Riviera Presentation

Vincent Oberle
Riviera Development Unit Team
Wireless Terminal BU
v-oberle@ti.com

WTBU Chipset Department
TI Proprietary information – Internal Data

Before getting started...

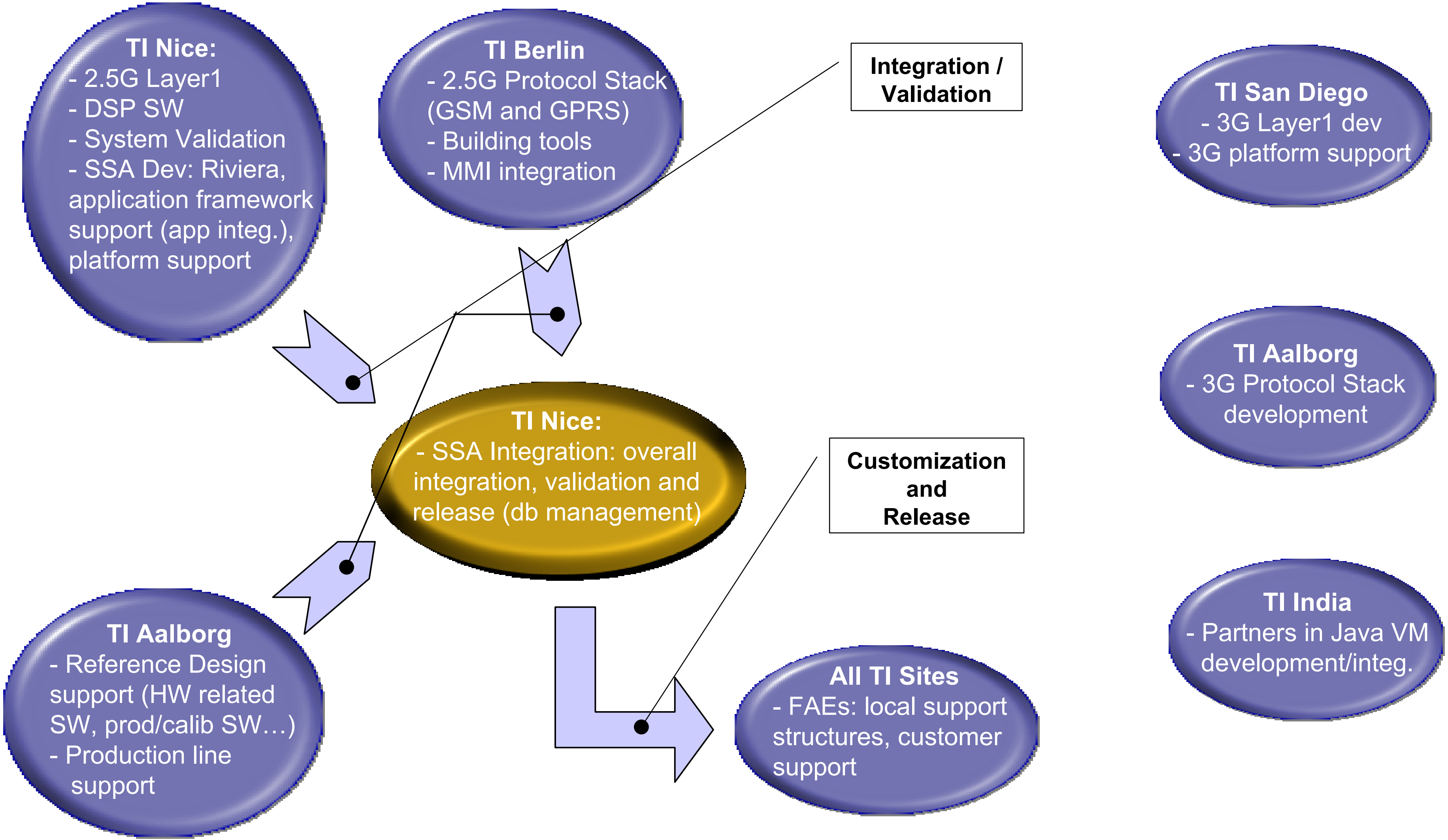
TI SW Activities for 2.5G

1/ Worldwide activities

2/ Nice Activities / SSA

TI SW Activities and Development Sites

2.5G Chipset



TI Nice:
- 2.5G Layer1
- DSP SW
- System Validation
- SSA Dev: Riviera, application framework support (app integ.), platform support

TI Berlin
- 2.5G Protocol Stack (GSM and GPRS)
- Building tools
- MMI integration

Integration / Validation

TI San Diego
- 3G Layer1 dev
- 3G platform support

TI Nice:
- SSA Integration: overall integration, validation and release (db management)

TI Aalborg
- 3G Protocol Stack development

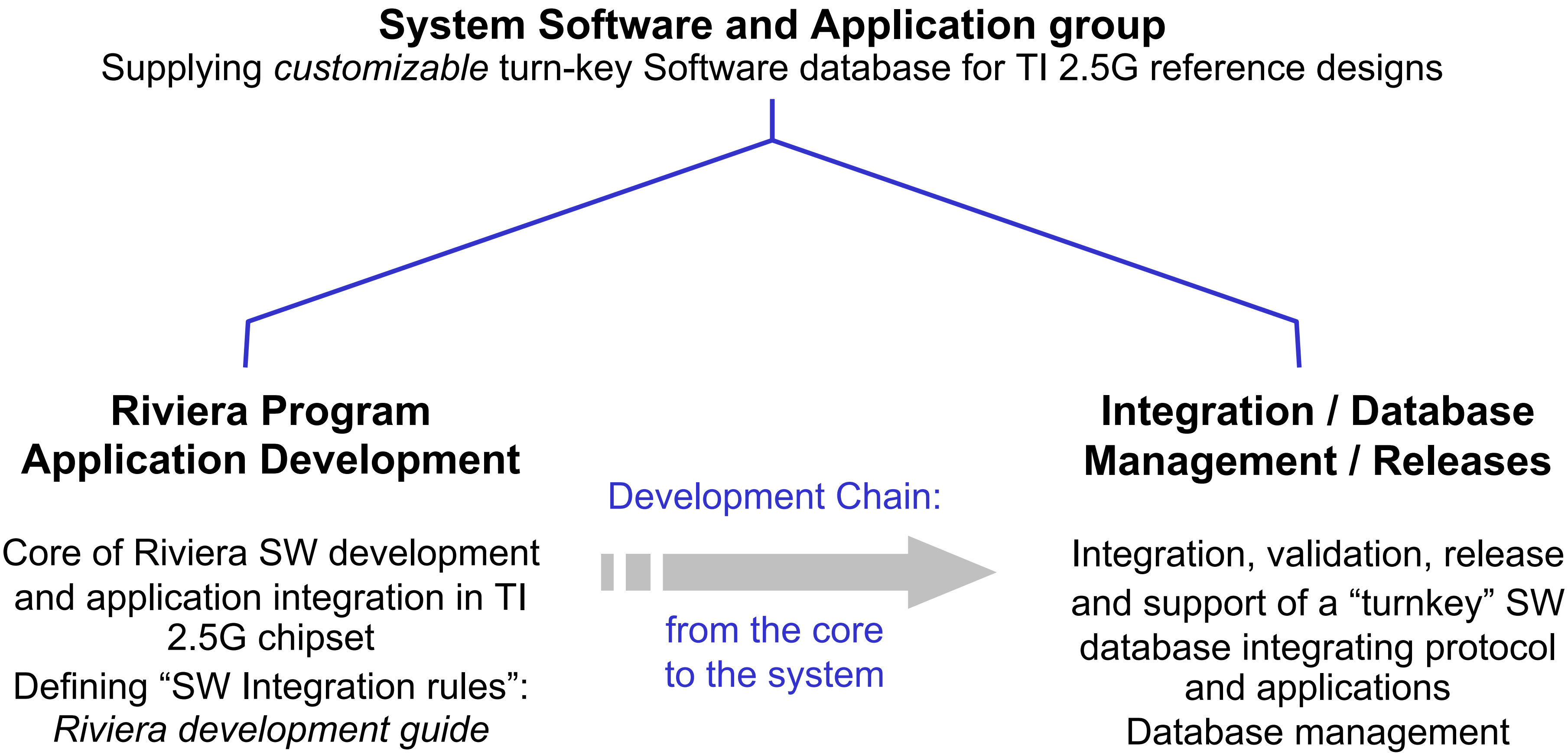
Customization and Release

TI Aalborg
- Reference Design support (HW related SW, prod/calib SW…)
- Production line support

All TI Sites
- FAEs: local support structures, customer support

TI India
- Partners in Java VM development/integ.

System Software and Application Group



System Software and Application group

Supplying *customizable* turn-key Software database for TI 2.5G reference designs

Riviera Program Application Development

Core of Riviera SW development and application integration in TI 2.5G chipset

Defining “SW Integration rules”: *Riviera development guide*

Development Chain:

from the core to the system

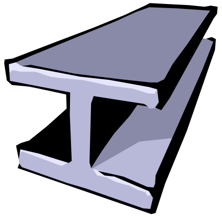
Integration / Database Management / Releases

Integration, validation, release and support of a “turnkey” SW database integrating protocol and applications

Database management

Today…

- Riviera Overview
- What is a SW environment



- Main concepts of Riviera Environment
- A few Riviera SW entities overview

- Riviera Coding Guidelines

- Riviera Integration with other SW environment

Riviera Initiative Presentation

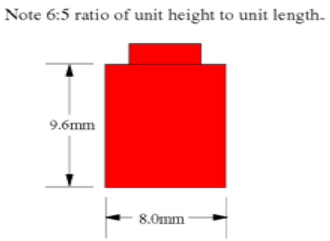
WTBU Chipset Department
TI Proprietary information – Internal Data

What triggered RIVIERA

- **Background:**
  More SW and applications are running on wireless terminals:
  - Increased complexity
  - SW is shared by more and more Development Teams/Companies
  - SW is key to Differentiate the Final Device
  - SW has severe impact on time to market
- **RIVIERA Objective:**
  Be able to easily develop, add new functionalities and applications on top of TI wireless solutions
  - Provide a complete SW Package, ready for FTA
  - Improve Time to Market, while Reducing Risk
  - Provide Customers with Easy Access to Differentiation
    - Customer can easily configure the SW package
    - Customer can focus development on key differentiator SW

Riviera Offer : an Enabling Technology

- A **SW Environment** (Riviera SW environment)
  - Runs on standard GSM/GPRS solution
  - Make any SW development easy to develop, to integrate and to customize on TI wireless real-time systems.
- A **SW Development Tool** (Riviera Tool) running on a single PC, emulating the GSM/GPRS target
- A **SW Database** (Riviera compliant SWs)
  - Includes most of the basic SWs needed for a GSM/GPRS solution
  - Regular releases , validated with GSM/GPRS PS are provided.
  - Very easy to customize, add custom SWs …



Riviera SW Environment Overview (1)

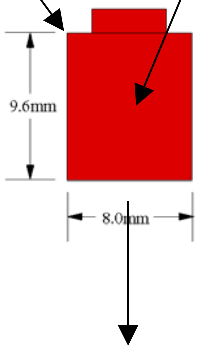
The environment defines what is outside the box and the shape of the box

The developers Define what is inside the box

Riviera SW Environment

=

- C code Library (Core SW)
- Methodology:
  - – **Development** methodology
  - – **Coding** rules / guidelines
  - – **Integration** methodology
  - – **Validation** methodology
  - – **Customization** method



The integrator Assembles the boxes to make a system

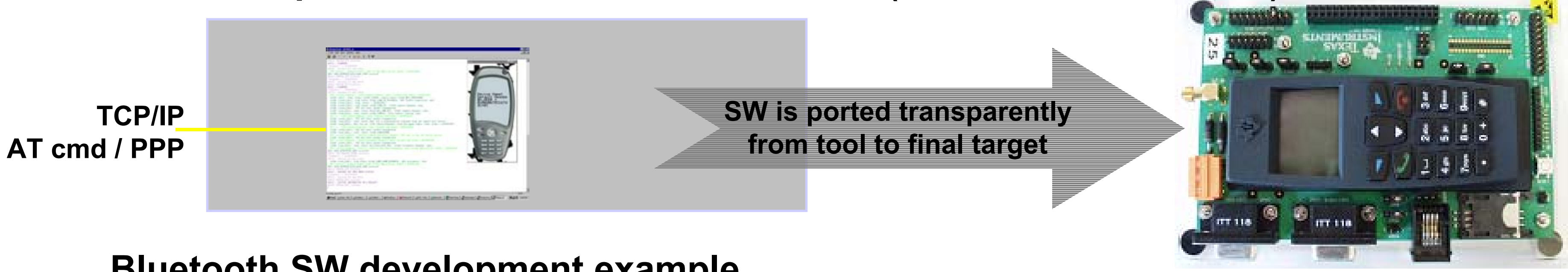
Riviera SW Environment Overview (2)

- **Easy to program:**
  - No need of wireless knowledge (abstraction layer)
  - Independant on HW roadmap
- **Safe:**
  - Modem resource are protected (memory available, real time constraints)
  - Usual developers do not have access to critical parts of the SW
- **Modular:**
  - SW components can easily be added / removed / changed, dynamically, using only resources when activated
  - Easy to integrate a new SWE and debug
- **Cost Efficient:**
  - SW environment has a small footprint (15 kB Flash, 5 kB RAM)
  - Optimal memory use in term of RAM and FLASH
  - Data handling optimisation (zero copy mechanism)

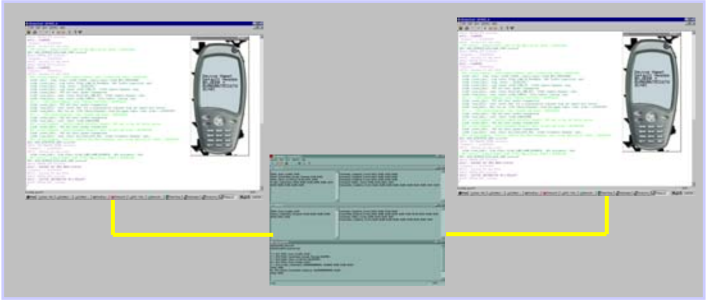
Riviera SW Development Tool (-Set)

SW development when no external connection is needed

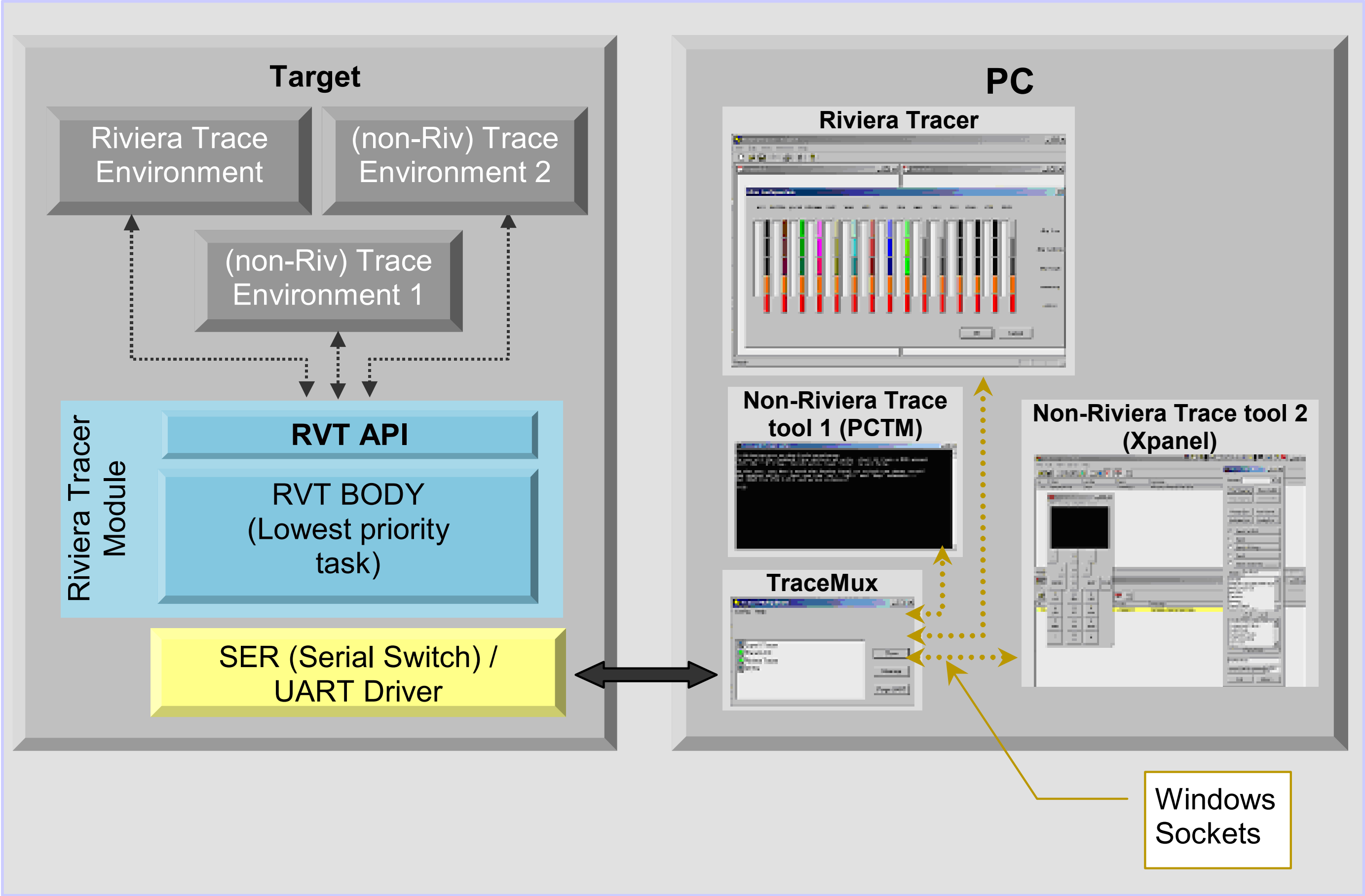
SW development with a Network connection (GSM/GPRS or TCP/IP)



Bluetooth SW development example



Riviera Trace Multiplexer / Riviera Tracer



Target

Riviera Trace Environment

(non-Riv) Trace Environment 2

(non-Riv) Trace Environment 1

Riviera Tracer Module

RVT API

RVT BODY (Lowest priority task)

SER (Serial Switch) / UART Driver

PC

Riviera Tracer

Non-Riviera Trace tool 1 (PCTM)

Non-Riviera Trace tool 2 (Xpanel)

TraceMux

Windows Sockets

Trace Multiplexer: Serial Mux / Unmux on PC and target for traces and other (TCP/IP, AT…)

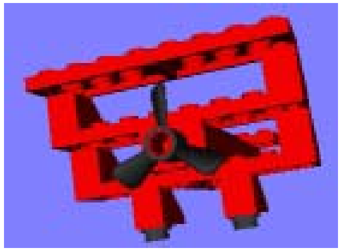
Allows to connect as many flows as required

Allows to use legacy / cust. application on PC (socket instead of COM: low rework)

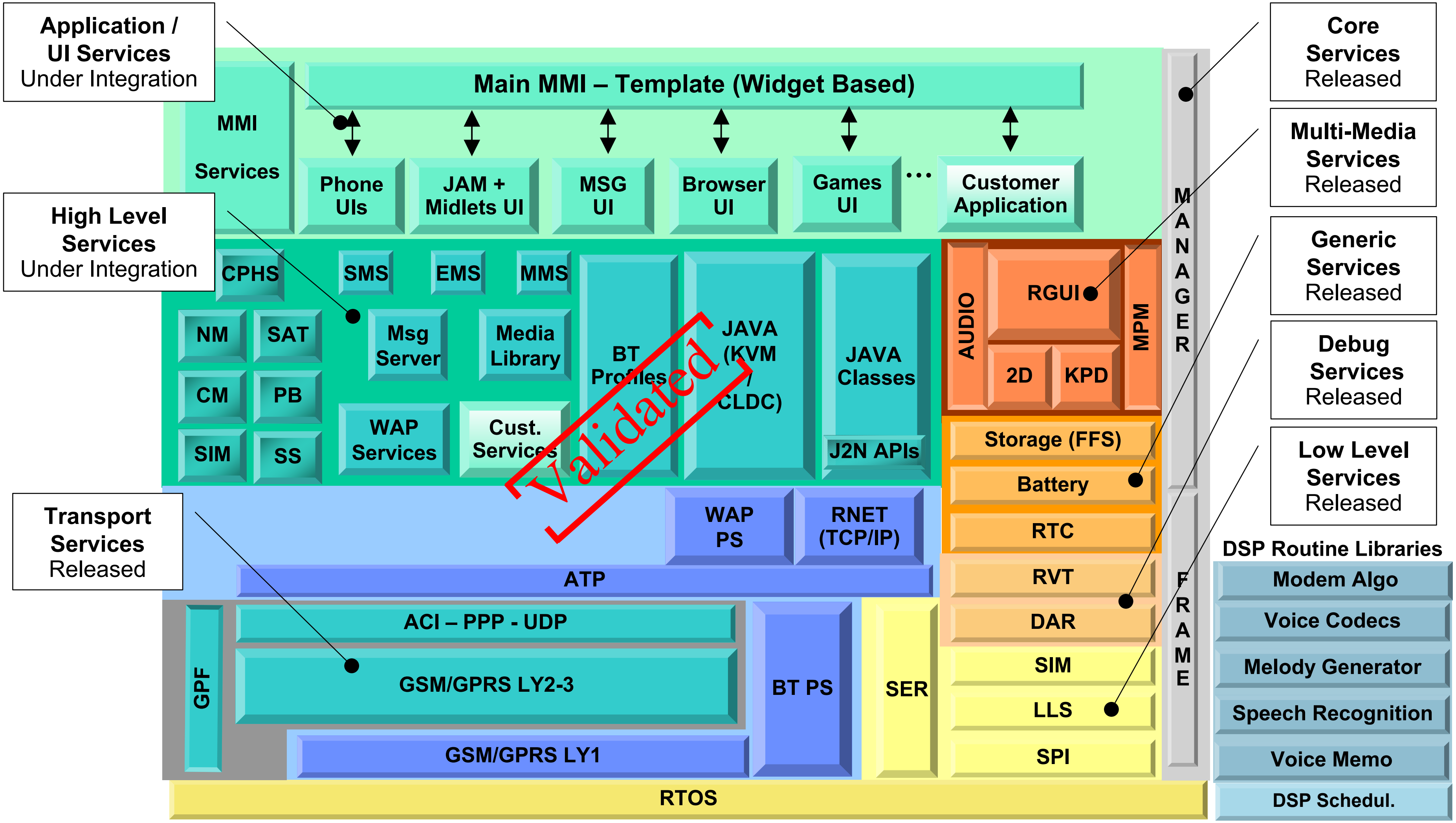
Riviera Tracer: Displays, filter and saves log messages coming from Riviera entities and from Layer1.

Riviera Database

- A **SW Database** (collection Riviera compliant SWs)
  - Easier for the developer of a SW Entity:
    - Service based APIs: Developers know to which bricks it should interact
  - Easier for the integrators of the final product:
    - Bricks interfacing and inter-dependencies has been checked. No intrusion of other native SW.
    - Integrator can have Riviera Environment along with other environment



Example of Riviera SW Database



Riviera Database: Products and Platforms

- **Riviera Database vs. Product Definition and Platform**
  - Transparent:
    - Ported and validated on every TI wireless platform (reference design)
    - Basic Support of new Platform within 2 months
  - Modular: any application can be added or removed depending on product definition
  - Optimal: all application are optimized on TI Chipset, application consumes resources only when running.
  - Opened: SSA database brings the needed standard bricks to build a product, but is opened to customization:
    - Existing SW customization (Look and Feel, application customization, pick and choose)
    - Adding of new custom application, through ease of porting.
  - Vertical:
    - Targets low cost as well as middle range products
    - 1 Single SW database for all TI wireless platforms (Calypso, Locosto1/2/3, Calypso+, Perseus2…)

=> Easy and fast MIGRATION to new TI wireless platforms
(Provides a roadmap for integration and optimization)

=> Easy handling of several "feature based" product lines / segments
(Easy definition of product features)

Riviera Database: Products and Platforms

⇒ Scalable and Flexible product definition.

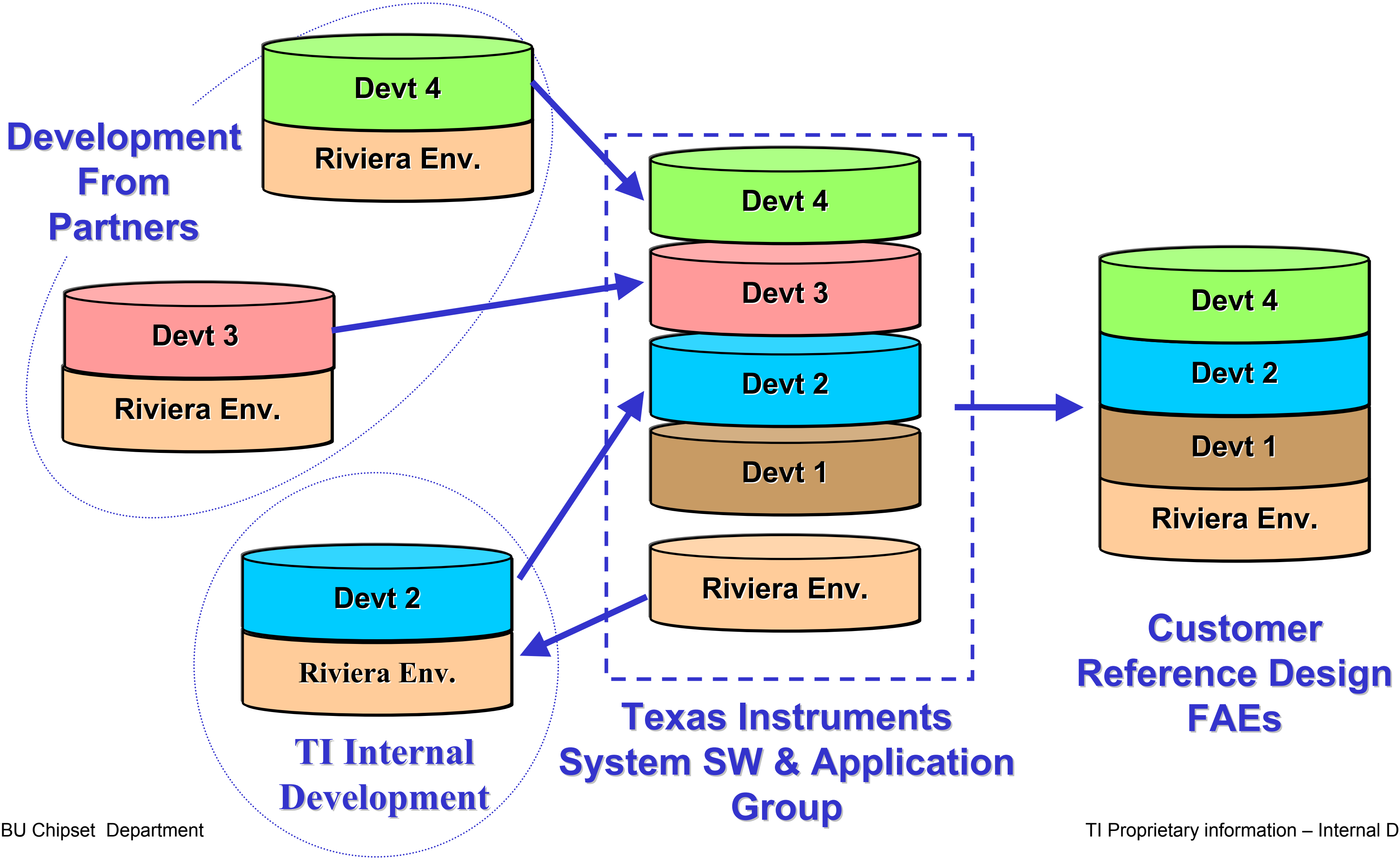
⇒ Maximum Synergy and reusability amongst platform,

⇒ Starting point available NOW (for all platforms!!).

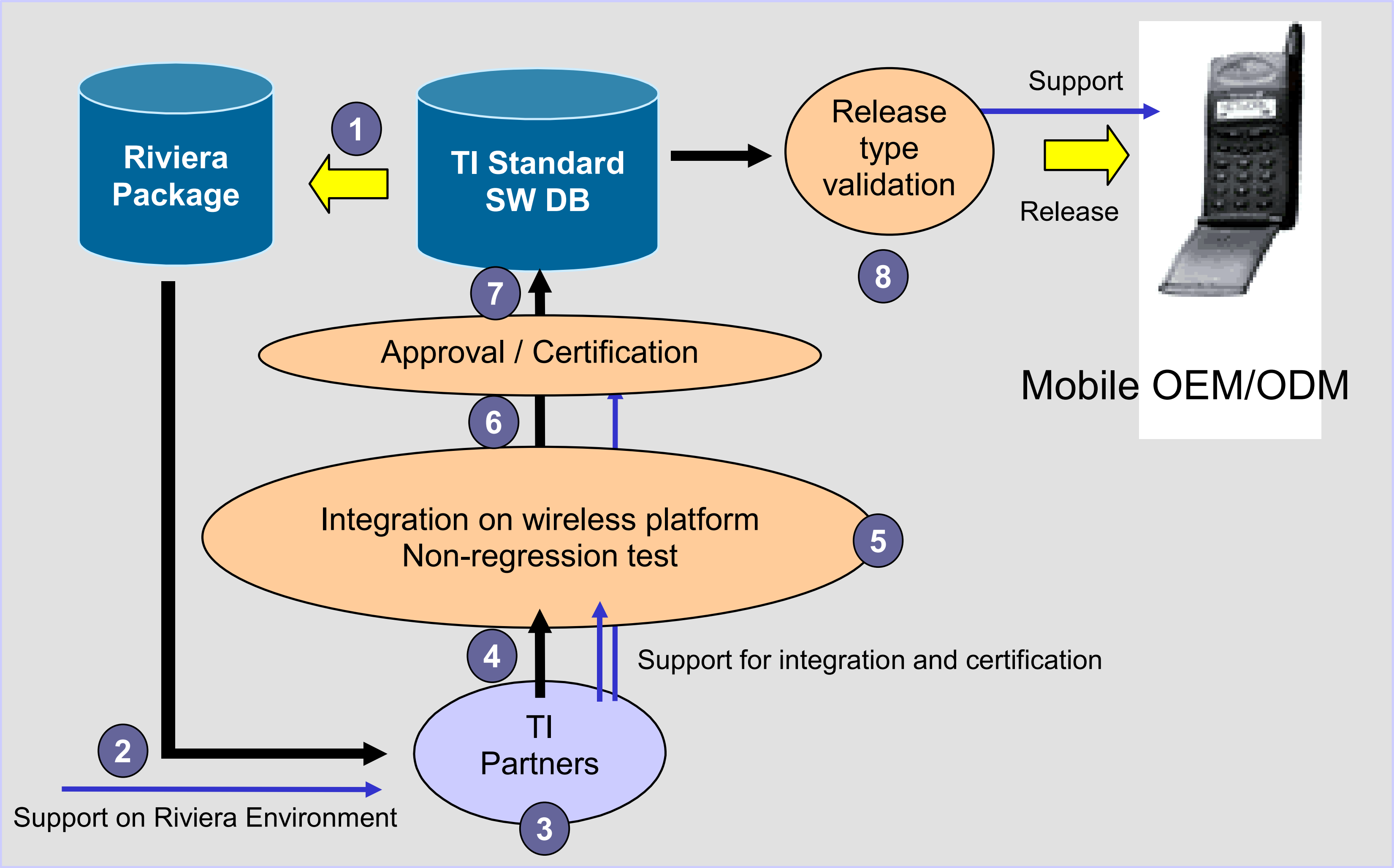
Riviera Release Flow

WTBU Chipset Department
TI Proprietary information – Internal Data

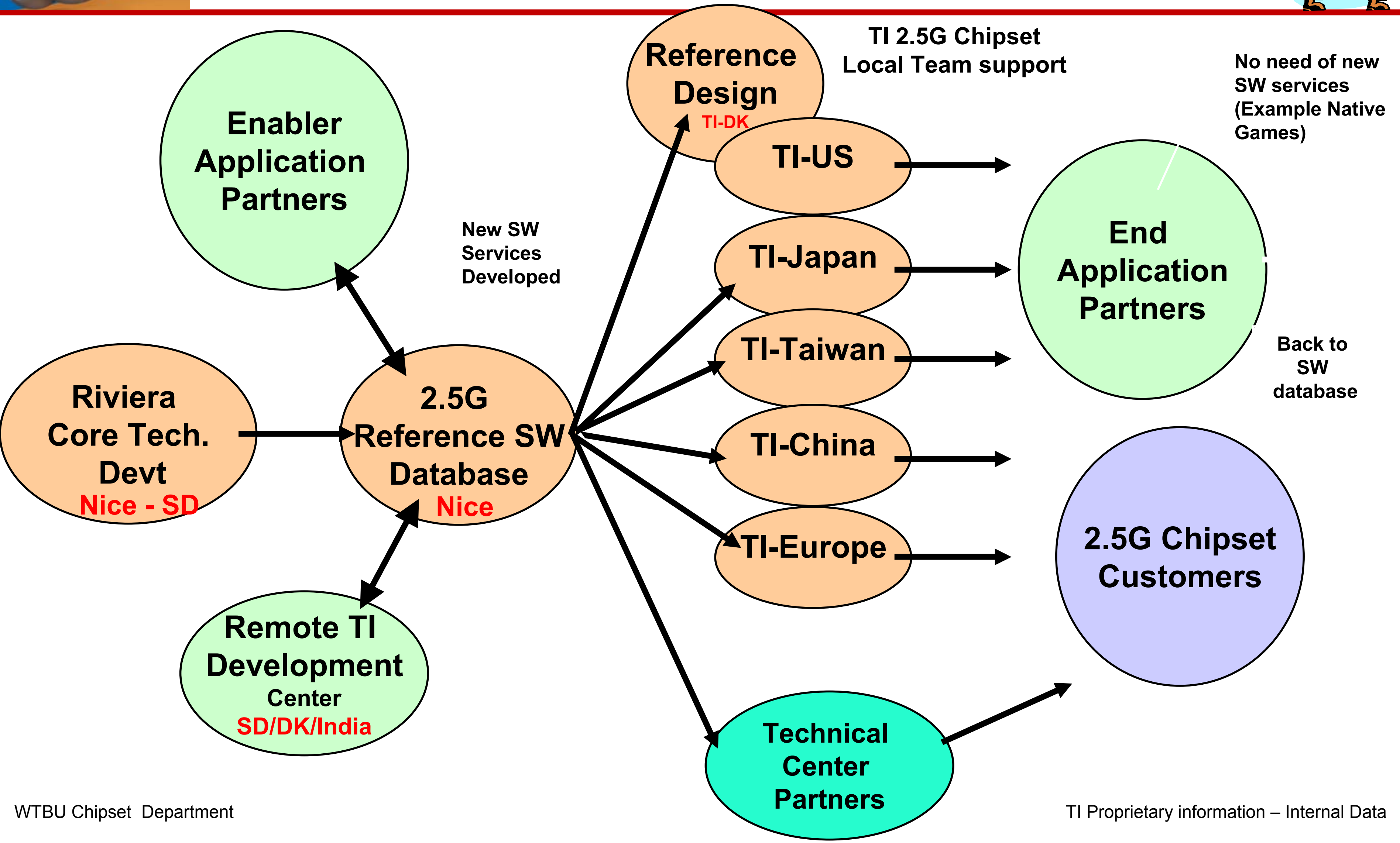
Riviera Development and Release Flow



Partnership Model for Standard SW

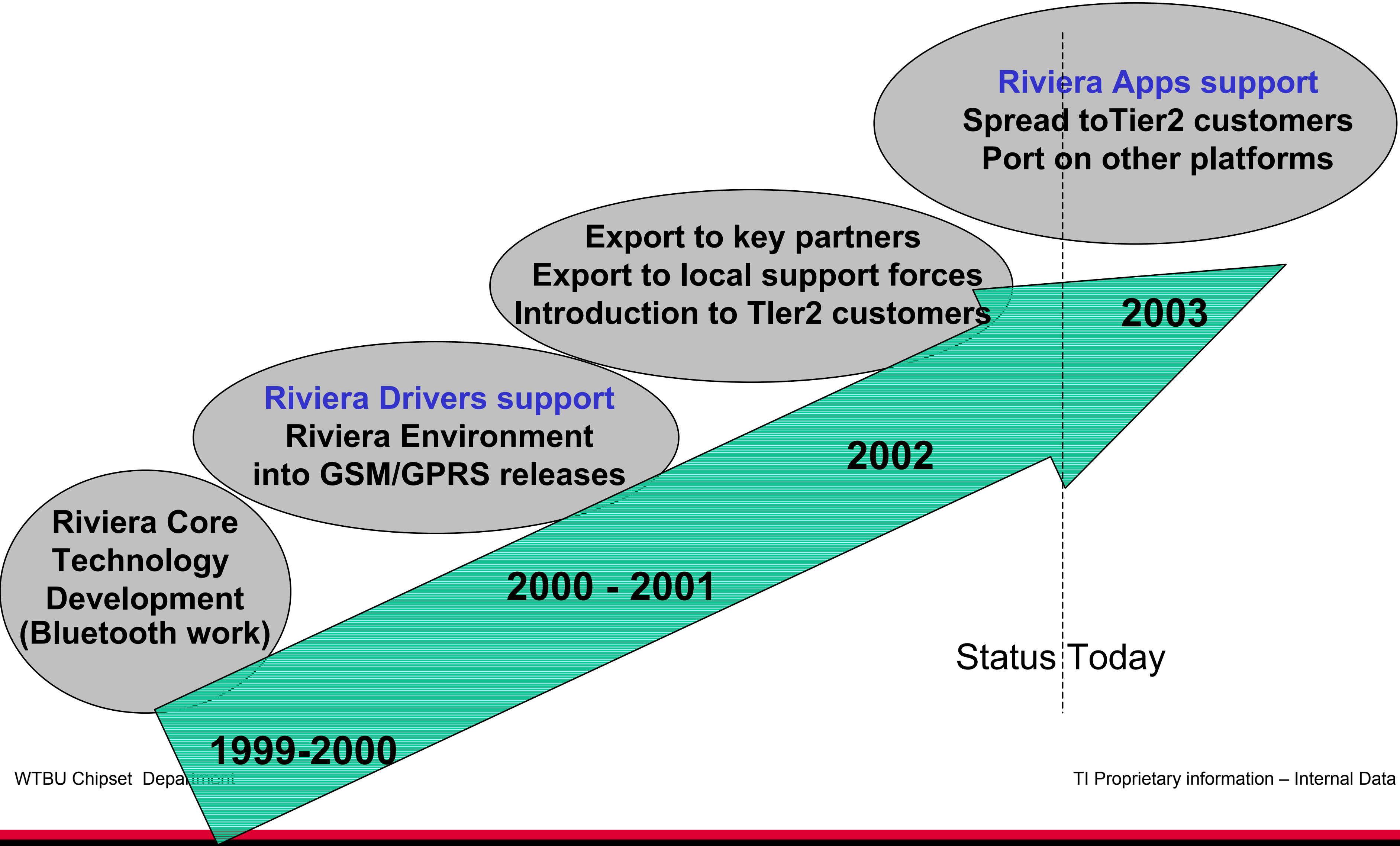


Riviera SW release and support Flow


Enabler Application Partners
New SW Services Developed
Riviera Core Tech. Devt
Nice - SD
2.5G Reference SW Database
Nice
Remote TI Development Center
SD/DK/India
Reference Design
TI-DK
TI 2.5G Chipset Local Team support
TI-US
TI-Japan
TI-Taiwan
TI-China
TI-Europe
Technical Center Partners
No need of new SW services (Example Native Games)
End Application Partners
Back to SW database
2.5G Chipset Customers

Riviera History

Riviera Initiative Steps



Riviera Core Technology Development (Bluetooth work)

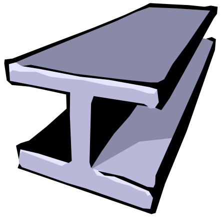
Riviera Drivers support
Riviera Environment
into GSM/GPRS releases

Export to key partners
Export to local support forces
Introduction to TIer2 customers

Riviera Apps support
Spread toTier2 customers
Port on other platforms

Today…

- Riviera Overview



- What is a SW environment

- Main concepts of Riviera Environment

- A few Riviera SW entities overview

- Riviera Coding Guidelines

- Riviera Integration with other SW environment

What is a SW environment? (1)

A SW environment allows to create and integrate a new SW block = SW Entity into a complex system without complex knowledge on this system.

How can we recover when SWE crashes?

How can we integrate the SWE and validate it in the full system?

Which rules to follow to code the SWE (generic types/coding guidelines/ file partitioning)?

New SWE to develop

How can we debug the SWE (trace system, memory scanning, buffer controls)?

How is the SWE started/ killed?

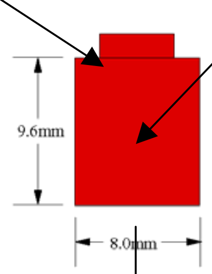
How does the SWE can have access to system resources (OS- drivers)?
Should not starve the rest of the system

How does the SWE communicate with others?

What is a SW environment? (2)

The environment defines what is outside the box and the shape of the box

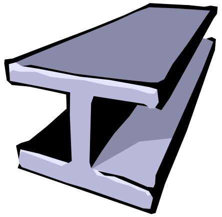
The developers define what is inside the box



The integrator assembles the boxes to make a system

Today…

- Riviera Overview
- What is a SW environment



- Main concepts of Riviera Environment
- A few Riviera SW entities overview

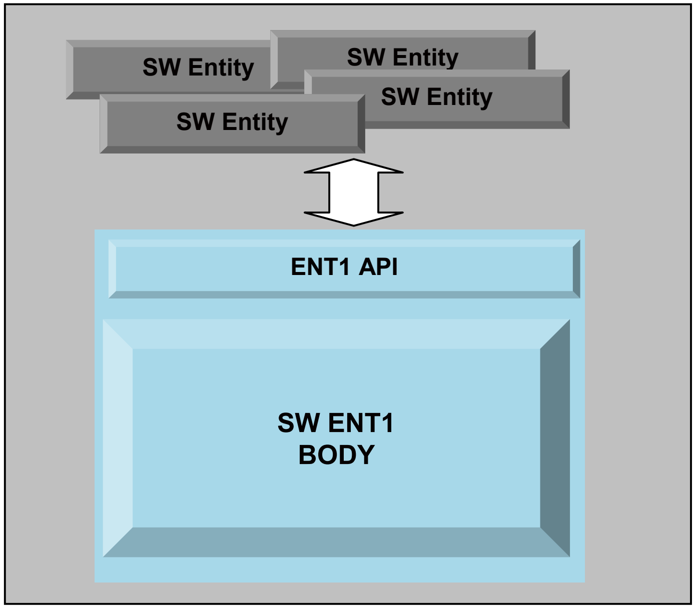
- Riviera Coding Guidelines
- Riviera Integration with other SW environment

SW entity Concept

WTBU Chipset Department
TI Proprietary information – Internal Data

SW Entity concept

- SW Entity = SW component, which provides a certain number of coherent services to the other SW components of the system.
- Software system = collection of SW entities.
- Exchanges performed between SW entities are exchange of service requests and service answers.

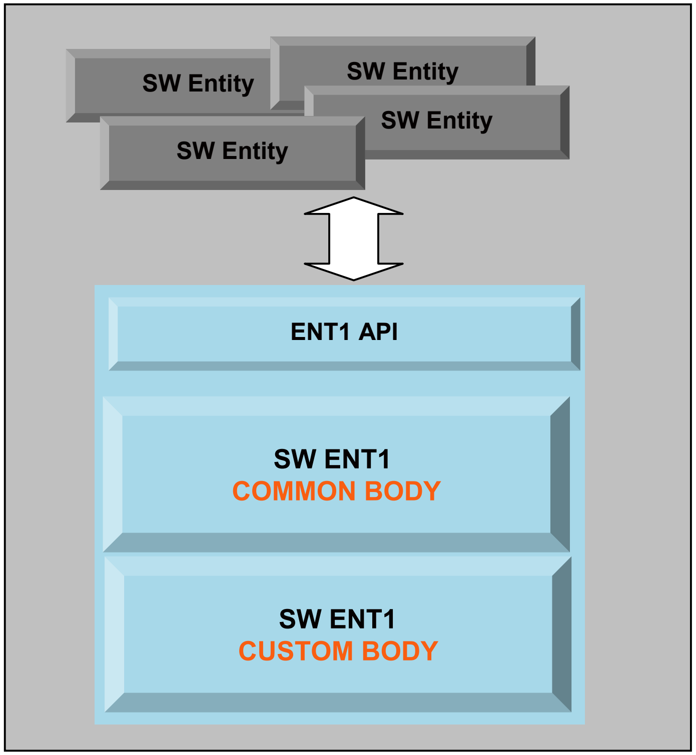


The SW is splitted in 2 distinct parts :

- API = SW allowing to access the services provided by the SW entity
- BODY = SW implementing the services

A Particular SWE : Riviera Drivers

- A driver is a SW entity which has a HW dependency

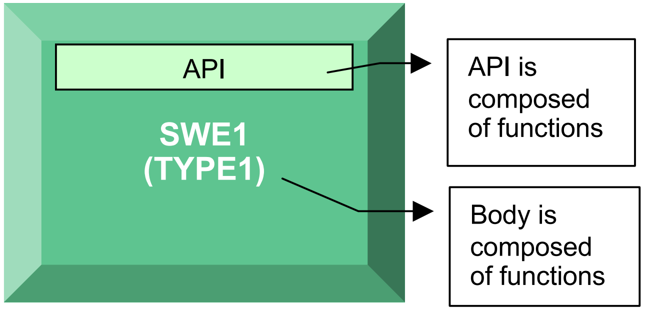


In most of the cases, the SW entity body of a driver is spitted into 2 pieces:

- A ‘common body’ : part of code independent from HW
- A ‘custom body’ : part that needs to be recoded when driver needs to be modified. Compilation flags allow support of different platforms
- API is kept unchanged in order to guarantee backward compatibility with the rest of the system

SW Entity types

Type 1 PASSIVE



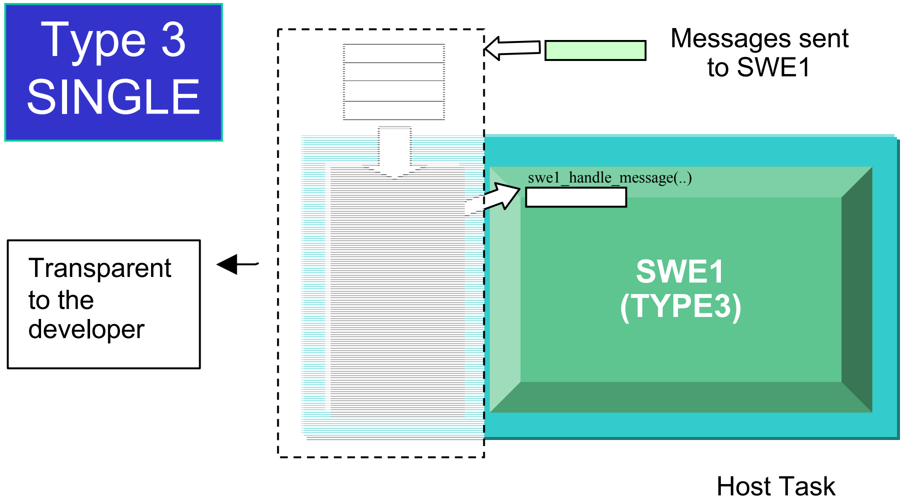
API

SWE1 (TYPE1)

API is composed of functions

Body is composed of functions

Type 3 SINGLE



Messages sent to SWE1

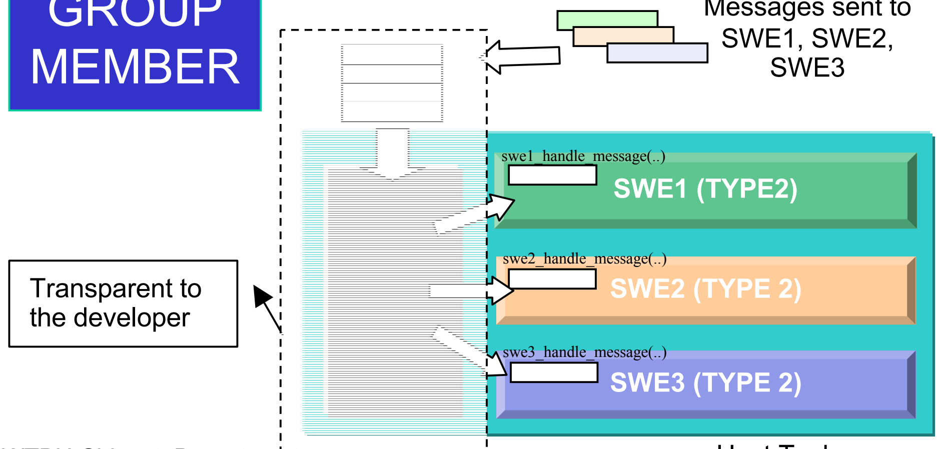
swe1_handle_message(..)

SWE1 (TYPE3)

Transparent to the developer

Host Task

Type 2 GROUP MEMBER



Messages sent to SWE1, SWE2, SWE3

swe1_handle_message(..)

SWE1 (TYPE2)

swe2_handle_message(..)

SWE2 (TYPE 2)

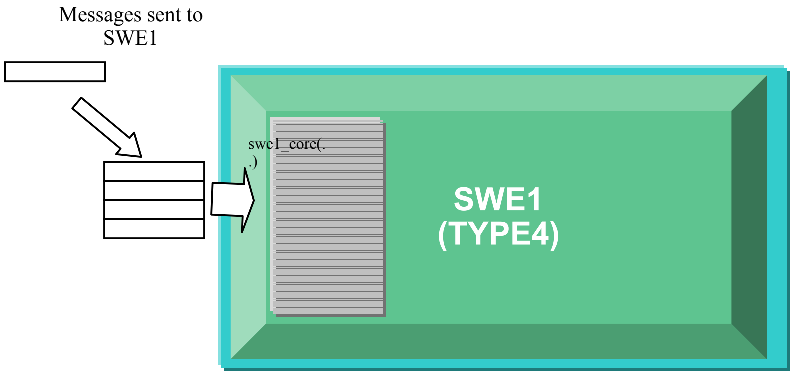
swe3_handle_message(..)

SWE3 (TYPE 2)

Transparent to the developer

Host Task

Type 4 SELF-MADE



Messages sent to SWE1

swe1_core(..)

SWE1 (TYPE4)

Creation / Destruction of SWE

- SW entity can be created and destroyed dynamically.
- Riviera Manager is in charge of the creation / destruction of the SW entities.

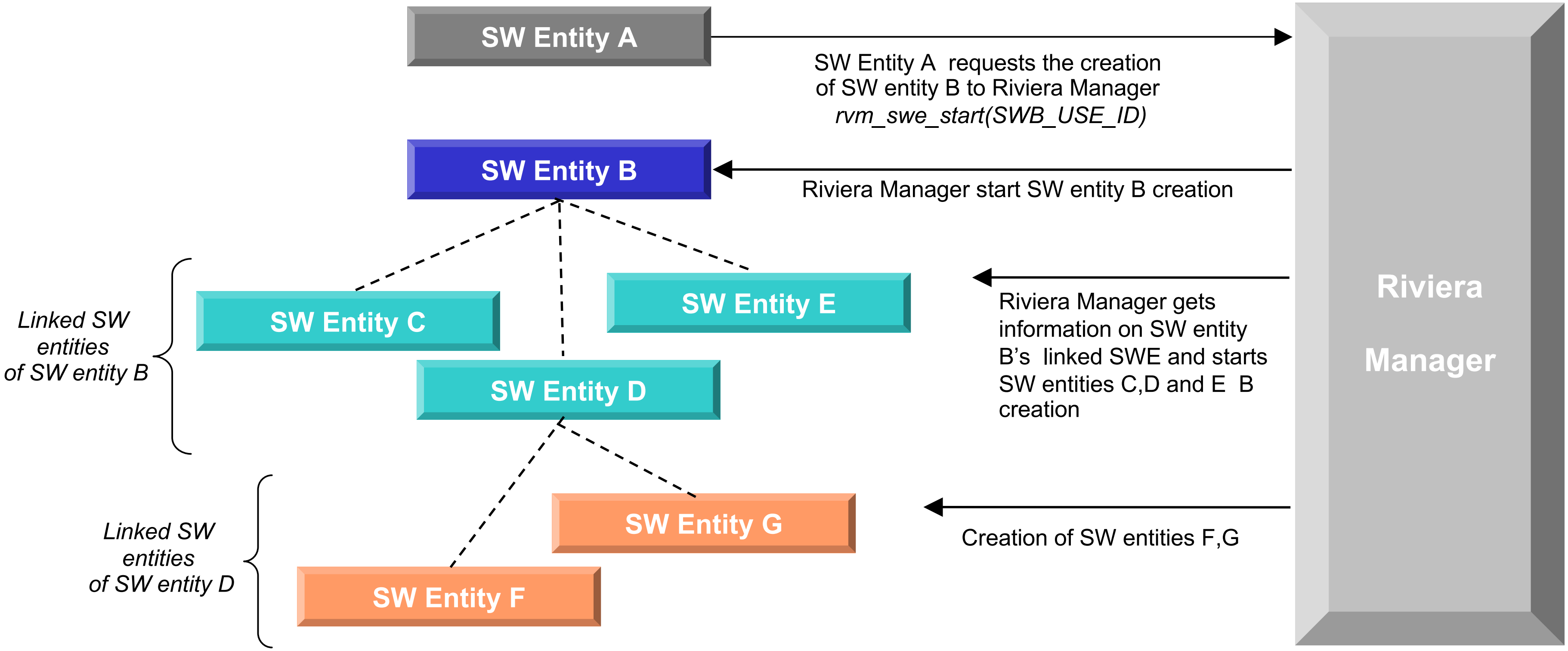
- To create a new SWE, just call:

  rvm_swe_start()
- and to destroy a SWE:

  rvm_swe_stop()

Linked SW Entities

- A SW entity can provide a list of other SW entities that need to be available for it.
- Recursive Activation / Destruction.


SW Entity A
SW Entity A requests the creation of SW entity B to Riviera Manager
rvm_swe_start(SWB_USE_ID)
SW Entity B
Riviera Manager start SW entity B creation
SW Entity E
SW Entity C
Linked SW entities of SW entity B
SW Entity D
Riviera Manager gets information on SW entity B's linked SWE and starts SW entities C,D and E B creation
Riviera
Manager
SW Entity G
Linked SW entities of SW entity D
Creation of SW entities F,G
SW Entity F

Generic functions: For who?

get_info	SWE1	SWE2	SWE3	SWE4
set_info	SWE1	SWE2	SWE3	SWE4
init	SWE1	SWE2	SWE3	SWE4
start	SWE1	SWE2	SWE3	
stop	SWE1	SWE2	SWE3	SWE4
kill	SWE1	SWE2	SWE3	SWE4
handle_message		SWE2	SWE3	
handle_timer		SWE2	SWE3	
core				SWE4

USE ID concept

- The USE ID is a unique static identifier of a SoftWare Entity.

- It is allocated during the registration process.

- It is used to identify a SWE from a static point a view:
  - To start it.
  - To stop it.
  - To retrieve information about it.
  - To send debug messages to a PC tool.

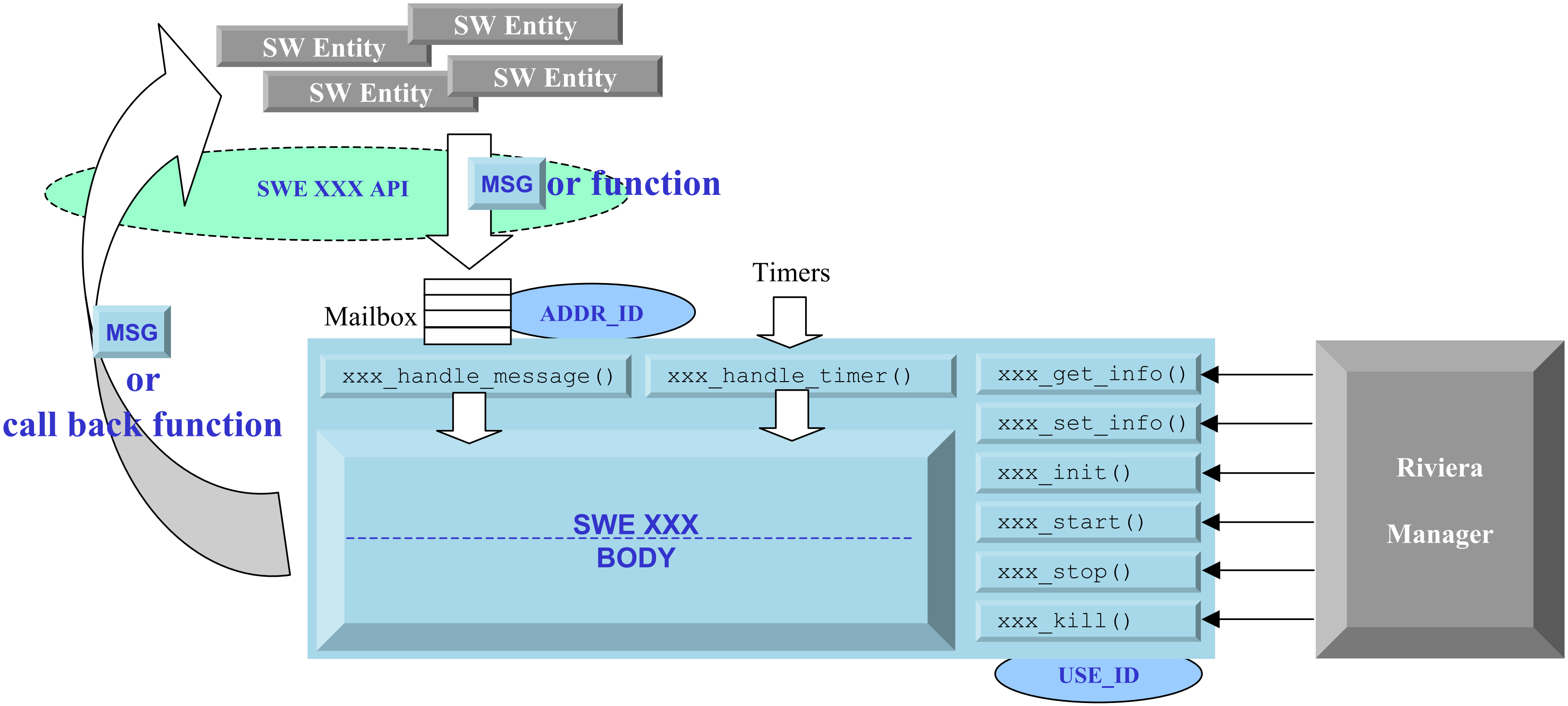
Address ID concepts

- The Address ID (ADDR_ID) is the unique path to a running SWE, it is allocated dynamically and might change when a SWE is started/stopped.

- It is allocated during the starting phase by RVM.
  - parameter of the …_set_info() function.
  - could be retrieved using the rvm_get_swe_information()

- It is used to identify a SWE from a dynamic point a view:
  - To send messages to it.

SW Entity concept summary



Inter SW entity Communication

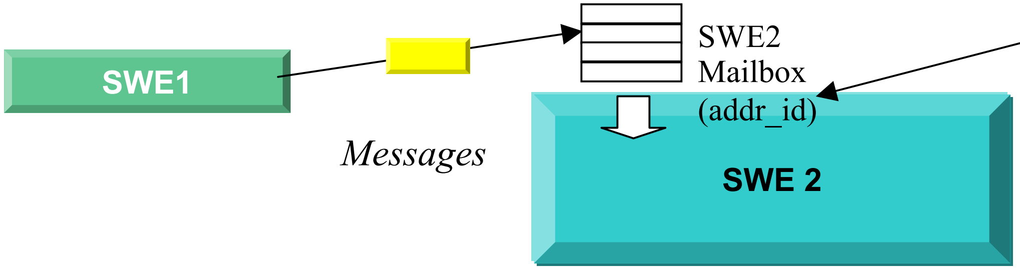
WTBU Chipset Department
TI Proprietary information – Internal Data

Inter SWE communication: Service request

SWE1 requests a service to SWE2

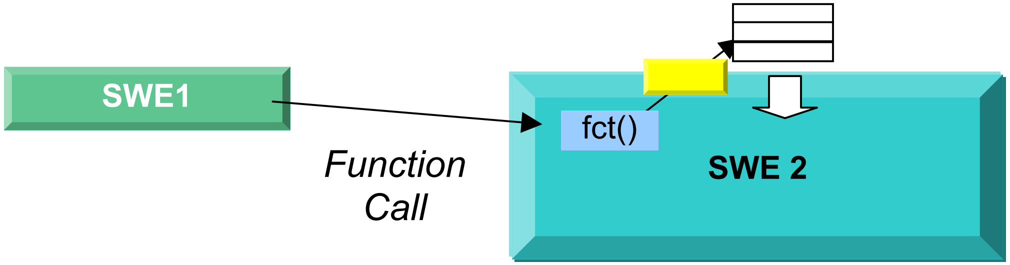
The way the service has to be requested depends on the SWE2 API.

- If SWE2 API is a set of messages: SWE1 should send service request messages into SWE2 mailbox.



The mailbox address is provided by RVM and is called the addr_id

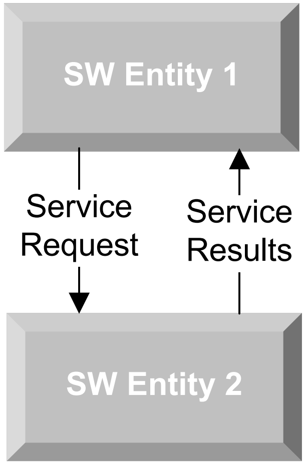
- If SWE2 API is a set of functions: SWE1 should call SWE2 API function



Inter SWE communication: Service answer

The 'Return-Path concept'

- Used by SWE1 (caller of the service) to indicate to SWE2 how it wants the answer of this service request back



- SWE 1 provides a return path to SWE 2. It is a C structure with 2 fields:
  - An address ID.
  - A callback function pointer.
- Address ID specified: the answer is sent as a message to the requester of the service.
  - Applicable in most of the cases
- Callback pointer is specified: the answer is sent as a parameter of the callback to the requester of the service.
  - Very useful in some cases, especially when the requester is not a Riviera Entity.

Memory handling

WTBU Chipset Department
TI Proprietary information – Internal Data

Memory Bank concept

- Efficient memory allocation on a complex system is crucial:
  - Memory implementation may differ from one target to another.
  - Many different SW entities use the same system memory.
  - Safe dynamic memory management is a MUST for a robust implementation.

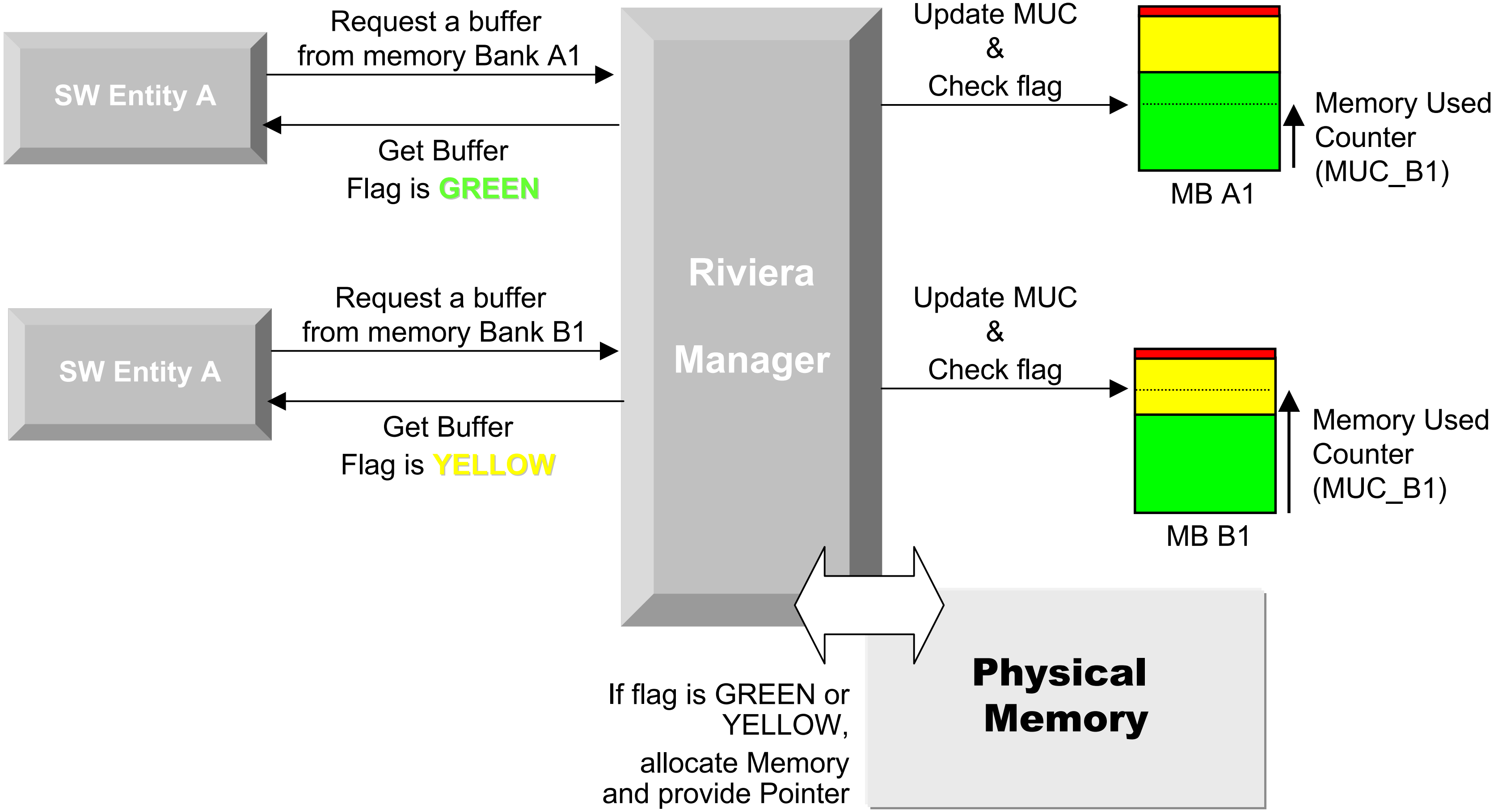
- A memory bank is a virtual amount of memory, allocated to a SW entity.
  - A memory bank is initialised with the maximum of memory that can be requested on the memory bank.
  - When a SW entity request memory, it indicates which memory bank the memory should be taken from. For example to get a buffer:

```
return_flag = rvf_get_buf(XXX_MB, size_of_buffer, &pointer_on_buffer);
```

  - To deallocate a buffer:

```
rvf_free_buf(pointer_on_buffer);
```

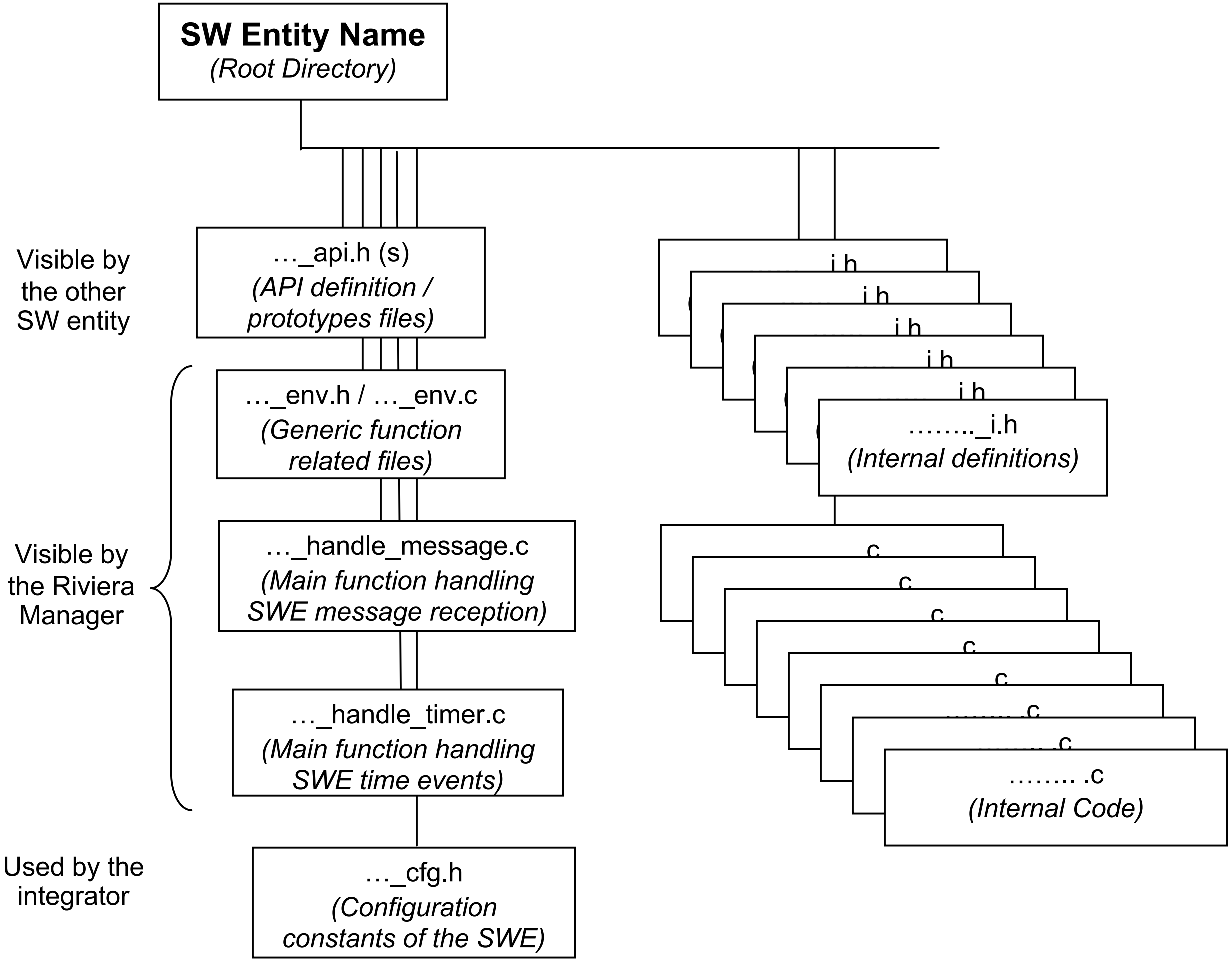
Memory Bank Color Flag


SW Entity A
Request a buffer from memory Bank A1
Get Buffer
Flag is GREEN
SW Entity A
Request a buffer from memory Bank B1
Get Buffer
Flag is YELLOW
Riviera Manager
Update MUC & Check flag
MB A1
Memory Used Counter (MUC_B1)
Update MUC & Check flag
MB B1
Memory Used Counter (MUC_B1)
Physical Memory
If flag is GREEN or YELLOW,
allocate Memory and provide Pointer

SW files structure

WTBU Chipset Department
TI Proprietary information – Internal Data

SW entity File structure



SW Entity Name
(Root Directory)

Visible by the other SW entity

…_api.h (s)
(API definition / prototypes files)

Visible by the Riviera Manager

…_env.h / …_env.c
(Generic function related files)

…_handle_message.c
(Main function handling SWE message reception)

…_handle_timer.c
(Main function handling SWE time events)

Used by the integrator

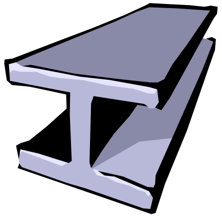
…_cfg.h
(Configuration constants of the SWE)

…….._i.h
(Internal definitions)

…….. .c
(Internal Code)

Today…

- Riviera Overview
- What is a SW environment



- Main concepts of Riviera Environment
- A few Riviera SW entities overview

- Riviera Coding Guidelines
- Riviera Integration with other SW environment

Riviera Manager - RVM

- RVM provides services related to SW entities management (creation / destruction / information…)
- RVM is created at initialization (‘heart’ of Riviera based system).
- API available depends of SWE type

- Main API functions are:
  - `rvm_swe_start(SW_ENTITY_USE_ID, return_path);`
  - `rvm_swe_stop(SW_ENTITY_USE_ID, return_path);`

Riviera Frame - RVF

- RVF provides services related to OS resource access
- RVF is created at initialization
- Communication:

`rvf_send_msg(addr_id,msg_p)`	all SWE TYPES
`rvf_read_mbox(…)`	TYPE 4
`rvf_send_event(…)`	TYPE 4
`rvf_lock_mutex(…)`	TYPE 3 and 4

- Memory Management:

`rvf_create_mb(…)`	all SWE TYPES
`rvf_get_buf(MB_ID,buf_size,&buf_p)`	all SWE TYPES
`rvf_free_buf(buf_p)`	all SWE TYPES

- Timers:

`rvf_delay(time)`	TYPE 3 and 4
`rvf_start_timer(…)`	TYPE 2,3 and 4

- Queue manipulation:
  - `rvf_enqueue(…) / rvf_dequeue(…) …`
- Debug:

`rvf_send_trace / rvf_dump_mem(…)`	all SWE TYPES

Flash File System - FFS

- FFS SW entity is providing services to handle file of permanent information (permanent storage in Flash).

- Main functions of the API are:
  - ffs_open(…)
  - ffs_close (…)
  - ffs_write (…)
  - ffs_read (…)
  - ffs_seek (…)
  - ffs_stat (…)
  - ffs_mkdir (…)
  - ffs_opendir (…)
  - …

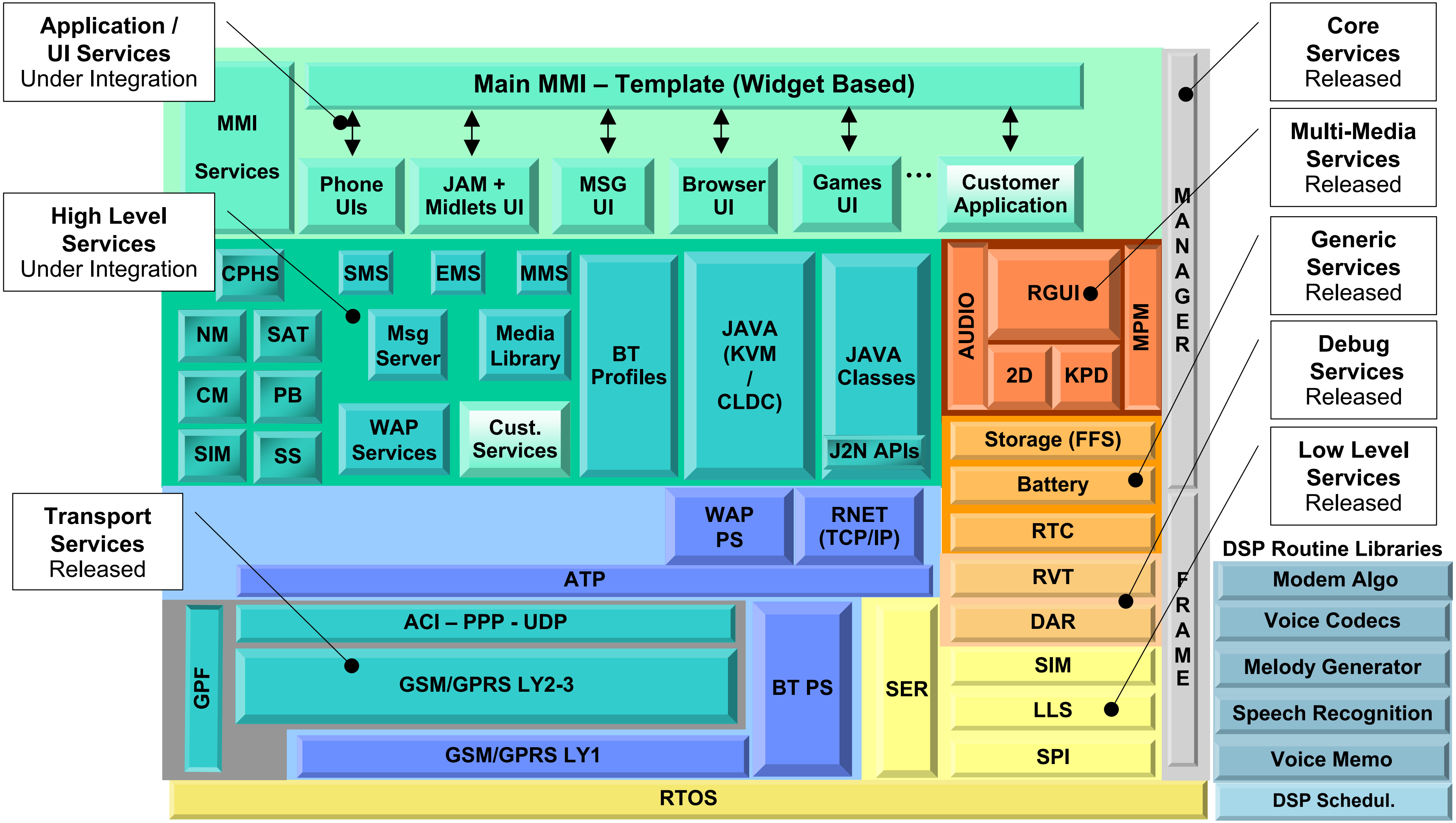
Audio - AUDIO

- AUDIO SW Entity is providing AUDIO services such as:
  - melody (multi-channel/instruments) / tones generation
  - voice memorization
  - Voice dialing / speech recognition
  - Audio speaker volume
- Main advantages:
  - High level API: easy to handle / easy to program
  - Handle directly FFS (long voice memo available with FFS chunked access)
  - Robust since add a layer of control and synchronization between the MMI and the low level (LY1 / DSP)
  - Transparent upgrade to other platforms (remove access to LY1 in future)

A few other SWEs

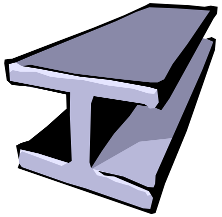
- RVT SWE (Riviera Tracer) provides a multiplexed access to a single UART link
- DAR SWE (Diagnose and Recovery) provides services to recover from scrach and get diagnose information on what happen just before the scrach
- Keypad SWE for keypad services
- R2D (Riviera 2D) for 2D graphic services
- RGUI (Riviera Widgets) for building MMI interfaces
- RTC (Real Time Clock) services
- ATP (Agnostic TransPort) : generic access to transport layers
- RNET (Riviera NETwork) : TCP/IP API, 3 configuration to enable step by step Software integration / validation.

Example of Riviera SW Database



Today…

- Riviera Overview
- What is a SW environment



- Main concepts of Riviera Environment
- A few Riviera SW entities overview

- Riviera Coding Guidelines
- Riviera Integration with other SW environment

Introduction

- In Riviera, next to the environment, the tools and the database, are the coding guidelines.
- Goal: Make the code of many different developers work together.

- Legend:
  - CR = CRITICAL: to be strictly followed
  - HR = HIGHLY RECOMMANDED: not critical but large influence on robustness of the SWE / amount of work needed to integrate the SWE
  - OP = OPTIONAL: it is recommended to follow this rule as a nice-to-have feature

Naming conventions (1)

- File naming

CR – Every file name of a SW entity must start with the SWE nickname.

Example: `rtc_process.c`

HR – Every SW entity should provide the following files:

- …`_api.h` Declarations needed to use the service of the SWE
- …`_env.h` Riviera Generic Functions declarations
- …`_env.c`, …`_handle_message.c`, …`_handle_timer.c`

  Coding of the Riviera Generic Functions
- …`_cfg.h` Constants that can be tuned by the integrator

OP – Other recommended file naming convention:

- Include filenames that are used only internally in the SWE should finish with the ‘_i’ extension.

Example: `rtc_messages_i.h`

Naming conventions (2)

- External Information

HR

  - All the information that is visible outside the SW entity should start with the SW entity nickname.

    Examples:

    - Extern functions: rvm_start_swe() and not start_swe(…)
    - Types: T_RVM_NAME and not T_NAME
    - Constants: #define RVF_MAX_TOTAL_MB (70)

- General naming convention

OP

  - Variable or function: lower case, underscore between words

    rvf_get_buf (…)

  - Constants: upper cases, underscore between words

    RVF_GREEN

  - Types: upper cases, underscore between words, start with ‘T_’
  - Pointers: end with “_p” for single indirection, “_pp” for double…

- Message Naming Convention

HR

  - Depending the kind of SW, several naming convention may be used for message naming.

Comments: Documentation Rules (1)

- Evident need of comments for:

HR

  - Using the SWE
  - Integrating the SWE
  - Maintaining the SWE

- Using formating rules

OP

  - Riviera uses Javadoc rules.
  - Rules are compatible with automatic documentation generation tools (= program creating automatically the documentation by looking on the source code).
    - Nice tool: Doxygen. Generates HTML, RTF and LaTeX format

Comments: Documentation Rules (2)

OP

Documentation blocks have to be identified with special tags:

```
/**
 * ... text ...
 */
```

- Content:
  - brief description (first sentence)
  - detailed description
  - tags of the form @tag describing parameters, return values, etc.
- Interesting tags:
  - @param
  - @return
  - @author
  - @version
  - @see
  - @deprecated
  - etc

Include Files

CR An include file shall contain only definitions, declarations, macros, function prototypes and conditional compilation statements.

- Every exported include file (such as …`api.h`) should start with:

CR

```
#ifndef __NAME_OF_THE_FILE_
#define __NAME_OF_THE_FILE_
```

and end with:

```
#endif
```

- The …`_api.h` files should not include any internal …`_i.h` files.

CR

Error Return

HR
- Every function should return an error indication.

OP
- Standard Riviera return-flag can be used.

The return type is T_RV_RET and can have following values:

– RV_OK	Function processed successfully
– RV_NOT_SUPPORTED	Requested process not supported
– RV_NOT_READY	Requested process cannot be processed now
– RV_MEMORY_ERR	A memory error occurred
– RV_INTERNAL_ERR	An internal error has occurred
– RV_INVALID_PARAMETER	A parameters is invalid

Standard Types and Libraries

- Standard Types

HR

  - The SW should not directly use the types of the compiler.
  - It should rather use the standard Riviera types.

    Example: (see general.h)

```
  typedef unsigned char   UINT8;
  typedef unsigned short  UINT16;
```

HR

Libraries Usage

  - The use of libraries like maths, strings…. should be limited as much as possible for portability and code size optimization.

- Special Functions use

CR

  - The following functions should not be used, especially in time critical part of the code:

```
sprintf(…)                mod(…)          div(…)
```

HR

  - The following function use is recommended:

```
memcpy(…)
```

State Machine Implementation

HR A state machine is code, which receives stimuli (messages in Riviera) and which reacts to the stimuli depending the current state.

One of the reaction can be the change of state.

HR Implementation:

- first check the state of the state machine
- then check the stimuli it received

```
handle_message (message) {
          switch (internal_state of the state_machine) {
          case STATE_1:
                    switch (message_id) {
                              case MESSAGE_1:break;
                              case MESSAGE_2: break;
                              default: break;
                    } break;
          case STATE_2:
                    switch (message_id) {
                              case MESSAGE_1: break;
                              case MESSAGE_2: break;
                              default: break;
                    } break;
          default: break;
          }
}
```

Miscellaneous

HR

- Avoid Global variables
  - Use the Global Variable Buffer

OP

- Braces {} and indentation
  - Line-up code by braces (braces is place on a new line) whenever a loop or conditional case is code.
  - Conditional code should be surrounded by braces even if it contains only one instruction.
  - Example:

```
if (return == HCI_ERROR)
{
          hci_error(error_parameters);
}
```

HR

- Avoid `if (x)` ➔ Replace by `if (x==NO_ERROR)`
- Avoid litteral numbers ➔ Use constant or preprocessor definitions
- Use the default label in all switch statements.
- Put a numeric value of a define or a macro definition between brackets.

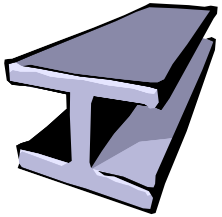
Summary: Riviera Coding guidelines

- Naming conventions:
  - File names (start with SWE nickname, mandatory files, internal files)
  - External information ➔ start with SWE nickname
  - Naming recommendations for functions, constants, variables, pointers…
- Comments format (Javadoc) ➔ automatic documentation generation
- General guidelines:
  - Error return, standard types, library usages, braces/indentation…
  - State machine implementation
  - No global variables ➔ Global Variable Buffer

Today…

- Riviera Overview

- What is a SW environment



- Main concepts of Riviera Environment

- A few Riviera SW entities overview

- Riviera Coding Guidelines

- Riviera Integration with other SW environment

Legacy Application Porting:

2 Different Options…

Flexibility for porting Applications:

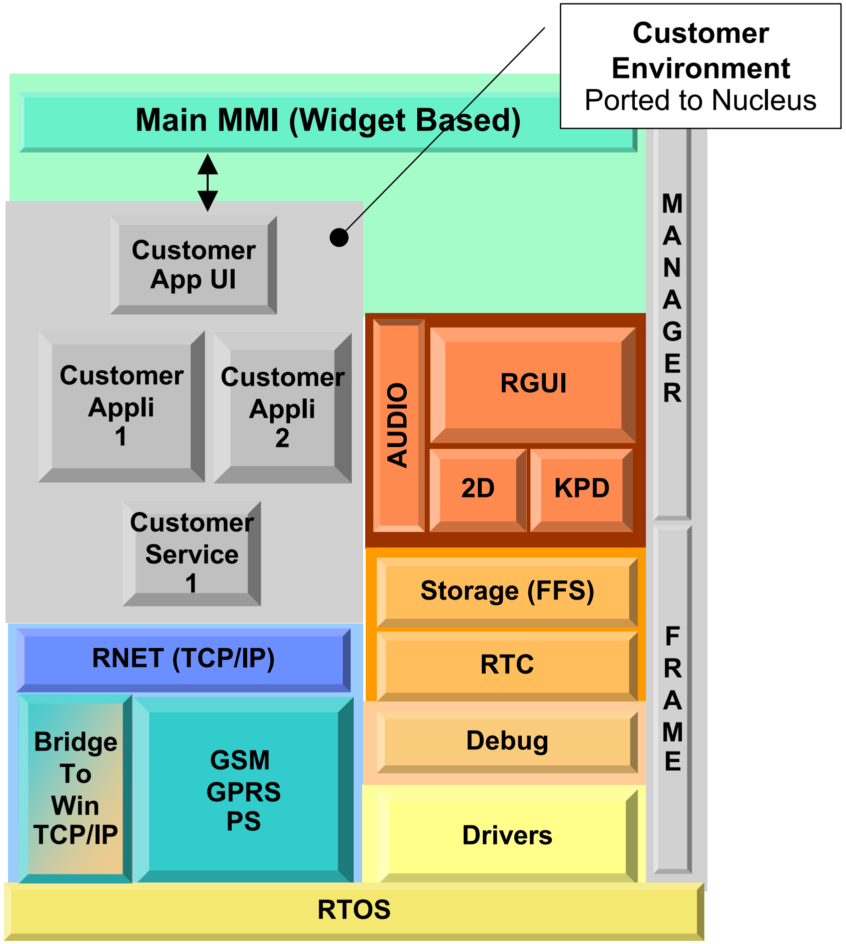
All Riviera services accessible both inside and outside Riviera Environment, thanks to:

- Function APIs
- “`return_path`” concept)

⇒ Riviera Environment can co-exist any other, Nucleus based Environment

⇒ Any Customer Application, running on Customer Environment, can access all Riviera Services.

Legacy Application Porting:
Option 1: Keeping Customer's Environment



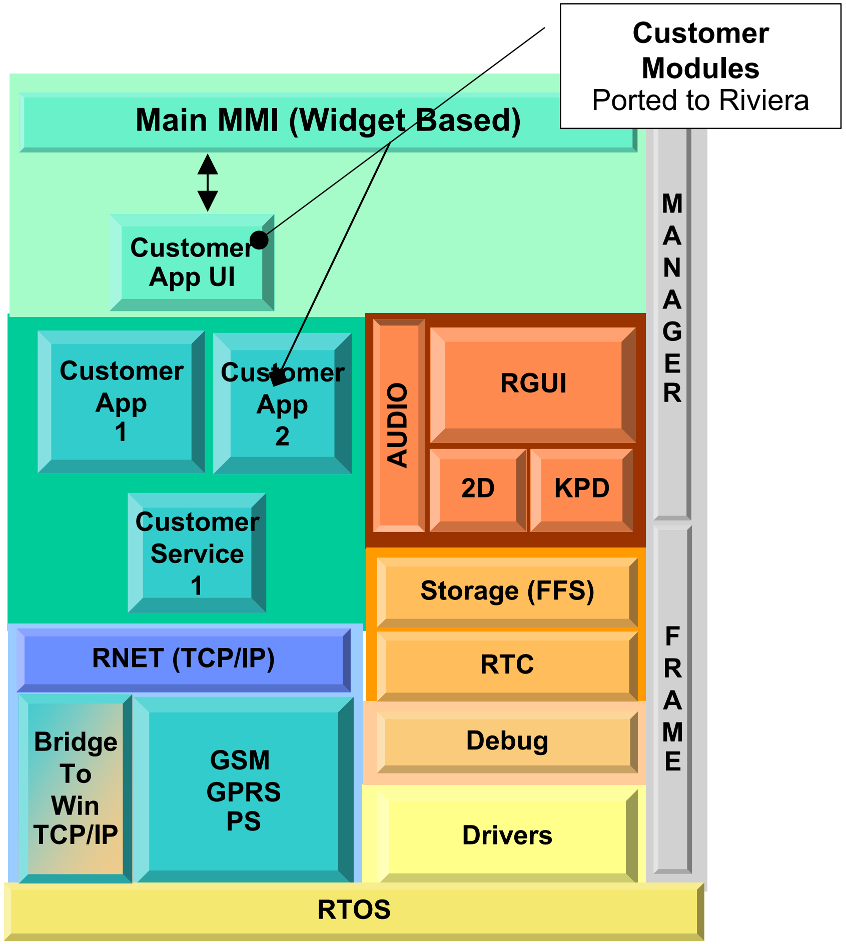
- **Customer Application, ported with Customer Environment, on top of Nucleus**
  (example of application using TCP/IP, FFS, UI Services…)

- **Drawbacks**:
  - Main is that no PC tool available to ease porting

- **Advantage:**
  - Easy and Fast porting, common framework to port all existing customer applications

Legacy Application Porting:

Option 2: Porting Customer's Application in Riviera



- **Customer Application, ported with Customer Environment, on top of Nucleus**
  (example of application using TCP/IP, FFS, UI Services…)

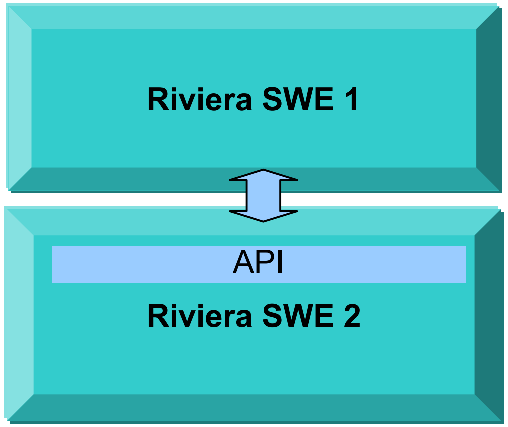
- **Advantage:**
  – Access to Riviera Tool Set (PC porting and validation, all necessary APIs available on PC)
  – Access to Riviera abstraction of the OS (ported to any TI chipset)

- **Drawbacks**:
  – Higher porting work.
  BUT: Customer defines intrusion (modular or 1 block porting)

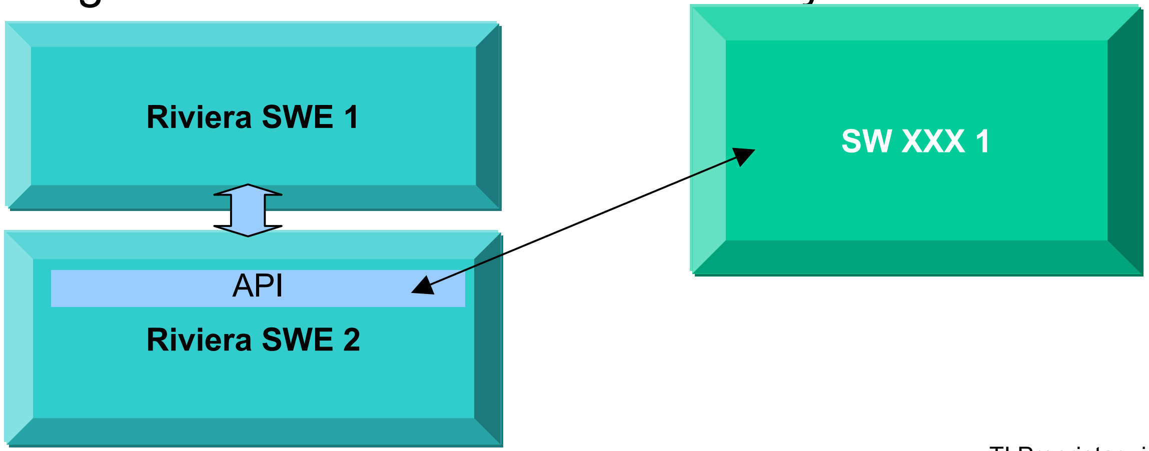
How can an external SW use Riviera Entity services

Service request and answer example

- Riviera Only case

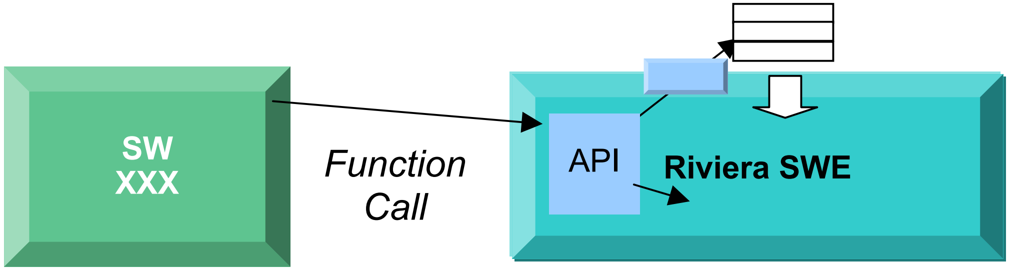


- External SW using services from a Riviera SW entity



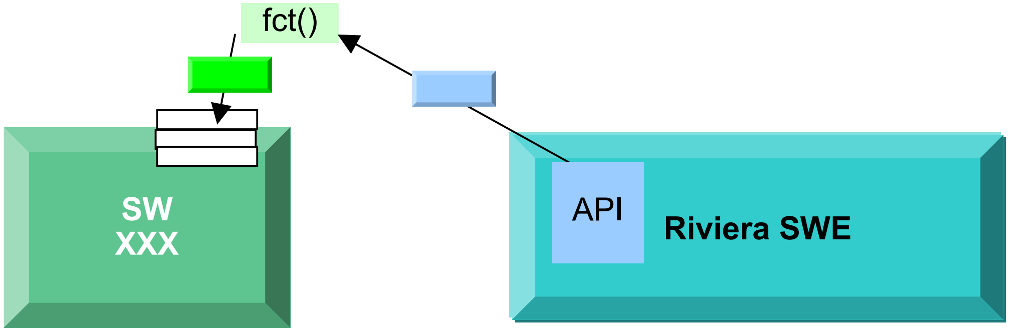
Non-Riviera SW Using Riviera Services

- External Environment requesting a Riviera Service



Services provided by Riviera to another environment are accessible through function call

- Riviera Service Answer provided to External Environment



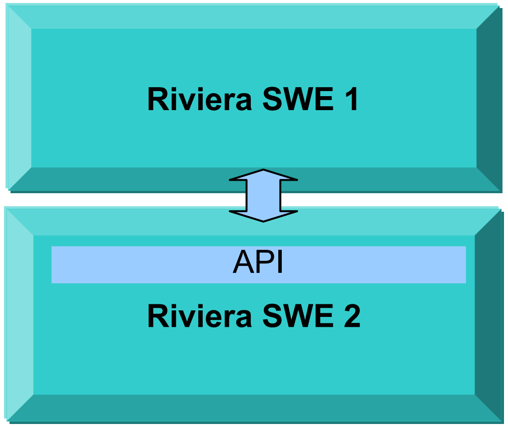
Thanks to the return path concept, Riviera message is sent to call back function provided

by the SW XXX, which can translate answer in the external environment format

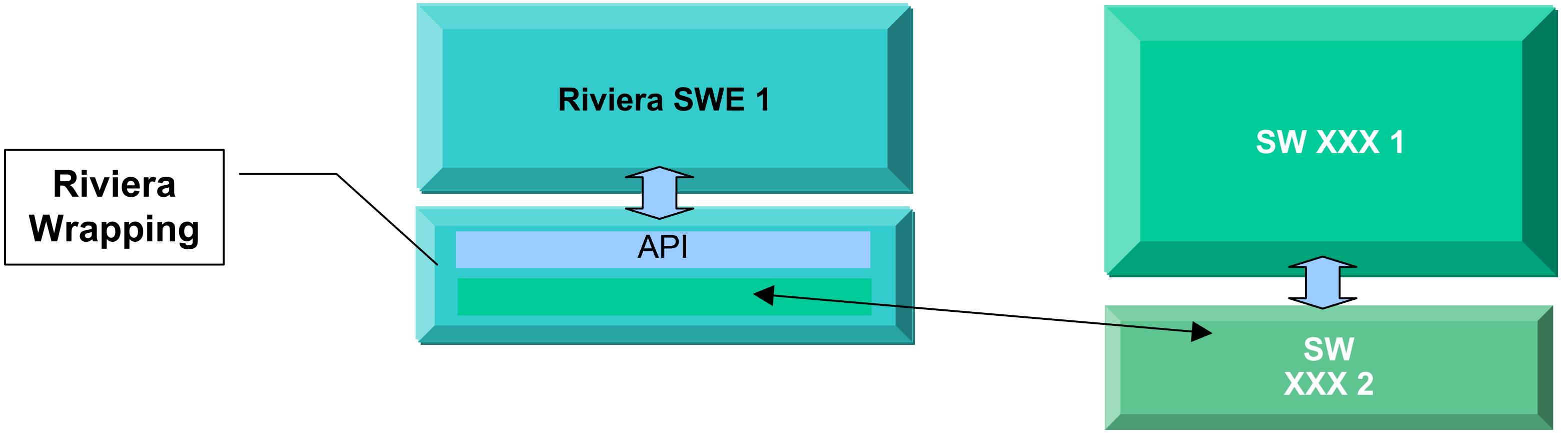
How can a Riviera SW entity use services from a external SW

Service request and answer example

- Riviera Only case



- Replacement of a Riviera SW entity by an external one



Conclusion

Riviera environment: Key features ?

- **Easy to program:**
  - No need of wireless knowledge (abstraction layer)
  - Independent on HW roadmap
- **Safe:**
  - Modem resource are protected (memory available, real time constraints)
  - Usual developers do not have access to critical parts of the SW
- **Modular:**
  - SW components can easily be added / removed / changed, dynamically, using only resources when activated
  - Easy to integrate a new SWE and debug
- **Cost Efficient:**
  - SW environment has a small footprint
  - Optimal memory use in term of RAM and FLASH
  - Data handling optimisation (zero copy mechanism)