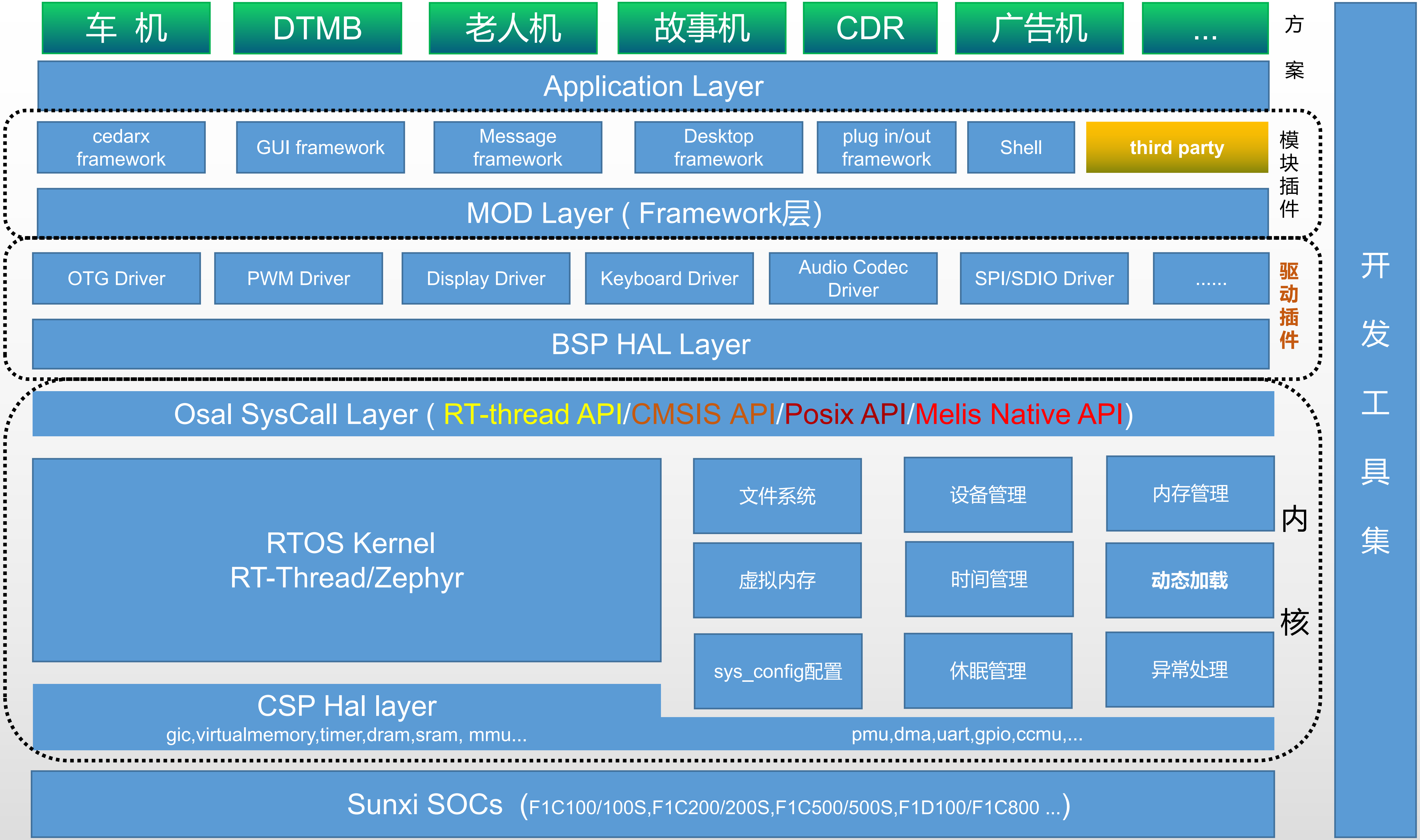


车 机

DTMB

老人机

故事机

CDR

广告机

...

方案

Application Layer

cedarx framework

GUI framework

Message framework

Desktop framework

plug in/out framework

Shell

third party

MOD Layer (Framework层)

模块插件

OTG Driver

PWM Driver

Display Driver

Keyboard Driver

Audio Codec Driver

SPI/SDIO Driver

......

BSP HAL Layer

驱动插件

Osal SysCall Layer (RT-thread API/CMSIS API/Posix API/Melis Native API)

RTOS Kernel
RT-Thread/Zephyr

文件系统

设备管理

内存管理

虚拟内存

时间管理

动态加载

sys_config配置

休眠管理

异常处理

CSP Hal layer

gic,virtualmemory,timer,dram,sram, mmu...

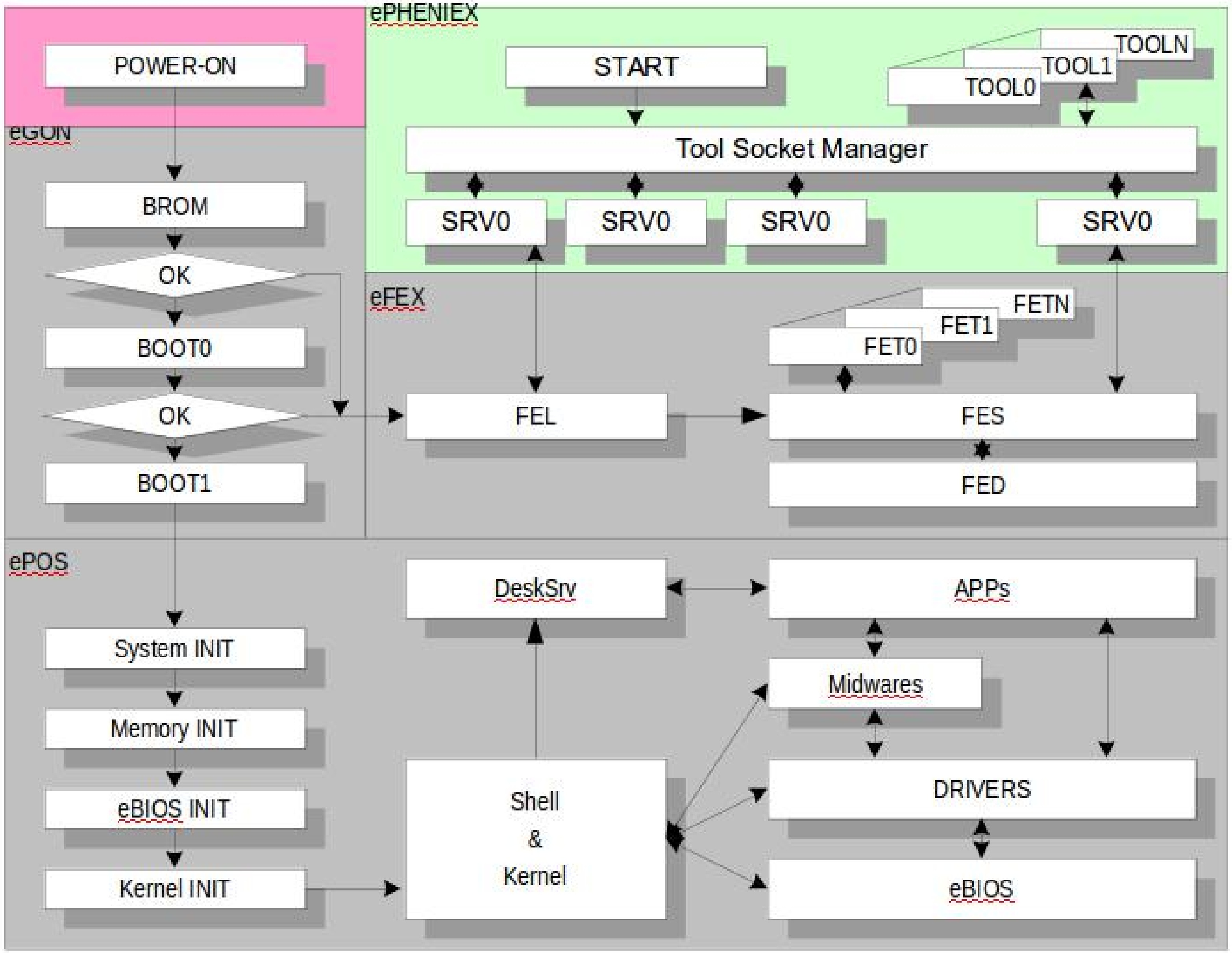
pmu,dma,uart,gpio,ccmu,...

内核

Sunxi SOCs (F1C100/100S,F1C200/200S,F1C500/500S,F1D100/F1C800 ...)

开发工具集

系统启动流程



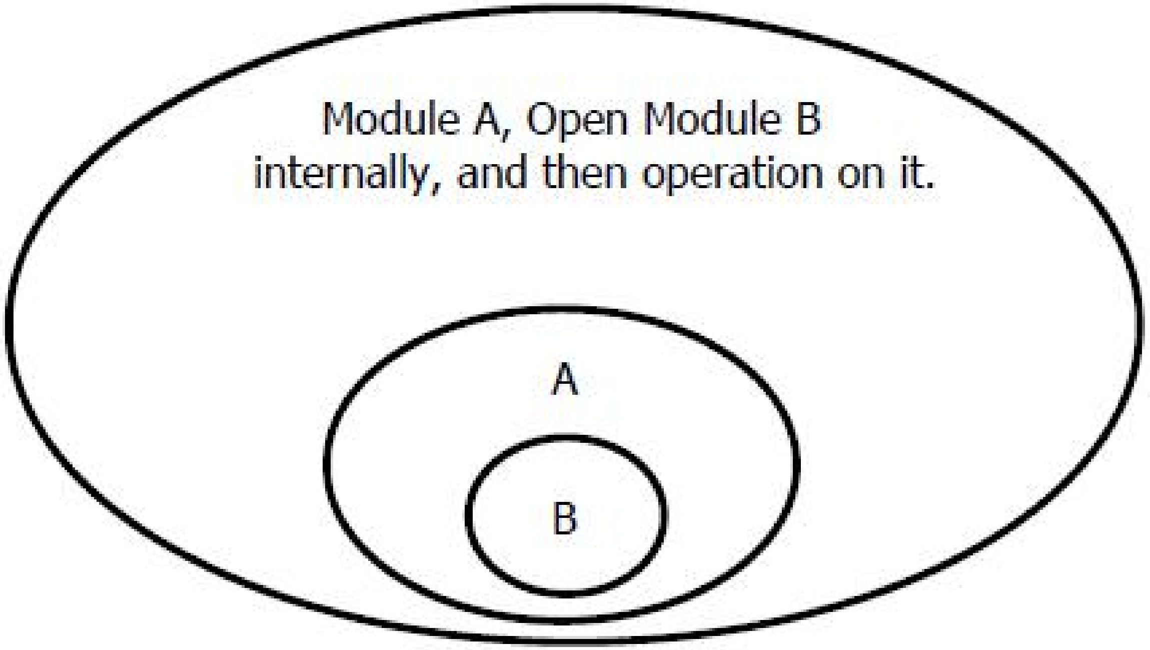
POWER-ON
eGON
BROM
OK
BOOT0
OK
BOOT1
ePHENIEX
START
TOOLN
TOOL1
TOOL0
Tool Socket Manager
SRV0
SRV0
SRV0
SRV0
eFEX
FETN
FET1
FET0
FEL
FES
FED
ePOS
System INIT
Memory INIT
eBIOS INIT
Kernel INIT
DeskSrv
APPs
Midwares
Shell
&
Kernel
DRIVERS
eBIOS

目前软件架构有哪些问题?

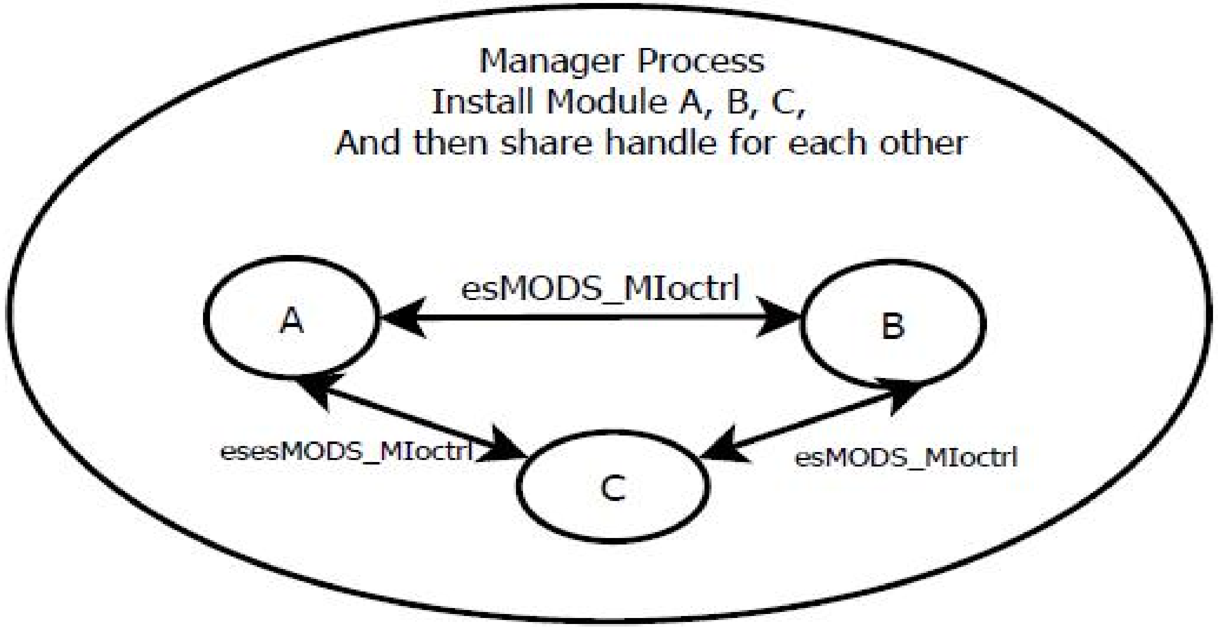
模块化提高了开发难度和降低了开发效率

- 除了内核，全部是模块。
  - 需要对业务逻辑进行封装，提供模块化标准接口
  - 需要提供链接脚本，查表寻找空闲内存区作为运行空间
  - 微内核工作模式，模块通信需要内核转发，跨模块通信复杂，
  - 需要考虑维护模块之间业务顺序，通常做法是业务层抽象独立的manager模块维护模块间的依赖关系和对工作流程进行控制。
  - 模块装载需要二级页表支持，系统复杂度增加。
  - 新客户将会有三成精力放在和产品功能无关的问题上。

新客户绝对不会接受这种开发方式！



Module A, Open Module B internally, and then operation on it.

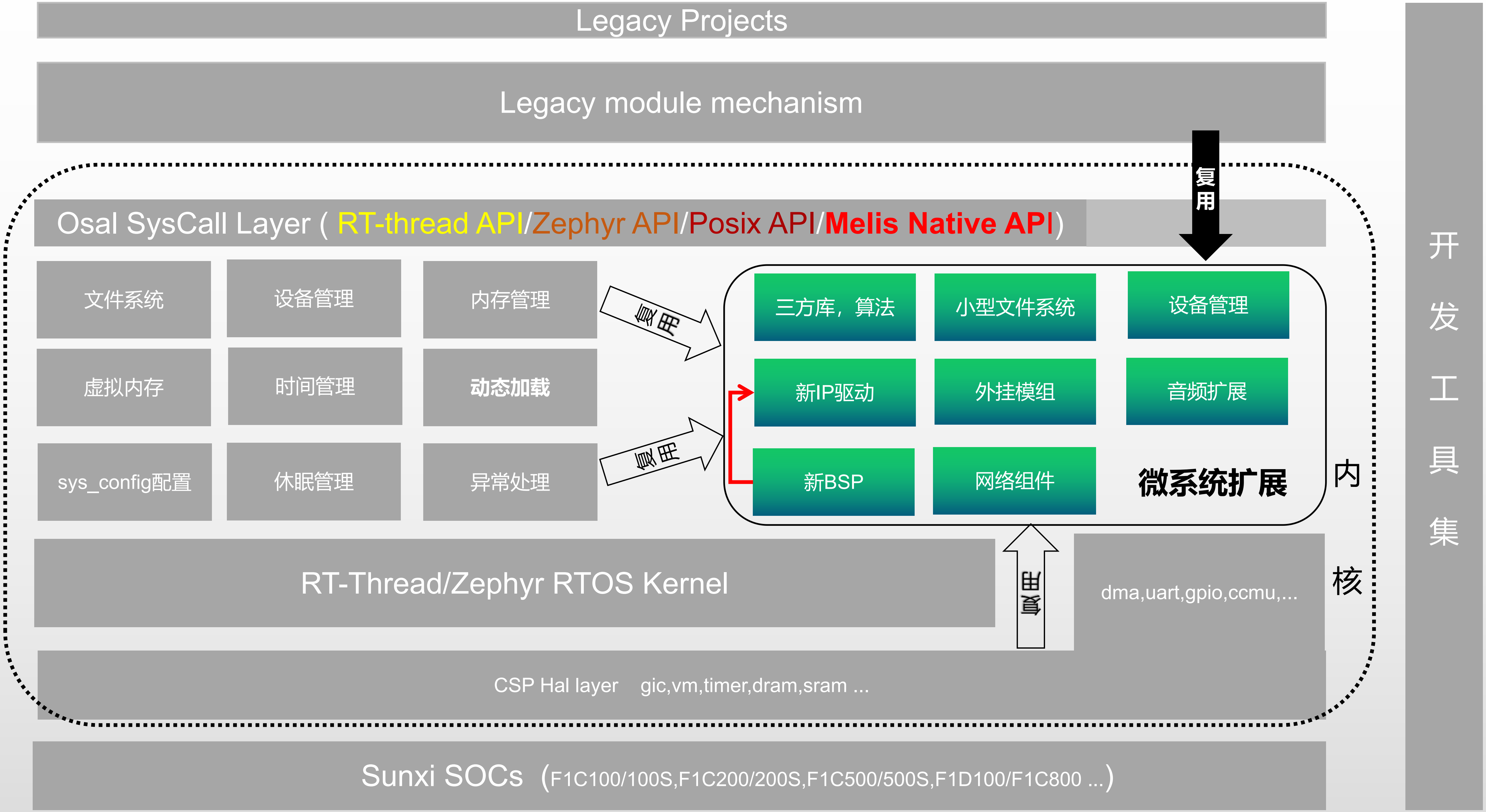


Manager Process
Install Module A, B, C,
And then share handle for each other

参考Linux-大内核小模块架构

- 抛弃微内核设计方式，采用“单一镜像内核” + “模块扩展”
- 采用unikernel架构，将方案做进内核.
- 新开发的BSP,Driver,中间件和应用方案放进内核，和内核链接到一起。
- 旧有的驱动模块和框架模块作为扩展特性，兼容进来。
- 新架构淡化模块机制，采用unikernel宏内核机制.

新架构-unikernel + legacy模块化扩展



Legacy Projects

Legacy module mechanism

复用

Osal SysCall Layer (RT-thread API/Zephyr API/Posix API/**Melis Native API**)

文件系统

设备管理

内存管理

虚拟内存

时间管理

动态加载

sys_config配置

休眠管理

异常处理

复用

复用

三方库，算法

小型文件系统

设备管理

新IP驱动

外挂模组

音频扩展

新BSP

网络组件

微系统扩展

内核

RT-Thread/Zephyr RTOS Kernel

复用

dma,uart,gpio,ccmu,...

CSP Hal layer   gic,vm,timer,dram,sram ...

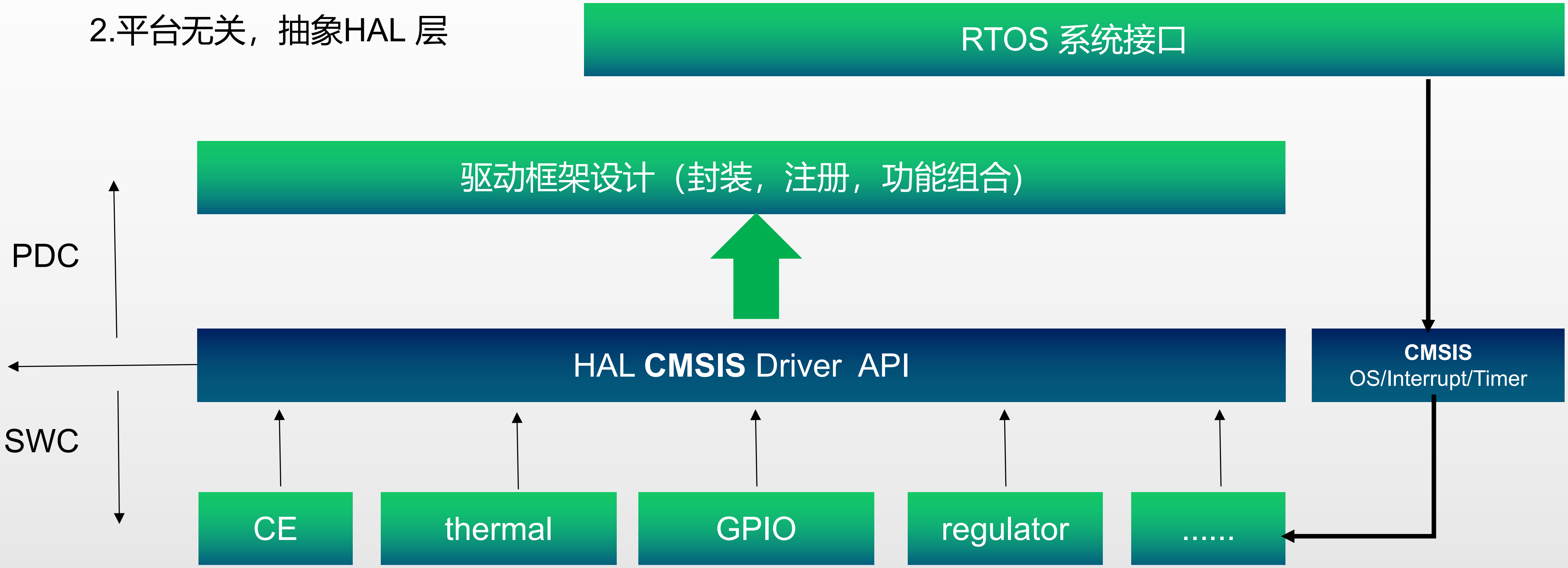
Sunxi SOCs （F1C100/100S,F1C200/200S,F1C500/500S,F1D100/F1C800 ...)

开发工具集

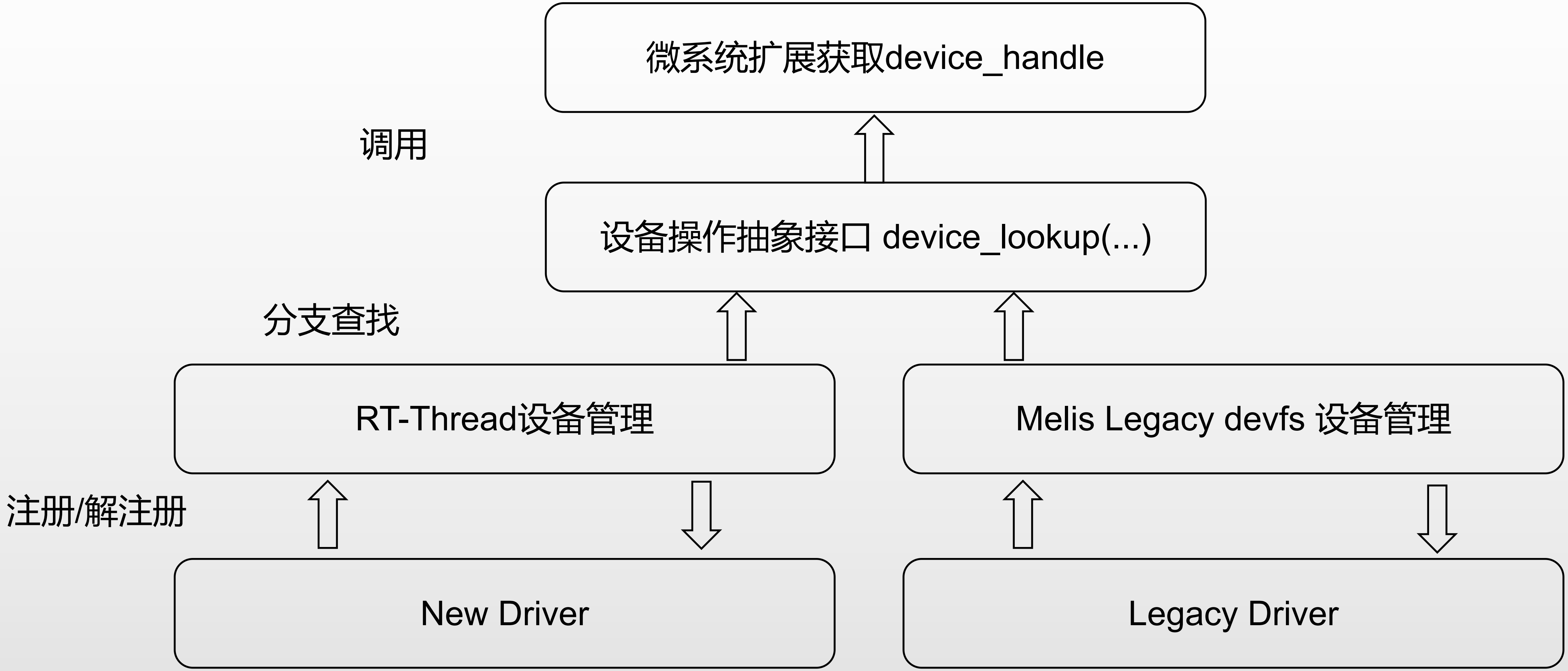
新架构-unikernel + legacy模块化扩展 - BSP系统无关

1.OS 无关，抽象OSAL 层

2.平台无关，抽象HAL 层



设备管理



微系统扩展获取device_handle

调用

设备操作抽象接口 device_lookup(...)

分支查找

RT-Thread设备管理

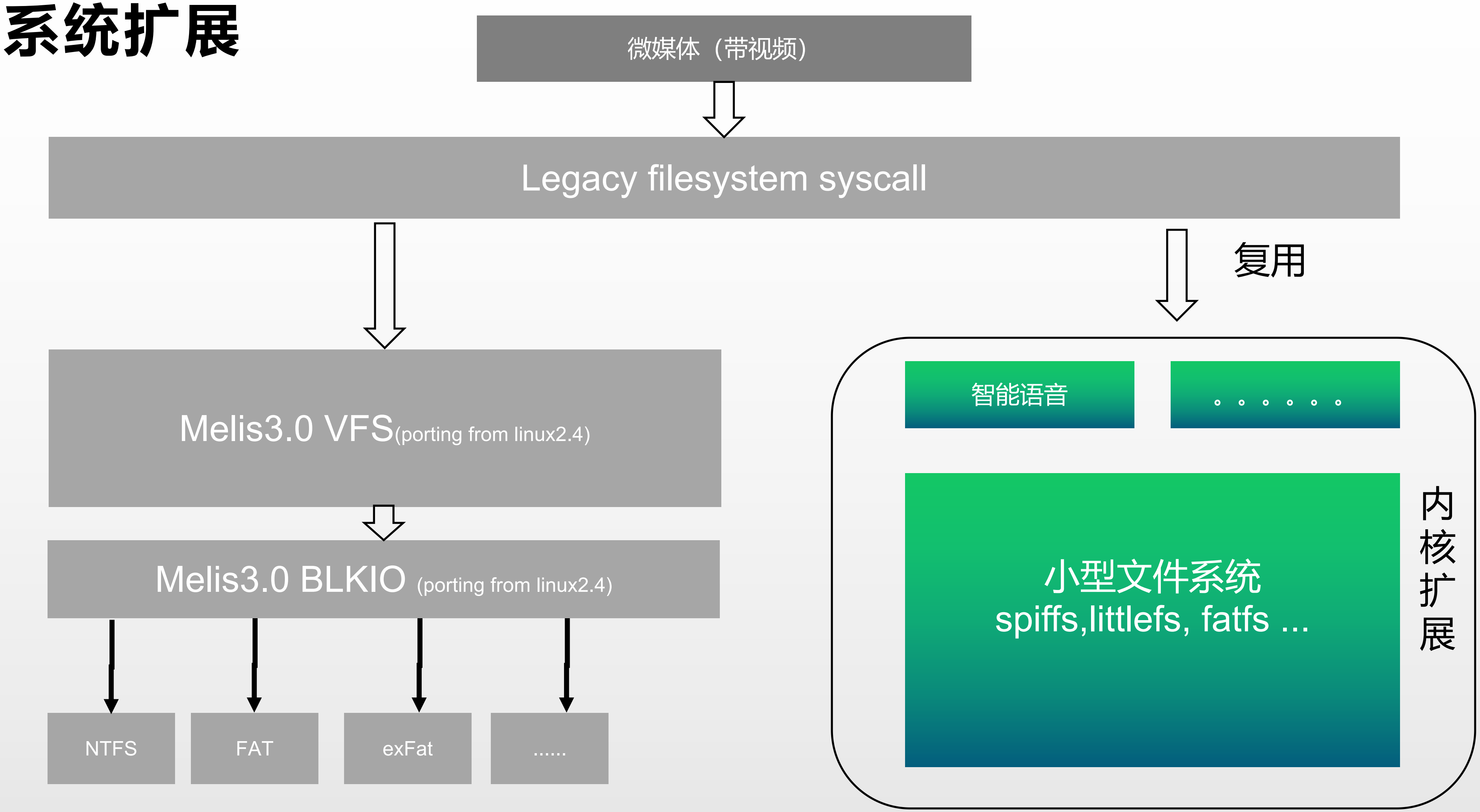
Melis Legacy devfs 设备管理

注册/解注册

New Driver

Legacy Driver

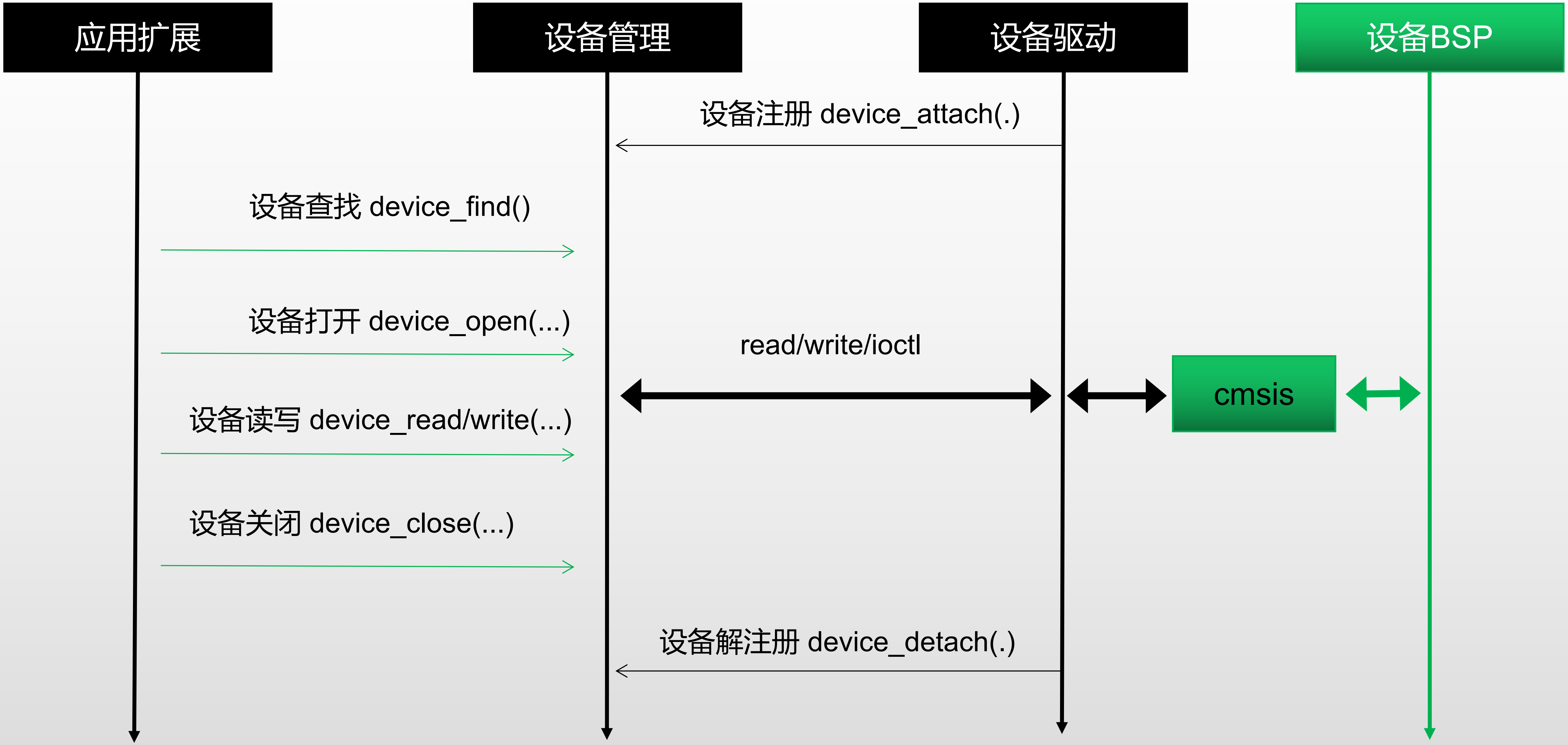
文件系统扩展


微媒体（带视频）
Legacy filesystem syscall
复用
Melis3.0 VFS(porting from linux2.4)
Melis3.0 BLKIO (porting from linux2.4)
NTFS
FAT
exFat
......
智能语音
。。。。。。
小型文件系统
spiffs,littlefs, fatfs ...
内核扩展

1.厚VFS,薄具体文件设备访问层
2.path_walk和blk buffer缓冲共享

pathwalk逻辑和缓冲，设备访问做在一起。

驱动开发标准：支持注册，解注册，打开，关闭等管理操作



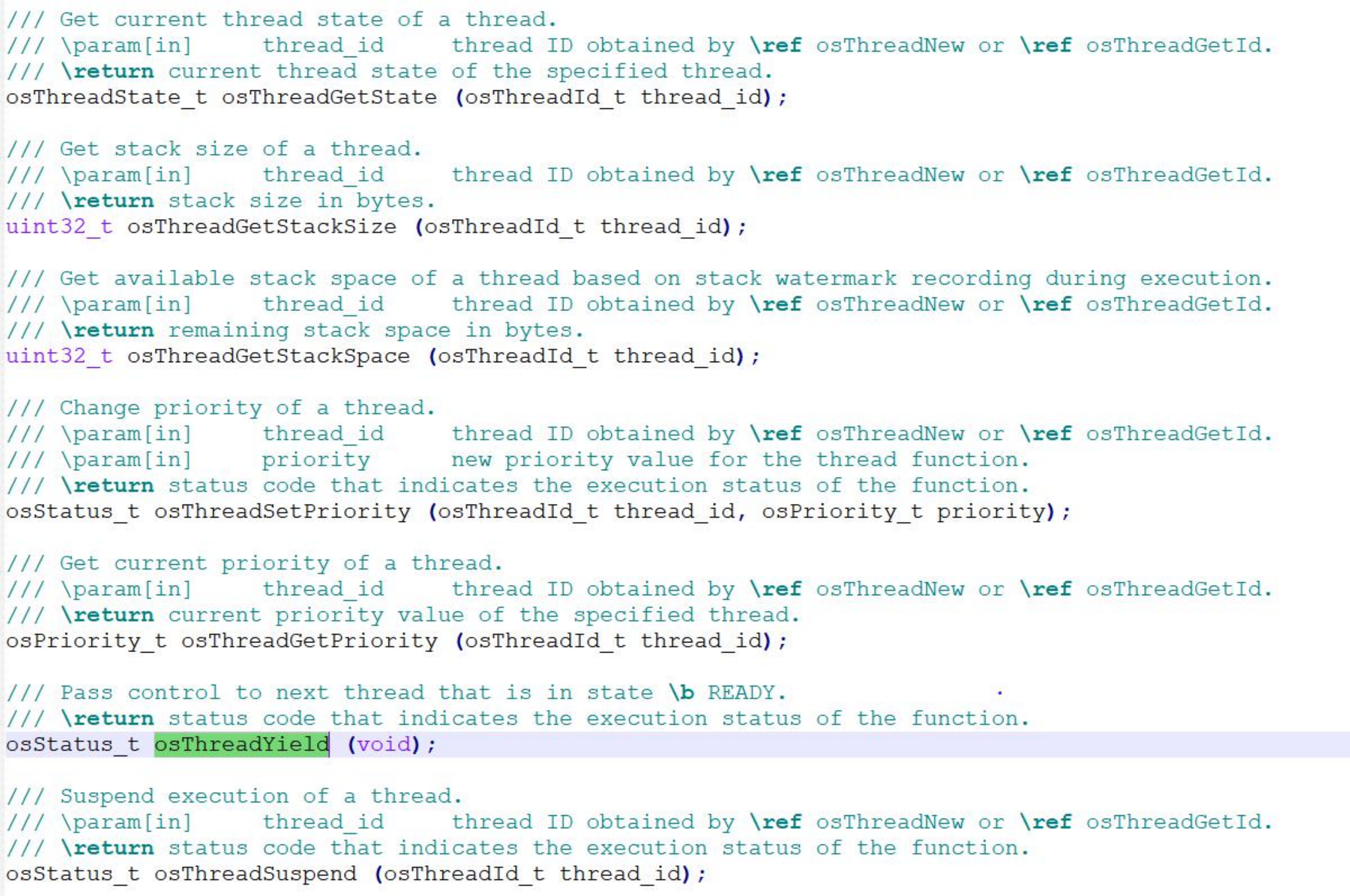
应用扩展
设备管理
设备驱动
设备BSP
设备注册 device_attach(.)
设备查找 device_find()
设备打开 device_open(...)
read/write/ioctl
cmsis
设备读写 device_read/write(...)
设备关闭 device_close(...)
设备解注册 device_detach(.)

CMSIS标准

- 面向cortex-m系列，并不一定完全follow
- 有些没有定义的，我们可以自己定义
- 重点在抽象化

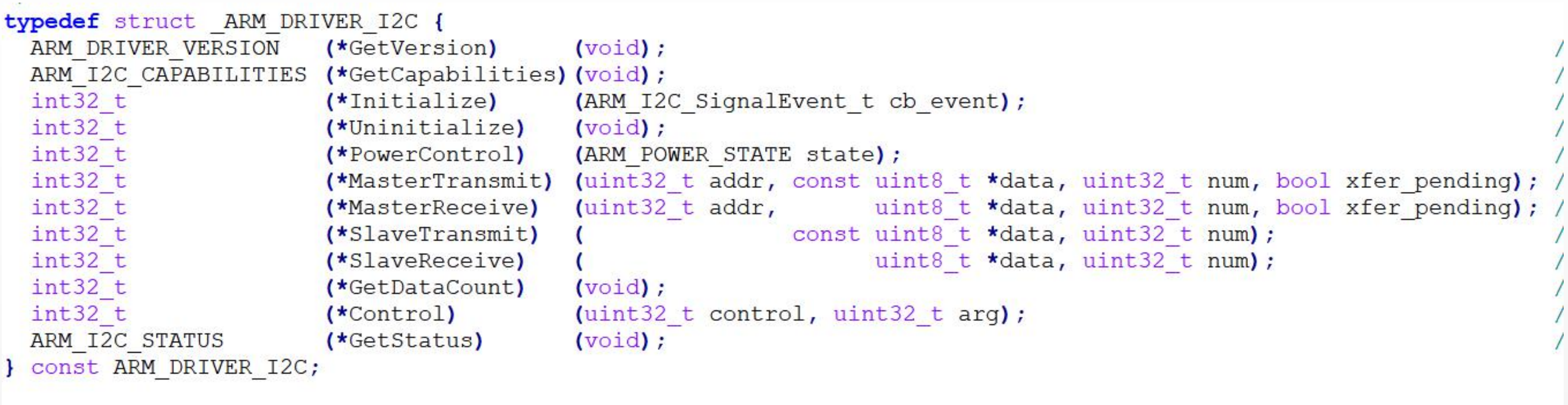
PSW不依赖SWC， SWC不依赖PSW， **共同依赖抽象层**

CMSIS OS标准

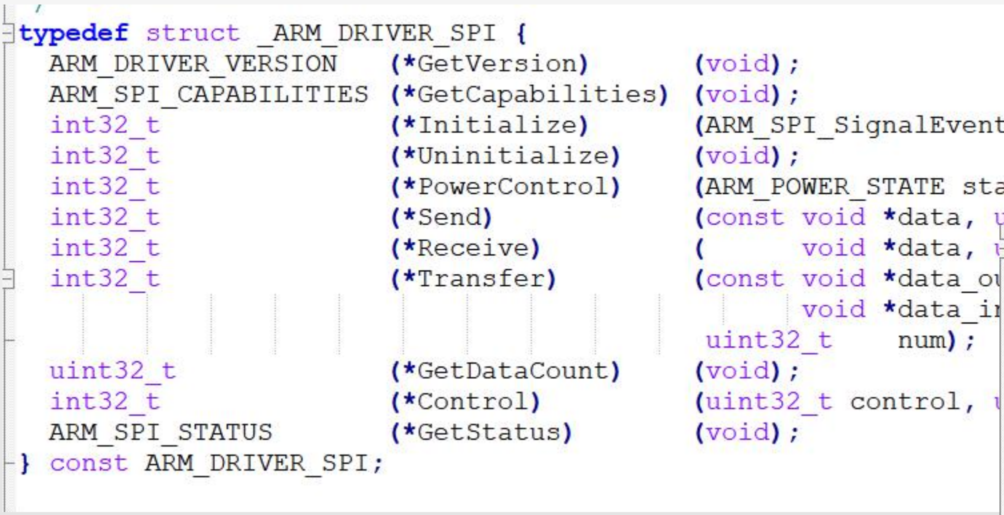


```
/// Get current thread state of a thread.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \return current thread state of the specified thread.
osThreadState_t osThreadGetState (osThreadId_t thread_id);

/// Get stack size of a thread.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \return stack size in bytes.
uint32_t osThreadGetStackSize (osThreadId_t thread_id);

/// Get available stack space of a thread based on stack watermark recording during execution.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \return remaining stack space in bytes.
uint32_t osThreadGetStackSpace (osThreadId_t thread_id);

/// Change priority of a thread.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \param[in]     priority      new priority value for the thread function.
/// \return status code that indicates the execution status of the function.
osStatus_t osThreadSetPriority (osThreadId_t thread_id, osPriority_t priority);

/// Get current priority of a thread.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \return current priority value of the specified thread.
osPriority_t osThreadGetPriority (osThreadId_t thread_id);

/// Pass control to next thread that is in state \b READY.
/// \return status code that indicates the execution status of the function.
osStatus_t osThreadYield (void);

/// Suspend execution of a thread.
/// \param[in]     thread_id     thread ID obtained by \ref osThreadNew or \ref osThreadGetId.
/// \return status code that indicates the execution status of the function.
osStatus_t osThreadSuspend (osThreadId_t thread_id);
```

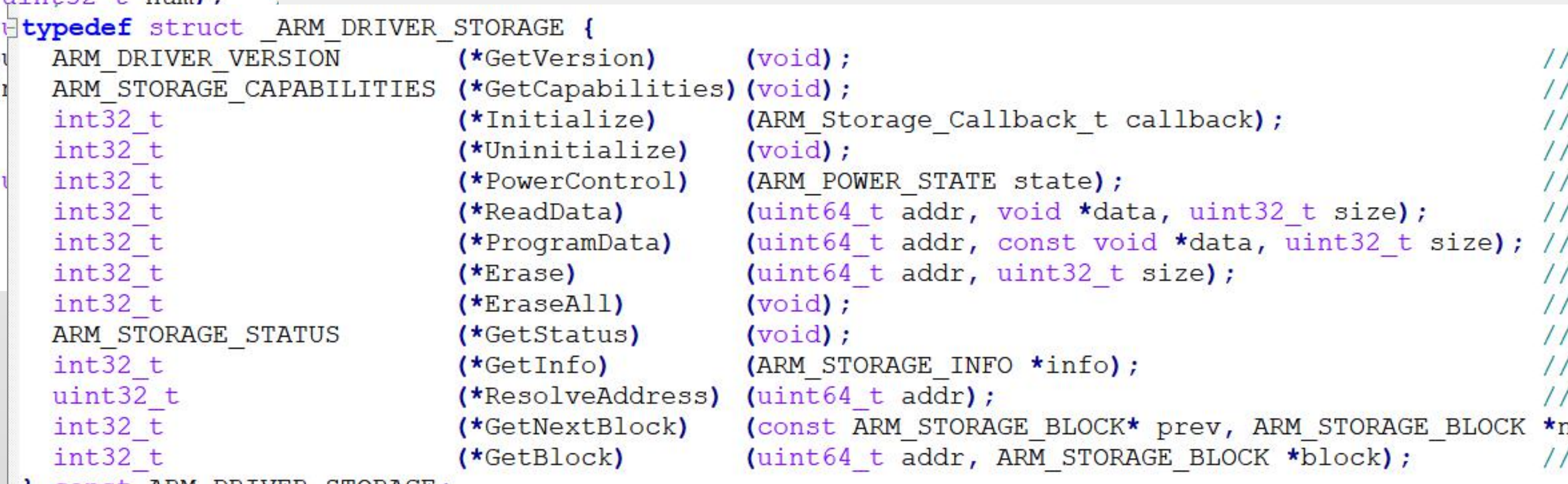
CMSIS Driver标准



```
typedef struct _ARM_DRIVER_I2C {
  ARM_DRIVER_VERSION     (*GetVersion)      (void);                                                          /
  ARM_I2C_CAPABILITIES (*GetCapabilities) (void);                                                          /
  int32_t                (*Initialize)      (ARM_I2C_SignalEvent_t cb_event);                                /
  int32_t                (*Uninitialize)    (void);                                                          /
  int32_t                (*PowerControl)    (ARM_POWER_STATE state);                                         /
  int32_t                (*MasterTransmit)  (uint32_t addr, const uint8_t *data, uint32_t num, bool xfer_pending); /
  int32_t                (*MasterReceive)   (uint32_t addr,       uint8_t *data, uint32_t num, bool xfer_pending); /
  int32_t                (*SlaveTransmit)   (               const uint8_t *data, uint32_t num);               /
  int32_t                (*SlaveReceive)    (                     uint8_t *data, uint32_t num);               /
  int32_t                (*GetDataCount)    (void);                                                          /
  int32_t                (*Control)         (uint32_t control, uint32_t arg);                                /
  ARM_I2C_STATUS         (*GetStatus)       (void);                                                          /
} const ARM_DRIVER_I2C;
```

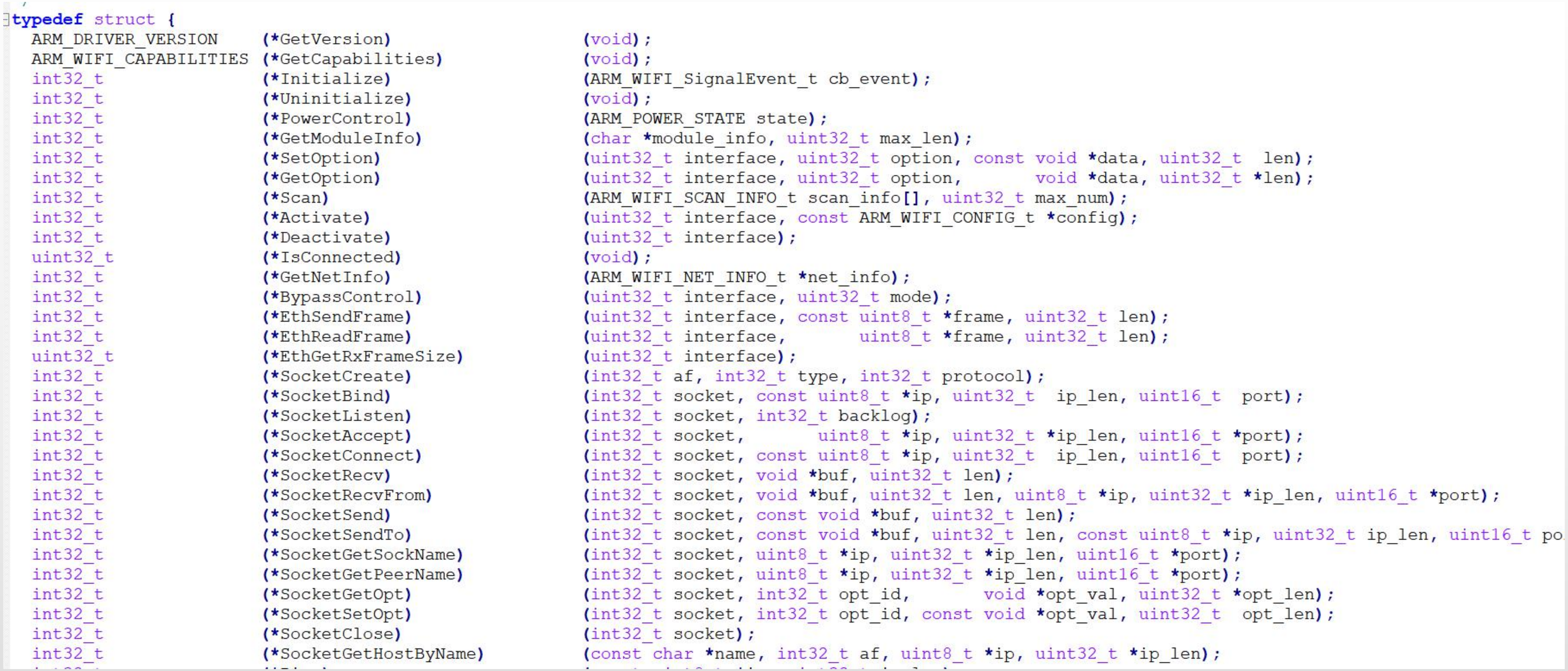


```
typedef struct _ARM_DRIVER_SPI {
  ARM_DRIVER_VERSION     (*GetVersion)      (void);
  ARM_SPI_CAPABILITIES (*GetCapabilities) (void);
  int32_t                (*Initialize)      (ARM_SPI_SignalEvent_t cb_event);
  int32_t                (*Uninitialize)    (void);
  int32_t                (*PowerControl)    (ARM_POWER_STATE state);
  int32_t                (*Send)            (const void *data, uint32_t num);
  int32_t                (*Receive)         (      void *data, u
  int32_t                (*Transfer)        (const void *data_ou
                                                   void *data_ir
                                             uint32_t     num);
  uint32_t               (*GetDataCount)    (void);
  int32_t                (*Control)         (uint32_t control, u
  ARM_SPI_STATUS         (*GetStatus)       (void);
} const ARM_DRIVER_SPI;
```



```
typedef struct _ARM_DRIVER_STORAGE {
  ARM_DRIVER_VERSION         (*GetVersion)       (void);                                                   //
  ARM_STORAGE_CAPABILITIES (*GetCapabilities) (void);                                                   //
  int32_t                    (*Initialize)       (ARM_Storage_Callback_t callback);                        //
  int32_t                    (*Uninitialize)     (void);                                                   //
  int32_t                    (*PowerControl)     (ARM_POWER_STATE state);                                  //
  int32_t                    (*ReadData)         (uint64_t addr, void *data, uint32_t size);               //
  int32_t                    (*ProgramData)      (uint64_t addr, const void *data, uint32_t size);         //
  int32_t                    (*Erase)            (uint64_t addr, uint32_t size);                           //
  int32_t                    (*EraseAll)         (void);                                                   //
  ARM_STORAGE_STATUS         (*GetStatus)        (void);                                                   //
  int32_t                    (*GetInfo)          (ARM_STORAGE_INFO *info);                                 //
  uint32_t                   (*ResolveAddress)   (uint64_t addr);                                          //
  int32_t                    (*GetNextBlock)     (const ARM_STORAGE_BLOCK* prev, ARM_STORAGE_BLOCK *n
  int32_t                    (*GetBlock)         (uint64_t addr, ARM_STORAGE_BLOCK *block);                //
} const ARM_DRIVER_STORAGE;
```

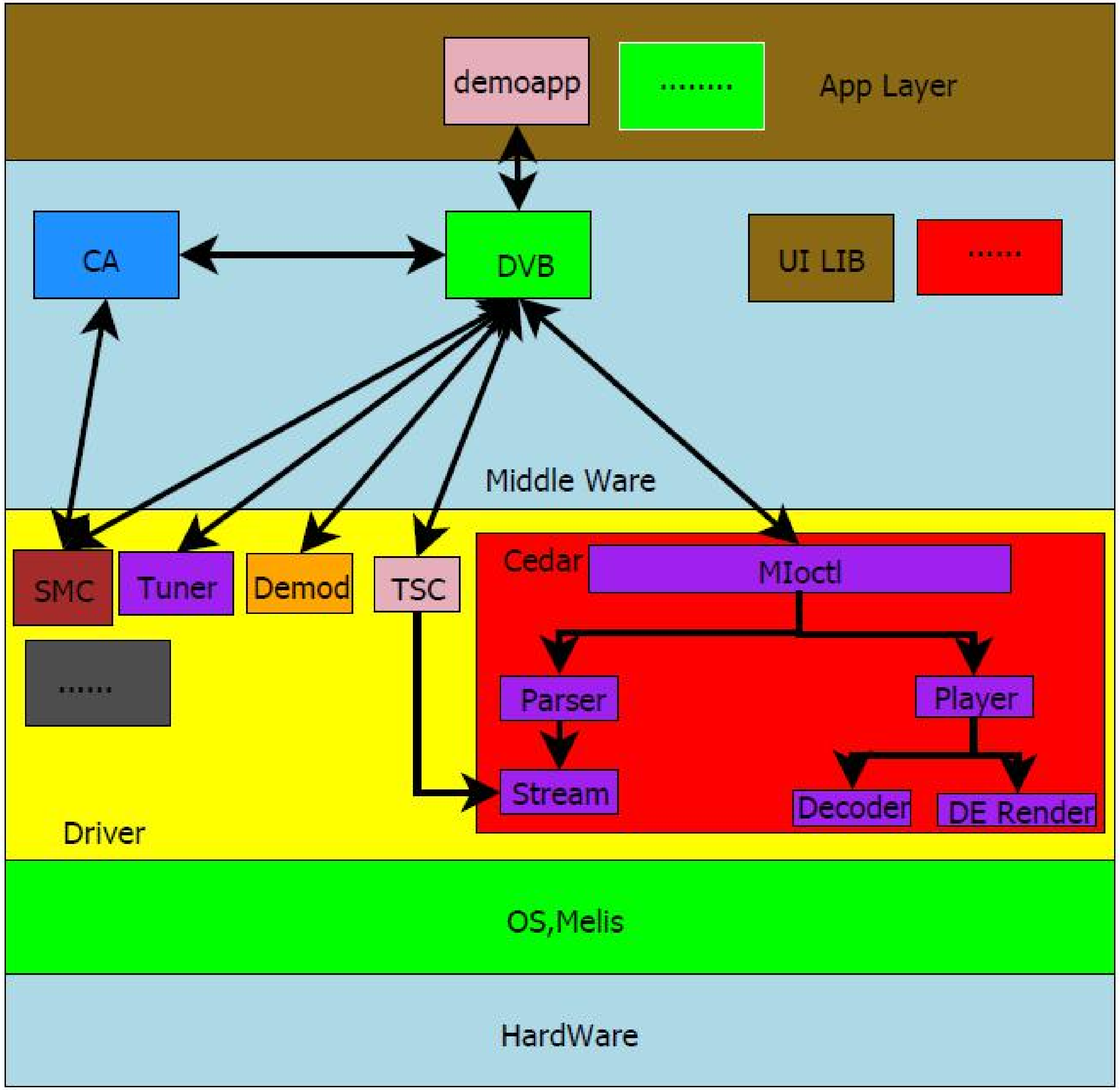
CMSIS Wifi 接口标准



```
typedef struct {
  ARM_DRIVER_VERSION     (*GetVersion)                  (void);
  ARM_WIFI_CAPABILITIES  (*GetCapabilities)             (void);
  int32_t                (*Initialize)                  (ARM_WIFI_SignalEvent_t cb_event);
  int32_t                (*Uninitialize)                (void);
  int32_t                (*PowerControl)                (ARM_POWER_STATE state);
  int32_t                (*GetModuleInfo)               (char *module_info, uint32_t max_len);
  int32_t                (*SetOption)                   (uint32_t interface, uint32_t option, const void *data, uint32_t  len);
  int32_t                (*GetOption)                   (uint32_t interface, uint32_t option,       void *data, uint32_t *len);
  int32_t                (*Scan)                        (ARM_WIFI_SCAN_INFO_t scan_info[], uint32_t max_num);
  int32_t                (*Activate)                    (uint32_t interface, const ARM_WIFI_CONFIG_t *config);
  int32_t                (*Deactivate)                  (uint32_t interface);
  uint32_t               (*IsConnected)                 (void);
  int32_t                (*GetNetInfo)                  (ARM_WIFI_NET_INFO_t *net_info);
  int32_t                (*BypassControl)               (uint32_t interface, uint32_t mode);
  int32_t                (*EthSendFrame)                (uint32_t interface, const uint8_t *frame, uint32_t len);
  int32_t                (*EthReadFrame)                (uint32_t interface,       uint8_t *frame, uint32_t len);
  uint32_t               (*EthGetRxFrameSize)           (uint32_t interface);
  int32_t                (*SocketCreate)                (int32_t af, int32_t type, int32_t protocol);
  int32_t                (*SocketBind)                  (int32_t socket, const uint8_t *ip, uint32_t  ip_len, uint16_t  port);
  int32_t                (*SocketListen)                (int32_t socket, int32_t backlog);
  int32_t                (*SocketAccept)                (int32_t socket,       uint8_t *ip, uint32_t *ip_len, uint16_t *port);
  int32_t                (*SocketConnect)               (int32_t socket, const uint8_t *ip, uint32_t  ip_len, uint16_t  port);
  int32_t                (*SocketRecv)                  (int32_t socket, void *buf, uint32_t len);
  int32_t                (*SocketRecvFrom)              (int32_t socket, void *buf, uint32_t len, uint8_t *ip, uint32_t *ip_len, uint16_t *port);
  int32_t                (*SocketSend)                  (int32_t socket, const void *buf, uint32_t len);
  int32_t                (*SocketSendTo)                (int32_t socket, const void *buf, uint32_t len, const uint8_t *ip, uint32_t ip_len, uint16_t po
  int32_t                (*SocketGetSockName)           (int32_t socket, uint8_t *ip, uint32_t *ip_len, uint16_t *port);
  int32_t                (*SocketGetPeerName)           (int32_t socket, uint8_t *ip, uint32_t *ip_len, uint16_t *port);
  int32_t                (*SocketGetOpt)                (int32_t socket, int32_t opt_id,       void *opt_val, uint32_t *opt_len);
  int32_t                (*SocketSetOpt)                (int32_t socket, int32_t opt_id, const void *opt_val, uint32_t  opt_len);
  int32_t                (*SocketClose)                 (int32_t socket);
  int32_t                (*SocketGetHostByName)         (const char *name, int32_t af, uint8_t *ip, uint32_t *ip_len);
```

例子：2015/2016DTMB框架图，和大平台源码级复用

模块化是开发便利性的障碍，不是源码复用的障碍
每个模块都要“带帽子”


demoapp
....
App Layer
CA
DVB
UI LIB
......
Middle Ware
SMC
Tuner
Demod
TSC
Cedar
MIoctl
......
Parser
Player
Stream
Decoder
DE Render
Driver
OS,Melis
HardWare