CSE 127: Computer Security

Sandboxing and isolation

Deian Stefan

Today

Lecture objectives:

- Understand basic principles for building secure systems
- Understand mechanisms used to build secure systems

Principles of secure design

- Least privilege
- Privilege separation
- Complete mediation
- Defense in depth
- Fail safe/closed
- Keep it simple

Some photos from Smith's A Contemporary Look at Saltzer and Schroeder's 1975 Design Principles and Wikipedia (e.g., https://en.wikipedia.org/wiki/Beaumaris_Castle)

The privilege separation recipe:

- Break system into compartments
- Ensure each compartment is isolated
- Ensure each compartment runs with least privilege
- Treat compartment interface as trust boundary

How do break things up?

Depends on the attacker model & isolation mechanism

What isolation mechanisms can we use?

- Hardware-based isolation: ←
  - Physical machine, CPU modes (e.g., rings), virtual memory (MMU), memory protection unit (MPU), trusted execution environments, ...
- Software-based isolation:
  - Language virtual machines (e.g., JavaScript), software-based fault isolation (e.g., WebAssembly), binary instrumentation, type systems, ...

What isolation mechanisms can we use?

- Hardware-based isolation:
  - Physical machine, CPU modes (e.g., rings), virtual memory (MMU), memory protection unit (MPU), trusted execution environments, ...
- Software-based isolation:
  - Language virtual machines (e.g., JavaScript), software-based fault isolation (e.g., WebAssembly), binary instrumentation, type systems, ...

Example: Multi-user OS

- In this system:
  - Users can execute programs (process)
  - Processes can access resources/assets
- What's the threat model?
  -

What do we want?

- Memory isolation
  - ➤ Process should not be able to access another's memory
- Resource isolation
  - ➤ Process should only be able to access certain resources

What do we want?

- Memory isolation
  - Process should not be able to access another's memory
- Resource isolation
  - Process should only be able to access certain resources

UNIX permission model

- Permissions granted according to UID
  - ➤ A process may access files, network sockets, ...
  - ➤ root (UID 0) can access everything
- Each file has access control list (ACL)
  - ➤ Grants permissions to users according to UIDs and roles (owner, group, other)
  - ➤ Everything is a file!

How does `passwd` work then?

There is more than one UID…

Process UIDs

- Real user ID (RUID)
  - Used to determine which user started the process
  - Typically same as the user ID of parent process
- Effective user ID (EUID)
  - Determines the permissions for process
  - Can be different from RUID (e.g., because `setuid` bit on the file being executed)
- Saved user ID (SUID)

`setuid` demystified (a bit)

- A program can have a `setiud` bit set in its permissions
- This impacts: fork and exec
  - Typically inherit three IDs of parent
  - If `setuid` bit set: use UID of file owner as EUID

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

`setuid` demystified (a bit)

- There are actually three bits:
  - setuid - set EUID of process to ID of file owner
  - setgid - set EGroupID 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

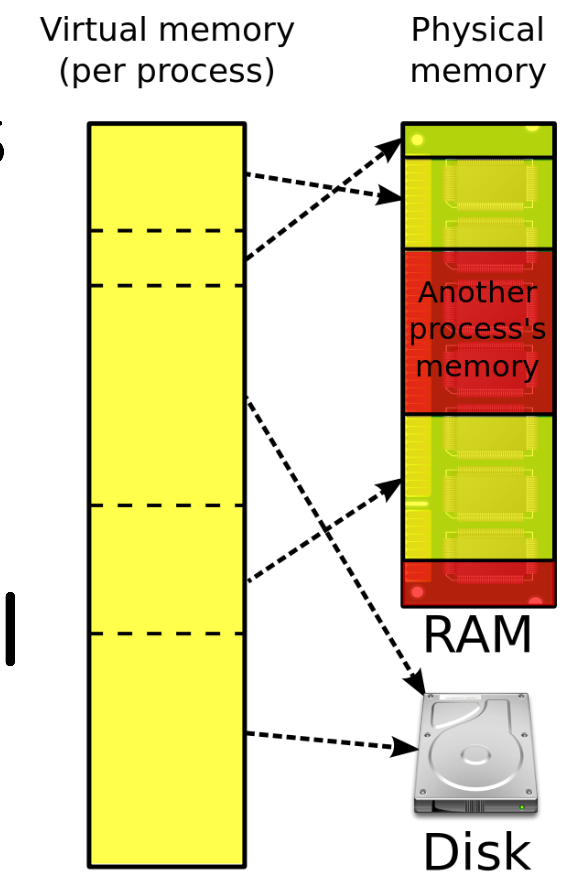
```
drwxrwxrwt 16 root root  700 Feb  6 17:38 /tmp/
```

What do we want?

- Memory isolation
  - Process should not be able to access another's memory
- Resource isolation
  - Process should only be able to access certain resources

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 goes through address translation (complete mediation)
  - Load, store, instruction fetch
- Who does the translation?

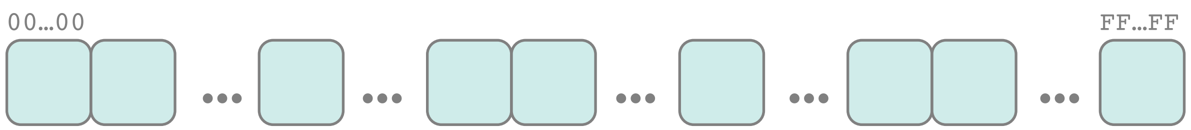
When do we do the translation?

- Every memory access goes through address translation (complete mediation)
  - 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)

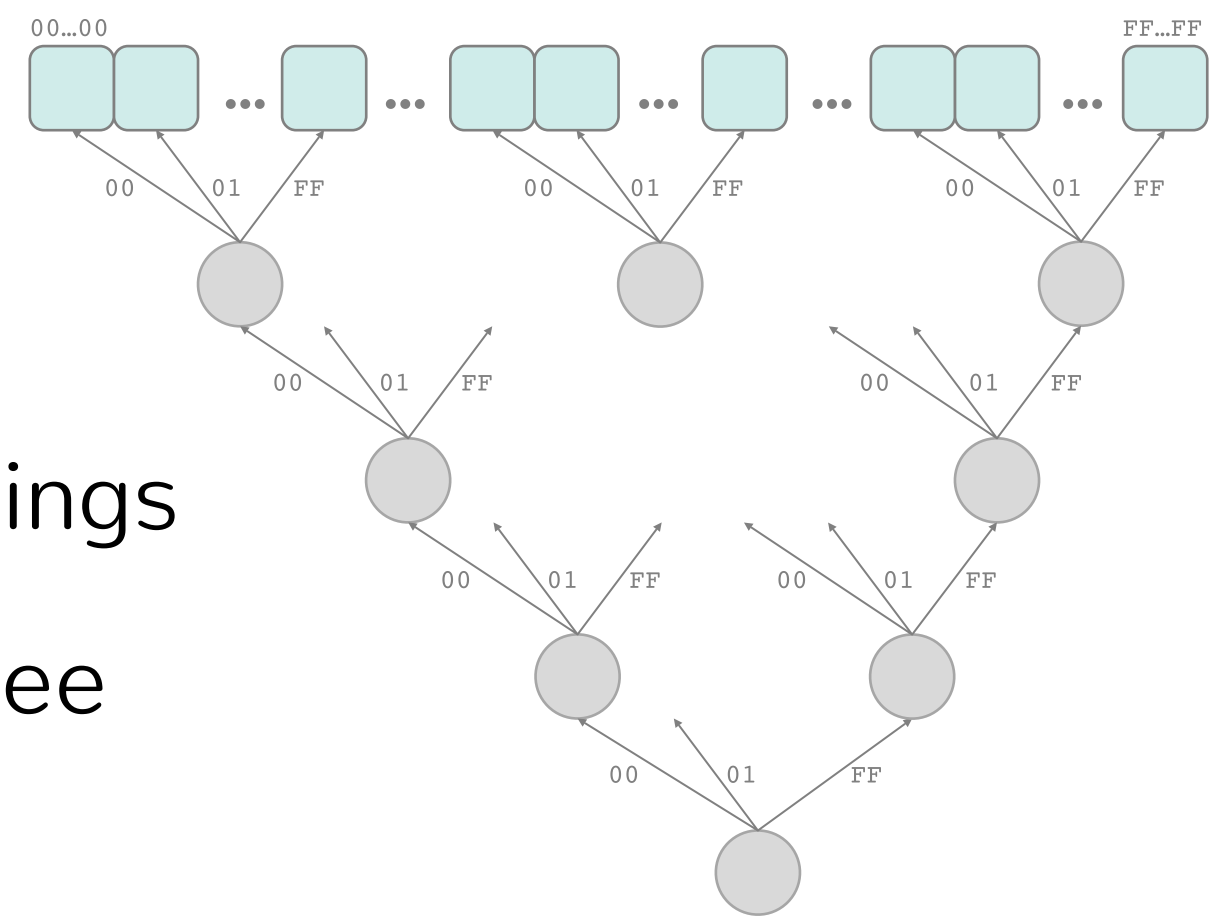


- Page: basic unit of translation
  - Usually 4KB = 2^{12}
- How many page mappings?
  - Still too big!
  - 52 bits * 2^{52} (208 petabytes)

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



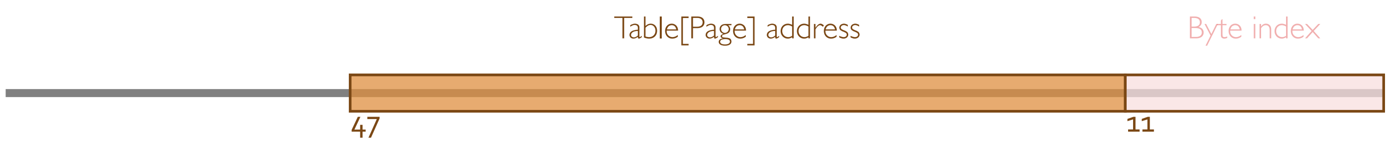
How do we get isolation between processes?

- 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

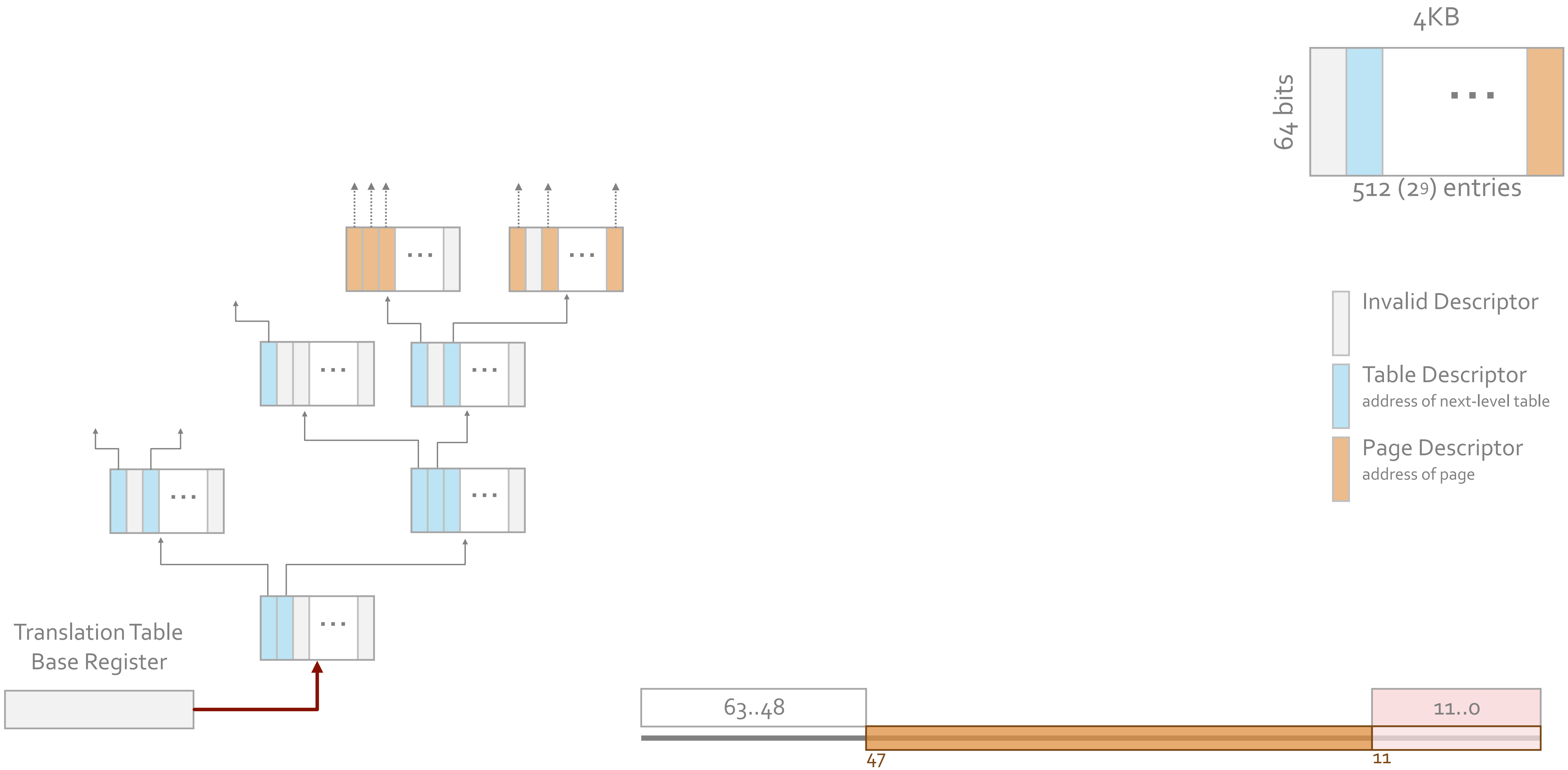
Example of page table walk

In reality, the full 64bit address space is not used.

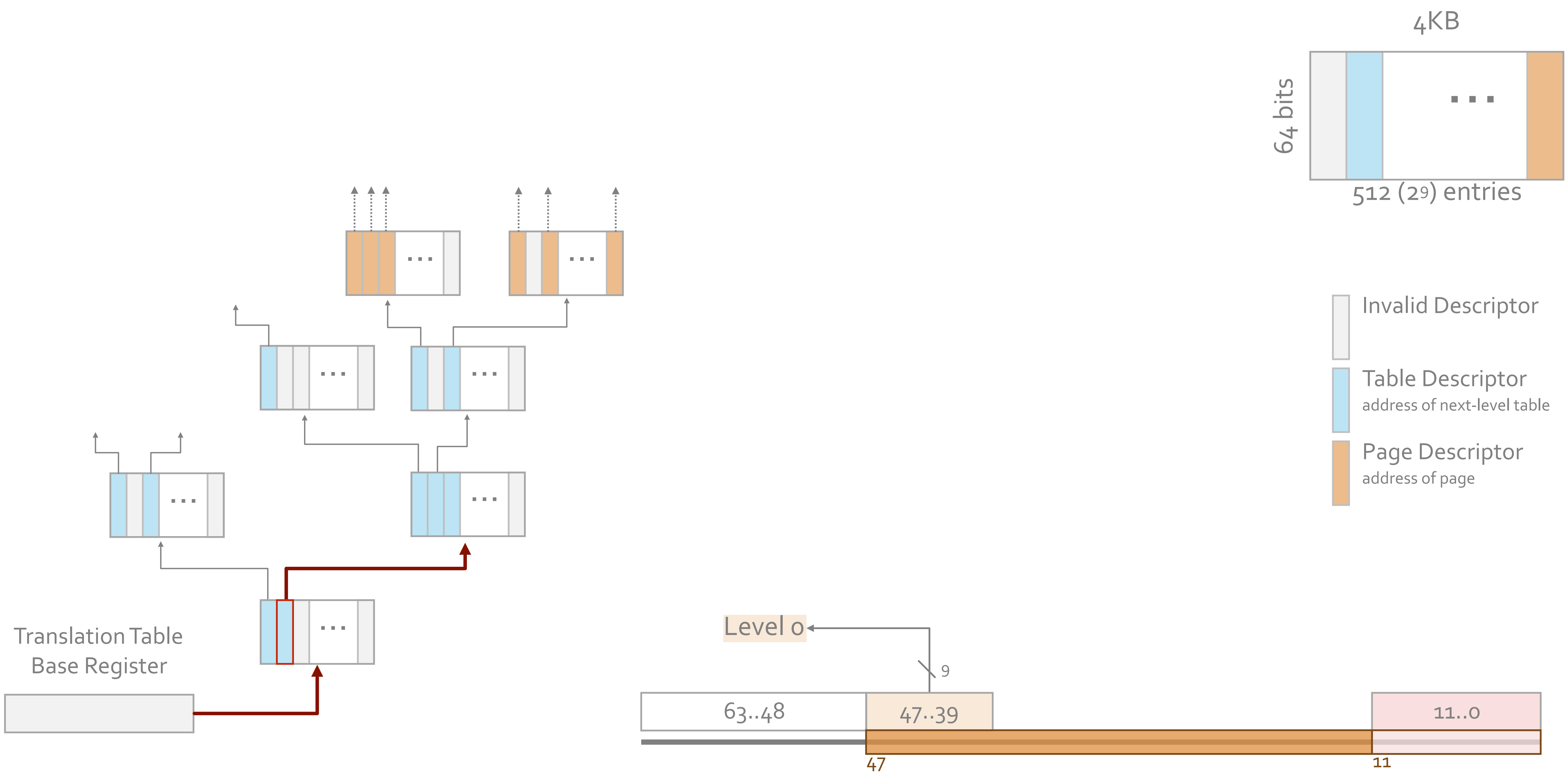
- Working assumption: 48bit addresses



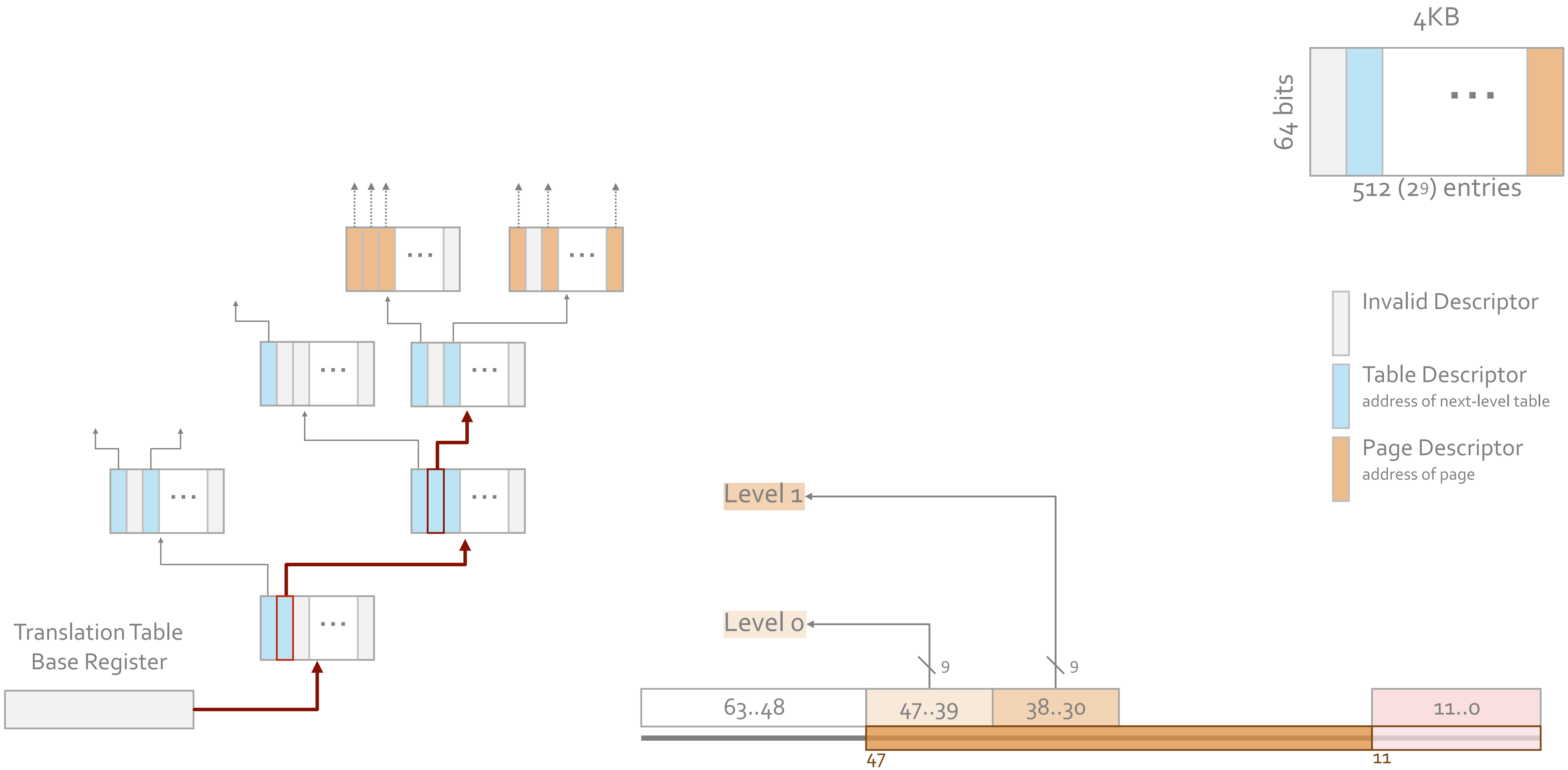
Page table walk



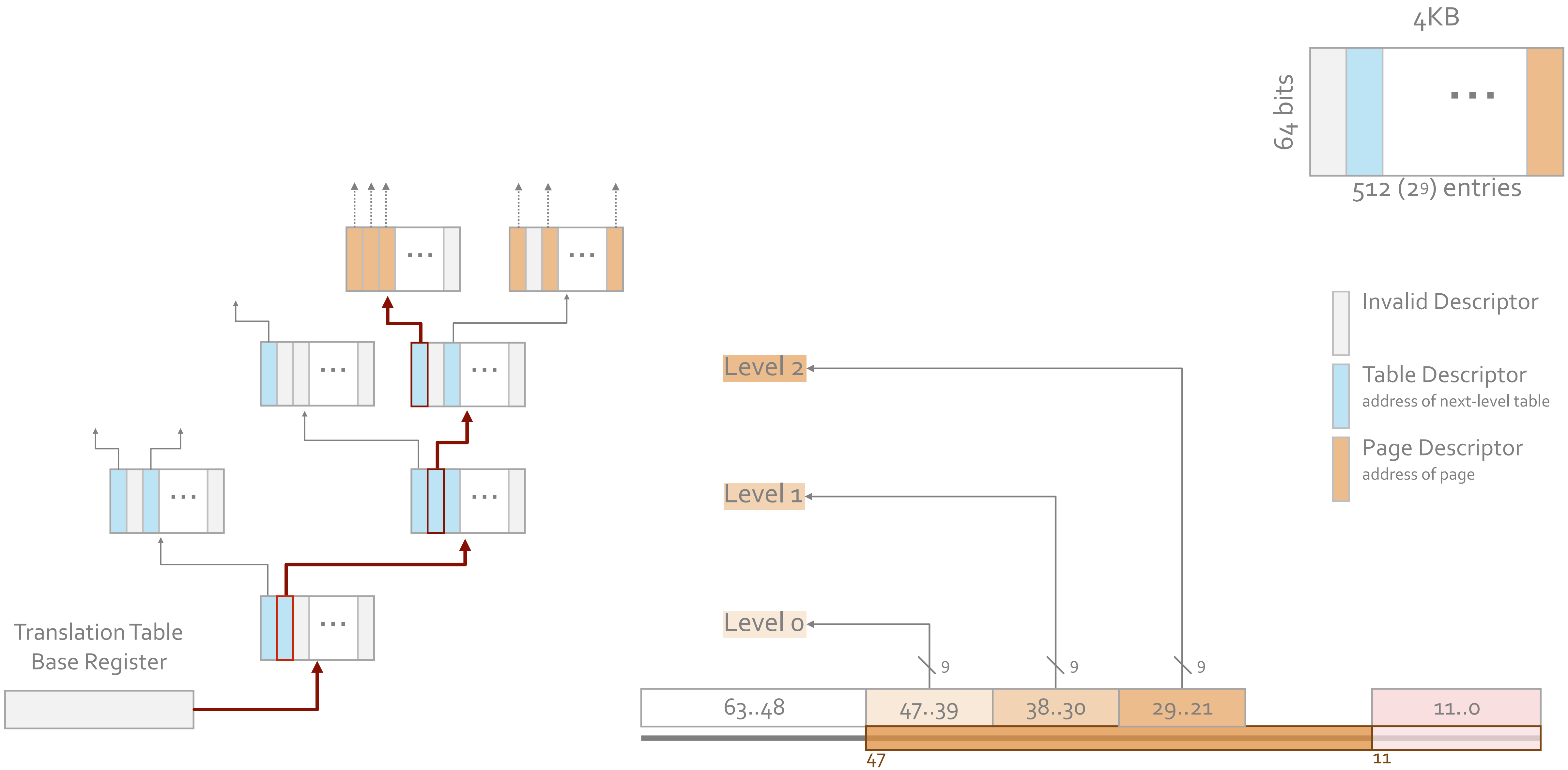
Page table walk



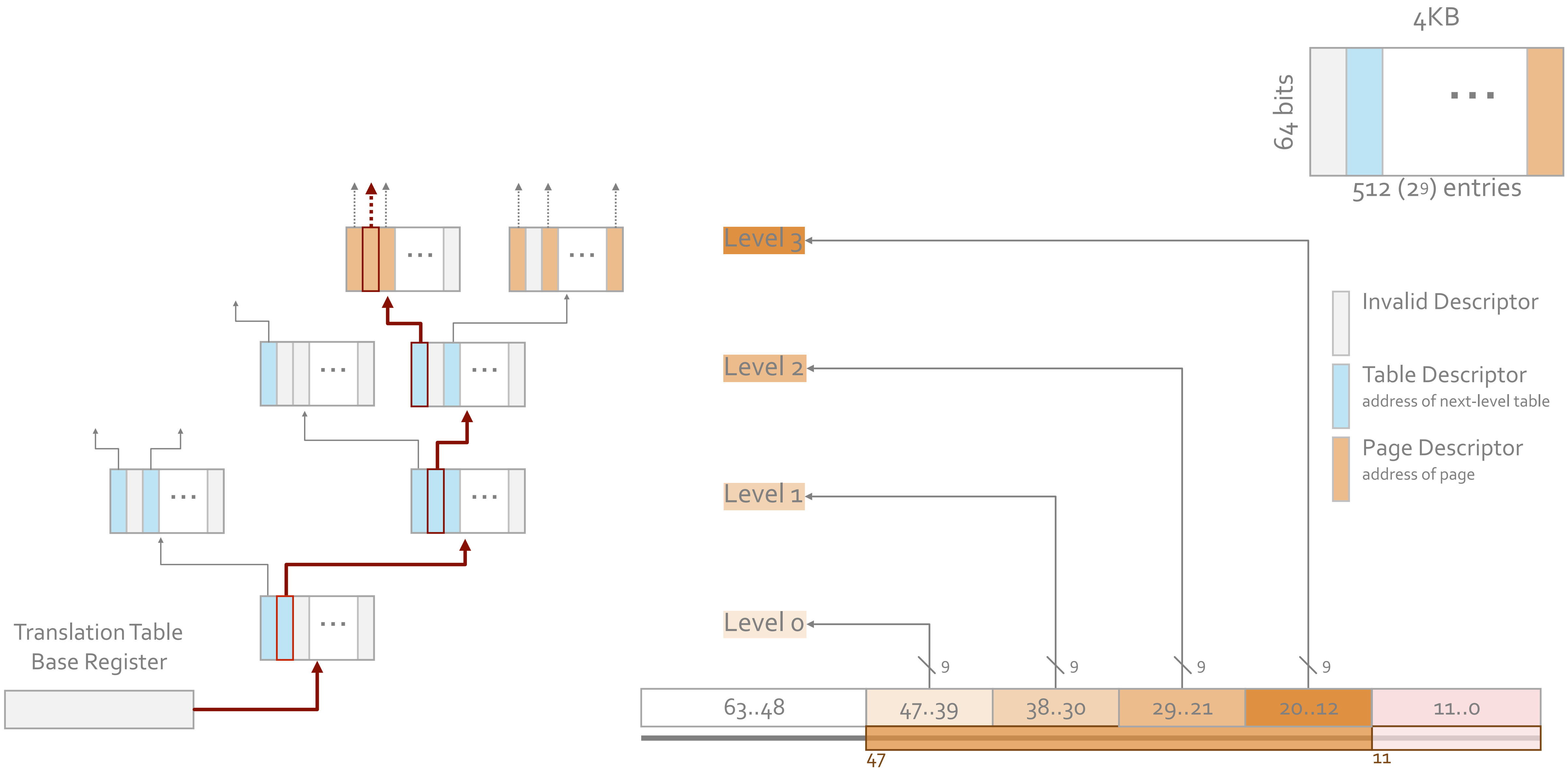
Page table walk



Page table walk



Page table walk



Make it fast: Translation Lookaside Buffer

Make it fast: Translation Lookaside Buffer

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

Make it fast: Translation Lookaside Buffer

- 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)

Make it fast: Translation Lookaside Buffer

- 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)
  - Access control: if mapping allows the mode of access

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!)

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 do we want?

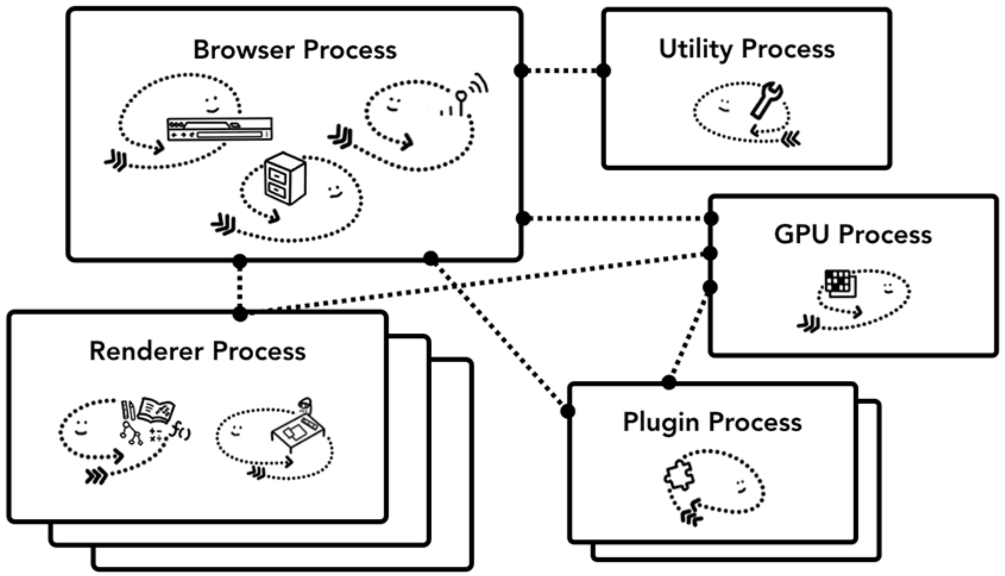
- Memory isolation ←
  - ➢ Process should not be able to access another's memory
- Resource isolation
  - ➢ Process should only be able to access certain resources

Process isolation and virtual memory are powerful abstractions... where else are they used?

Process isolation and virtual memory are powerful abstractions... where else are they used?

Example: Modern browsers

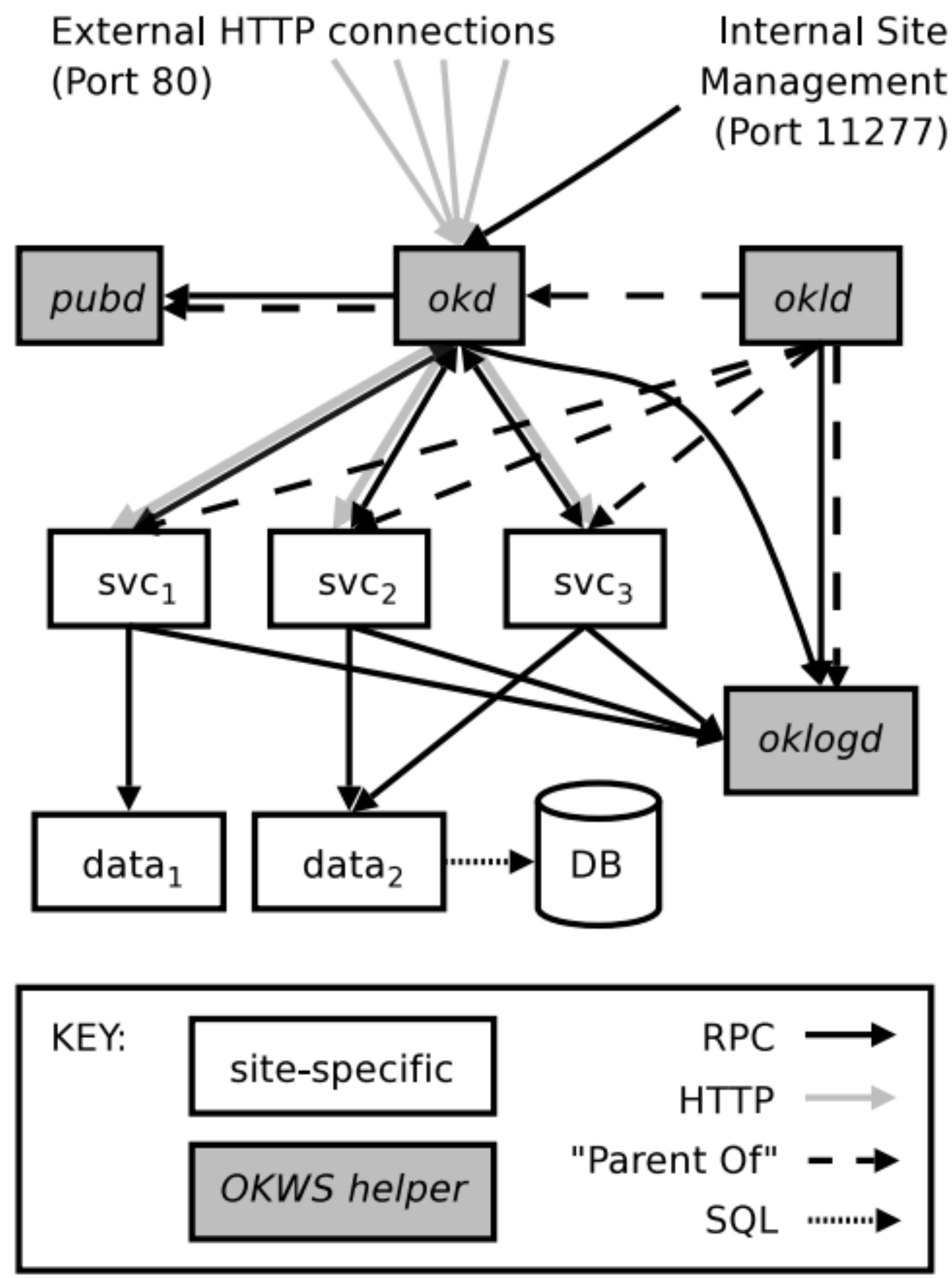
- Browser process
  - ➤ Handles the privileged parts of browser (e.g., network requests, address bar, bookmarks, etc.)
- Renderer process
  - ➤ Handles untrusted, attacker content: JS engine, DOM, etc.
  - ➤ Communication restricted to remote procedure calls
- Many other processes (GPU, plugin, etc)



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

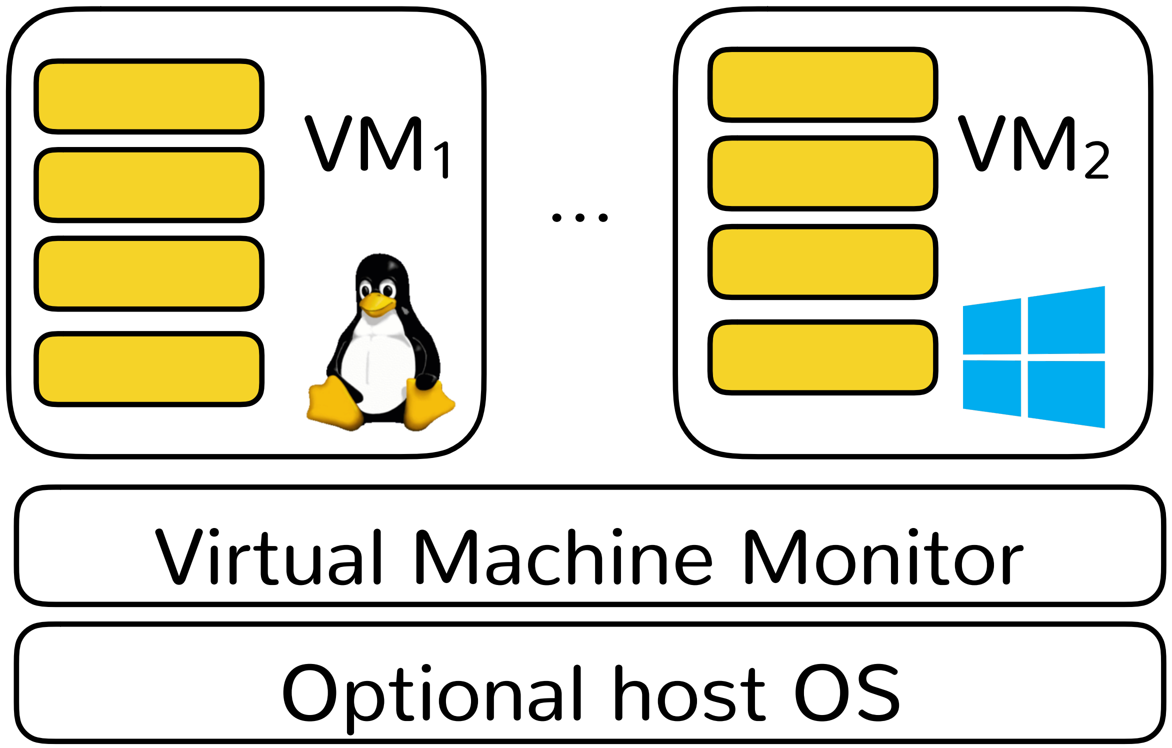
Example: OK$_{\text{Cupid}}$W$_{\text{eb}}$S$_{\text{erver}}$

- Privilege separate services
  - Each service runs with unique UID
  - Memory + FS isolation
- Communication limited to structured RPC



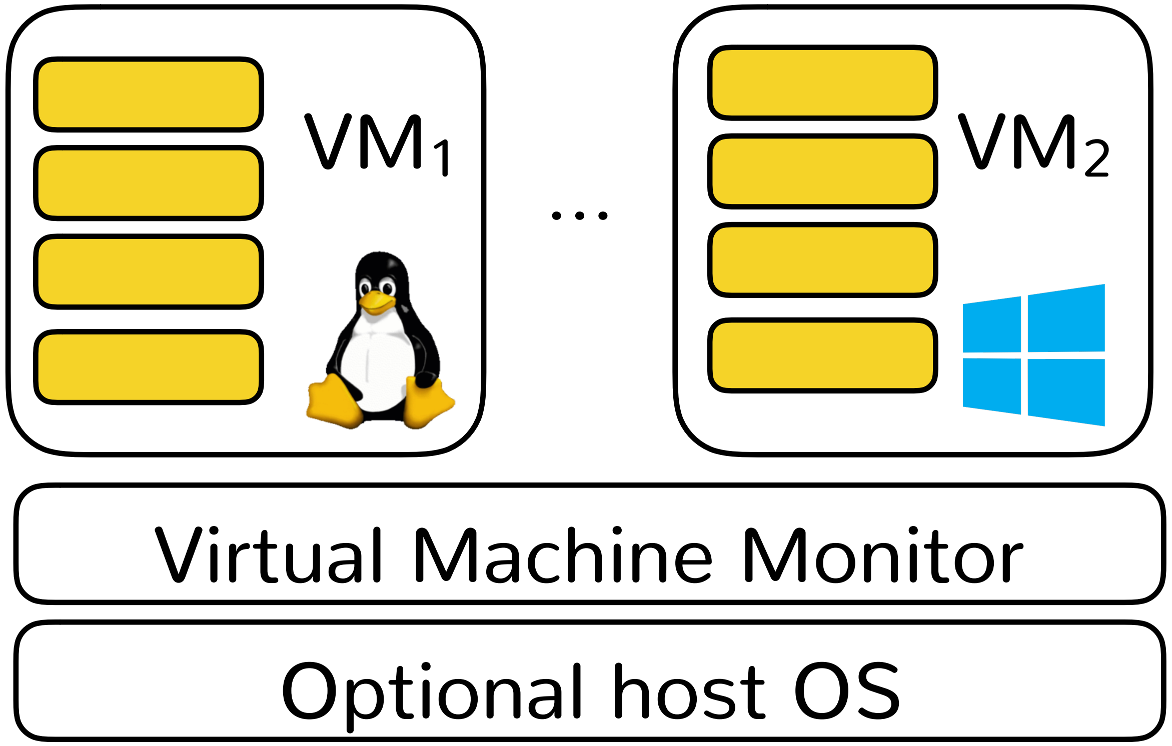
Process isolation and virtual memory are powerful abstractions... where else are they used?

Example: Virtual machines



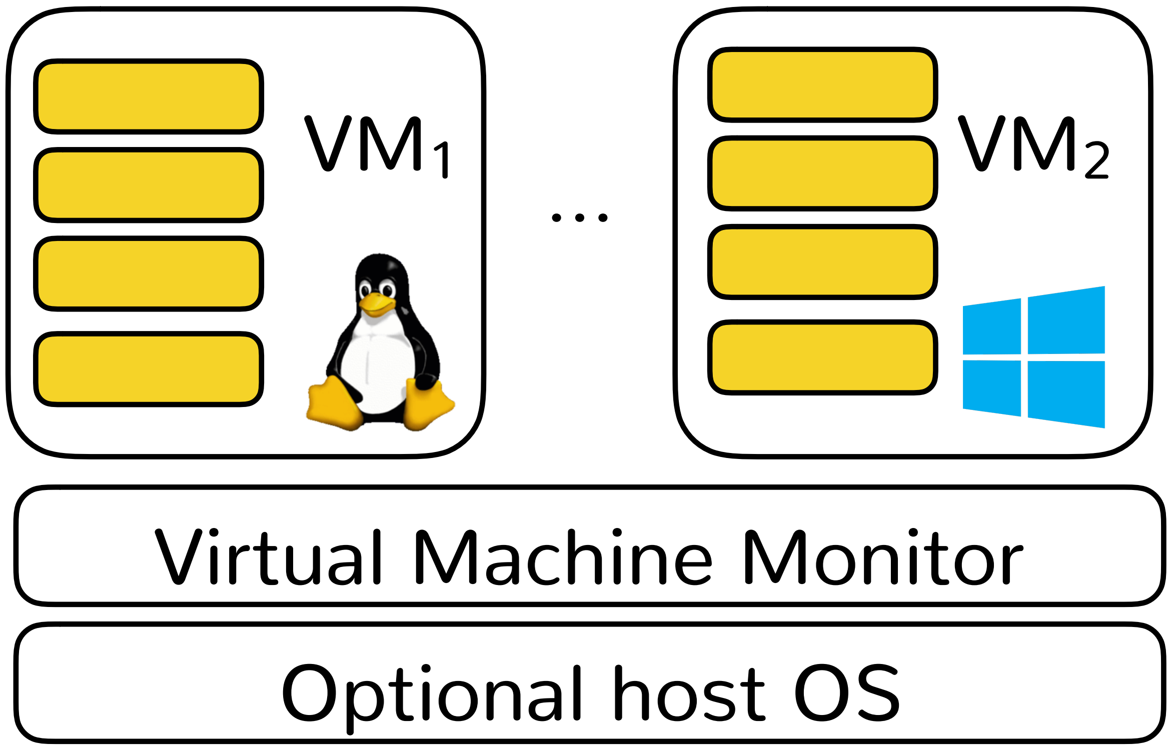
Example: Virtual machines

- Isolate VMs from each other
  - Nested page tables allows VM OS to map guest PA to machine PA



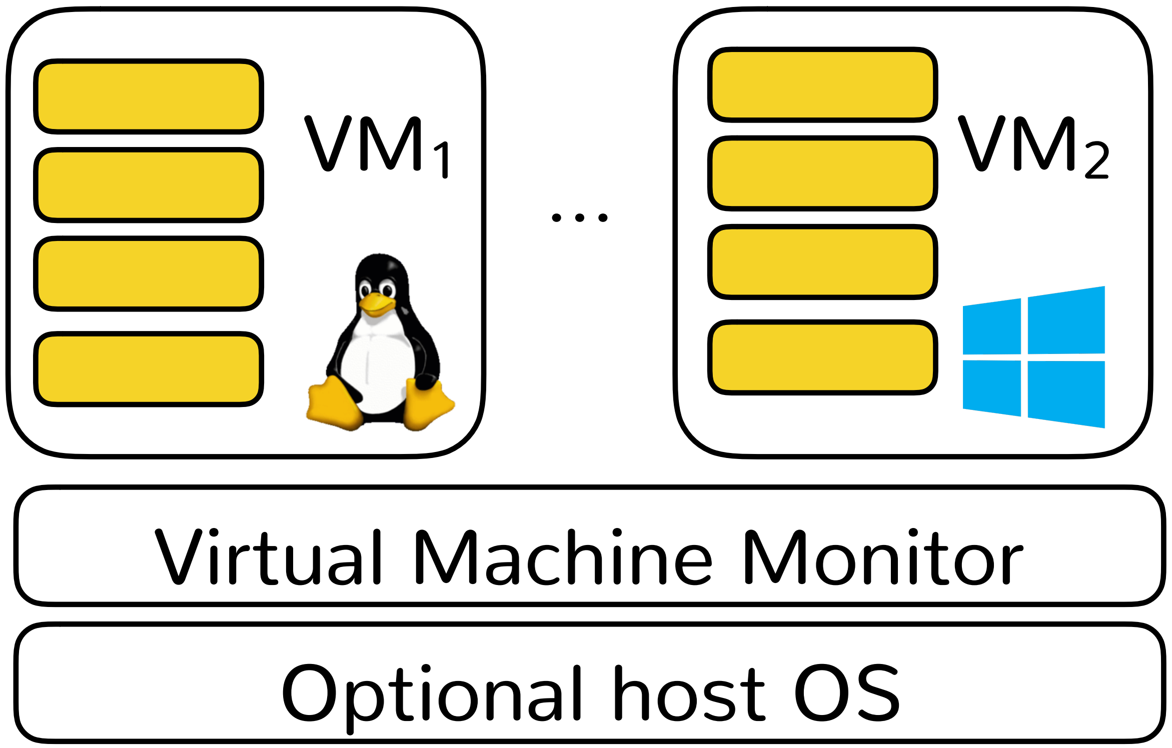
Example: Virtual machines

- Isolate VMs from each other
  - Nested page tables allows VM OS to map guest PA to machine PA
  - TLB entries are also tagged with VM ID (VPID)



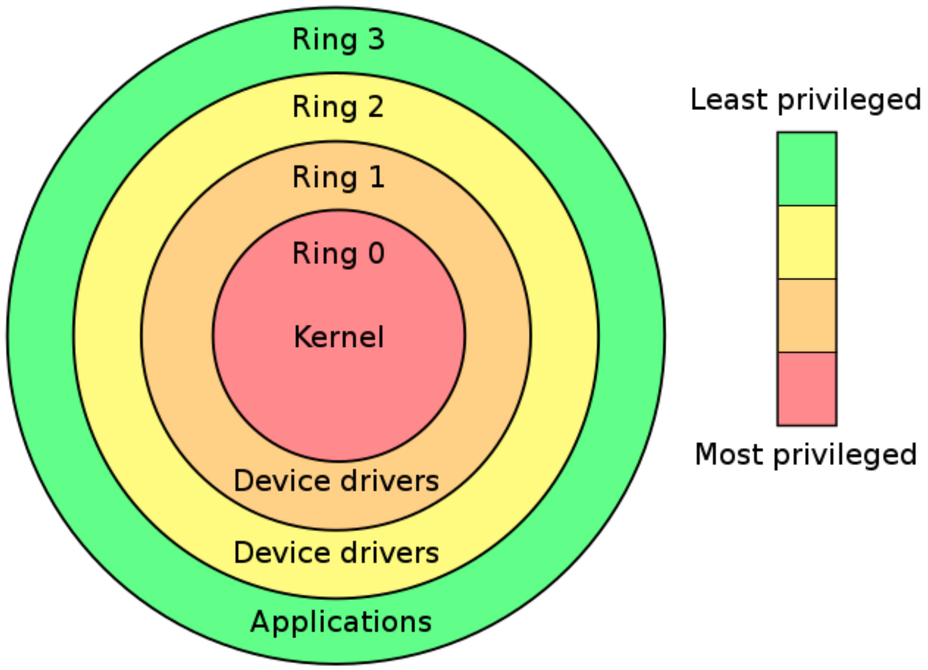
Example: Virtual machines

- Isolate VMs from each other
  - Nested page tables allows VM OS to map guest PA to machine PA
  - TLB entries are also tagged with VM ID (VPID)
- Interface between VMs and VMM: hypercalls



Example: Kernel isolation

- Kernel is isolated from user processes
  - ➤ Separate page tables
  - ➤ Processor privilege levels ensure userspace code cannot use privileged instructions
- Interface between userspace and kernel: syscalls



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

What isolation mechanisms can we use?

- Hardware-based isolation:
  - Physical machine, CPU modes (e.g., rings), virtual memory (MMU), memory protection unit (MPU), trusted execution environments, ...
- Software-based isolation:
  - Language virtual machines (e.g., JavaScript), software-based fault isolation (e.g., WebAssembly), binary instrumentation, type systems, ...

Software-based isolation

- Why would we want to isolate things in software?
  - ➢ Don’t have hardware-enforcement mechanism
  - ➢ Process abstraction is too costly

Software-based isolation

- How can we isolate components in software?
  - Memory isolation: instrument all loads and stores
  - Control flow integrity: ensure all control flow is restricted to CFG that instruments loads/stores
  - Complete mediation: disallow “privileged” instructions
  - Springboard and trampolines for crossing boundary

Software-based isolation (SFI)

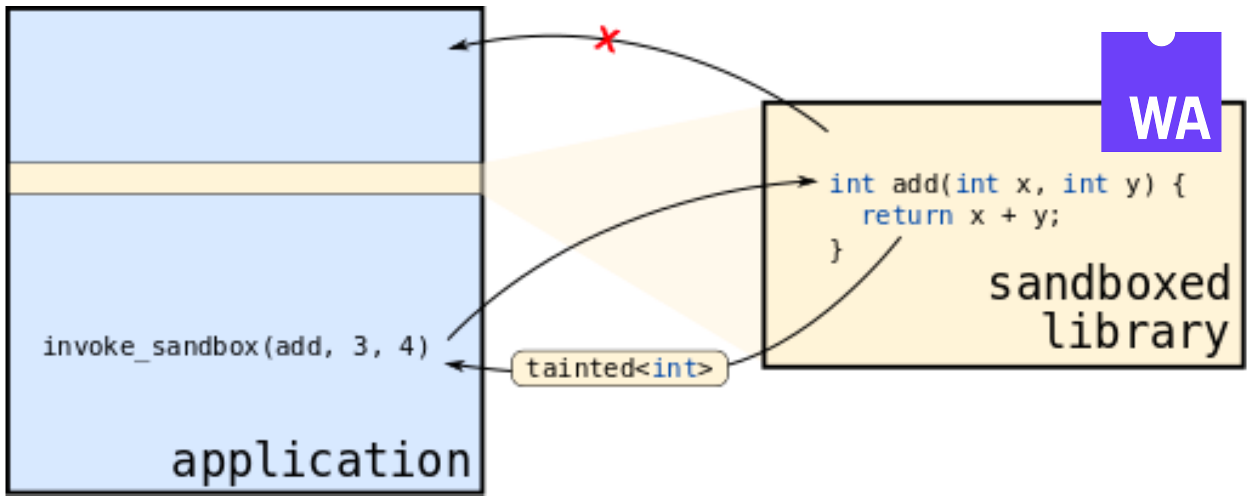
- How can we isolate components in software?
  - <u>Memory isolation:</u> instrument all loads and stores

Software-based isolation (SFI)

- How can we isolate components in software?
  - <u>Memory isolation:</u> instrument all loads and stores
  - <u>Control flow integrity:</u> ensure all control flow is restricted to CFG that instruments loads/stores
  - <u>Complete mediation:</u> disallow "privileged" instructions
  - Syscall-like interface between isolated code

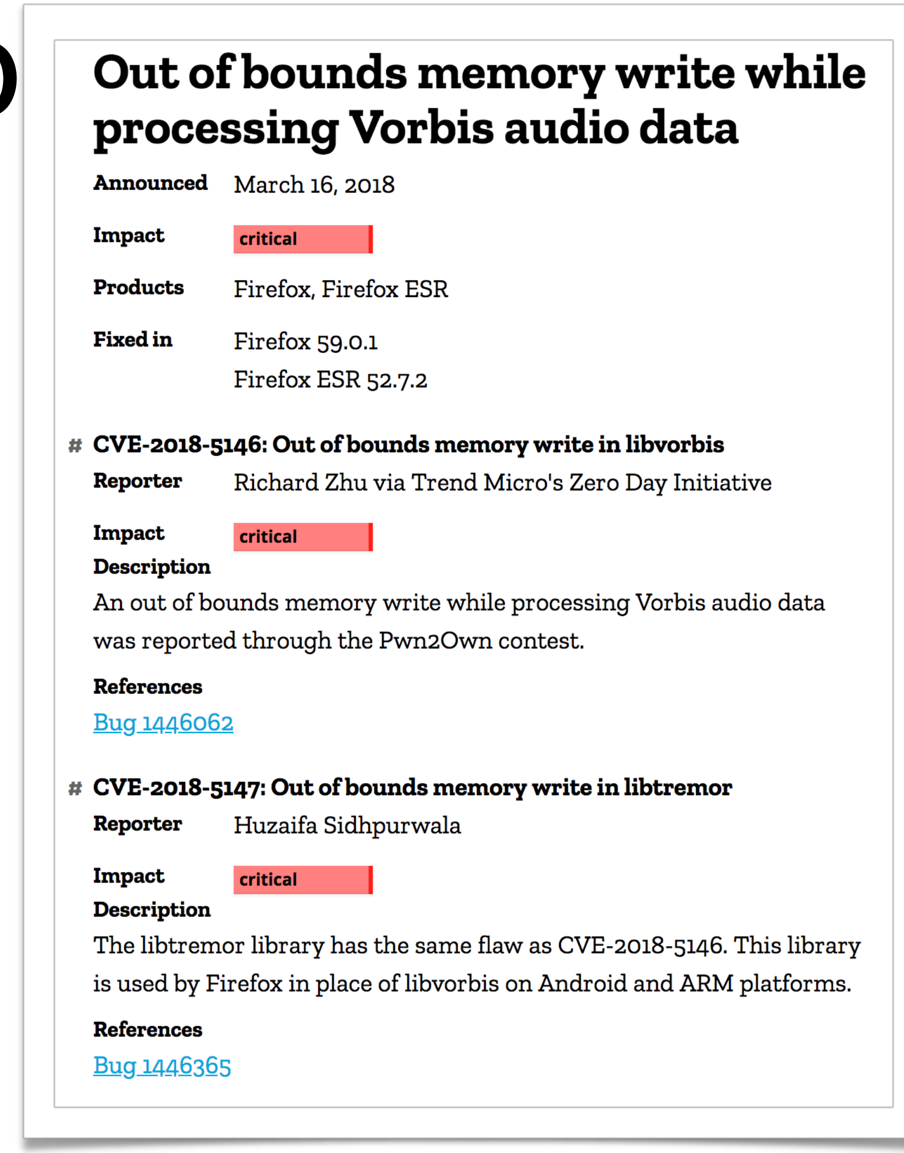
Example: library sandboxing in Firefox

- Privilege separate renderer by isolating libraries
  - Why?
  - Isolation in software via WebAssembly
- Interface between libs and Firefox is typed



Example: library sandb ox

- Privilege separate renderer by isolating libraries
  - Why?
  - Isolation in software via WebAssembly
- Interface between libs and Firefox is typed



Out of bounds memory write while processing Vorbis audio data

Announced March 16, 2018

Impact critical

Products Firefox, Firefox ESR

Fixed in Firefox 59.0.1
Firefox ESR 52.7.2

CVE-2018-5146: Out of bounds memory write in libvorbis

Reporter Richard Zhu via Trend Micro's Zero Day Initiative

Impact critical

Description

An out of bounds memory write while processing Vorbis audio data was reported through the Pwn2Own contest.

References

Bug 1446062

CVE-2018-5147: Out of bounds memory write in libtremor

Reporter Huzaifa Sidhpurwala

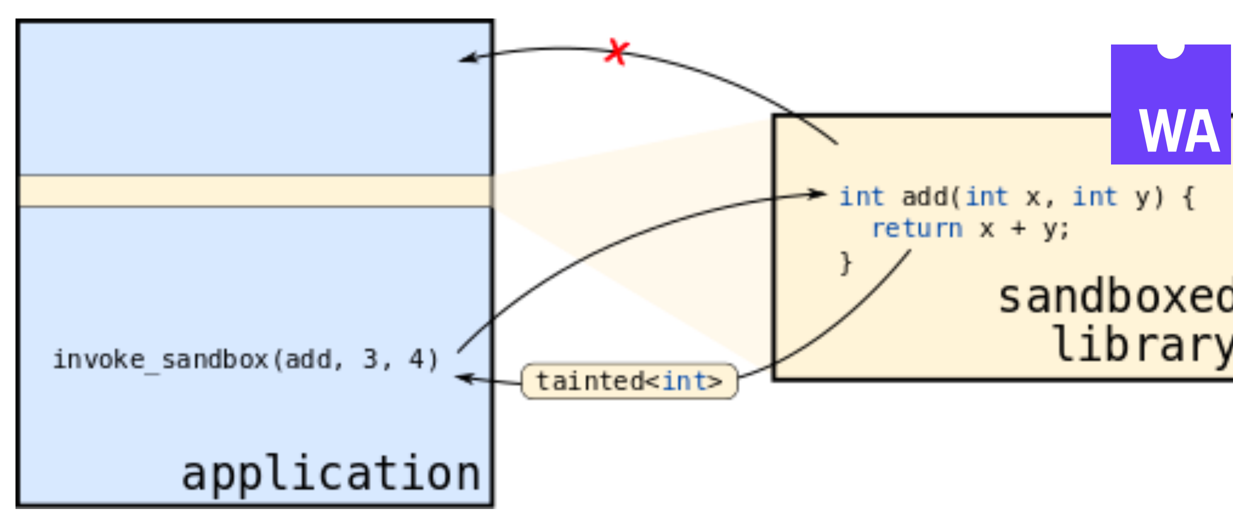
Impact critical

Description

The libtremor library has the same flaw as CVE-2018-5146. This library is used by Firefox in place of libvorbis on Android and ARM platforms.

References

Bug 1446365



Need to get the interface right

- Isolation is not enough
  - To do anything useful we typically need to cross trust boundaryIsolation is not enough
  - E.g., syscalls, hypercalls, runtime calls
- Need to ensure that the *calls are correct
  - Must keep track of whether you're operating on untrusted data or not
  - Incorrect implementations -> confused deputy attacks

Example: library isolation in Firefox

```
void create_jpeg_parser() {

 jpeg_decompress_struct jpeg_img;
 jpeg_source_mgr        jpeg_input_source_mgr;


 jpeg_create_decompress(&jpeg_img);
 jpeg_img.src = &jpeg_input_source_mgr;
 jpeg_img.src->fill_input_buffer = /* Set input bytes source */;


 jpeg_read_header(&jpeg_img /* ... */);
 uint32_t* outputBuffer =  /* ... */;

 while (/* check for output lines */) {
    uint32_t size = jpeg_img.output_width * jpeg_img.output_components;

    memcpy(outputBuffer, /* ... */, size);
 }
}
```

Can we make this easier? (Kernel)

- Explicit functions for copying data:
  - `copy_to_user()` and `copy_from_user()`
- HW to prevent kernel from accessing user data
  - ARM Privilege Access Never/Privileged eXecute Never
- Support for limiting and filtering system calls
  - E.g., browsers use seccomp-bpf to restrict the syscall interface of untrusted processes (and thus pwnage via kernel exploitation)

Can we make this easier? (browser)

- Restrict interface to RPC
  - Generate RPC interface from interface description languages
  - RPC ensure type and memory safety
- Tainted types (RLBox)
  - Eliminate confused deputy attacks by forcing trusted code to validate all untrusted data before using it

Example: library isolation in Firefox

```
void create_jpeg_parser() {

 auto sandbox = rlbox::create_sandbox<wasm>();
 tainted<jpeg_decompress_struct*> p_jpeg_img = sandbox.malloc_in_sandbox<jpeg_decompress_struct>();
 tainted<jpeg_source_mgr*>        p_jpeg_input_source_mgr = sandbox.malloc_in_sandbox<jpeg_source_mgr>();

 sandbox.invoke(jpeg_create_decompress, p_jpeg_img);
 p_jpeg_img->src = p_jpeg_input_source_mgr;

 p_jpeg_img->src->fill_input_buffer = /* Set input bytes source */;

 sandbox.invoke(jpeg_read_header, p_jpeg_img /* ... */);
 uint32_t* outputBuffer =  /* ... */;

 while (/* check for output lines */) {
    uint32_t size = (p_jpeg_img->output_width *  p_jpeg_img->output_components).copy_and_verify(
        [](uint32_t val) -> uint32_t {
           assert(val <= outputBufferSize);
           return val;
        });
    memcpy(outputBuffer, /* ... */, size);
 }
}
```

Today

Lecture objectives:

- Understand basic principles for building secure systems
- Understand mechanisms used to build secure systems