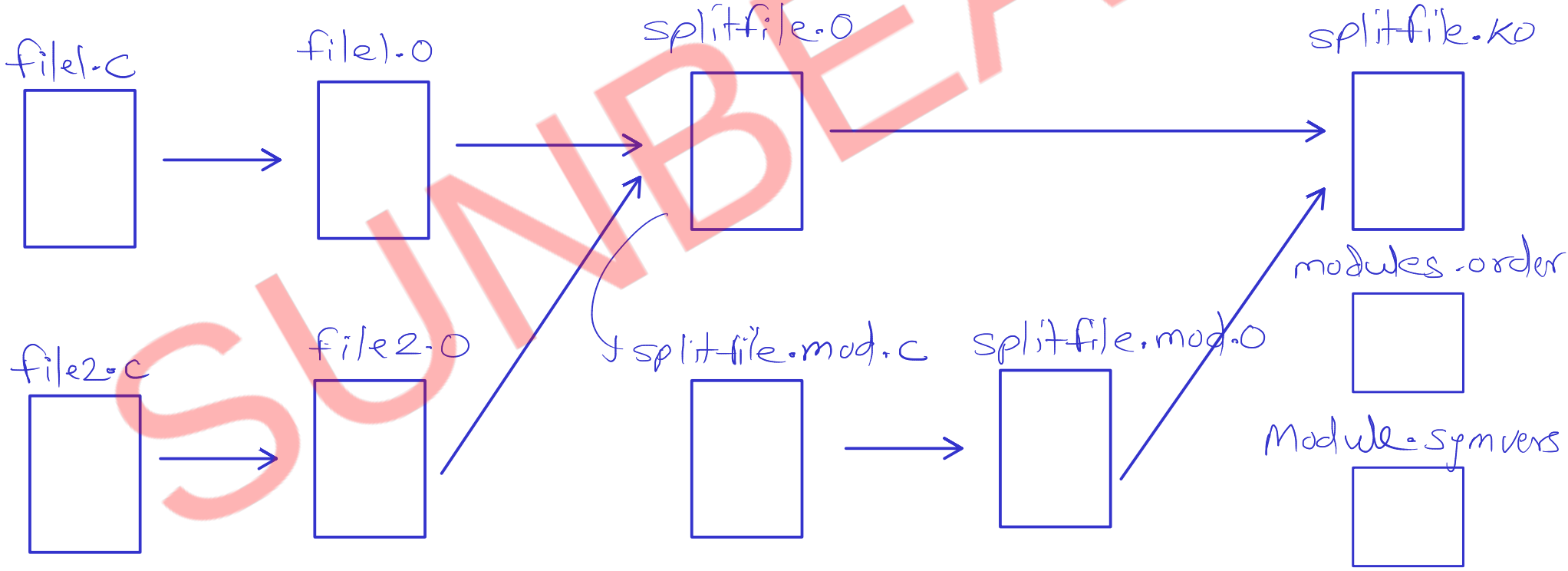
Compiling multi-file modules

- To make kernel modules code maintainable,
it is common practice to divide the code into multiple source files.
- To compile such files, Makefile should be updated.
- Makefile:

```
multifile-objs = file1.o file2.o
obj-m = multifile.o
```

- Compilation flow:



Kernel module parameters

- Module parameters are used to give configuration options while loading the module.
- Some of these configurations may be modified via /sys/module entry.
- The module params are declared as static global variables in the module.

- Supported data types are:
  bool, invbool, charp, int, short, long, uint, ushort, ulong

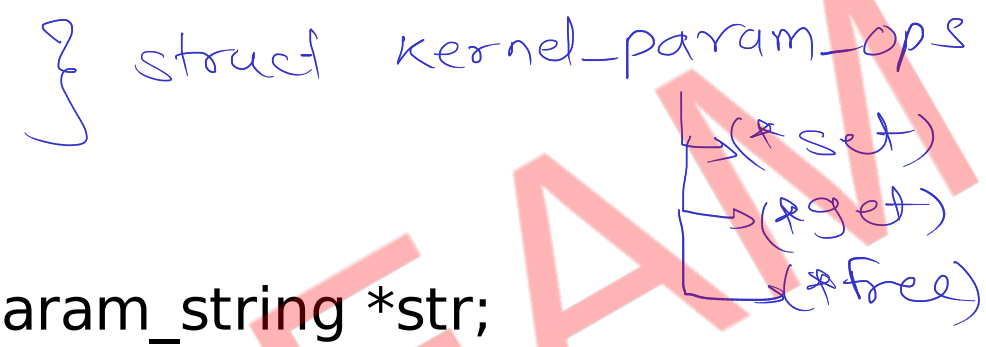
- Module param syntax
  - module_param(varname, datatype, permissions);
- Module param array syntax
  - module_param_array(varname, datatype, ele_count, permissions);
- Module param can be exposed with different name
  module_param_named(user_name, varname, datatype, permissions);

- Module param permissions can be given using macros:
  S_IRUSR, S_IWUSR, S_IXUSR,
  S_IRGRP, S_IWGRP, S_IXGRP,
  S_IROTH, S_IWOTH, S_IXOTH.

Kernel module parameters internals

- Internally module parameters are stored in a struct kernel_param.
  It has following members:

```
const char *name;
param_set_fn set;
param_get_fn get;
union {
    void *arg;
    const struct kparam_string *str;
    const struct kparam_array *arr;
};
```



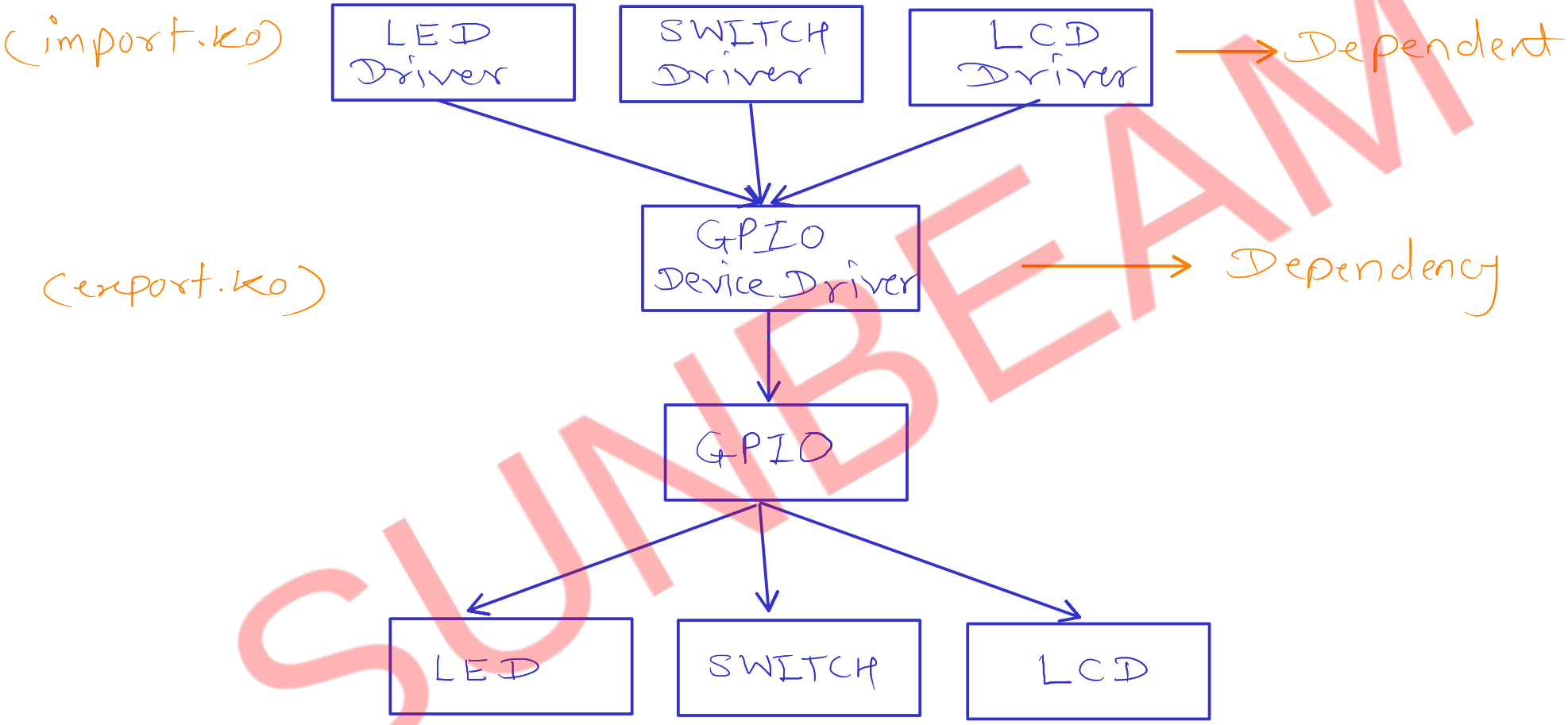
- The kernel_param struct variables are added into "__params" section.
- The param name and type information is added into ".modinfo" section.

- The param names & params are exported from module and become part of kernel symbol table.

- Module param value can be accessed or modified using /sys/module/mod_name/parameters/param_name.

Module stacking

One module is calling functions/variables from another module.



Module stacking

- A kernel module can export symbols using macros:
    - EXPORT_SYMBOL(symbol)
        - it exports the given symbol, so that it can be used by any other kernel module.
    - EXPORT_SYMBOL_GPL(symbol)
        - Similar to EXPORT_SYMBOL() for exporting symbols with "_gpl" tag.
        - These symbols can only be used by GPL licensed modules.

- Compiling multiple modules using Makefile.

```
obj-m := export.o import.o
```

- Loading & unloading kernel modules:
    - terminal> sudo insmod export.ko
    - terminal> sudo insmod import.ko
    - terminal> sudo rmmod import.ko
    - terminal> sudo rmmod export.ko

Module stacking internals

EXPORT_SYMBOL(symbol):
* Creates a name of symbol as "static const char[]" and store it in special section "__ksymtab_strings".

* Also create a struct "kernel_symbol" variable that stores address of the variable and its name in another special section "__ksymtab".

* If Linux kernel version control is enabled, CRC is also stored in special section "__kcrctab". e.g. if a variable by name "get_rms" is exported using "EXPORT_SYMBOL", then this macro internally produces following code:

* static const char __kstrtab_get_rms[]
__attribute__((section("__ksymtab_strings"))) = "get_rms";

* static const struct kernel_symbol __ksymtab_get_rms __attribute_used__
__attribute__((section("__ksymtab" ""), unused)) = { (unsigned long)&get_rms,
__kstrtab_get_rms };

* During loading a module these special sections (__ksymtab) are read from .ko file and information in that is added into kernel symbol table.