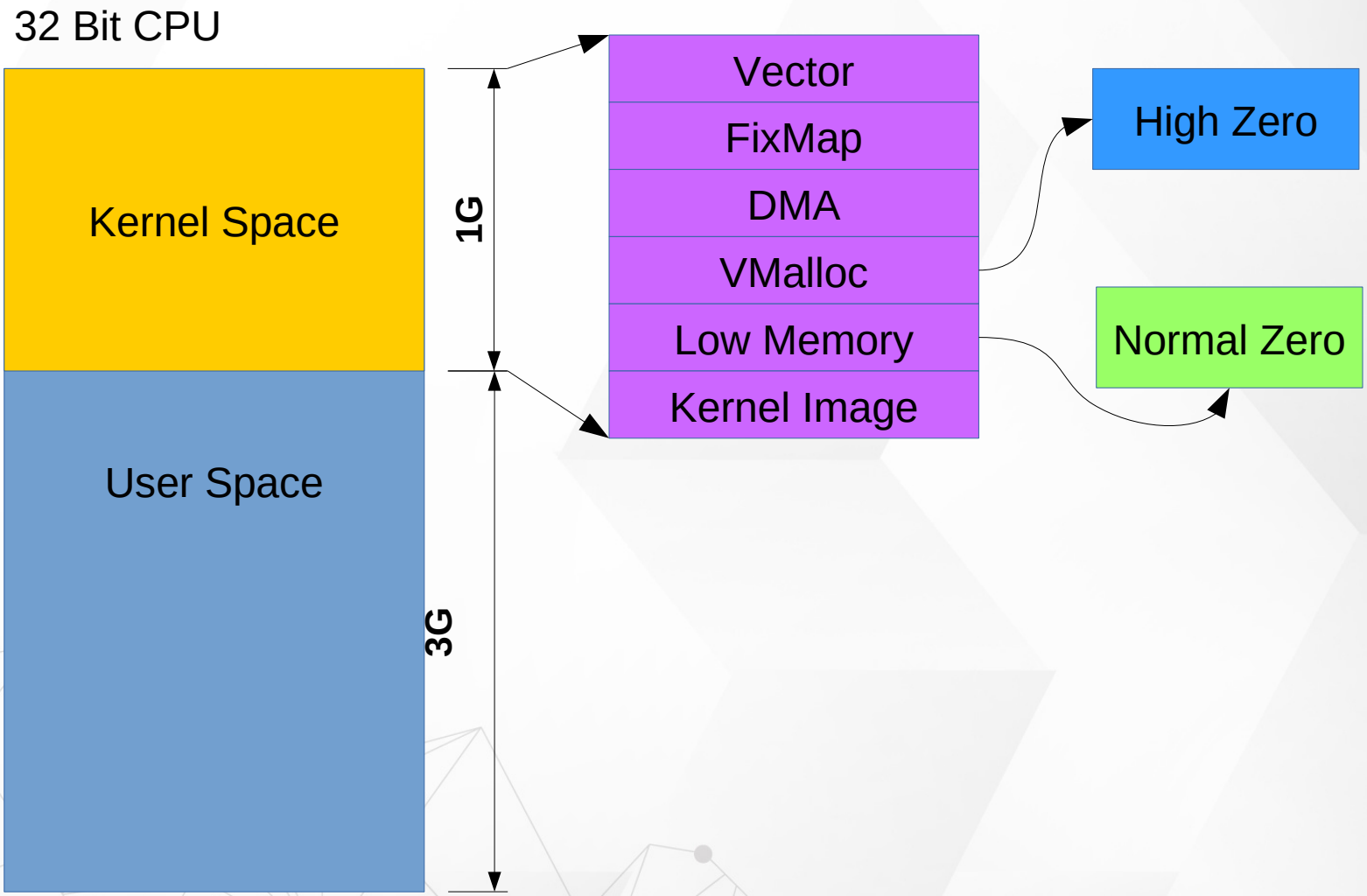
Allocating Memory

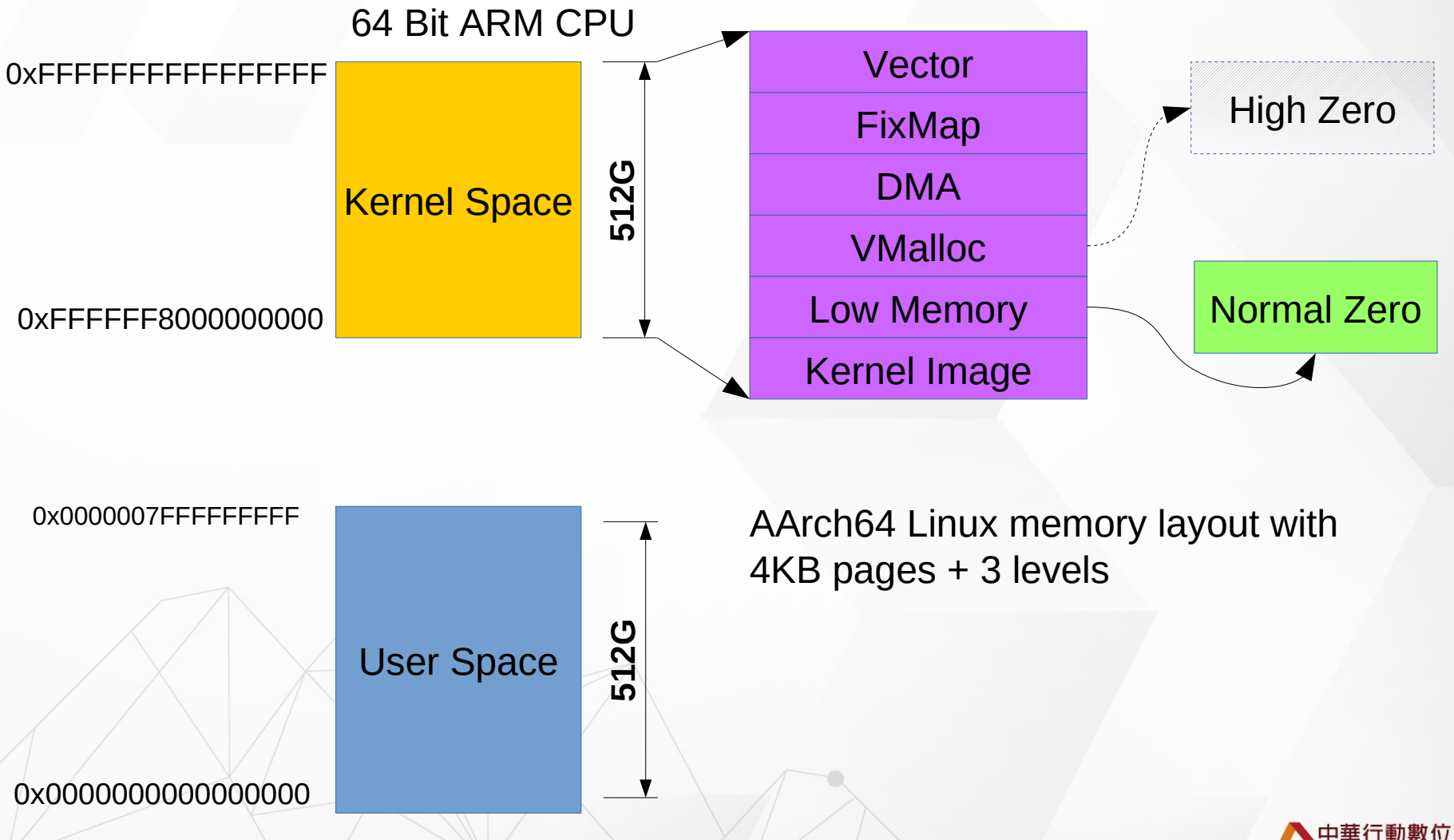
Kernel Allocate Memory API

- **kmalloc()**
- **kfree**
- __get_free_page()
- __free_page()
- __get_free_pages()
- __free_pages()
- vmalloc()
- vfree()
- **ioremap()**
- **iounmap()**

Kernel Memory Configuration



Kernel Memory Configuration



AArch64 Linux memory layout with 4KB pages + 3 levels

The Flags Argument

- **kmalloc** prototype

```
#include <linux/slab_def.h>
void *kmalloc(size_t size, int flags);
```

- **Flags**
  - **GFP_KERNEL**
    - the most commonly used flag
    - **Cannot** be used in **atomic** context

kmalloc

- kmalloc(size_t size, gfp_t flags);
  - Allocates **consecutive** virtual/physical memory pages
  - Offset by **PAGE_OFFSET**
    - **PAGE_OFFSET 2^12 (4K)**
  - No changes to page tables
  - Allocate memory limit : **128K**

Memory Zones

- Normal memory
  - **GFP_KERNEL**
- DMA-capable memory
  - **__GFP_DMA**
  - Platform dependent
  - First 16MB of RAM on the x86 for ISA devices
  - PCI devices have no such limit
- High memory
  - **__GFP_HIGHMEM**
  - Platform dependent
  - > 32-bit addressable range

Driver SHT21

```
static int sht21_probe(struct i2c_client *client,
    const struct i2c_device_id *id)
{
    struct device *dev = &client->dev;
    struct device *hwmon_dev;
    struct sht21 *sht21;

    if (!i2c_check_functionality(client->adapter,
                    I2C_FUNC_SMBUS_WORD_DATA)) {
        dev_err(&client->dev,
            "adapter does not support SMBus word transactions\n");
        return -ENODEV;
    }

    sht21 = devm_kzalloc(dev, sizeof(*sht21), GFP_KERNEL);
    if (!sht21)
        return -ENOMEM;

    sht21->client = client;

    mutex_init(&sht21->lock);

    hwmon_dev = devm_hwmon_device_register_with_groups(dev, client->name,
                                sht21, sht21_groups);
    return PTR_ERR_OR_ZERO(hwmon_dev);
}
```

kmalloc and flag

The Size Argument

- Kernel manages physical memory in **pages**
  - Needs special management to allocate small memory chunks
- Linux creates pools of memory objects in fixed sizes
  - 32-byte
  - 64-byte,
  - 128-byte memory objects
- **`kmalloc create smallest allocation unit`**
  - 32 or 64 bytes
- Largest portable allocation unit is **128KB**

Supplement

get_free_page and Friends

- For allocating **big chunks of memory**, it is more efficient to use a **page-oriented** allocator

- To allocate pages, call

Kernel API

```
/* returns a pointer to a zeroed page */
get_zeroed_page(unsigned int flags);

/* does not clear the page */
__get_free_page(unsigned int flags);

/* allocates multiple physically contiguous pages */
__get_free_pages(unsigned int flags, unsigned int order);
```

get_free_page and Friends

- **flags**
  - Same as **flags** for **kmalloc**
- **order**
  - Allocate 2^{order} pages
    - **order** = 0 for 1 page
    - **order** = 3 for 8 pages
  - Can use **get_order(size)** to find out **order**
  - Maximum allowed value is about 10 or 11
  - See **/proc/buddyinfo** statistics

Page and order

get_free_page and Friends

- Subject to the same rules as **`kmalloc`**
- To free pages, call

```
void free_page(unsigned long addr);
void free_pages(unsigned long addr, unsigned long order);
```

- Make sure to free the same number of pages
  - Or the memory map becomes corrupted

Sample – get_free_page (1)

3-1-2-buddy

```
static int buddy_init(void)
{
    printk("Buddy driver installed...\n");

    order = 4;

    ptr = (void *)__get_free_pages(GFP_KERNEL, order);
    if (ptr)
        printk("2^%d * %ld bytes allocated\n", order, PAGE_SIZE);

    order2 = get_order(PAGE_SIZE * 4);
    ptr2 = (void *)__get_free_pages(GFP_KERNEL, order2);
    if (ptr2)
        printk("2^%d * %ld bytes allocated\n", order2, PAGE_SIZE);

    return 0;
}
```

Sample – get_free_page (2)

```
static void buddy_exit(void)
{
    printk("buddy driver removed...\n");
    printk("buddy driver removed order  %d ...\n", order);
    printk("buddy driver removed order2  %d ...\n", order2);

    free_pages((unsigned long)ptr, order);
    free_pages((unsigned long)ptr2, order2);
}
```

vmalloc and Friends

- Allocates a **virtually contiguous** memory region
  - Not consecutive pages in physical memory
    - Each page retrieved with a separate **`alloc_page`** call
    - Less efficient
  - Can sleep (cannot be used in atomic context)

vmalloc and Friends

- **vmalloc**-related prototypes

```
#include <linux/vmalloc.h>

void *vmalloc(unsigned long size);
void  vfree(void * addr);
void *ioremap(unsigned long offset, unsigned long size);
void  iounmap(void * addr);
```

vmalloc and Friends

- **`ioremap`** builds page tables
  - **Does not allocate memory**
  - Takes a physical address (**`offset`**) and return a virtual address
    - **Useful to map the address of a Hardware buffer to kernel space**
  - Should use **`readb`** and other relation functions to access remapped memory