ORACLE

# Towards an Alternative Guest Memory Architecture

**Joao Martins**
Snr. Principal Software Engineer

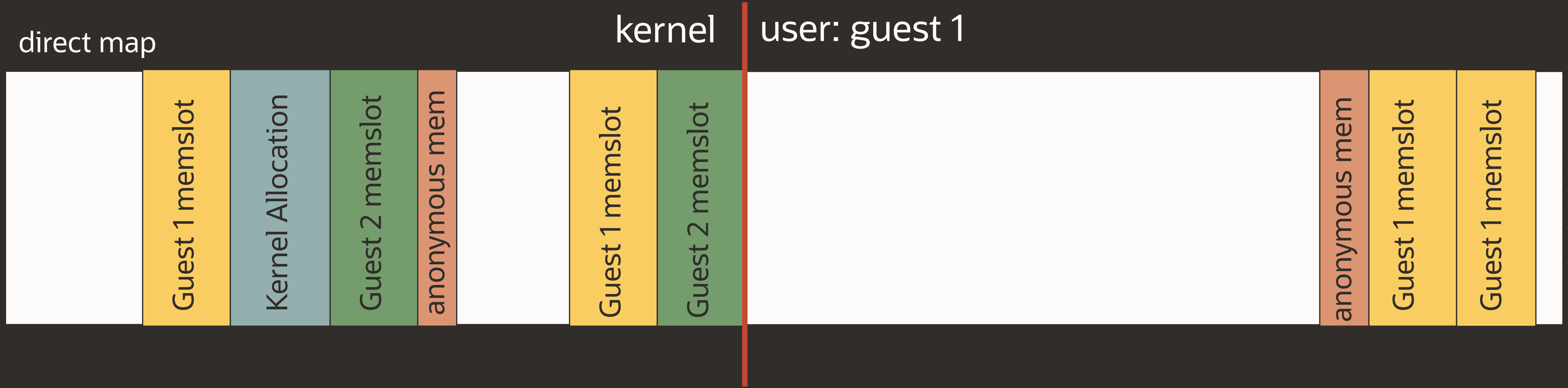
# Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

# Motivation

## The direct map

- Linux keeps a direct map of all physical memory
- This address space is modified in many ways (ioremap, add_pages, boot)
- Portions are defined by various metadata (e.g. struct page, memblock)
- What reflected in page tables maybe managed in higher order chunks



## Metadata

- The direct map is backed by metadata called `struct page`
- Used by page cache and anon, required for most kernel services
- Tracks references to a PFN, mappings, amongst others
- Usually accessible via get_user_pages() and PFN to metadata conversions

## `struct page`/metadata overheads

- **64 bytes** in size per PAGE_SIZE (4096 on x86_64)
  - KVM adds another 8bytes per spte
  - PTEs another 8byte per 4K albeit amenable to hugepte
  - "Just" 1.75% of host physical memory, right?
- Extrapolating:
  - 32 - 36 GB for a 2Tb machine
  - 128 - 160 GB for an 8Tb machine
  - 1.2 TB for a 64Tb machine

## Security vulnerabilities

Spectrev1: speculatively execute arbitrary instructions that load data from the direct map.

Spectrev2: similar to variant 1 but for indirect branches.

Spectrev3: could leak any memory from the direct map in userspace

Spectrev4: utilize values speculatively read out-of-order to leak data mapped by the kernel

L1TF / MDS: data in the direct map could be leaked through CPU resources like the L1 data cache for L1TF, or micro-architectural buffers (MDS)

Most mean that data mapped by the CPU can be leaked:

**kVA address space maps all memory, therefore all is leakable**

## Can we do better for hypervisors?

- `struct page` does not **really** reflect underlying page size in the page tables
- Modern platforms won’t need most kernel services (e.g no I/O)
- Efficiency lost in a mostly idle structure
- While potentially leaking customer data through the direct map

# Removing struct page?

## /dev/mem and mem=NNN

- A metadata-less map with a single contiguous chunk of any memory kind
- Only 4K PTEs (no huge pages)
- No access control between multiple VMMs
- VMM needs to manage multiple VMAs with different page offsets
- No way to give that memory back

  Can target *any* memory

# The Device DAX Interface

- Character device mmap-able providing strict direct access to memory
- Memory map initialized at device creation
- Gives control to application on clearing, access, alignment, exceptions
- Provides facilities to return memory back to the kernel (dax_kmem)

An emulated devdax device through `memmap=X@Y`

Requires user knowledge of memory map to select a RAM range

*Note: devdax != (pmem || fsdax)*

# Device DAX Limitations

- Inherits design of NVDIMM persistent namespaces
  - No support for discontiguous regions
- Long device initialization times due to ‘struct page’ init/clearing
- No use of compound pages (despite providing hugepage ptes)

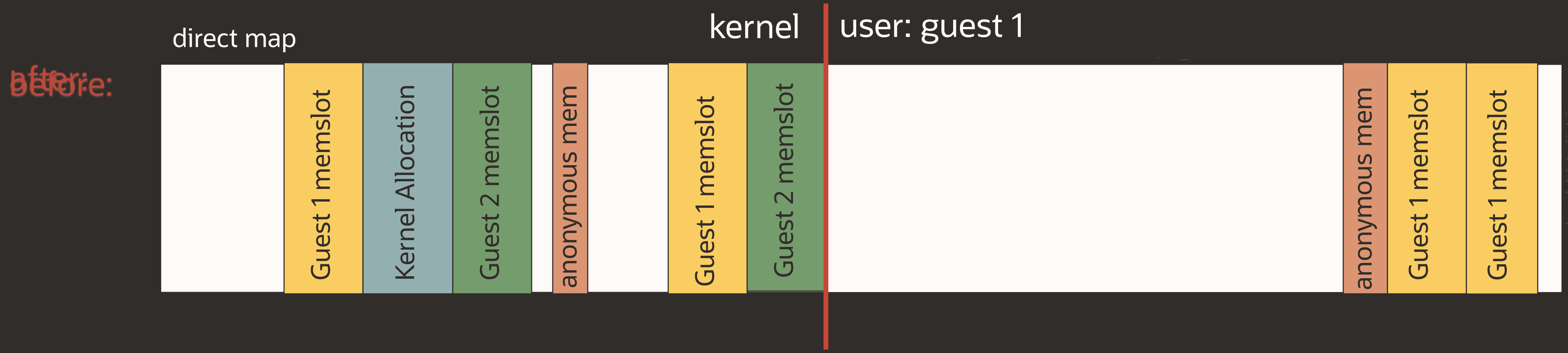
Need arch support for devmap PTEs

## Device DAX for volatile memory (`dax_hmem`)

- Removes the need for the memmap=X@Y hack
- Uses **actual** conventional memory in EFI
  - EFI memory map entries marked with EFI_MEMORY_SP attribute
- Ability for firmware to dedicate memory for userspace
- Will also be used when platforms support HMAT (e.g. HBM2)
- Application can pick ranges to create or restore a memory device (kexec)
  - Or use DAX simple range allocator

# Device DAX without struct page

- Introduce a VM_PFNMAP mmap() with static PFN mappings
- Leverage existing KVM support for VM_PFNMAP (by Karim Ahmed)
- 2M/1G Hugepage support for PAGE_SPECIAL
- Fix the cop out of MCEs on memory not managed by kernel
  Reserve memory types for PFNs to mapped as WB



# Example usage

Adding to kernel command line (example host is 386G per node, on 2 nodes):

```
efi_fake_mem=368G@16G:0x40000,368G@400G:0x40000
```

Dumping ranges or dax regions:

```
$ cat /proc/iomem | grep Soft
        200000000-ffffffff : Soft Reserved
        1200000000-1ffffffff : Soft Reserved
$ daxctl list -Riu
```

Using it on Qemu:

```
-m 30G
-object memory-backend-file,id=mem,size=30G,mem-path=/dev/dax0.1,share=on,align=1G
-numa node,memdev=mem
```

Managing the region:

```
$ daxctl create-device -s 30G [--no-metadata] -a 1G -r 0
$ daxctl disable-device dax0.1 && daxctl destroy-device dax0.1
$ daxctl disable-device dax0.1 && daxctl reconfigure-device dax0.1 -s 16G
```

## Use cases for pageless/secret memory in KVM

- Let KVM bind these DAX devices
- Replace all SLAB/SLUB usage call sites with this pageless/secret memory

  e.g. struct kvm_ioapic ; kvm_vcpu ; struct kvm_pit; vcpu arch data

- Userspace VMM usage as a global memory pool for all its allocations

## Advantages

- Guest memory not as prone to leakage
- Gain a ton of memory back for more guests
- Memory preserved for VMM fast live restart
- No need to hunt all those SpectreV1 gadgets

## Pitfalls

- Case by case basis to support no struct page
- Stripped of most kernel services
  (only KVM, mm, VFIO*)
- Needs better tracking across subsystems
  (i.e. follow_pfn() not enough?)
- I/O only copy based
  (e.g. `vhost_net.experimental_zcopytx=0`)

# Future directions?

# ASI and Pageless Memory

- Deny list (Pageless) versus Allow list (ASI)
- Both complement each other: protecting VMM and kernel private data
- Perhaps a potential performance improvement?

## Huge Pages and less struct pages?

- I/O could work better without struct pages
- Remove assumption over PAGE_SIZE/PAGE_MASK in offset computation
- Letting subsystems learn to only deal with head pages
- Example is recent support for Large Pages in the page cache work

# Huge Pages and less struct pages

- Do we need all those pages for a preallocated hugetlbfs/DAX page?
- What falls apart if portions of vmemmap reuse the same pages?

  https://lore.kernel.org/linux-mm/20200915125947.26204-1-songmuchun@bytedance.com/

- DAX support for “regular” huge pages

## Current Status and Conclusions

- 5.10 will have improved partitioning of DAX regions
- `struct page` removal and better DAX hugepages support effort to continue
- Tries to introduce a boundary between hypervisor and guest memory
- Taught us that the kernel mapping isn't always necessary

# Resources

- struct page, The linux physical page frame data structure

  https://blogs.oracle.com/linux/struct-page%2C-the-linux-physical-page-frame-data-structure

- pageless DAX initial proposal: http://lore.kernel.org/r/20200110190313.17144-1-joao.m.martins@oracle.com

  latest: https://github.com/jpemartins/linux pageless-dax

# Thank you

**Questions?**