Driver Model in U-Boot

Design goals, architecture, benefits, test methodology, limitations, future

U-Boot Mini-Summit 13-Oct-14
Simon Glass
sjg@chromium.org

Agenda

- Why driver model?
- Design goals
- Architecture
- Benefits
- Test methodology
- Limitations
- Next?

Why driver model?

- Device init and access is ad-hoc
  - scsi_init(), mmc_init(), nand_init()
- Many subsystems only allow one driver
  - But I have USB2 and USB3!
- Communication between subsystems is tricky
  - How does an I2C expander or PMIC provide a GPIO?
- Hard to answer simple questions
  - How many GPIOs? What is my serial console?
- Board file functions provide the glue
  - What GPIO provides my MMC card detect?

Design goals

- Replace ad-hoc code with a consistent framework
- Support U-Boot’s unique requirements
  - Numbered devices
  - Devices grouped by class
  - Relocation and SPL
- Small overhead (memory and CPU)
- Still allow lazy init of devices (no global probe)
- Built-in device tree support

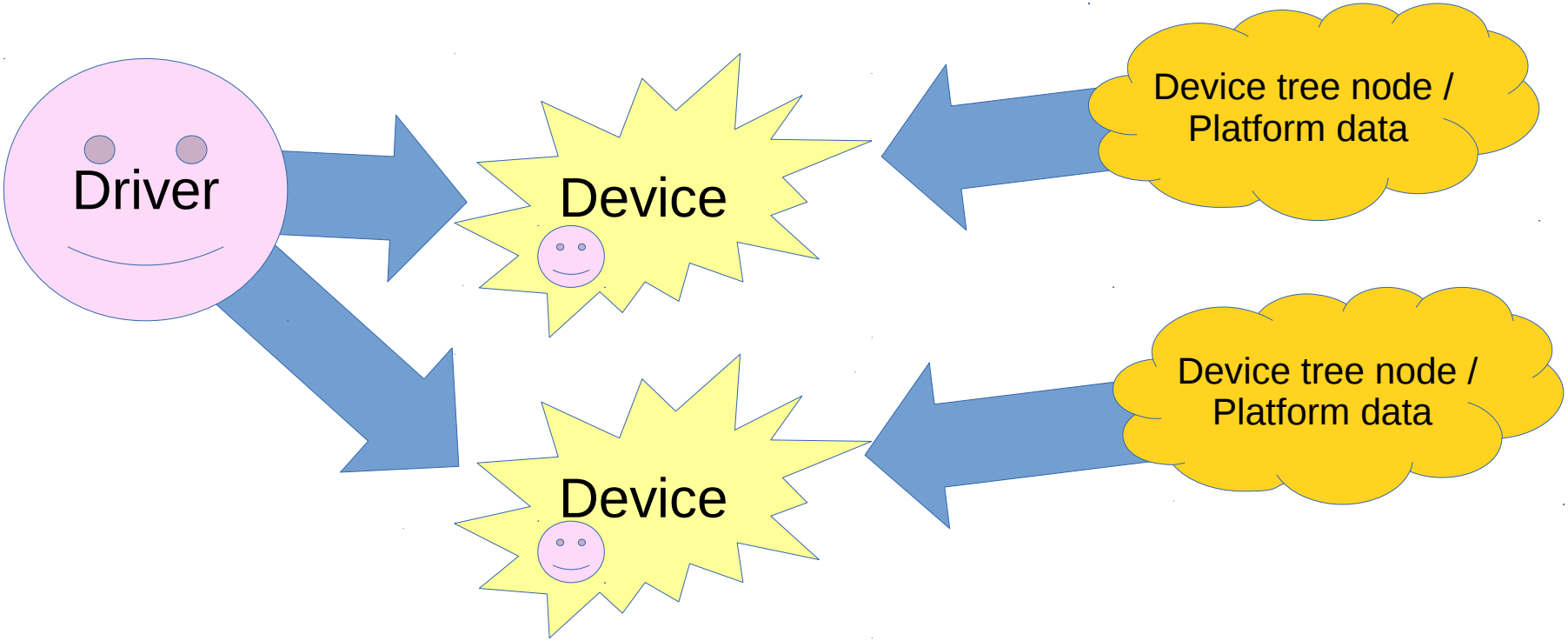
History

- An email in 2010
  - ”As for U-Boot, there currently is no driver model. Therefore I'd like to start a discussing on this topic so we can converge towards a reasonable result.“
- Started as University Project in 2012
  - Marek Vasut, Pavel Herrmann, Viktor Křivák, Tomas Hlavacek
- Initial RFC April 2013
- Mainline (v9 series) in 2014.04

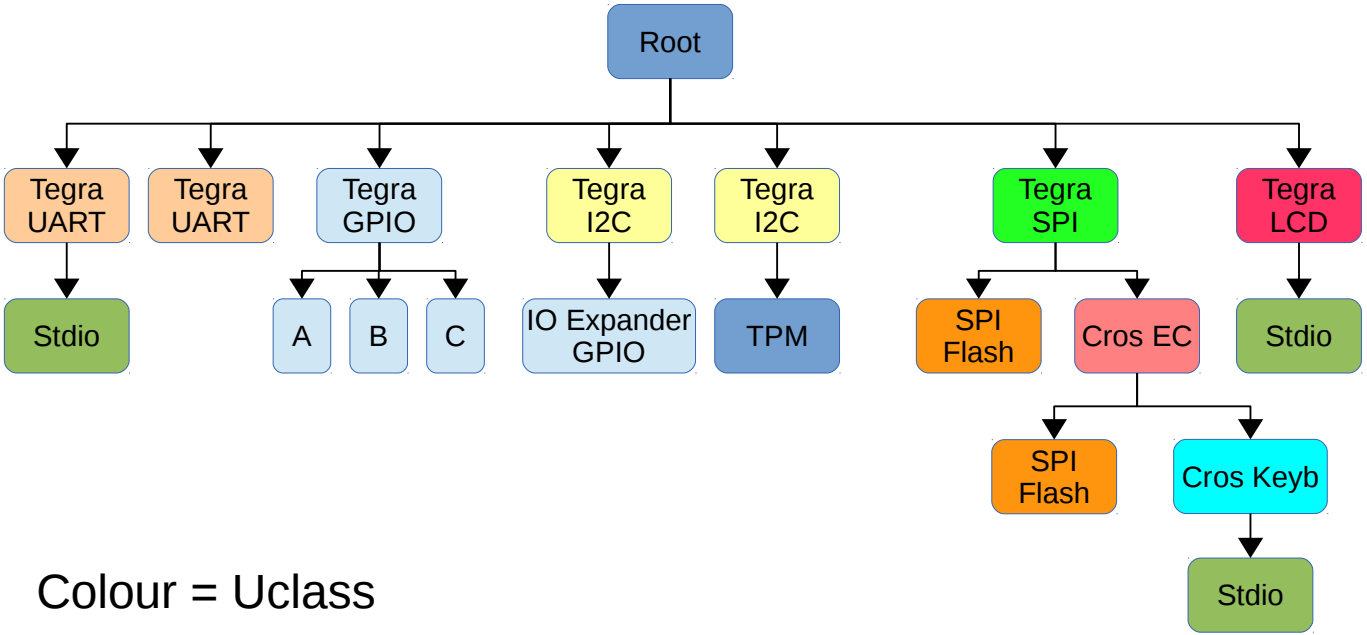
Concepts

- Uclass
- Udevice
- Platform data
- Device tree
- Hierarchy
- Bind
- Probe
- Sequence numbers

Driver → Devices

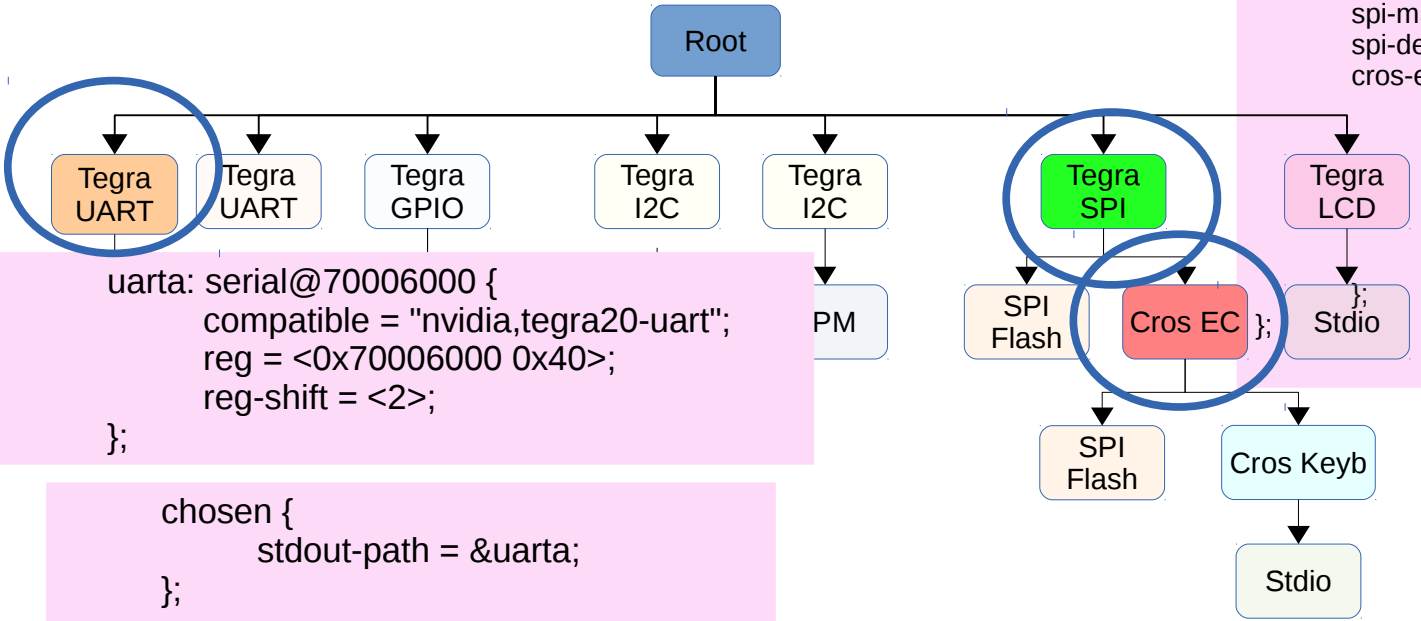


Architecture – Devices, Uclasses, Hierarchy



Colour = Uclass

Architecture – Binding with Device Tree




```
uarta: serial@70006000 {
        compatible = "nvidia,tegra20-uart";
        reg = <0x70006000 0x40>;
        reg-shift = <2>;
};
```

```
chosen {
        stdout-path = &uarta;
};
```

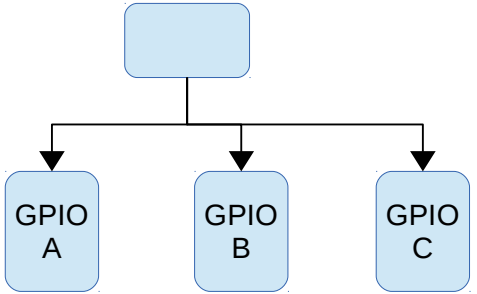
```
spi@131b0000 {
        spi-max-frequency = <1000000>;
        spi-deactivate-delay = <100>;
        cros-ec@0 {
                reg = <0>;
                compatible = "google,cros-ec";
                spi-max-frequency = <5000000>;
                ec-interrupt = <&gpio 182 1>;
                optimise-flash-write;
                status = "disabled";
        };
};
```

Architecture – Binding with Platform Data

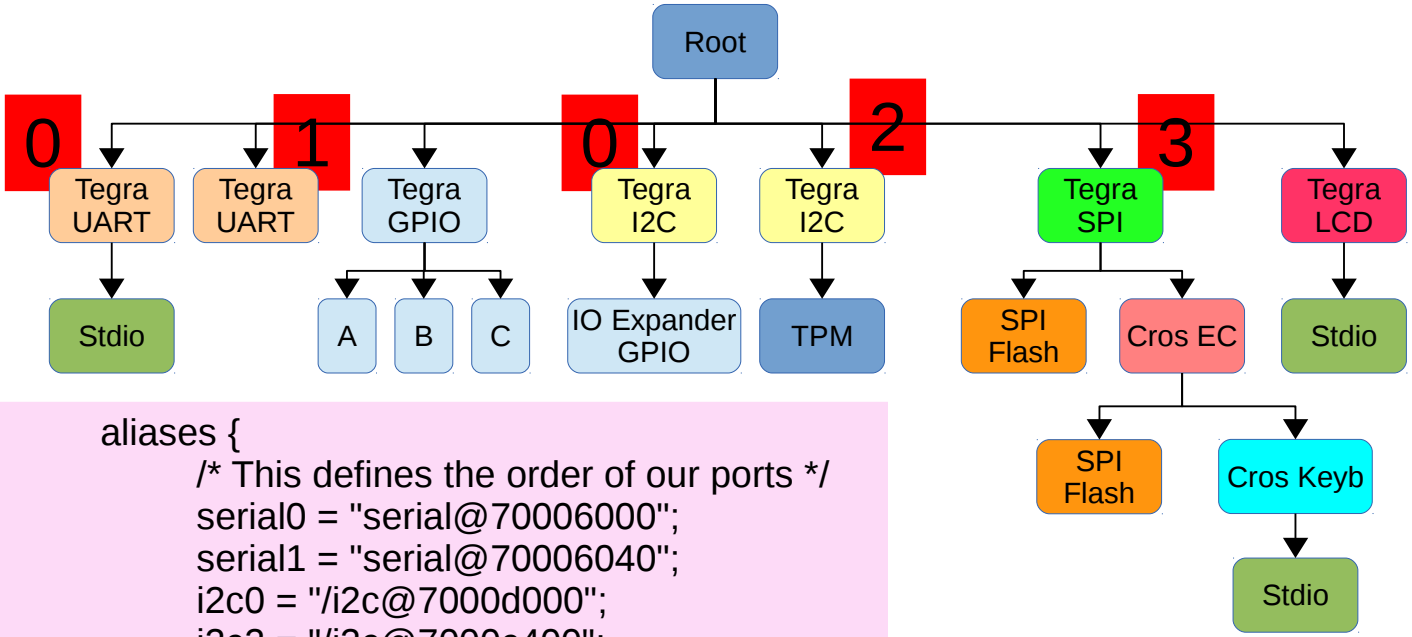
```
/* Platform data for each GPIO port */
struct at91_port_platdata {
        uint32_t base_addr;
        const char *bank_name;
};

/* Platform data for the GPIOs */
static const struct at91_port_platdata at91sam9260_plat[] = {
        { ATMEL_BASE_PIOA, "A" },
        { ATMEL_BASE_PIOB, "B" },
        { ATMEL_BASE_PIOC, "C" },
};

U_BOOT_DEVICES(at91sam9260_gpios) = {
        { "gpio_at91", &at91sam9260_plat[0] },
        { "gpio_at91", &at91sam9260_plat[1] },
        { "gpio_at91", &at91sam9260_plat[2] },
};
```

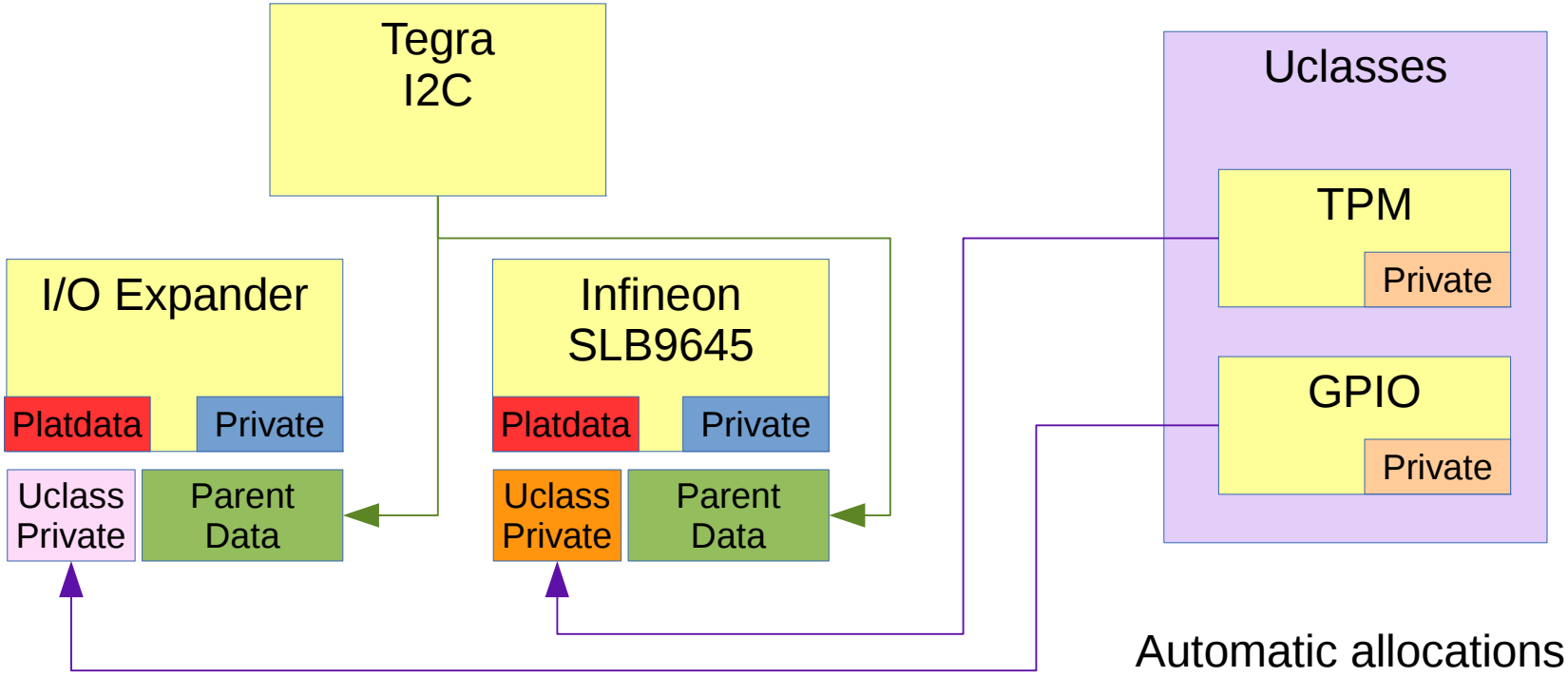


Architecture: Probe and Sequence Numbers




```
aliases {
        /* This defines the order of our ports */
        serial0 = "serial@70006000";
        serial1 = "serial@70006040";
        i2c0 = "/i2c@7000d000";
        i2c2 = "/i2c@7000c400";
};
```

Memory allocation



Benefits

- Consistent view of devices
- Automatic memory allocation
- Clear device lifecycle
- Automatic binding* and probing
- Good test coverage
- Easier device init

* SPL and pre-relocation bind a subset

Test methodology

- Automated tests using sandbox
- Each uclass has its own tests
- Script runs all tests
- Untested code does not work
  - May as well be deleted

```
Running 29 driver model tests
Test: dm_test_autobind
Test: dm_test_autoprobe
Test: dm_test_bus_children
Device 'd-test': seq 3 is in use by 'b-test'
Device 'c-test@0': seq 0 is in use by 'a-test'
Device 'c-test@1': seq 1 is in use by 'd-test'
Test: dm_test_bus_children_funcs
Test: dm_test_bus_children_iterators
Test: dm_test_bus_parent_data
...
```

Limitations

- It does have a learning curve
- Still some missing features
  - We find more as we add more drivers
- Pervasive code impact
- Not 100% compatible with Linux
  - Binding, formal classes, automatic allocation
- Hard to convert old boards (breakage!)
  - Leading to drivers which build both ways

Steps to convert a subsystem

- Kconfig - CONFIG_DM_<subsys>
- Uclass – split out ops functions, private data
- Sandbox driver – private data, device tree, platform data
- Automated test and how-to doc (with time estimate!)
- Port driver(s)
  - Can you change all users? => Remove old code
  - Need old bahaviour? =? #ifdef, don't duplicate common code

doc/driver-model/spi-howto.txt

Policy proposals for patches merged to 2015.01

- For new drivers and rewrites:
  - **Require** that they use DM
    - Will require enabling DM in boards that use the driver
  - **Request** porting existing boards
- For new boards
  - **Require** that they use DM when available for drivers they use
  - **Request** porting existing drivers to DM
- For new subsystems
  - **Require** that they use DM (generally)

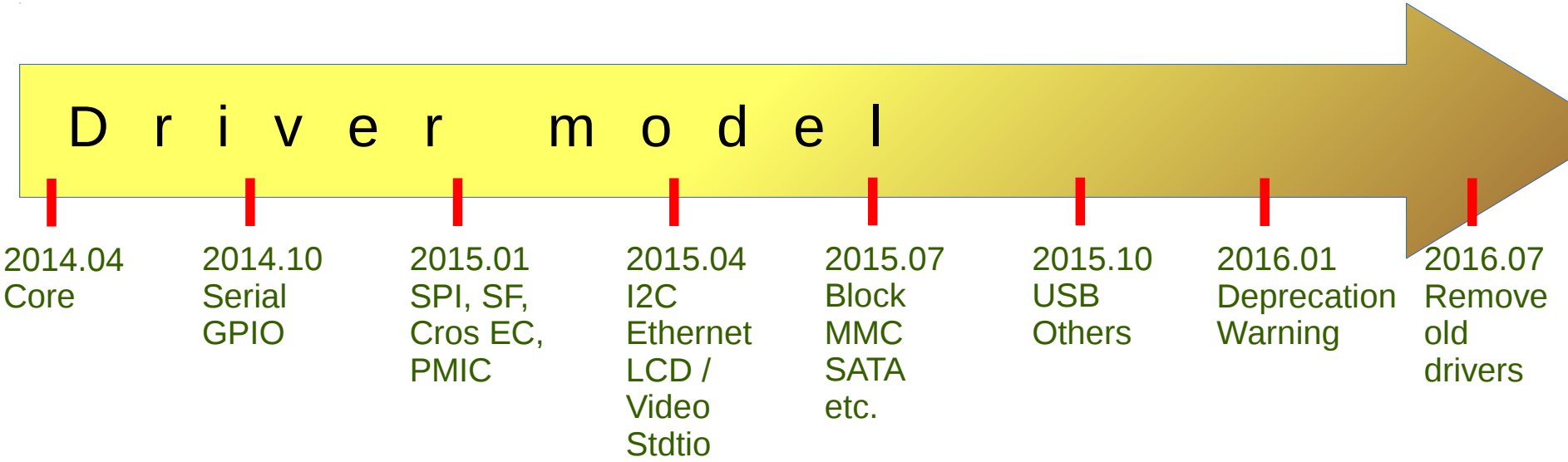
Current status

- Pre- and post-relocation support merged
- SPL support patches pending
- Series and GPIO driver conversion patches pending
  - Serial conversion 7 drivers
  - GPIO conversion 9 drivers
- SPI and SPI flash patches pending (and cros_ec)
  - Close to merging

Future work

- Merge outstanding patches (October, November 2014)
- PMIC framework (patches sent)
- Deal with I2C (Christmas present)
- Easy-ish: LCD and video, Ethernet, crypto, hash, compression, input, RTC, sound, watchdog, stdio, filesystems (call for volunteers!)
- Hard-ish: Block devices (MMC, USB, SATA, IDE) and USB (maintainers?)
  - NAND (already has its own Linux-based framework)
- Other interesting ideas
  - SOC clock framework

Proposed time line



Summary

- Driver model is available in U-Boot
  - Tree at u-boot-dm/working
- Efficient, small footprint
- Supports resource-constrained boot loader environment
- Goal to transition over next two years
- Some policy items TBD

Thank you