



Guide To Creating Device Drivers for T-Kernel

Rev.1.00.01 (3.Sep,2007)

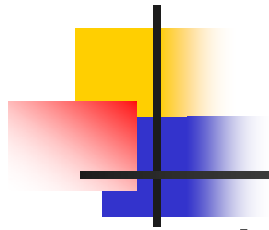
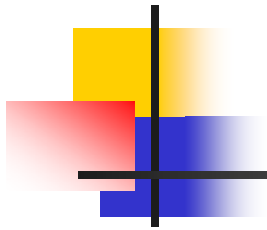
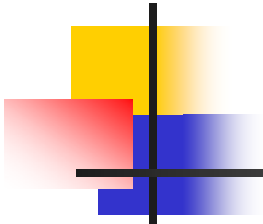


Table of Contents

- 1. Introduction**
- 2. Outline of the Device Drivers**
- 3. Device Driver Interface Library**
- 4. Implementation of Device Drivers**
- 5. Implementation Supplements**
- 6. References**



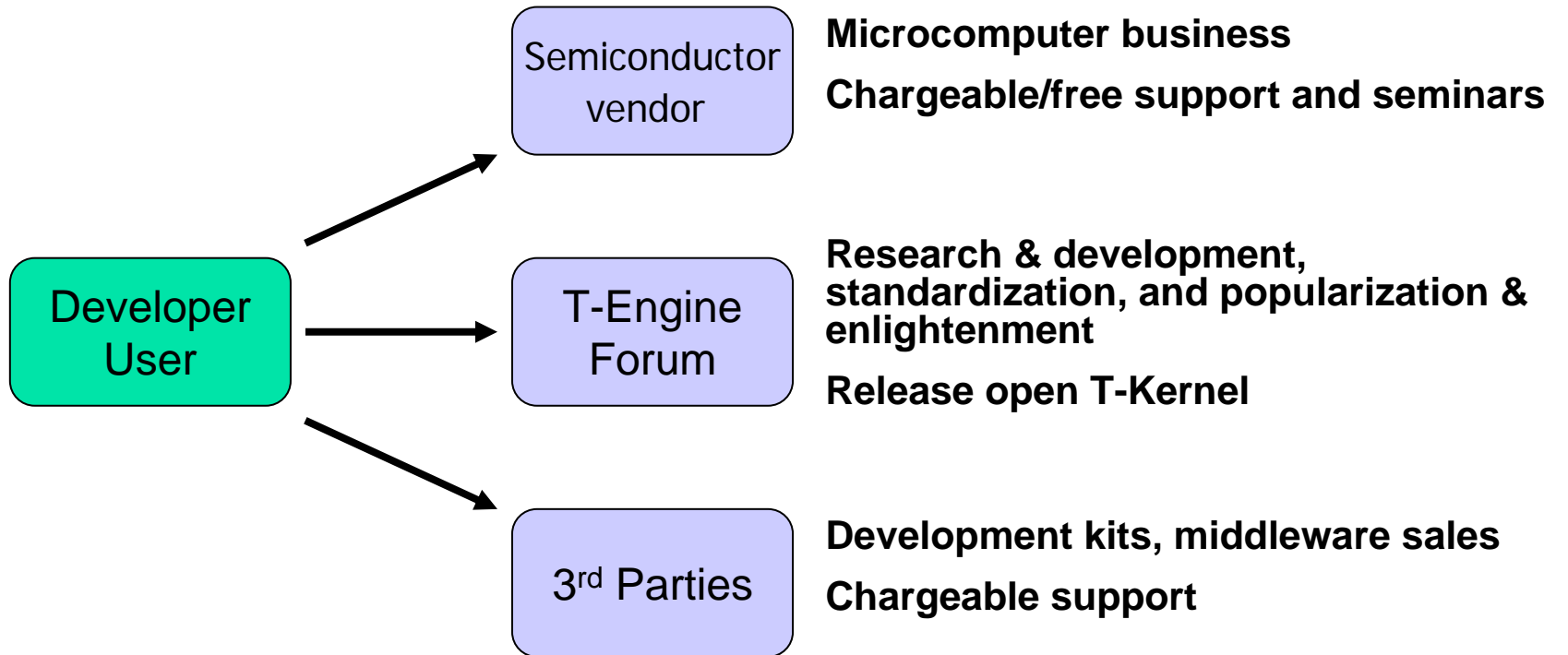
1.Introduction

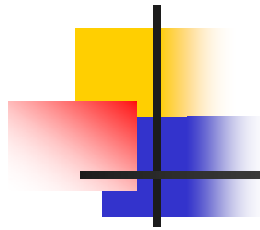


1.Introduction

■ Background

T-Engine environment around the user (developer)





1.Introduction

■ Things needed when you develop a driver

-The following can be obtained from T-Engine Forum Website

T-Kernel source code

T-Kernel specification, Implementation specification

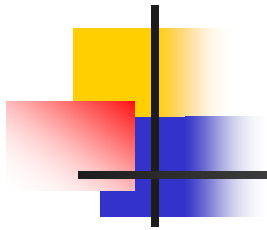
T-Engine standard device drivers specification

Device driver interface library

· The following items are usually bundled with the T-Engine

Development kit

development environment, T-Monitor, driver, and various other items



2. Outline of the Device Drivers



2. Outline of the Device Drivers

■ What is a device driver?

A device driver is the program that mainly controls hardware. In other words, a device driver is the software that provides a common interface for the high-order software, enabling it to access devices without concern for the differences in hardware.

■ T-Engine device drivers

Driver specifications for the devices incorporated in the T-Engine are stipulated.

- RS-232C driver
- LAN driver
- System disk driver
- Clock (RTC) driver
- Console driver
- USB manager
- PCMCIA card manager
- eTRON SIM driver
- Keyboard/Pointing device driver
- Screen driver

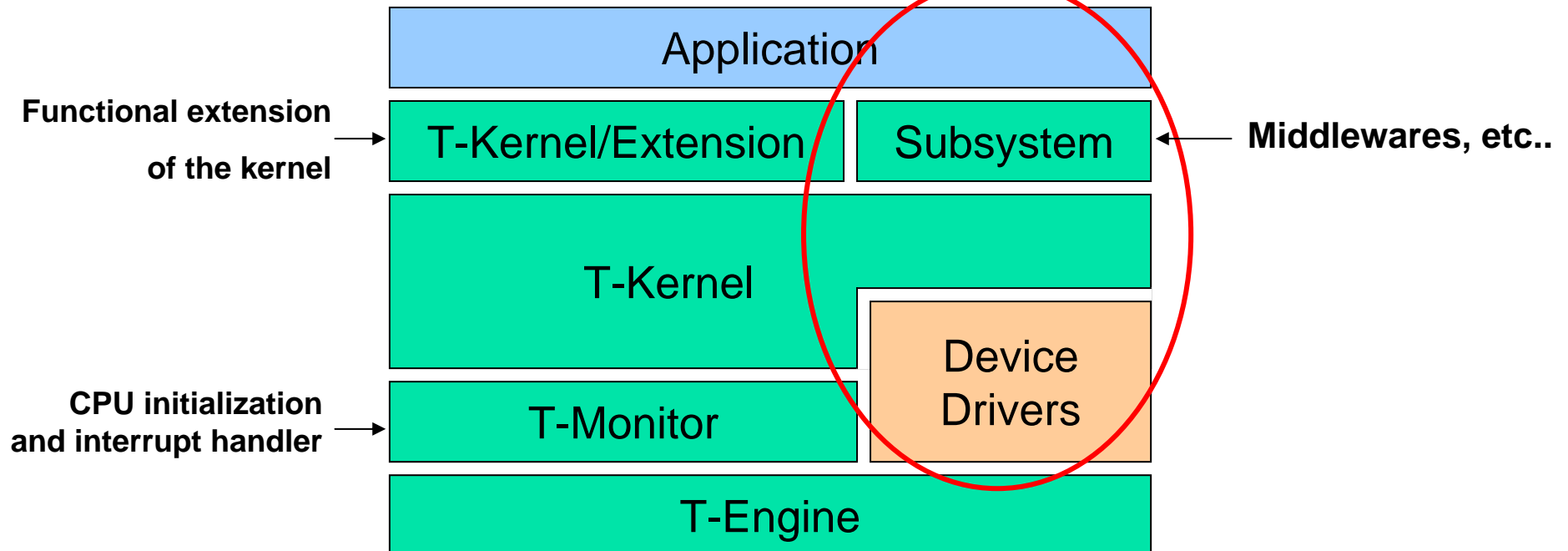
2. Outline of the Device Drivers

■ T-Engine Software structure

T-Monitor : Initializes the CPU, handles exceptions/interrupts, and provides debug facility

T-Kernel/Extension : Extends the T-Kernel facility to offer advanced OS facility

Subsystem : Middleware, etc.

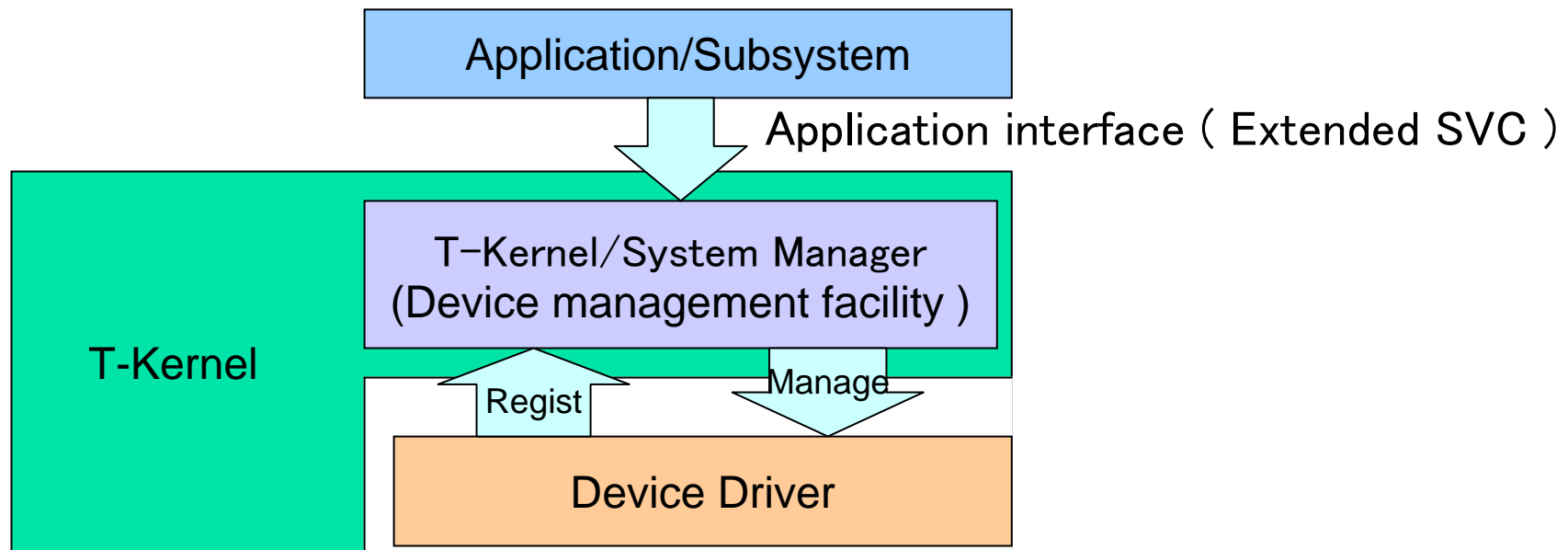


2. Outline of the Device Drivers

■ Applications and device drivers

The device driver is managed using the device management facility of the T-Kernel/System Manager.

The device driver is accessed from applications or middleware using service calls of the device management facility.



2. Outline of the Device Drivers

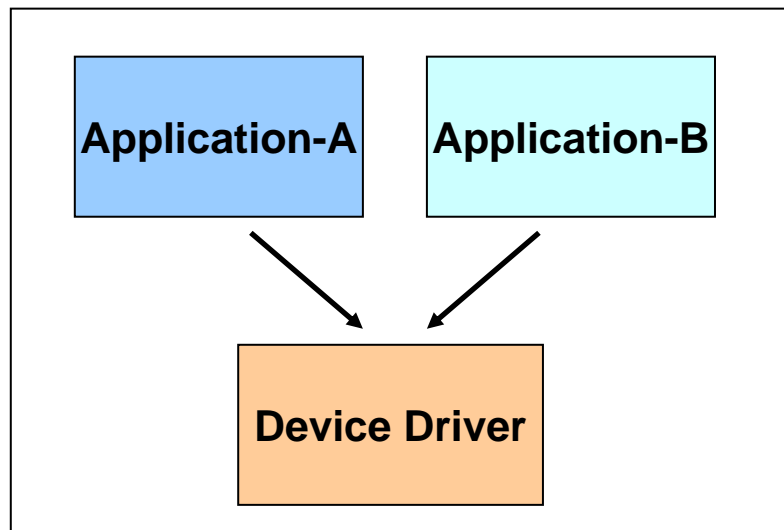
■ Device management facility

This management facility is provided for the device driver facility to be used from applications.

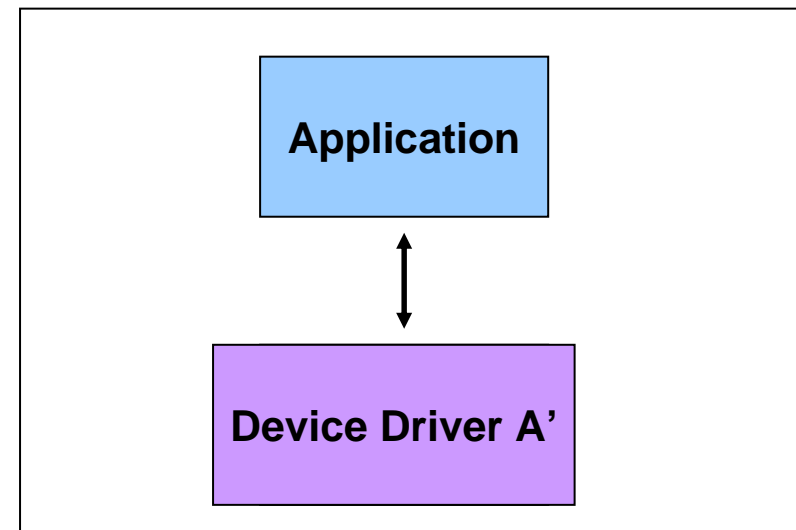
It permits driver software to be ported between different applications or CPUs easily.

A dynamic registration/removal of drivers also makes effective use of hardware resources and upgrading of driver versions (replacement) possible.

Increased portability



Versionup, etc.





2. Outline of the Device Drivers

■ Application interface of the device management facility

The device management facility is implemented as a sub-system of the T-Kernel, providing the extended service calls listed below.

tk_opn_dev : Opens a device to get it prepared for access

tk_cls_dev : Closes a device

tk_rea_dev : Reads data from a device (asynchronous)

tk_srea_dev : Reads data from a device (synchronous)

tk_wri_dev : Writes data to a device (asynchronous)

tk_swri_dev : Writes data to a device (synchronous)

tk_wai_dev : Waits for a request by tk_rea_dev or tk_wri_dev to complete

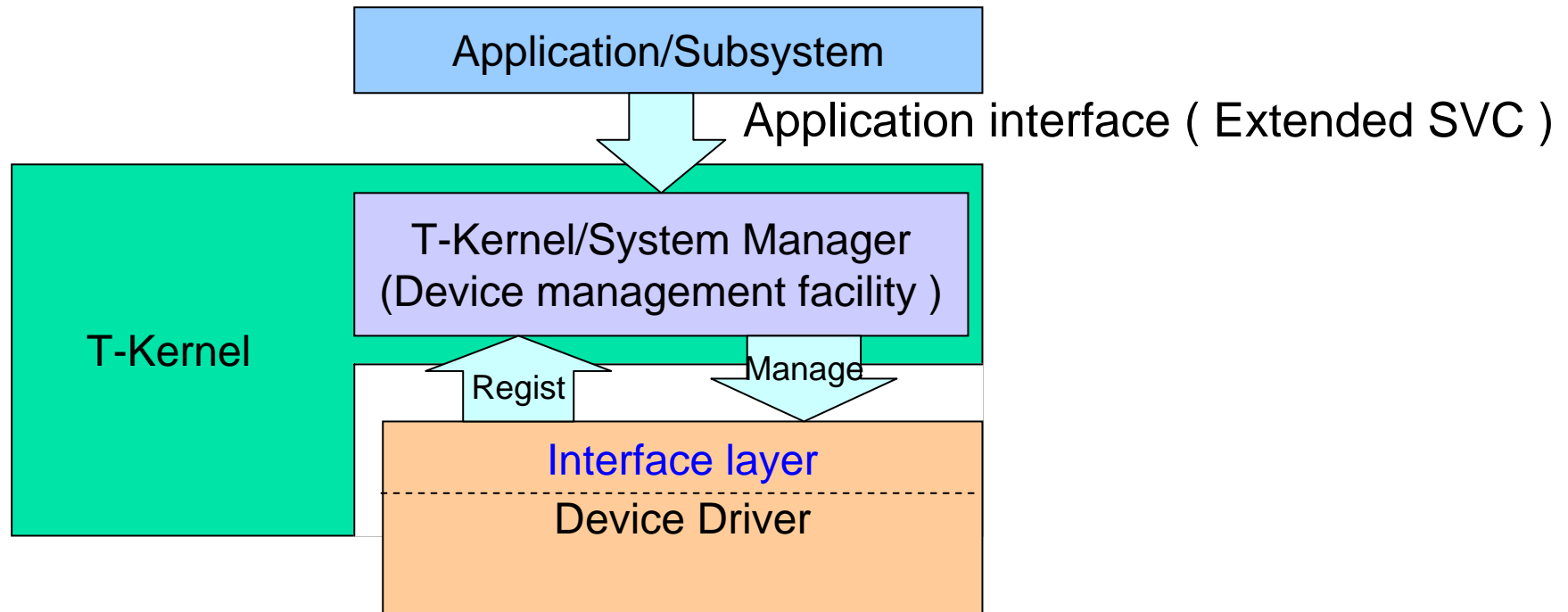
tk_def_dev : Registers a device

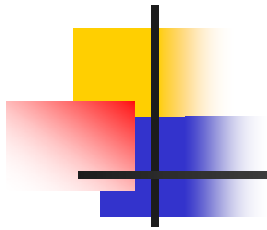
* For details, refer to the T-Kernel specification

2. Outline of the Device Drivers

■ Device driver interface

This is the standard interface to manage or control device drivers from the device management facility.





3. Device Driver Interface Library

3. Device Driver Interface Library

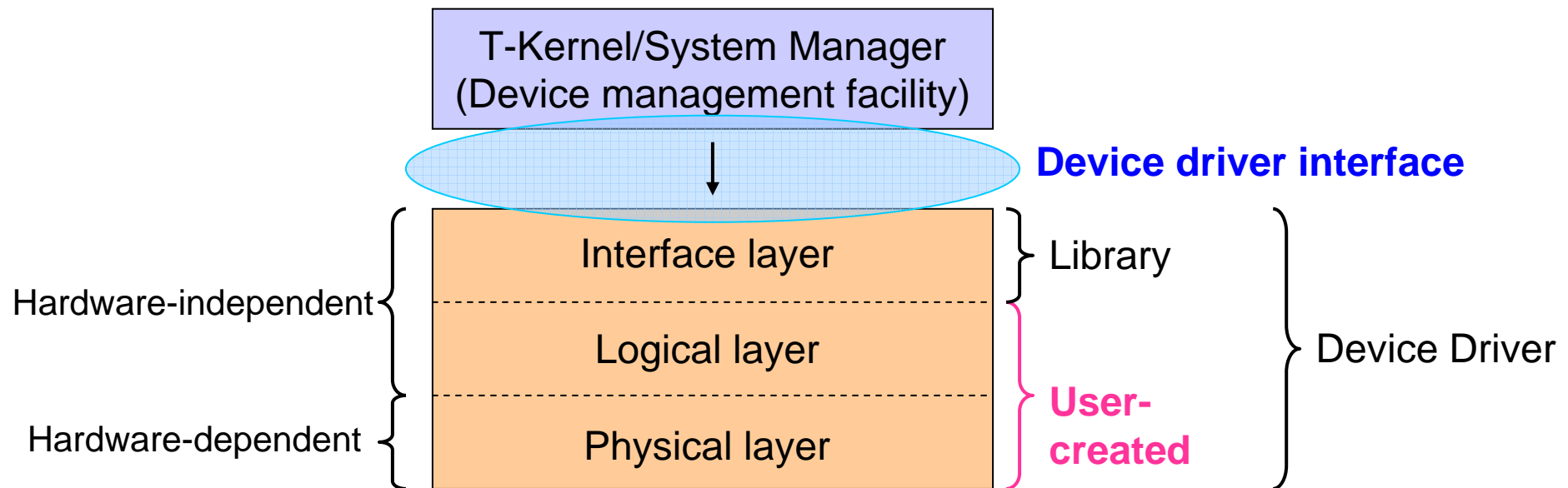
■ Configuration of the device driver

Interface layer: Interfacing with the device management facility

Logical layer: Hardware-independent processing

Physical layer: Hardware-dependent processing (actual device control)

Expansion to other MCU types is accomplished by only adding a physical layer





3. Device Driver Interface Library

■ Processing of the interface layer

- Checks parameters (interface part)
- Checks address space (access rights)
- Manages requests (by accepting requests and returning results in response)

and so on

Processing of the interface layer like those described above are the same for any driver concerned

Putting these tasks together in library form helps to take some of the load off the device driver developers.



3. Device Driver Interface Library

■ Device driver interface library

- Simple device driver interface (SDI)

This interface is provided for a simple driver that can service or respond to requests from applications immediately without keeping them waiting in it.
Example: Time-of-day clock (RTC) driver .

- Generic device driver interface (GDI)

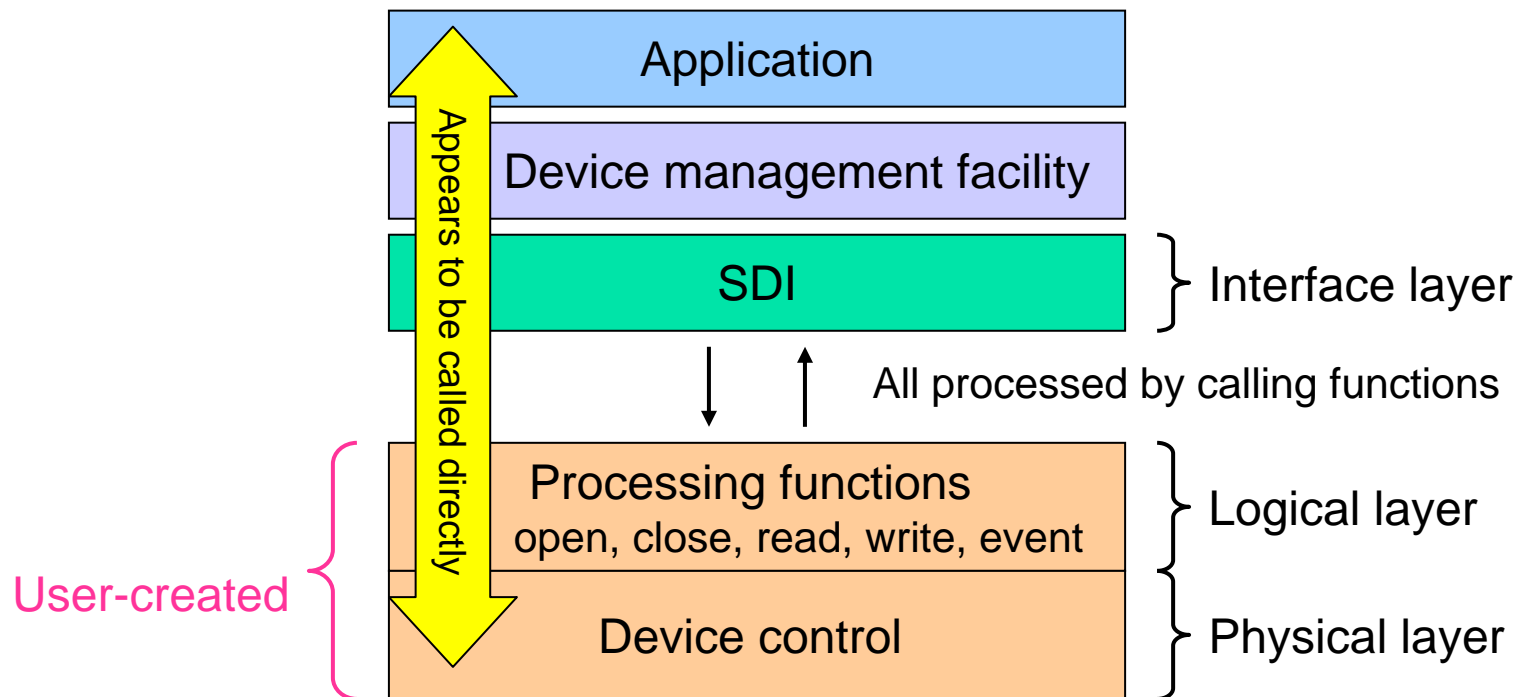
This interface is provided for a most general driver that sometimes withholds processing of requests from applications by keeping them waiting in it. This type of driver specifically includes the one that requires suspension of requests.

Example: RS-232C (serial) driver, storage related driver, etc.

3. Device Driver Interface Library

■ Simple device driver interface (SDI)

All of device driver processing is accomplished by calling functions from the SDI. For an application, this appears to be called directly from it. In no case will multiple input/output requests go to the driver at the same time.



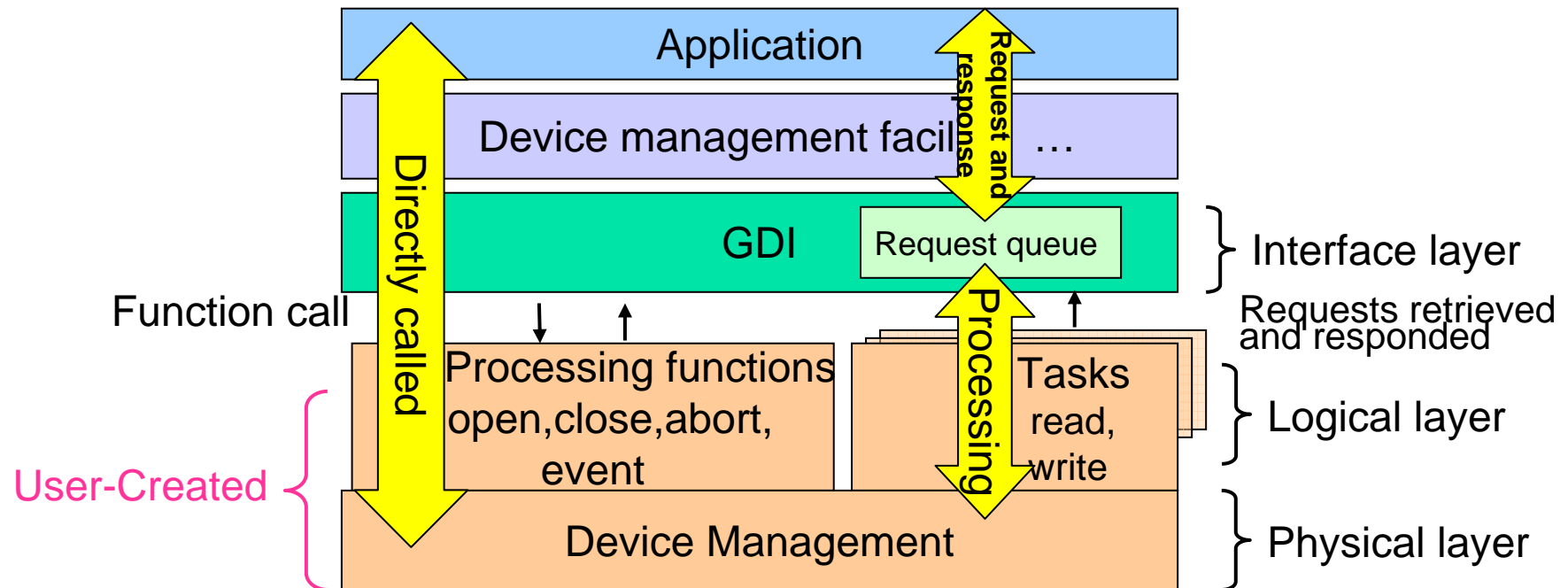
3. Device Driver Interface Library

■ Generic device driver interface

This interface has a queue, enabling the driver to accept multiple input/output requests at the same time.

Processing for input/output requests is performed in a driver task. The task retrieves input/output requests one at a time before processing them and returns the result of processing.

The driver has more than one processing task, so that as many requests as the number of tasks can be processed at the same time.





3. Device Driver Interface Library

■ APIs of the Device Driver Interface Library

- APIs common to the SDI and GDI

(1) Device registration:

Registers device information in the device management facility, said information including a device name, attribute, and processing functions (e.g., open or close).

(2) Device removal:

Removes a device from the device management facility.

(3) Device information acquisition:

Acquires device information such as device ID and extended information that was set when registered.

- APIs specific to the GDI

(1) Request retrieval:

Retrieves input/output requests from a queue in which they are queued up. If there are no input/output requests at all, the driver waits in the API.

(2) Request-complete response:

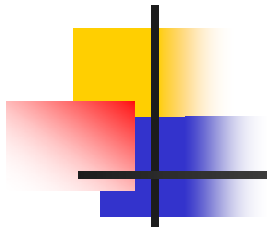
Returns the result of processing for the input/output requests serviced.



3. Device Driver Interface Library

■ Supplements

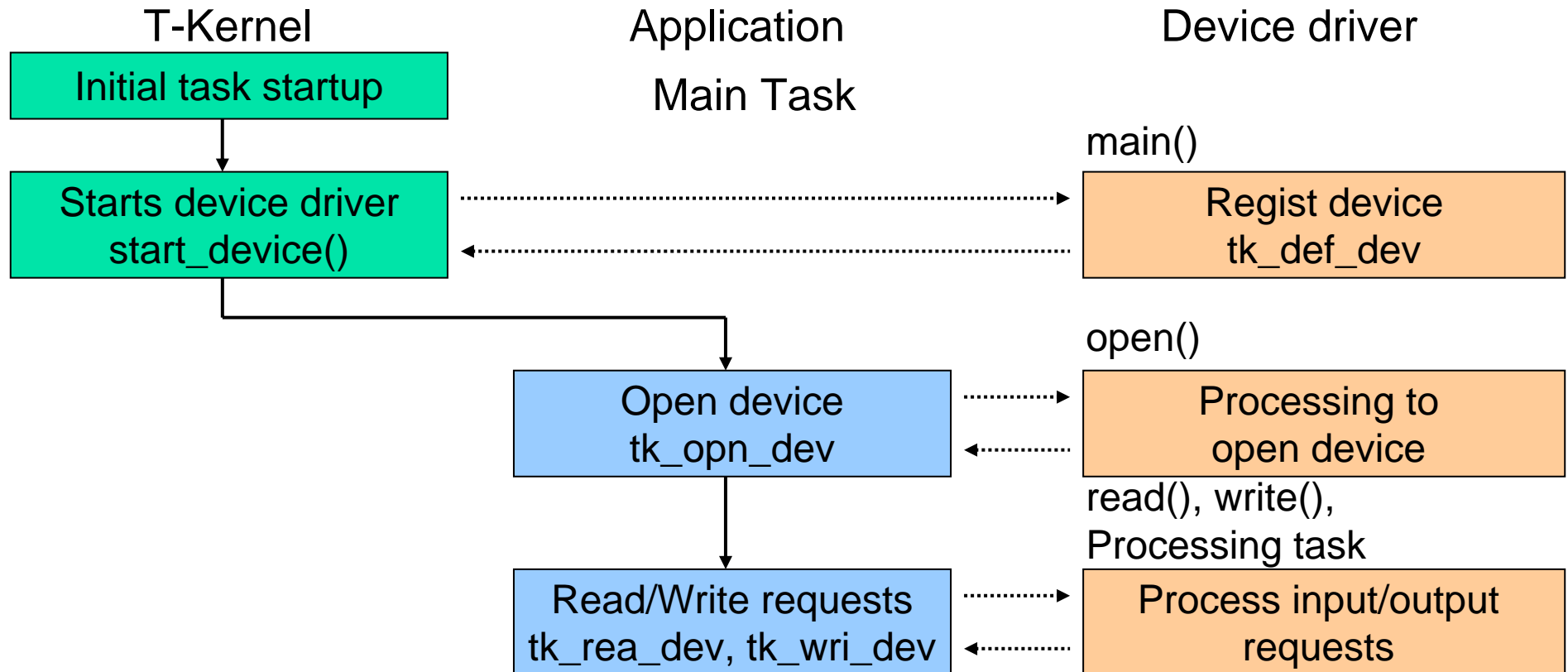
- This library is intended to simplify the implementation of device drivers, and does not always need to be used. For the sake of portability, however, driver development by use of this library is recommended.
- This library is include T-Kernel/Standard Extension sample driver and also included with the T-Engine development kit.



4. Implementation of Device Drivers

4. Implementation of Device Drivers

■ Flow of processing





4. Implementation of Device Drivers

■ Processing functions prepared by drivers using the SDI

| | | |
|--------------|------------------------------|--|
| Main | Entry routine | Starts or registers a device. |
| Open | Open device | Processing necessary to start using a device. It is invoked by a <code>tk_opn_dev()</code> system call. |
| Close | Close device | Processing necessary to stop using a device. It is invoked by a <code>tk_cls_dev()</code> system call. |
| Read | Device input request | Aborts the processing of a request under execution, with control returned to the application. It is invoked when request processing needs to be aborted, as in the case of <code>tk_cls_dev()</code>. |
| Write | Device output request | Writes data to a device. It is invoked by a <code>tk_wri_dev()</code> system call. |
| Event | Driver event | Processes a request event to a device driver. It is called from <code>tk_evt_dev()</code> system call, etc. |



4. Implementation of Device Drivers

■ For the clock (RTC) driver case (Example)

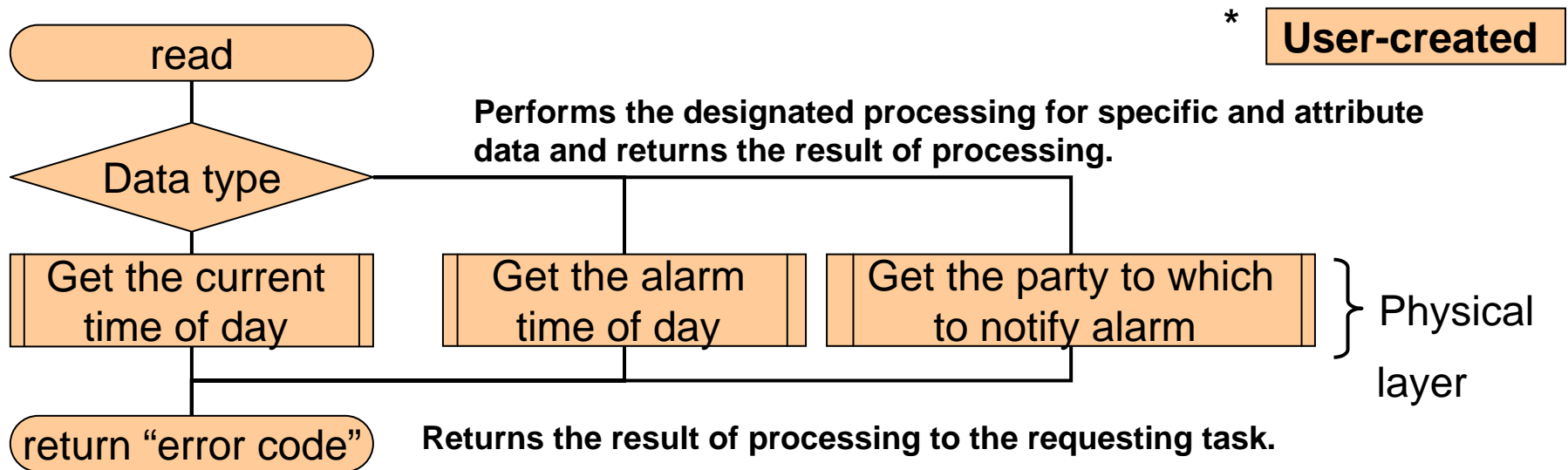
| | |
|--------------------------|--|
| Main | Initializes a device or registers a device. |
| Open | None. |
| Close | None. |
| Read | Acquires the current time of day, etc. |
| Write | Sets the current time of day, etc. |
| Event | None |
| Interrupt handler | Notifies an alarm, etc. |

**If no particular processing is needed, open and close can be omitted.
Further, event is unused either.**

4. Implementation of Device Drivers

■ read (reads specific or attribute data)

After determining the type of data, the driver performs the designated processing for each data type.



4. Implementation of Device Drivers

■ Specific data and attribute data

The device data is classified by a “data number.”

- **Specific data: Data number ≥ 0**

This is the data specific to a device, where data numbers are defined for each device.

- **Disk** : Data number = physical block number

- **Serial line**: Serial data (number = 0) only

- **Attribute data: Data number < 0**

This data includes the driver or device status, set data, etc.

The “current time-of-day setting/acquisition” and “alarm time-of-day setting/setting” of the RTC driver also comprise attribute data.

```
ID tk_rea_dev ( ID dd, INT start, VP buf, INT size, TMO tmout )
```

dd Device descriptor

start Start position to read (≥ 0 : specific data, < 0 : attribute data)

buf Buffer in which the read data is stored

size The size of data to read

tmout Request acceptance wait time-out time (in ms)

Return value

Request ID or error



4. Implementation of Device Drivers

■ Processing functions needed for drivers using the GDI

| | | |
|--------------|----------------------------|---|
| Main | Entry routine | Starts or registers a device, starts a processing task, and registers an interrupt handler. |
| Open | Open device | Processing necessary to start using a device. It is invoked by a tk_opn_dev() service call. |
| Close | Close device | Processing necessary to stop using a device. It is invoked by a tk_cls_dev() service call. |
| Abort | Stop device request | Aborts the processing of a request under execution, with control returned to the application. It is invoked when request processing needs to be aborted, as in the case of tk_cls_dev(). |
| Event | Driver event | Processes a request event to a device driver. It is called from tk_evt_dev(), etc. |



4. Implementation of Device Drivers

■ Processing tasks and input/output processing needed for drivers using the GDI

| | | |
|--------------|--------------------------------|--|
| Task | Request processing task | Retrieves from the queue a device input/output request from the application and performs the necessary input/output processing. |
| Read | Device input request | Reads specific or attribute data from a device. Upon abortion request from the abort function, the driver aborts the processing underway. |
| Write | Device output request | Writes specific or attribute data to a device. Upon abortion request from the abort function, the driver aborts the processing underway. |



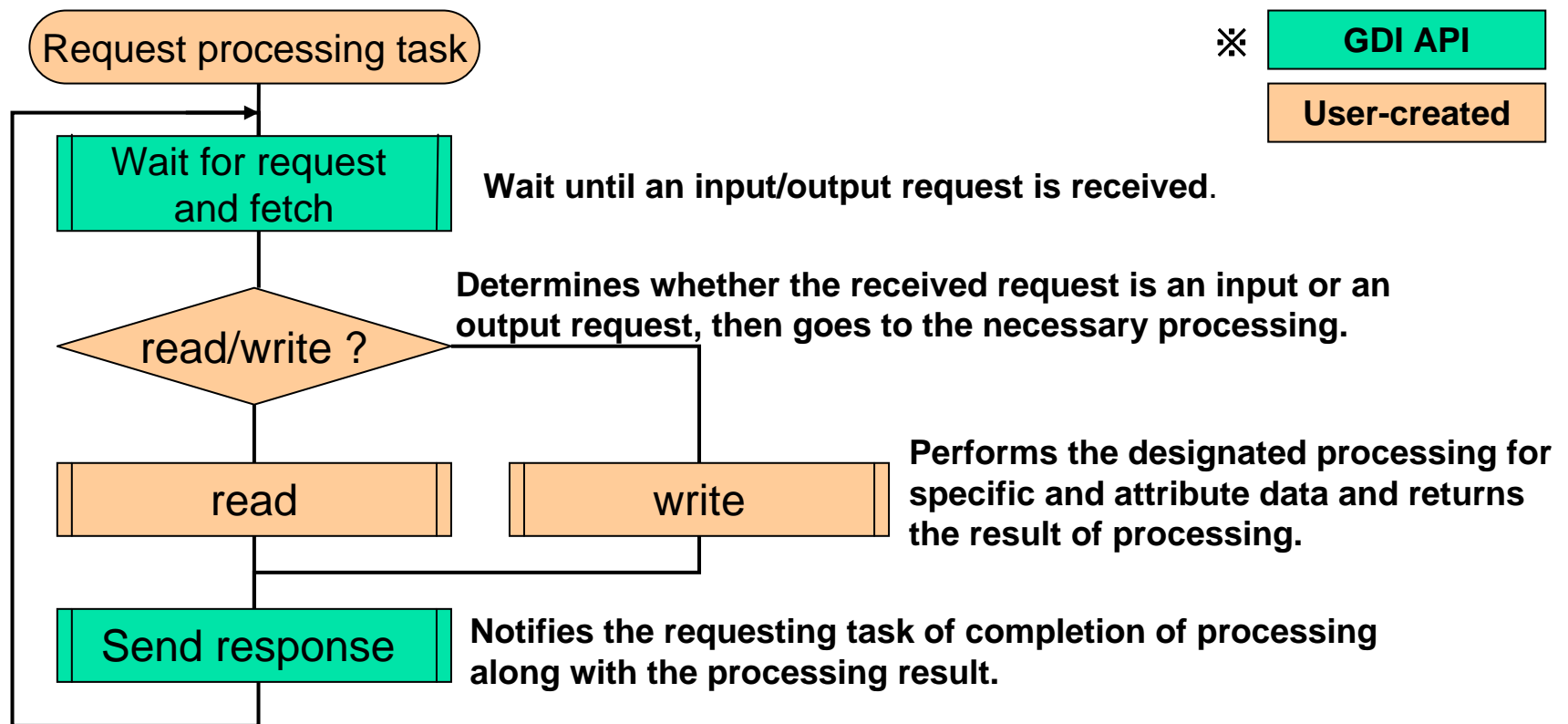
4. Implementation of Device Drivers

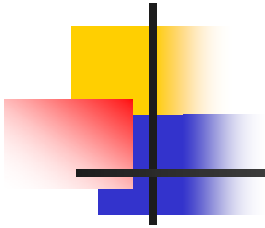
■ For the RS-232C driver case (Example)

| | |
|--------------------------|---|
| Main | Starts or registers a device, starts a processing task, and registers an interrupt handler. |
| Open | Opens a specified port. |
| Close | Closes a specified port. |
| Abort | Aborts transmit/receive processing. |
| Event | None. |
| Task | Acquires an input/output request and returns the result as a response. It also performs the necessary processing according to the acquired input/output request. |
| Read | Receives serial data, as well as acquires various setup data. |
| Write | Transmits serial data, as well as sets various setup data. |
| Interrupt handler | Serial data ready for sending, reception or communication error interrupt, etc. |

4. Implementation of Device Drivers

■ Request processing tasks (read or write specific and attribute data)





5. Implementation Supplements



5.Implementation Supplements

■ Initialization of device drivers

Add a device driver entry routine (startup and registration processing) to `start_device` (kernel/sysdepend/device/[TARGET]/devinit.c) .

```
/*
 * Start processing after T-Kernel starts
 *   Called from the initial task contexts.
 */
EXPORT ER start_device( void )
{
    ClockMain(NULL, NULL); /* Initializes clock driver */

    return E_OK;
}
```



5. Implementation Supplements

■ Link with the T-Kernel

Add the objects of device drivers to `kernel/sysmain/build/[TARGET]/Makefile` (for the static link case).

Manager and Driver objects (links to the kernel)

K_OBJ = `$(BD)/driver/clock/build/$(TETYPE)_$(MACHINE)/clock.o`



5.Implementation Supplements

■Others

- Regarding the sub-system ID

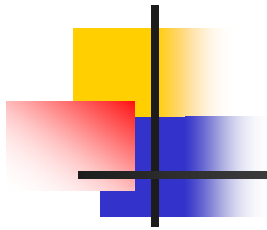
The T-Engine standard device drivers are assigned their sub system IDs in the T-Engine specification (fixed IDs).

However, no documents are found in which these definitions are written.

- Regarding support for the T-Format of device drivers

Devised to ensure smooth distribution of middleware, this is an arrangement to avoid unnecessary competition between software.

This format stipulates the source code styles of file names, symbol names, etc., as well as the binary format and document types.



6.References



6. References

■ Books

T-Kernel Standard Handbook, Revised New Edition

Supervised by K. Sakamura; Edited and written by T-Engine Forum

■ References from the T-Engine Forum

- T-Kernel specification v1.00.00
- T-Engine Introduction for Middleware Developers v1.01
- T-Engine Standard Device Driver Specification TEF040-S211-01.00.00
- T-EngineT-Engine Device Driver Interface Library Specification
TEF040-S211-01.00.00

■ References for the members of the T-Engine Forum

Device Driver Implementation Guidelines, 2nd Edition WG040-031027-u02

In addition to the above, other references are available including the T-Engine manuals and mounted device specifications supplied with the development kit.