```c
struct scatterlist {
#ifdef CONFIG_DEBUG_SG
	unsigned long	sg_magic;
#endif
	unsigned long	page_link;
	unsigned int	offset;
	unsigned int	length;
	dma_addr_t	dma_address;
#ifdef CONFIG_NEED_SG_DMA_LENGTH
	unsigned int	dma_length;
#endif
};
```

in include/linux/scatterlist.h

```c
static inline void sg_set_buf(struct scatterlist *sg, const void *buf,
			      unsigned int buflen)
{
#ifdef CONFIG_DEBUG_SG
	BUG_ON(!virt_addr_valid(buf));
#endif
	sg_set_page(sg, virt_to_page(buf), buflen, offset_in_page(buf));
}
```

sg_set_buf()可以看出struct scatterlist fields的意义。

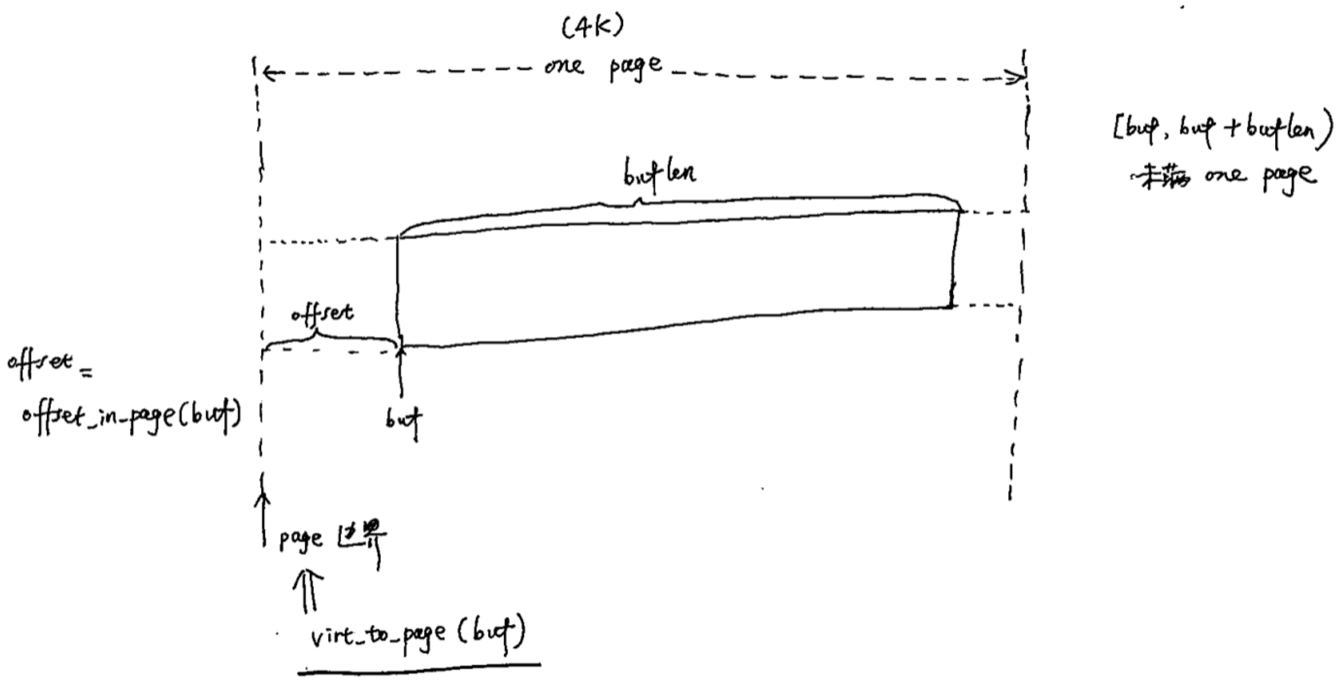
```c
static inline void sg_set_page(struct scatterlist *sg, struct page *page,
			       unsigned int len, unsigned int offset)
{
	sg_assign_page(sg, page);
	sg->offset = offset;
	sg->length = len;
}
```

unsigned long  page_link;

指向该scatterlist所管理的struct page。

该page中的空间可能只有部分内容是属于该scatterlist管理的，所以有

```c
	unsigned int	offset;
	unsigned int	length;
```



```
#define offset_in_page(p)       ((unsigned long)(p) & ~PAGE_MASK)
```

---

```
        unsigned long   page_link;
```

page_link的最底2 bits被用于额外用途。

```
static inline struct page *sg_page(struct scatterlist *sg)
{
#ifdef CONFIG_DEBUG_SG
        BUG_ON(sg->sg_magic != SG_MAGIC);
        BUG_ON(sg_is_chain(sg));
#endif
        return (struct page *)((sg)->page_link & ~0x3);
}
```

```
#define sg_chain_ptr(sg)        \
        ((struct scatterlist *) ((sg)->page_link & ~0x03))
```

真正指向struct page的address是page_link的高30 bits.

---

```c
#define sg_is_chain(sg)         ((sg)->page_link & 0x01)
#define sg_is_last(sg)          ((sg)->page_link & 0x02)
```

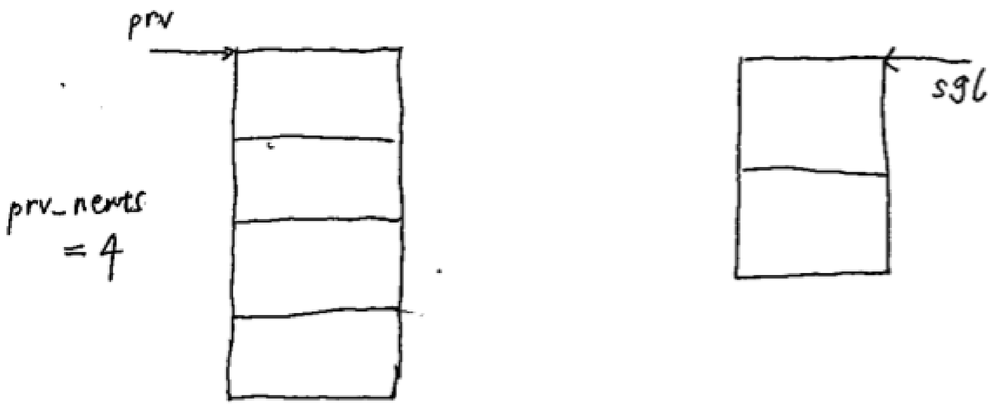
bit 0 --- chain flag

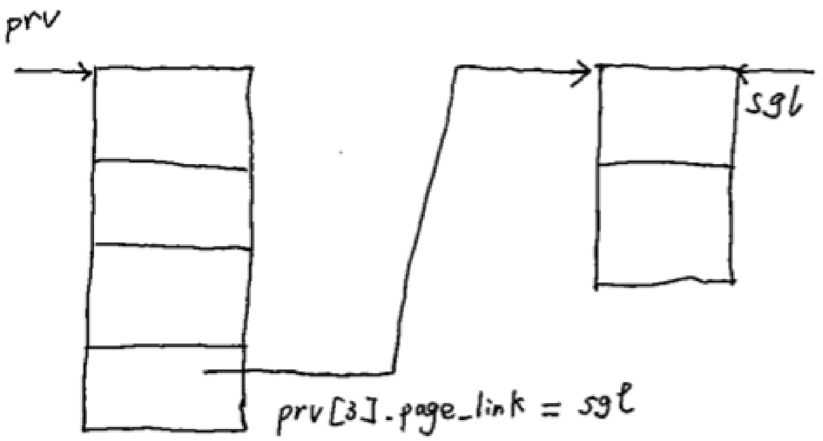
bit 1 --- last scatter list flag

```c
/**
 * sg_chain - Chain two sglists together
 * @prv:        First scatterlist
 * @prv_nents:  Number of entries in prv
 * @sgl:        Second scatterlist
 *
 * Description:
 *   Links @prv@ and @sgl@ together, to form a longer scatterlist.
 *
 **/
static inline void sg_chain(struct scatterlist *prv, unsigned int prv_nents,
                            struct scatterlist *sgl)
{
#ifndef CONFIG_ARCH_HAS_SG_CHAIN
        BUG();
#endif

        /*
         * offset and length are unused for chain entry.  Clear them.
         */
        prv[prv_nents - 1].offset = 0;
        prv[prv_nents - 1].length = 0;

        /*
         * Set lowest bit to indicate a link pointer, and make sure to clear
         * the termination bit if it happens to be set.
         */
        prv[prv_nents - 1].page_link = ((unsigned long) sgl | 0x01) & ~0x02;
}
```

What is chain in scatter list ?

sg_chain()演示了chain的概念。





prv[3].page_link = sgl

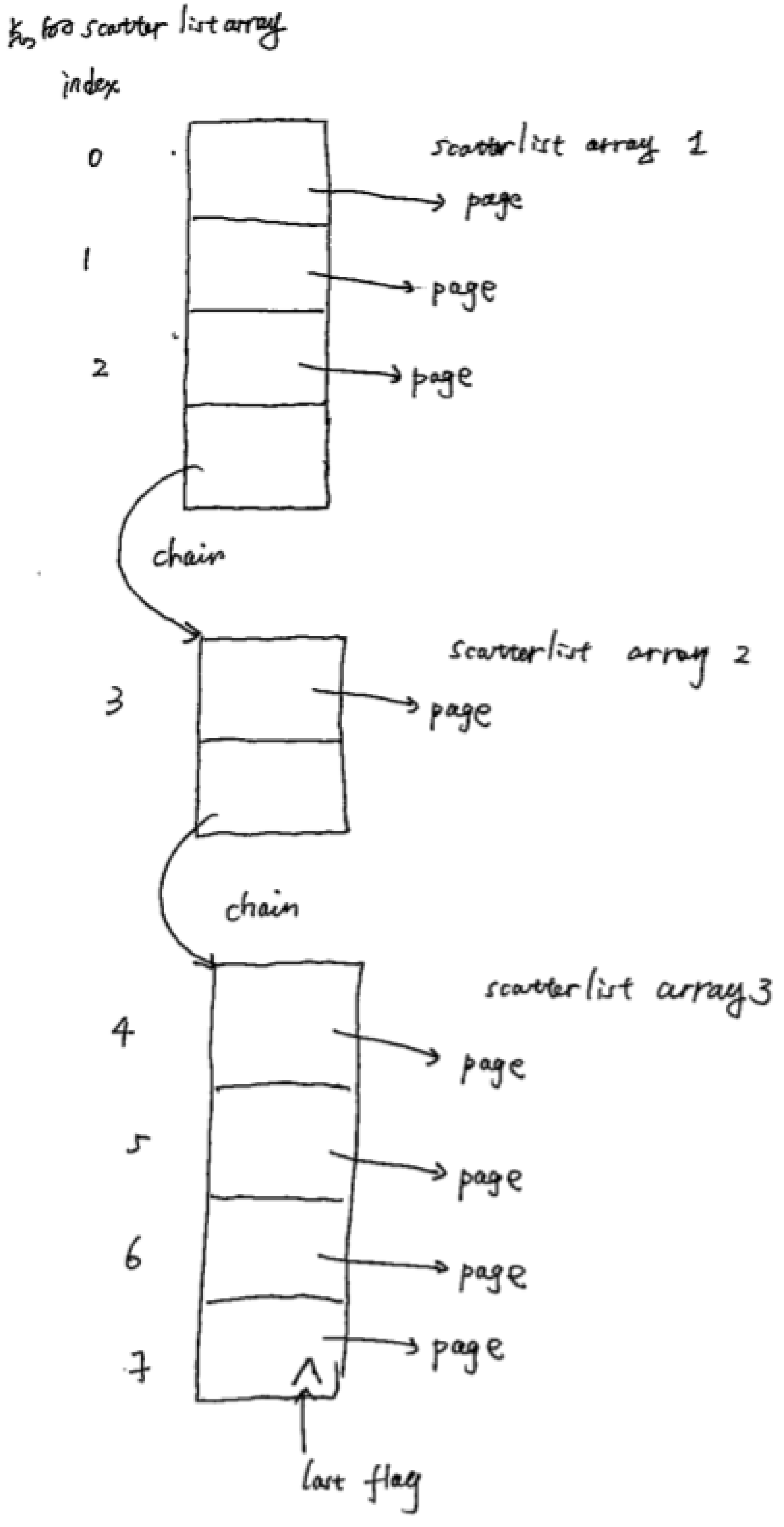
prv[3].page_link.chain flag = 1

$\left\{\begin{array}{l}\text{prv[3]. offset} = 0 \\ \text{prv[3]. length} = 0\end{array}\right.$ ，当 chain flag 置位，该 fields 无意义！

```c
/**
 * sg_mark_end - Mark the end of the scatterlist
 * @sg:          SG entryScatterlist
 *
 * Description:
 *   Marks the passed in sg entry as the termination point for the sg
 *   table. A call to sg_next() on this entry will return NULL.
 *
 **/
static inline void sg_mark_end(struct scatterlist *sg)
{
#ifdef CONFIG_DEBUG_SG
        BUG_ON(sg->sg_magic != SG_MAGIC);
#endif
        /*
         * Set termination bit, clear potential chain bit
         */
        sg->page_link |= 0x02;
        sg->page_link &= ~0x01;
```

设置last scatter list flag.

这样多个scatter可以串联起来，象下图所示。



sg-next()就是enumerate该实际上由3段
scatterlist组成的"array".

```c
/**
 * sg_next - return the next scatterlist entry in a list
 * @sg:         The current sg entry
 *
 * Description:
 *   Usually the next entry will be @sg@ + 1, but if this sg element is part
 *   of a chained scatterlist, it could jump to the start of a new
 *   scatterlist array.
 *
 **/
struct scatterlist *sg_next(struct scatterlist *sg)
{
#ifdef CONFIG_DEBUG_SG
        BUG_ON(sg->sg_magic != SG_MAGIC);
#endif
        if (sg_is_last(sg))
                return NULL;

        sg++;
        if (unlikely(sg_is_chain(sg)))
                sg = sg_chain_ptr(sg);

        return sg;
}
```