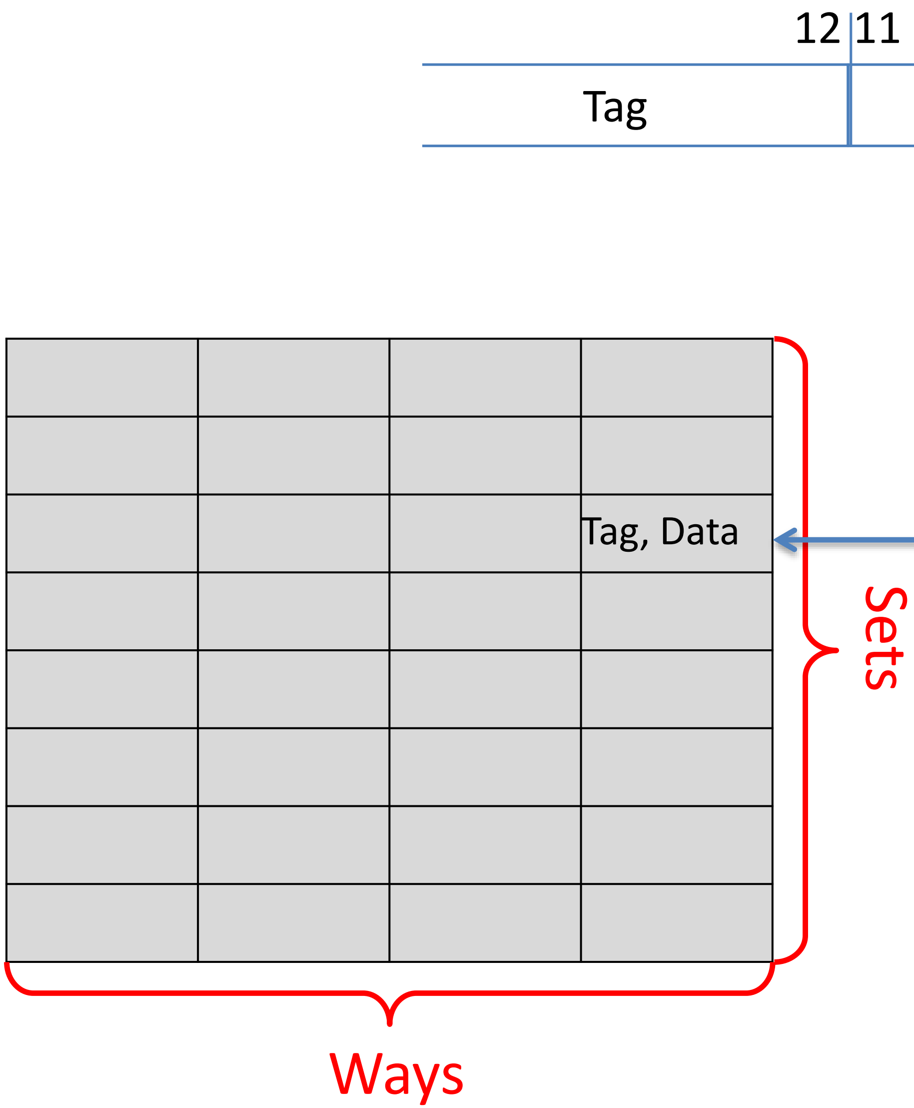
Microarchitectural Side-Channel Attacks

Part 2

Yuval Yarom

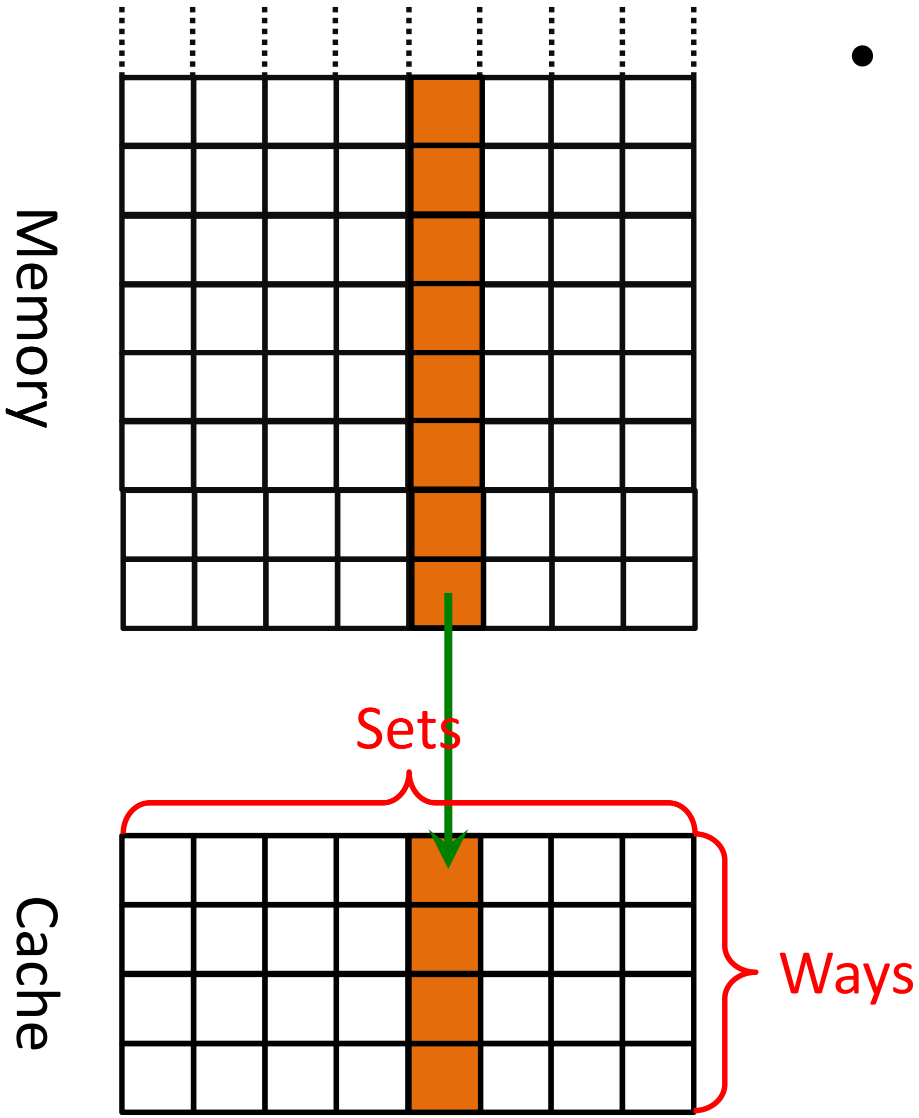
The University of Adelaide and Data61

X86 L1 Cache



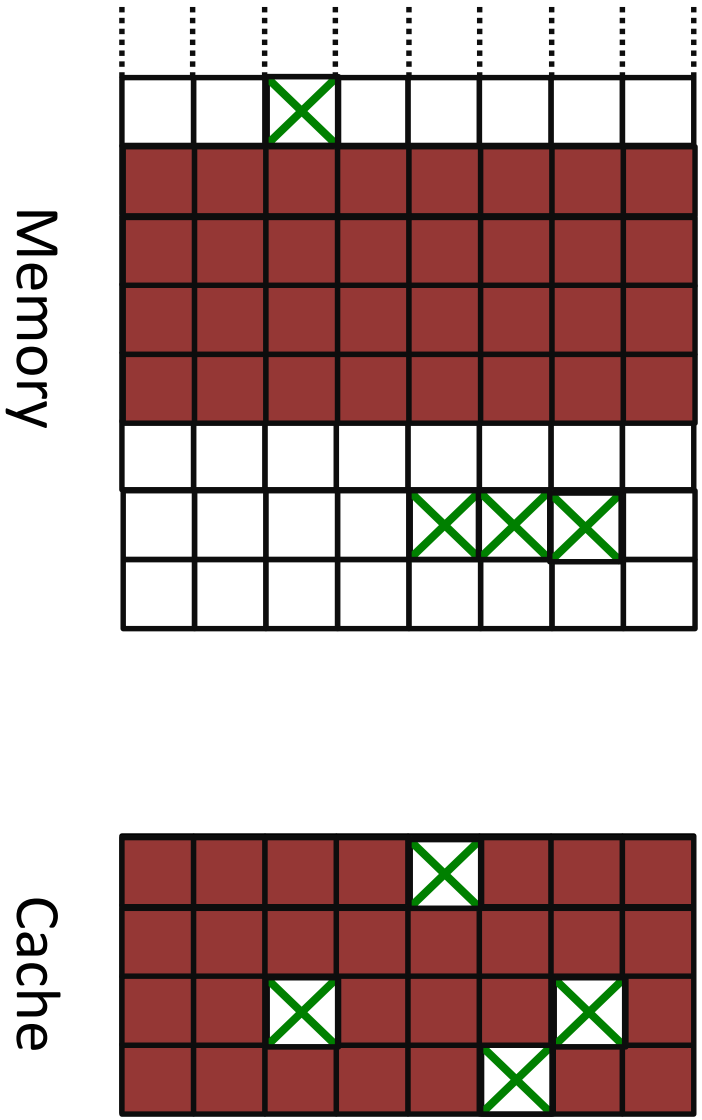
- Stores fixed-size (64B) *lines*
- Arranged as multiple (64) *sets*, each consisting of multiple (8) *ways*.
- Each memory line maps to a single cache set
  - Bits 6-11 select the set
  - Can be cached in any of the ways in the set

X86 L1 Cache



- Better visualised when the cache is rotated

Prime+Probe [OST05, Per05]



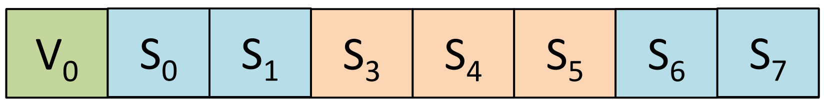
- Attacker chooses a cache-sized memory buffer
- Attacker accesses all the lines in the buffer, filling the cache with its data
- Victim executes, evicting some of the attackers lines from the cache
- Attacker measures the time to access the buffer
  - Accesses to cached lines is faster than to evicted lines

The gritty details

- The observer effect
  - Our code uses the cache - want to minimise our footprint
- The optimising compiler removes what it thinks is dead code
  - Not optimising increases the code's footprint
  - Solution:
    - Know your optimiser
    - Use assembly language

Thrashing

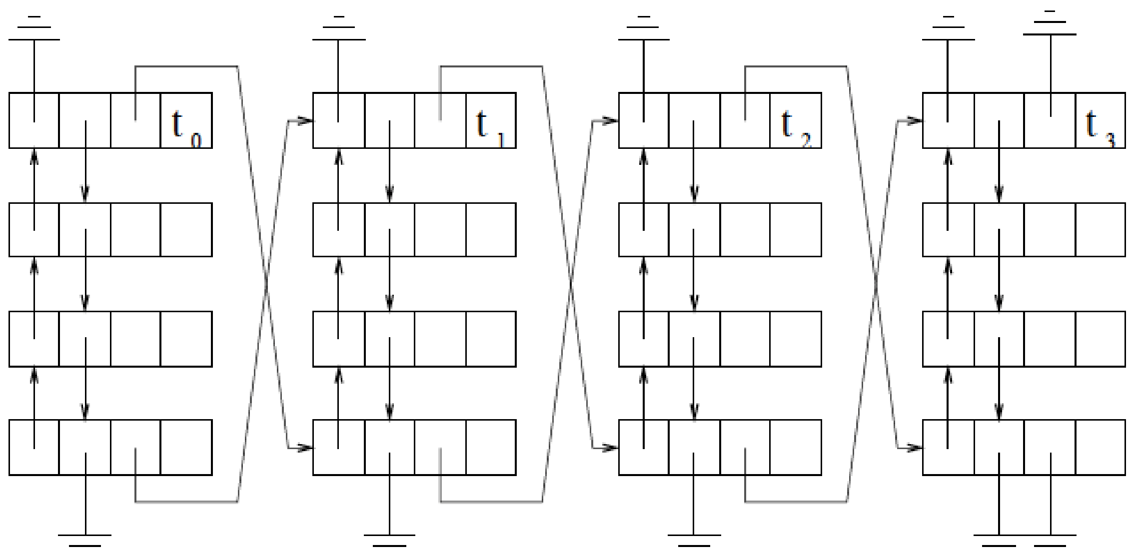
- The cache uses a Pseudo-LRU replacement
- Our probe + victim access can cause thrashing



- Solution [TOS10]: Zig-zag on data

Hardware prefetcher

- Aims to improve temporal locality
- Brings data to the cache before it is required
  - We do not want that!!!
- Solution [TOS10]: pointer chasing
  - [Per05] uses data dependency for achieving the same result



Data streams

- The cache aims to predict regular access patterns with fixed strides
- Linear access within a page may trigger the mechanism
- Solution [OST10]: Randomise order of probed sets

Putting it all together

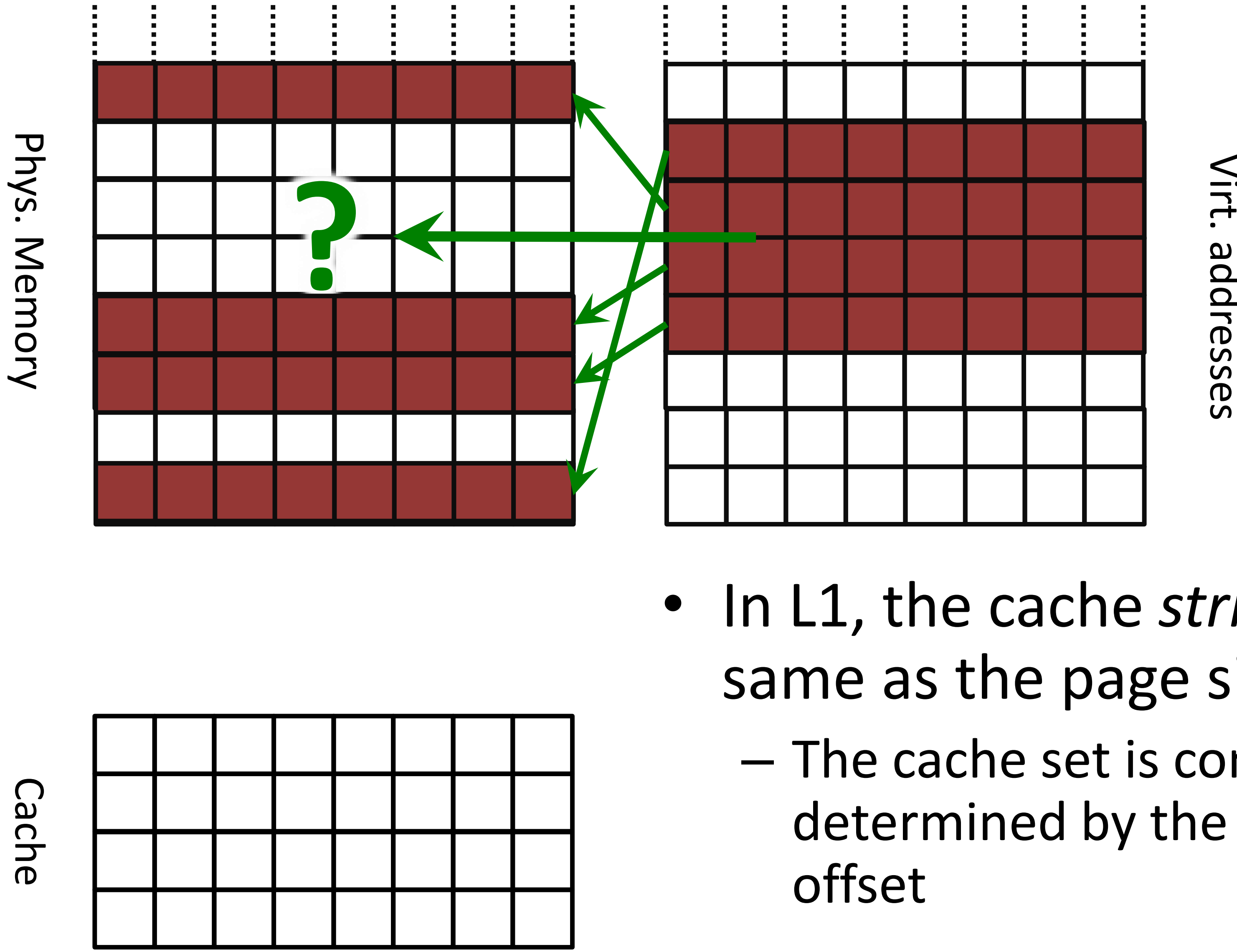
- L1-capture

- With L1-rattle
- With rungnupg

Challenges to last-level cache attacks

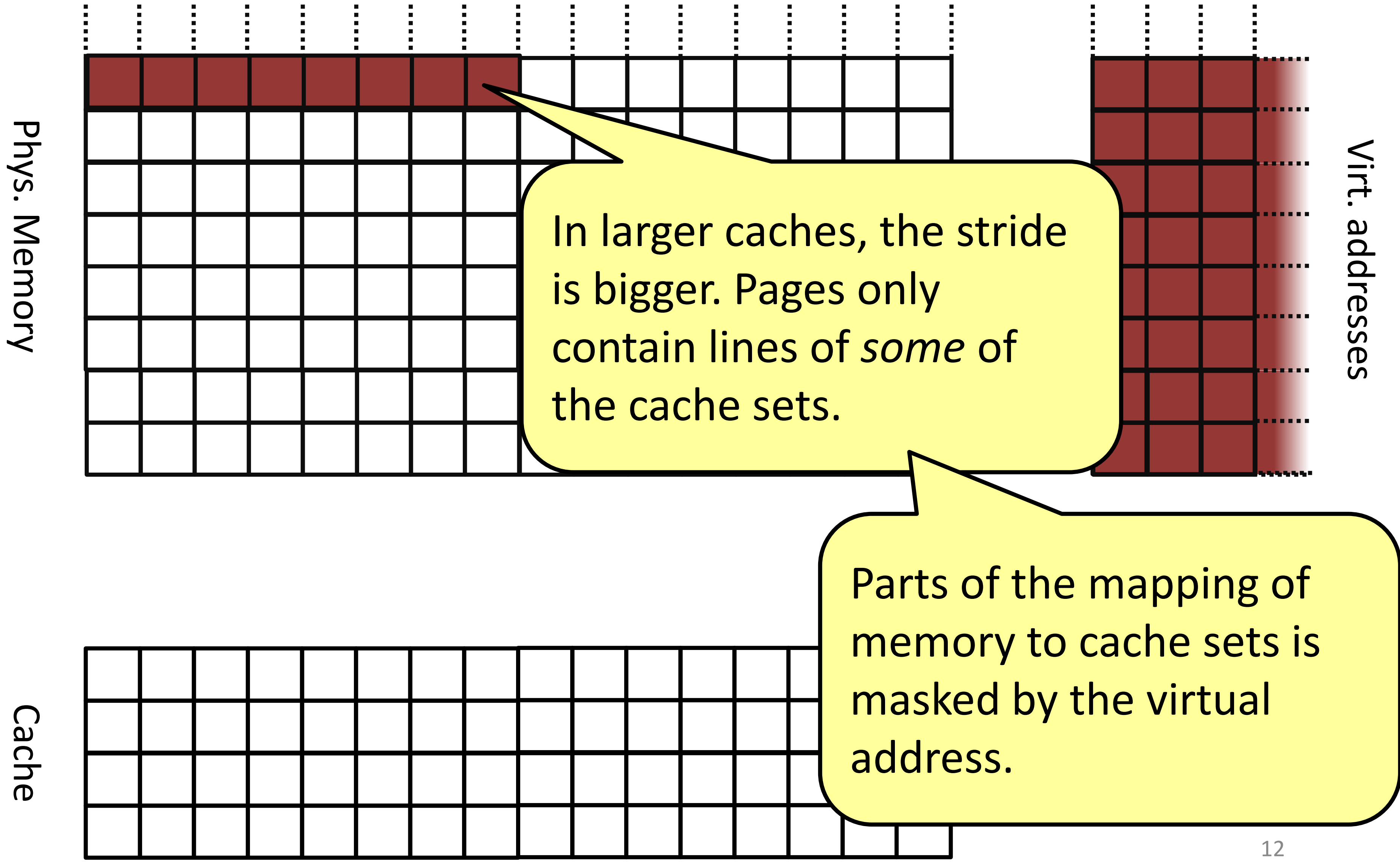
- Difficulty in finding memory lines that map to a given cache set
  - Virtual memory
  - LLC slices
- Large cache size and longer cache access times mean LLC Prime+Probe is very slow
- Visibility of the victim memory access at the LLC
  - Intel inclusive cache takes care of the issue

Virtual vs. Physical addresses

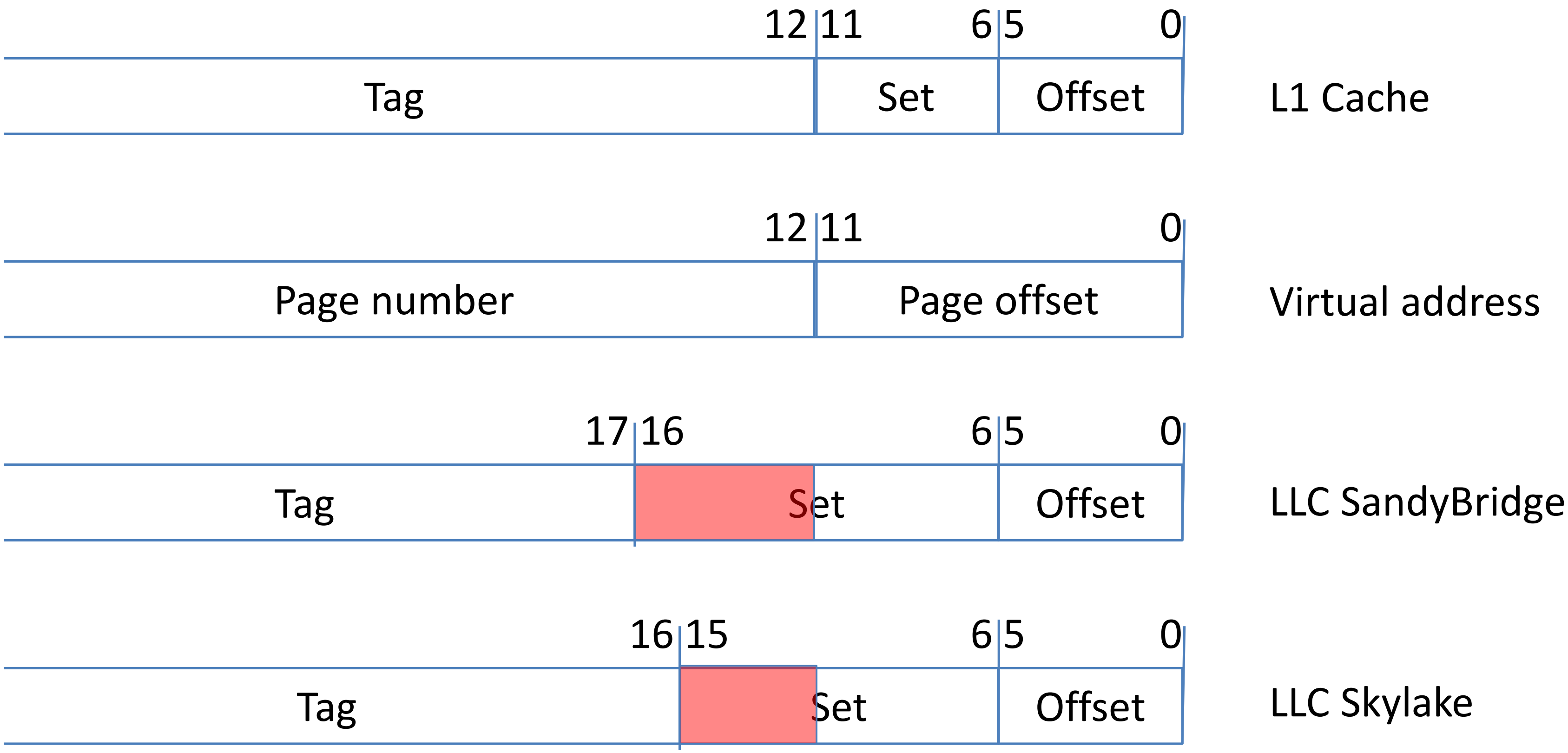


- In L1, the cache *stride* is the same as the page size (4KiB)
  - The cache set is completely determined by the page offset

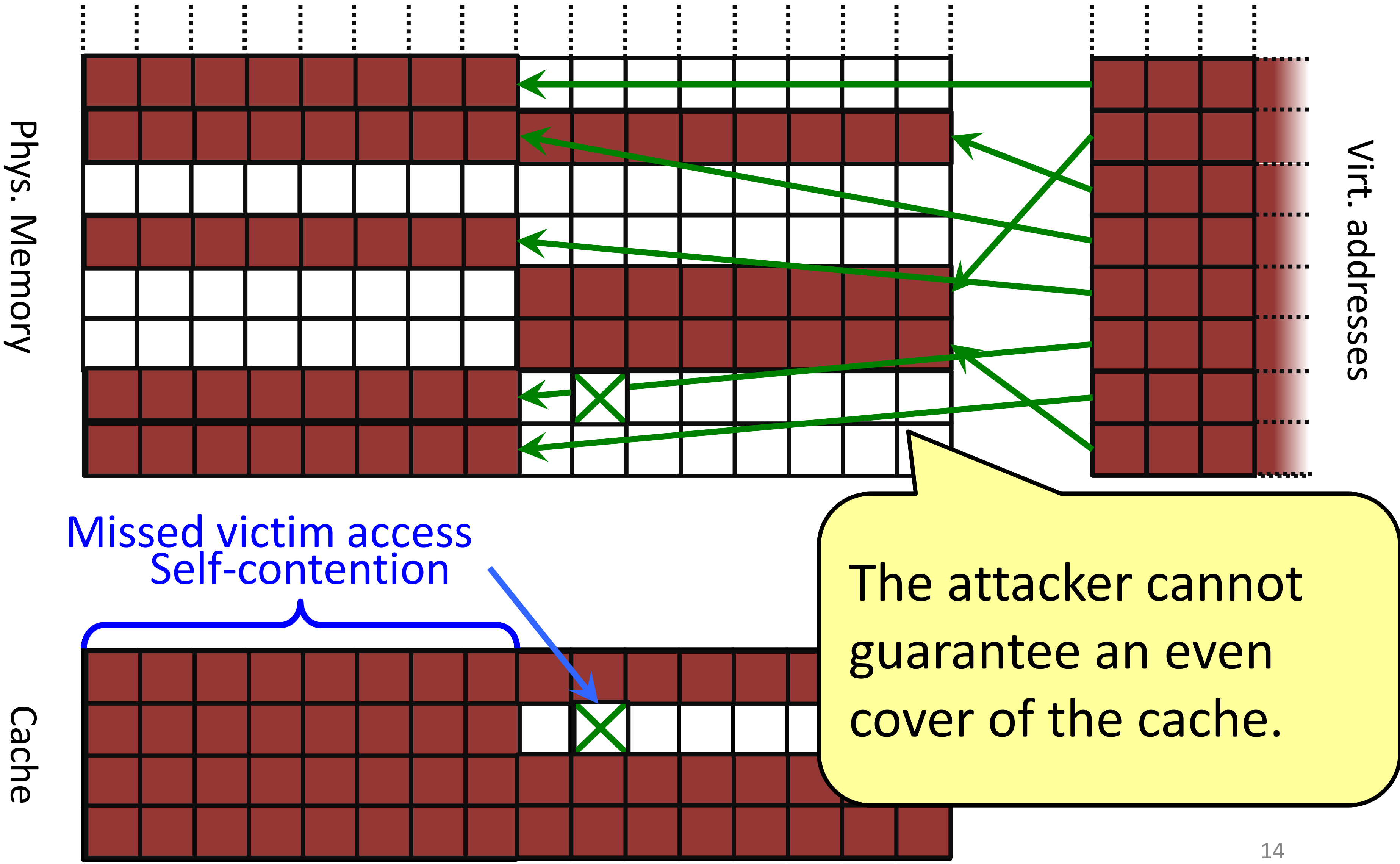
Addressing uncertainty



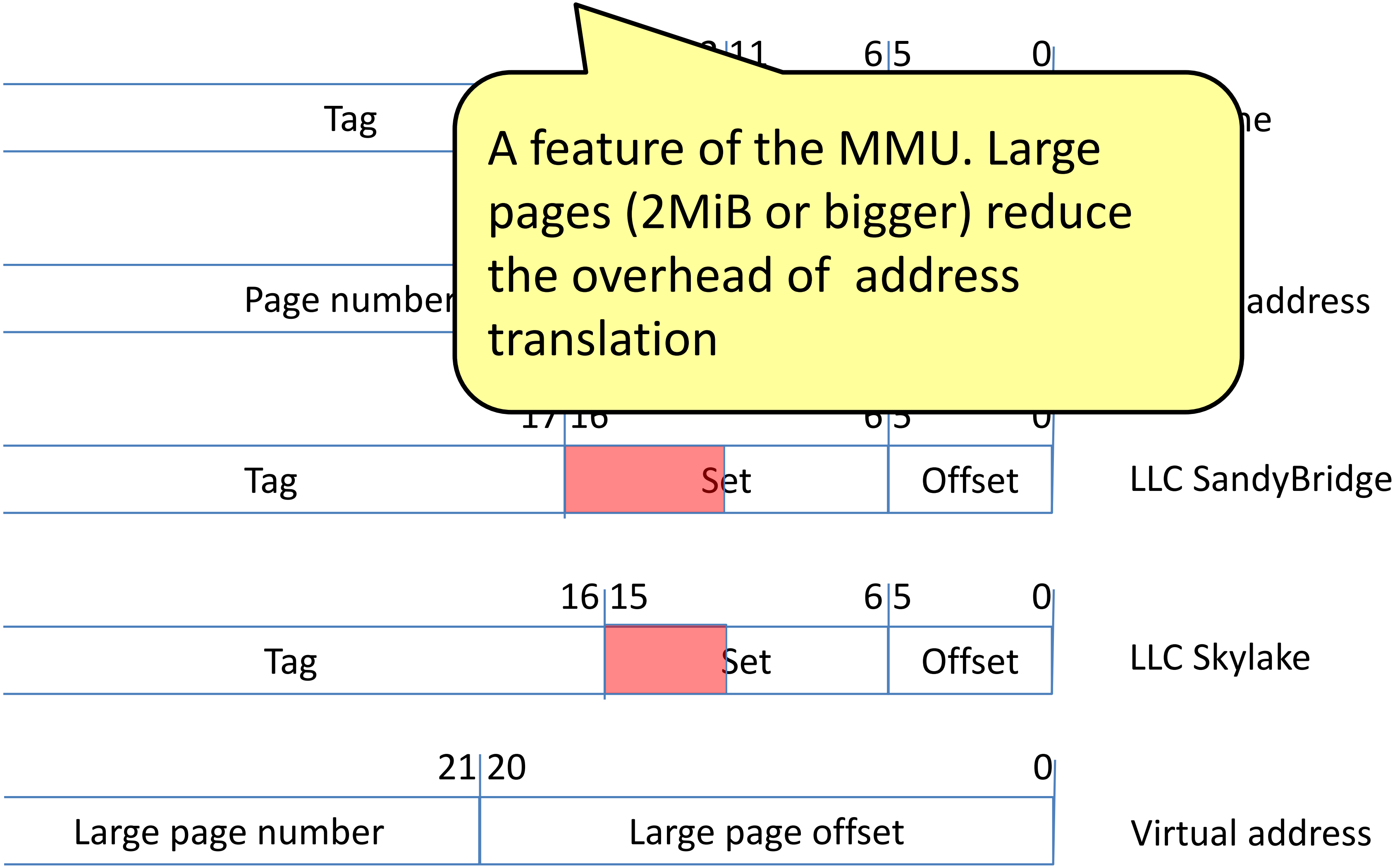
Address fields



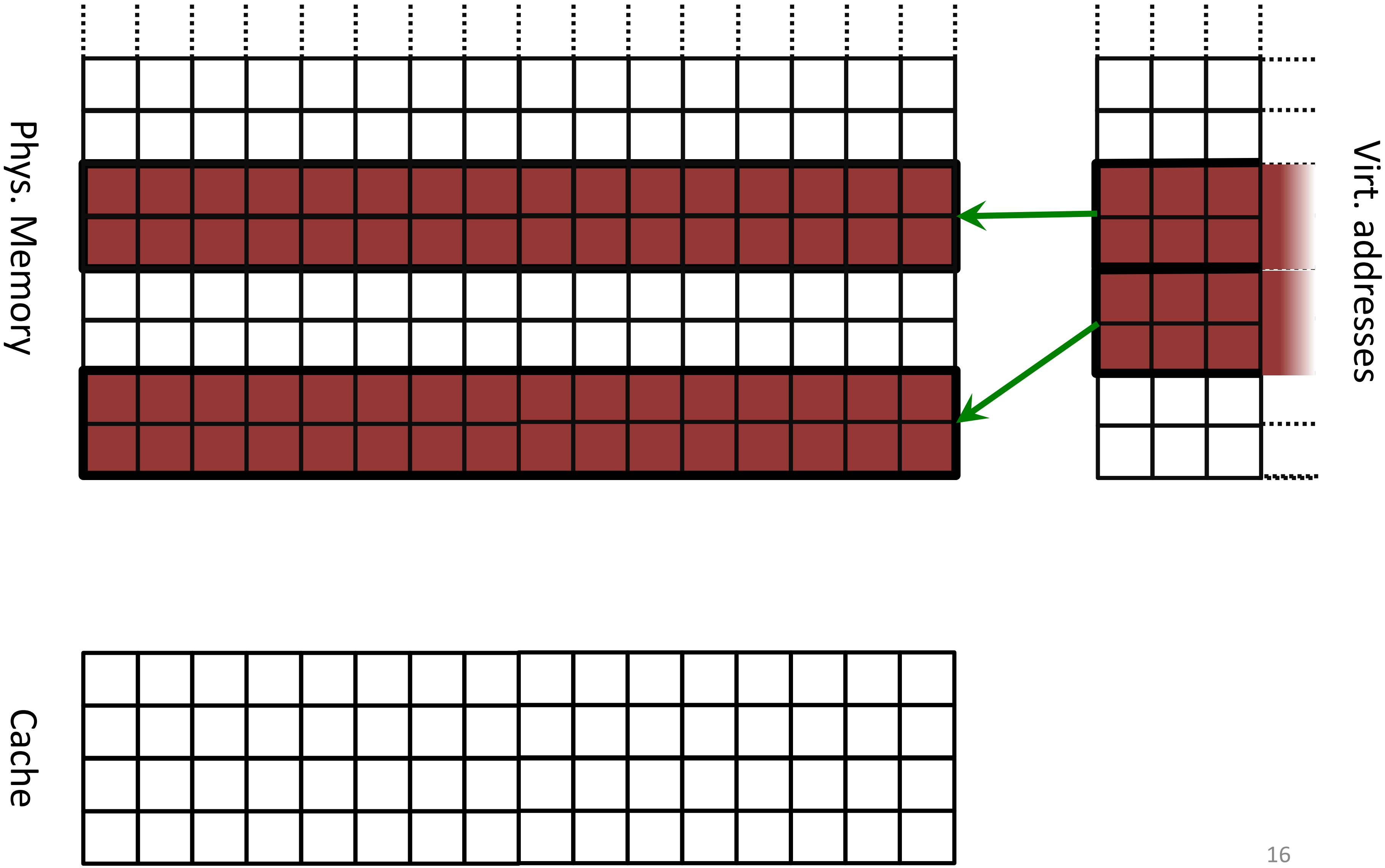
Addressing uncertainty



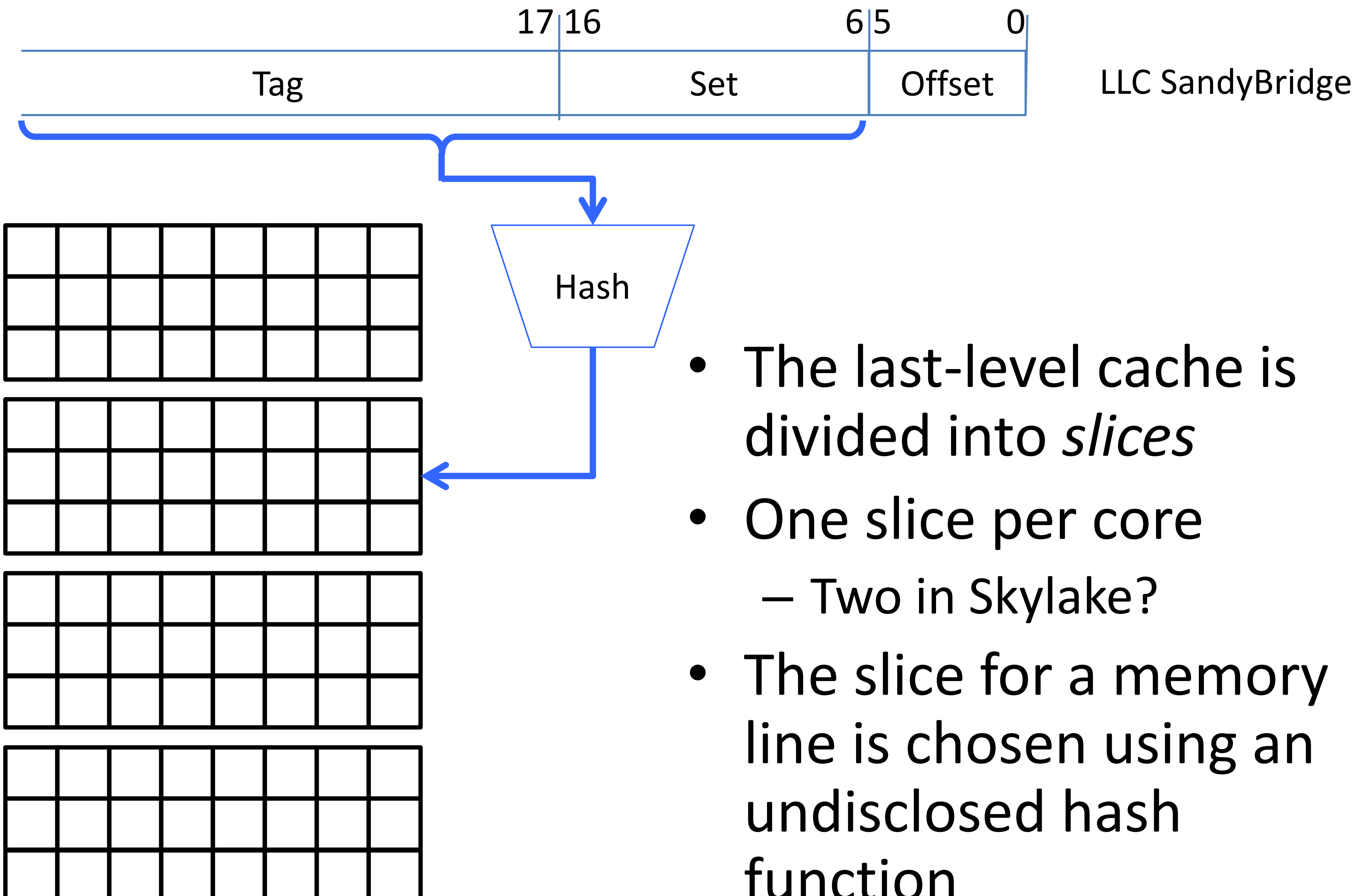
Solution : Use large pages [LYG+15,IES15]



Solution – large pages



Intel LLC Slices



- The last-level cache is divided into *slices*
- One slice per core
  - Two in Skylake?
- The slice for a memory line is chosen using an undisclosed hash function

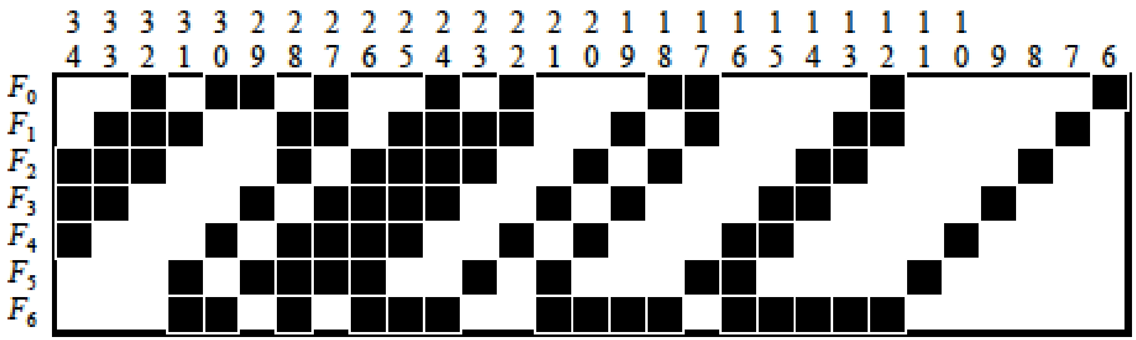
Reverse Engineering the Hash Function

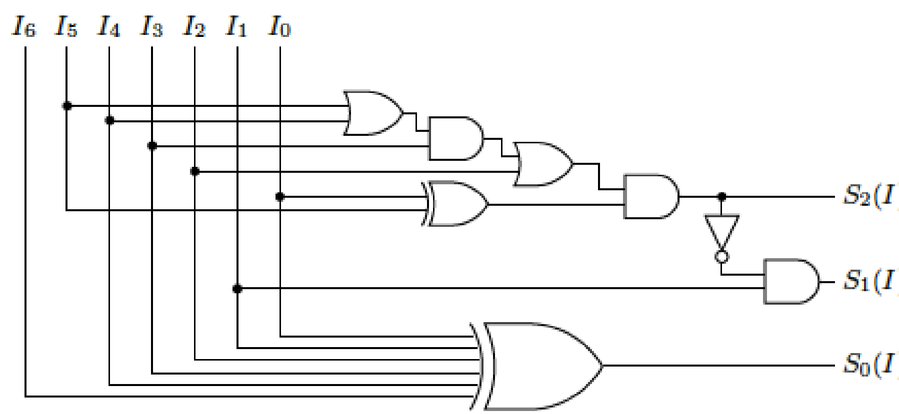
- [MLN+15] Use performance counters to RE linear hash functions (number of cores is a power of two)

		Address Bit																															
		37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06
2 cores	o_0						⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
4 cores	o_0					⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1				⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕				⊕	
8 cores	o_0		⊕	⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕	⊕	⊕		⊕		⊕		⊕				⊕
	o_1	⊕		⊕	⊕	⊕		⊕		⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕		⊕		⊕				⊕	
	o_2	⊕	⊕	⊕	⊕			⊕	⊕			⊕	⊕			⊕	⊕			⊕			⊕			⊕	⊕				⊕		

Reverse Engineering the Hash Function

- [YGL+15] use timing to RE the function for 6 cores





Reverse Engineering the Hash Function

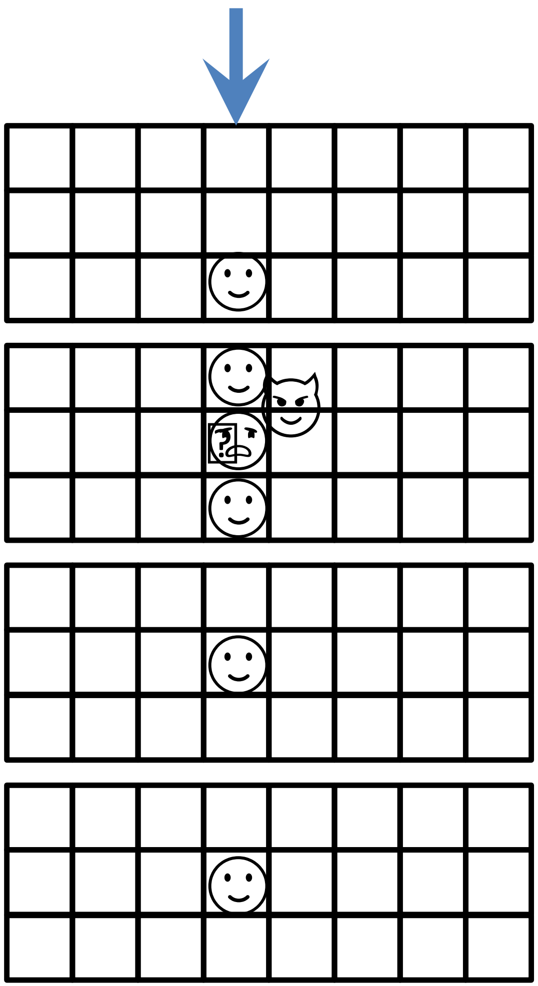
- [MLN+15] Use performance counters to RE linear hash functions (number of cores is a power of two)
- [YGL+15] use timing to RE the function for 6 cores

- But – need physical addresses
  - Can be done on Linux < 4.0
  - Unless running in a VM

Probing the hash function

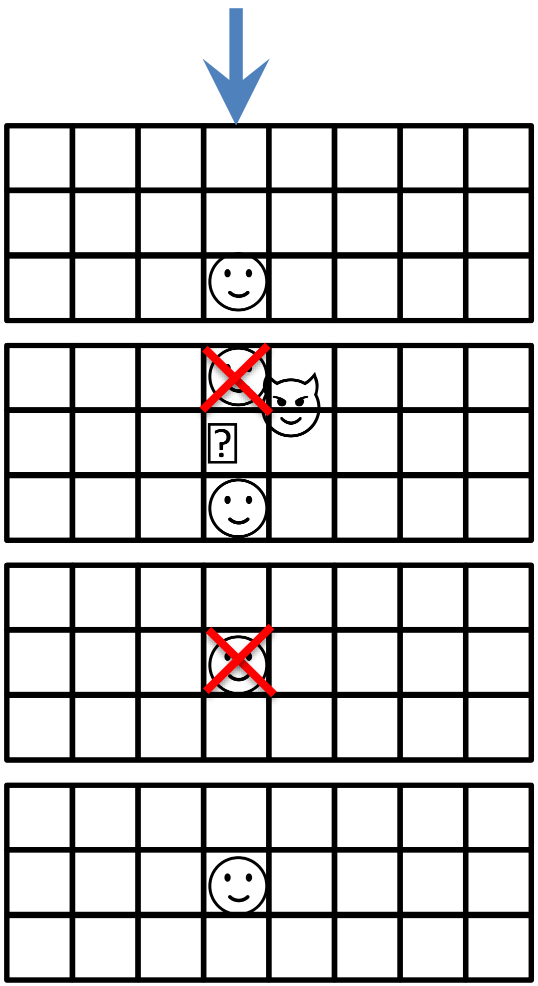
- Start with a set of potentially conflicting memory lines
  - At least twice as many as the totalsize of the cache sets
- *Expand* a subset until it conflicts on a single set in a single slice
- *Contract* the subset until it contains only lines of the conflicted set
- *Collect* all of the lines of the conflicting set from the original set
- Repeat until the original set is (almost) empty

Expand



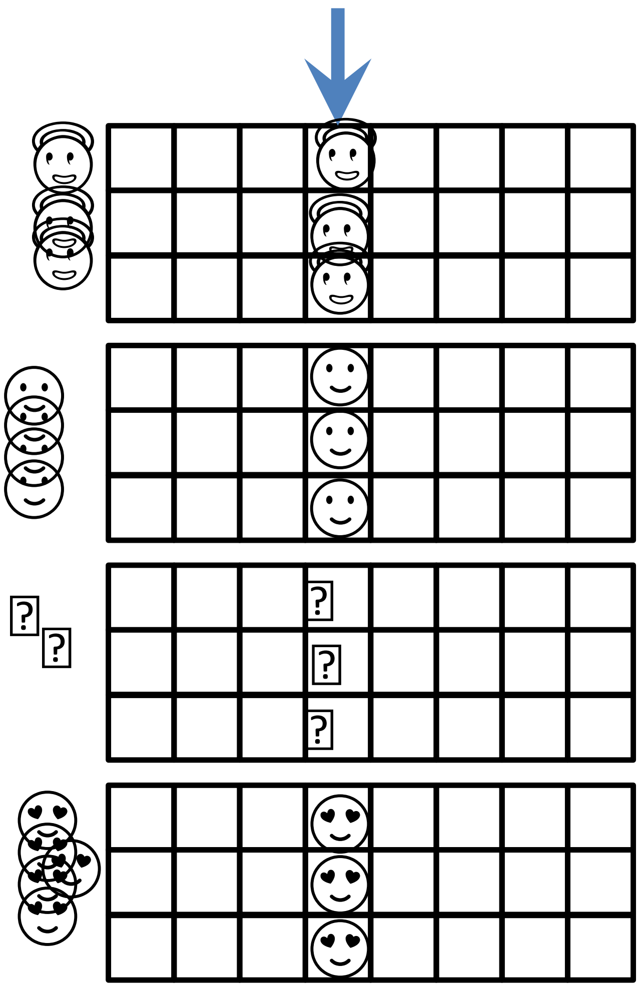
- Start from an empty subset
- Iteratively add lines to the subset as long as there is no self-eviction
- Self-eviction is detected by priming a potential new member, accessing the current subset and timing another access to the potential new member

Contract



– Iteratively remove lines from the subset checking for self-eviction
– Only keep members if self-eviction disappears when removed

Collect



– Scan original set, looking for members that conflict with the contracted subset

Demo

- L3-capture
- L3-capturecount

Slow LLC Prime+Probe times

- L1 (32KB) probe:
  - 64 sets * 8 ways *4 cycles = 2,048 cycles
- Small last-level cache (6MB):
  - 8,192 sets * 12 ways * ~30 cycles = ~3,000,000 cycles
- We cannot probe the entire LLC in a reasonable time, but probing one cache set is fast
- Our solution:
  - Probe one or a few cache sets at a time
  - Look for temporal patterns rather than spatial footprints

Demo

- L3-scan

Countermeasures

- Hardware
- System
- Software

Hardware countermeasure

- Don't share
- Hardware cache partitioning [DJL+12]
  - Intel Cache Allocation Technology
- Cache Randomisation [WL06]

System Countermeasures

- Limited sharing
  - Don't share memory
  - Don’t share cores
- Cache Colouring [BLRB94,SSCZ11]
  - STEALTHMEM [KPM12]
- CATalyst [LGY+16]
- CacheBar [ZRZ16]

Software Countermeasures

- Constant-time programming
  - No variable-time instructions
  - No secret-dependent flow control
  - No secret-dependent memory access

Non constant-time [YB14]

```
for (; i >= 0; i--)
    {
    word = scalar->d[i];
    while (mask)
        {
        if (word & mask)
            {
            if (!gf2m_Madd(group, &point->X, x1, z1, x2, z2, ctx)) goto err;
            if (!gf2m_Mdouble(group, x2, z2, ctx)) goto err;
            }
        else
            {
            if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx)) goto err;
            if (!gf2m_Mdouble(group, x1, z1, ctx)) goto err;
            }
        mask >>= 1;
        }
    mask = BN_TBIT;
    }
```

Constant-time

```
for (; i >= 0; i--) {
    word = scalar->d[i];
    while (mask) {
        BN_consttime_swap(word & mask, x1, x2, group->field.top);
        BN_consttime_swap(word & mask, z1, z2, group->field.top);
        if (!gf2m_Madd(group, &point->X, x2, z2, x1, z1, ctx))
            goto err;
        if (!gf2m_Mdouble(group, x1, z1, ctx))
            goto err;
        BN_consttime_swap(word & mask, x1, x2, group->field.top);
        BN_consttime_swap(word & mask, z1, z2, group->field.top);
        mask >>= 1;
    }
    mask = BN_TBIT;
}
```

Constant-time

```
    assert((condition & (condition - 1)) == 0);
    assert(sizeof(BN_ULONG) >= sizeof(int));

    condition = ((condition - 1) >> (BN_BITS2 - 1)) - 1;

    t = (a->top ^ b->top) & condition;
    a->top ^= t;
    b->top ^= t;

#define BN_CONSTTIME_SWAP(ind) \
        do { \
                t = (a->d[ind] ^ b->d[ind]) & condition; \
                a->d[ind] ^= t; \
                b->d[ind] ^= t; \
        } while (0)

    switch (nwords) {
    default:
        for (i = 10; i < nwords; i++)
            BN_CONSTTIME_SWAP(i);
        /* Fallthrough */
    case 10:
        BN_CONSTTIME_SWAP(9);   /* Fallthrough */
    case 9:
        BN_CONSTTIME_SWAP(8);   /* Fallthrough */
```

Bibliography

[ABF+15] T. Allan, B. B. Brumley, K. Falkner, J. van de Pol and Y. Yarom, "Amplifying Side Channels Through Performance Degradation", ePrint 2015/1141

[AK09] O. Acıiçmez, Ç. K. Koç, "Microarchitectural Attacks and Countermeasures," in Ç. K. Koç, ed. Cryptographic Engineering, Springer US, 2009

[AS07] O. Acıiçmez and J-P Seifert, "Cheap Hardware Parallelism Implies Cheap Security", FDTC 2007

[Ber05] D. J. Bernstein, "Cache-timing attacks on AES", http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2005

[DJL+12] L. Domnister, A. Jaleel, J, Loew, N. Abu-Ghazaleh and D. Ponomarev, "Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks", TACO 8(4), 2012

[GBK11] D. Gullasch, E. Bangerter and S. Krenn, "Cache Games -- Bringing Access-Based Cache Attacks on {AES} to Practice", IEEE S&P 2011

[GYCH16] Q. Ge, Y. Yarom, D. Cock and G. Heiser, "A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware", ePrint 2016/613

[GMWM16] D. Gruss, C. Maurice, K. Wagner and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack", DIMVA 2016

[GSM15] D. Gruss , R. Spreitzer and S. Mangard, "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches", USENIX Security 2015

[IES15] G. Irazoqui, T. Eisenbarth and B. Sunar, "S$A: A Shared Cache Attack that Works Across Cores and Defies VM Sandboxing – and its Application to AES", IEEE S&P 2015.

[IES16] G. Irazoqui, T. Eisenbarth and B. Sunar, "Cross processor cache attacks", AsiaCCS 2016

[IIES14] G. Irazoqui, M. S. Inci, T. Eisenberth and B. Sunar, "Wait a minute! A fast, Cross-VM Attack on AES", RAID 2014

[Lam73] B. W. Lampson, "A Note on the Confinement Problem", CACM 16(10), 1973

[LYG+15] F. Liu, Y. Yarom, Q. Ge, G. Heiser and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical", , IEEE S&P 2015

[MLN+15] C. Maurice, N. Le Scourance, C. Neumann, O. Heen and A. Francillon, "Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters", RAID 2015

[NS16] M. Neve and J-P Seifert, "Advances on access-driven cache attacks on AES," 13th SAC, 2006

Bibliography

[OST06] D. A. Osvik, A. Shamir and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES", CT-RSA 2006

[pag02] D. Page, "Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel," ePrint 2002/169

[Pag03] D. Page, "Defending against cache-based side-channel attacks," Information Security Technical Report, 8(1), 2003

[Per05] C. Percival, "Cache Missing for Fun and Profit", BSDCan, 2005

[PGM+16] P. Pessl, D. Gruss, C. Maurice, M. Schwarz and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks", USENIX Security 2016

[TOS10] E. Trome, D. A. Osvik, A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures", JoC 23(1), 2010

[TTMM02] Y. Tsunoo, E. Tsujihara, K. Minematsu and H. Miyauchi, "Cryptanalysis of Block Ciphers Implemented on Computers with Cache", International Symposium on Information Theory and Its Applications, Oct. 2002.

[WL06] Z. Wang and R. B. Lee, "Covert and Side Channels Due to Processor Architecture", ACSAC 2006

[WS12] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks", NoCS 2012.

[WXW12] Z. Wu, Z. Xu and H. Wang, "Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud", USENIX Security 2012

[YB14] Y. Yarom and N. Benger, "Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack", ePrint 2014/140

[YF14] Y. Yarom and K. Falkner, "FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack", USENIX Security 2014

[YGH16] Y. Yarom, D. Genkin and N. Heninger, "CacheBleed: A Timing Attack on OpenSSL Constant Time RSA", CHES 2016

[YGL+15] Y. Yarom, Q. Ge, F. Liu, R.B. Lee and G. Heiser, "Mapping the Intel Last-Level Cache", ePrint 2015/905

[ZXZ16] X. Zhang, Y. Xiao and Y. Zhang "Return-Oriented Flush-Reload Side Channels on ARM and Their Implications for Android Devices", CCS 2016