CSE 127: Computer Security

Process isolation, VMs and side channel

Deian Stefan

Slides adopted from Stefan Savage

Process Isolation

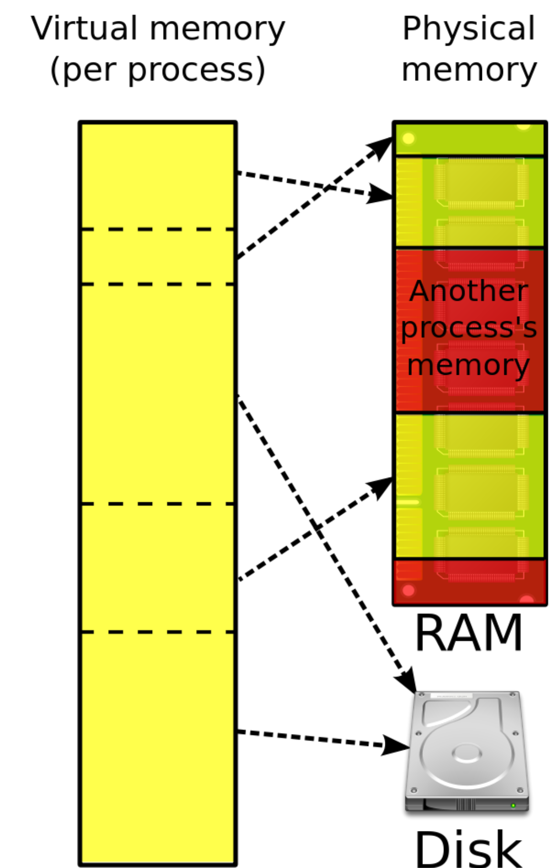
- Process boundary is a trust boundary
  - Any inter-process interface is part of the attack surface
- How are individual processes isolated from each other?
  -

Process Isolation

- Process boundary is a trust boundary
  - ➤ Any inter-process interface is part of the attack surface
- How are individual processes isolated from each other?
  - ➤ Each process gets its own virtual address space, managed by the operating system

Virtual Memory

- Memory addresses used by processes are virtual addresses
- Who maps VAs to PAs?
  - ➤ The operating system + MMU



https://en.wikipedia.org/wiki/Virtual_memory#/media/File:Virtual_memory.svg

How do we get isolation?

Virtualized view of memory with limited visibility/
access to the underlying memory space

How do we translate VAs?

- Using 64-bit ARM architecture as an example...
- How to practically map arbitrary 64bit addresses?
  - 64 bits * 2^{64} (128 exabytes) to store any possible mapping

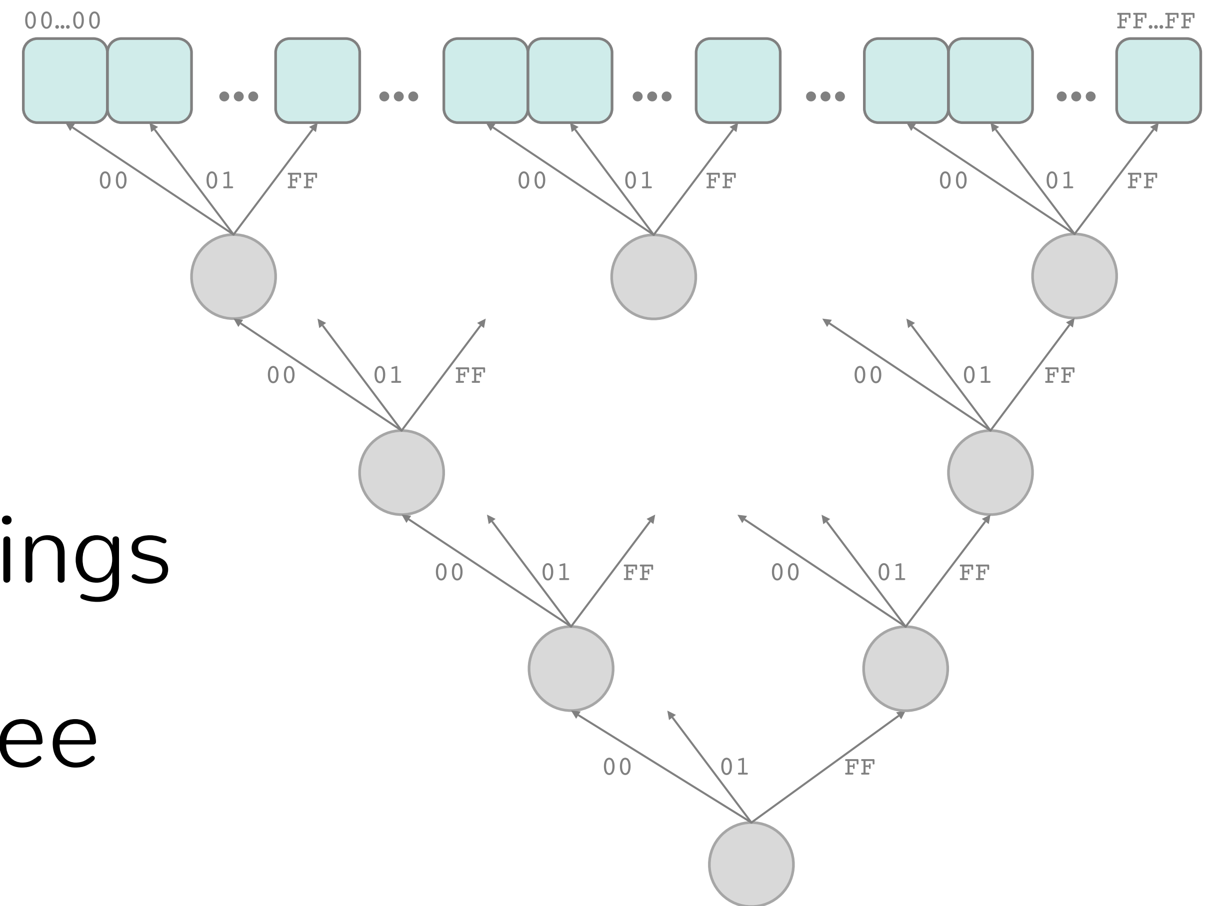
Address Translation

- Page: basic unit of translation
  - Usually 4KB
- How many page mappings?
  -

Address Translation

- Page: basic unit of translation
  - Usually 4KB
- How many page mappings?
  - 52 bits * 2^{52} (208 petabytes)

So what do we actually do?



Multi-level Page Tables

- Sparse tree of page mappings
- Use VA as path through tree
- Leaf nodes store PAs
- Where is the root kept?

What are the nodes of the trees?

- Page tables!
  - Data structures used to store address mapping
- Each table (node) is:
  - Array of translation descriptors
  - What's the size of a page table?

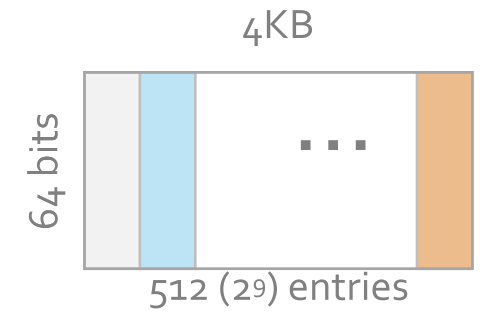
How do we use these tables?

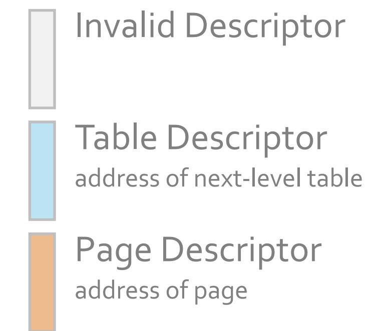
- Organized into a tree of descriptors
  - Iteratively resolve n bits of address at a time
  - Each descriptor is either
    - Page descriptor (leaf node)

How do we use these tables?

- Organized into a tree of descriptors
  - Iteratively resolve n bits of address at a time
  - Each descriptor is either
    - Page descriptor (leaf node)
    - Table descriptor (internal node)

Page table walk





Translation Table
Base Register

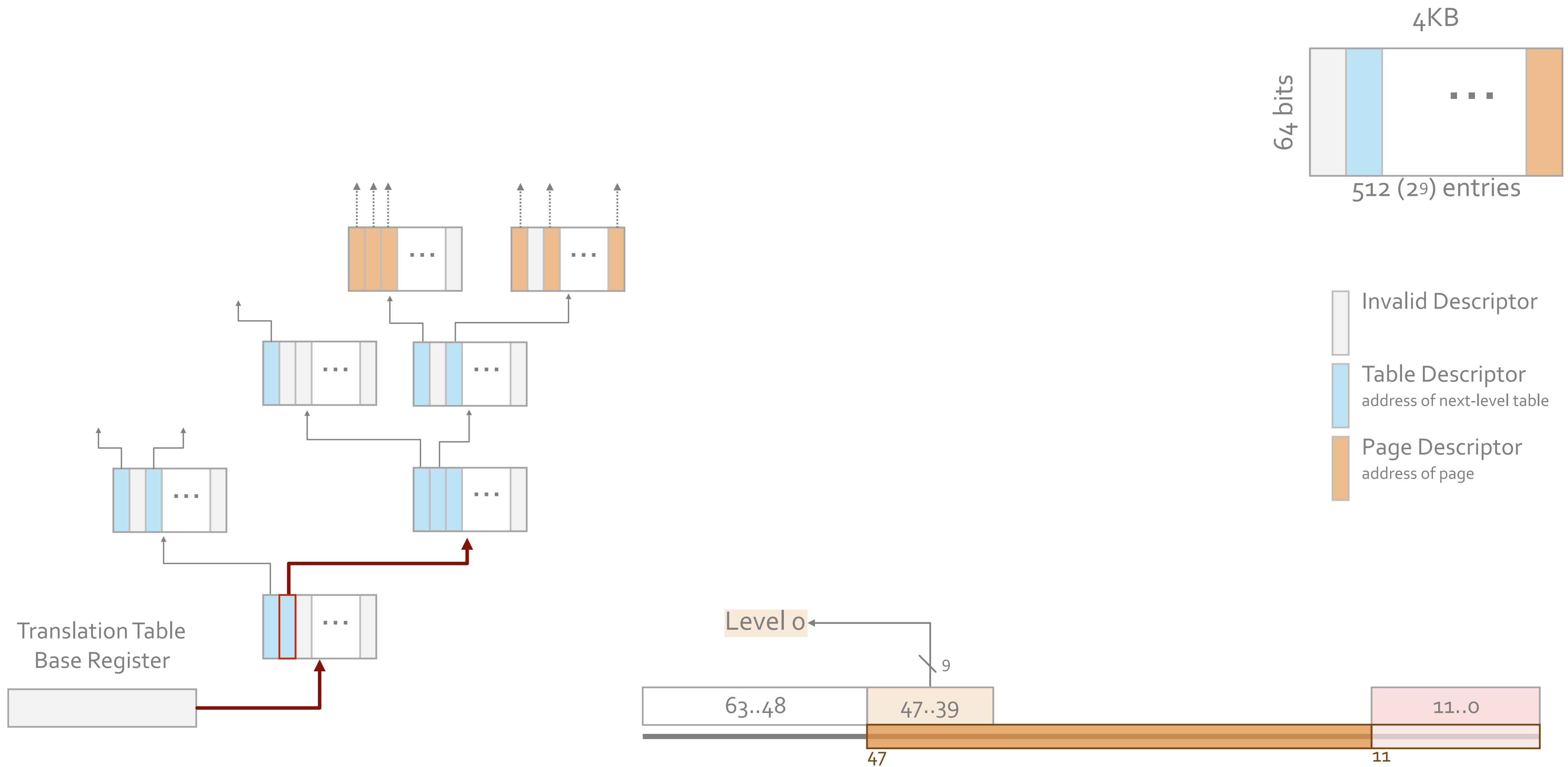
63..48

47

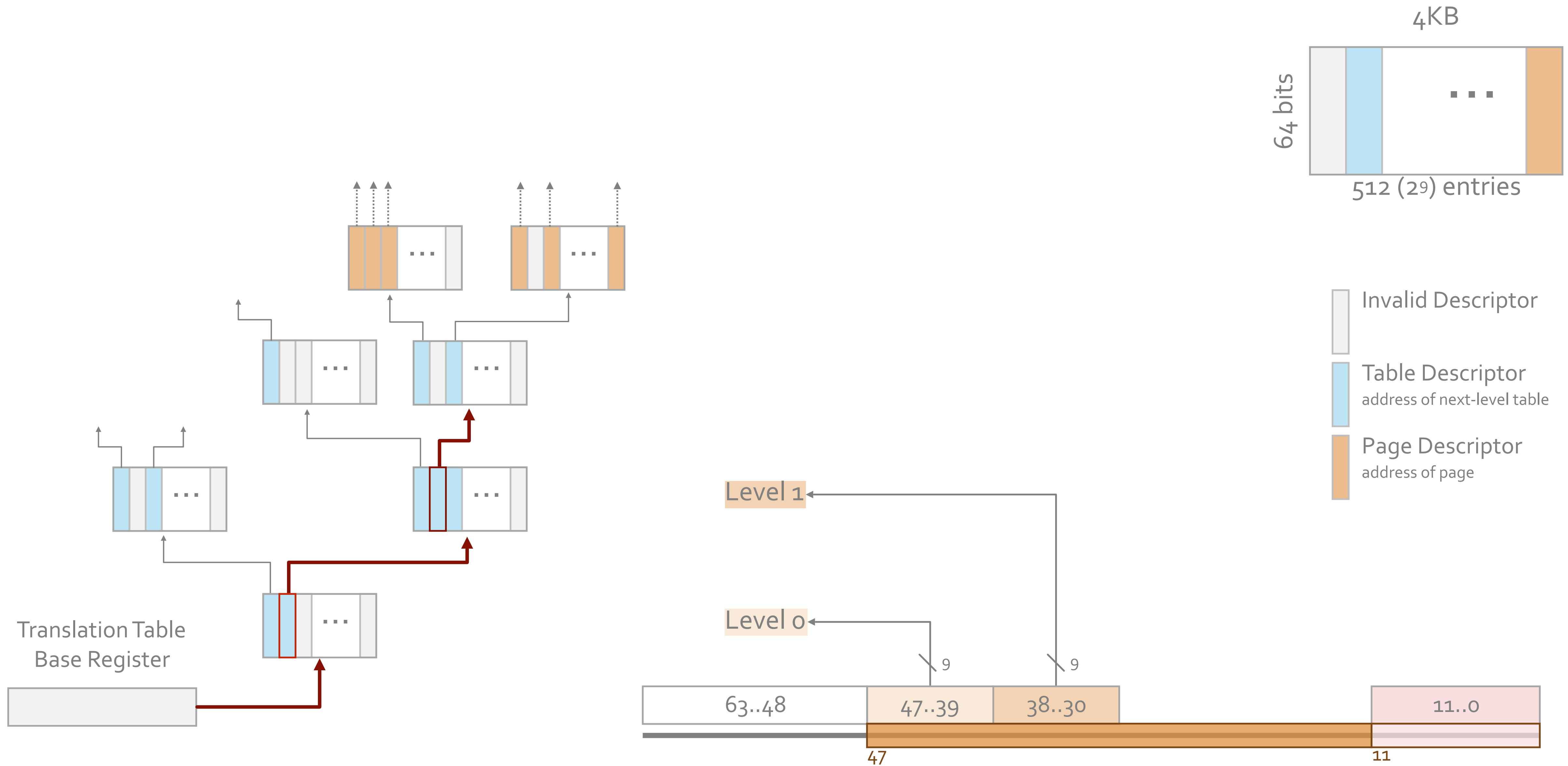
11..0

11

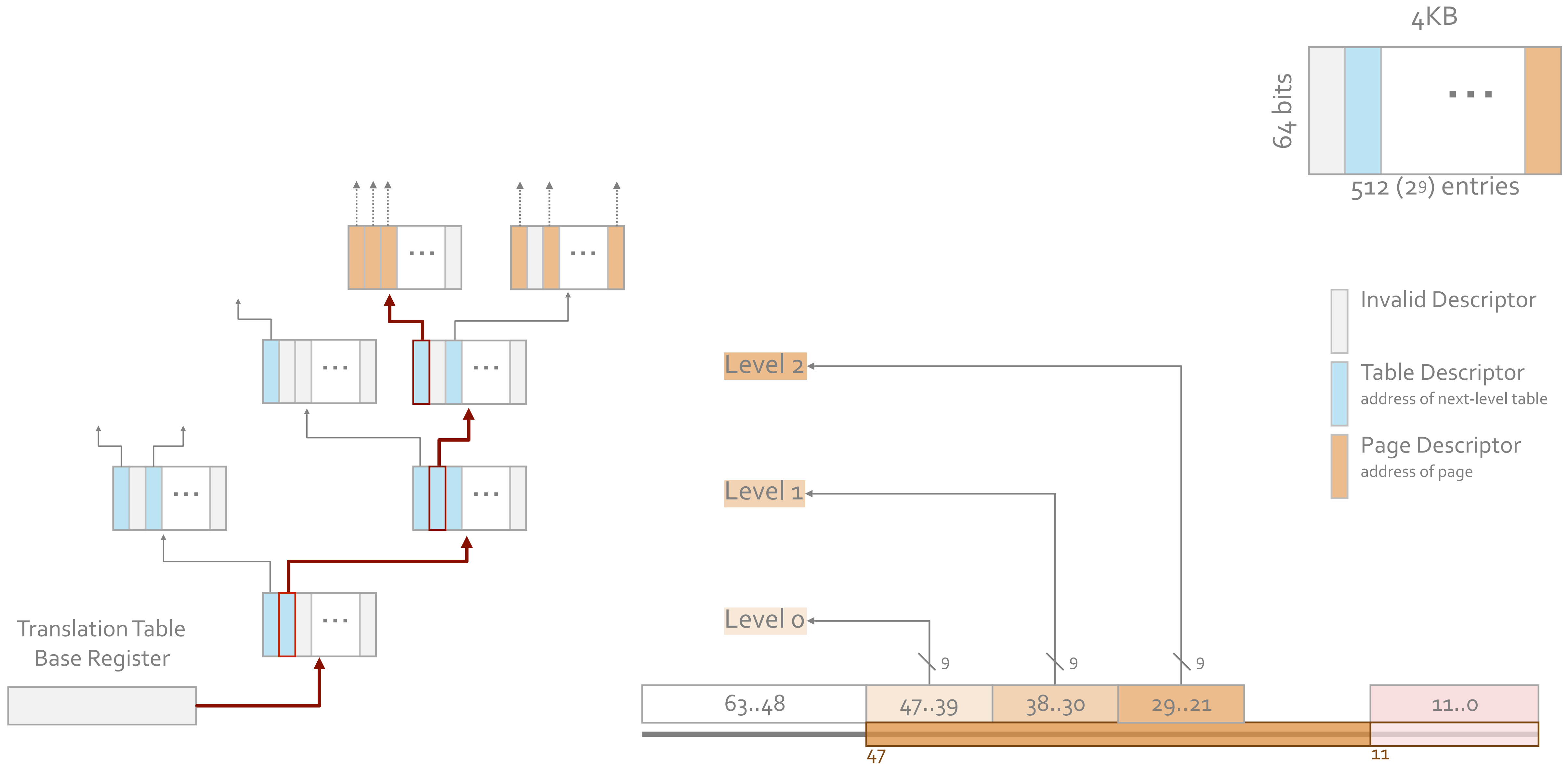
Page table walk



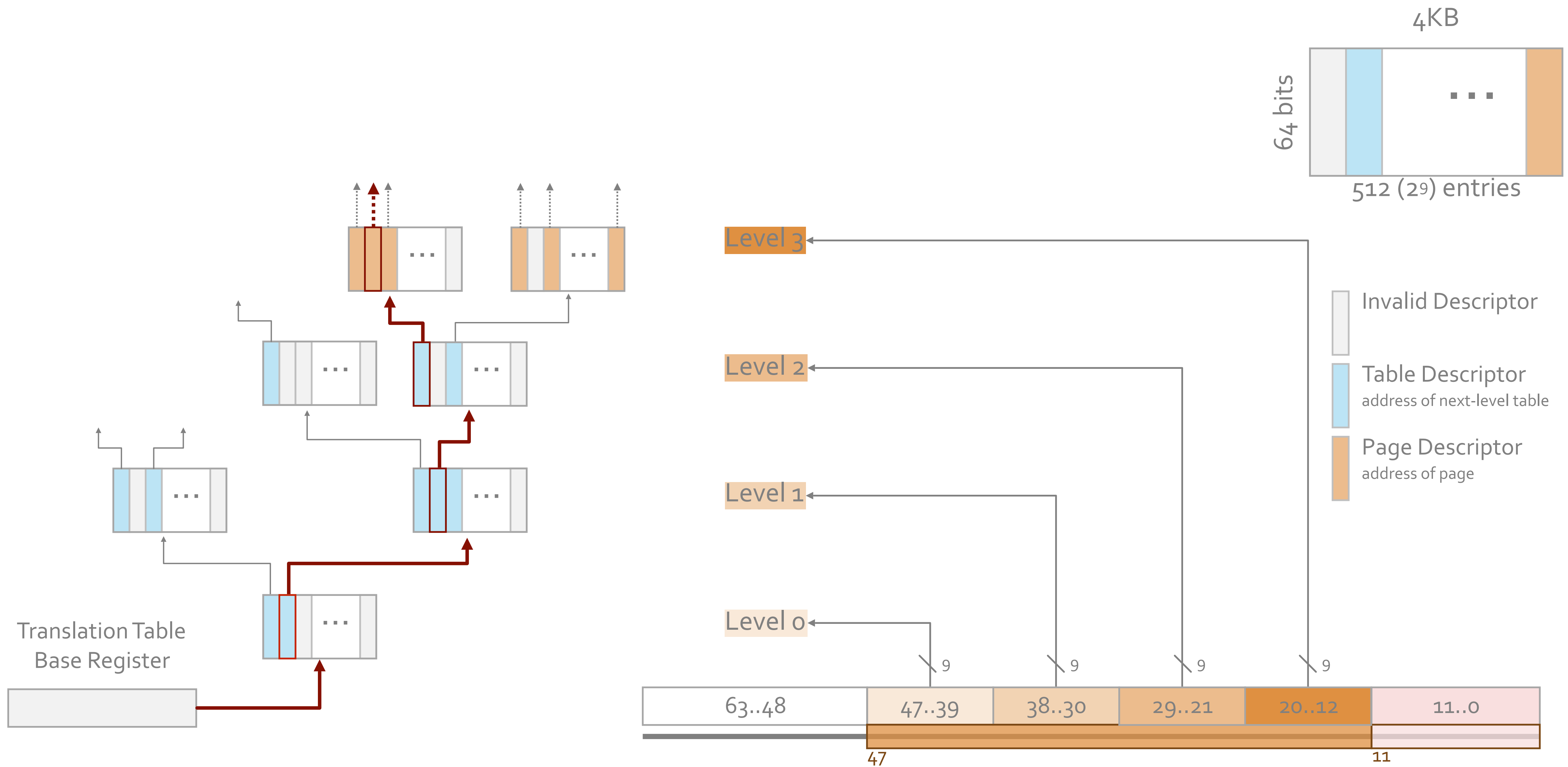
Page table walk



Page table walk



Page table walk



When do we do translation?

- Every memory access a process performs goes through address translation
  - Load, store, instruction fetch
  - Why is this necessary?

When do we do translation?

- Every memory access a process performs goes through address translation
    - Load, store, instruction fetch
    - Why is this necessary?
- Who does the translation?

When do we do translation?

- Every memory access a process performs goes through address translation
  - Load, store, instruction fetch
  - Why is this necessary?
- Who does the translation?
  -

When do we do translation?

- Every memory access a process performs goes through address translation
  - Load, store, instruction fetch
  - Why is this necessary?
- Who does the translation?
  - MMU

Translation Lookaside Buffer (TLB)

- Small cache of recently translated addresses
  - Before translating a referenced address, the processor checks the TLB
- What does the TLB give us?
  -
  -

Translation Lookaside Buffer (TLB)

- Small cache of recently translated addresses
  - ➤ Before translating a referenced address, the processor checks the TLB
- What does the TLB give us?
  - ➤ Physical page corresponding to virtual page (or that page isn't present)
  - ➤ If page mapping allows the mode of access (access control)

Access Control

- Not everything within a processes' virtual address space is equally accessible
- Page descriptors contain additional access control information
  - Read, Write, eXecute permissions
  - Who sets these bits?

How do we get process isolation?

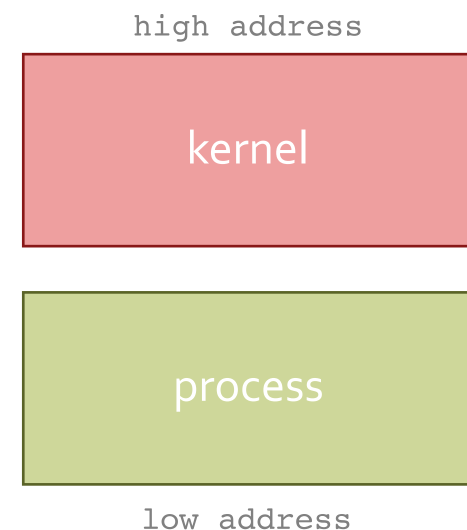
- Each process gets its own tree
  - When you context switch: need to change root
  - What do you do about TLB?
    -
    -

How do we get process isolation?

- Each process gets its own tree
  - ➤ When you context switch: need to change root
  - ➤ What do you do about TLB?
    - ➤ Most often you flush
    - ➤ Don’t need to flush if HW has process-context identifiers (PCIDs)

Beyond process isolation

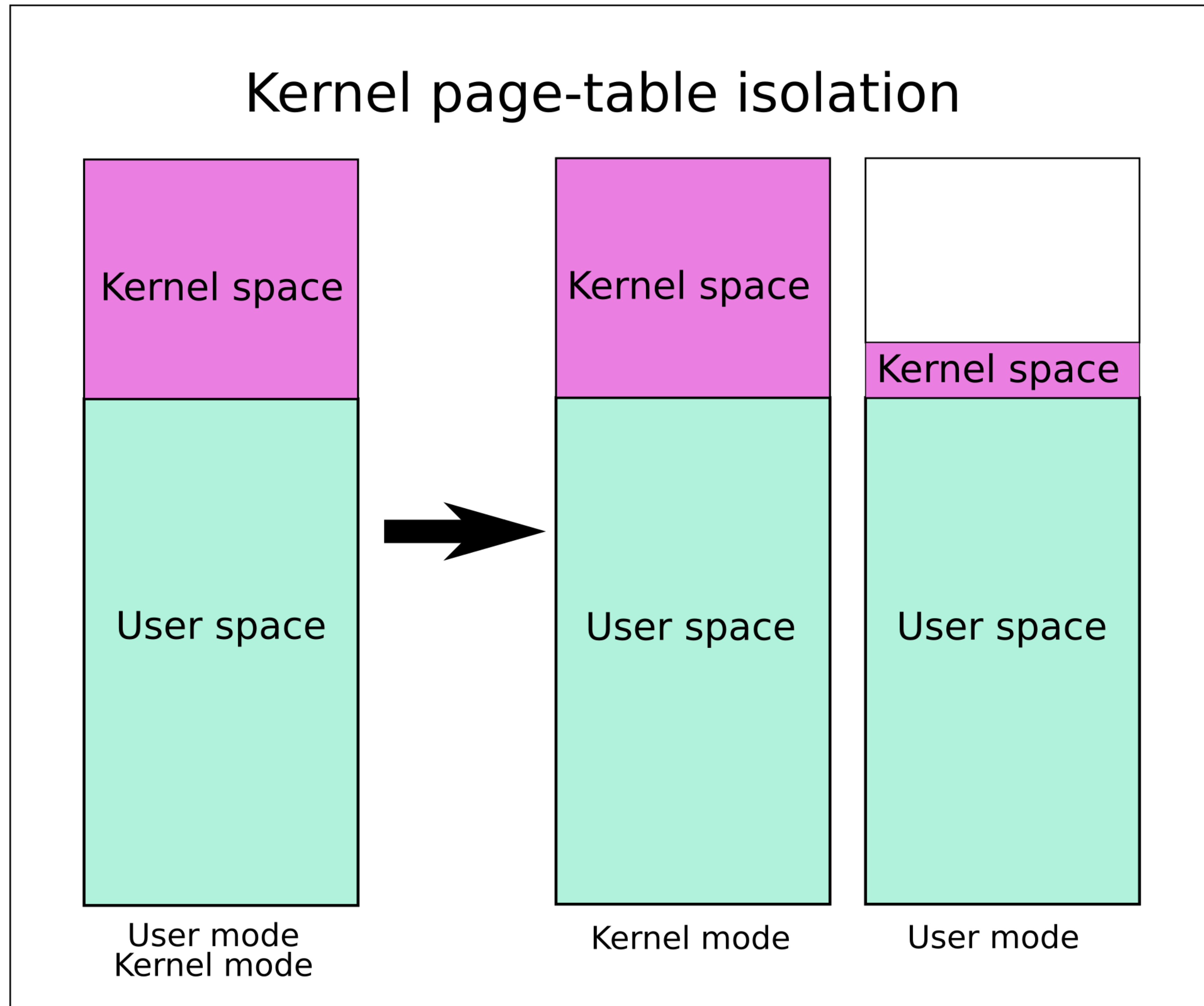
- Kernel’s virtual memory space is mapped into every process, but made inaccessible in usermode
  - Why?
- What happens on sys call?
  - Translation Table Base Register updated
- Do all processes share kernel?



Kernel security

- Threat model:
  - Confidentiality and integrity of kernel memory and control flow must be protected from compromise by usermode processes
  - All usermode processes are untrusted and potentially malicious
- Operating model:
  - Usermode processes make frequent calls into the kernel, with data passing back and forth

Meltdown broke this, so we have:



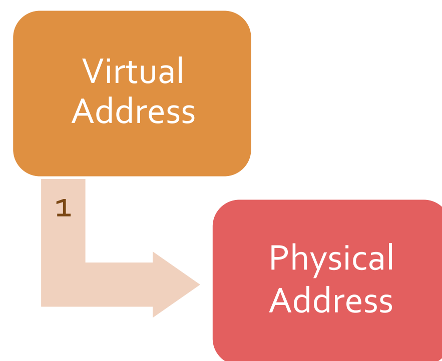
https://en.wikipedia.org/wiki/Kernel_page-table_isolation#/media/File:Kernel_page-table_isolation.svg

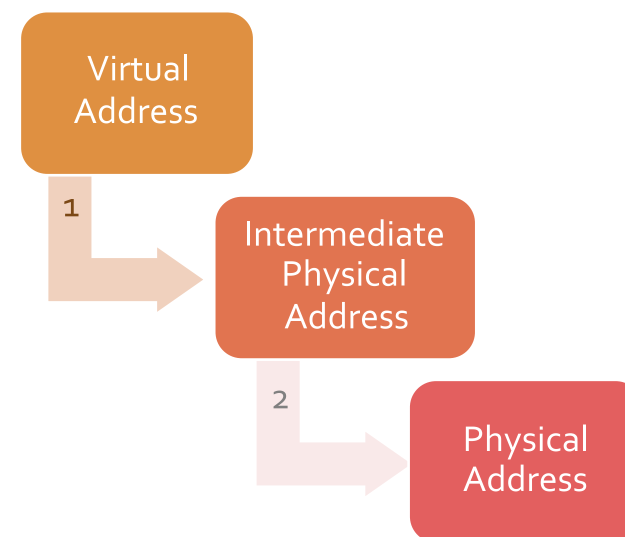
Beyond process isolation: VMs

- VM: the hardware running the OS is virtualized
  - Each OS is oblivious to this happening (mostly)
  - Hypervisor implements VM environment and provides isolation between VMs
  - Are processes within guest OS still isolated?

How does address translation work?

- Multiple stages of address translation to support virtualization
  - Hardware support for this (extended/nestate page tables)





VM security

- Details vary a lot between processor architectures and operating system kernels
  - Even within an architectural family, details may vary a lot between specific processors
  - Even within an operating system, details may vary a lot between specific kernel versions

How can we break isolation?

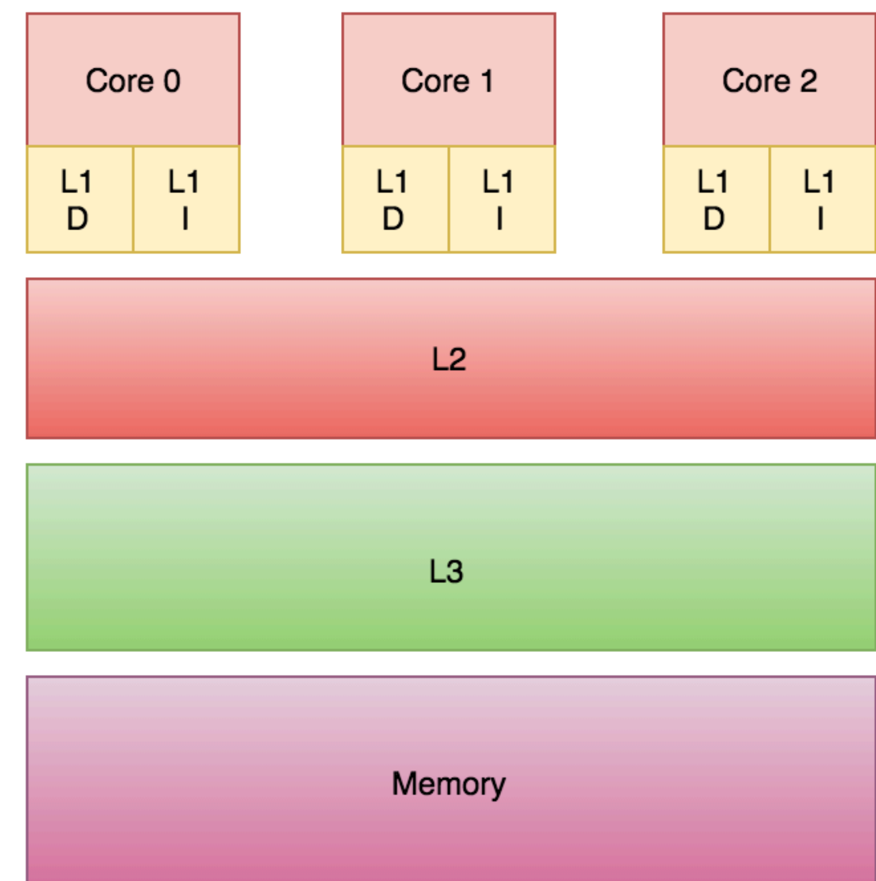
Cache side channels

Cache

- Main memory is huge… but slow
- Processors try to “cache” recently used memory in faster, but smaller capacity, memory cells closer to the actual processing core

Cache hierarchy

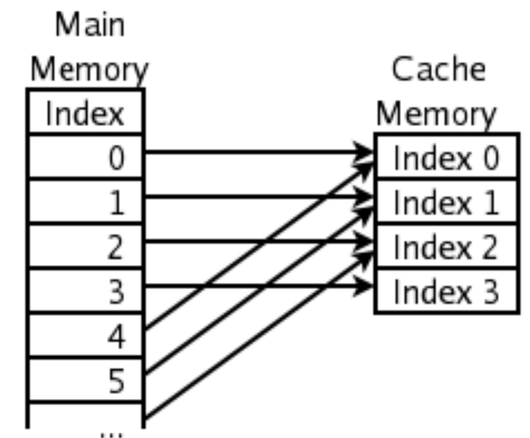
- Caches are such a great idea, let’s have caches for caches!
- The close to the core, the:
  - Faster
  - Smaller



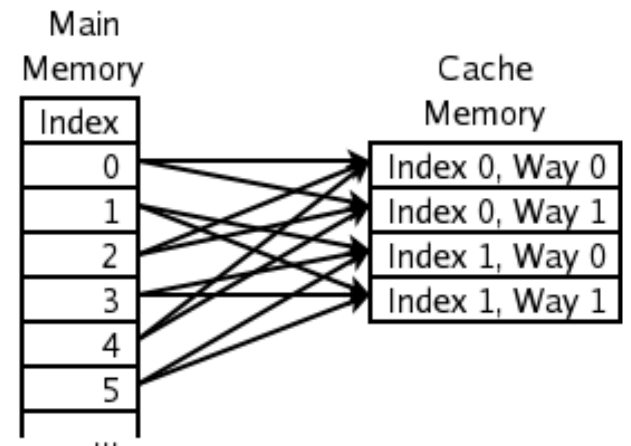
https://en.wikipedia.org/wiki/Cache_hierarchy

How is the cache organized?

- Cache line: unit of granularity
  - ➤ E.g., 64 bytes



- Cache lines grouped into sets
  - ➤ Each memory address is mapped to a set of cache lines

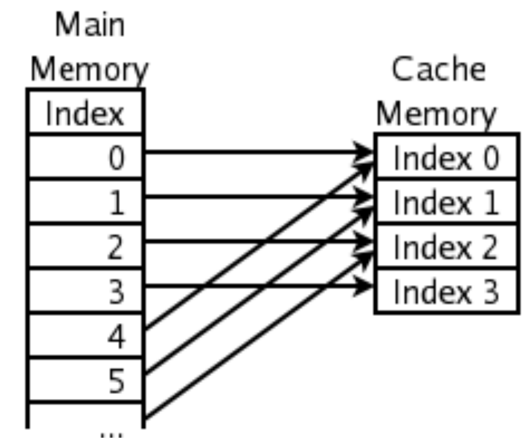


- What happens when we have collisions?
  - ➤

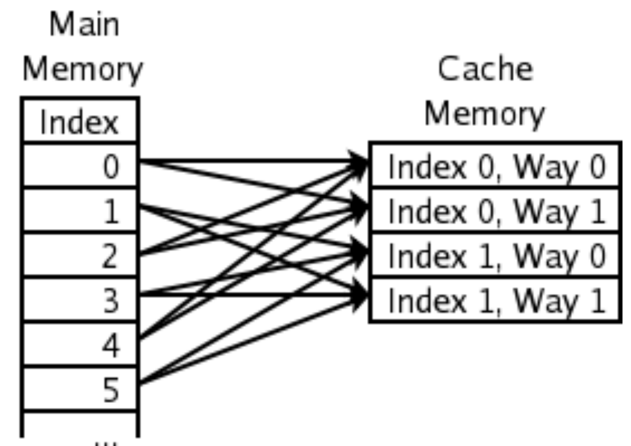
https://en.wikipedia.org/wiki/CPU_cache

How is the cache organized?

- Cache line: unit of granularity
  - E.g., 64 bytes



- Cache lines grouped into sets
  - Each memory address is mapped to a set of cache lines



- What happens when we have collisions?
  - Evict!

https://en.wikipedia.org/wiki/CPU_cache

Cache side channel attacks

- Cache is a shared system resource
  - ➤ Not isolated by process, VM, or privilege level
  - ➤ "Just a performance optimization"
- Can we abuse this shared resource to learn information about another process, VM, etc.?

Thread model

- Attacker and victim are isolated (e.g., processes) but on the same physical system
- Attacker is able to invoke (directly or indirectly) functionality exposed by the victim
    - What's an example of this?
- Attacker should not be able to infer anything about the contents of victim memory

What is a side channel?

- Many algorithms (e.g., crypto) have memory access patterns that are dependent on sensitive memory contents
  - If attacker can observe access patterns: learn data

How do we observe patterns?

- Approach 1
  - Arranges to put the cache into a known state
  - Lets victim run
  - Checks to see what changed: what addresses have appeared or disappeared from cache
- How do we know if something in cache changed?
  -

How do we observe patterns?

- Approach 1
  - Arranges to put the cache into a known state
  - Lets victim run
  - Checks to see what changed: what addresses have appeared or disappeared from cache
- How do we know if something in cache changed?
  - Time access to it

How do we observe patterns?

- Approach 2
  - Times normal victim operation (baseline)
  - Makes controlled changes to cache contents: evicting or fetching specific addresses
  - Times victim operation again

Evict & Time

- Run the victim code several times and time it
- Evict (portions of) the cache
  - How?
- Run the victim code again and time it
- If it is slower than before, cache lines evicted by the attacker must've been used by the victim
  - We now know something about the <u>addresses</u> accessed by victim code

Prime & Probe

- Prime the cache
  - Access many memory locations so that previous cache contents are replaced
- Let victim code run
- Time access to different memory locations, slower means evicted by victim
  - We now know something about the addresses accessed by victim code

Quite a few other approaches

- Evict and Time (shared memory)
- Prime and Abort (TSX)
- …

Flush & Reload

(Only for shared memory)

- Flush the cache
- Let victim code run
- Time access to different memory locations, faster means used by victim
  - ➢ We now know something about the addresses accessed by victim code

How practical are these?

- “Our robust and error-free channel even allows us to build an SSH connection between two virtual machines, where all existing covert channels fail.”

How practical are these?

- "Our robust and error-free channel even allows us to build an SSH connection between two virtual machines, where all existing covert channels fail."
  - Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud
    - by Clementine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Kay Romer, Stefan Mangard