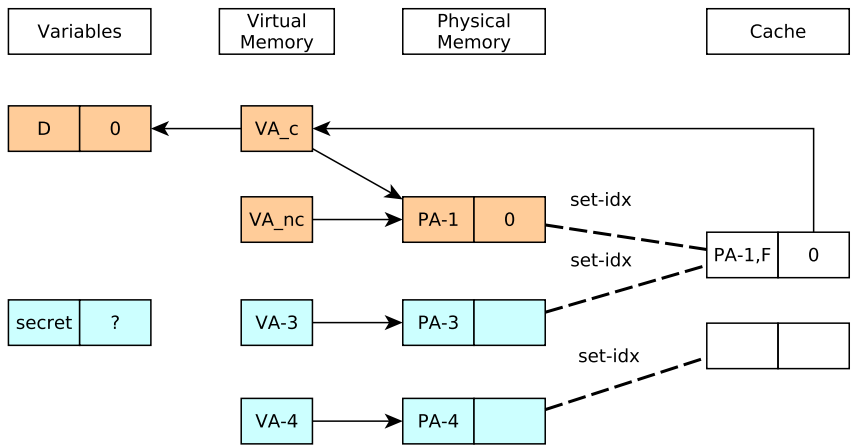
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
* A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



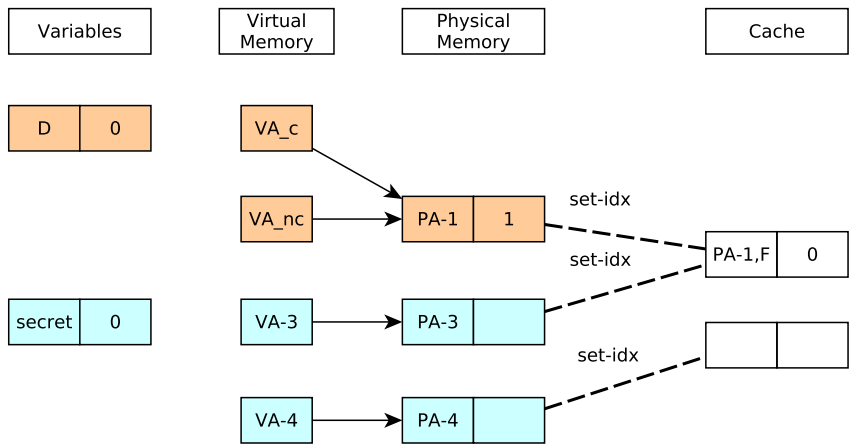
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
* A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



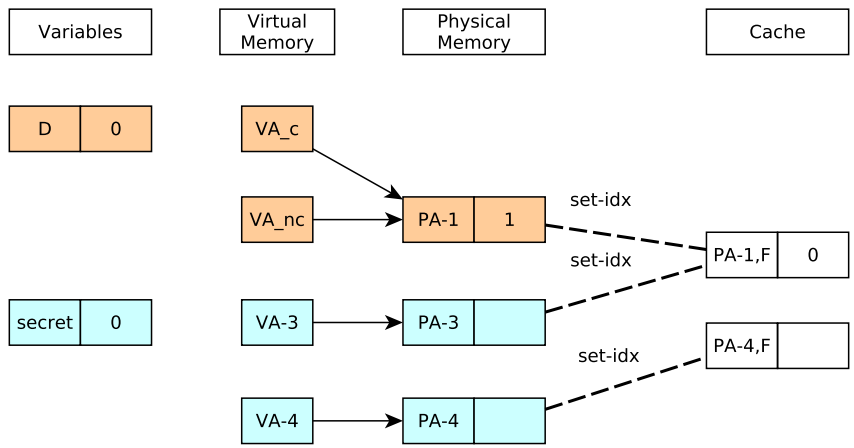
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
* A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



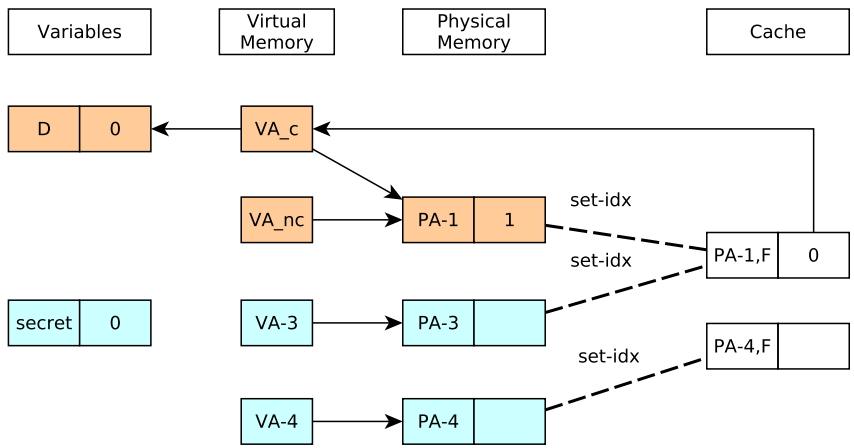
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
* A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



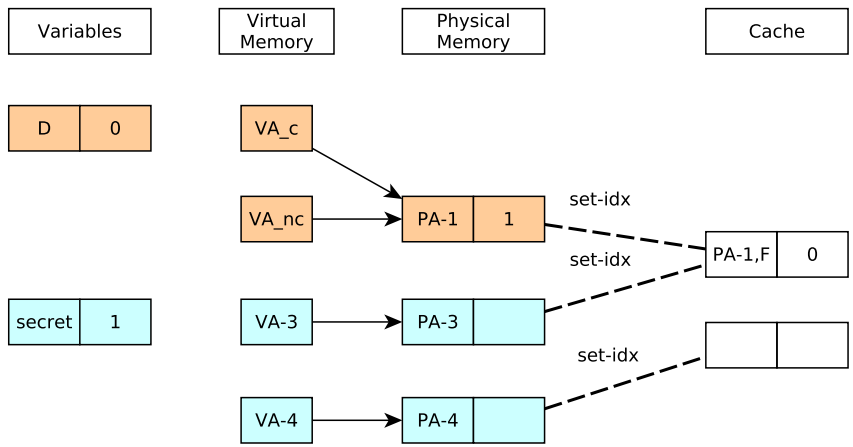
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
* A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



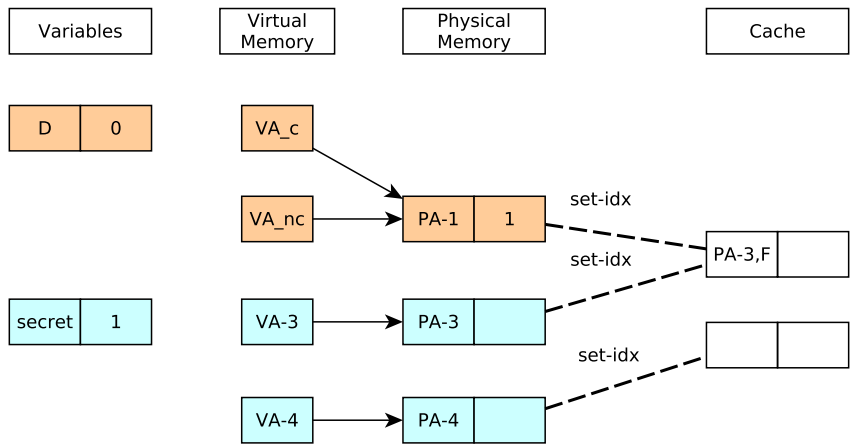
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
* A4) write(VA_nc, 1)
  A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



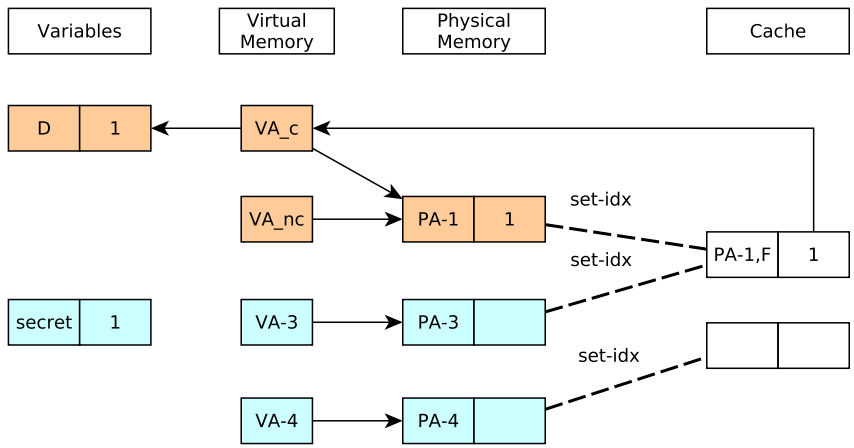
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
* A5) call victim
  A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



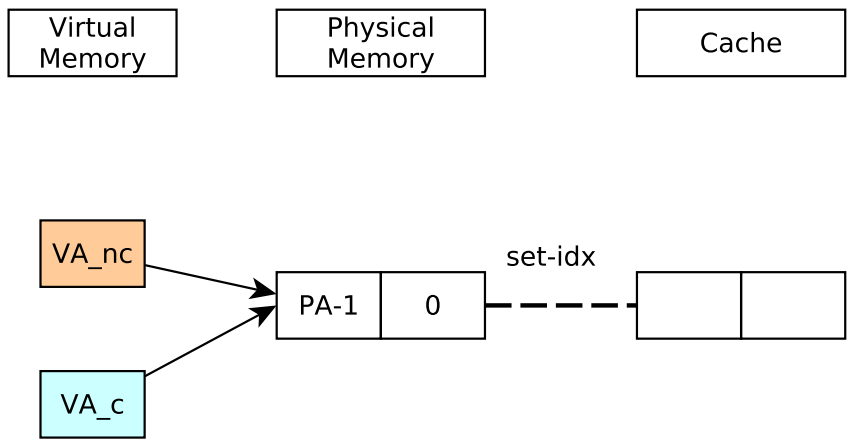
Attacking confidentiality using data-caches

```
  A1) invalidate(VA_c)
  A2) write(VA_nc, 0)
  A3) D = read(VA_c)
  A4) write(VA_nc, 1)
  A5) call victim
* A6) D = read(VA_c)

  V1) if secret
        access(VA3)
      else
        access(VA4)
```



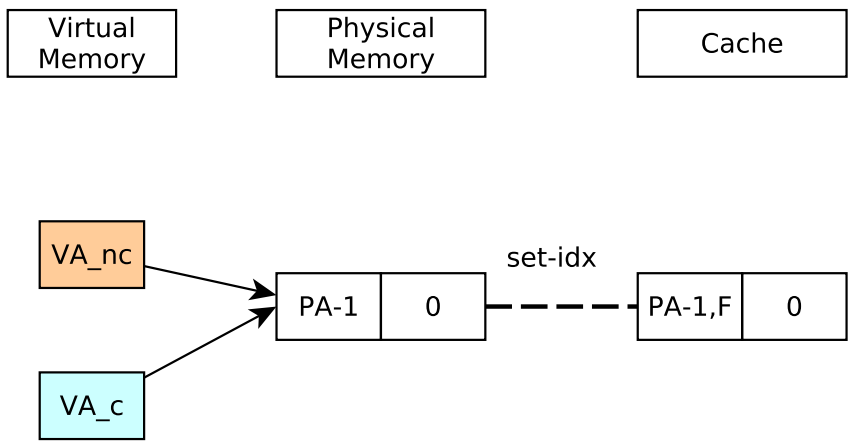
Attacking integrity using data-caches (TOCTTOU)

```
V1) D = access(VA_c)
...
A1) write(VA_nc, 1)
...
V2) D = access(VA_c)
V3) if not policy(D)
        reject
...
    [evict VA_c]
...
V4) use(VA_c)
```



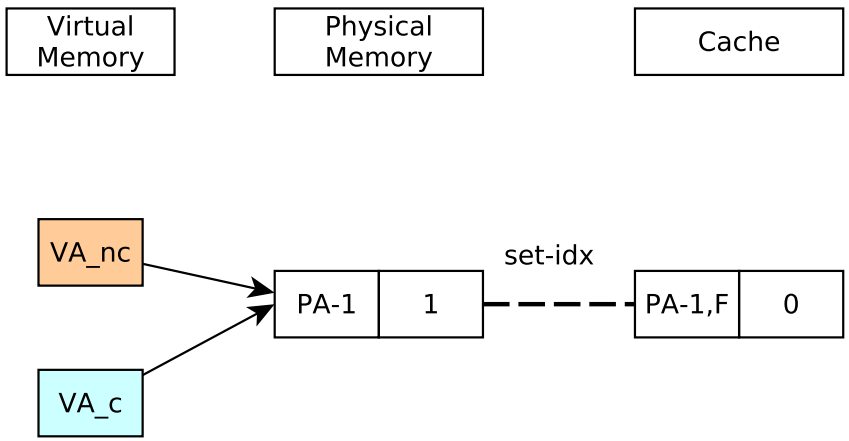
Attacking integrity using data-caches (TOCTTOU)

```
* V1) D = access(VA_c)
  ...
  A1) write(VA_nc, 1)
  ...
  V2) D = access(VA_c)
  V3) if not policy(D)
          reject
  ...
      [evict VA_c]
  ...
  V4) use(VA_c)
```



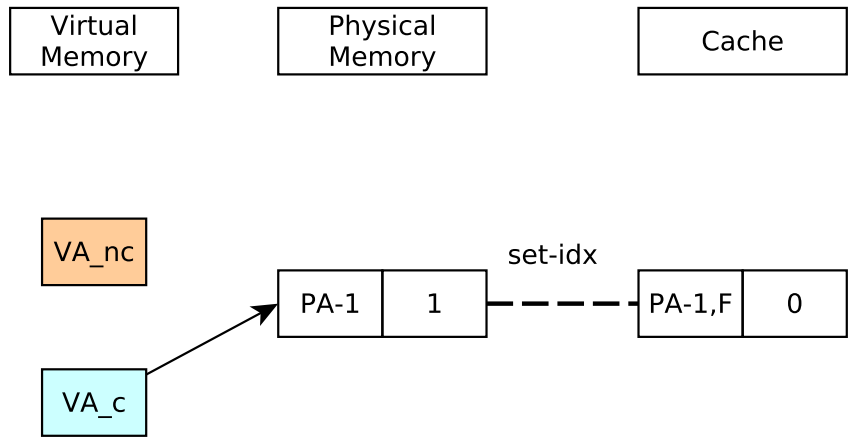
Attacking integrity using data-caches (TOCTTOU)

```
  V1) D = access(VA_c)
  ...
* A1) write(VA_nc, 1)
  ...
  V2) D = access(VA_c)
  V3) if not policy(D)
          reject
  ...
      [evict VA_c]
  ...
  V4) use(VA_c)
```



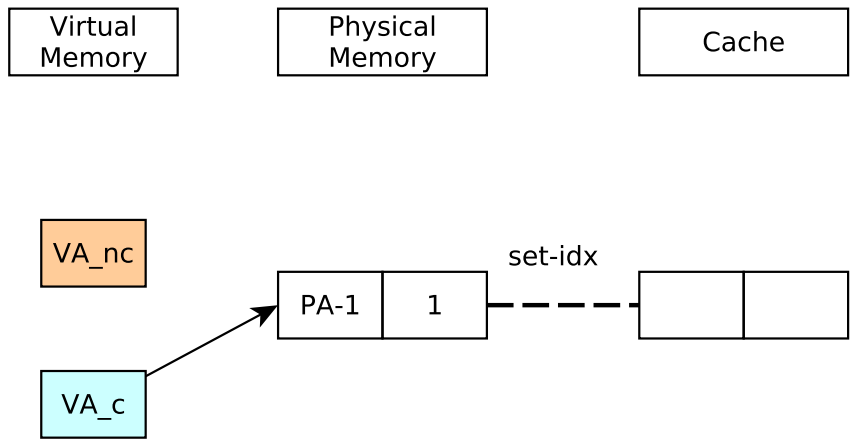
Attacking integrity using data-caches (TOCTTOU)

```
  V1) D = access(VA_c)
  ...
  A1) write(VA_nc, 1)
* ...
  V2) D = access(VA_c)
  V3) if not policy(D)
          reject
  ...
      [evict VA_c]
  ...
  V4) use(VA_c)
```



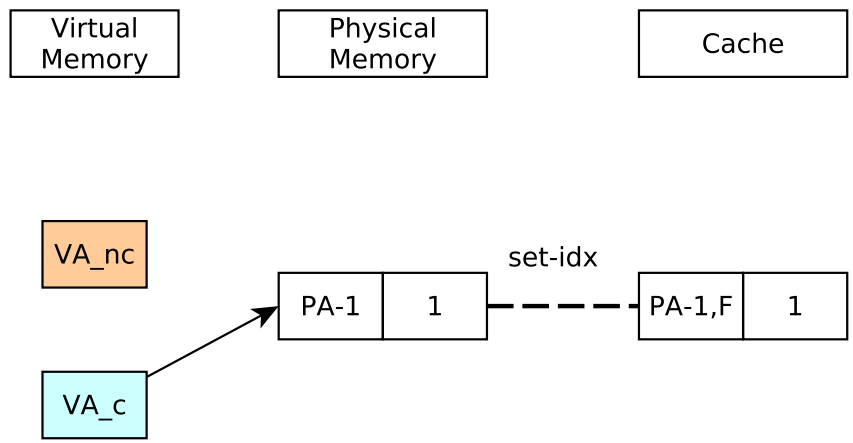
Attacking integrity using data-caches (TOCTTOU)

```
  V1) D = access(VA_c)
  ...
  A1) write(VA_nc, 1)
  ...
  V2) D = access(VA_c)
  V3) if not policy(D)
          reject
  ...
*     [evict VA_c]
  ...
  V4) use(VA_c)
```



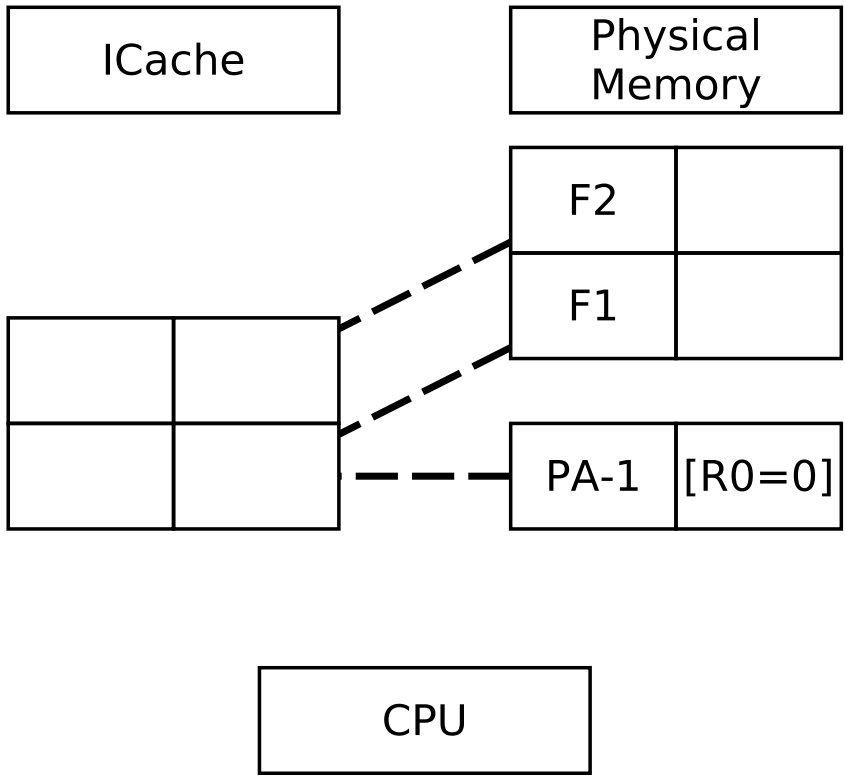
Attacking integrity using data-caches (TOCTTOU)

```
  V1) D = access(VA_c)
  ...
  A1) write(VA_nc, 1)
  ...
  V2) D = access(VA_c)
  V3) if not policy(D)
          reject
  ...
      [evict VA_c]
  ...
* V4) use(VA_c)
```



Attacking Confidentiality Using Instruction Caches

```
* A1) jmp A8
  A2) write(&A8, {R0=1})
  A3) call victim
  A4) jmp A8
  A5) D = R0
      ...
  A8) R0=0
  A9) return

  V1) if secret
        jmp f1
      else
        jmp f2
```



Case study: AES

- Platform: Raspberry PI 2
- Victim: AES service in TrustZone
- Attacker: Non-secure OS
- AES uses precomputes SBoxes (1KB=16 lines)

$$c[j] = K_n[j] \oplus T_4[s_{n-1}[j]]$$

- 128-bit AES key extracted after 850 encryptions

Case study: Violating Spatial Isolation in a Hypervisor

- Platform: Beagleboard MX
- Victim: paravirtualizing hypervisor using direct paging
- Attacker: Non-secure guest
- Page table stored in the guest memory
- PTs created by the guest / made read only / validated / activated
- Validation of stale data/activation of non-valid page table
- Attacker takes complete control of the system

Case study: Extraction of exponent from a modular exponentiation procedure

- Platform: Raspberry PI 2
- Victim: modular exponentiation procedure in Trustzone
- Attacker: Non-secure guest
- Non pc-secure procedure
- Invocation of function depends on the secret exponent
- Attacker identifies the execution path

Countermeasures

- For the side-channels, standard timing approaches
  - pc-secure code, secret independent memory accesses, ...
- For integrity: guarantee coherency of accessed memory
  - cache-flushes, explicit eviction of cache-lines, ...
- Specific for the new attacks
  - access the cache lines independently on secret
  - avoid uncacheable aliases for a region of memory

Countermeasures: hypervisor

LMbench	Native	Hyp	ACPT	SelFI	Flush
read	0.84	2.19	2.20	2.20	2.38
fork+execve	2068	5249	5248	6285	39029
pagefaults	3.76	11.21	11.12	21.55	332.82

Application benchmark	Native	Hyp	ACPT	SelFI	Flush
tar (500K)	70	70	70	70	190
tar (2M)	230	210	200	210	370
dd (10M)	90	140	140	160	990
dd (40M)	330	500	450	600	3830
jpg2gif(5KB)	60	60	60	60	130
jpg2bmp(5KB)	40	40	40	40	110
jpegtrans(270', 5KB)	10	10	10	10	80
bmp2tiff(90 KB)	10	10	10	10	60
tif2rgb(200 KB)	10	20	20	20	120
sox(aif2wav 100KB)	20	20	20	30	140

Countermeasure: AES

AES encryption	5 000 000 × 16B		10 000 × 8KB	
	Time	Throughput	Time	Throughput
Original SBoxes	23s	3.317 MB/s	13s	6.010 MB/s
Compact Last SBox	24s	3.179 MB/s	16s	4.883 MB/s
Scrambled Last SBox	30s	2.543 MB/s	20s	3.901 MB/s
Uncached Last SBox	36s	2.119 MB/s	26s	3.005 MB/s
Scrambled All SBoxes	132s	0.578 MB/s	125s	0.625 MB/s
Uncached All SBoxes	152s	0.502 MB/s	145s	0.539 MB/s

Repairing the Integrity Verification

- critical resources cannot be directly (or indirectly) affected by untrusted SW
- for these resources the actual system must behave according to the formal abstract specification
- main verification condition: for every address that belongs to the critical resources, if there is a cache hit and the corresponding cache line differs from the main memory then the cache line must be dirty

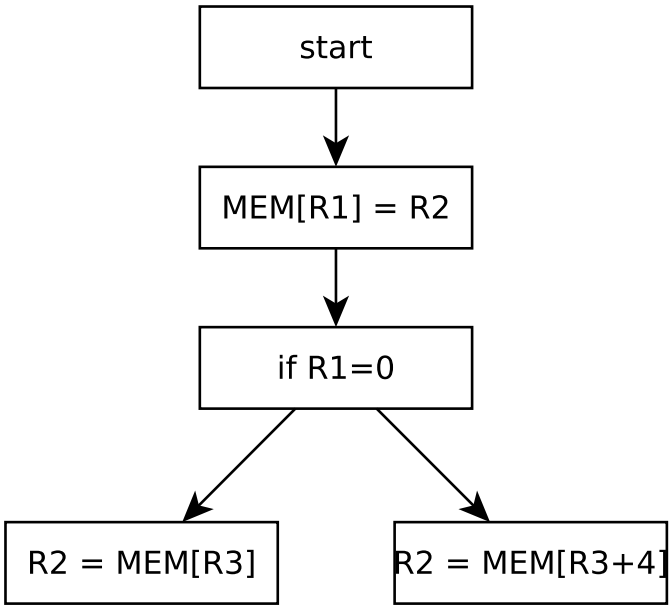
Repairing the Confidentiality Verification

- no side channel is present due to caches
- the attacker can observe all the resources that can affect the eviction
  - cache line tag
  - cache line emptyness
- goal: after the execution of an arbitrary functionality these resources do not depend on confidential data

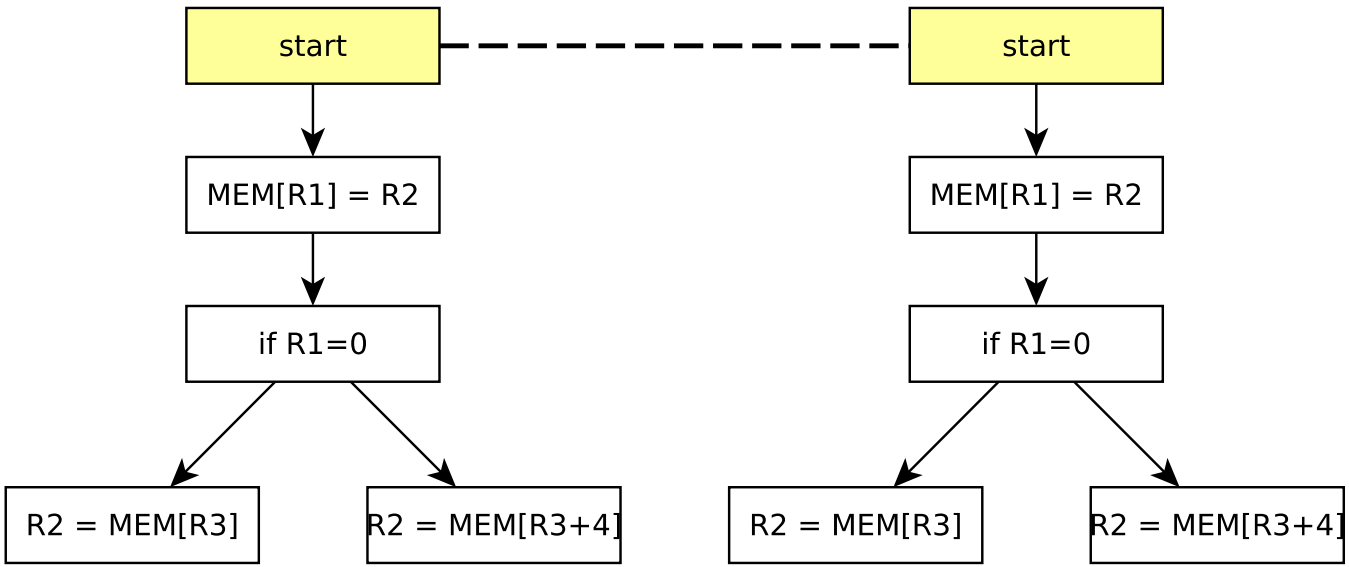
Repairing the Confidentiality Verification

- two executions of a program, starting from equivalent non-confidential resources
- architectural property: equivalence of line emptiness and tag is preserved by cache safe operations
  - same type of memory access
  - equivalence of the tag and index of the PC
  - equivalence of the tag and index of the address accessed
- verification condition is a relational observation equivalence
- we use existing tools for relational verification that support trace based observations

Relational verification



Relational verification



$$assume(P)$$

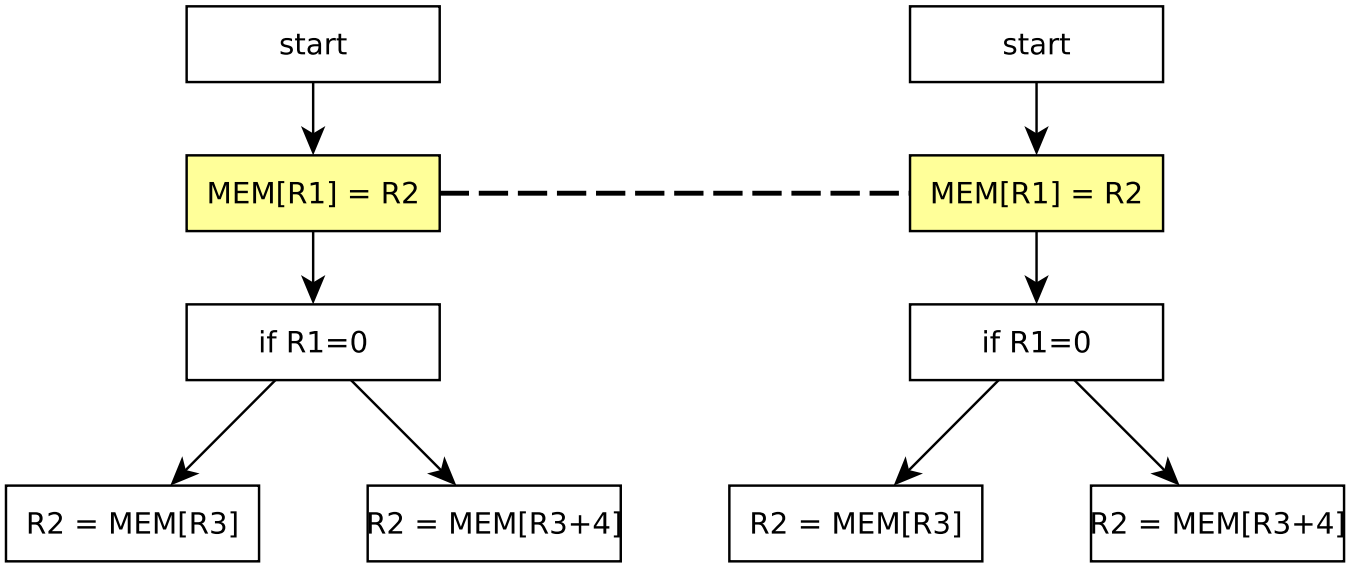
$$R3 > 42$$

$$\forall a \in [1\ MB, 2\ MB].mem(a) = mem'(a)$$

$$R0 = R0'$$

$$R0 + R1 = R0' + R1'$$

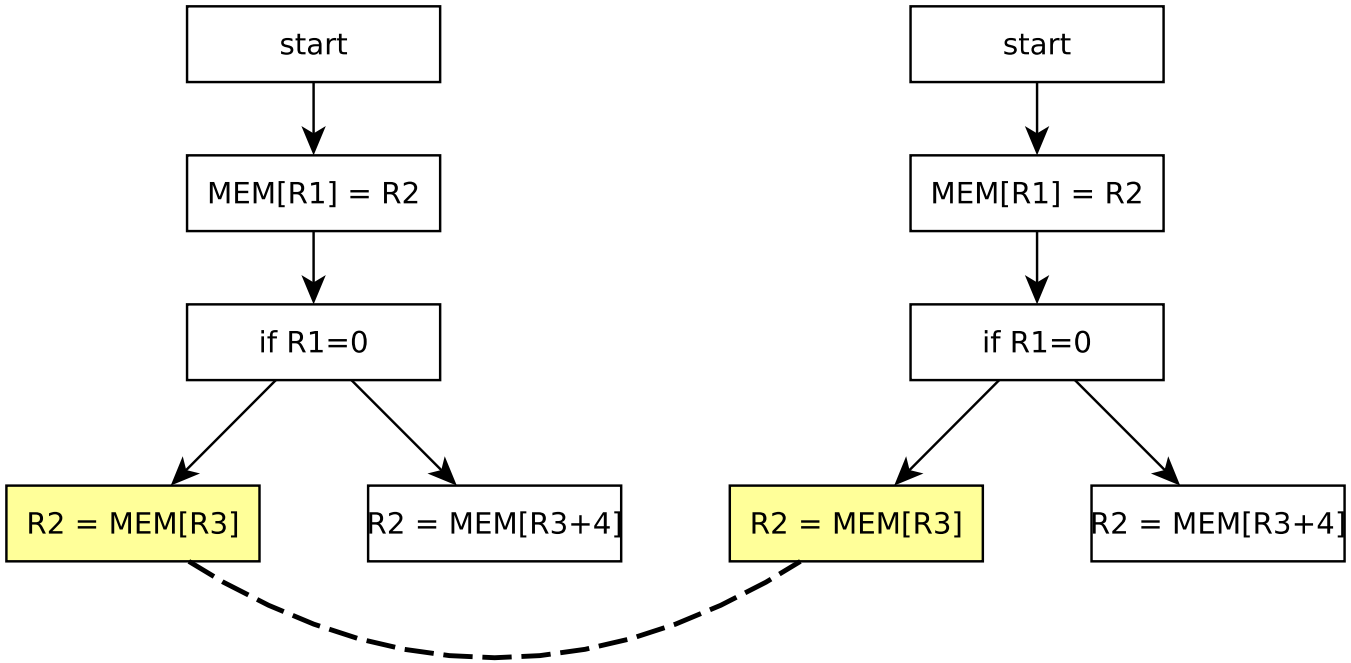
Relational verification



$$P \Rightarrow tag(R1) = tag(R1')$$

$$P \Rightarrow idx(R1) = idx(R1')$$

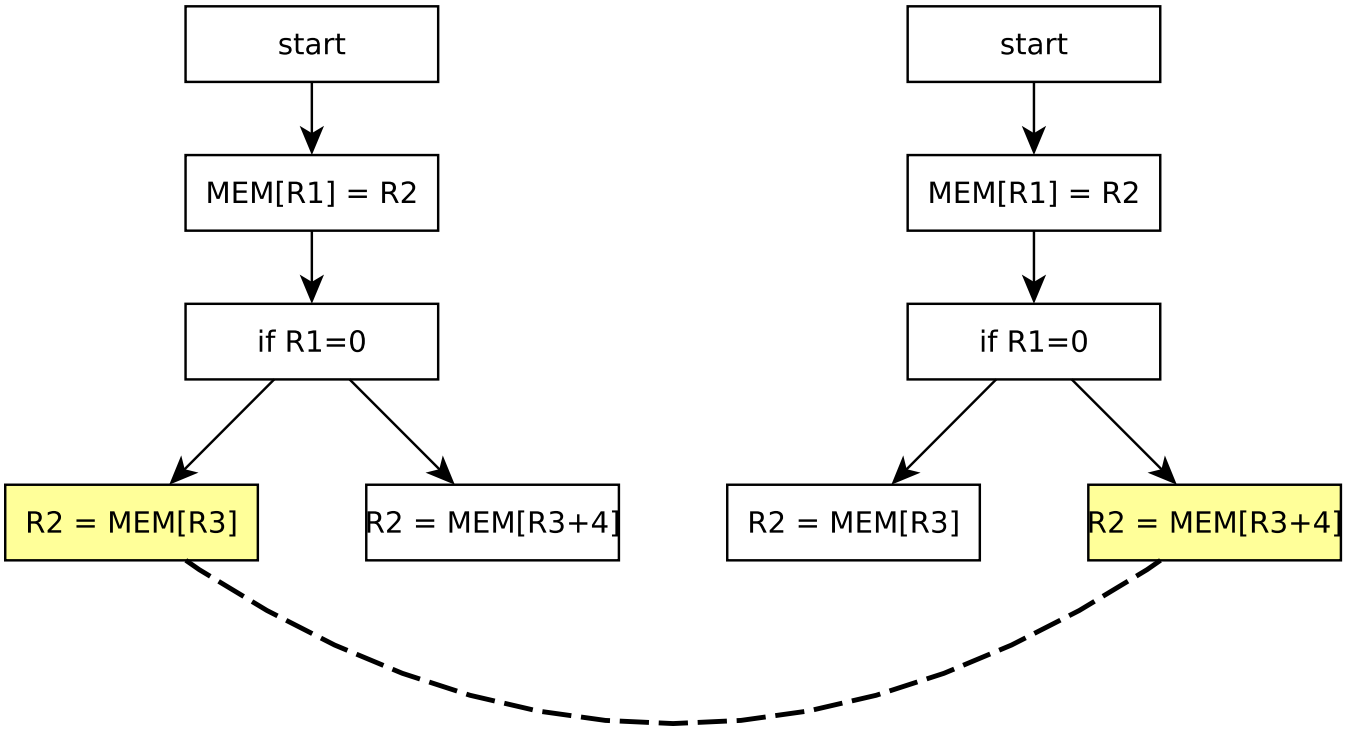
Relational verification



$$P \wedge (R1 = 0) \wedge (R1' = 0) \Rightarrow tag(R3) = tag(R3')$$

$$P \wedge (R1 = 0) \wedge (R1' = 0) \Rightarrow idx(R3) = idx(R3')$$

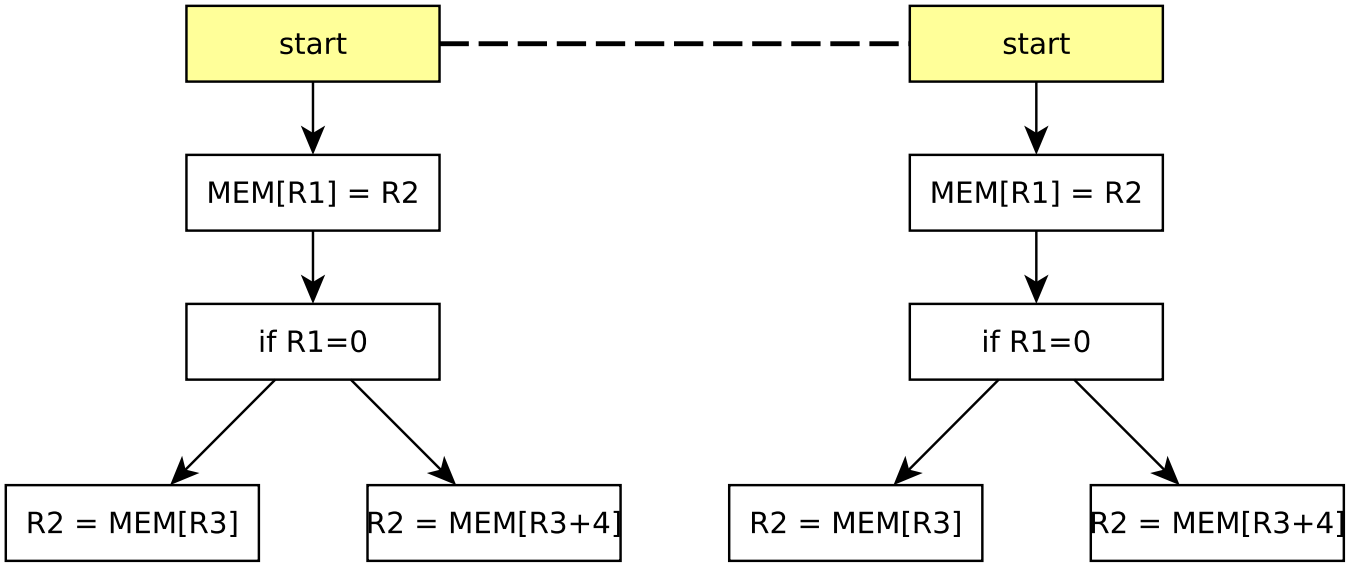
Relational verification



$$P \wedge (R1 = 0) \wedge (R1' \neq 0) \Rightarrow tag(R3) = tag(R3' + 4)$$

$$P \wedge (R1 = 0) \wedge (R1' \neq 0) \Rightarrow idx(R3) = idx(R3' + 4)$$

Relational verification



- Symbolic execution and weakest precondition propagation
- SMT solver to check conditions and early prune of unreachable path-pairs
- Force same memory operations
- For instruction cache observations are tag and index of the program counter

Ongoing work

- Repair of formal verification
- TLBs / Branch prediction / ...
- Experimentaiton in multi-core
- Experimentaiton using GPU
- Evaluating HW countermeasures

Thank you