System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud

Taesoo Kim, Marcus Peinado, Gloria Mainar-Ruiz

MIT CSAIL *Microsoft Research*

Security is a big concern in cloud adoption

Security still the 'No. 1 obstacle' to cloud adoption

By Bobbie Johnson | Jun. 20, 2012, 10:17am PT | 2 Comments

International companies are still wary of cloud adoption because of concerns over data security and legal exposure, including worries about American government interference.

Speaking at GigaOM's Structure 2012 conference in San Francisco, Juergen Urbanski of Deutsche Telekom's T-Systems said that European customers, in particular, were wary of moving to the cloud because of their security fears.

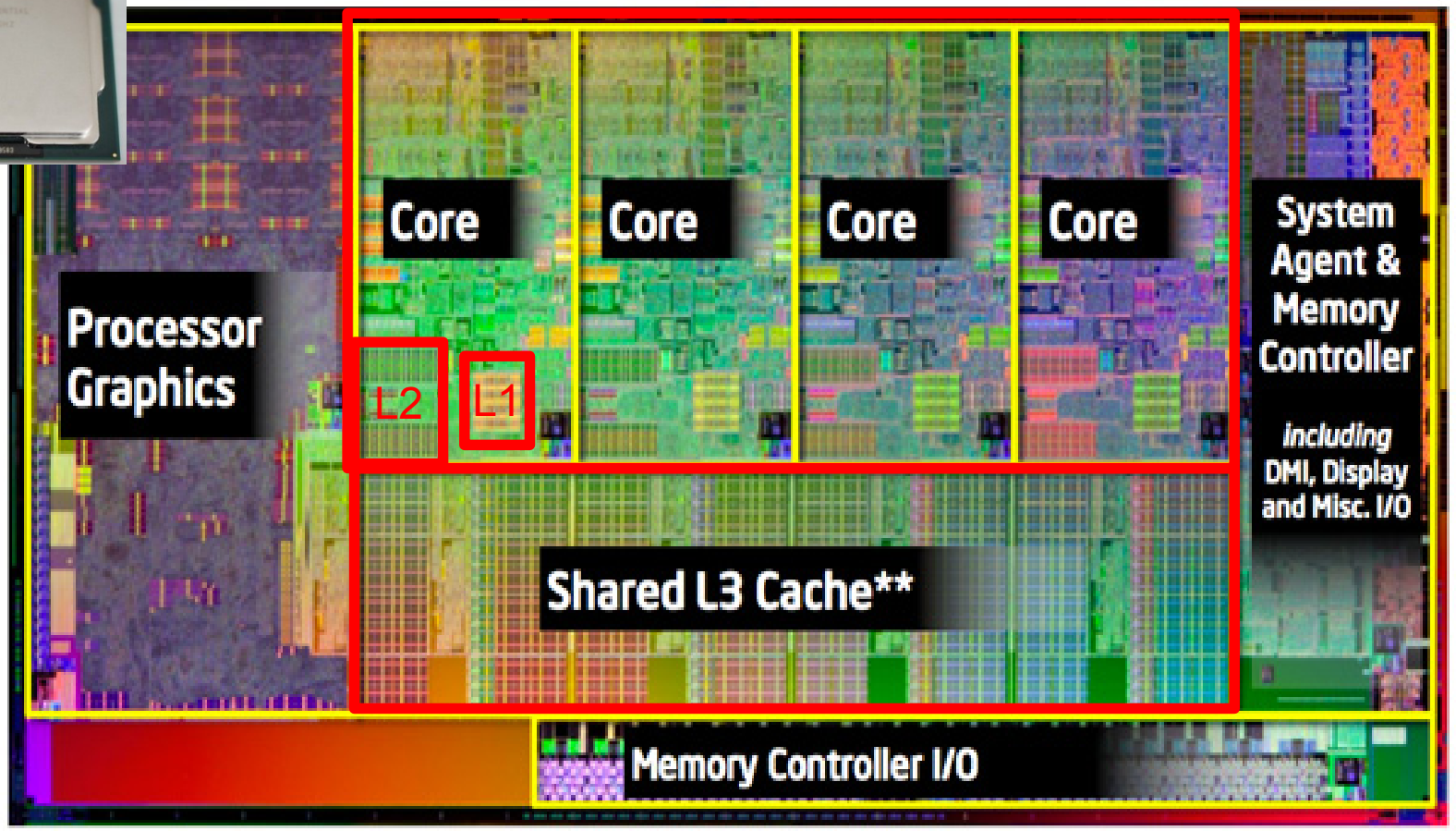
(L to R) Jurgen Urbanski of T-Systems, Tony Lucas of Flexiant, Steve Collen of Huawei
(c) 2012 Pinar Ozger pinar@pinarozger.com

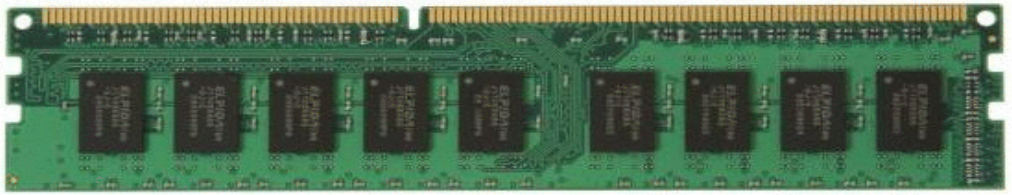
Why are cache-based side channel attacks important?

- CPU cache is the most **fine-grained shared resource** in the cloud environment

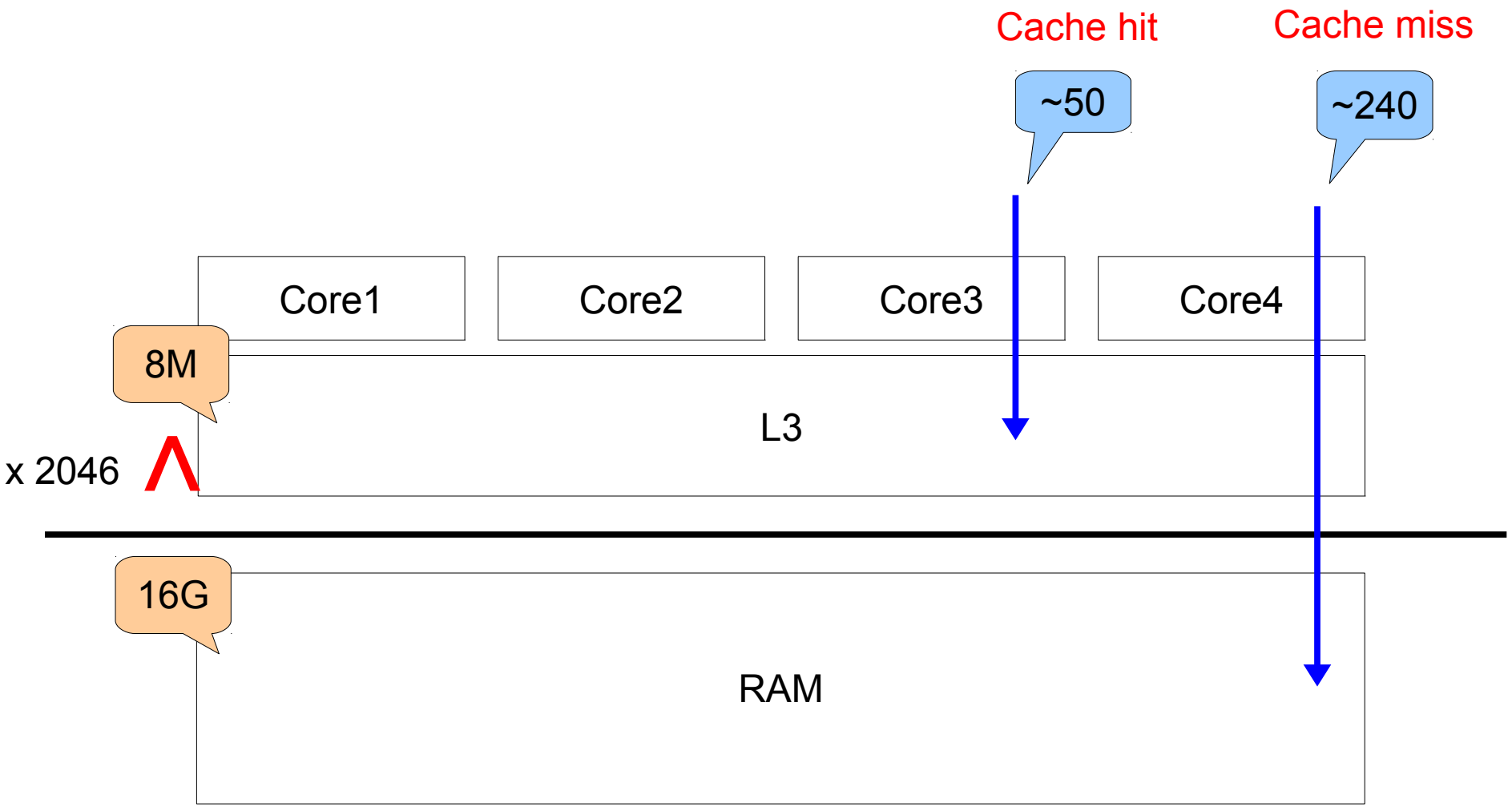
- Cache-based side channel attacks:
  - 2003 DES by Tsunoo et al. (with $2^{26.0}$ samples)
  - 2005 AES by Bernstein et al. (with $2^{18.9}$ samples)
  - 2005 RSA by Percival et al. (-)
  - …
  - 2011 AES by Gullasch et al. (with $2^{6.6}$ samples)

Background: CPU & Memory



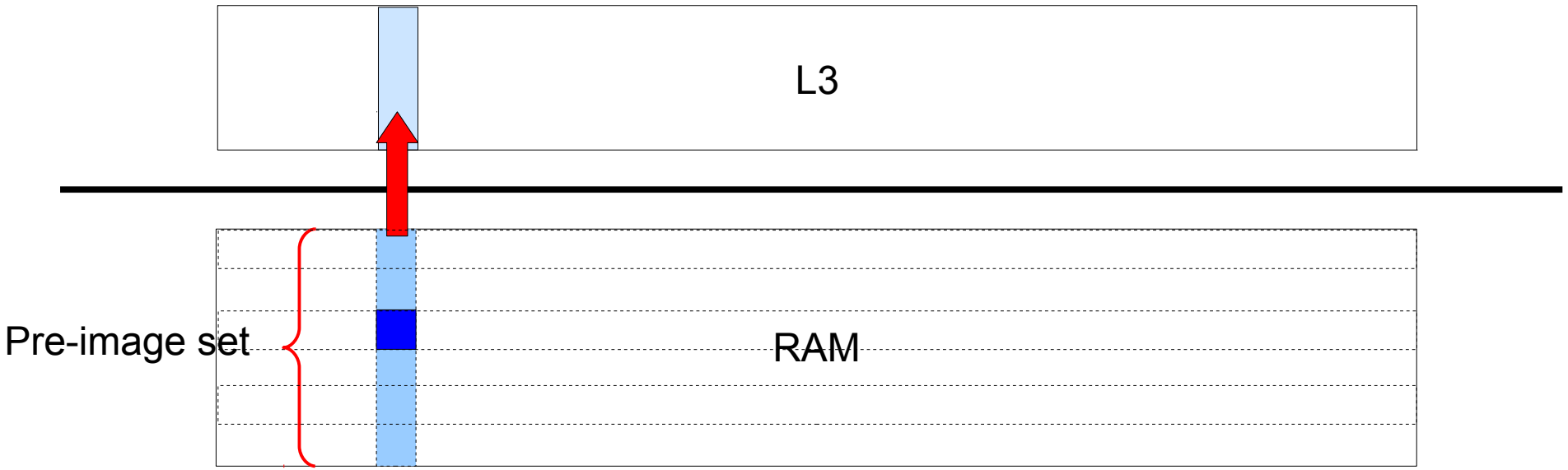


Background: cache structure



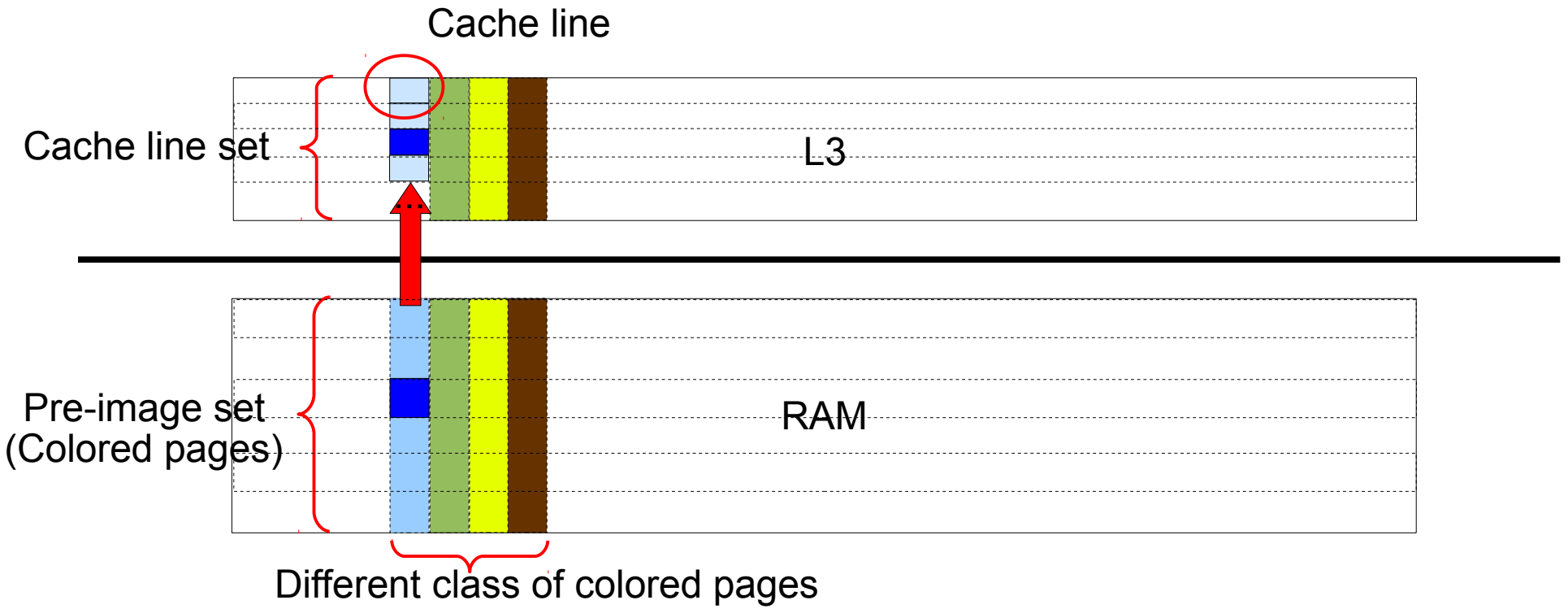
Background: cache terminologies

- **Pre-image set:** set of memory mapped into the same cache line

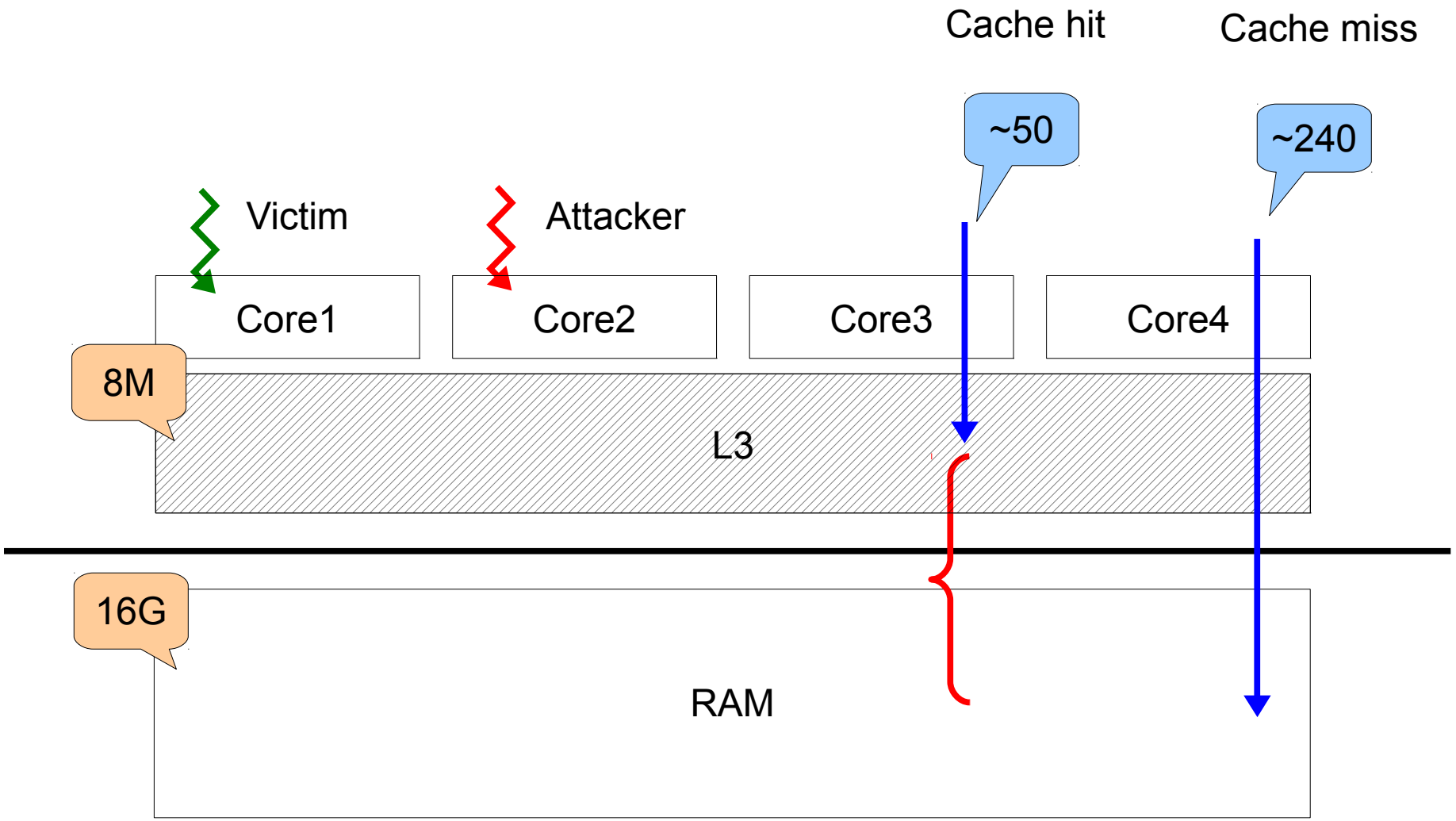


Background: cache terminologies

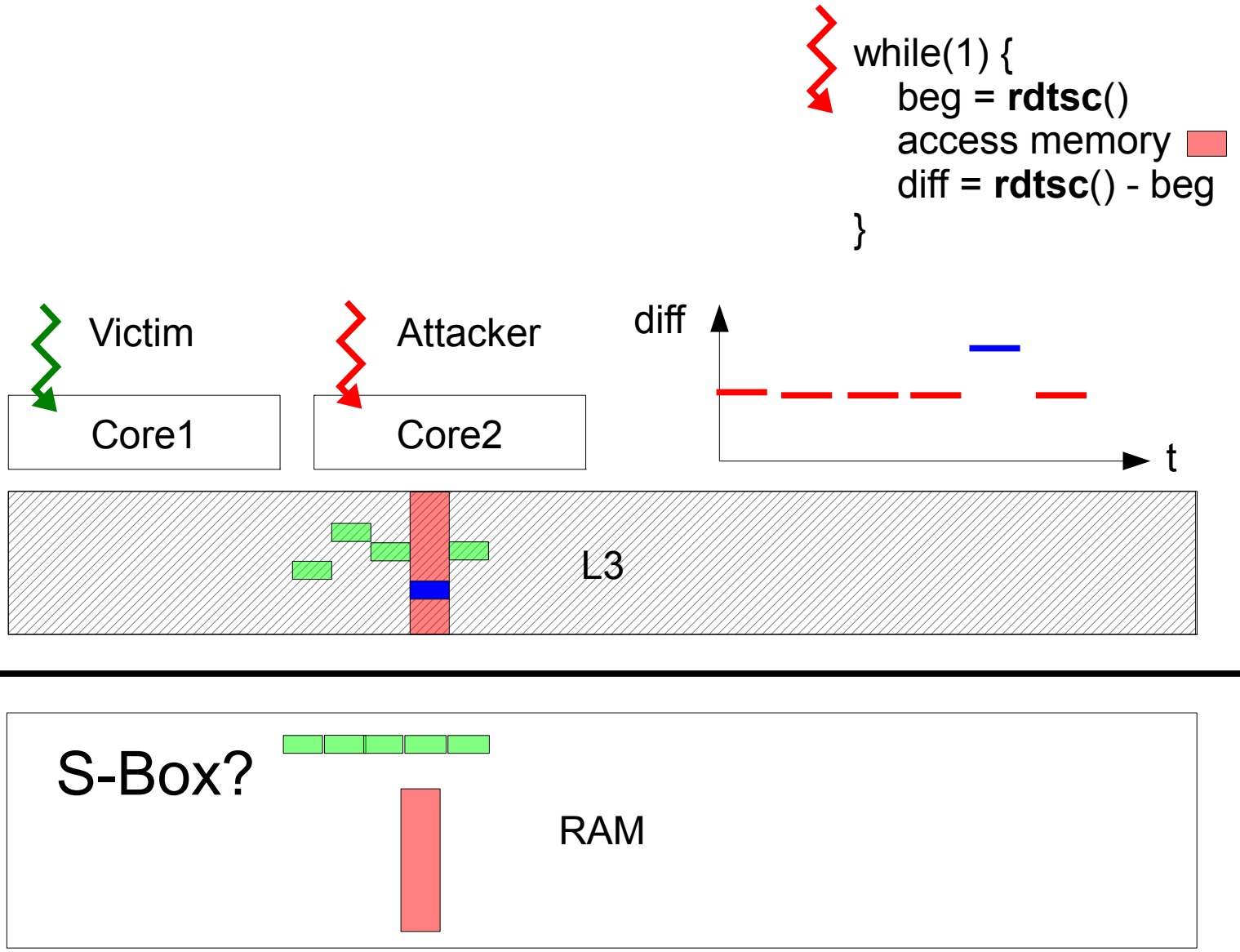
- **Pre-image set:** set of memory mapped into the same cache line
- **Cache line set**: set of cache lines mapped by the same pre-image set



Background: cache-based side channel



Cache-based side channel attacks (cache attacks)



Types of cache attacks

- **Time**-driven attacks

  : measure **access time** depending on states of cache

  - **Passive** time-driven attacks

    : measure total execution time of victim

  - **Active** time-driven attacks

    : manipulate states of cache

- **Trace**-driven attacks

  : **probe** which cache lines victim has **accessed**

→ Attackers should **co-locate** with a victim

Goal

To provide **cloud** tenants **a protection mechanism** against cache attacks:

- Active time-driven attacks
- Trace-driven attacks

But our solution still provides:

- Minimal **performance** overhead
- Compatible with **commodity hardware**

Idea: protect only sensitive data

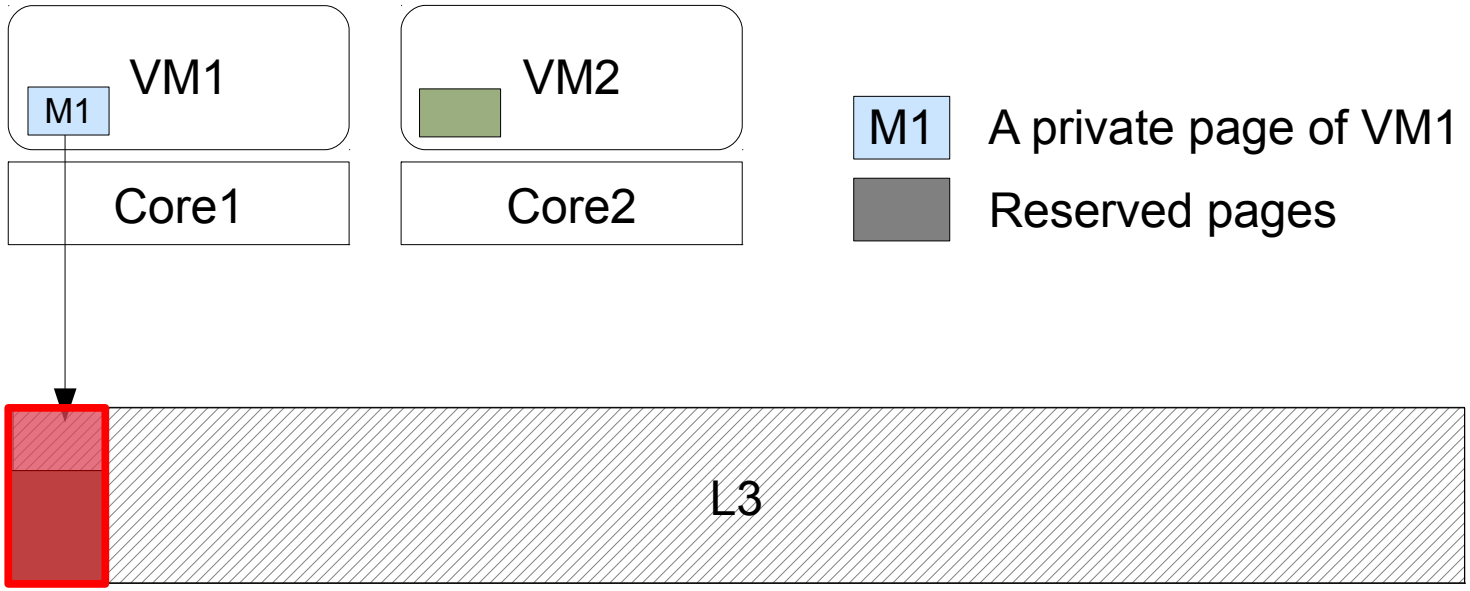
- Give **a private page** to each cloud tenant
  - No other tenants can cause cache interference

- Load sensitive data to the private page

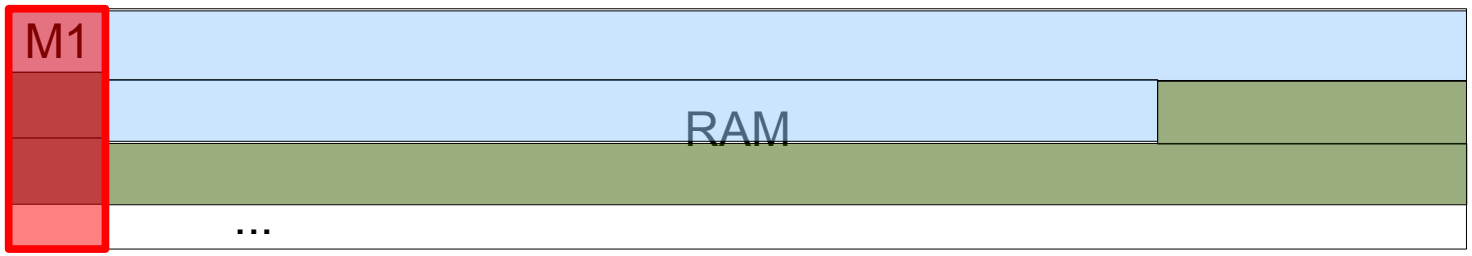
```
void *sm_alloc(size_t size)
void sm_free(void *ptr)
```

Strawman: construct a private page

- **Do not assign** pre-image sets of the private pages (same colored pages) to other VMs

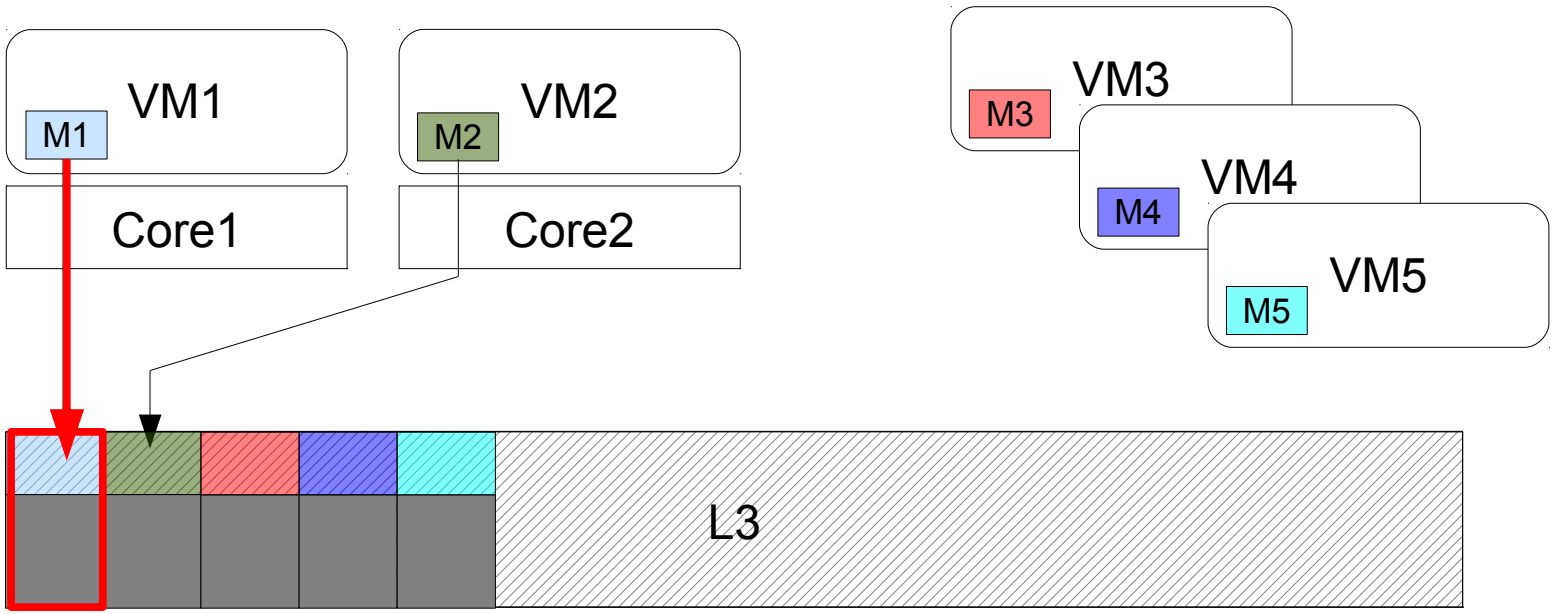


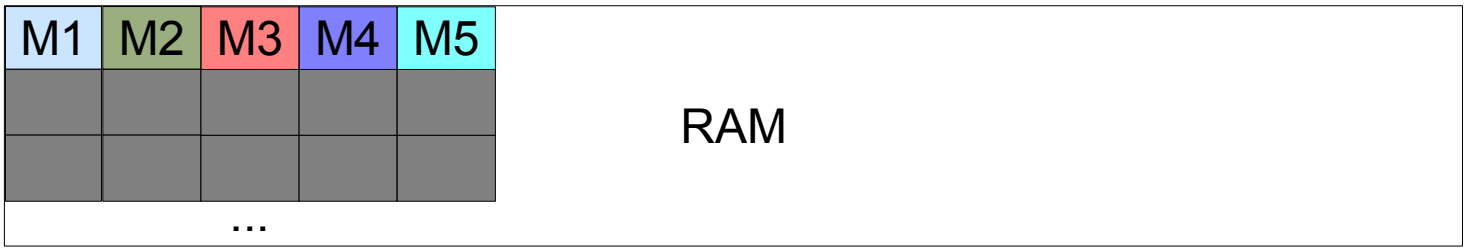
~1% Reserved



Strawman: assign a private page to each VM

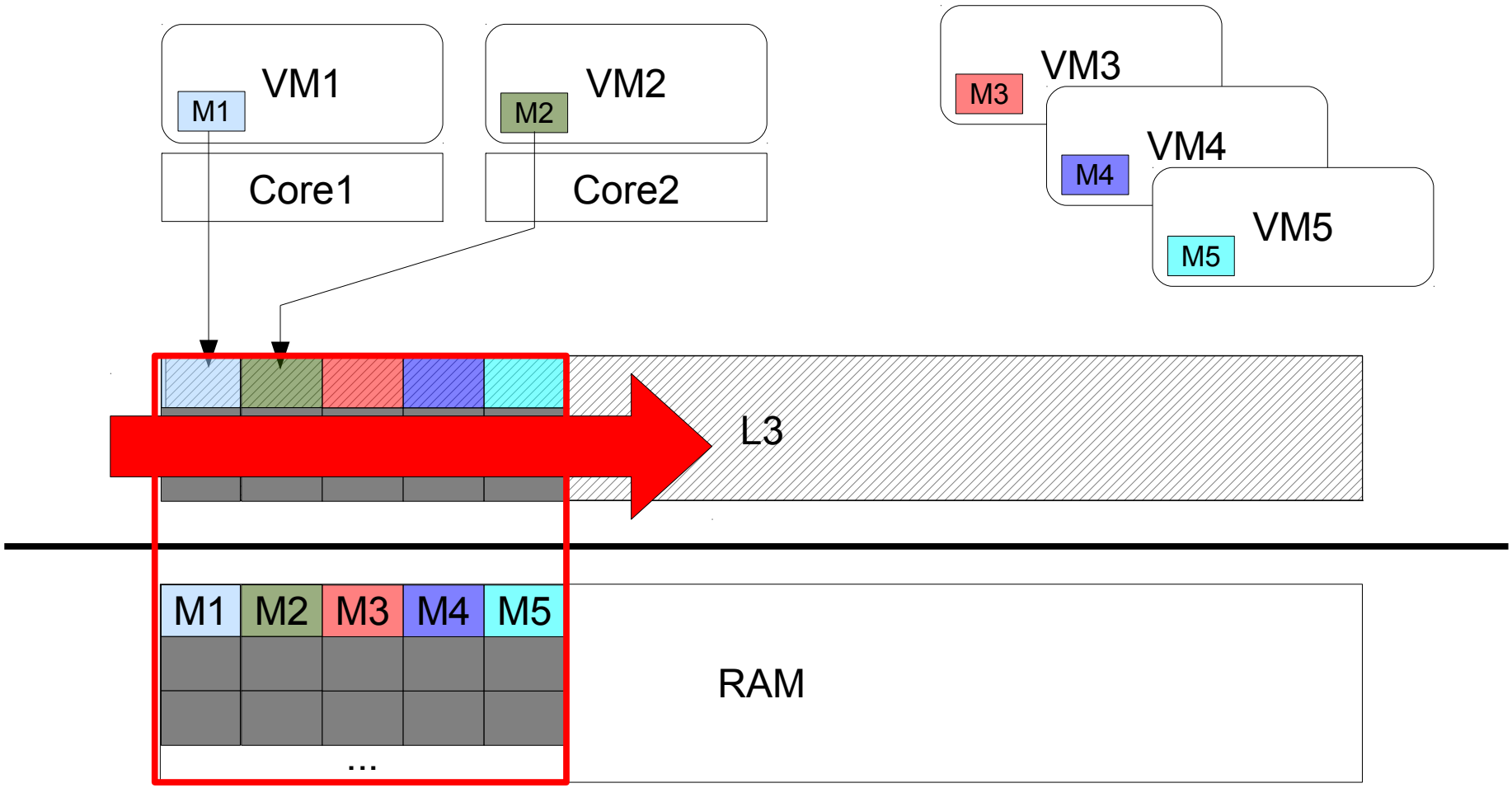
1. How to make sure that a private page stays in the cache?





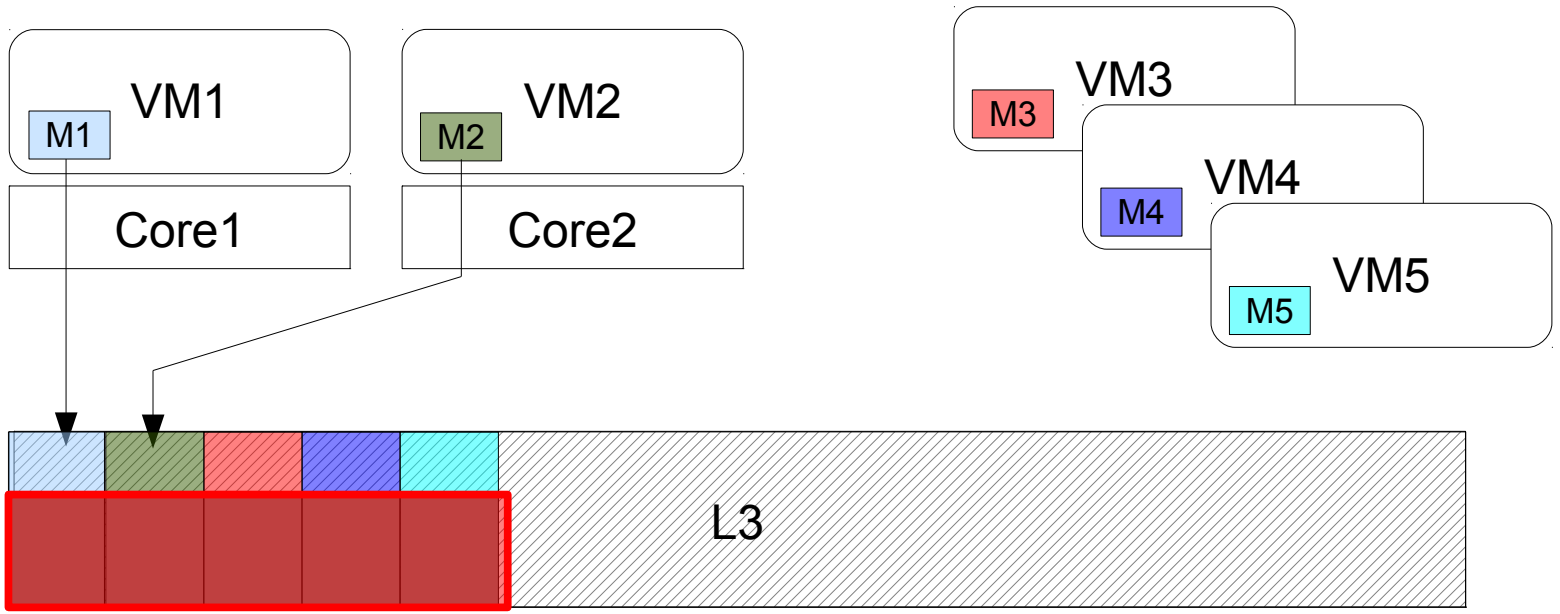
Strawman: assign a private page to each VM

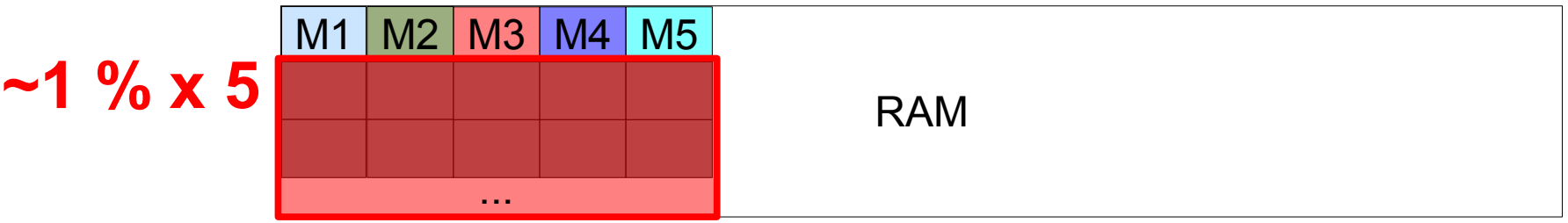
2. How to make it scalable if we increase the number of VMs?



Strawman: assign a private page to each VM

3. How to utilize the reserved regions?





Three challenges

1. How to make sure that a private page stays in the cache?

   → **Lock cache lines**

2. How to make it scalable if we increase the number of VMs?

   → **Assign** a private page **per core**

3. How to utilize the reserved regions?

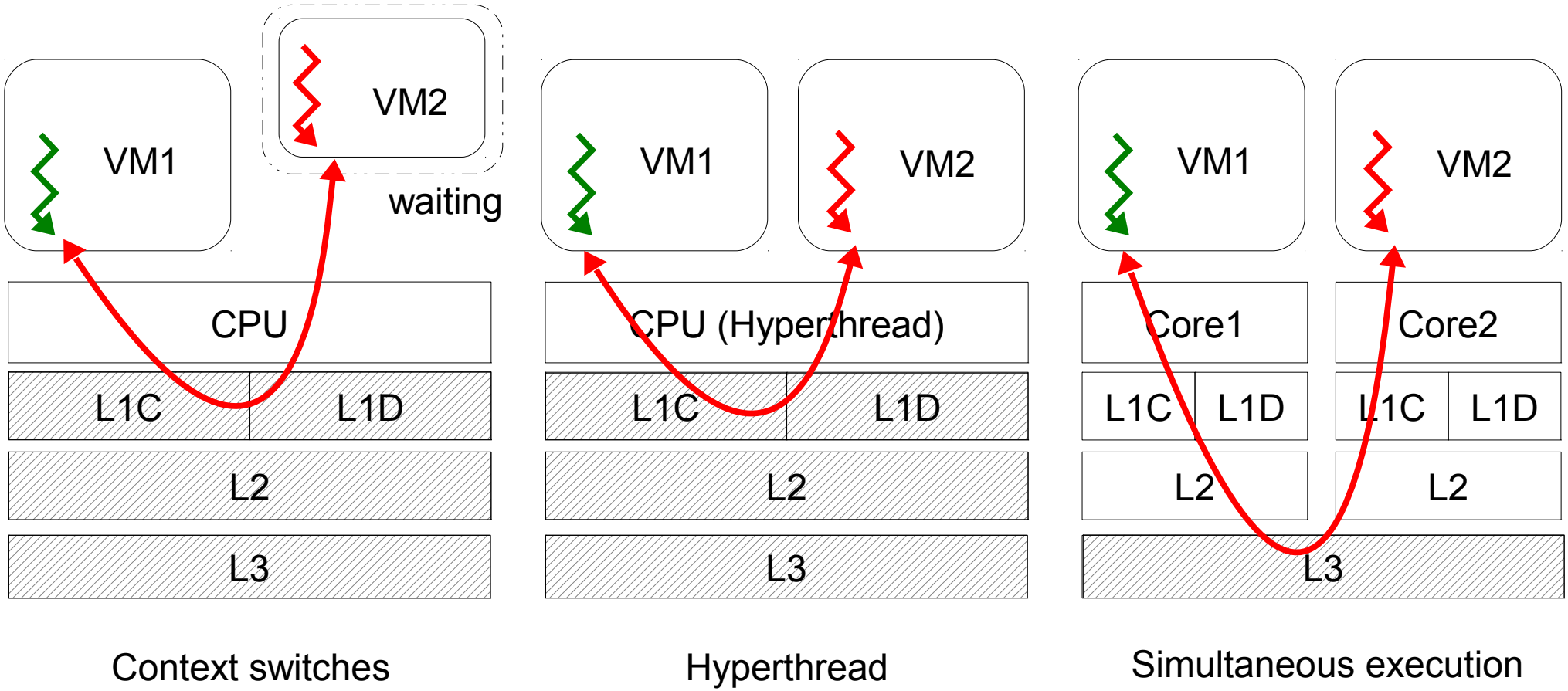
   → **Mediate accesses** on reserved regions

1. Locking cache lines

- **Locked**: **never evicted** from the cache

- **Inertia** property of cache (shared LLC):
  - An **eviction** only can happen when there is an **attempt to add** another item into the cache
  - Cache lines will **stay** still until we **access** an address that is **not in the cache**

Cache interference

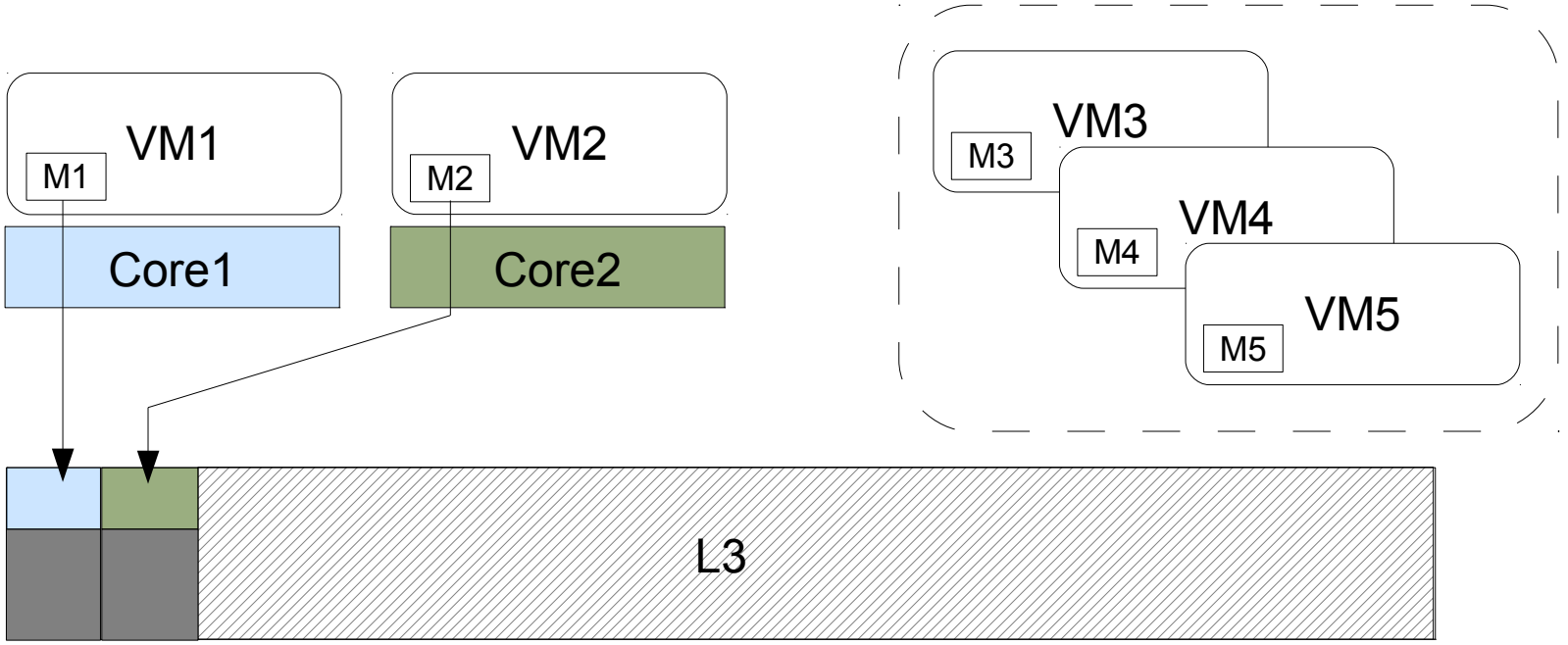


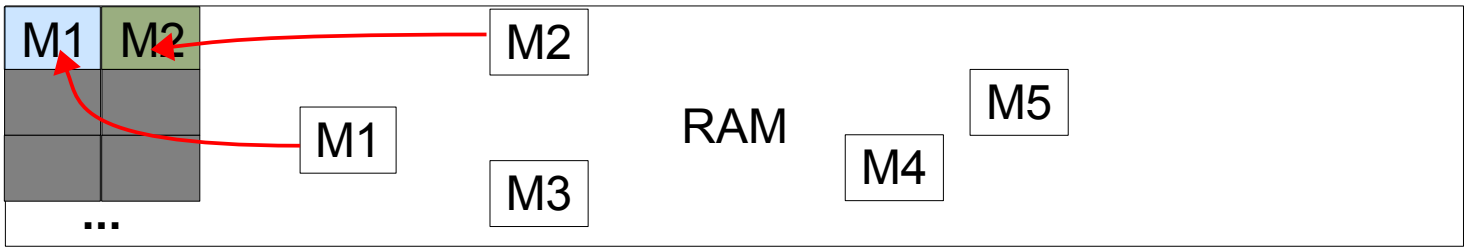
Keep cache lines locked

- Context switch:
  - **Reload** locked cache lines
- Hyperthread:
  - Force **gang** schedule
    (no two VMs run on the same core simultaneously)
- Simultaneous execution:
  - **Never map** pages that **collide** with private pages

2. Assign a private page per core

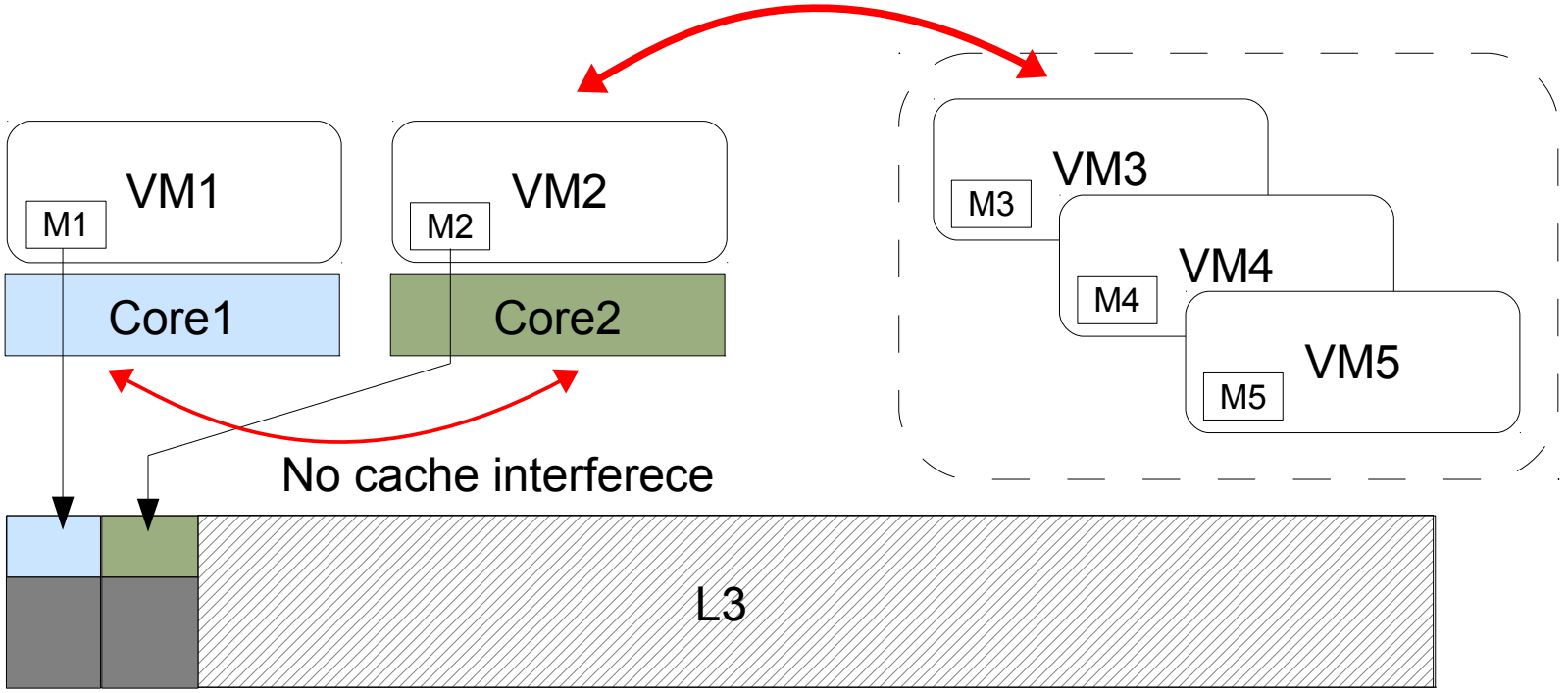
- Load a private page of active VM onto the private page of the core

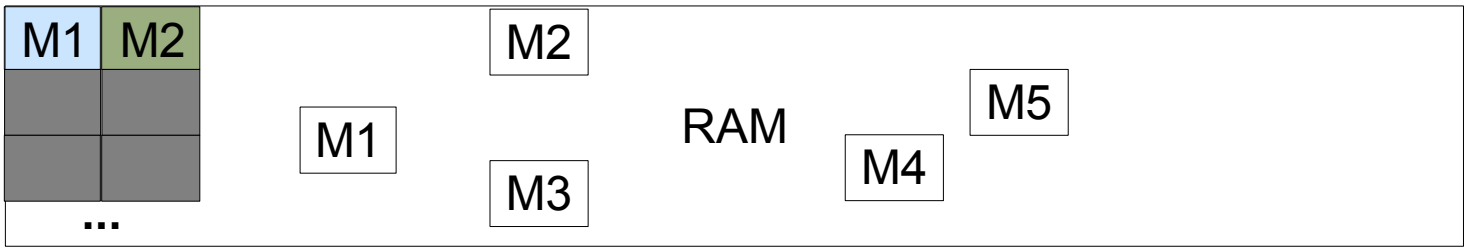




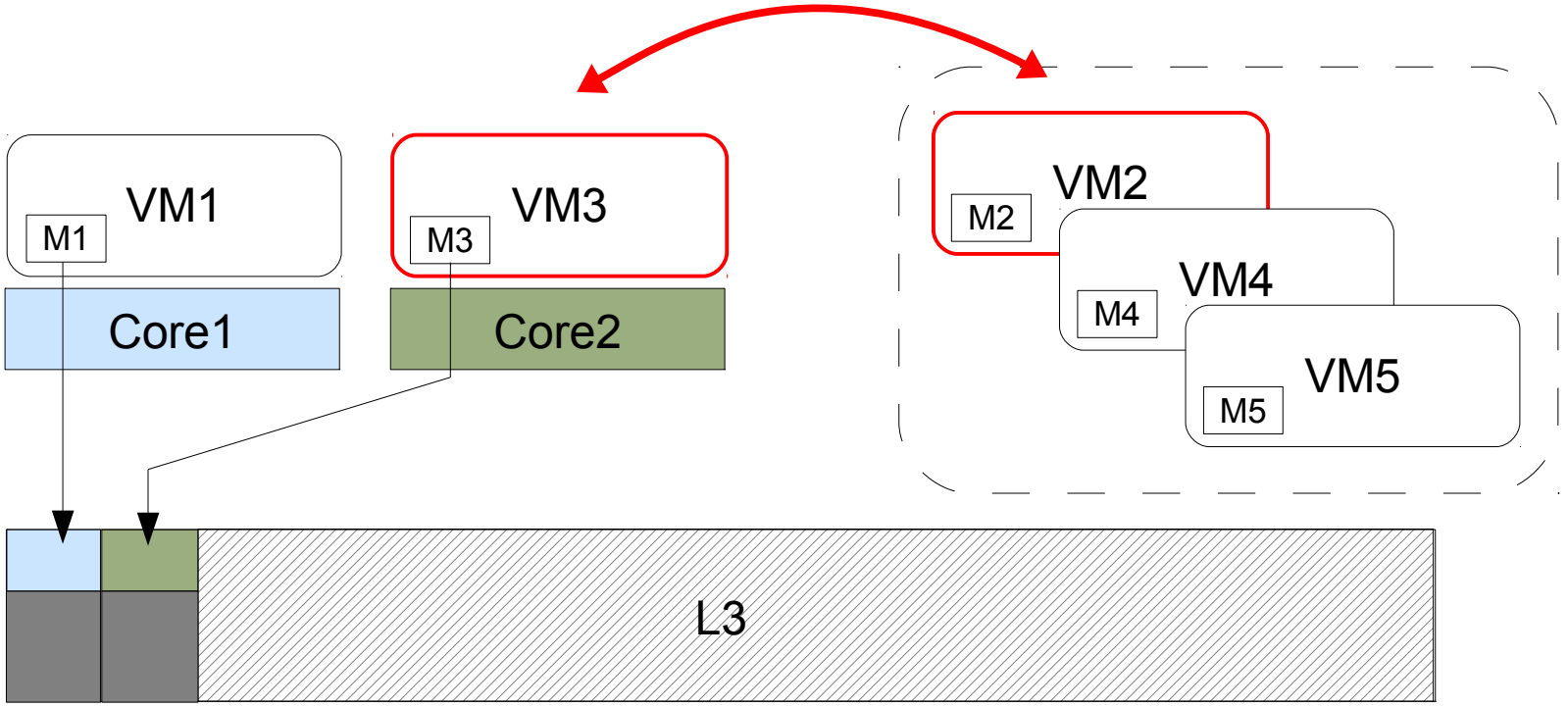
2. Assign a private page per core

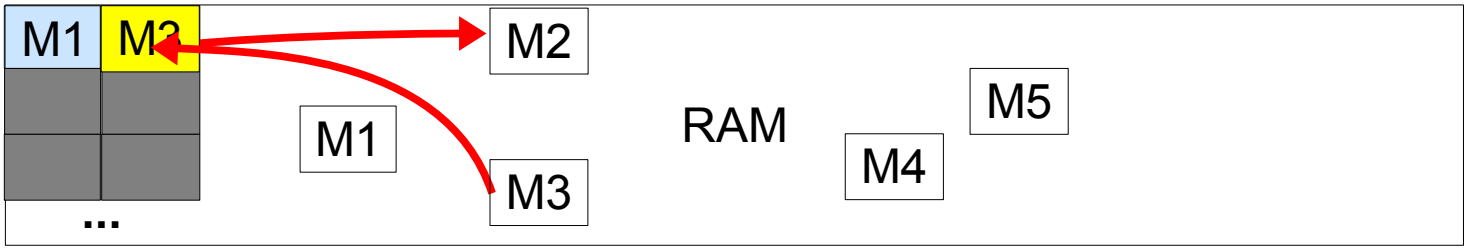
- No cache interference between running VMs



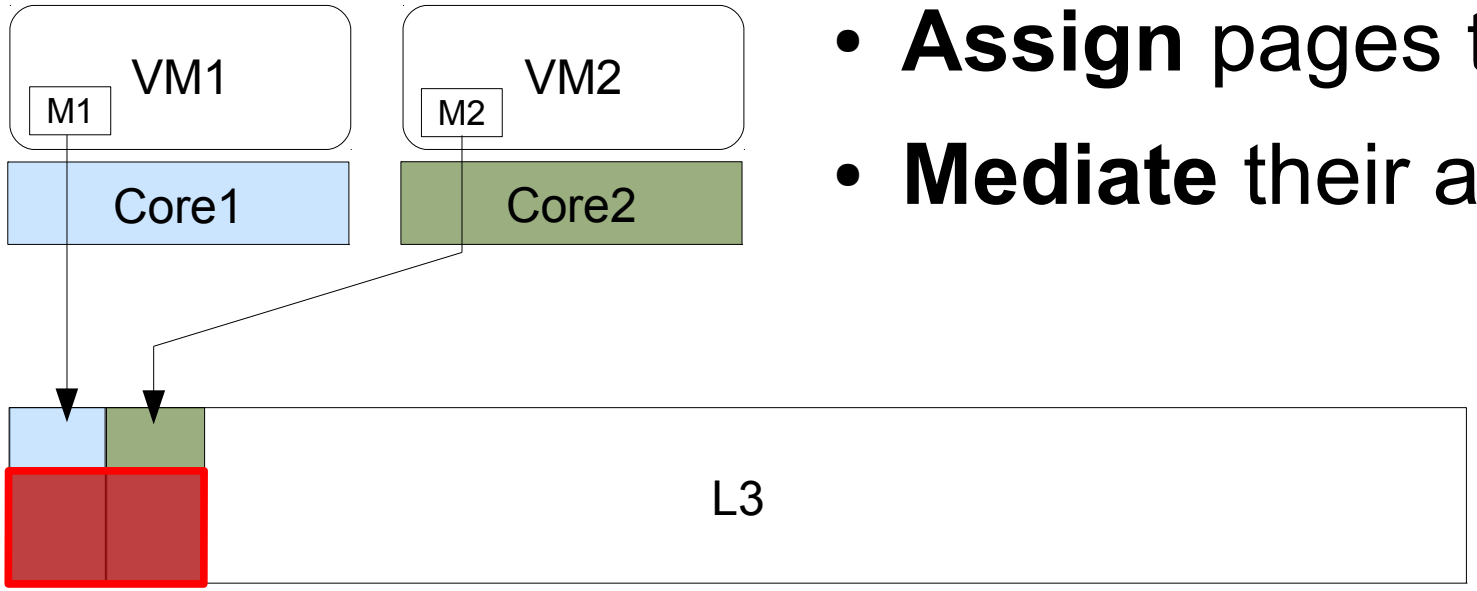


Save / load private pages on context switch

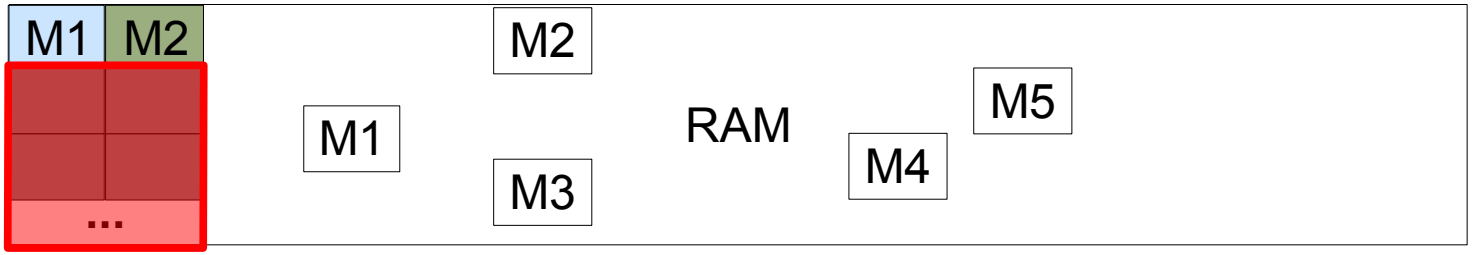




3. Utilize reserved regions

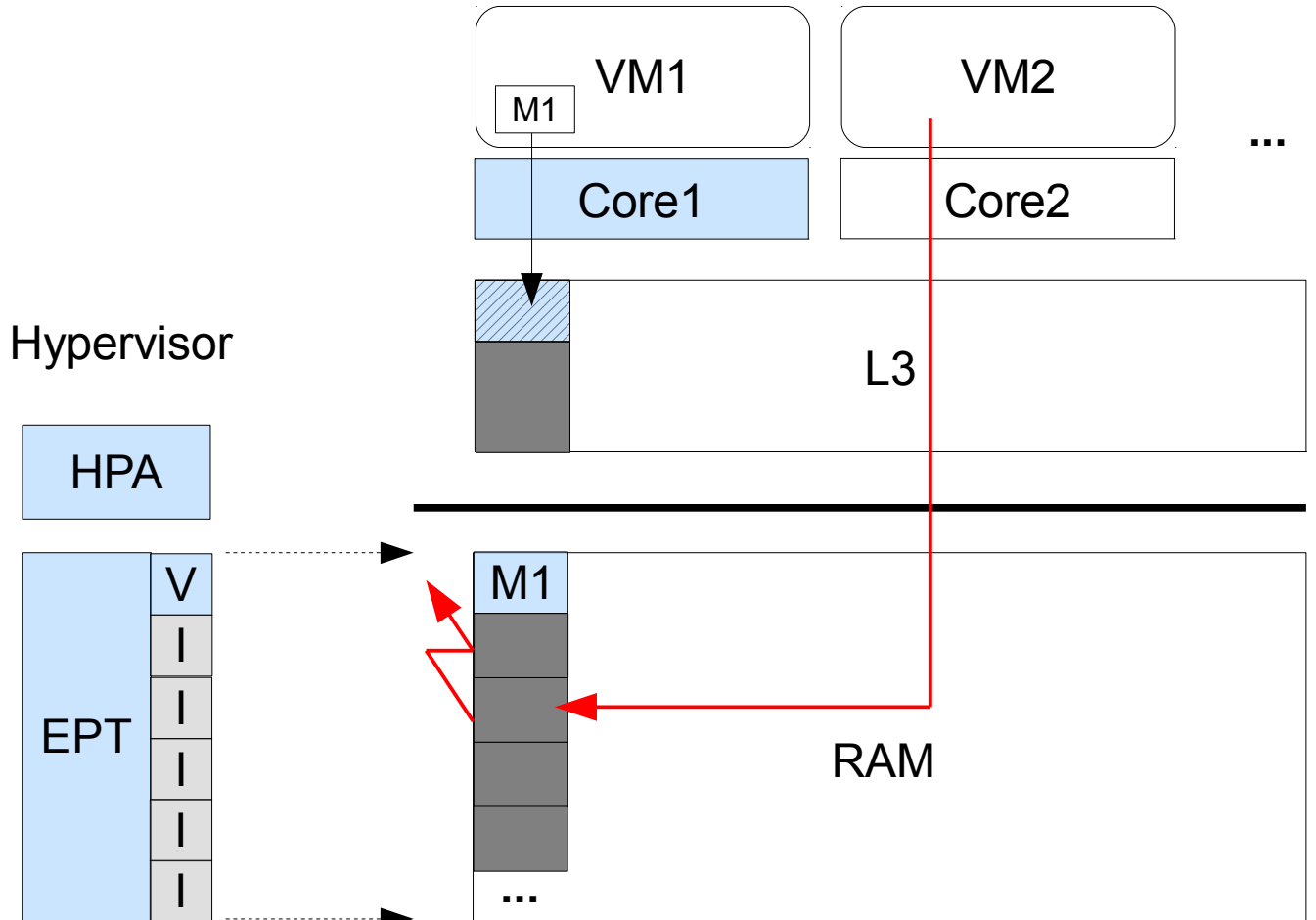


- **Assign** pages to VMs
- **Mediate** their accesses

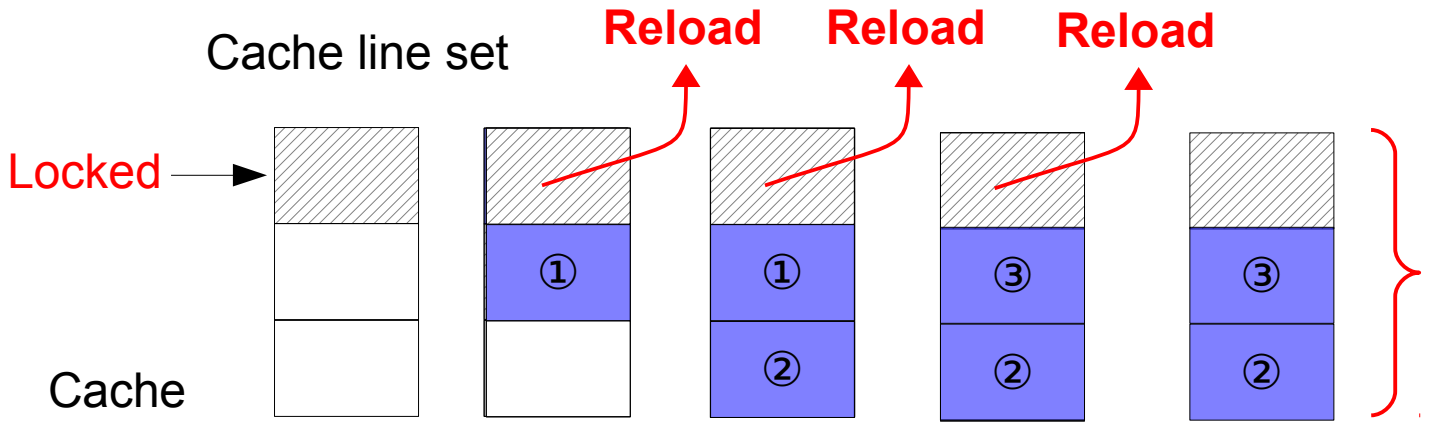


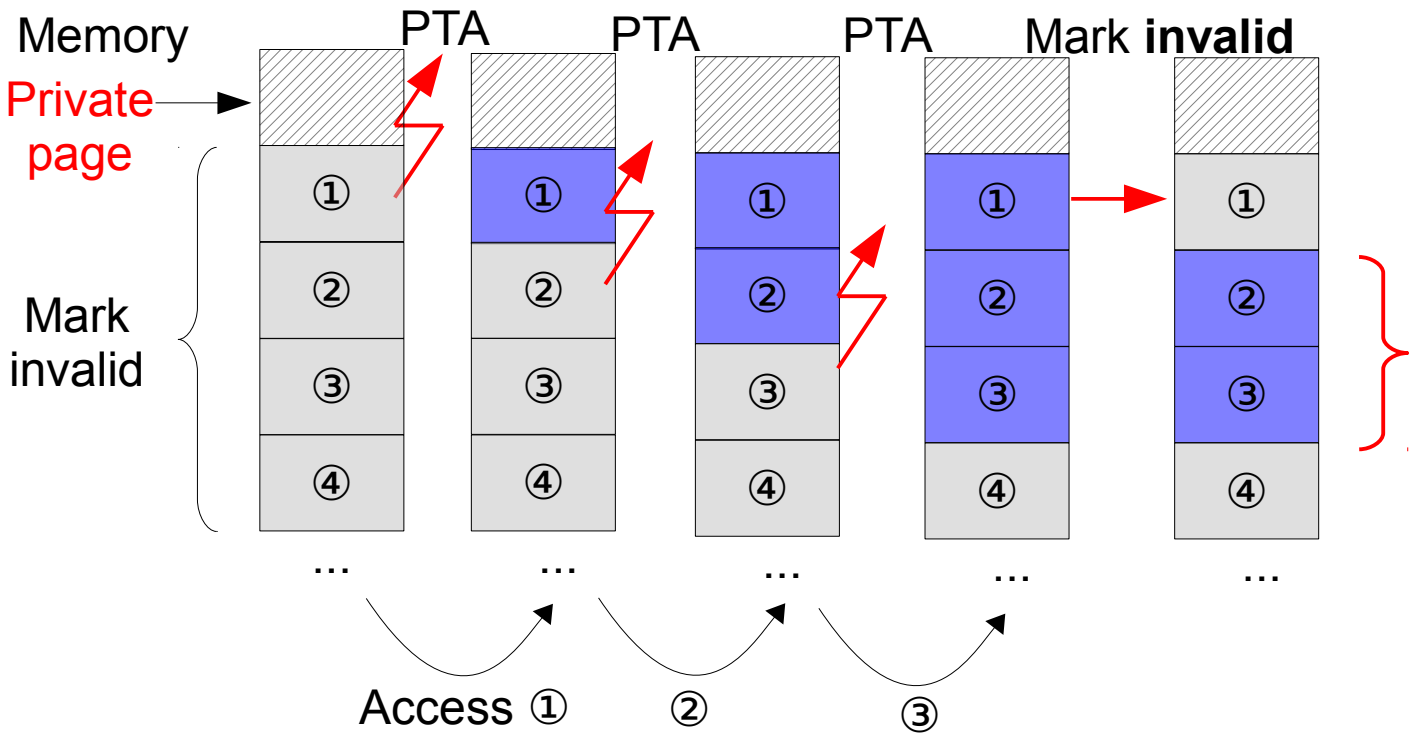
Page Table Alert (PTA)

- Mark **invalid** on reserved pages (pre-image sets)
- Mediate their accesses in the **page fault** handler



Handle Page Table Alert (PTA)





Summary of design

- Tenants use a private page for sensitive data
- Assign a private page per core
  - Use fixed amount of reserved memory
  - Load a private page of VM on one of the core
- Utilize reserved regions
  - Assign reserved regions to VMs as usual
  - Mediate their accesses with PTA

Implementation: *StealthMem*

- Host OS: Windows Server 2008 R2
  - ***bcdedit***: configure **reserved** area as **bad pages**
- Hypervisor: **HyperV**
  - Disable **large pages** (2MB/4MB)
  - Mediate ***invd***, ***wbinv*** instructions from VMs
  - Expose **a single private page** to VM

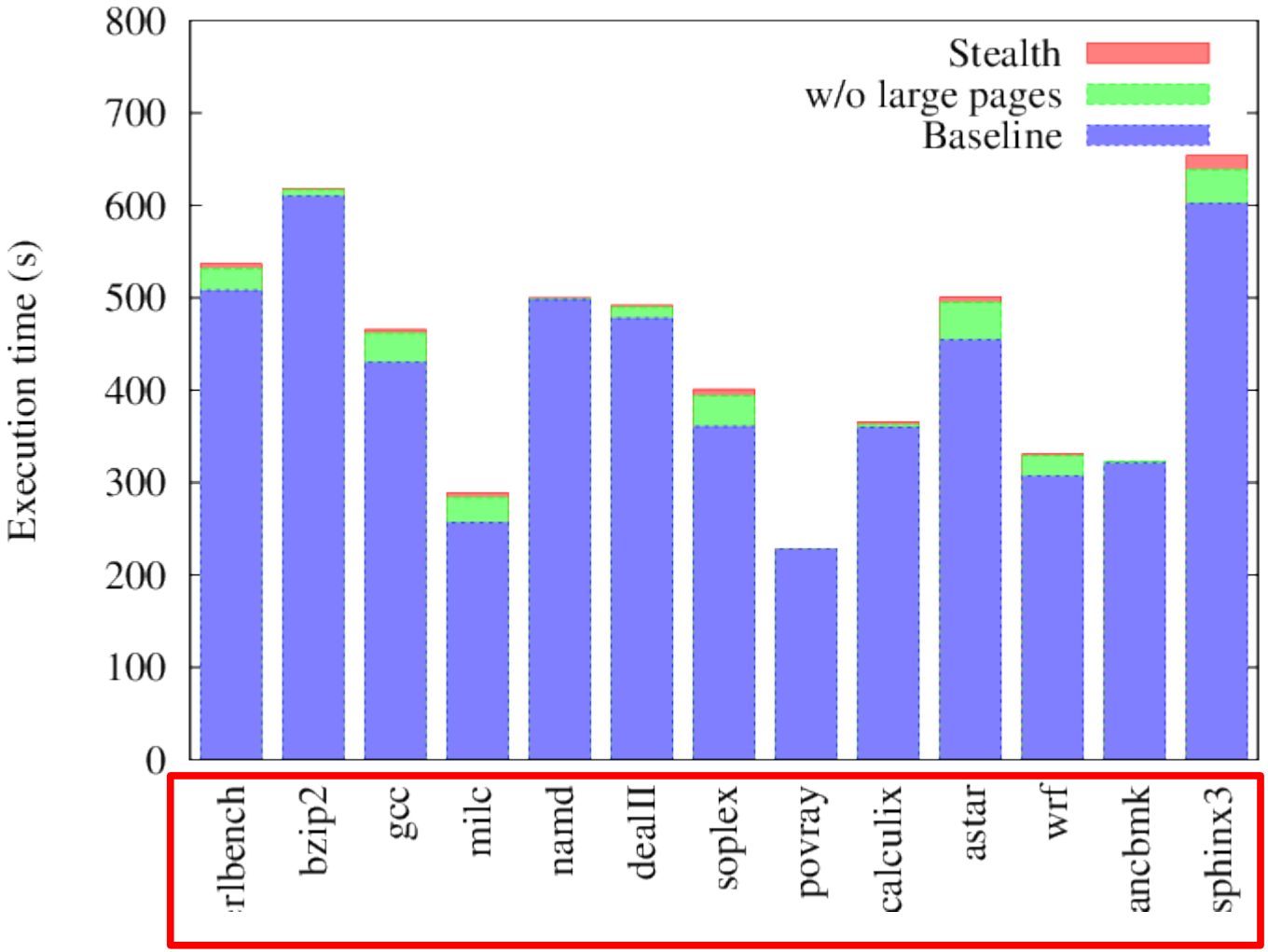
Component	Modified lines of code
Bootmgr/Winloader	**500** lines of C
HyperV	**5,000** lines of C

Evaluation

- How much overhead?
  - How does it compare with the stock HyperV?
  - How does it compare with other mechanisms?
  - How to understand overhead characteristics?

- How easy to adopt in existing applications?
  - How to secure popular block ciphers?

Overhead without large pages



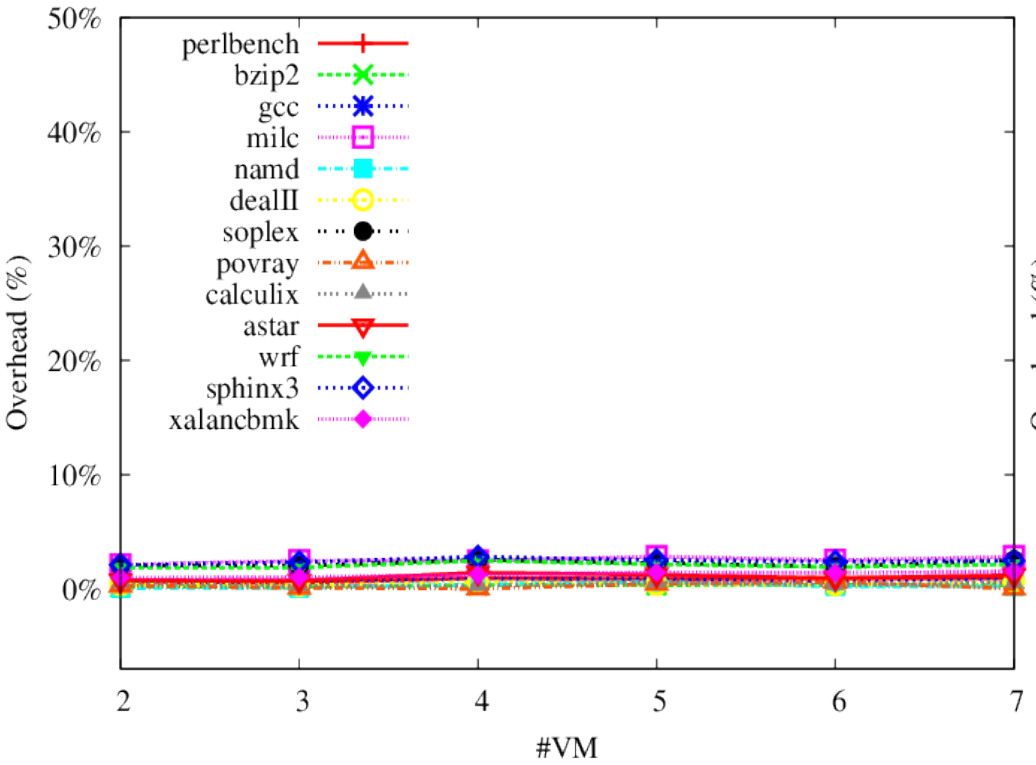
	Average
w/o large pages	-4.9%
StealthMem	-5.9%

Run **Spec2006**

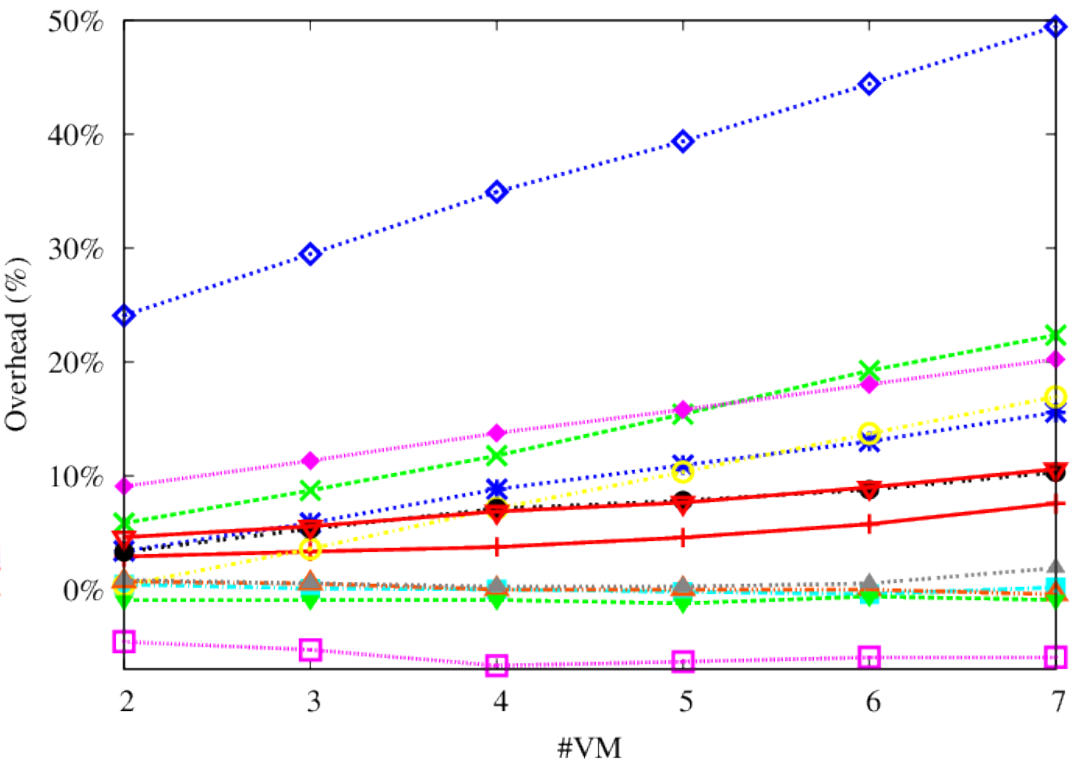
Compare with PageColoring

- **PageColoring**: statically divide caches per VM
- Run **SPEC2006** with **various #VM**

StealthMem



PageColoring

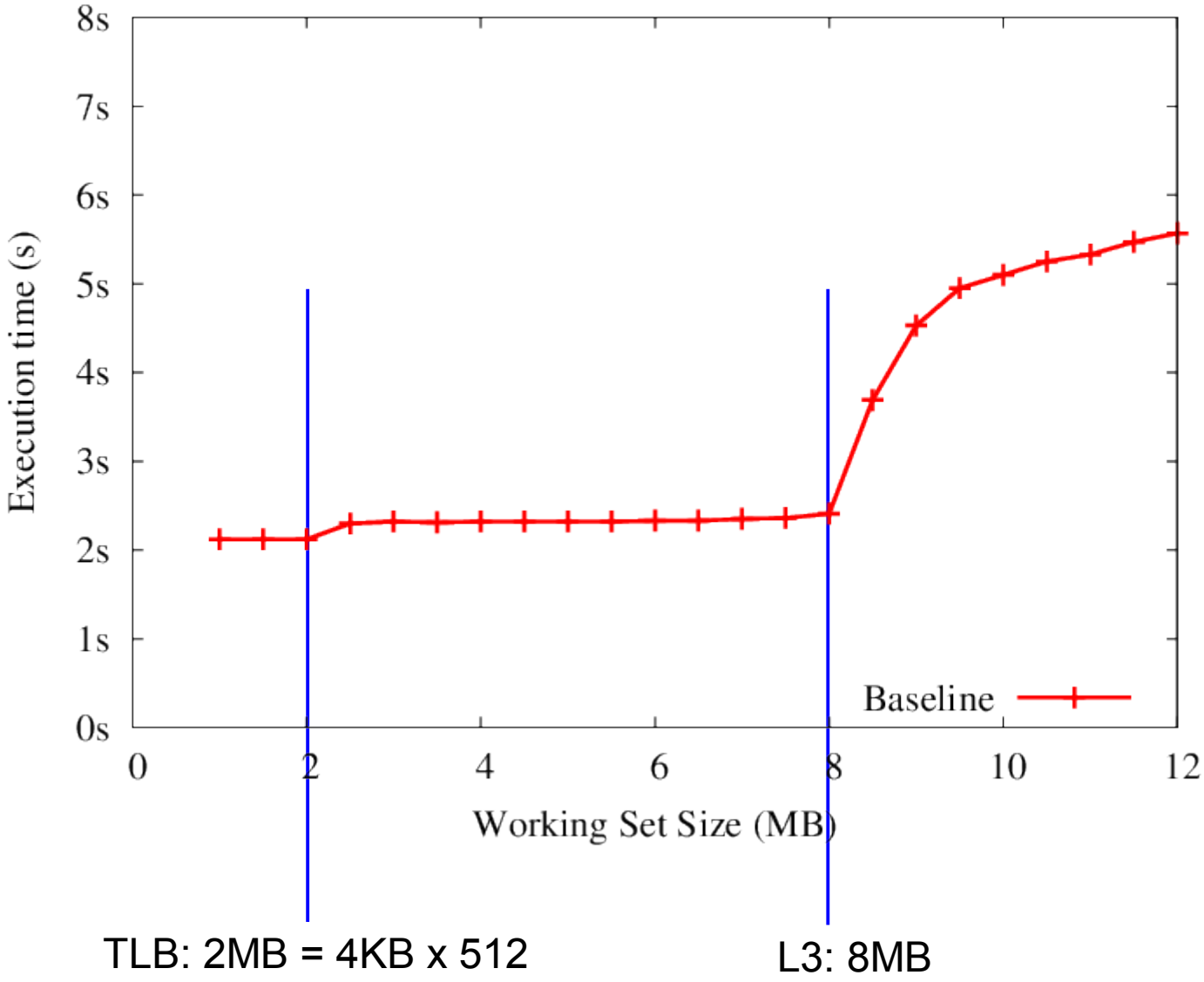


Microbench: overheads with various working sets

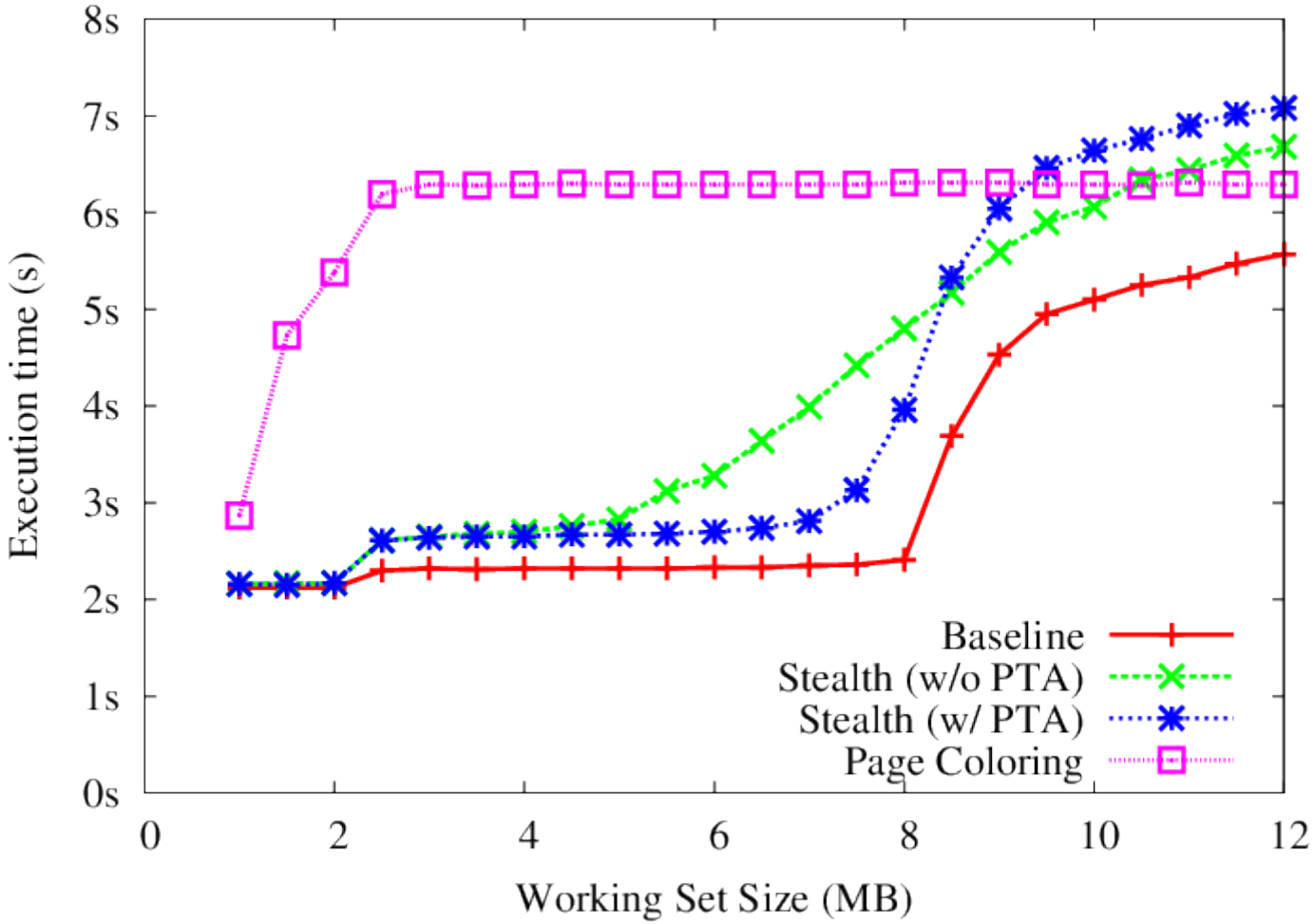
- Microbench:
  - Working set: vary array size between **1~12 MB**
  - Read array in **quasi-linear** fashion
  - Measure **execution time**

- Settings:
  - Each VM has a private page
  - **7 VMs**: one VM runs microbench while others idle
    - Baseline, PageColoring
    - StealthMem (**w/o** PTA): **do not utilize** reserved regions
    - StealthMem (**w/** PTA) : **utilize** reserved regions with PTA

Microbench: overheads with various working sets



Microbench: overheads with various working sets



Modifying existing applications

- e.g., modify **Blowfish** to use StealthMem

original

```
static unsigned long S[4][256];
```

modified

```
typedef unsigned long ULA[256];
static ULA *S;

<@initialization function>
S = sm_alloc(4*4*256);
```

Encryption	Size of S-box	LoC changes
DES	256 * 8 = 2 kB	**5** lines
AES	1024 * 4 = 4 kB	**34** lines
Blowfish	1024 * 4 = 4 kB	**3** lines

Overhead of secured ciphers

- Encryption **throughput** of DES / AES / Blowfish
  - Baseline: unmodified version
  - Stealth: **secured S-Box** with StealthMem

	A **small** buffer (50,000 bytes)			A **large** buffer (5,000,000 bytes)		
Cipher	Baseline	**Stealth**		Baseline	**Stealth**	
DES	60 MB/s	58	-3%	59 MB/s	57	-3%
AES	150 MB/s	143	-5%	142 MB/s	135	-5%
Blowfish	77 MB/s	75	-2%	75 MB/s	74	-2%

Related work

- Initial **abstraction** of StealthMem (by Erlingsson and Abadi)

- **Hardware-based**:
  - Obfuscating access patterns: PLcache, RPcache ...
  - Dynamic cache partitioning
  - App. specific hardware: AES encryption instruction

  → **StealthMem** works on **commodity hardware**

- **Software-based**:
  - Static partitioning: PageColoring
  - App. specific mitigation: reducing timing channels

  → **StealthMem** provides **flexible**, **better performance**

Conclusion

- **StealthMem**: an efficient system-level **protection** mechanism against **cache-based side channel attacks**

- **Implement** the abstraction of StealthMem
- **Three** new techniques:
  - **Locking** cache lines
  - **Assigning** a private page per core
  - **Mediating** access on the private pages with PTA