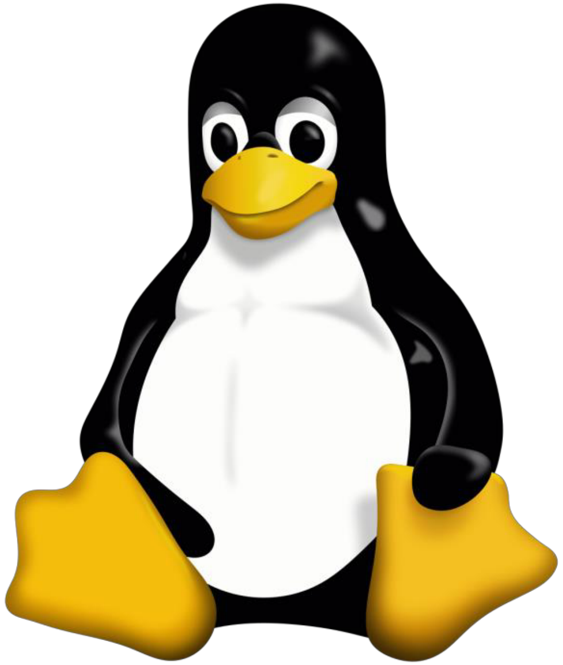
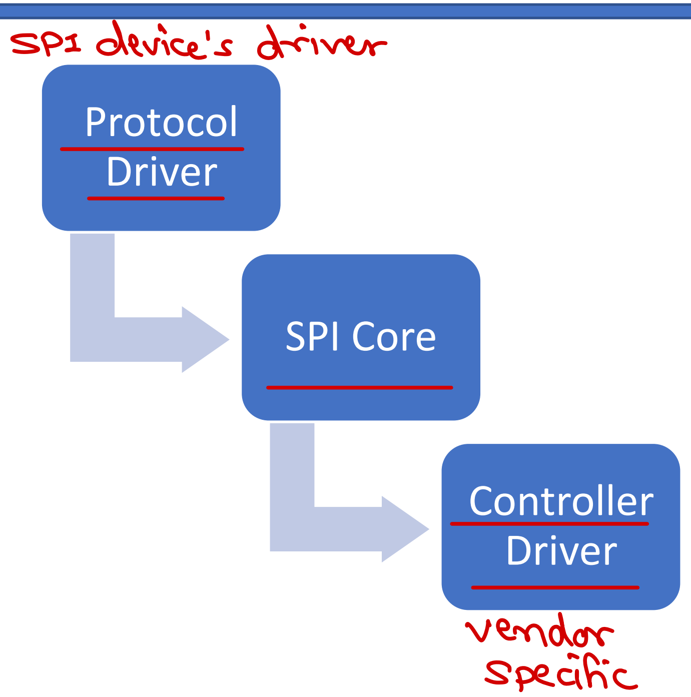


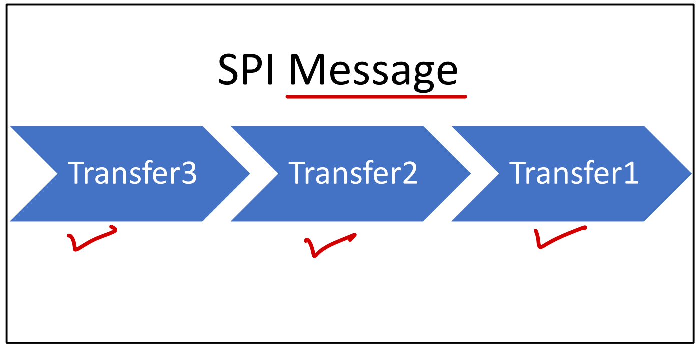
Linux Character Device Driver

Sunbeam Infotech

Linux SPI Sub-system

- SPI sub-system has 3 parts
  - SPI core – provides core data structures, registration, cancellation and unified interface for SPI drivers. It is platform independent. (kernel/drivers/spi/spi.c).
  - SPI controller driver – low-level (hardware register level) platform specific driver usually implemented by vendor. Loaded while system booting & provides appropriate read(), write().
  - SPI protocol driver – handle/interact with SPI device. The interaction is in terms of messages and transfers.
- SPI Transfers and Messages
  - Transfer – defines a single operation between master and slave. Use tx/rx buffer pointers and optional delay/chip select behaviour after op.
  - Message – atomic sequence of transfer. Argument to all SPI read/write functions.





SPI device driver

newer kernel → struct spi_master
(5.x)

SPI mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

- Get the SPI Controller driver.
  - *struct spi_controller * spi_busnum_to_master(u16 bus_num);*
- Add the slave device to the SPI Controller.
  - *struct spi_board_info my_dev_info = { .modalias = "my_spi_driver", .max_speed_hz = 4000000, .bus_num = 1, .chip_select = 0, .mode = SPI_MODE_0 };*
  - *struct spi_device * spi_new_device(struct spi_controller *ctlr, struct spi_board_info *chip); - spi_alloc_device() + spi_add_device();*
- Configure the SPI
  - *int spi_setup(struct spi_device *spi); // call after any change in spi_device.*
- Transfer the data between master and slave.
  - *int spi_sync_transfer(struct spi_device *spi, struct spi_transfer *xfers, unsigned int num_xfers);*
  - *int spi_async(struct spi_device *spi, struct spi_message *message);*
  - *int spi_write_then_read(struct spi_device * spi, const void * txbuf, unsigned n_tx, void * rxbuf, unsigned n_rx);*
- At the end remove the device & driver.
  - *void spi_unregister_device(struct spi_device *spi);*

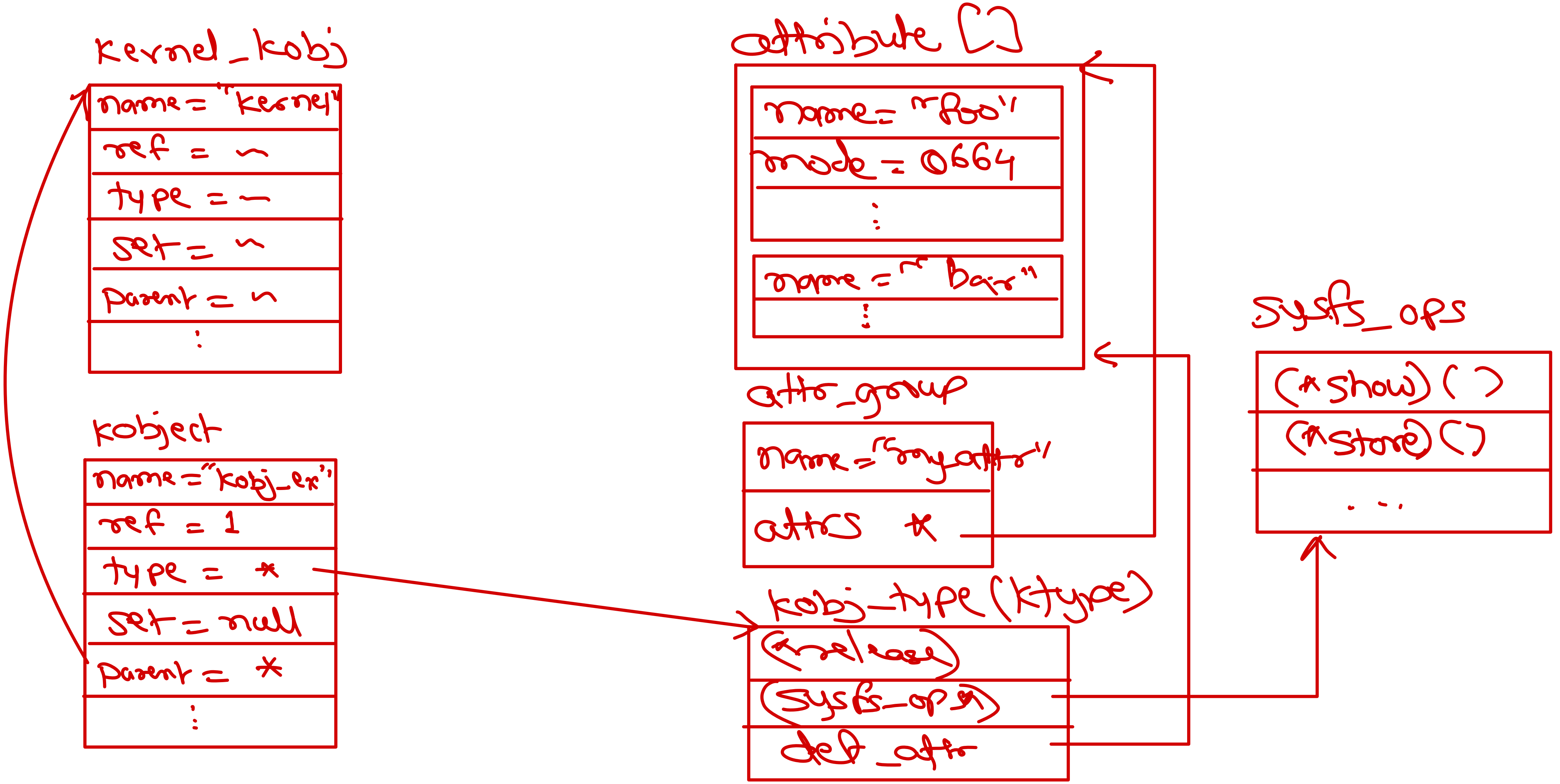
```
struct spi_board_info {
  char modalias[SPI_NAME_SIZE];
  const void      *platform_data;
  const struct property_entry *properties;
  void *controller_data;
  int irq;
  u32 max_speed_hz, mode;
  u16 bus_num, chip_select;
};
```

struct kobject

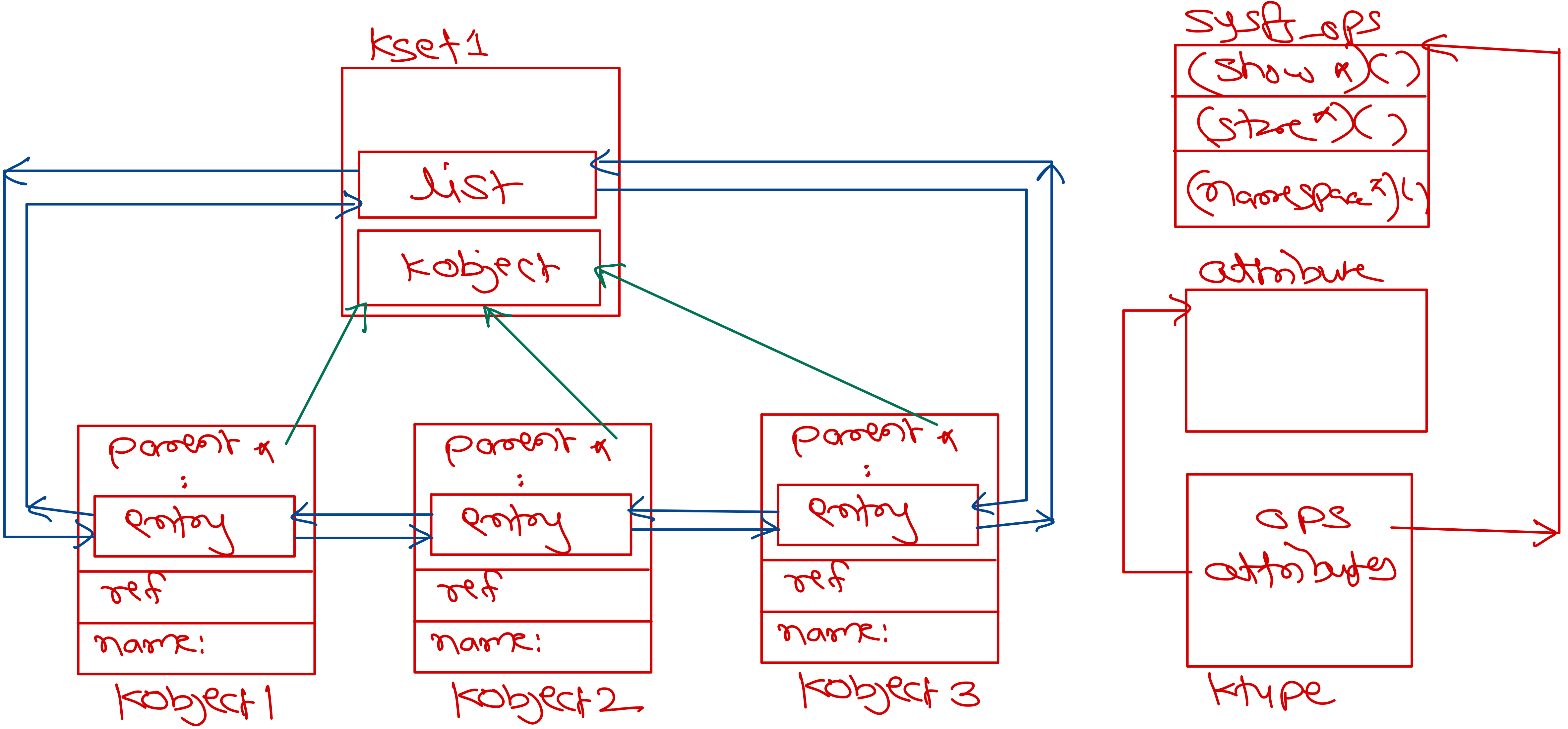
- Keeping track of various C struct objects is common need throughout the kernel.
- From Linux kernel 2.5 *struct kobject* is added for following functionalities.
- It provides following functionalities
    - Reference counting
    - Manage list of objects
    - Locking of sets
    - Exporting object properties to sysfs
- To avail these functionalities embed kobject into the desired struct.
- kobject functions: kobject_init(), kobject_get(), kobject_put(), kobject_add(), kobject_cleanup(), kobject_register(), kobject_unregister().

```
struct kobject {
   const char *k_name;
   struct kref kref;
   struct list_head entry;
   struct kobject *parent;
   struct kset *kset;
   struct kobj_type *ktype;
   struct sysfs_dirent *sd;
};
```

Example kobject



KSet


kset1
list
kobject
parent *
:
entry
ref
name:
Kobject1
parent *
:
entry
ref
name:
Kobject2
parent *
:
entry
ref
name:
Kobject 3
sysfs_ops
(show *)( )
(store *)( )
(namespace *)( )
attribute
ops
attributes
ktype

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>