CSE 127: Computer Security

Isolation and side-channels

Deian Stefan

Some slides adopted from Nadia Heninger,John Mitchell, Dan Boneh, and Stefan Savage

Today

Lecture objectives:

- Understand basic principles for building secure systems
- Understand mechanisms used in building secure systems
- Understand a key limitation of these principles: side-channels

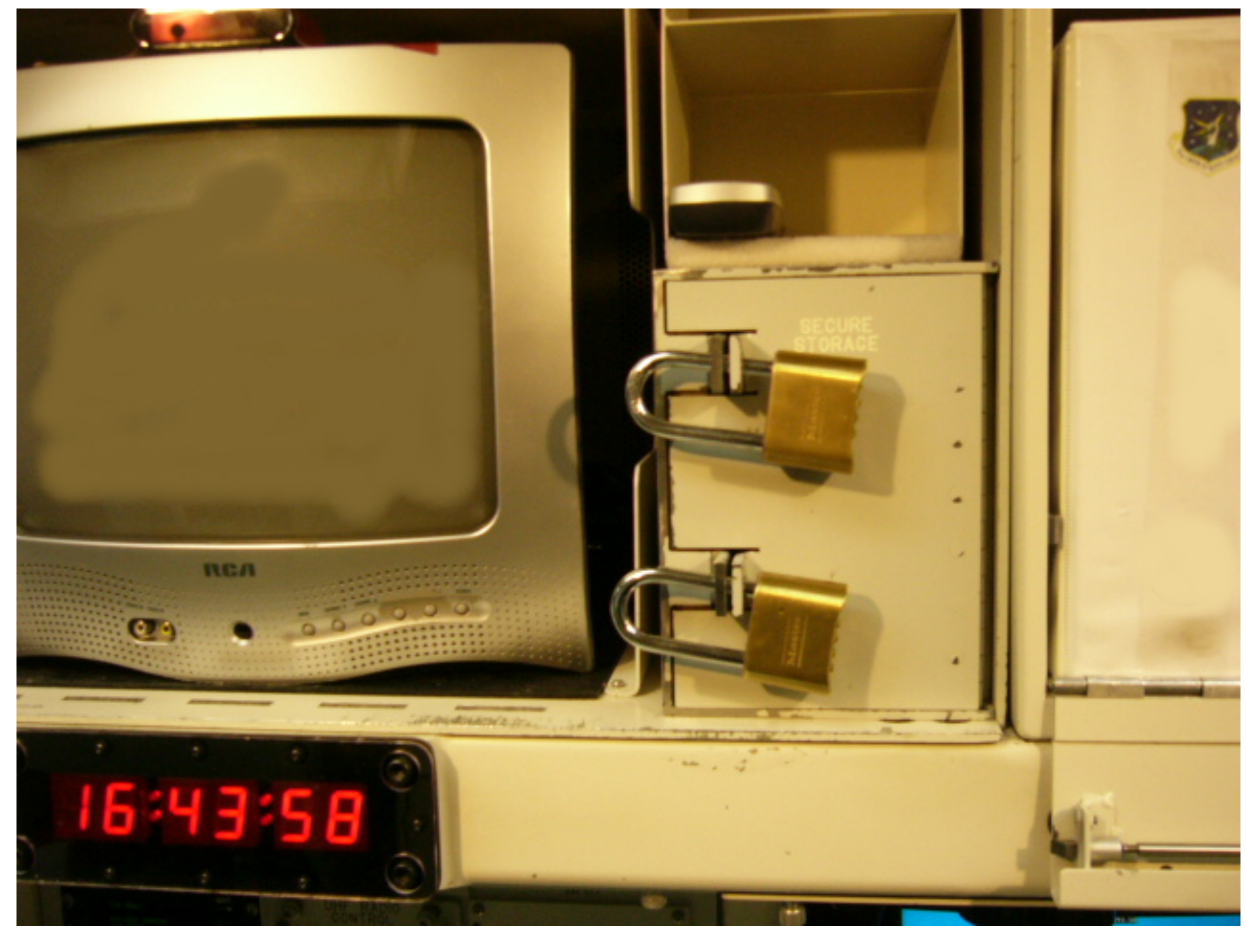
Principles of secure design

- Principle of least privilege
- Privilege separation
- Defense in depth
  - Use more than one security mechanism
  - Fail securely/closed
- Keep it simple

Principles of secure design

- Principle of least privilege ← almost always
- Privilege separation ← come in pair
- Defense in depth
  - ➤ Use more than one security mechanism
  - ➤ Fail securely/closed
- Keep it simple

Where have we seen this before?



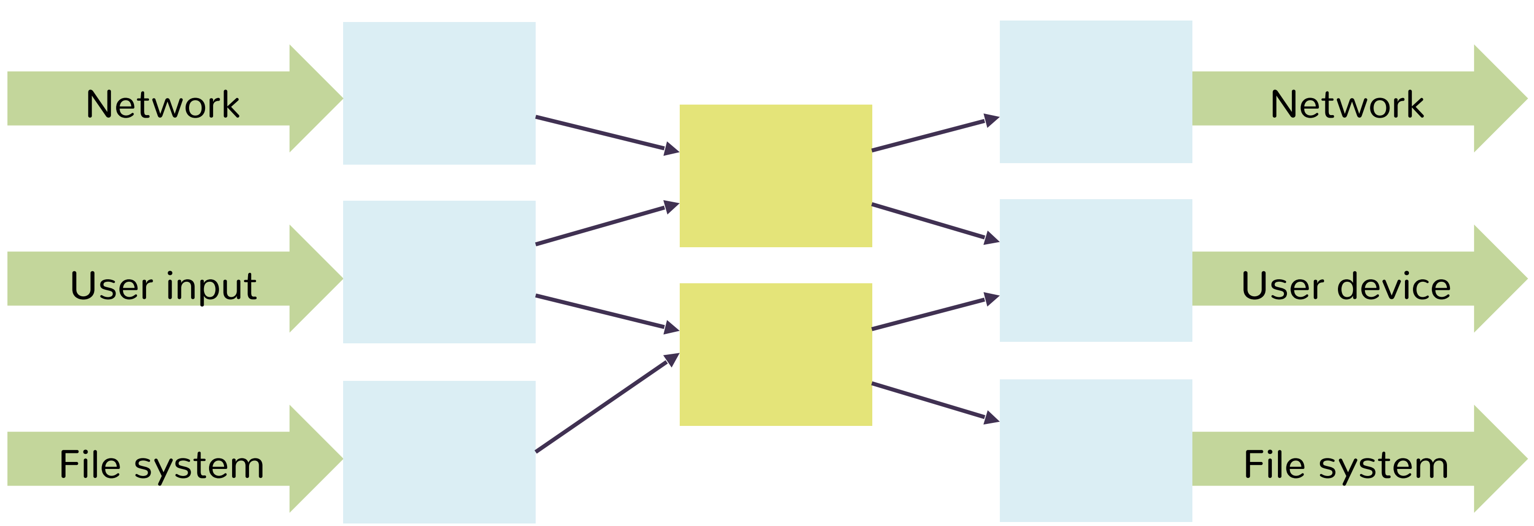
least privilege

privilege separation

High-level idea

- Separate the system into isolated least-privileged compartments
- Mediate interaction between compartments according to security policy

- What's the goal/attacker model assumption?
  - Limit the damage due to any single compromised component



What is the unit of isolation?

- It depends!
  - Physical Machine
  - Virtual Machine
  - OS Process
  - Library
  - Function
  - ...

coarse grain

↓

fine grain

What is the unit of isolation?

- It depends!
  - Physical Machine
  - Virtual Machine
  - OS Process
  - Library
  - Function
  - ...

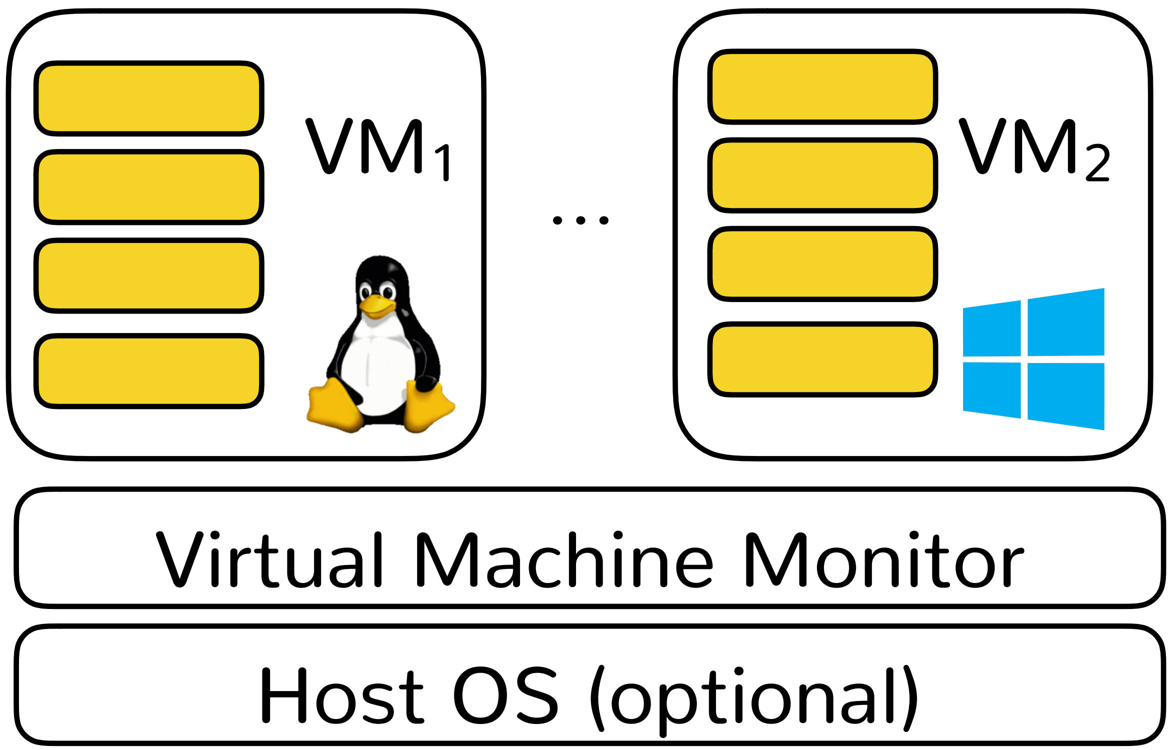
What is the unit of isolation?

- It depends!
    - Physical Machine
    - Virtual Machine
    - OS Process
    - Library
    - Function
    - ...

most popular,

focus in class

The Virtual Machine abstraction (Isolate guest OSes and apps)



The process abstraction (Isolate apps from each other)

- OS ensures that processes are memory isolated from each other
- In UNIX, each process has set of UIDs
  - Used to mediate which files process can read/write
- Conceptually easy to further restrict privileges
  - To do anything useful (e.g., open socket, read file, etc.) process must perform syscall into kernel; interpose on all syscalls and allow/deny according to policy

How are these used to to build secure (least-privileged and privilege separated) systems?

Brief interlude: How do user IDs (UIDs) work?

- Permissions in UNIX granted according to UID
  - A process may access files, network sockets, ...
- Each process has UID
- Each file has ACL
  - Grants permissions to users according to UIDs and roles (owner, group, other)
  - Everything is a file!

How many UIDs does a process have?

Process UIDs

- Real user ID (RUID)
  - ➢ same as the user ID of parent (unless changed)
  - ➢ used to determine which user started the process
- Effective user ID (EUID)
  - ➢ from setuid bit on the file being executed, or syscall
  - ➢ determines the permissions for process
- Saved user ID (SUID)
  - ➢ Used to save and restore EUID

SetUID demystified (a bit)

- Root
  - ID=0 for superuser root; can access any file
- fork and exec system calls
  - Typically inherit three IDs of parent
  - Exec of program with setuid bit: use owner of file
- setuid system call lets you change EUID

SetUID demystified (a bit)

- There are actually 3 bits:
  - setuid - set EUID of process to ID of file owner
  - setgid - set $EG_{roup}ID$ of process to GID of file
  - sticky bit
    - on: only file owner, directory owner, and root can rename or remove file in the directory
    - off: if user has write permission on directory, can rename or remove files, even if not owner

Examples of setuid and sticky bits

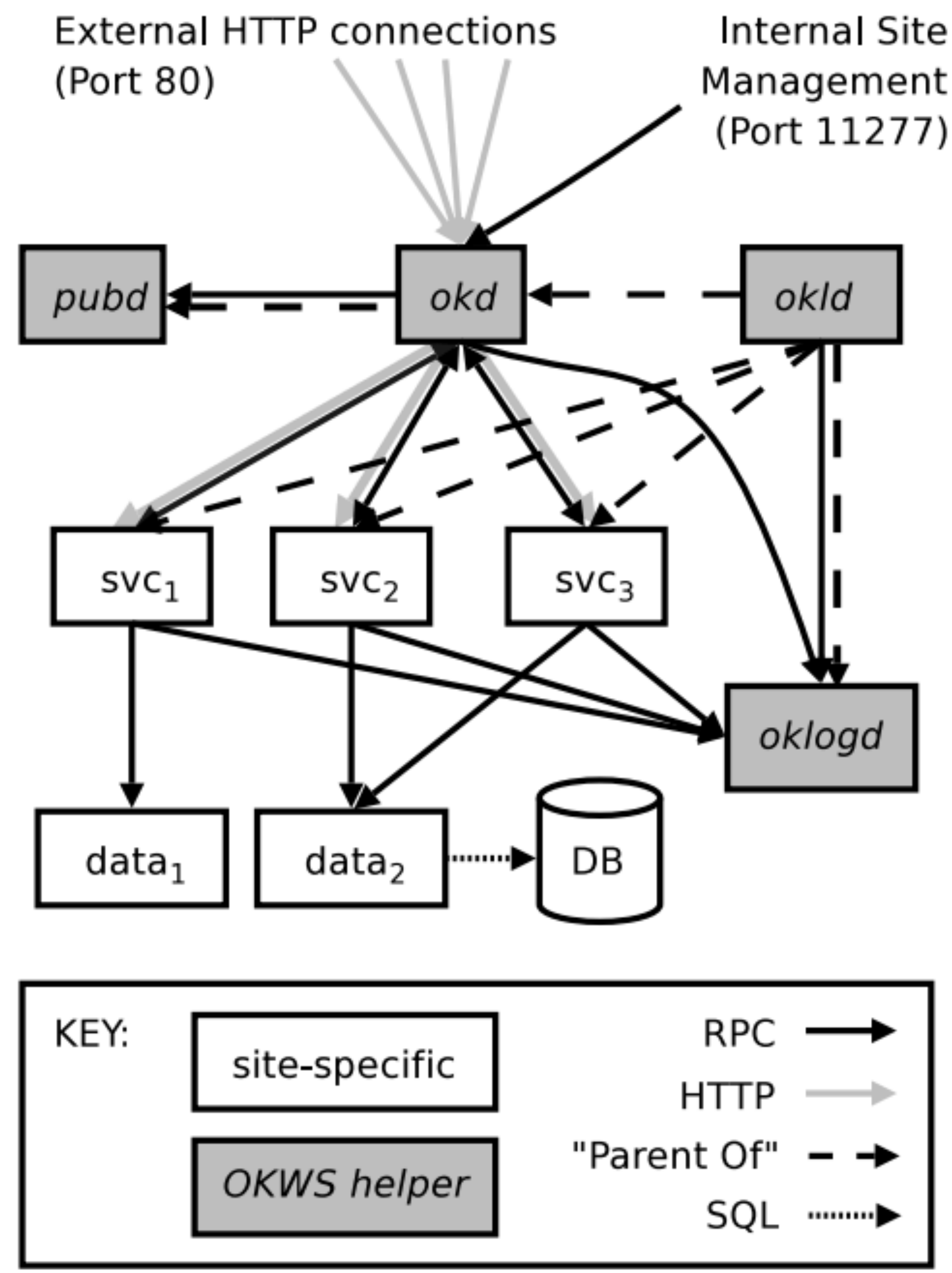
```
-rwsr-xr-x 1 root root 55440 Jul 28  2018 /usr/bin/passwd

drwxrwxrwt 16 root root  700 Feb  6 17:38 /tmp/
```

Example 1: Android

- Each app runs with own process UID
  - Memory + file system isolation
- Communication limited to using UNIX domain sockets + reference monitor checks permissions
  - User grants access at install time + runtime

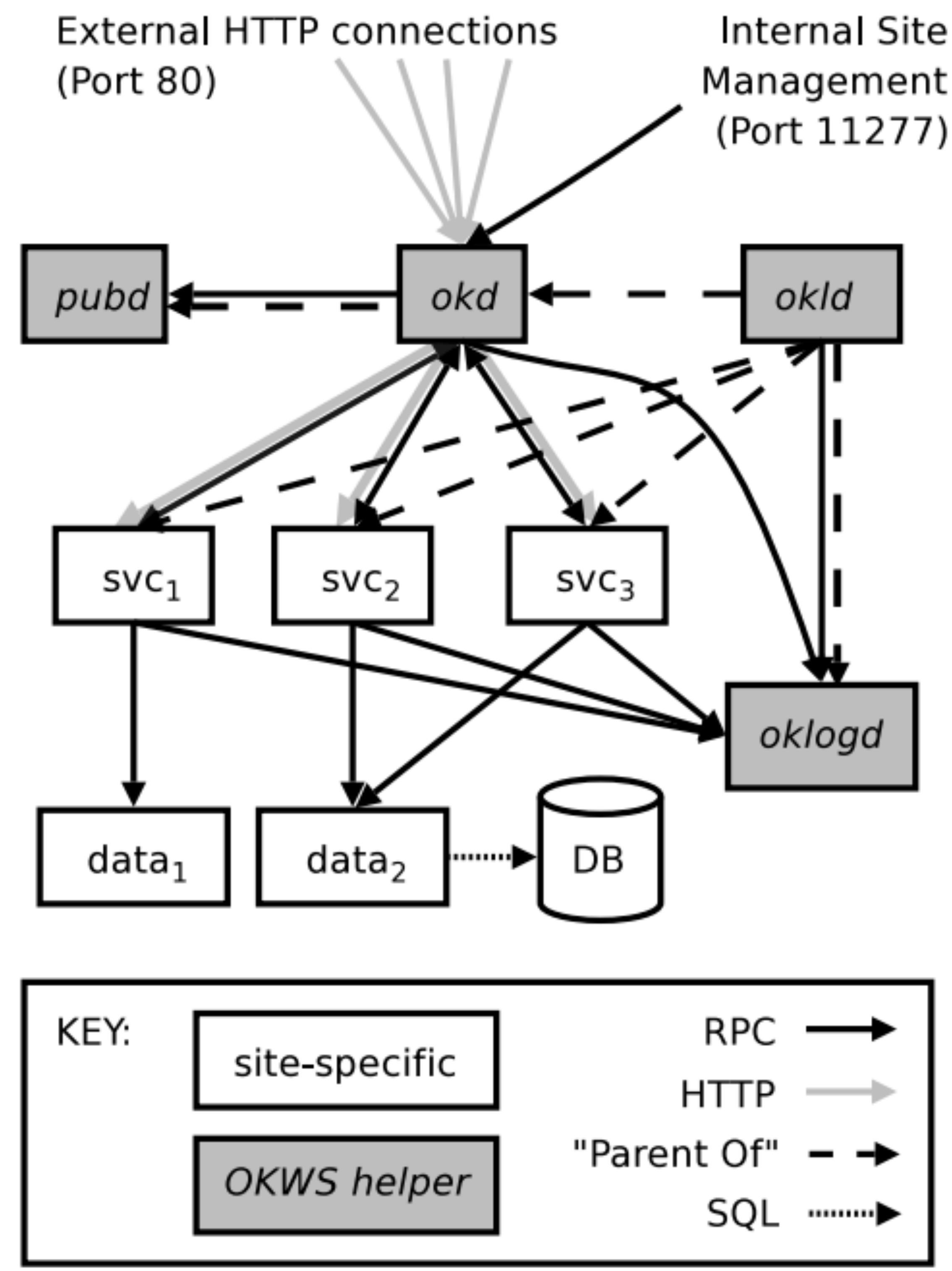
Example 2: OK_{Cupid} W_{eb} S_{erver}

- Each service runs with unique UID
    - Memory + file system isolation
- Communication limited to structured RPC



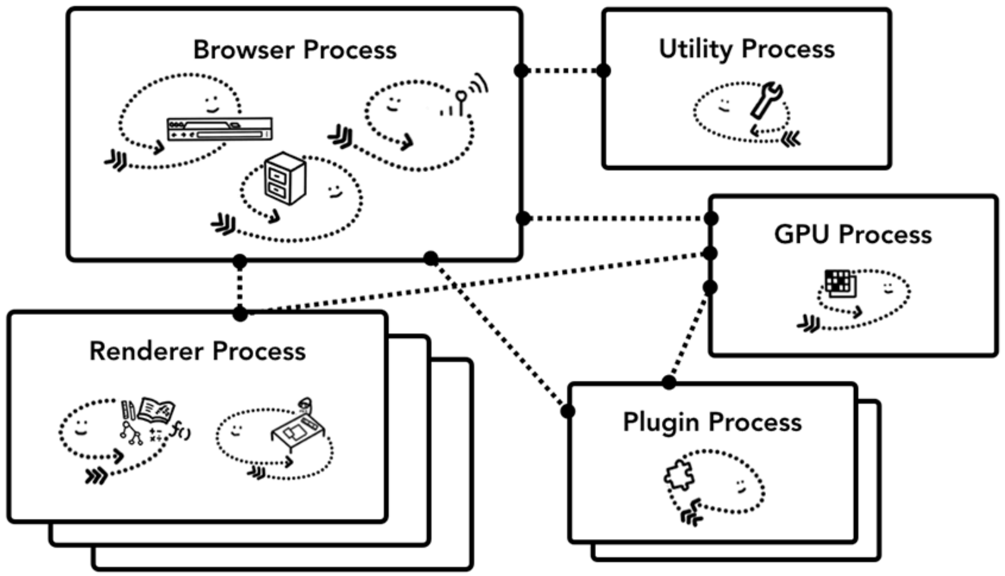
Example 2: OK$_{\text{Cupid}}$W$_{\text{eb}}$S$_{\text{erver}}$

process	*chroot* jail	run directory	uid	gid
okld	`/var/okws/run`	`/`	`root`	`wheel`
pubd	`/var/okws/htdocs`	`/`	`www`	`www`
oklogd	`/var/okws/log`	`/`	`oklogd`	`oklogd`
okd	`/var/okws/run`	`/`	`okd`	`okd`
svc_1	`/var/okws/run`	`/cores/51001`	`51001`	`51001`
svc_2	`/var/okws/run`	`/cores/51002`	`51002`	`51002`
svc_3	`/var/okws/run`	`/cores/51003`	`51003`	`51003`



Example 3: Modern browsers

- Browser process
  - ➤ Handles the privileged parts of browser (e.g., network requests, address bar, bookmarks, etc.)
- Renderer process
  - ➤ Handles untrusted, attacker content: JS engine, DOM, etc.
  - ➤ Communication restricted to RPC to browser/GPU process

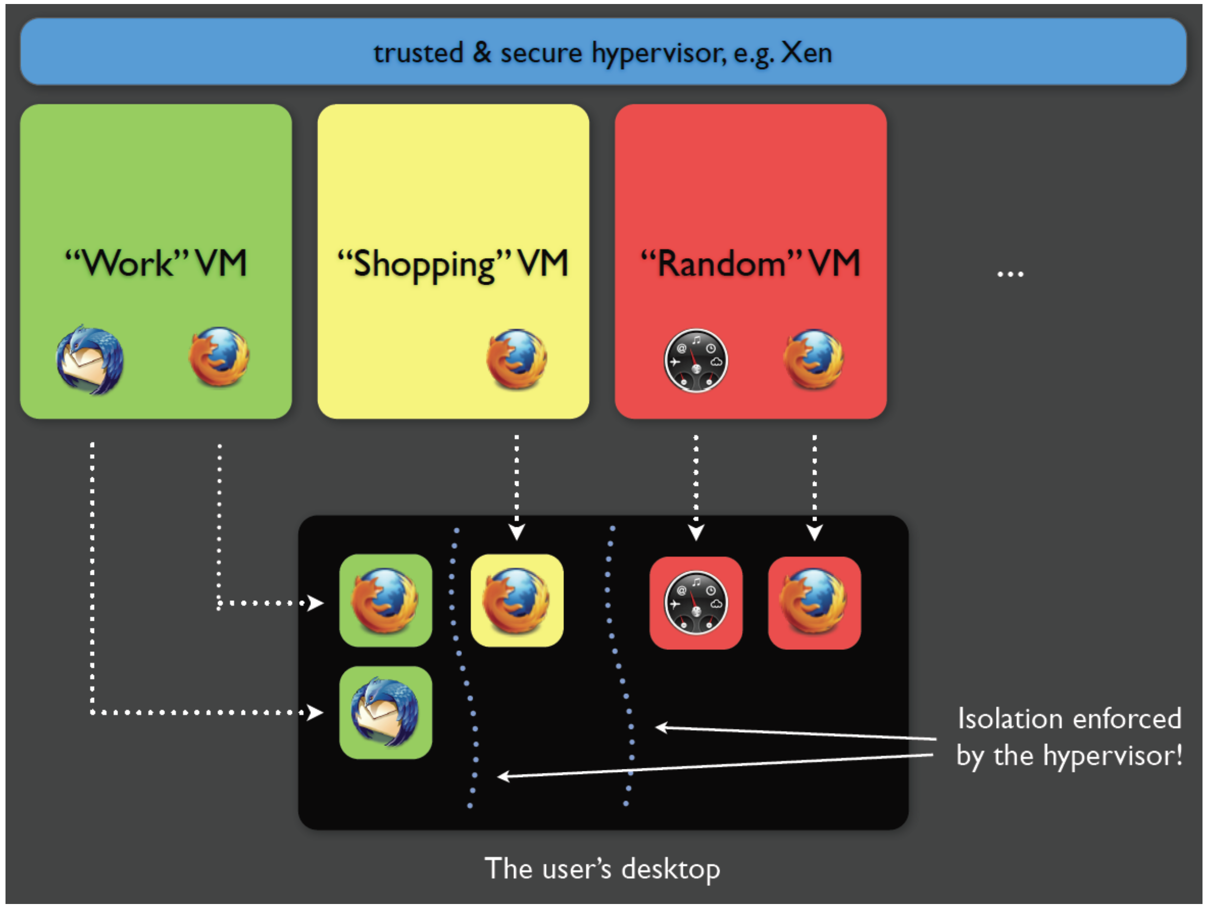


- Many other processes (GPU, plugin, etc)

https://developers.google.com/web/updates/2018/09/inside-browser-part1

Example 4: Qubes OS

- Trusted domain
  - ➤ VM that manages the GUI and other VMs
- Network, USB domains
  - ➤ Isolated domains that handle untrusted data
  - ➤ Communicates with other VMs via firewall domain



- AppVM domains
  - ➤ Apps run in isolation, in different VMs

Today

Lecture objectives:

- Understand basic principles for building secure systems
- Understand mechanisms used in building secure systems
- Understand a key limitation of these principles: side-channels

Many mechanisms at play

- ACL on files used by OS to restrict which processes (based on UID) can access files (and how)
- Namespaces (in Linux) are used to partition kernel resources (e.g., mnt, pid, net) between processes
  - Core part of Docker and other's containers
- Syscall filtering (seccomp-bpf) is used to allow/deny system calls and filter on their arguments
- Etc.

A common, necessary mechanism: memory isolation

A common, necessary mechanism: memory isolation

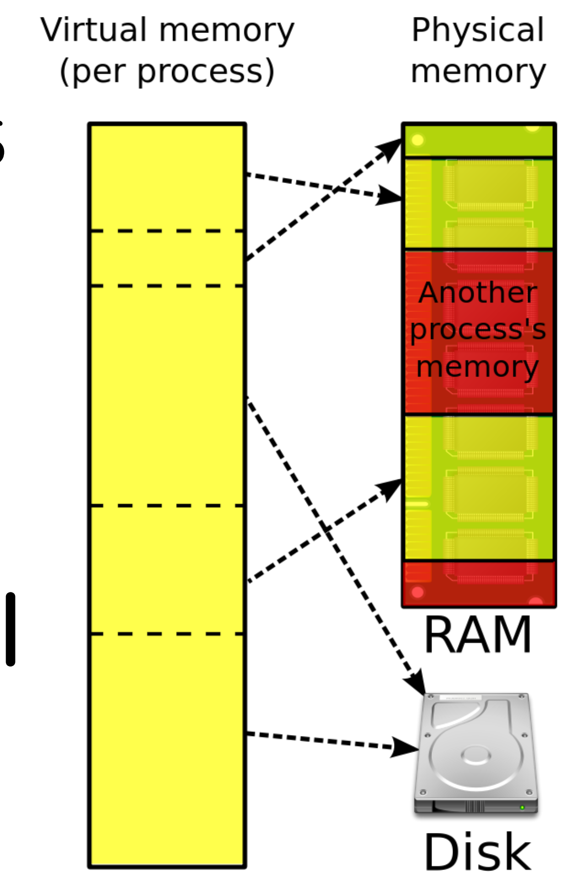
- VM, OS process, and even finer grained in-process isolation all rely on memory isolation
- Why?

A common, necessary mechanism: memory isolation

- VM, OS process, and even finer grained in-process isolation all rely on memory isolation
- Why?
  - ➤ If attacker can break memory isolation, they can often hijack control flow!

Process memory isolation

- How are individual processes memory-isolated from each other?
  - ➤ Each process gets its own virtual address space, managed by the operating system
- Memory addresses used by processes are virtual addresses (VAs) not physical addresses (PAs)
  - ➤ When and how do we do the translation?



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

When do we do the translation?

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

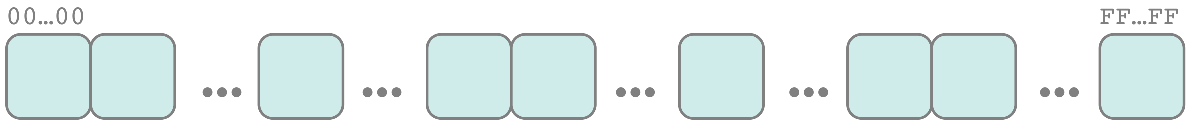
When do we do the translation?

- Every memory access a process performs goes through address translation
    - Load, store, instruction fetch
- Who does the translation?
    - The CPU’s memory management unit (MMU)

How does the MMU translate VAs to PAs?

- Using 64-bit ARM architecture as an example…
- How do we translate arbitrary 64bit addresses?
  - We can't map at the individual address granularity!
  - 64 bits * 2^{64} (128 exabytes) to store any possible mapping

Address translation (closer)

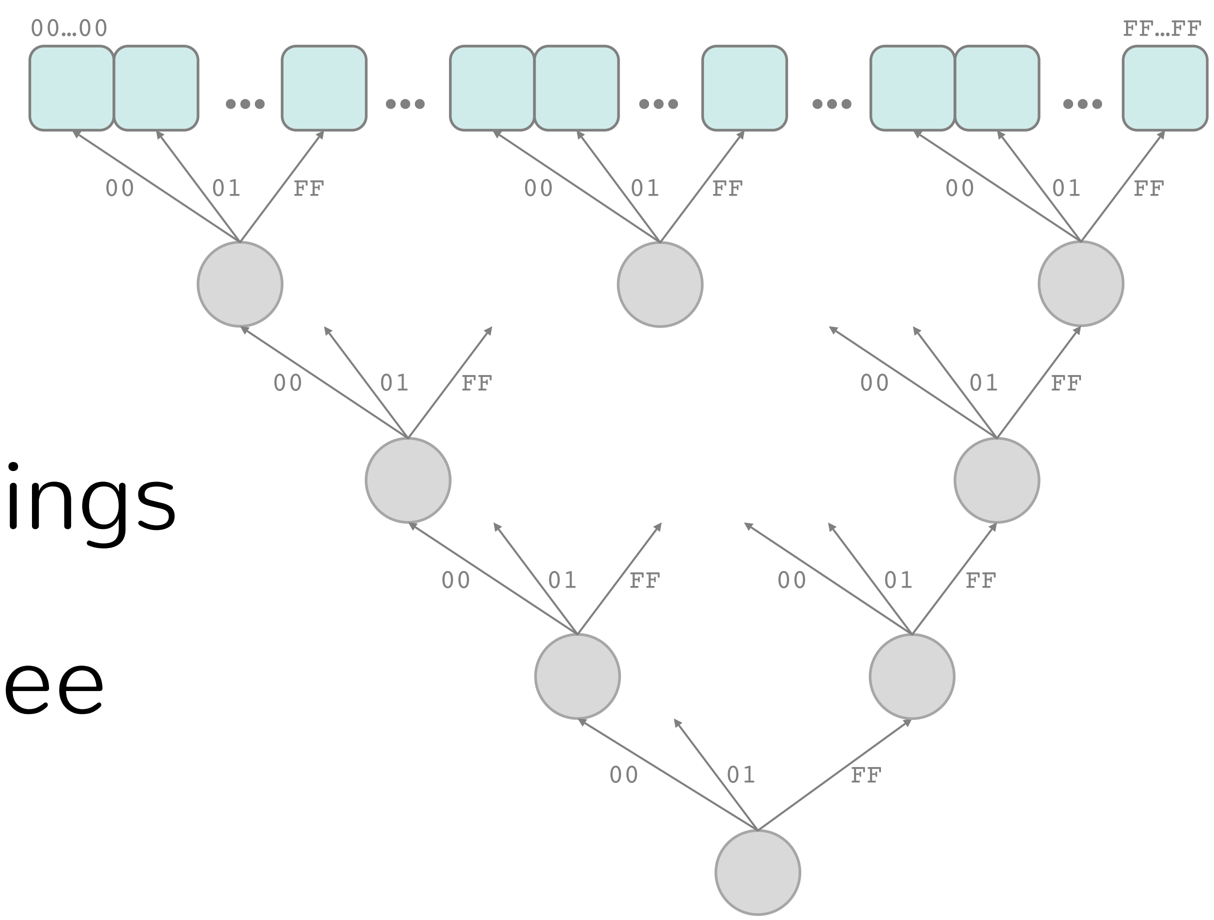


- Page: basic unit of translation
  - ➤ Usually 4KB = 2^{12}
- How many page mappings?
  - ➤ Still too big!
  - ➤ 52 bits * 2^{52} (208 petabytes)

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
- Root is kept in register so MMU can walk the tree



How do we get isolation between processes?

- Each process gets its own tree
  - Tree is created by the OS
  - Tree is used by the MMU when doing translation
    - This is called “page table walking”
  - When you context switch: OS needs to change root
- Kernel has its own tree

Access control

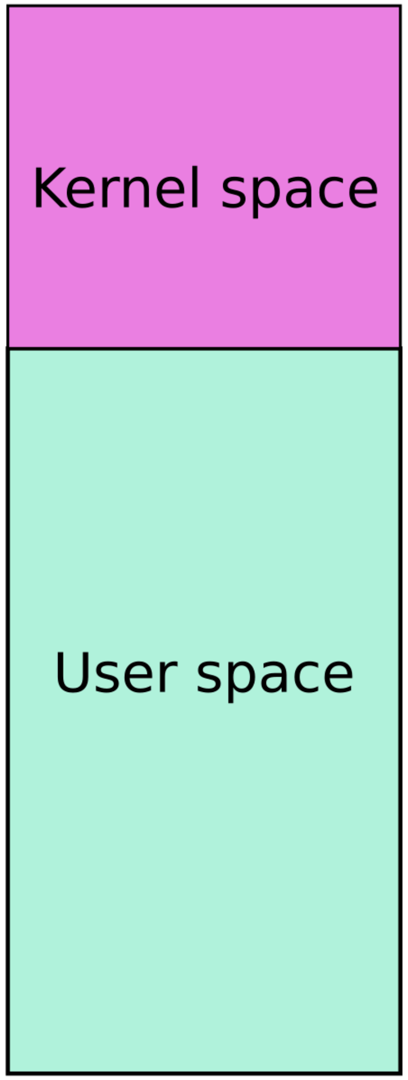
- Not everything within a processes' virtual address space is equally accessible
- Page descriptors contain additional access control information
  - Read, Write, eXecute permissions
  - Who sets these bits? (The OS!)

Example of access control usage

*This changed due to Meltdown.

Example of access control usage

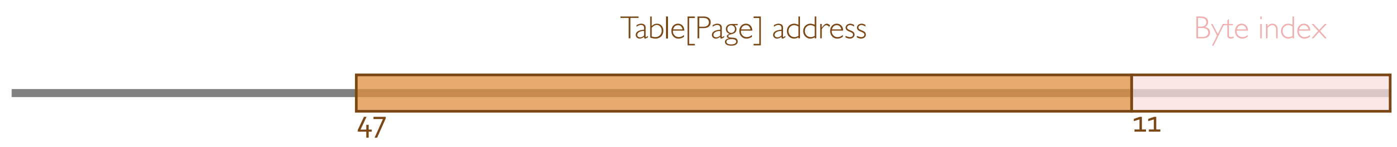
- Kernel’s virtual memory space is* mapped into every process, but made inaccessible in usermode
  - Makes context switching fast!



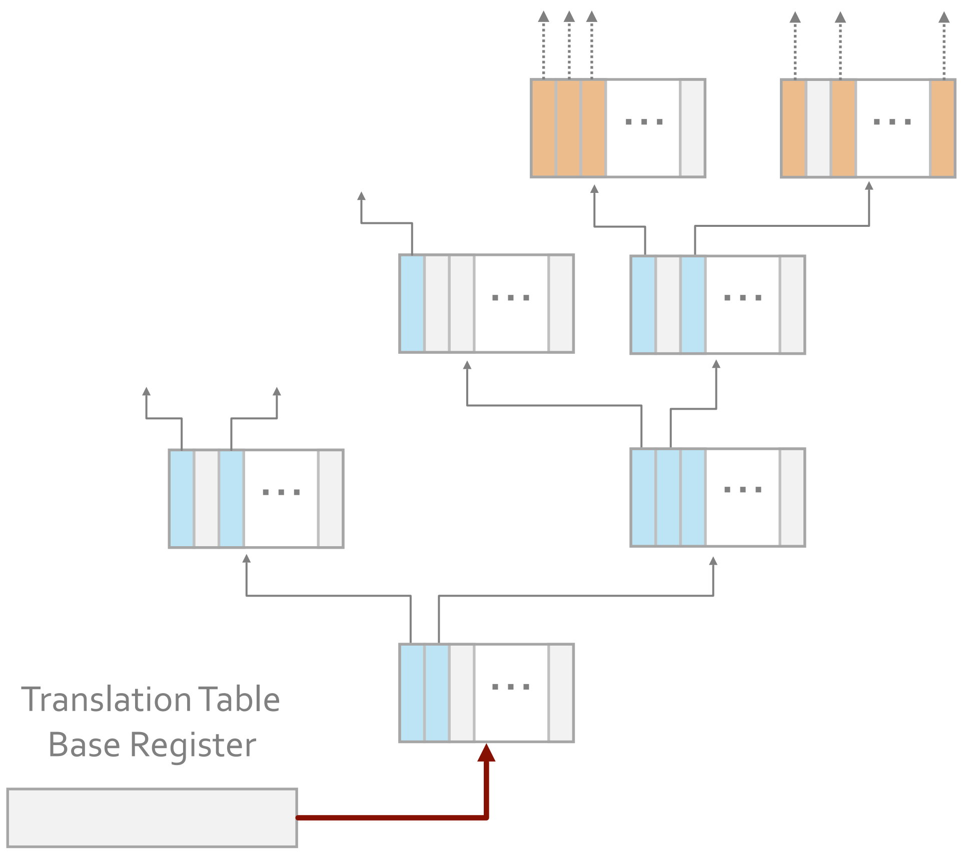
*This changed due to Meltdown.

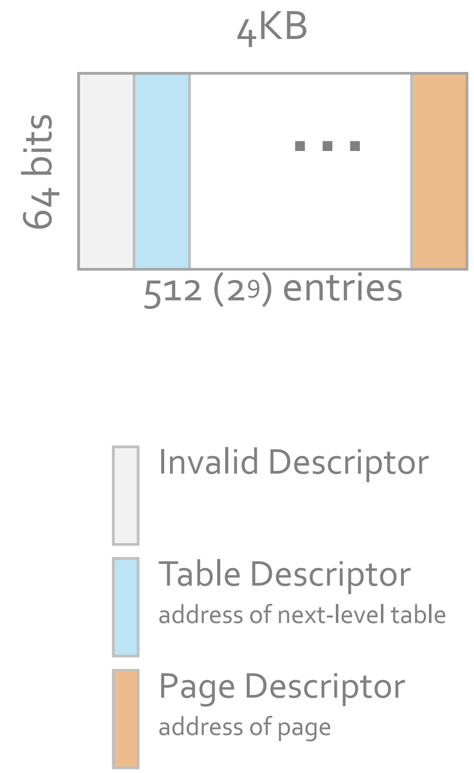
Example of page table walk

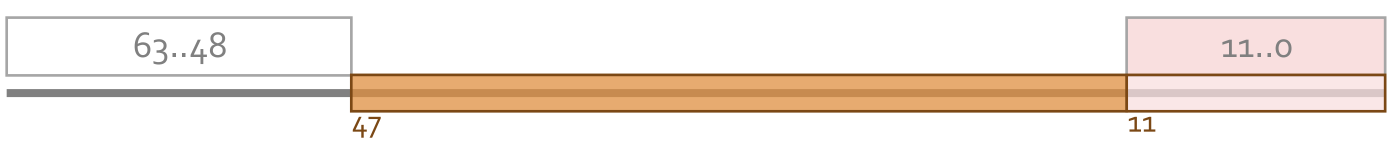
- In reality, the full 64bit address space is not used.
  - Working assumption: 48bit addresses



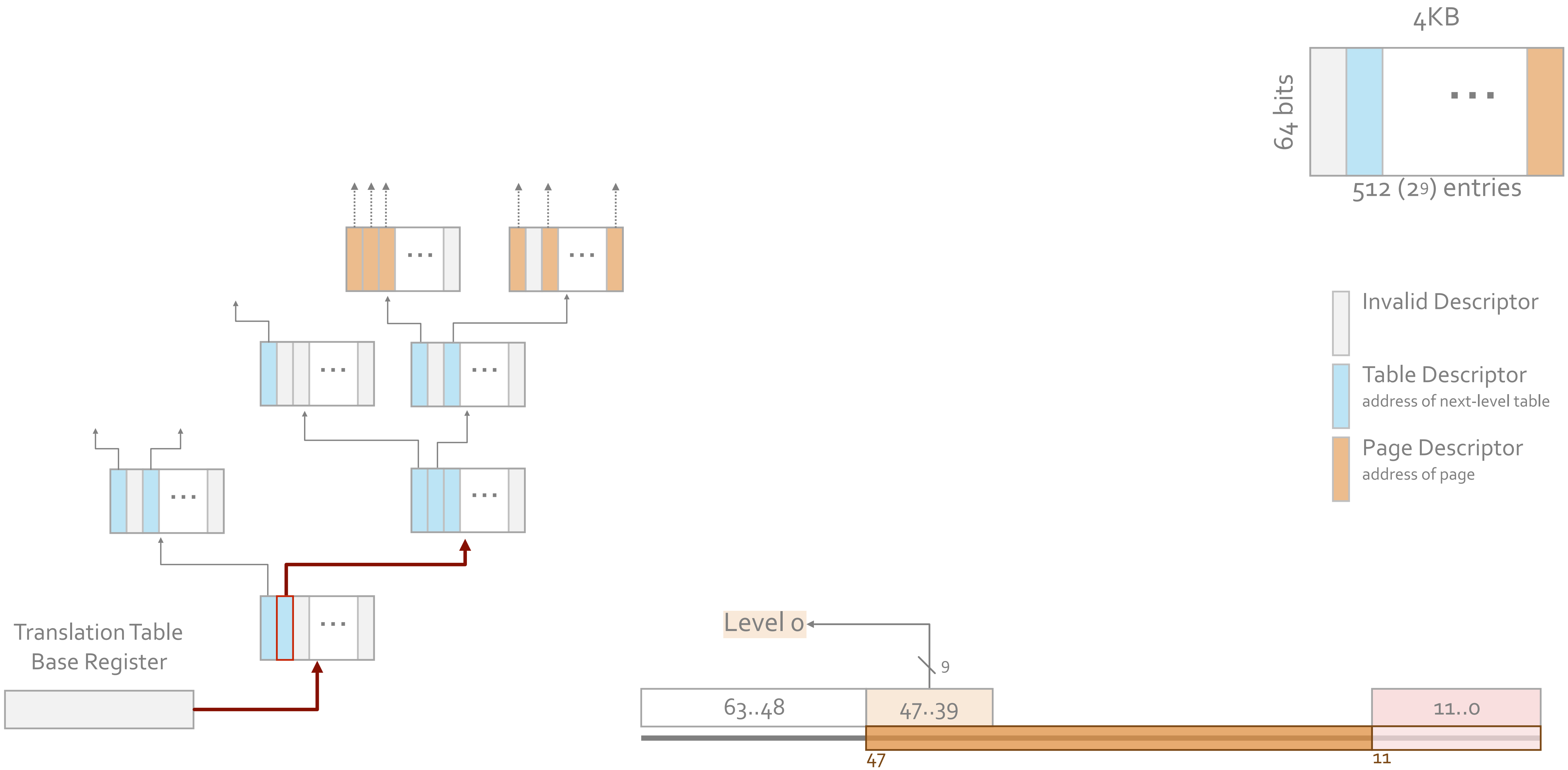
Page table walk



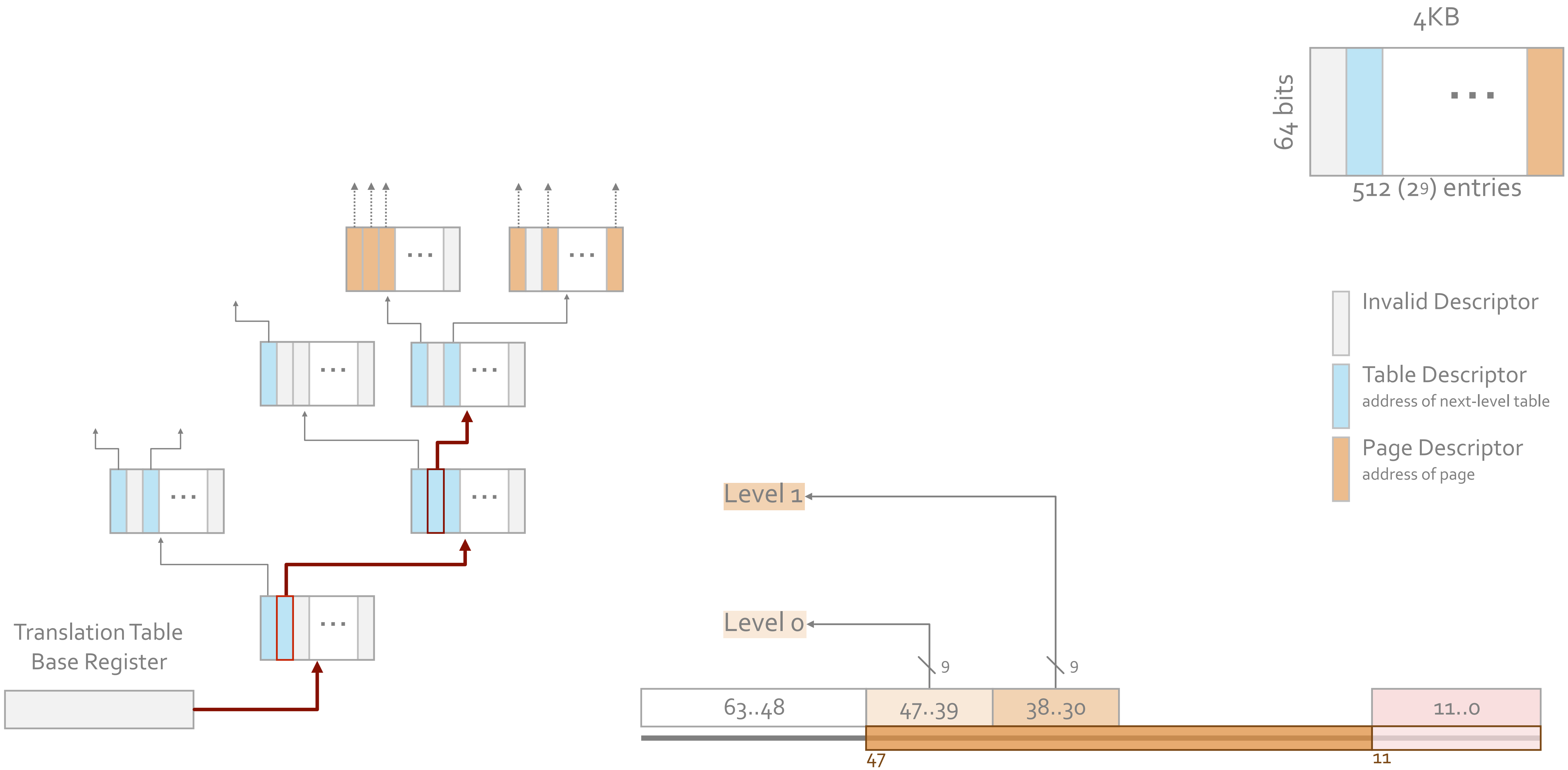




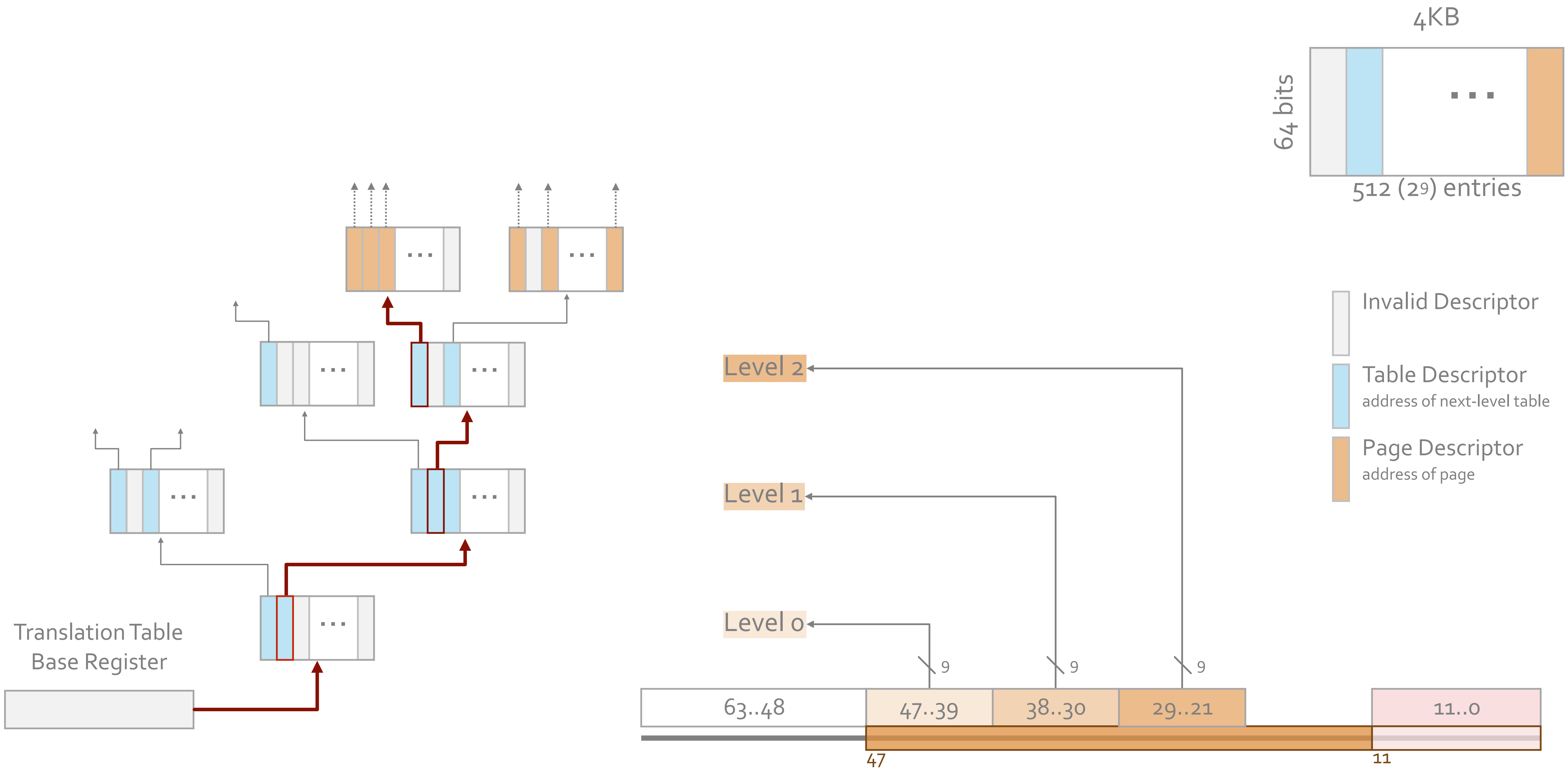
Page table walk



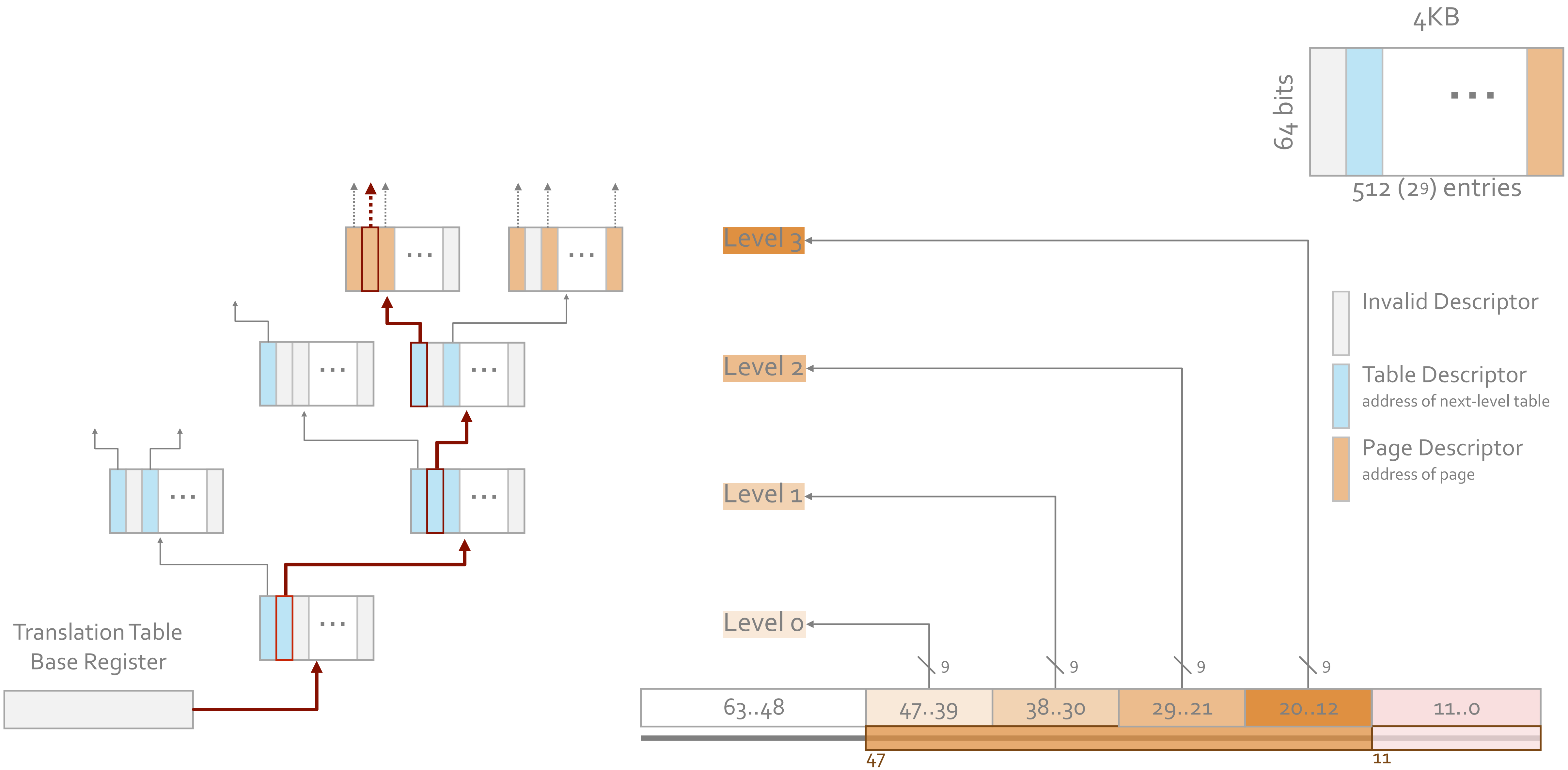
Page table walk



Page table walk



Page table walk



How do we make this fast?

Translation Lookaside Buffer (TLB)

How do we make this fast?

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?

How do we make this fast?

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)

How do we make this fast?

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)

What should we do about TLB on context switch?

What should we do about TLB on context switch?

- Can flush the TLB (was most popular)
- If HW has process-context identifiers (PCID), don't need to flush: entries in TLB are partitioned by PCID

What about memory isolation for VMs?

How is the memory of VMs isolated?

- Need to isolate process in one VM from the process (or the kernel) of another VM
- Address translation is more complicated
  - VM/Guest VA to VM PA translation is not enough
  - Why not?

How is the memory of VMs isolated?

How is the memory of VMs isolated?

- Modern hardware has support for extended/nested page table entries
  - Allows VM OS to map guest PA to machine/host PA without calling into VMM

How is the memory of VMs isolated?

- Modern hardware has support for extended/nested page table entries
  - Allows VM OS to map guest PA to machine/host PA without calling into VMM
- What do we do about the TLB?

How is the memory of VMs isolated?

- Modern hardware has support for extended/nested page table entries
  - Allows VM OS to map guest PA to machine/host PA without calling into VMM
- What do we do about the TLB?
  - TLB entries are also tagged with VM ID (VPID)

How is the memory of VMs isolated?

- Modern hardware has support for extended/nested page table entries
  - ➤ Allows VM OS to map guest PA to machine/host PA without calling into VMM
- What do we do about the TLB?
  - ➤ TLB entries are also tagged with VM ID (VPID)
- How do we isolate VMM from guest VMs?

How is the memory of VMs isolated?

- Modern hardware has support for extended/ nested page table entries
  - Allows VM OS to map guest PA to machine/host PA without calling into VMM
- What do we do about the TLB?
  - TLB entries are also tagged with VM ID (VPID)
- How do we isolate VMM from guest VMs?
  - Similar to kernel: VMM is assigned VPID 0

Today

Lecture objectives:

- Understand basic principles for building secure systems
- Understand mechanisms used in building secure systems
- Understand a key limitation of these principles: side-channels

How can you defeat VM/process isolation?

How can you defeat VM/process isolation?

- Find a bug in the kernel or hypervisor!
  - Kernels are huge and have a huge attack surface: syscalls
  - Developers make mistakes—from forgetting to check and sanitize values that come from user space to classical memory safety bugs.

How can you defeat VM/process isolation?

- Find a bug in the kernel or hypervisor!
  - Kernels are huge and have a huge attack surface: syscalls
  - Developers make mistakes—from forgetting to check and sanitize values that come from user space to classical memory safety bugs.
- Find a hardware bug
  - E.g., Meltdown breaks process isolation

How can you defeat VM/process isolation?

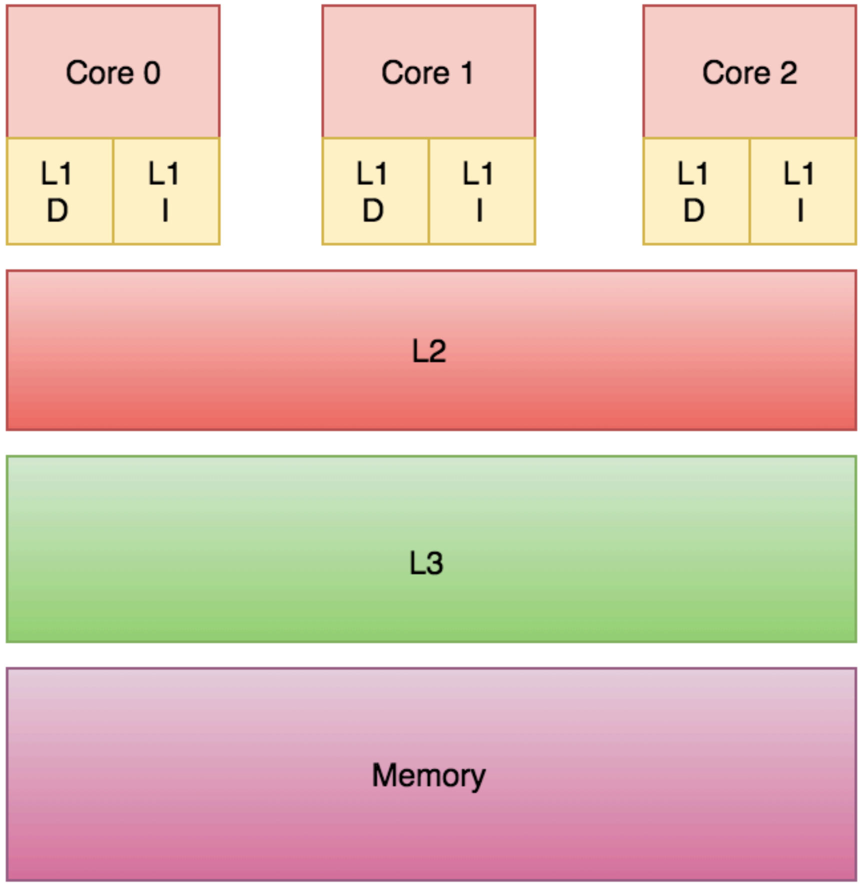
- Find a bug in the kernel or hypervisor!
  - Kernels are huge and have a huge attack surface: syscalls
  - Developers make mistakes—from forgetting to check and sanitize values that come from user space to classical memory safety bugs.
- Find a hardware bug
  - E.g., Meltdown breaks process isolation
- Exploit OS/hardware side-channels
  - Cache-based side channels are the easiest/most popular

What is the 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

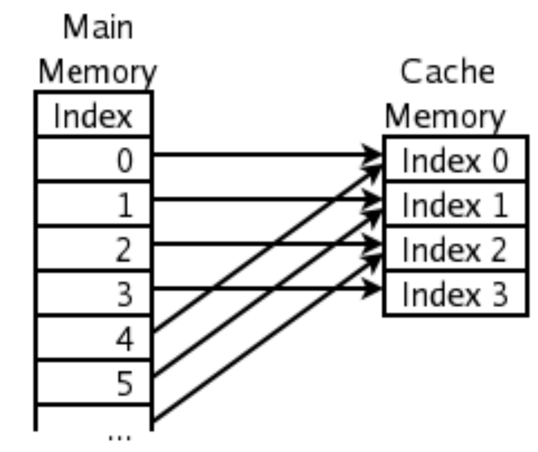
- Caches are such a great idea, let's have caches for caches!
- The close to the core, the:
  - Faster
  - Smaller



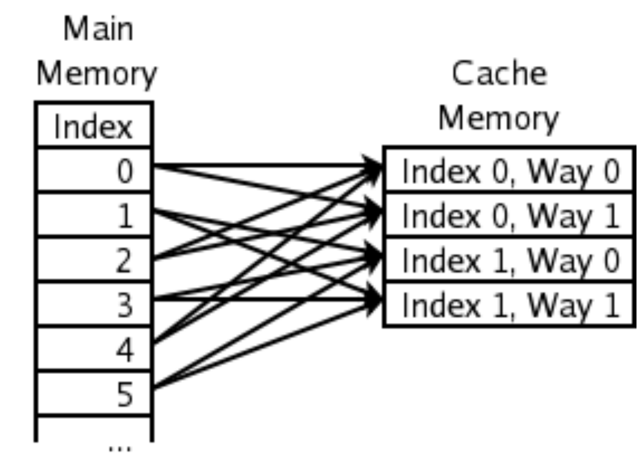
https://en.wikipedia.org/wiki/Cache_hierarchy

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?
  - Evict!

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

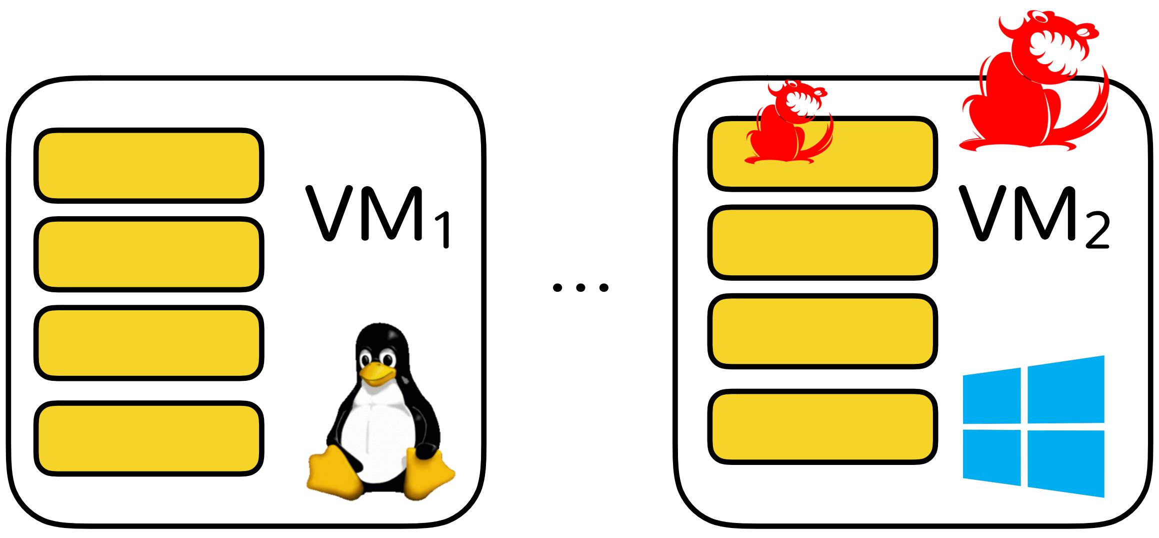
Cache side channel attacks

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

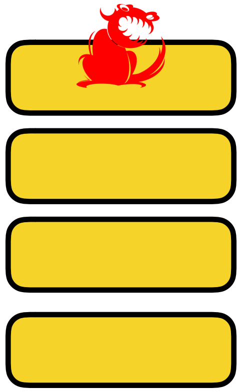
Threat model

- Attacker and victim are isolated (e.g., in separate 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

Threat model: co-located VM



Threat model: co-located process



What is a side channel?

- Many algorithms have memory access patterns that are dependent on sensitive memory contents
  - What are some examples of this?
- So? If attacker can observe access patterns they can learn secrets

Quite a few approaches

- Evict and Time
- Prime and Probe
- Flush and Reload
- Prime and Abort
- Flush and Flush

Quite a few approaches

- Can work on different caches (L1 to L3)
- Can work on both I$ and D$
- Assumption: VA to PA mapping known to attacker
  - Not all rely on this but can often infer this

Evict & Time

- ➤ Run the victim code several times and time it
- ➤ Evict cache line(s)
- ➤ 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
  - ➤ In some cases addresses are secret (e.g., AES)

Prime & Probe

- Prime the cache
  - Access many memory locations so that previous cache contents are replaced
- Let victim code run
- Time access to own memory locations (slower means evicted by victim)
  - We now know something about the <u>addresses</u> accessed by victim code

Flush & Reload

(Only for shared memory)

- Flush (specific lines from) the cache
- Let victim code run
- Time access to different memory locations, faster means used by victim
  - We now know something about the <u>addresses</u> 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