Link architecture and MCFW

Agenda

- Terminologies and assumptions
- Component distribution
- Software stack
- Frame structure
- Link architecture
- Additional Links
- MCFW
- Code walk thorough
- Q and A

Terminologies

- MCFW – MultiChannel FrameWork – its just an interface!!
- Links – components in phase II software
- Chains – A complete data flow created connecting Links in phase II software

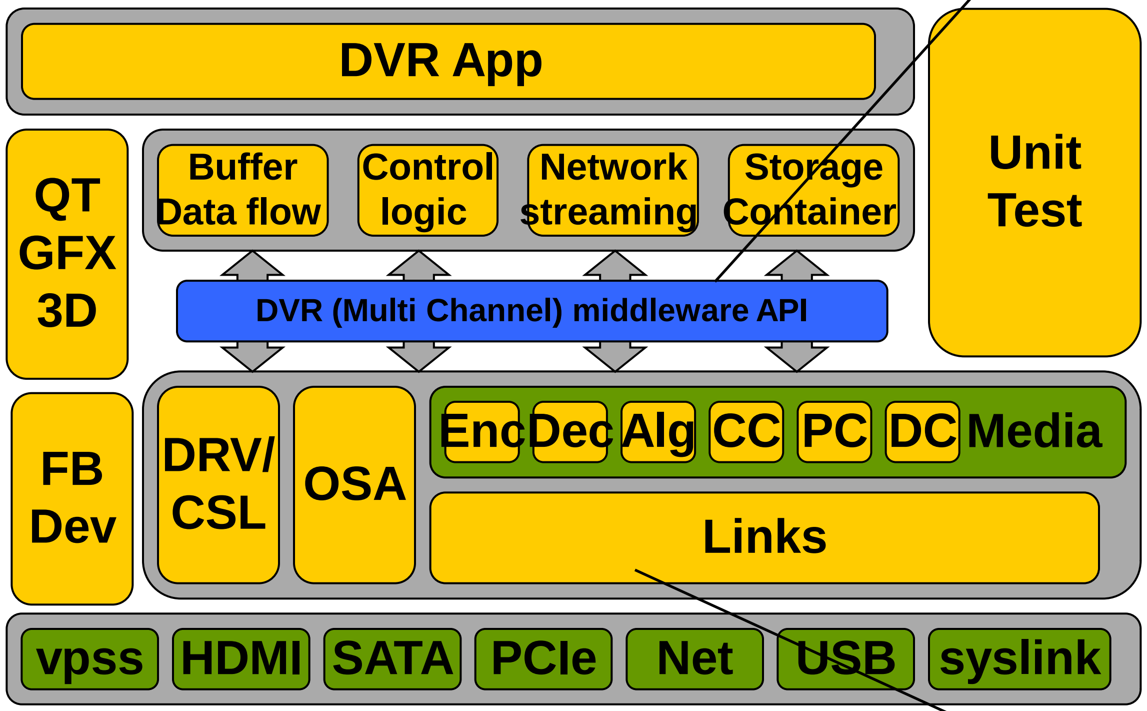
Assumptions

- You are familiar with the package names those come with dvr rdk mainly, SysLink, Ipc, FC, lsp etc…

Component distribution

- M3 Video
  - Encoder
  - Decoder
- M3 Vpss
  - Capture
  - Display
  - Noise Filter
  - Scalar
  - De-Interlacer
- A8
  - Network Stack
  - Hard disk read writes
  - ILClient (application)
- Dsp
  - OSD (On Screen Display)
  - SCD (Scene change Detection)

Software stack



API layer abstracting the underneath Framework

Application layer
-- Platform independent

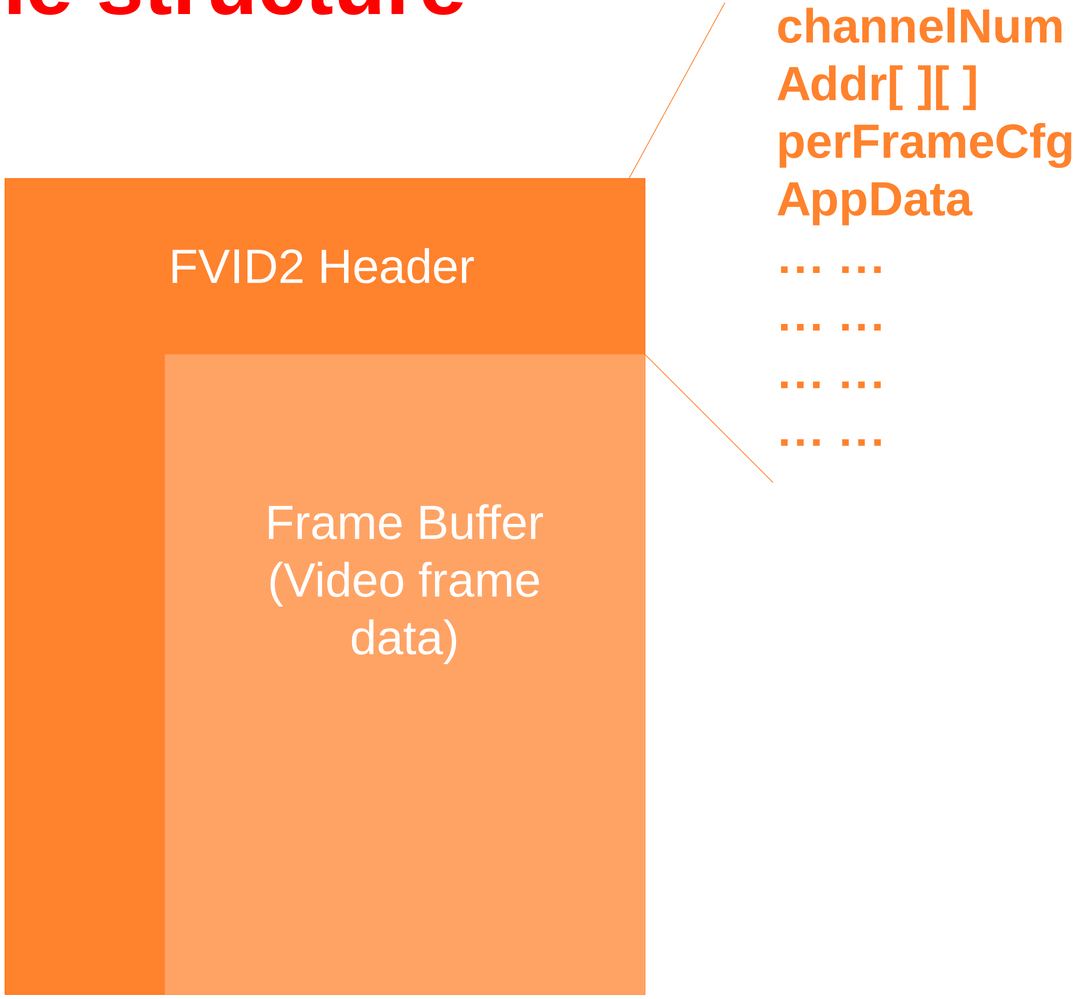
Function layer
-- Platform independent

Framework
CSL/DRV module
OSA – buf/log/task
media – VPSS and Video

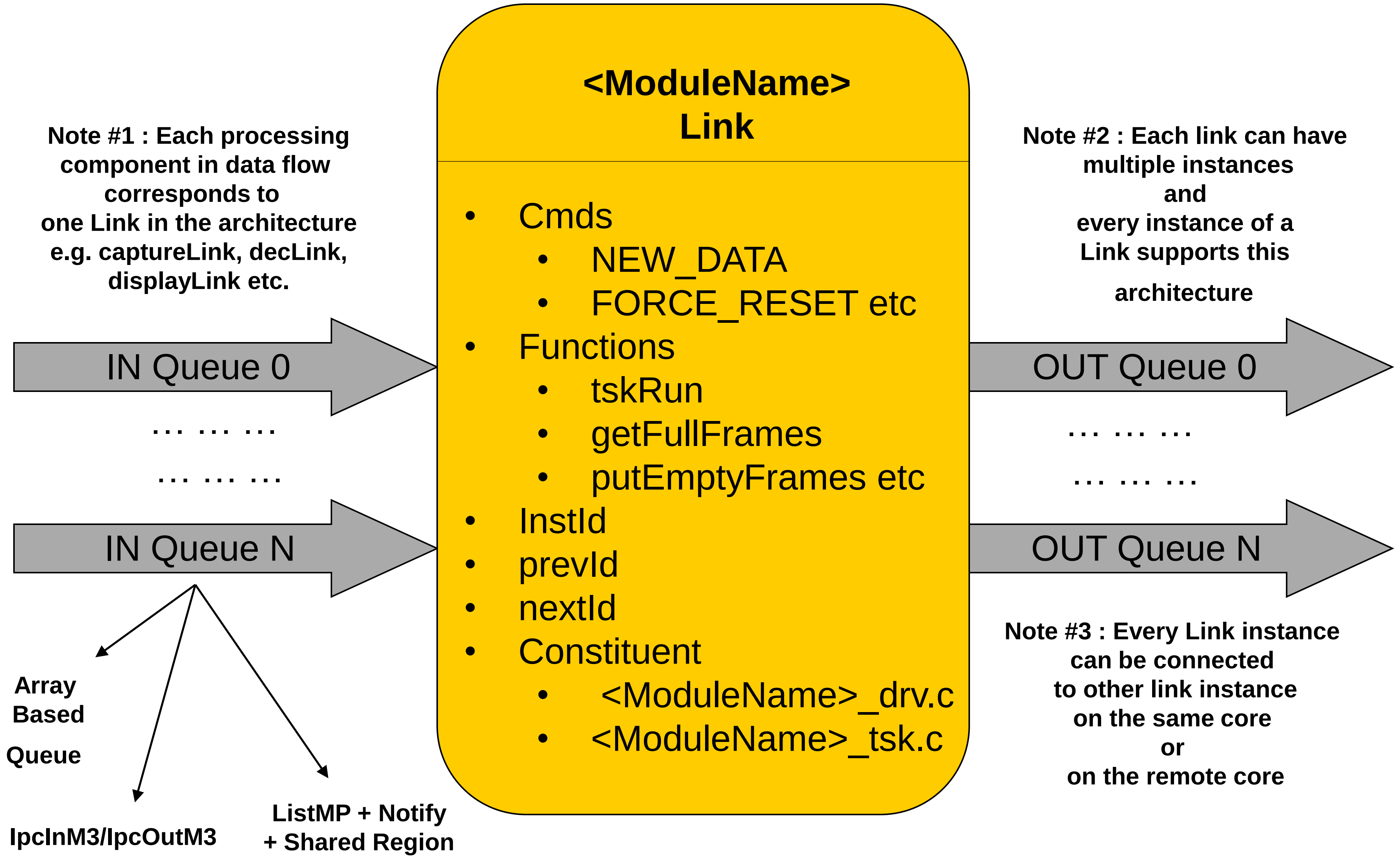
Linux & drivers

Links exchange FVID2 frames to transfer the data across

Frame structure



Link architecture



<ModuleName>
Link
Cmds
NEW_DATA
FORCE_RESET etc
Functions
tskRun
getFullFrames
putEmptyFrames etc
InstId
prevId
nextId
Constituent
<ModuleName>_drv.c
<ModuleName>_tsk.c
Note #1 : Each processing component in data flow corresponds to one Link in the architecture e.g. captureLink, decLink, displayLink etc.
Note #2 : Each link can have multiple instances and every instance of a Link supports this architecture
IN Queue 0
… … …
… … …
IN Queue N
OUT Queue 0
… … …
… … …
OUT Queue N
Note #3 : Every Link instance can be connected to other link instance on the same core or on the remote core
Array Based Queue
IpcInM3/IpcOutM3
ListMP + Notify + Shared Region

Inter-Link communication / Data transfer

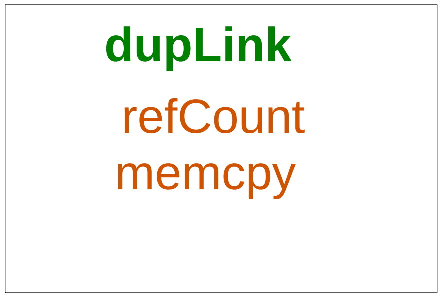
- In the Link based architecture, queues used to transfer data across are **chosen intelligently**. The following three inter link frame exchange mechanisms are used
- Intra-processor links
  - Example, from capture to noise filter which run on the same processor.
  - Simple and efficient array based queue's are used for frame exchange.
- Inter M3 (Video / VPSS) links
  - Example from NF to encode (via IPC M3 OUT/IN Link) which run on VPSS M3 and Video M3 (sharing a uni-cache).
  - IPC ListMP with Notify is used with frame information pointer (FVID2_Frame) being passed directly without any cache operations and address translation since both M3 share the same uni-cache.
- Inter processor (M3 to A8 or DSP)
  - Example from encode to Bitstream IN (via IPC OUT/IN Link) which run on Video M3 and Host A8.
  - This type of communication is achieved using ListMP, Notify and Shared Region modules from SysLink component.
  - M3 side cache operations are performed using XDC cache APIs
  - A8 side cache operations are performed using SysLink cache APIs

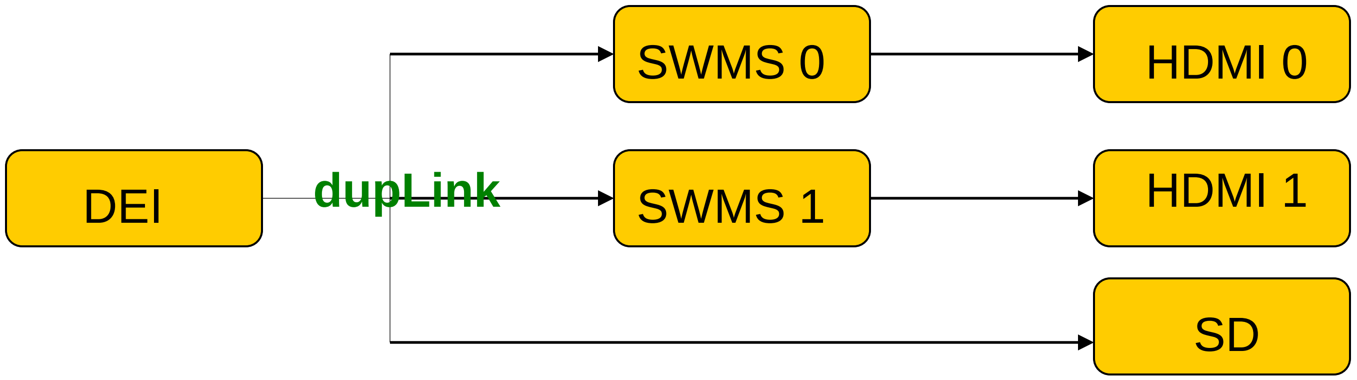
Additional Links

- To achieve the data flow for multiple possible usecases, having Link only for components is not sufficient
- We need following additional Links other than component links
  - dupLink
  - mergeLink
  - ipcBitsInLinkHLOS
  - ipcBitsOutLinkHLOS
  - ipcBitsInLinkRTOS
  - ipcBitsOutLinkRTOS
  - ipcFrameinLinkHLOS
  - ipcFrameOutLinkHLOS
  - ipcFrameInLinkRTOS
  - ipcFrameOutLinkRTOS
  - Null Link
  - NullSrc Link

Additional Links (continued..)

- Dup Link - why exactly it is needed?





•MergeLink

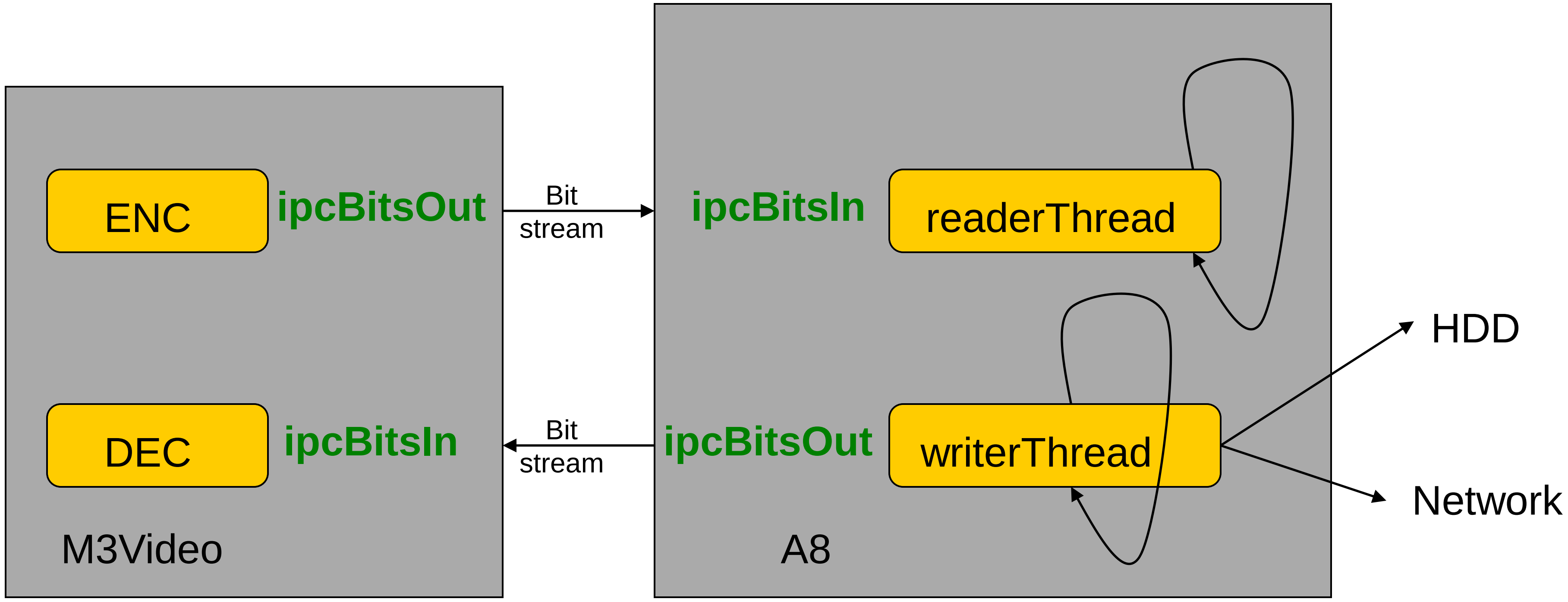
–Given N in put streams it produces single output stream.

–No memcpy involved in merge

–It has a pre-mapped input to output channel table

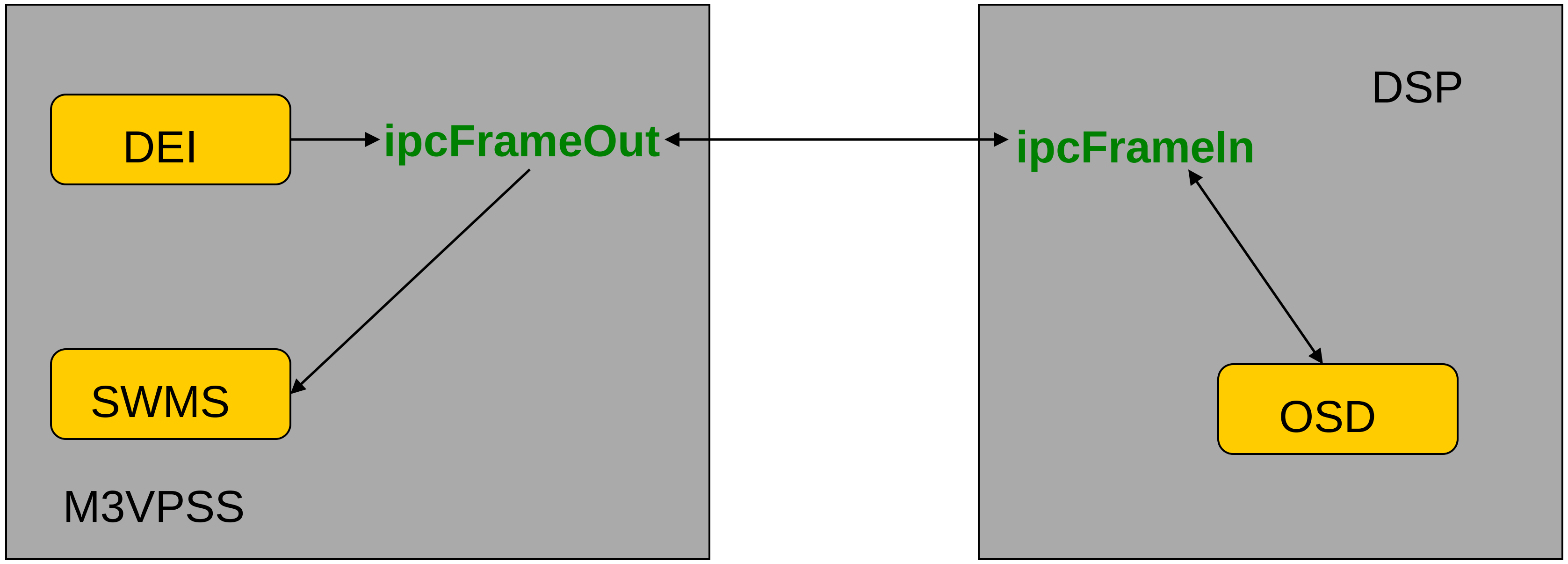
Additional Links (continued..)

- ipc**Bits**In/OutLinkRTOS/HLOS
  - Used send and receive bit stream across processors



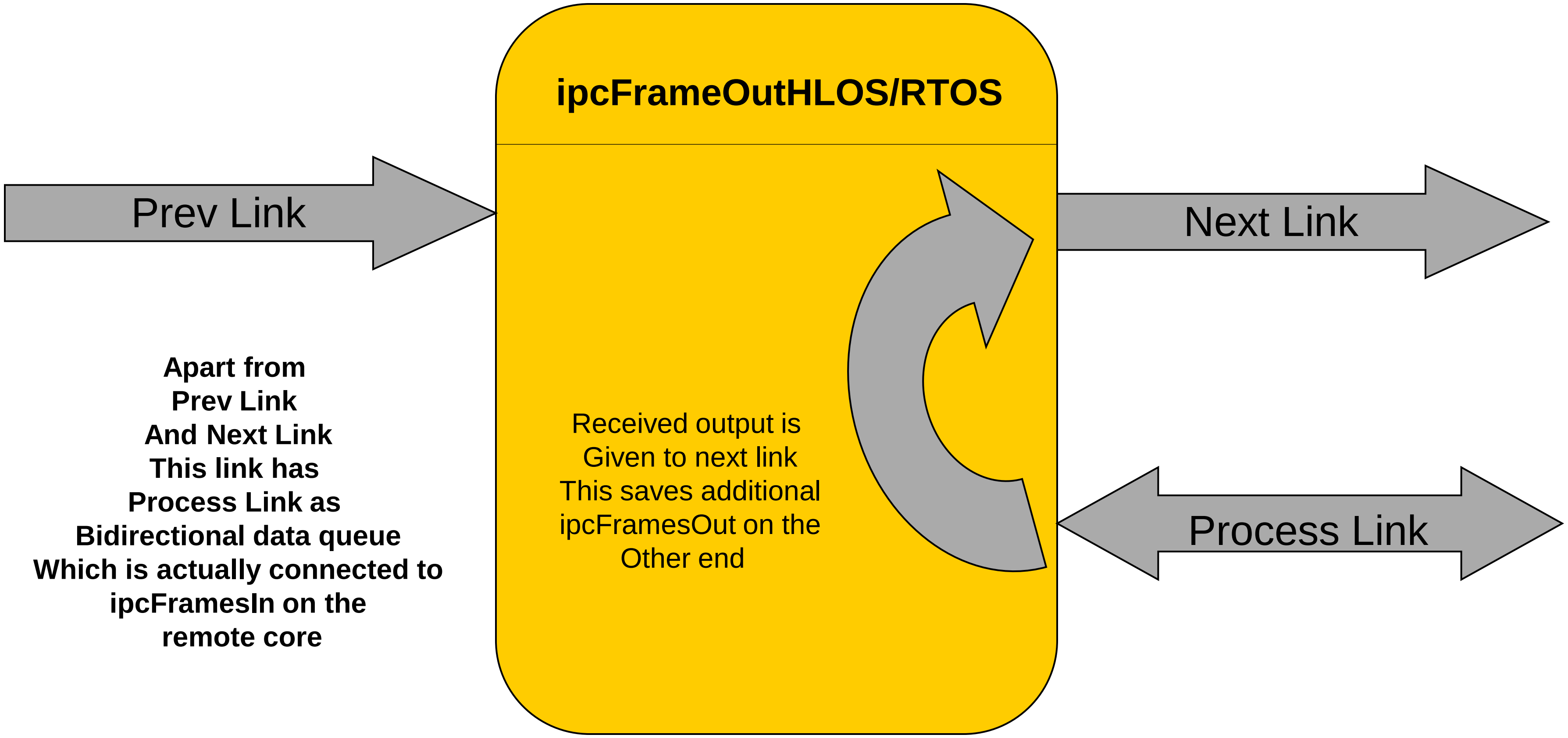
Additional Links (continued..)

- ipc**Frame**In/OutLinkHLOS/RTOS
  - Used send and receive frames across processors



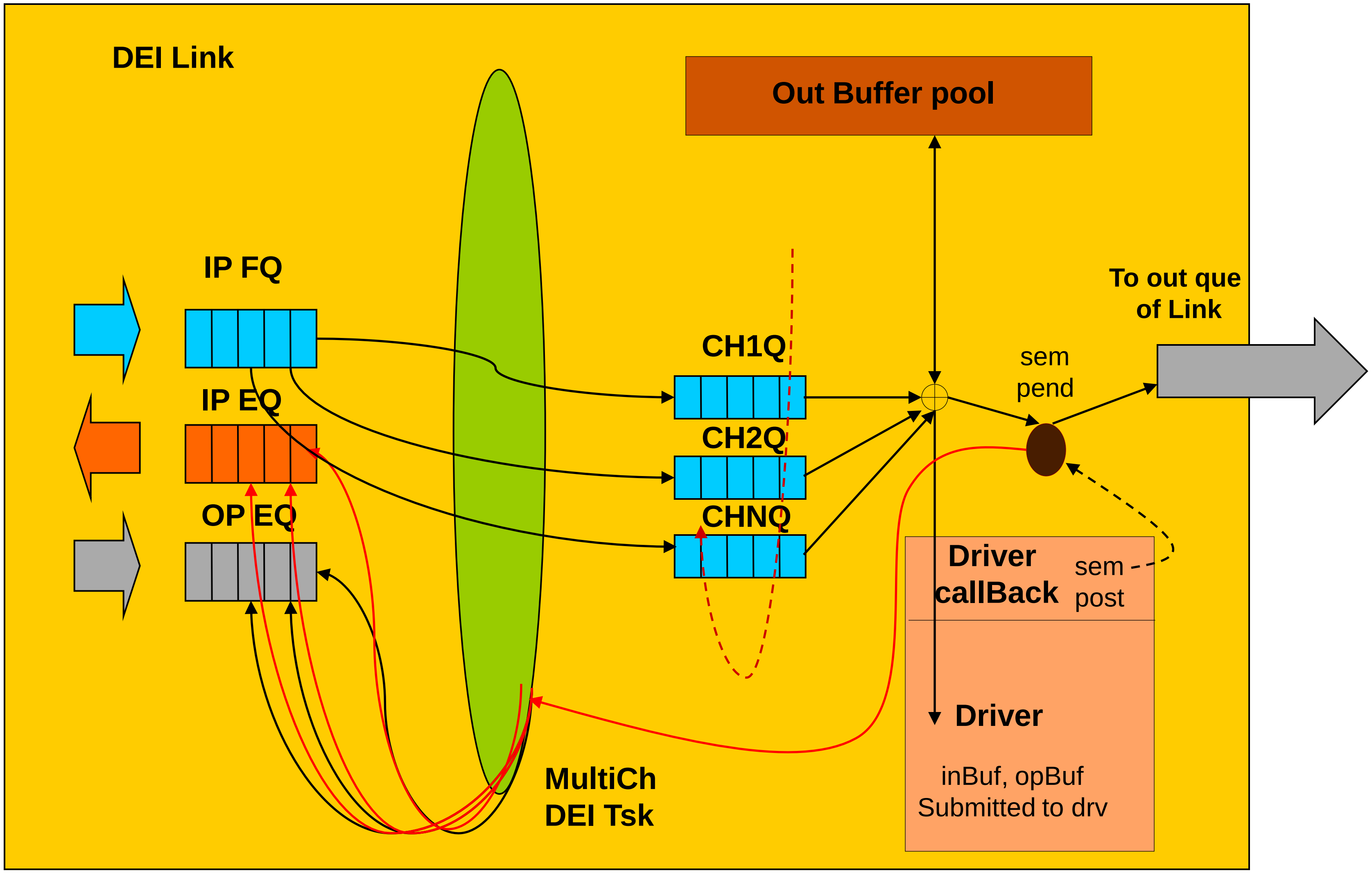
ipcFrameOut Link – different implementation

- ipc**Frame**In/OutLinkHLOS/RTOS
  - Used send and receive frames across processors

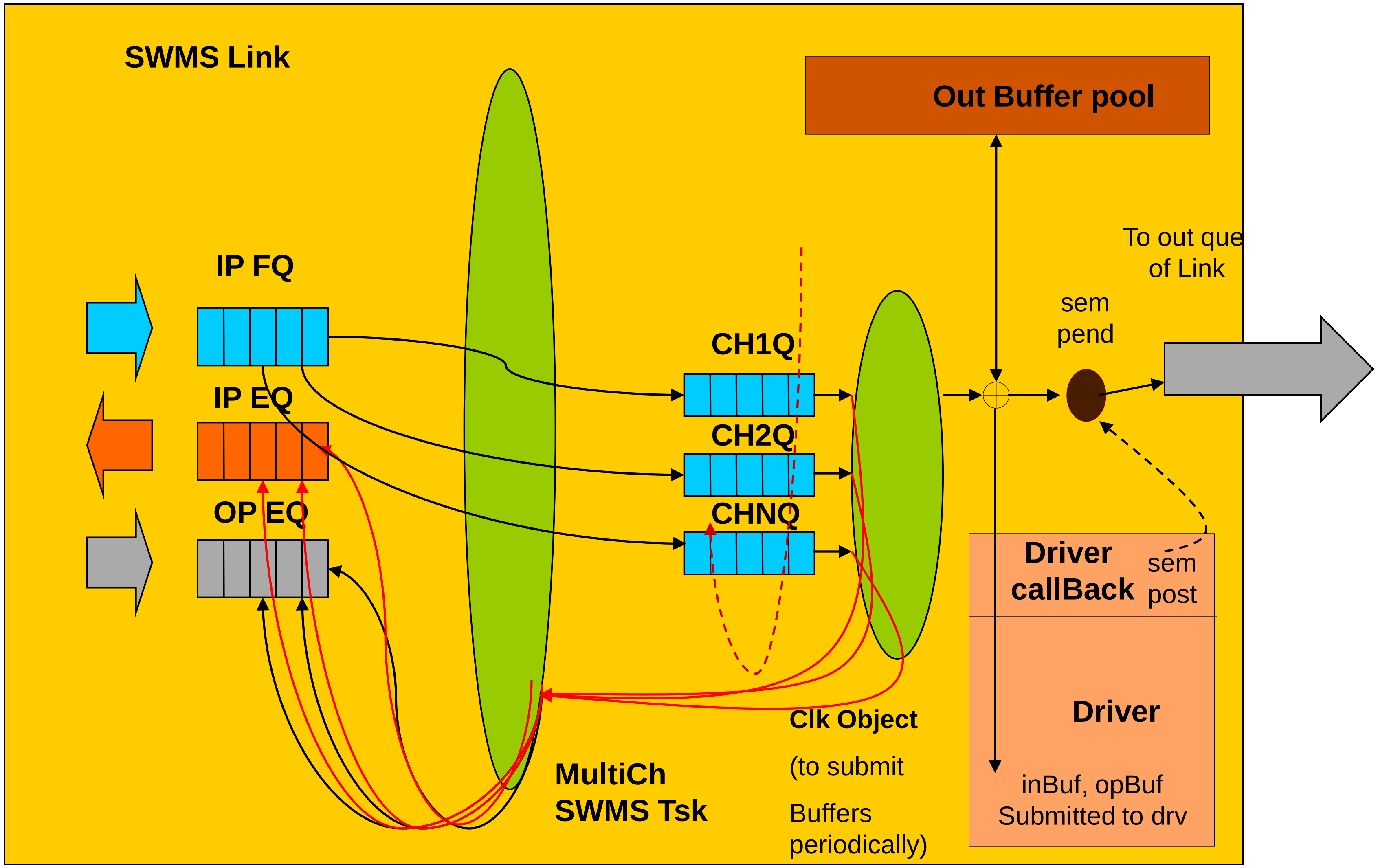

ipcFrameOutHLOS/RTOS
Prev Link
Next Link
Process Link
Received output is Given to next link This saves additional ipcFramesOut on the Other end

Apart from Prev Link And Next Link This link has Process Link as Bidirectional data queue Which is actually connected to ipcFramesIn on the remote core

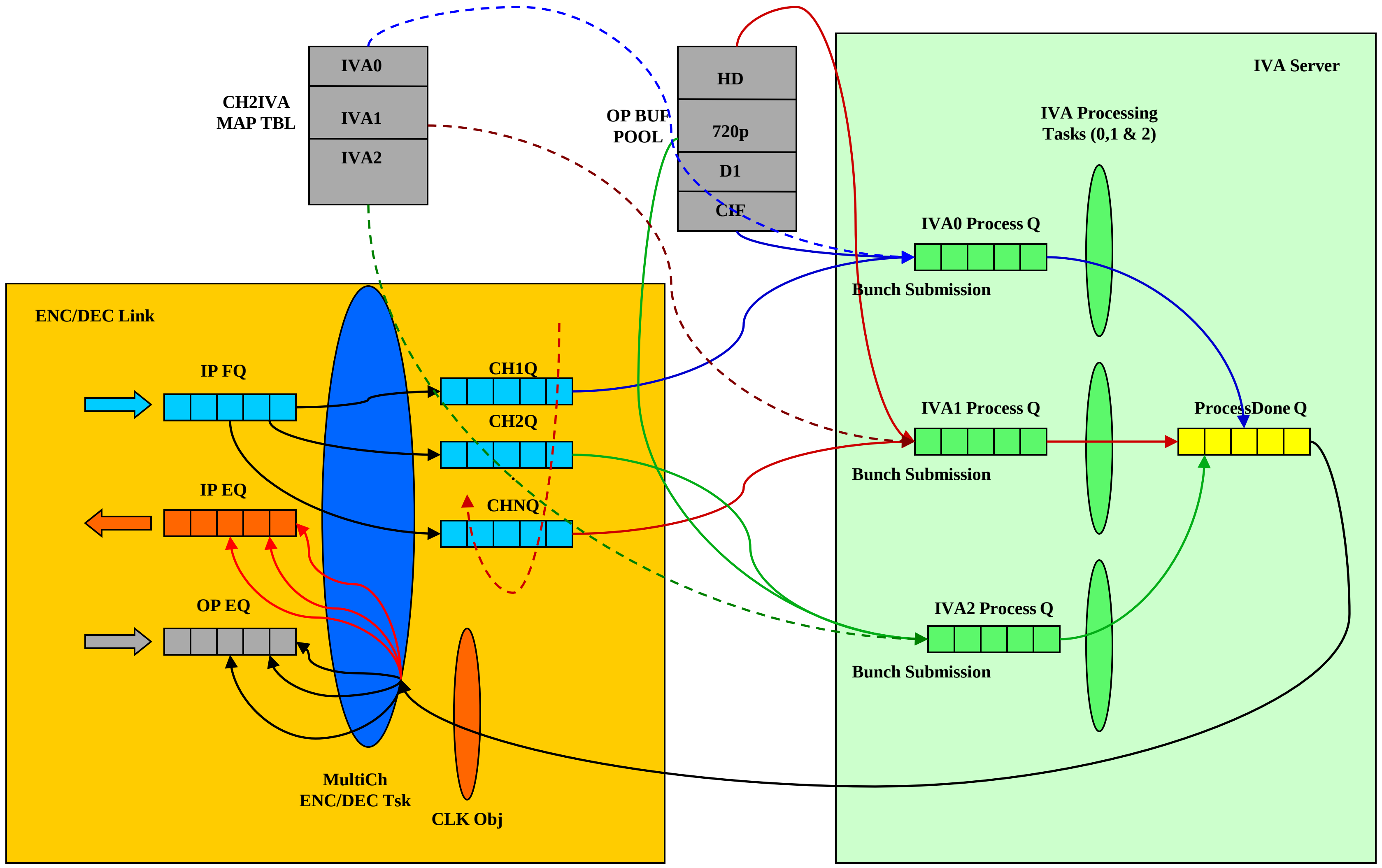
M3VPSS Links data flow (e.g. DEI)



SWMS Link (special case)



M3VIDEO Links data flow (e.g. ENC / DEC)



ENC/DEC Link – I/F & Threading Model

- ENC/DEC link architecture:
  - In Ducati videoM3 Encoder & Decoder are the two main links and each integrates codec Algorithm and the IVA scheduler
  - I/O interface
    - Input: Has one input frame Full Queue from previous link which feeds the link
    - Input: One empty frame Queue part of previous link to free the processed input
    - Output: Has one output buffer Full Queue to send out the processed data
    - Output: One or more empty buffer Queue to receive the free frames from the next link
  - Encoder & decoder links are exactly same. Only difference is they integrates different codec algorithms. So, Enc/Dec links architecture are not separately discussed here
  - Please refer the previous slide for Enc/Dec architecture details
  - Enc/Dec links have one link Task, 3 processing Tasks and one periodic Task
  - Another major difference of Enc/Dec Links from other links are they support single Output Queue
    - Use of External merge link is NOT required here
    - Send out the output data in the single out queue even for multi-resolution scenarios reduces the application complexity.
    - Single out Queue helps to have the same application code works across various multi-resolution scenarios

ENC/DEC Link – Threading Model (cont.)

- Enc Link Task:
  - Responsible for command/data processing events
  - A single task which reads the input data
  - Create the multi-channel process list and submit the same to the IVA specific process Queue
  - Send out/free the output buffer/frames
- IVA Processing Tasks:
  - One IVA Processing Task per IVA-HD for Encoder Links
  - Reads the process list from its process Queue
  - Populates the multi-codec ProcessParamsList and ProcessParams by populating the arguments such as inArgs, outArgs, inBuffs & outBufs for each channels
  - Invokes the multi-codec .process call
  - And finally put the processed list in to its process Done Queue once the codec complete its process call
- Periodic Clock Object:
  - This a BIOS clock object runs periodically and post the “GET_PROCESSED_DATA “ event
- Task priority:
  - All these tasks runs concurrently
  - IVA Processing Tasks has the highest priority followed by Enc Link Task & Clock Object
  - By making the IVA processing task as the highest priority ones ensures the HDVICP usage to the maximum

ENC/DEC Link – Data Flow

- Step1: Enc Link Task:
  - Data processing kick start once the Enc link task receives the data event “SYSTEM_CMD_NEW_DATA”
  - Reads the input data from the input FULL Queue
  - Put the frames in Enc Link channel specific queue
  - Checks for the output buffer availability
  - If output buffer available de-queue the input frame from channel specific queue
  - De-queue an output buffers from the common output buffer pool
  - Create the channel process list (a container holds the input frame & output buffer)
  - Create multichannel process list and put into the appropriate IVA process queue.
  - An IVA channel Map applied from the application context will decides the Enc link to put the process list to which IVA process queue.
- Step2: IVA process Task:
  - De-queue the multichannel process list from its input process queue
  - Populates the multi-codec ProcessParamsList and ProcessParams by populating the arguments such as inArgs, outArgs, inBuffs & outBufs for each channels
  - Call the XDM multi codec process call and wait (XDM .process is a blocking call)
  - Put the process list into the common process done Queue once the encode process complete. Process done queue is common across all the 3 IVAs

ENC/DEC Link – Data Flow (cont.)

- Step3: Periodic Clock Object:
  - This periodic clock object post the data event ENC_LINK_CMD_GET_PROCESSED_DATA to the Enc Link Task
- Step4: Enc Link Task:
  - On receiving the ENC_LINK_CMD_GET_PROCESSED_DATA message, its De-queue the process list from its process Done queue
  - Send out the output buffer to the next link, by putting them into the outputs Full Queue
  - Free the input frames by putting the same in the input Empty Queue

ENC/DEC Link – Data Flow (cont.)

- A few other notable points
  - The Link architecture & data flow of the decoder link is exactly same as encoder link, hence NOT discussed separately
  - Each IVA will have to process from a MAX of TWO process Queues when both encoder & decoder channels are mapped to the same IVA
  - Enc/Dec links can work in Tiled as well as Non Tiled format
  - Enc/Dec link uses the Framework Component (FC) iresman API to acquire and release the IVA-HDs
  - Data structures for multi-codec process APIs as below

```
typedef struct {
  IVIDENC2_Handle handle;
  IVIDEO2_BufDesc *inBufs;
  XDM2_BufDesc *outBufs;
  IVIDENC2_InArgs *inArgs;
  IVIDENC2_OutArgs *outArgs;
} IH264ENC_ProcessParams;
```

```
typedef struct {
  XDAS_Int32 numEntries ;
  IH264ENC_ProcessParams processParams
                [IH264ENC_MAX_LENGTH_PROCESS_LIST];
} IH264ENC_ProcessParamsList ;
```

ENC/DEC Link – IVA Channel Map

IVA allocation Table:

- This help the app to allocate/distributes different channels to different IVA-HDs as per his wish
- This a table that can be set from Application
- Enable to use the single videoM3 binaries for different apps & various data flows
- Reduce the Ducati side complexity & give flexibility to app integrator
- Efficient mapping reduces the codec/data reload overheads
- Data structure of IVA channel map & IVA allocation table as below

```
typedef struct SystemVideo_Ivahd2ChMap {
      UInt32 EncNumCh;
      /**< Number of Encoder channels */
      UInt32 EncChList[SYSTEMVIDEO_MAX_IVACH];
      /**< Encoder channel list */
      UInt32 DecNumCh;
      /**< Number of Decoder channels */
      UInt32 DecChList[SYSTEMVIDEO_MAX_IVACH];
      /**< Decoder channel list */
} SystemVideo_Ivahd2ChMap;

typedef struct SystemVideo_Ivahd2ChMap_Tbl {
      UInt32 isPopulated;
      /**< Flag to verify if the table is populated */
      SystemVideo_Ivahd2ChMap ivaMap[SYSTEMVIDEO_MAX_NUMHDVICP];
      /**< Structure to assign the video channles to all 3 IVA-HDs */
} SystemVideo_Ivahd2ChMap_Tbl;
```

MCFW

VCAP

VENC

VDEC

VDIS

Basic Fuctions
1. params_init()
2. Init()
3. deInit()
4. start()
5. Stop()

Helper Fuctions are module specific
enableChn()
disableChn()
changeBitRate()
switchMosaic()
etc

VSYS

Basic Fuctions
1. create()
2. delete()

It works based On usecase Id and knows every thing about usecase

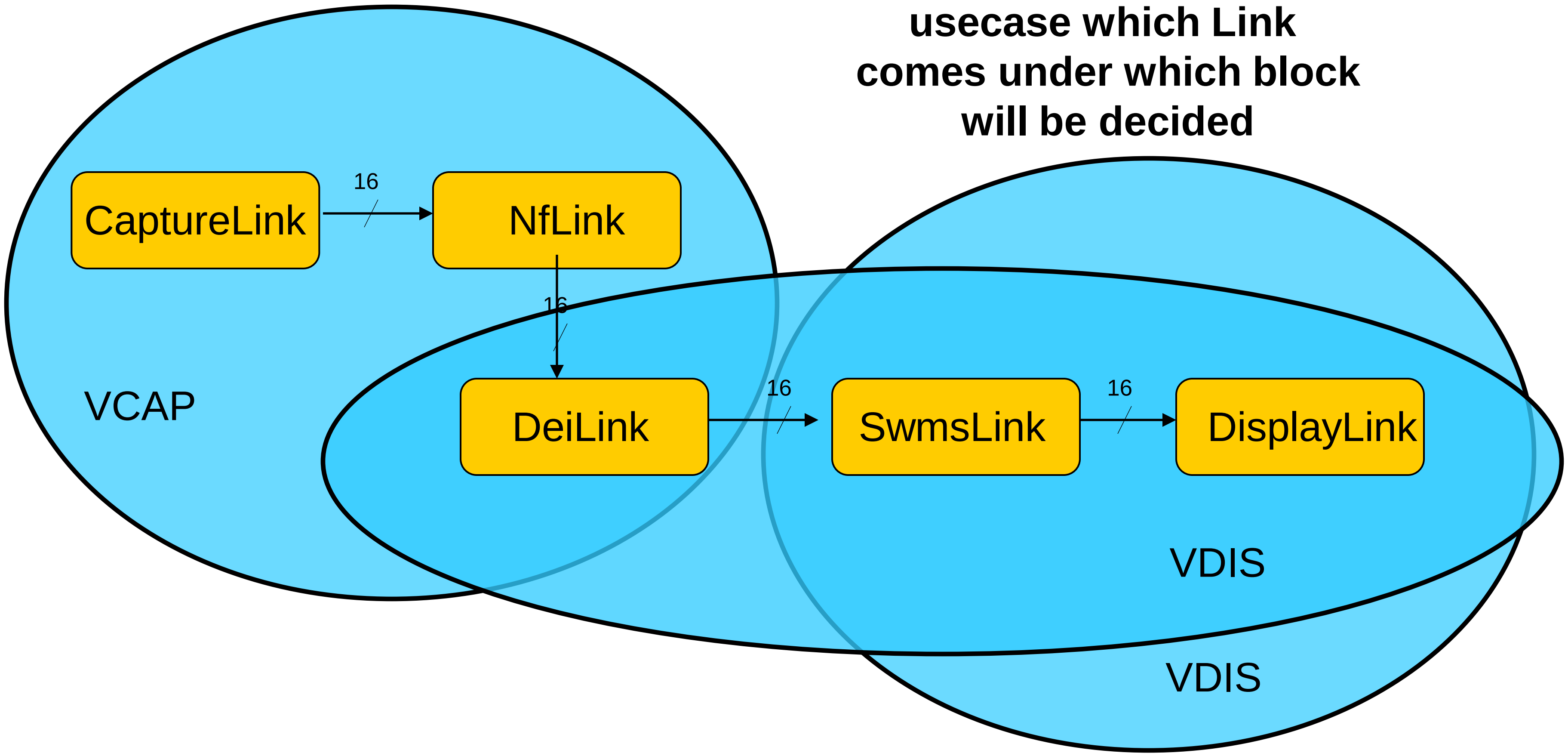
Do you have a your own usecase? Dig down Vsys_create() in existing demo and you will find what to modify and where.

MCFW continued…

- A Multi-Channel video system consists of four sub-systems as listed below
  - Capture – This will capture multi-channels from input video ports, optionally noise filter and/or de-interlacer and/or chroma downsample based on the input source
  - Display – This will take input from capture and decode sub-system, and show them multiple channels in different user-defined mosaic combinations on multiple display devices.
  - Encode – This will take input from capture and encode the video including “sub-stream” encode and give the encode bitstream to the user
  - Decode – This will take input bitstreams for multi-channels from user and provide as input to the display subsystem
- Essentially, MCFW provides common API that can be used to program Links. MCFW has been designed considering this scenario.

McFW and Links

Based on the usecase which Link comes under which block will be decided



Code walk through

Questions?

Thank you