Accton
Making Partnership Work

Introduction to Simba High Level Architecture

By Charlie Chen

Training Purpose

- To understand the high level architecture of the Simba platform
- To understand the important infrastructure design of the Simba platform

Outline

- Terminology
- Modularized Design
- System Operation Flow
- Loader and OS adopted in Simba
- Simba Runtime Layered Architecture
- Major System Components in Simba
- Generic Flow of a Forked Process

Terminology

- CSC → Computer Software Component. A module which is designed for a specific feature. And it will contain OM and MGR.
- OM → Object Management. A.k.a. database.
- MGR → Manager. MGR APIs will perform transactions specific to features of the CSC.
- PMGR, POM → Provide wrapper functions for MGR OM APIs which will encapsulate input arguments into request ipc msg and convert respond ipc msg into output arguments.

N.B. OM which is created on shared memory does not have POM interface.

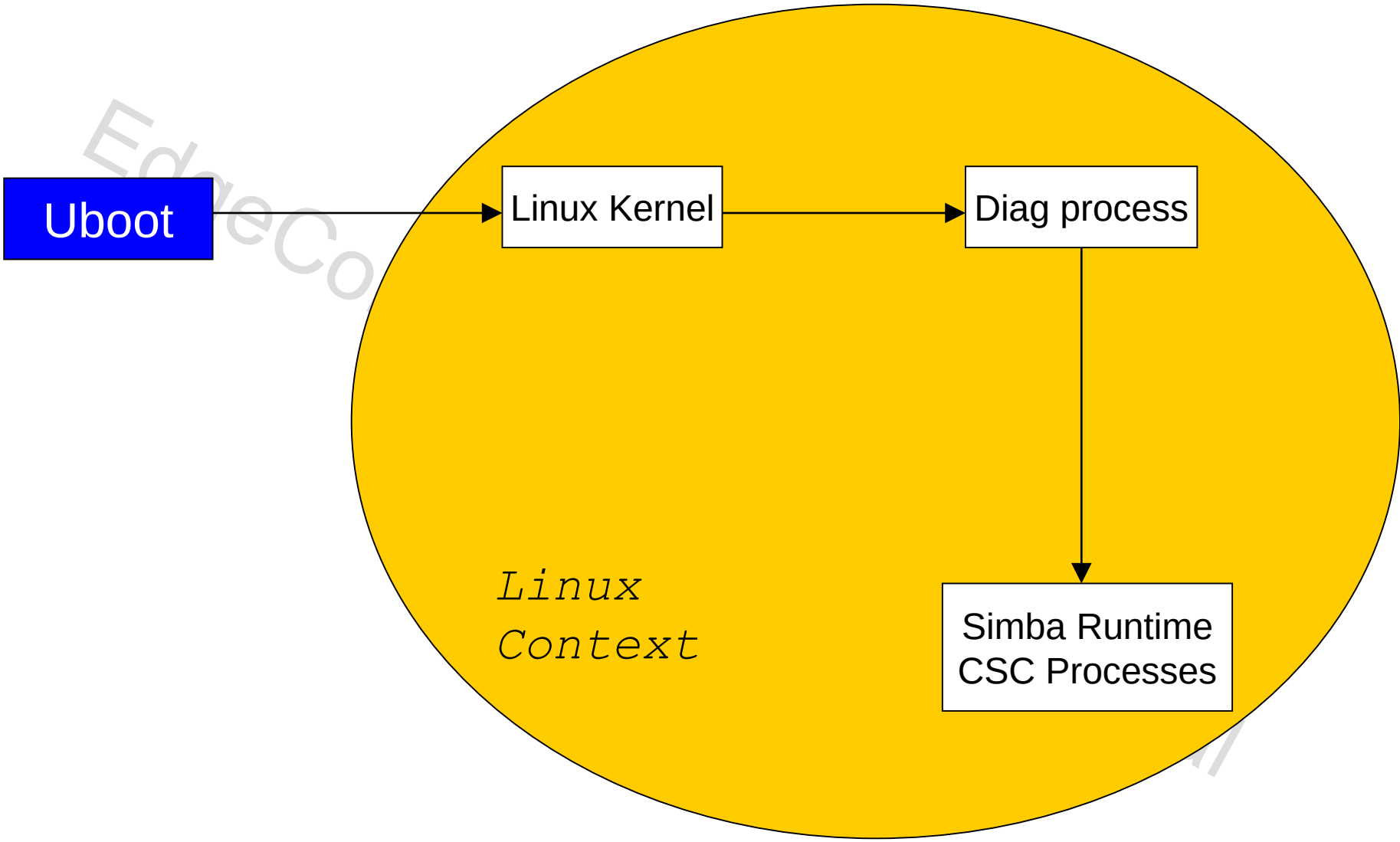
Modularized Design

- For each CSC, there will be a corresponding constant defined in “sys_cpnt.h”. For example, the line shown below will add “web” into the build.

```
#define SYS_CPNT_WEB  TRUE
```

- The source code related to the CSC will be enclosed by “#if (SYS_CPNT_WEB==TRUE)” for the example shown above.

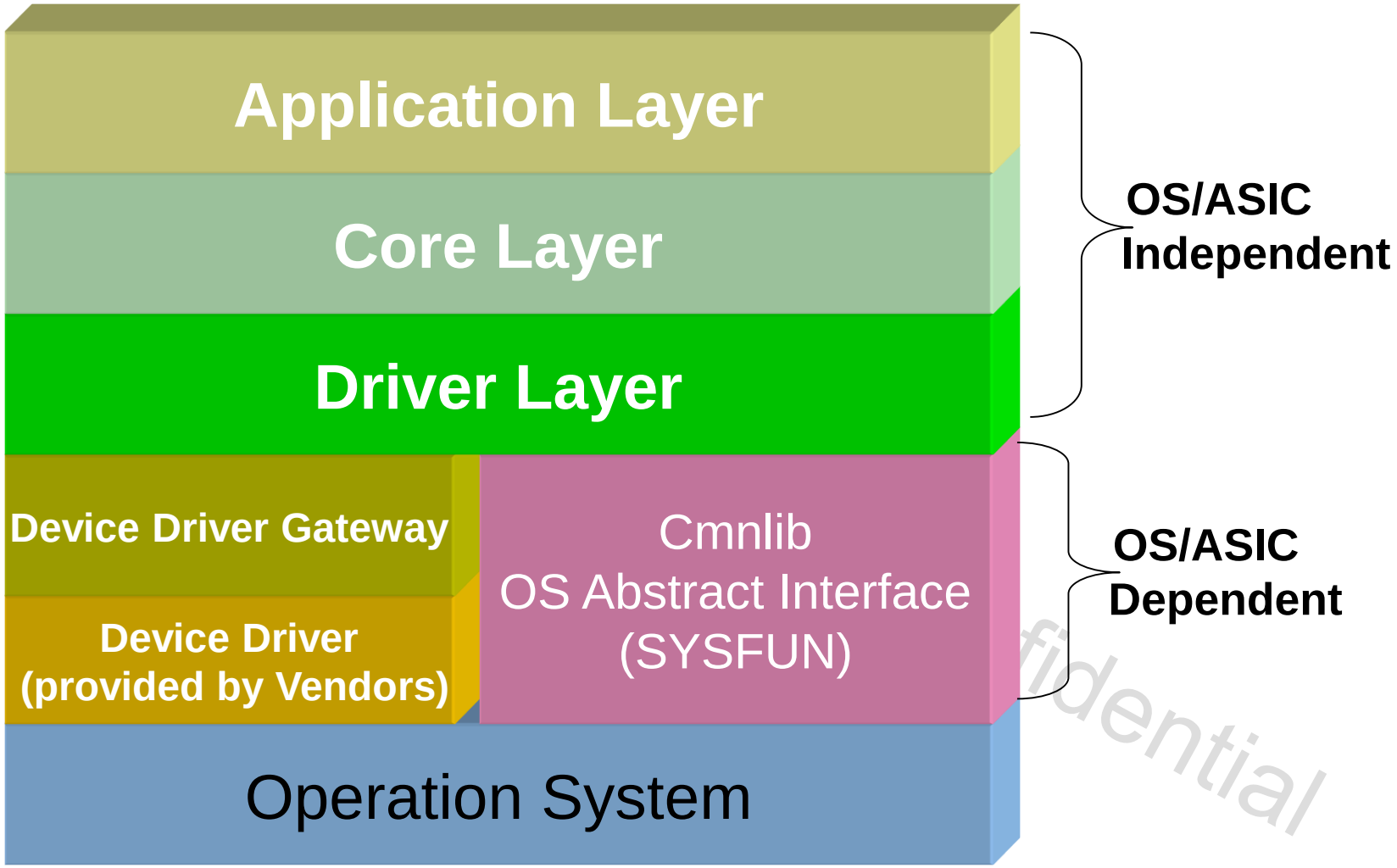
System Operation Flow



Loader and OS Adopted in Simba

- Loader: Uboot
- OS: Linux kernel
- C library: glibc

Simba Runtime Layered Hierarchy



N.B. Partial of modules in Cmnlib is OS Independent (e.g. sorted list).

Major System Components in Simba

- SYSFUN Message
- SYSFUN Event
- SYSINIT
- Callback Design
- CSC Group
- Communications among CSC Groups
- Rules about CSC Groups

SYSFUN Message (1/2)

- Two types of message
  - Synchronous message
    - The request message is handled by the message receiver and a response message is sent back to the message originator.
  - Asynchronous message
    - The asynchronous message just be enqueued to the message receiver's queue.

UI32_T SYSFUN_SendRequestMsg(UI32_T msgq_handle, SYSFUN_Msg_T * req_msg_p, UI32_T wait_time, UI32_T event, UI32_T res_msg_size, SYSFUN_Msg_T* res_msg_p);

SYSFUN Message (2/2)

- Two types of message queue
  - Uni-direction message queue(Rcv only)
    - Only support asynchronous messages.
  - Bi-direction message queue(Rcv and Respond)
    - Support both synchronous and asynchronous messages.

UI32_T SYSFUN_CreateMsgQ(UI32_T msgq_key, UI32_T msgq_type, UI32_T *msgq_handle_p);

msgq_type :
SYSFUN_MSGQ_UNIDIRECTIONAL/SYSFUN_MSGQ_BIDIRECTIONAL

Sysfun Event (1/2)

- Event is a bitmap with data type unsigned long(32 bits)
- Each thread can define the events it needs.
- SYSFUN_SendEvent(UI32_T tid, UI32_T event)
  - SYSFUN API to send event to the given tid.

Sysfun Event (2/2)

- SYSFUN_RecvEvent(UI32_T wait_event, UI32_T flags, int timeout, UI32_T *received_event)
  - SYSFUN API for each thread to receive the event that had been sent to it.
  - The caller of the API can choose to wait on all of the events in wait_event or just any one of the events in wait_event.
  - The caller of the API can determine the longest time to wait for a event by the argument “timeout”. It is also possible to perform unblocked operation (i.e. no wait).

SYSFUN_SendEvent Example

```
#define AMTRDRV_MGR_EVENT_ADDRESS_OPERATION
(1<<2)

BOOL_T AMTRDRV_MGR_DeleteAllAddr(void)
{
    AMTR_TYPE_BlockedCommand_T * blocked_command_p ;
    blocked_command_p =
AMTRDRV_OM_GetBlockCommand(0);
    blocked_command_p->blocked_command =
AMTR_TYPE_COMMAND_DELETE_ALL;
    SYSFUN_SendEvent(AMTRDRV_OM_GetAsicComTaskId(),
AMTRDRV_MGR_EVENT_ADDRESS_OPERATION);
    …
}
```

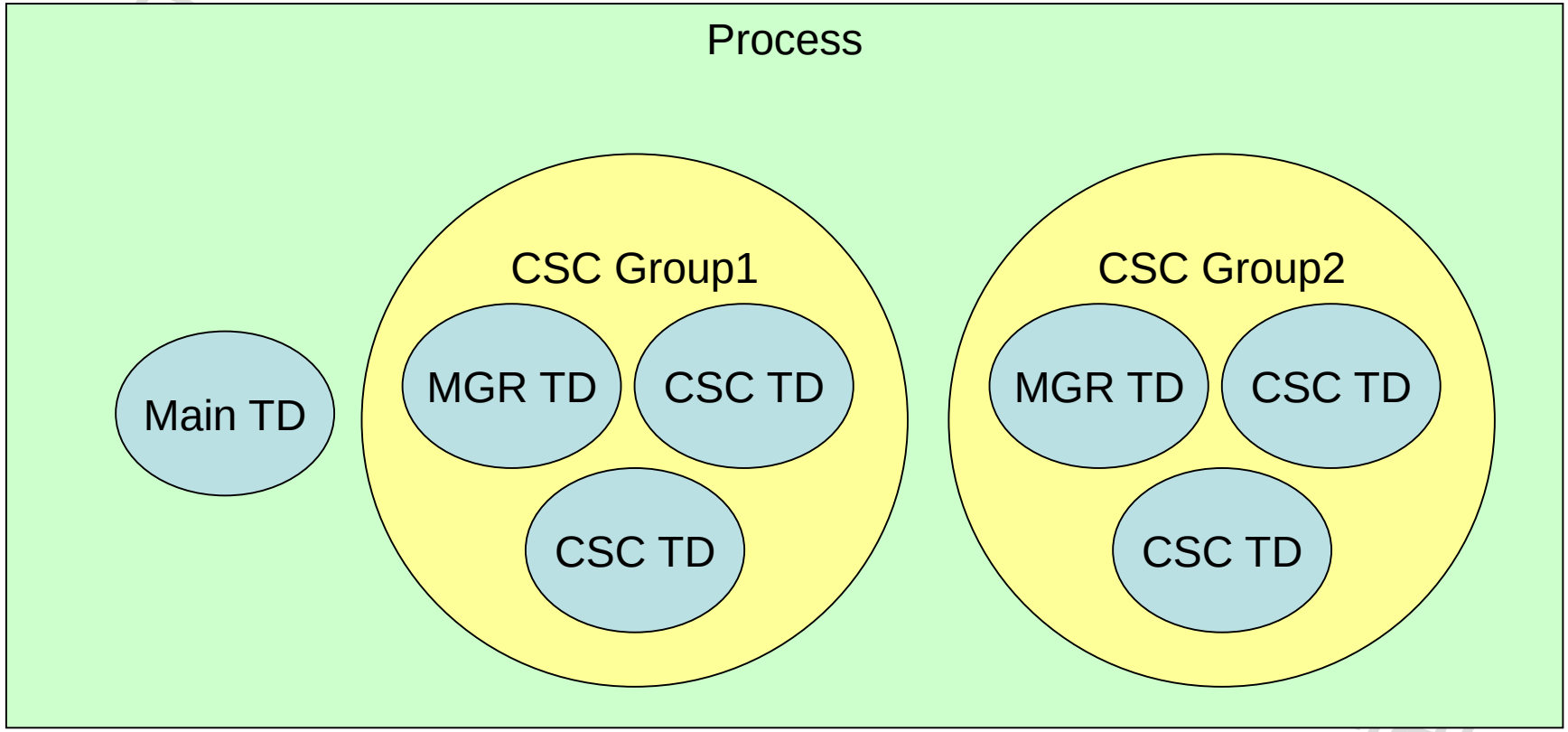
```
static void AMTRDRV_ASIC_COMMAND_TASK_Main(void)
{
    …
    SYSFUN_PeriodicTimer_Start(amtrdrv_asic_command_task_tmid,
                                          amtr_update_addr_table_ticks,
                                          AMTRDRV_MGR_EVENT_TIMER);
    events = 0;
    while(TRUE)
    {
        /* wait event
         */
        SYSFUN_ReceiveEvent ((AMTRDRV_MGR_EVENT_TIMER |
                              AMTRDRV_MGR_EVENT_ENTER_TRANSITION_MODE |
                              AMTRDRV_MGR_EVENT_ADDRESS_OPERATION),
                              SYSFUN_EVENT_WAIT_ANY,
                              (events!=0)? SYSFUN_TIMEOUT_NOWAIT:
                                           SYSFUN_TIMEOUT_WAIT_FOREVER,
                              &pending_events);
        events |= pending_events;

        if (events & AMTRDRV_MGR_EVENT_ADDRESS_OPERATION)
            …
    }
        …
}
```

SYSINIT

- SYSINIT is a program that is executed before any CSC processes spawns.
- SYSINIT will initialize system-wised resources(shared memory, sempahore) which will be used by CSCs.

Concept of a CSC Group



TD means “thread”.

Communications among CSC Groups

- Communications among CSC Groups are done through Synchronous/Asynchronous message queues.
- Each mgr thread in a CSC group owns a message queue which is assigned a pre-defined specific id. The mgr thread is responsible for handling all of the IPC request messages belong to the CSCs in the CSC group.
- All threads in the system are able to get the message queue handle by the pre-defined specific id.

Rules about CSC Group (1/3)

- The main thread is responsible for handling POM IPC messages if needed.
- Each CSC group must have a mgr thread to handle the IPC messages.
- Message queues will be used in communications among CSC groups.

Rules about CSC Group (2/3)

- A CSC may call MGR APIs of the other CSCs which belong to the same CSC group.
- A CSC has to call PMGR APIs of the other CSCs which belong to the CSC group different from the calling CSC.

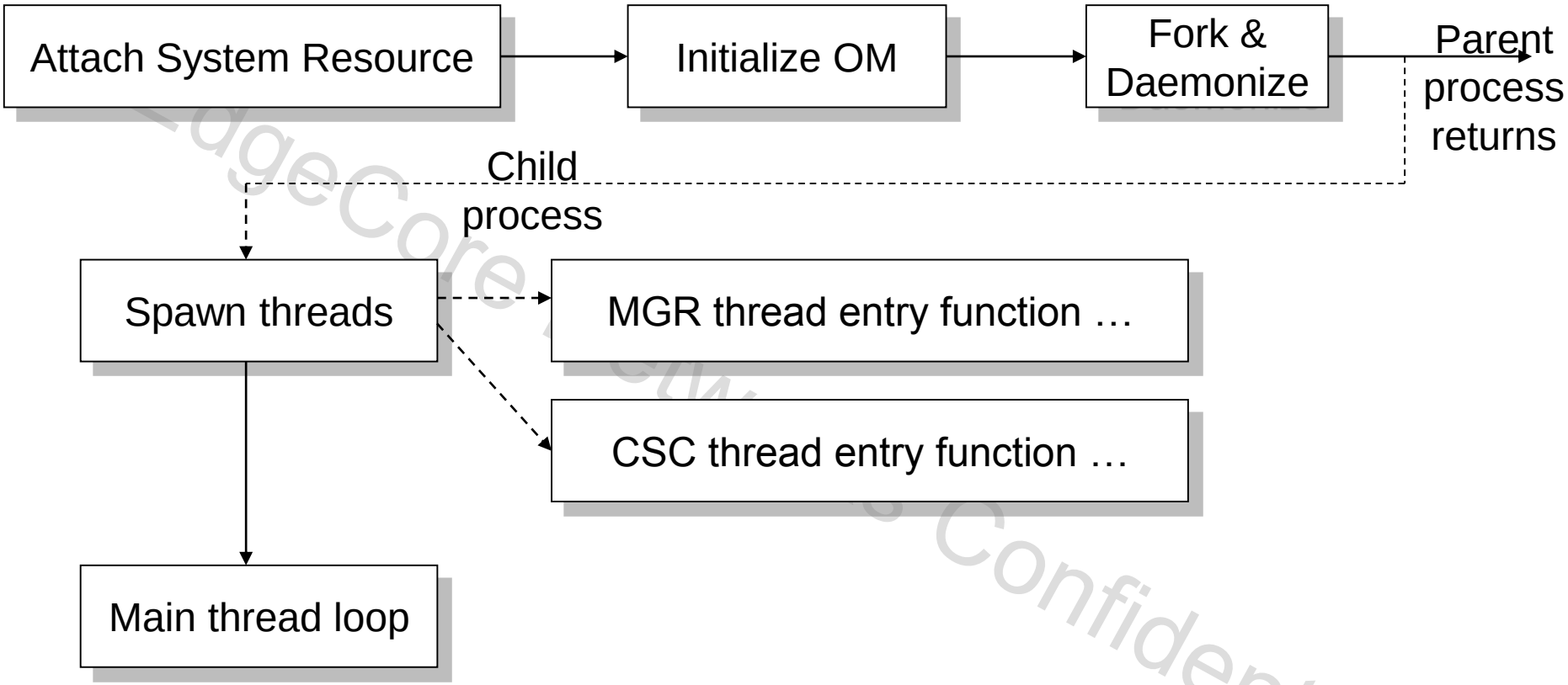
Rules about CSC Group (3/3)

- A CSC is allowed to call OM APIs(**read operations only**) of the other CSCs which belong to the same process.
- A CSC needs to call POM APIs(**read operations only**) of the other CSCs which belong to the process different from the calling CSC.

Callback Design

- Callbacks within the same CSC group
  - Direct function calls
- Callbacks to CSCs in the other CSC groups
  - Through SYS_CALLBACK_MGR
- SYS_CALLBACK_MGR provides APIs to handle the delivery of asynchronous SYSFUN message to CSC groups that need to receive.

Generic Flow of a Forked Process



Q&A

Thank you for your attention