Embedded Operating Systems

Minsoo Ryu

Department of Computer Science and Engineering
Hanyang University

msryu@hanyang.ac.kr

Outline

1. Introduction to Embedded Operating Systems
2. Examples of Embedded Operating Systems
3. Multitasking without OS Support

Definition of Embedded OS

❐ OSes used in embedded systems

- Past embedded systems did not use operating systems, but implemented in firmware
- Modern embedded systems are adopting operating systems

❐ Main requirements on embedded OS

- Small size requirement for limited resources
- Real-time services
- Predictability

Why OS in Embedded Systems?

❒ Technical aspects

- Multitasking
- Networking and communications
- Storage and memory management
- Protection and security
- GUI

❒ Economic aspects

- Time-to-market
- Low NRE (non-recurring engineering) cost 연구, 설계, 검사하는 최초의 비용
- Maintenance cost
  - 40% - 80% in the software lifecycle

Typical Embedded Operating Systems

- ❒ RTOS
  - VxWorks, QNX, pSOS, VRTX, Nucleus, WinCE
- ❒ Mobile OS
  - iOS, Android, Tizen, WinCE, Symbian, PalmOS
- ❒ Open general purpose OS
  - Linux, Solaris 10, FreeBSD, eCOS
- ❒ DSP OS
  - ARC/MQX, DSP/BIOS
- ❒ Others
  - TinyOS (smart sensor), Nachos (education), Xinu (education),
  - IOS (Internetwork OS, Cisco routers)
  - ThreadX (pripherals)

Commercial RTOSes

❐ Commercial RTOSes

- **QNX, VxWorks, pSOS, VRTX, Nucleus**

❐ Features

- **Multithreading, real-time synchronization and POSIX APIs**
- **Lightweight kernel**
  - Even scaled down to fit in the ROM of the system
- **Modularized structure**
  - Minimum kernel + service components
- **Responsiveness**
  - Minimal interrupt and switching latency

RTOS does not mean “fast”

- RTOS adds runtime overheads
- Slightly slower performance than OS-less systems

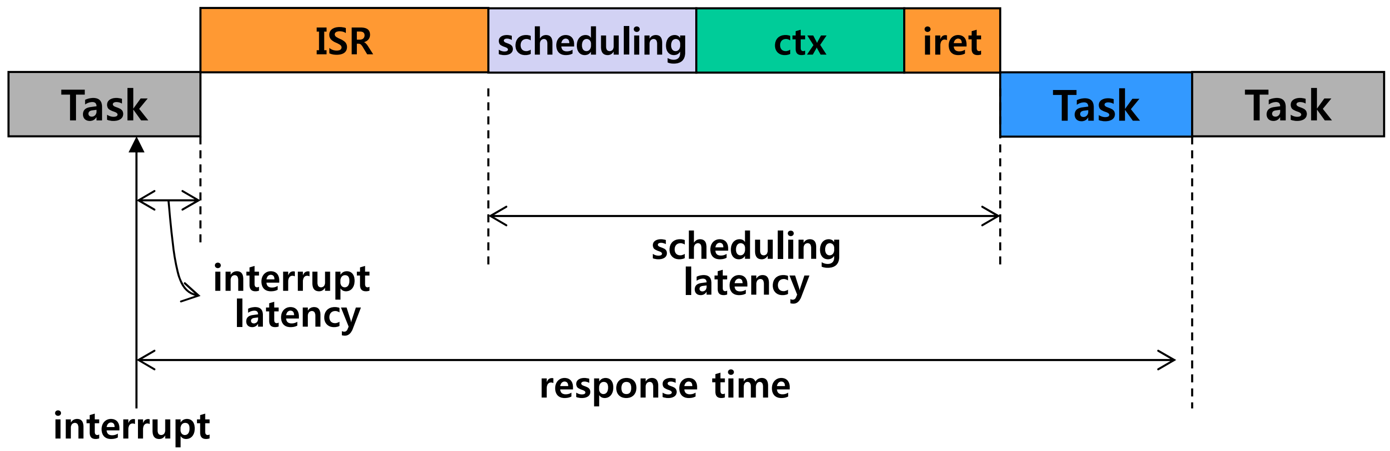
RTOS does mean “deterministic”

- Guarantees “worst case” times
- Better performance than GPOS

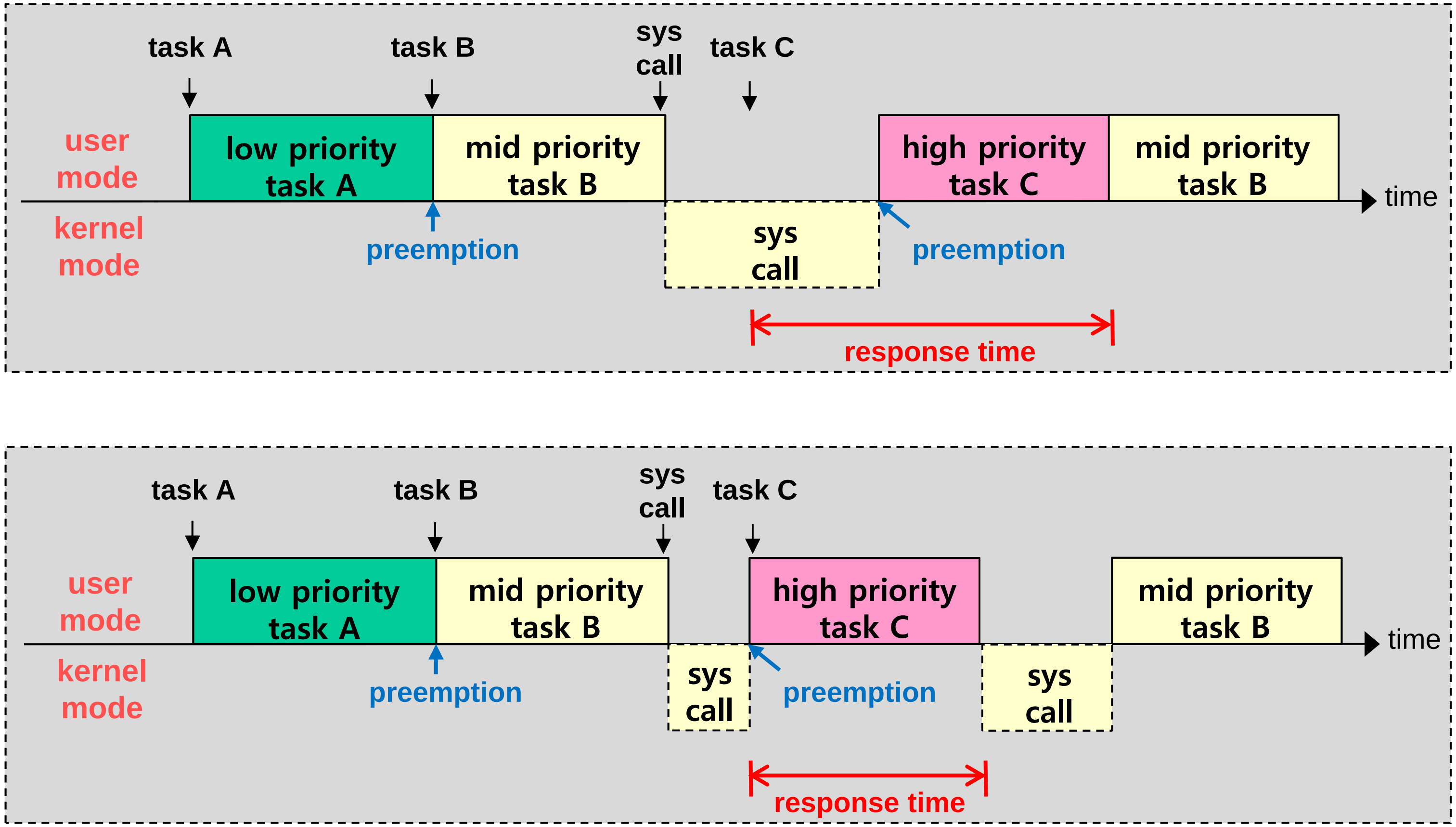
Timing Guarantees from RTOS

❐ Key requirements

- **Fully preemptive priority-based scheduling**
  - High priority tasks preempt low ones
- **Bounded latency**
  - eg.) interrupt off time < α and scheduling latency < β



Partially Preemptive vs. Fully Preemptive

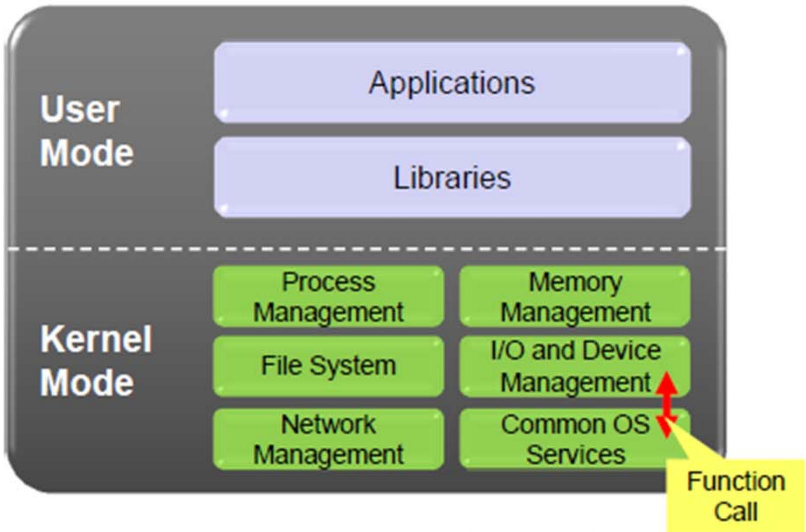


system call 실행시 더 높은 task의 preeptive 허용 X
-> real time scheduling 불가능

Monolithic Kernel vs. Microkernel

portability 증가
각각 모듈화되어있어서 코드끼리의 연관성이 적어서 모듈 교체가 쉽다.
monolithic보다 성능이 나쁘다.
real time에서 채택됨.
80년대 초반에 rtos 출현했었는데, micro kernel이 그 당시 주목받던 구조였기 때문에

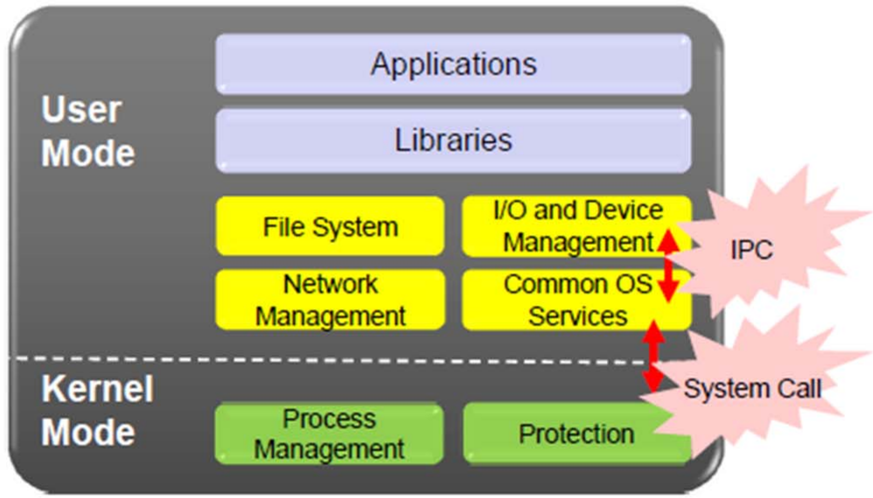
Monolithic Kernel



- Every kernel functionality runs in kernel mode
- Better Performance
- E.g., UNIX, Linux, MS Windows 9x

대부분 os의 구조
각각의 모듈들이 타이트하게 연결되어 있다.

Microkernel

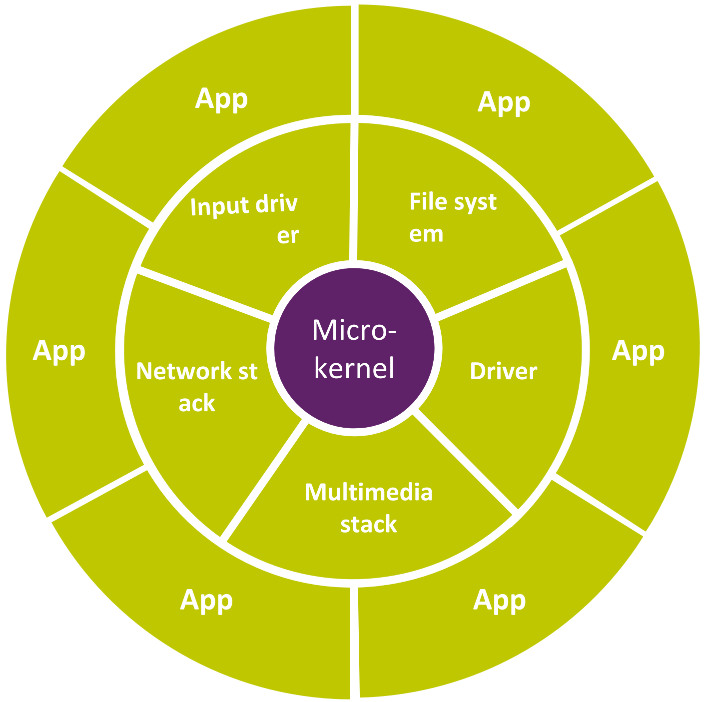


- Most kernel functionality runs in user mode except only the elementary functions
- Better extensibility and modularity
- E.g., MINIX, L4, QNX

QNX Neutrino RTOS

❒ Microkernel structure

- Moves as much from the kernel into "*user*" space

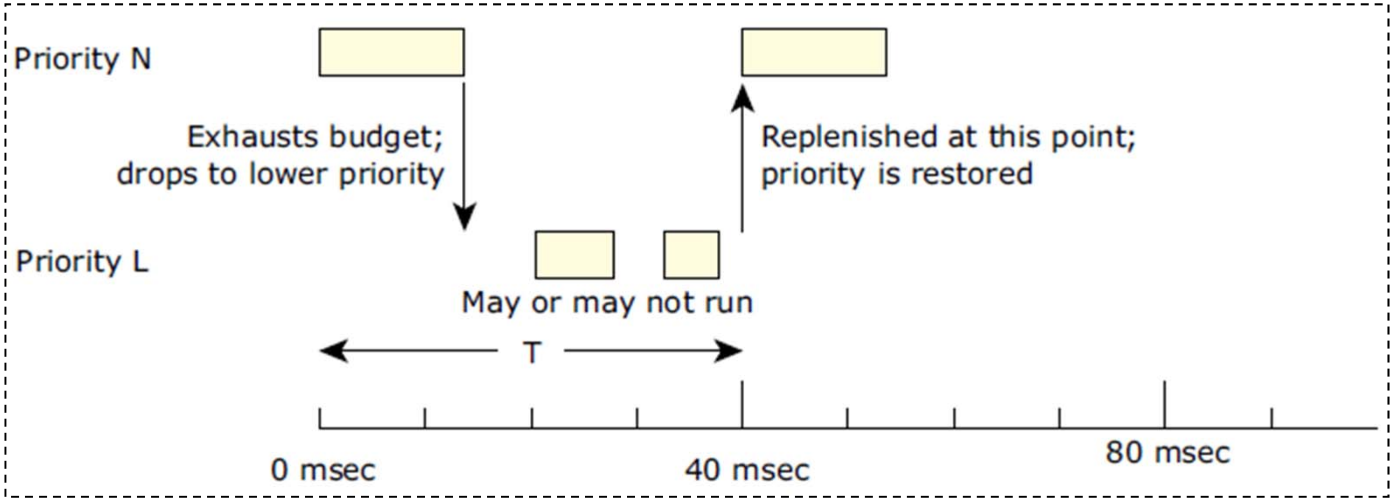


❒ Rich features

- Momentics C/C++ IDEs
- Photon microGUI
- HMIs such as HTML5 and Qt
- Multimedia and acoustics support
- Mobile connectivity framework and navigation

QNX Support for Sporadic Scheduling

- ❒ **Scheduling of real-time periodic tasks**
  - **Initial budget: the amount of time a task is allowed to run**
  - **Replenishment period: the period of task execution**



QNX High Availability Manager

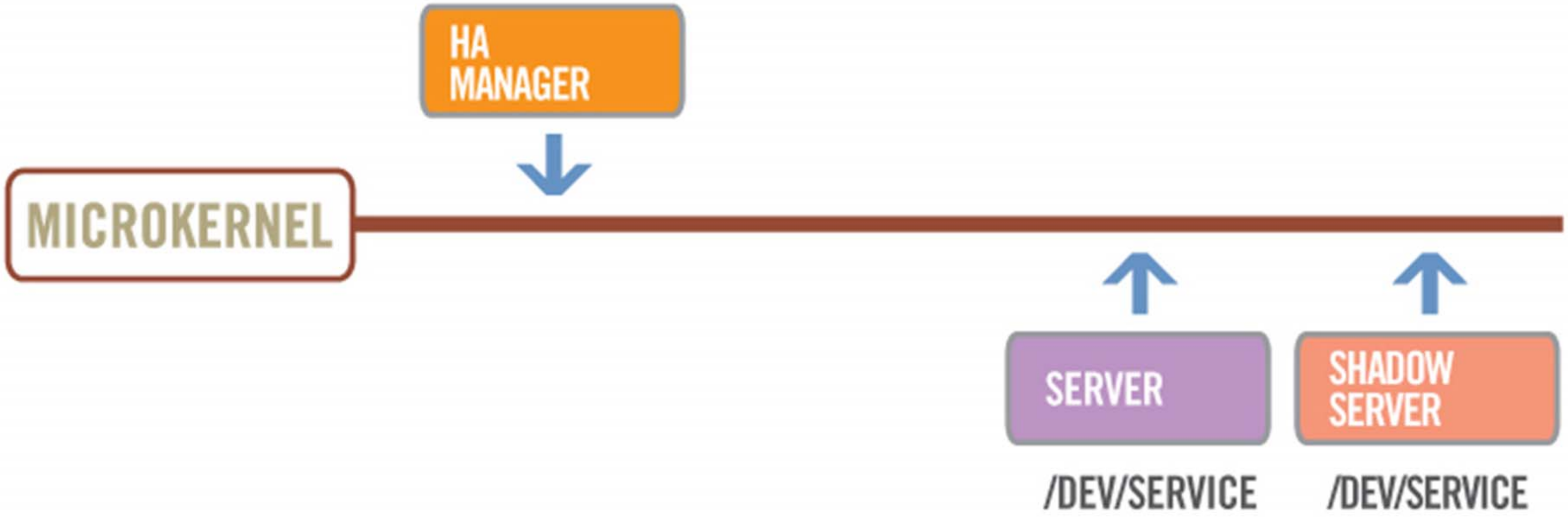
❒ HAM (High Availability Manager)

- **Monitor tasks and services (smart watchdog)**
- **Perform multistage recovery**

❒ HAM Hierarchy

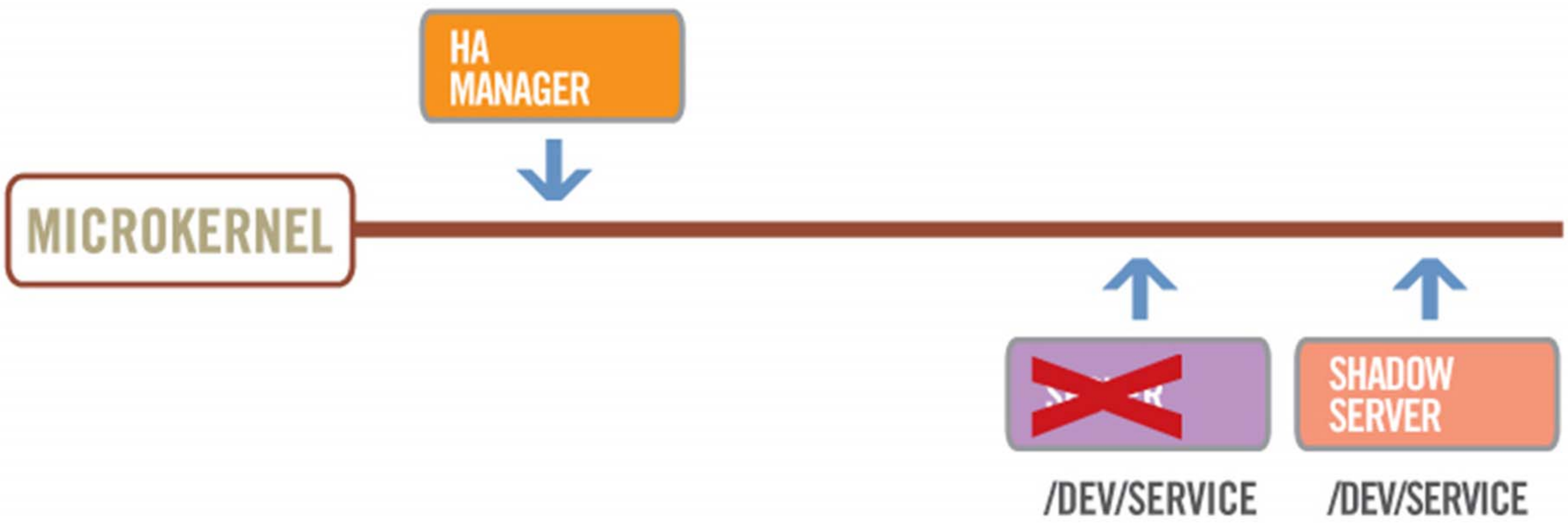
- **Entities**
  - Units of observation and monitoring
- **Conditions**
  - Represent entity states
- **Actions**
  - Executed when the appropriate conditions are true with respect to a specific entity

HAM Example



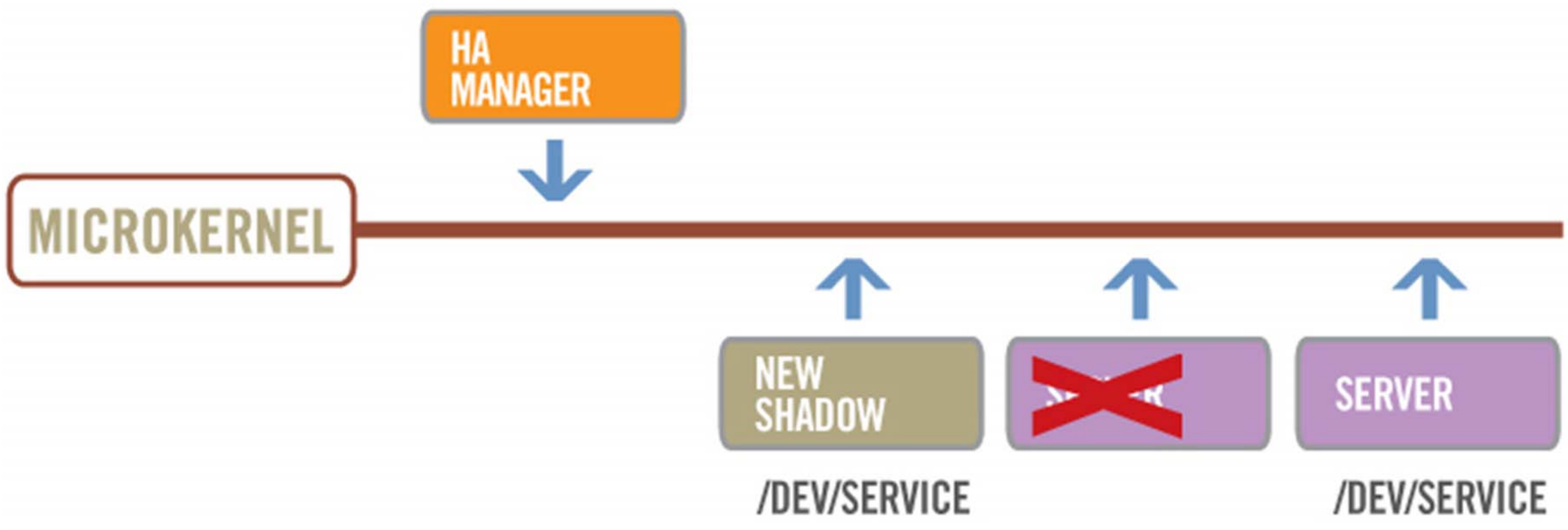
- A second “shadow” server runs

HAM Example



- A second “shadow” server runs
- If a fault occurs, new clients use the shadow server

HAM Example



- Start a new “shadow” server

QNX Certification

- ❒ **Several certified variants for use in products with high criticality and low tolerance for failure**

safety
rtos에서 제일 중요한 요소
rtos 오류시 (예를 들어 자동차의 급발진) 형사 책임을 프로그래머가 받을 수 있기 때문에, 국제 표준을 지킬 경우 안전을 보장해주기 위해서 반드시 지켜야 한다.
국제 표준 존재
IEC6150 (전자) ISO26262 (자동차) ASIL - 아식 레벨이 존재해서 각각의 등급을 판단. DO-178B (항공기)

Operating System	Certification/Compliance
QNX OS for Automotive	ISO 26262 ASIL D
QNX OS for Medical	IEC 62304
QNX OS for Safety and Security	Common Criteria ISO/IEC 15408 Evaluation Assurance Level (EAL) 4+ and IEC 61508 Safety Integrity Level 3 (SIL 3)
QNX OS for Security	Common Criteria ISO/IEC 15408 Evaluation Assurance Level (EAL) 4+
QNX OS for Safety	IEC 61508 Safety Integrity Level 3 (SIL 3)

Linux for Real-Time?

❒ Linux is a general purpose OS

- Aimed at high performance and stability

❒ Old Linux did not support

- Full preemption and bounded latency

❒ Current Linux supports

✓ Deterministic scheduler (SCHED_FIFO and EDF)

✓ Priority inheritance mutexes

✓ Lockable memory (disabling demand paging)

✓ High resolution timers

? Bounded latency inside the kernel

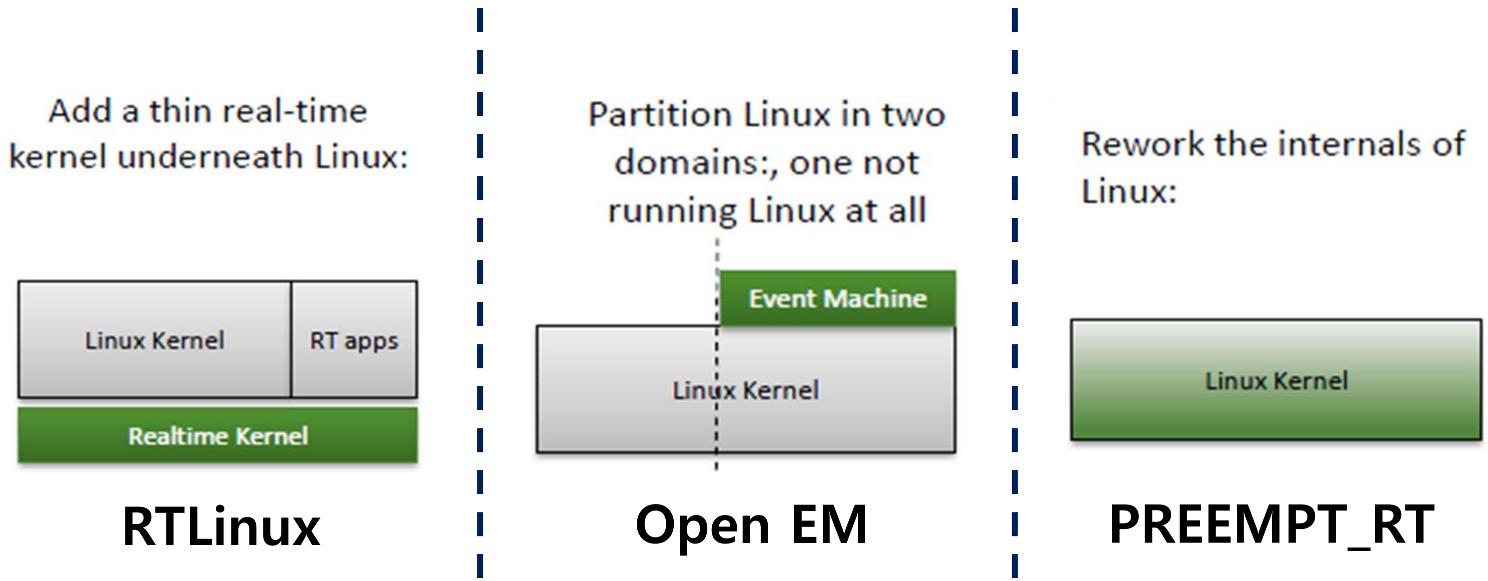
? Boot-up time

linux는 rtos가 아니다.
1. interrupt bound 제공 x
2. fully preemptive 만족

Linux Variants for RT

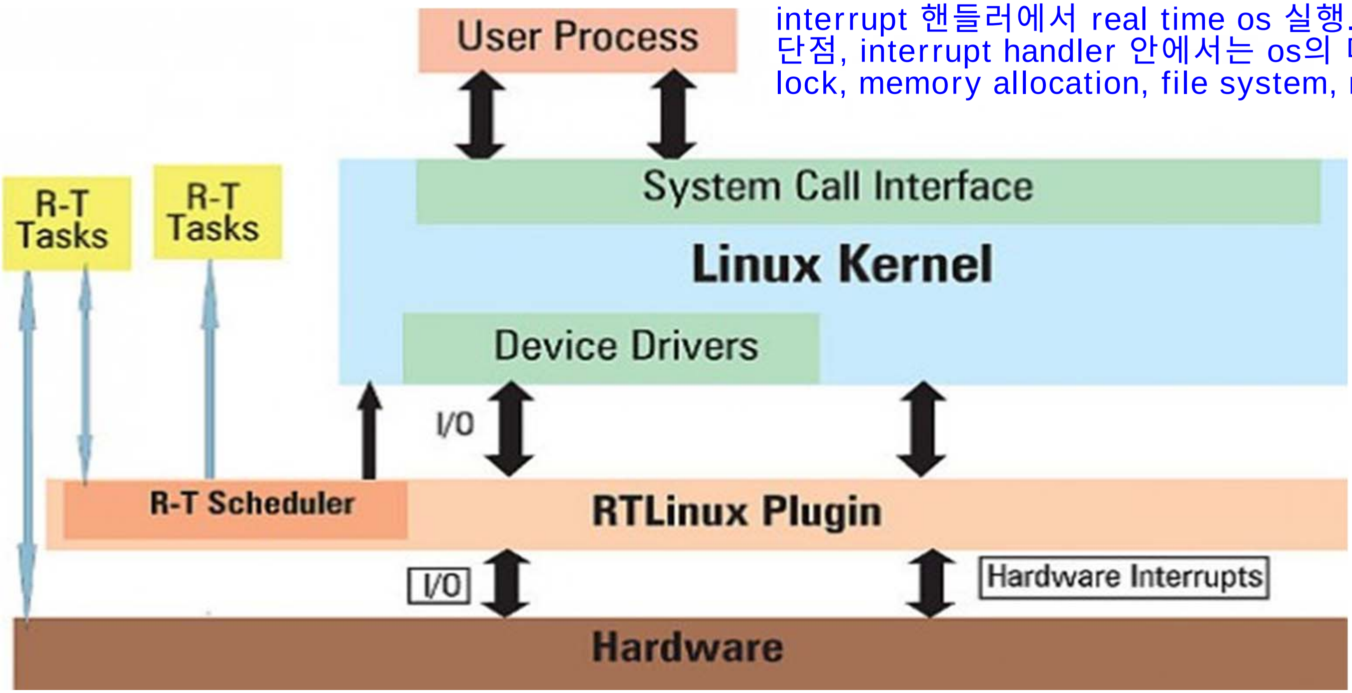
❒ Extensions for real-time systems

- RTLinux
- Open Event Machine
- PREEMP_RT patch



RTLinux

RTLinux
interrupt 핸들러에서 real time os 실행.
단점, interrupt handler 안에서는 os의 다른 기능을 사용 불가능.
lock, memory allocation, file system, network등등 사용 불가능



- ❒ Developed by Victor Yodaiken, Michael Barabanov, Cort Dougan as a commercial product at FSMLabs
- ❒ Wind River Systems acquired FSMLabs in February 2007, but ended it in 2011

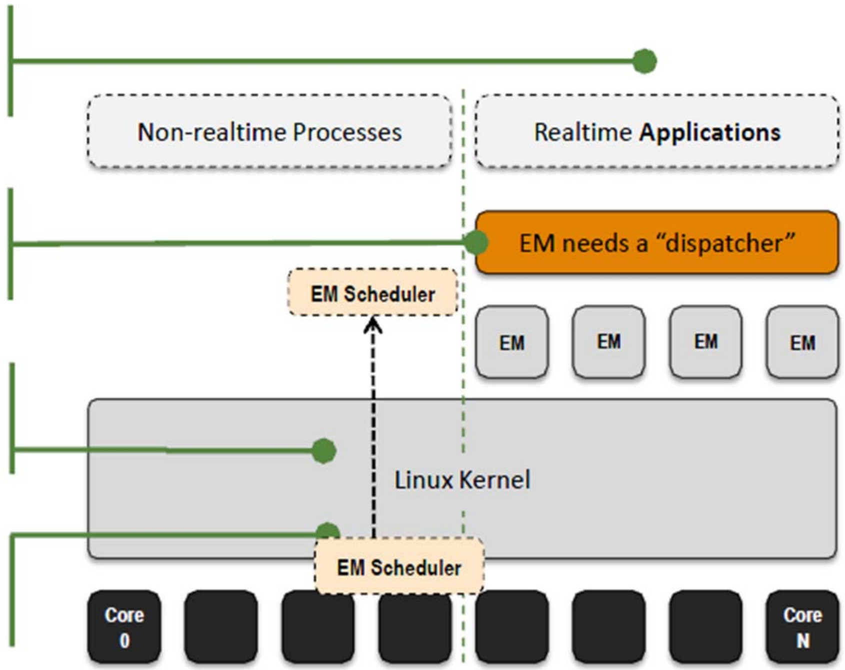
Open Event Machine

EM partitions the system into one RT domain and one non-RT domain

EM uses an event-based model, a run-to-completion model, for individual "context-less" work packages (NO threading or OS model)

Non-essential Linux processes and interrupts are migrated away from the EM cores

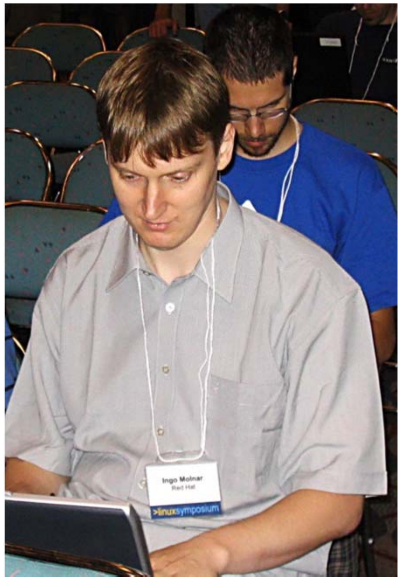
EM does not need a special interrupt handling model. Needs a "scheduler" in either Linux partition OR in HW

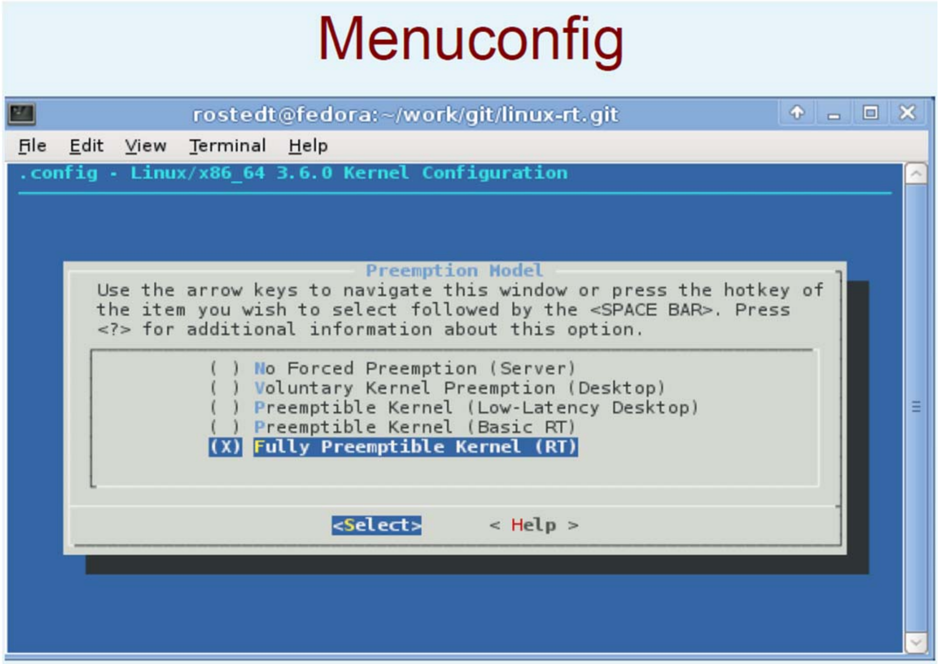


❒ Open EM has been chosen as a starting point for the OpenDataPlaneTM (ODP) initiative by Linaro, the not-for-profit engineering organization developing open source software for the ARM® architecture

PREEMPT_RT

- PREEMPT_RT has merged into the mainline kernel
  - Since August 2006 by Ingo Molnar *et al*.
  - https://www.kernel.org/pub/linux/kernel/projects/rt/





PREEMPT_RT

❒ Features

- Preemptible critical sections
- Preemptible interrupt handlers
- Priority inheritance for in-kernel spinlocks and semaphores
- Deferred operations
- Some latency-reduction measures

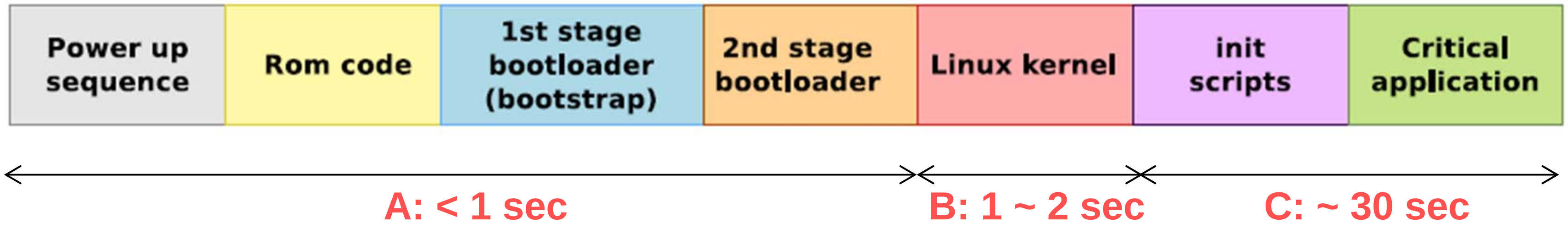
❒ Work is still going on

Linux Boot-up Time

❒ Generic boot sequence

A. Firmware initialization phase
B. Kernel initialization phase
C. User Space initialization phase

rtos
boot-up time 부팅 시간이 중요
why? 기존에 전자적으로 처리가 되어서 처리 속도가
빠른 것에 사람들이 익숙해져서 처리가 느리면 유저
입장에서 불평을 느낀다.



❒ Typical boot time

- < 60 seconds: desktops, set top boxes, GPS devices, …
- < 30 seconds: smartphones
- < 5 seconds: smart TVs

Boot-up Time Reduction

❒ General techniques for boot-up time reduction

- Optimize each job of the boot sequence
- See backup slides for details

❒ Snapshot-based techniques

- Save RAM content to nonvolatile storage before power-off
- Upon power-on, restore RAM state

❒ Limitations of snapshot approach

- Stability problem
- Retake snapshot every time power is turned off
- Take snapshot only at “stable” state
- Android adds another SW layer, increasing snapshot size

GPL License Issue

- GPL guarantees end users the freedoms to use, study, share (copy), and modify the software
  - Written by Richard Stallman of the Free Software Foundation (FSF) for the GNU project

GPL
- 영업 코드 공개해야해서 비즈니스 할때 문제된다.
- 파생된 모든 코드는 GPL이다. library를 쓰는 것도 파생되는 거라서 GPL이다,
- 안드로이드의 경우 커널(GPL) - 안드로이드(GPL) - 앱 (GPL) 따라서, 앱 GPL이라서 모든 코드 공개해야한다.

- Restriction
  - Modified or derived code must be released under GPL

- LGPL (Lesser GPL)

LGPL
- library 써도 GPL에 포함되지 않는다.
- 따라서, 안드로이드 앱의 library 경우 LGPL이라서 앱 소스 코드를 공개안해도 된다.

  - Library linking is allowed without releasing the code
  - Android uses Bionic libc derived from BSD libc to isolate apps from GPL and LGPL

Multitasking without OS Support

Four Approaches to Multitasking

- Four approaches
  - Polling-based approach (PA)
  - Event-based approach (EA)
  - Schedule-based approach (SA)
  - Thread-based approach (TA)

- The first three can be implemented without operating system's support

- The above approaches can be combined
  - PA + EA, PA + SA, EA + SA, PA + TA …

Polling-based Approach

❒ Extremely simple

- **No preemption and no context switch**

```
void main(void)
{
    while(1) {
            if (check_A) {
                    service_A();
            }
            if (check_B) {
                    service_B();
            }
            if (check_C) {
                    service_C();
            }
    }
}
```

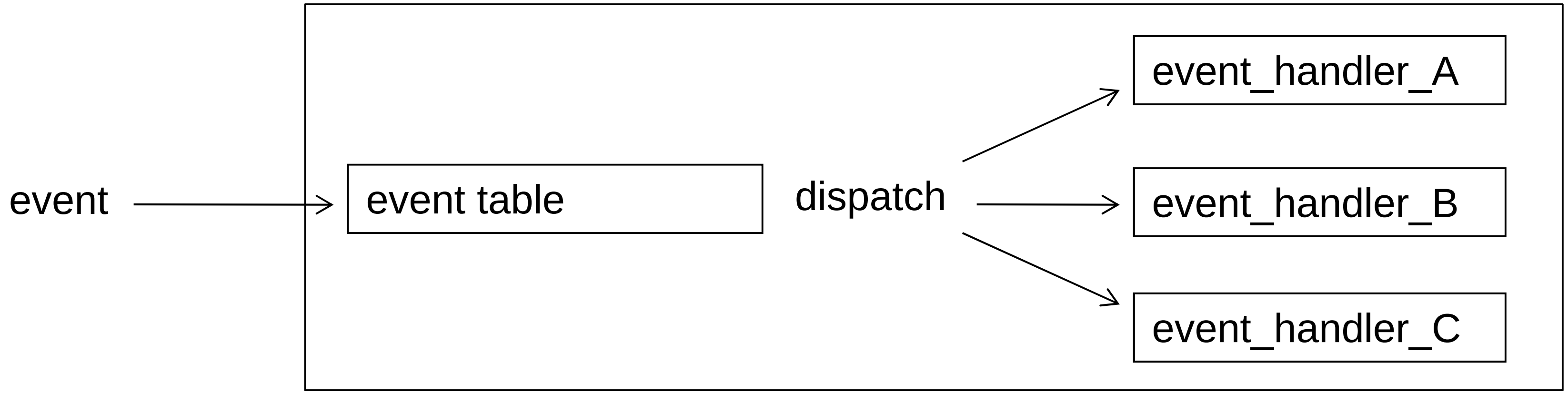
Problems with PA

- ❒ **PA does not support multi-rate systems**
  - **Voice recorder samples microphone at 20kHz, samples switches at 15Hz, and updates display at 4Hz**
- ❒ **Polling frequency is limited by the time required to execute the loop**
  - **We may obtain more performance by testing more often**
    - A/B/A/C/A/B/A/C …
- ❒ **Waiting time may be very long**
  - **In the worst case, we need to wait for all services to be completed (no preemption)**
- ❒ **The architecture is fragile** 부서지기 쉬운
  - **Adding a new service will affect timing of all other services**
  - **Changing rates is tedious and difficult**

Event-based Approach

❒ A typical approach for interrupt handling

- **Possible to execute periodic tasks by using timer events**
- **Response time is short**



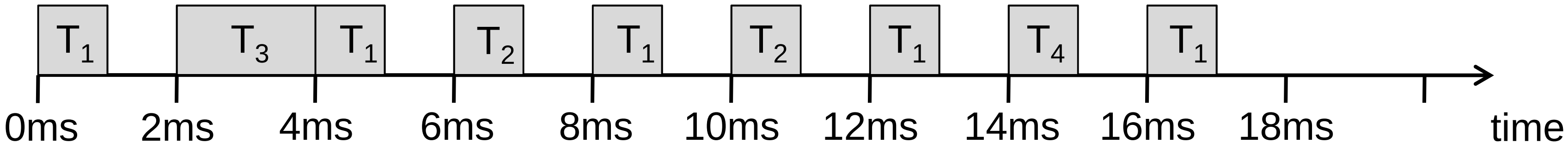
Problems with EA

- ❒ **Waiting time may be long**
  - We need to wait for the current job to be completed (no preemption)
- ❒ **It is hard to control the execution rate of each task**
  - It may depend upon the frequency of event occurring
- ❒ **Hard to apply a priority scheme**
  - But some hardware supports priorities

Schedule-based Approach

- Basically meant for real-time periodic task scheduling
  - Off-line schedule creation
  - On-line task dispatching according to the schedule
  - Uses periodic timer interrupts (time-triggered)

- Four task scheduling: T_i(period, execution time) =calendar scheduling
  - $T_1(4, 1)$, $T_2(10, 1)$, $T_3(20, 2)$, $T_4(20, 1)$



Some Issues with SA

- **Advantages**
  - **Deterministic behavior**
  - **Amenable to timing analysis**
    말을 잘 듣는
  - **Jitter control**

- **We need an algorithm for off-line schedule creation**
  - **RM (rate monotonic)**
    - Tasks with small periods get high priorities
  - **EDF (earliest deadline first)**
    - Tasks with early deadlines get high priorities

- **We need WCET (worst-case execution time) for each task**
  - **Schedule must be based on WCETs**
  - **If not, tasks may overrun**

Cyclic Executive Example

```
#define MAX 10
int schedule[MAX] = {1,3,1,2,1,2,1,4,1,0}; <- task 번호
int tick = 0;

/* timer interrupt every 2ms */

void Timer_ISR()
{
      switch (schedule[tick]) {
              case 0: break;
              case 1: execute_task_1(); break;
              case 2: execute_task_2(); break;
              case 3: execute_task_3(); break;
              case 4: execute_task_4(); break;
      }

      tick = (tick + 1)%MAX;
}
```

Thread-based Approach

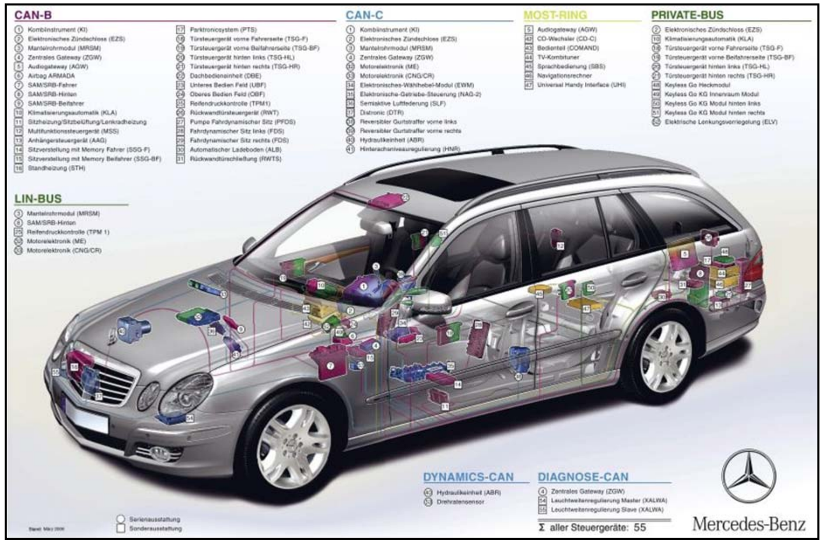
programmer가 직접 context switch timing&code 구현해야한다.

- ❒ Good for managing the inherent concurrency of an application
- ❒ Easy to apply a priority scheme or real-time scheduling policy
- ❒ Response time can be short
  - Preemption is supported
  - Blocking can be allowed in a thread, but there is no blocking for the entire application

AUTomotive Open System ARchitecture

Software Crisis?

- ❒ **Complexity: a lot of ECUs, buses, and gateways**
- ❒ **Productivity: poor reusability and composability**
- ❒ **Diversity: diverse OEMs and their suppliers**



- **55 ECUs**
- **7 Buses of 4 types**
- **4 Gateways**
- **More than 10 million LOC**

AUTomotive Open System ARchitecture

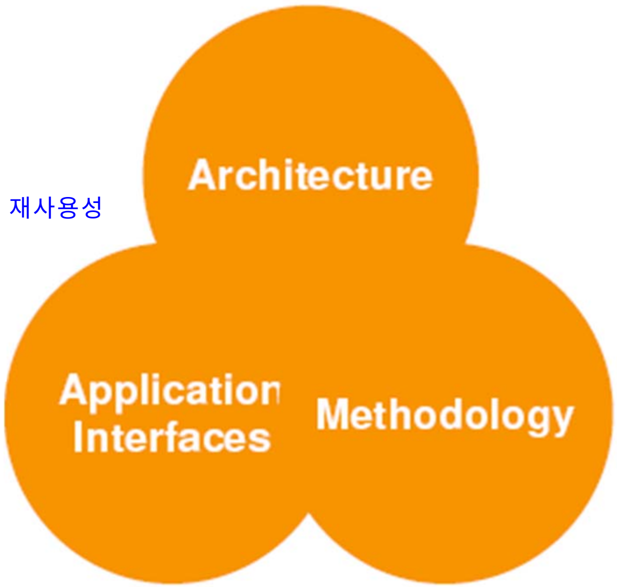
AUTOSAR - 자동차 S/W convention

❒ Standardized system S/W architecture

- OSEK kernel
- OS services and H/W abstractions

❒ Component-based application S/W 재사용성

- Standard component interfaces
- Communication middleware

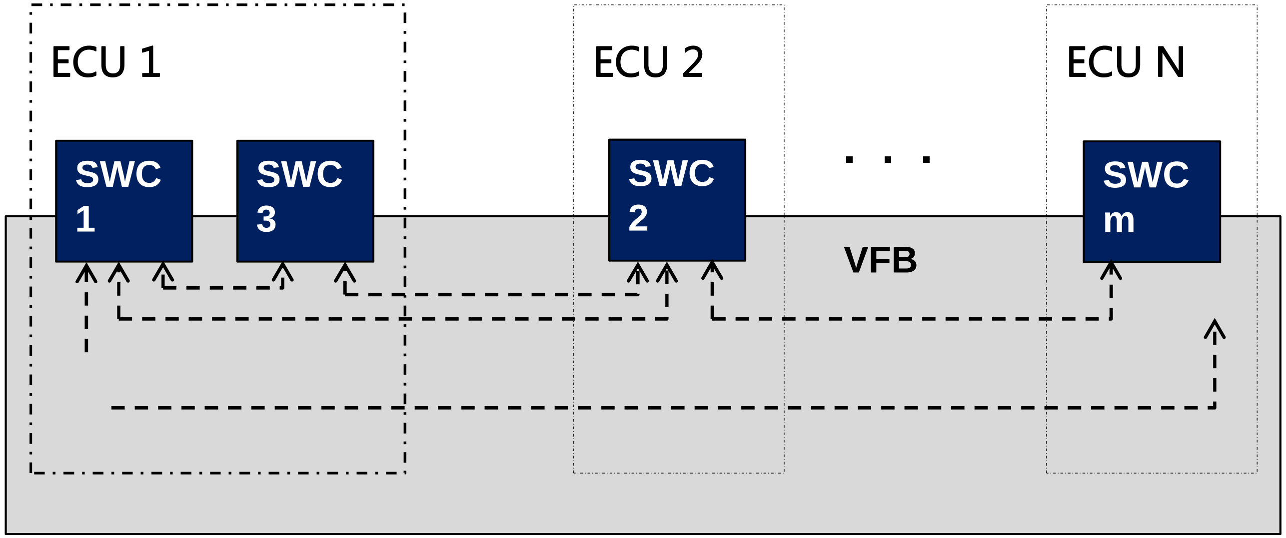


❒ Model-based development

- Authoring tools and XML formats
- Automatic code generation

SW Components and Virtual Functional Bus

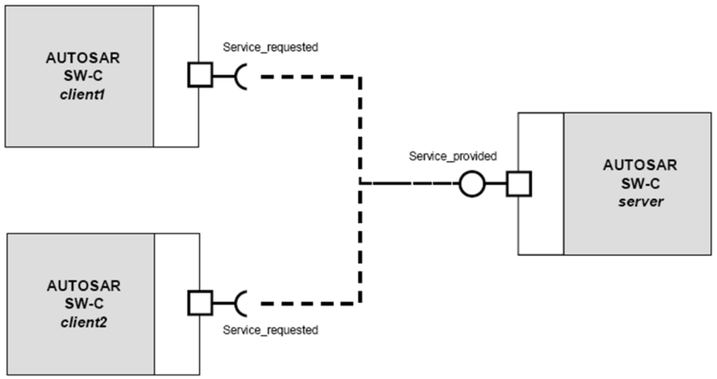
- ❒ **SWC (Software Components)**
  - Standardized interfaces
  - Reusability and composability
- ❒ **VFB (Virtual Functional Bus)**
  - Separation between software components and infrastructure



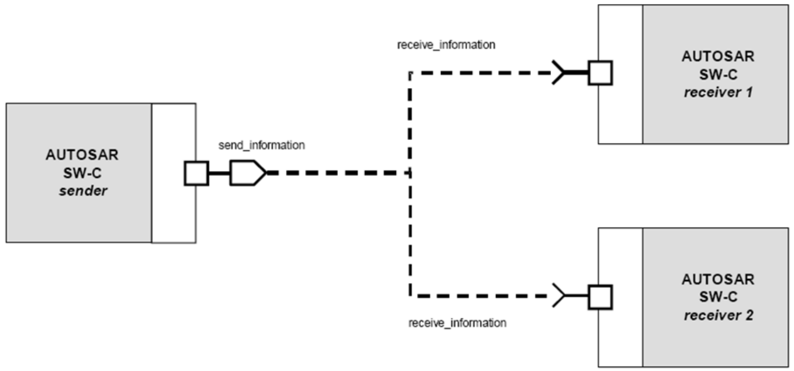
SWC and VFB

❒ Two types of communication

- **Client-Server Communication**
- **Sender-Receiver Communication**



Client-Server Communication



Sender-Receiver Communication

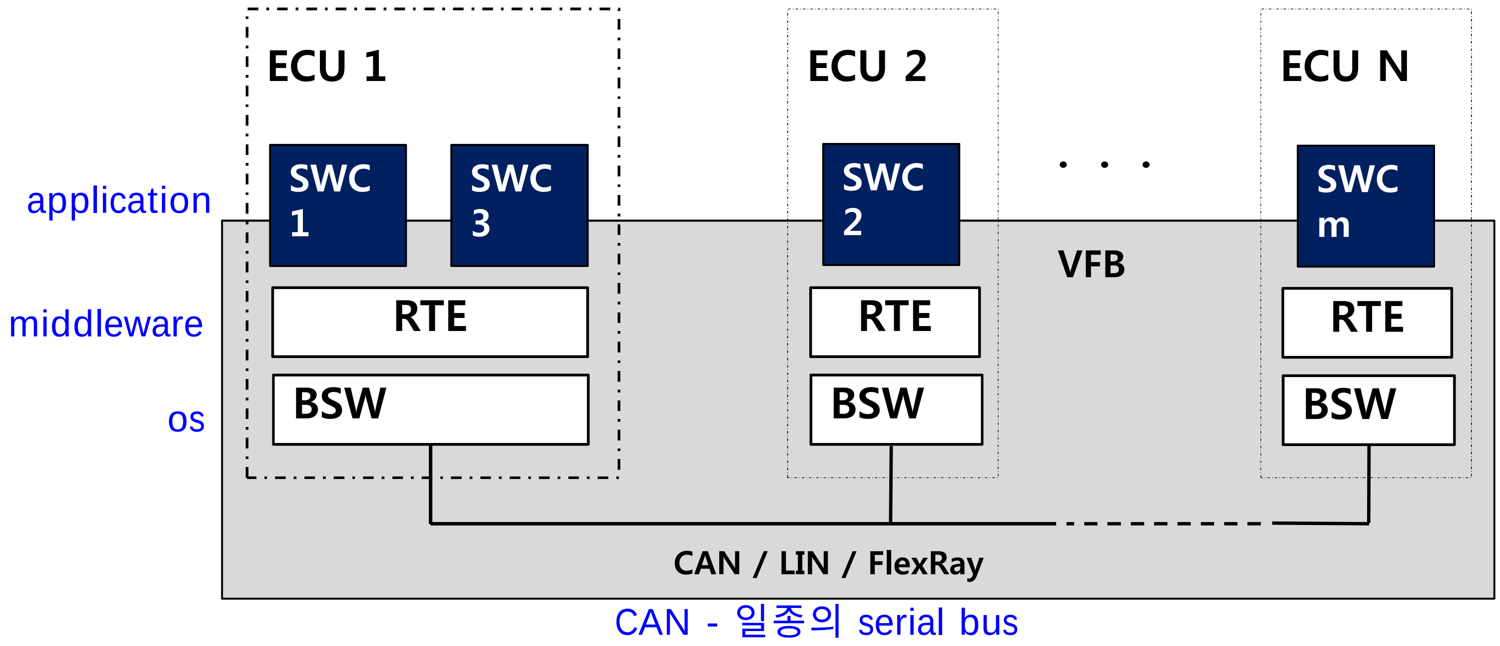
Runtime Environment and Basic SW

❒ RTE (Runtime Environment)

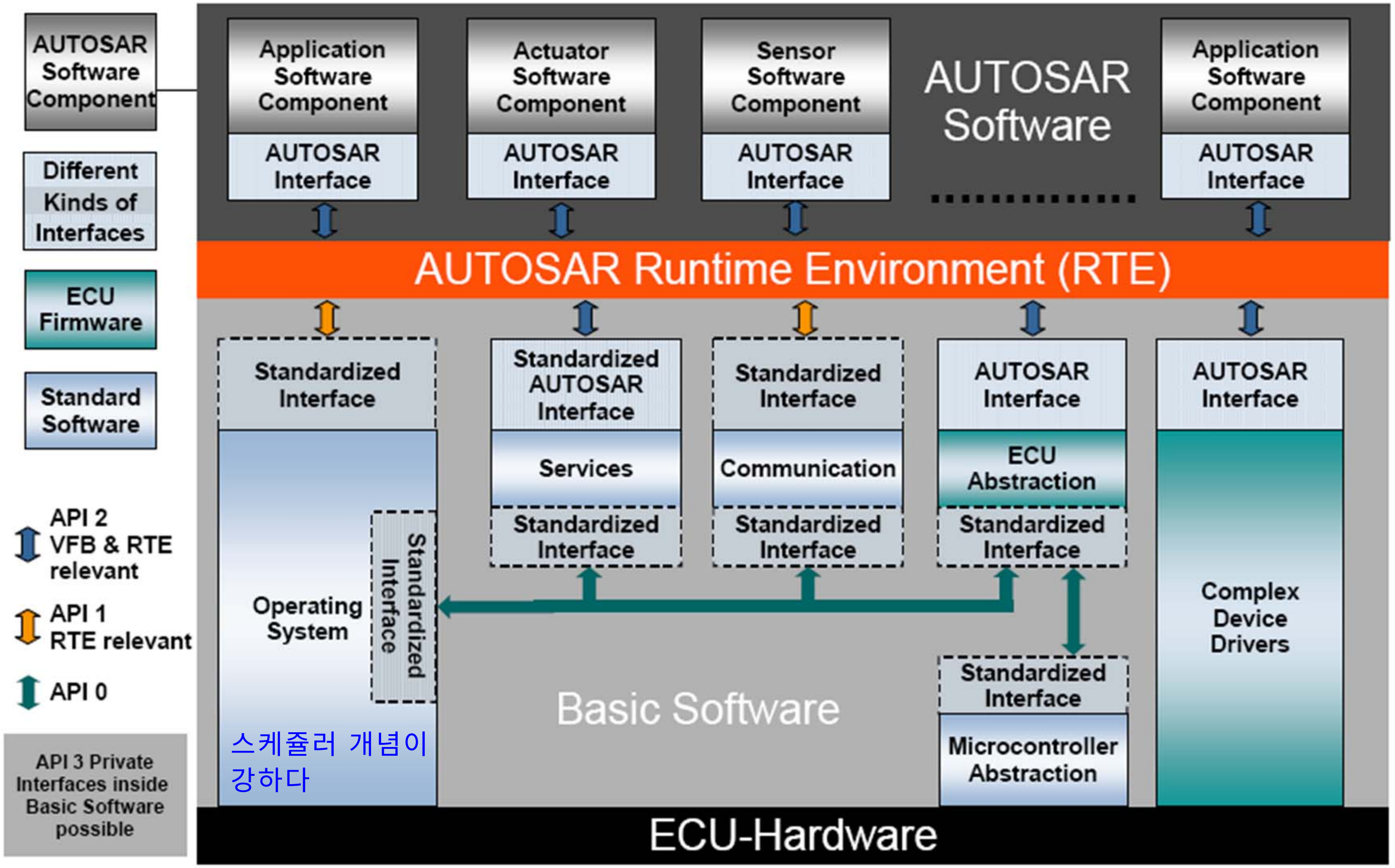
- Middleware for inter- and intra-ECU communication

❒ BSW (Basic Software)

- Traditional OS services



Layered Architecture



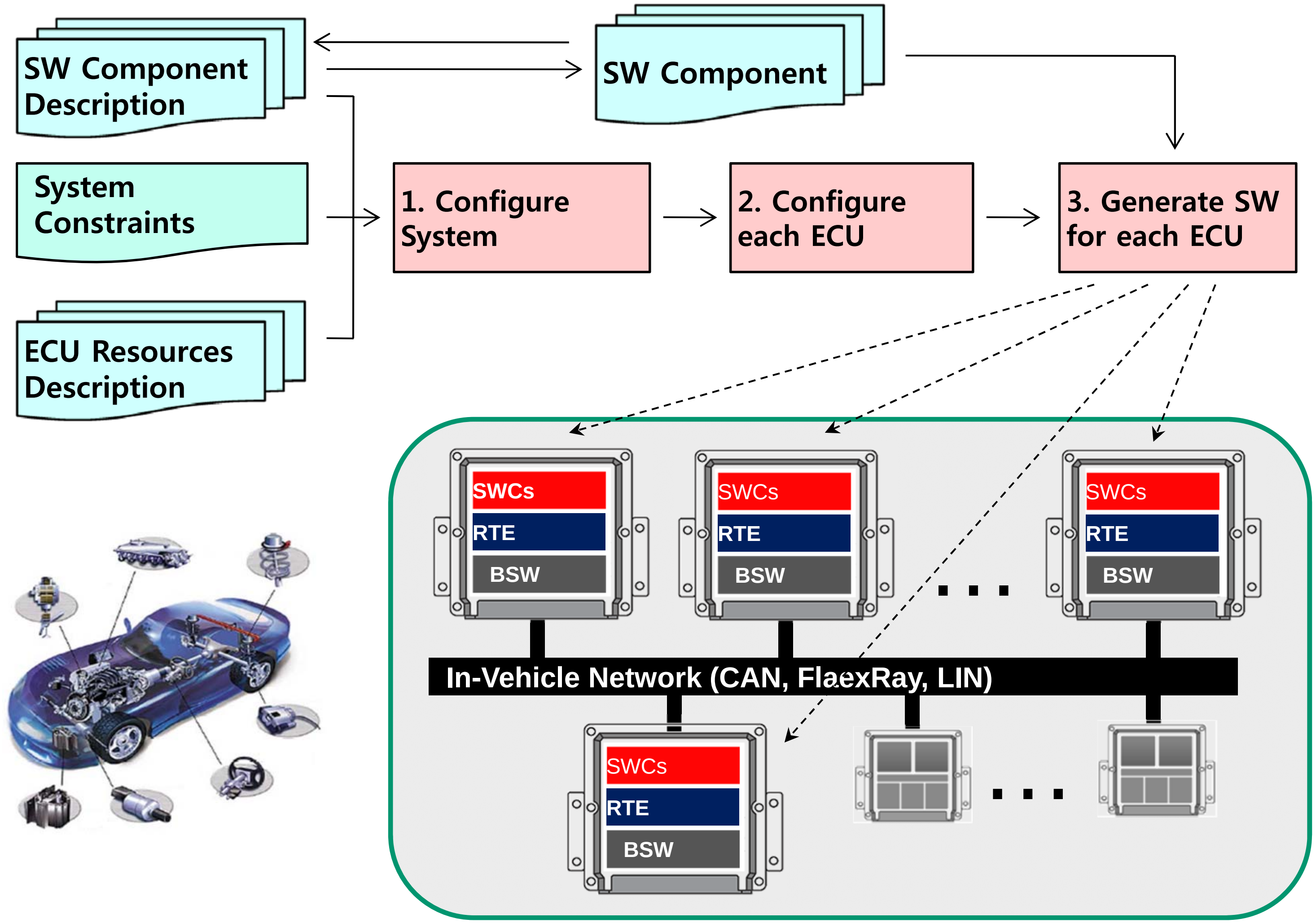
microprocessor
- cpu

microcontroller ~ SOC
- cpu+ram+통신

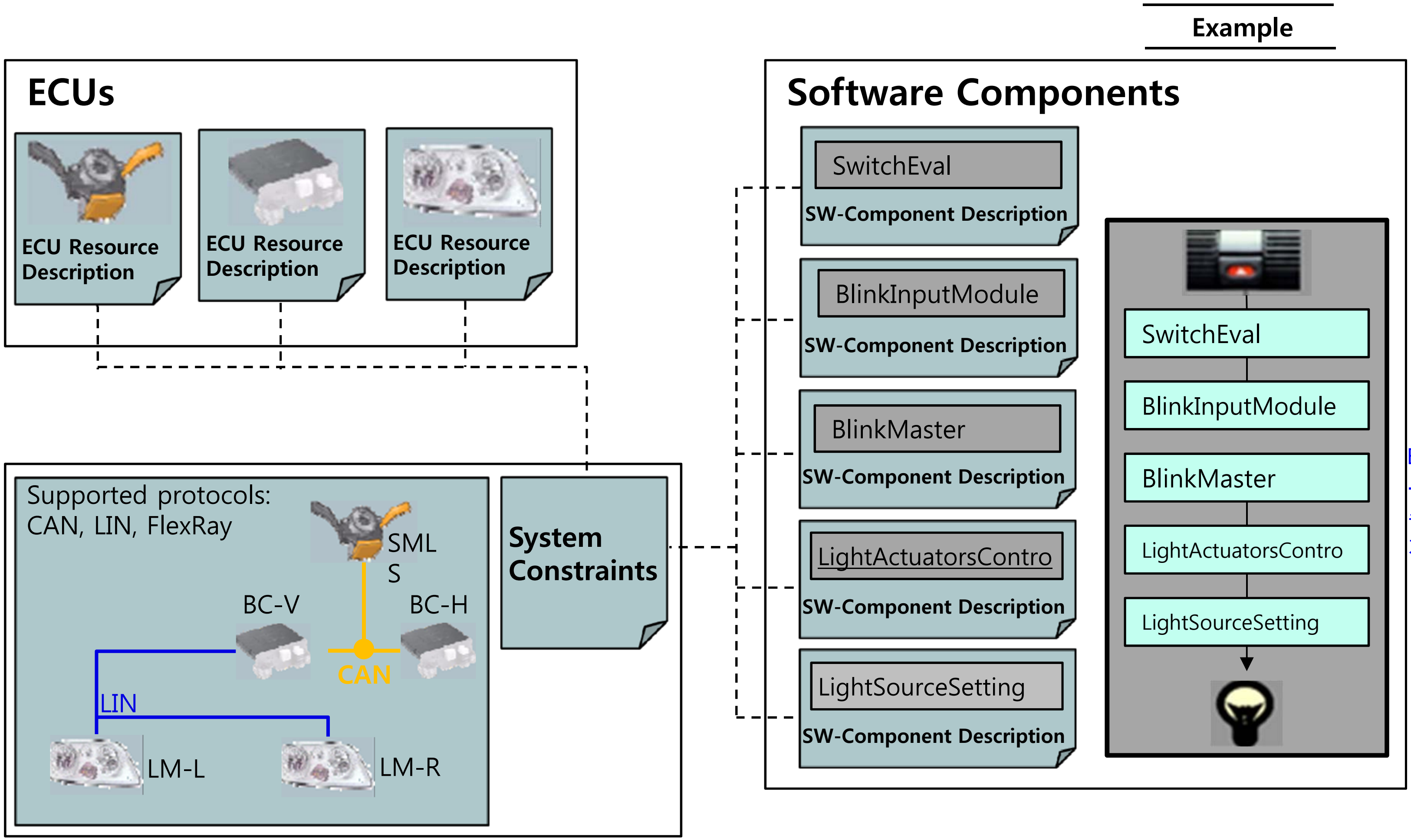
Methodology

대표 자동차 s/w 회사
1. vector tool
component <- 프로그래머가 s/w 작성
BSW RTE
2. eb tool

=> BSW/RTE가 자동으로 정해져서
compile 후 binrary
executable 만듬



Input Descriptions



자동차 회사 hierarchy
조립업체
ex) 현대차
|
부품업체
티어 1 현대 모비스,만도
|
티어 2 SW,POWER 생산
| 등등등..

BMW,폭스바겐
- AUTOSAR 표준을 맞춰서 수출해야 가능 ! S/W 자체 개발이 현실적으로 어렵다.

Input Descriptions

SW Component Description

- General characteristics (name, manufacturer, etc.)
- Communication properties:
  - p_ports
  - r_ports
  - interfaces
- Inner structure (composition)
  - sub-components
  - connections
- Required HW resources:
  - processing time
  - scheduling
  - memory (size, type, etc.)

ECU Resource Description

- General characteristics (name, manufacturer, etc.)
- Temperature (own, environment, cooling/heating)
- Available signal processing methods
- Available programming capabilities
- Available HW:
  - uC, architecture
  - Memory
  - Interfaces (CAN, LIN, MOST, FlexRay)
  - Periphery (sensor/actuator)
- SW below RTE for microcontroller
- Signal path from Pin to ECU-abstraction

System Description

- Network topology
  - bus systems: CAN, LIN, FlexRay
  - connected ECUs, Gateways
  - power supply, system activation
- Communication (for each channel)
  - K-matrix
  - gateway table
- Mapping / Clustering of SW components

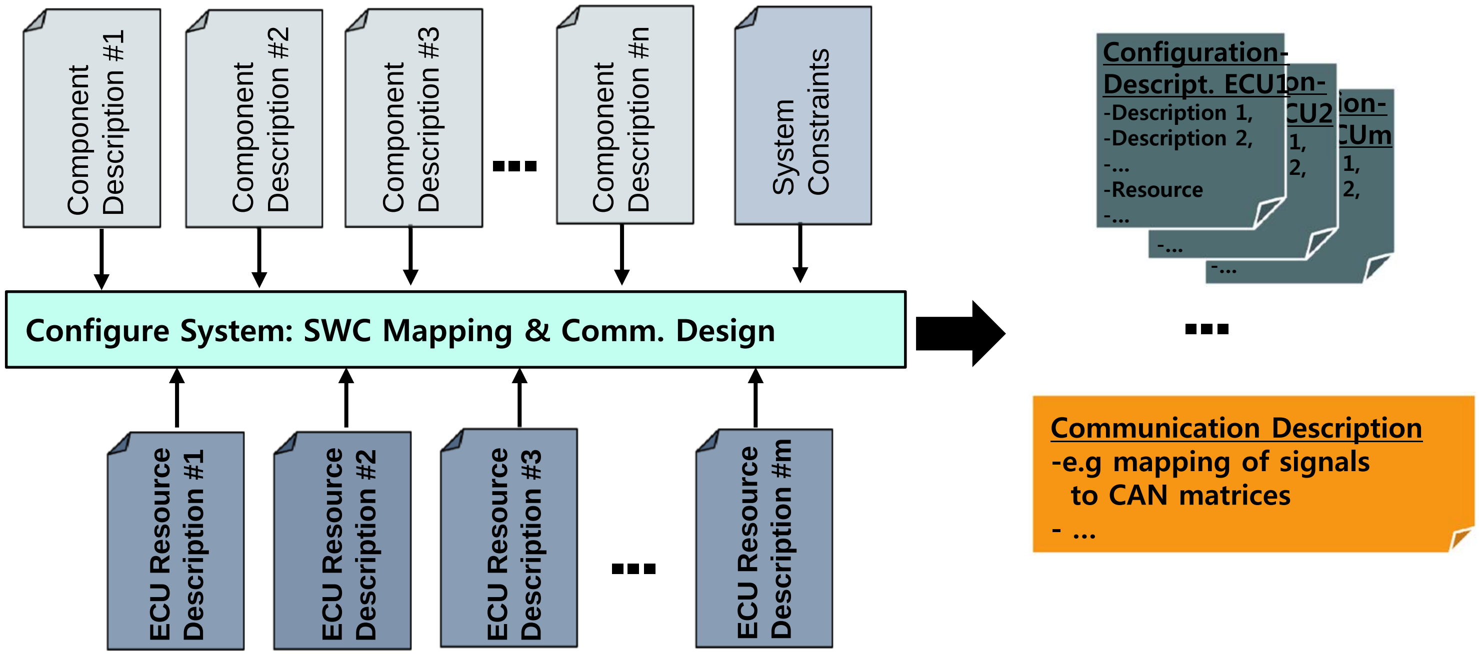
System Configuration

❒ SWC mapping

- Maps the software components to the ECUs

❒ Communication design

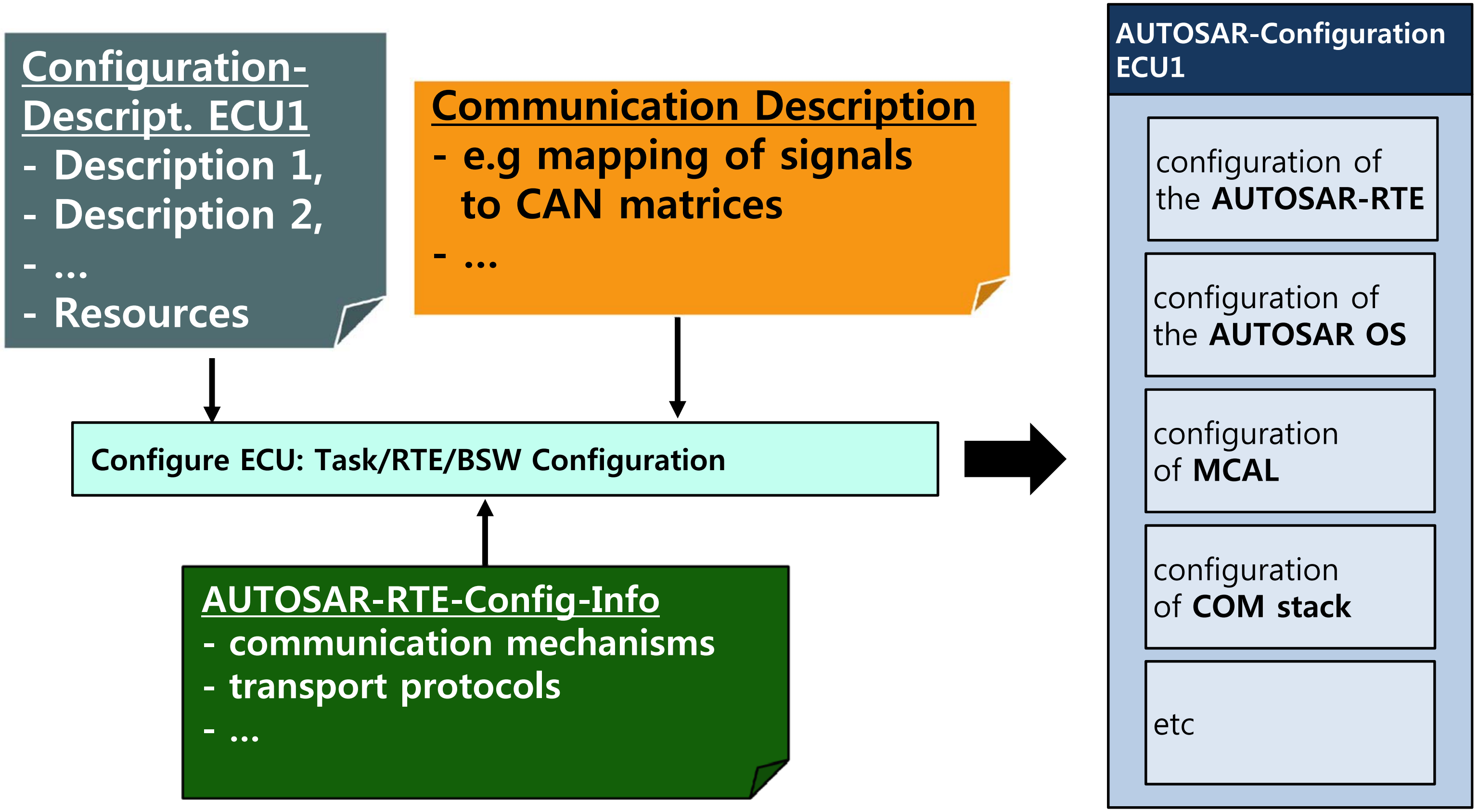
- Completely describes the frames running on the networks and the contents and timing of those frames



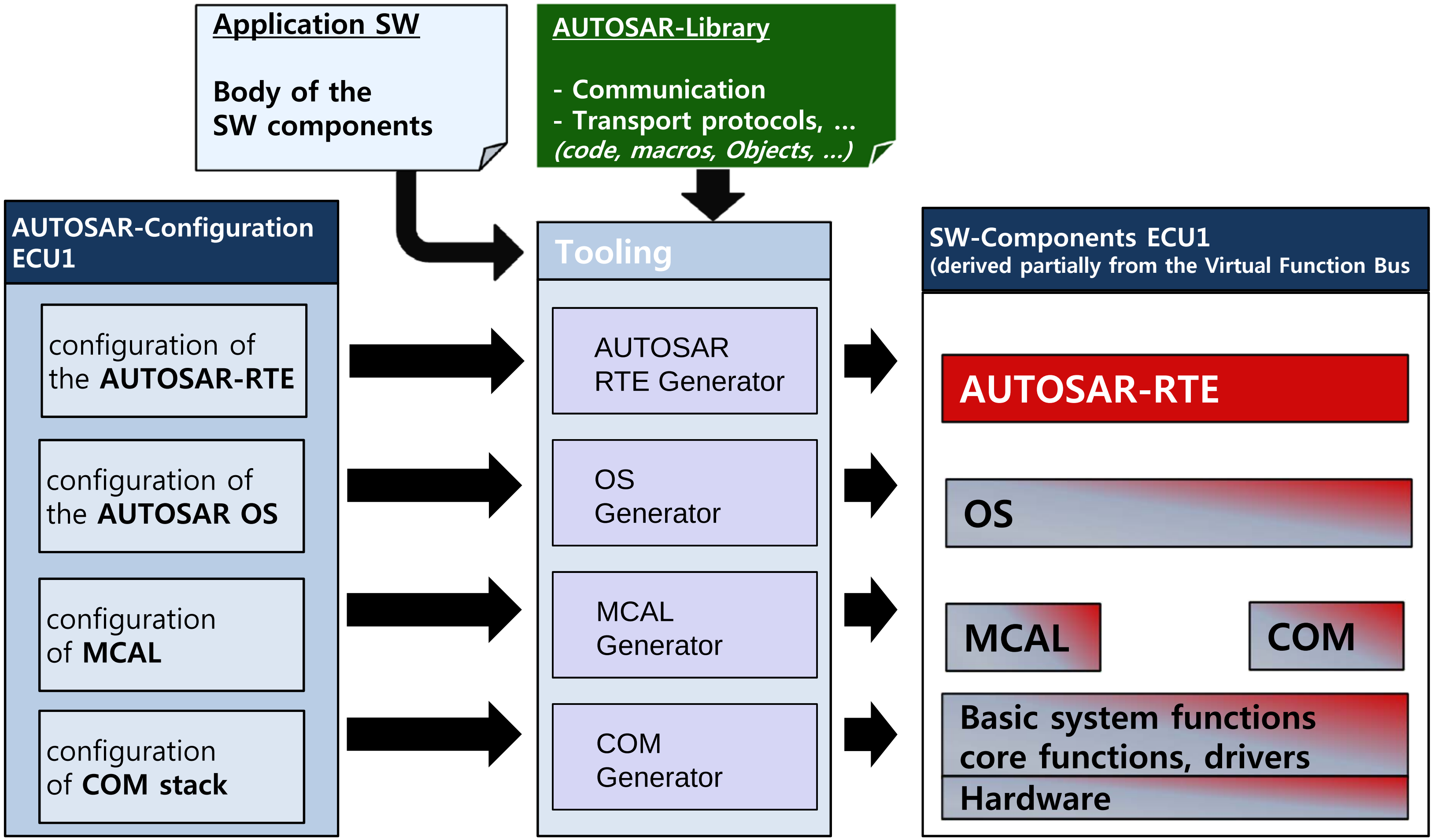
ECU Configuration

component안에 독립적으로 실행될수 있는 코드 - Runnable <- os(scheduler)가 독립적인 thread로 실행될수 있게 해준다.

- ❒ **Runnable-to-task mapping and task scheduling**
- ❒ **RTE and BSW configuration**



Software Generation



Workflow and Tools

Work	Tools
1. Control Function Design	• MATLAB, Simulink, Stateflow (Mathworks) → MIL (Model-in-the-Loop) simulation support
2. Code Generation	• Real-Time Workshop Embedded Coder (Mathworks) • TargetLink (dSPACE)
3. System Architecture Design	• SystemDesk (dSPACE) → SIL (SW-in-the-Loop) simulation support • DaVinci Developer (Vector Informatik) • RTA-RTE (ETAS) • Volcano Vehicle System Architect (Mentor Graphics)
4. BSW Configuration and Generation	• EB tresos Studio and AutoCore (Electrobit) • MICROSAR (Vector Informatik)
5. Experimenting, Testing and Debugging	• ControlDesk (dSPACE) • NUnit (Vector Informatik) • DaVinci Component Tester (Vector Informatik) • EB tresos Inspector (Electrobit)

matlab 선언형 구조로 code translation이 상대적으로 쉽다.