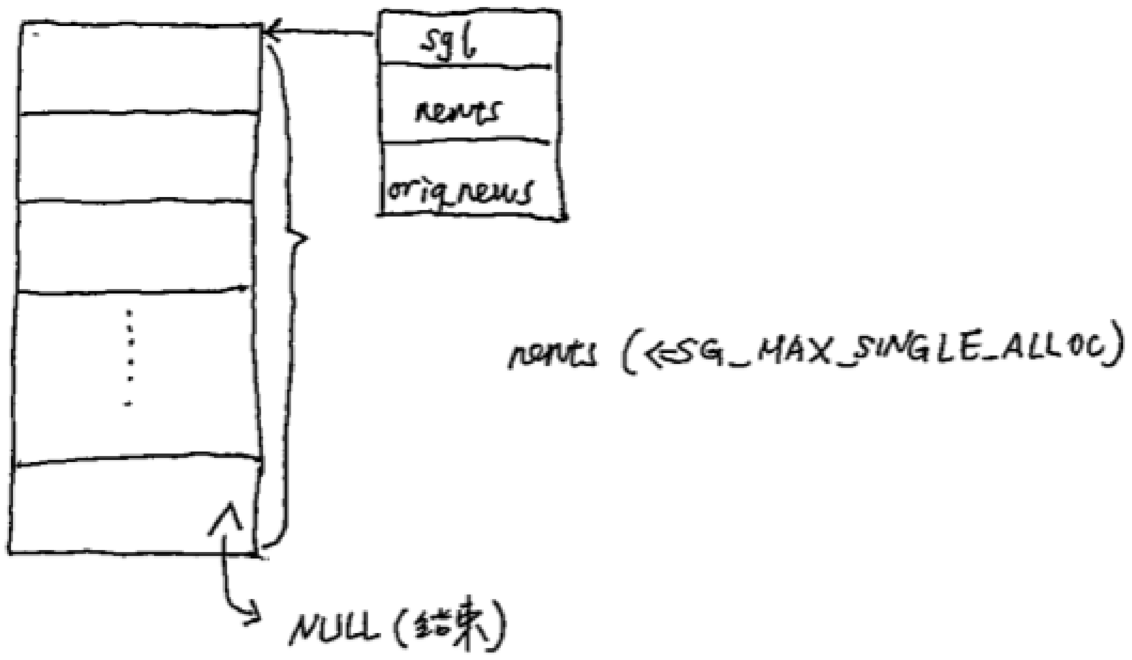
```c
/**
 * sg_alloc_table - Allocate and initialize an sg table
 * @table:      The sg table header to use
 * @nents:      Number of entries in sg list
 * @gfp_mask:   GFP allocation mask
 *
 *  Description:
 *    Allocate and initialize an sg table. If @nents@ is larger than
 *    SG_MAX_SINGLE_ALLOC a chained sg table will be setup.
 *
 **/
int sg_alloc_table(struct sg_table *table, unsigned int nents, gfp_t gfp_mask)
{
        int ret;

        ret = __sg_alloc_table(table, nents, SG_MAX_SINGLE_ALLOC,
                               NULL, gfp_mask, sg_kmalloc);
        if (unlikely(ret))
                __sg_free_table(table, SG_MAX_SINGLE_ALLOC, false, sg_kfree);

        return ret;
}
```

allocate nents scatterlist entry.

问题是这里的nents的大小！

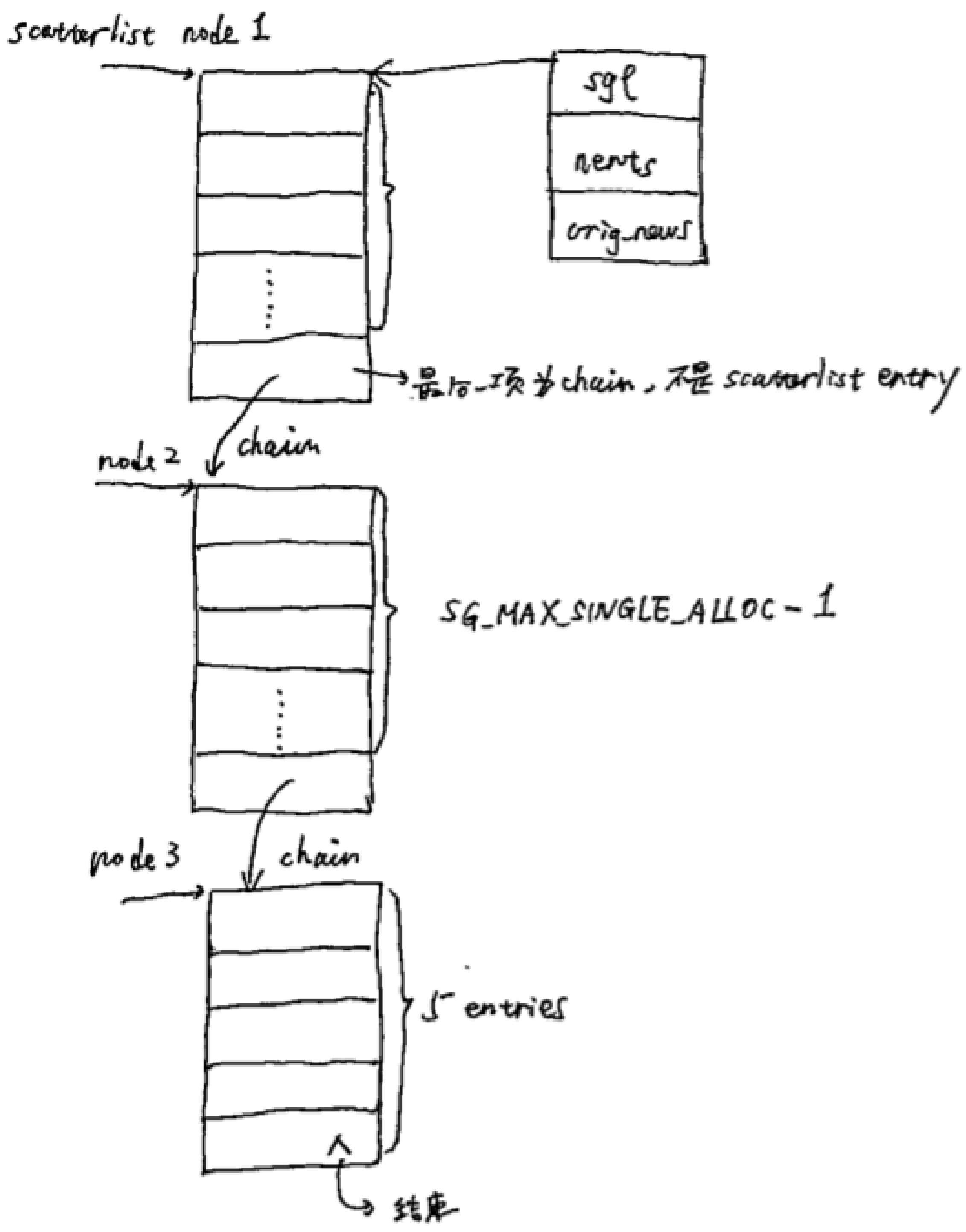
if nents <= SG_MAX_SINGLE_ALLOC

    allocate scatterlist array



else if nents > SG_MAX_SINGLE_ALLOC

比如nents = SG_MAX_SINGLE_ALLOC * 2 + 3



#define SG_MAX_SINGLE_ALLOC   (PAGE_SIZE / sizeof(struct scatterlist))

one page可以包含的scatterlist entries

__sg_alloc_table()实现了上面的algorithm

```
int __sg_alloc_table(struct sg_table *table, unsigned int nents,
                     unsigned int max_ents, struct scatterlist *first_chunk,
                     gfp_t gfp_mask, sg_alloc_fn *alloc_fn)
{
        struct scatterlist *sg, *prv;
        unsigned int left;

        memset(table, 0, sizeof(*table));

        if (nents == 0)
                return -EINVAL;
#ifndef CONFIG_ARCH_HAS_SG_CHAIN
        if (WARN_ON_ONCE(nents > max_ents))
                return -EINVAL;
#endif

        left = nents;
        prv = NULL;
        do {
                unsigned int sg_size, alloc_size = left;

                if (alloc_size > max_ents) {
                        alloc_size = max_ents;              ①
                        sg_size = alloc_size - 1;             ②
                } else
                        sg_size = alloc_size;

                left -= sg_size;                            ③

                if (first_chunk) {
                        sg = first_chunk;
                        first_chunk = NULL;
                } else {
                        sg = alloc_fn(alloc_size, gfp_mask);
                }
                if (unlikely(!sg)) {
                        /*
                         * Adjust entry count to reflect that the last
                         * entry of the previous table won't be used for
                         * linkage.  Without this, sg_kfree() may get
                         * confused.
                         */
                        if (prv)
                                table->nents = ++table->orig_nents;

                        return -ENOMEM;
                }

                sg_init_table(sg, alloc_size);
                table->nents = table->orig_nents += sg_size;

                /*
                 * If this is the first mapping, assign the sg table header.
                 * If this is not the first mapping, chain previous part.
                 */
                if (prv)
                        sg_chain(prv, max_ents, sg);
                else
                        table->sgl = sg;

                /*
                 * If no more entries after this one, mark the end
                 */
                if (!left)
                        sg_mark_end(&sg[sg_size - 1]);

                prv = sg;
        } while (left);

        return 0;
}
```

①

allocate max_ents entries

②

max_ents entries的最后entry，即sg[max_ents - 1]是作为chain使用的，所以 - 1

③

left -= sg_size;

在alloc_size > max_ents情况下等于

left -= alloc_size - 1

还是因为最后一项是用作chain的。