- **Embed**

  it means that something firmly and deeply into something equipment and software embedded in system to carry out specific behavior.

  It requires

  Must not stop during running (fault-tolerant) / Must be restored when error happens / Must be run at low performance if it needs (degradation in errors) / Must respond in real-time / Must respond within a given period (deterministically) / Must be low-power consumption / CPU must be run seamlessly in small memory, low frequency / Must be run seamlessly in bad condition

  That is,

  Optimal design to meet specific functions / Small, light in general / Low power consumption / Typically runs in severe environment

- **Real-Time (RT) System** is time deterministic system related to processing any task.

- **Hard Real-Time System (HRT)**

  Task must be done in time. Ex) Nuclear power plant, Detection of fire outbreak, Aircraft, Space shuttle, etc.

- **Soft Real-Time System (SRT)**

  Task may not be done in time; however, result degrades after its deadline. Ex) PC, Information devices, Network devices, etc.

- **Processor**

  Most important part in embedded system. Many kinds of products released by silicon vendors.

- **Microprocessor**

  CPU core only including data part and control part.

- **Microcontroller**

  CPU core including memories of various sizes and types and peripherals.

- **Read Only Memory (ROM)**

  In case of power off, program and data in memory must not be disappeared.

  BIOS resides Erasable Programmable ROM.

- **Random Access Memory (RAM)**

  Used mainly to store large program and data.

  Dynamic RAM be used as mass storage.

  Static RAM has limited memory capacity. In case of high speed read/write requirement such as cache memory.

- **Polling**

  Microprocessor periodically checks the registers of the device. Simple implementations.

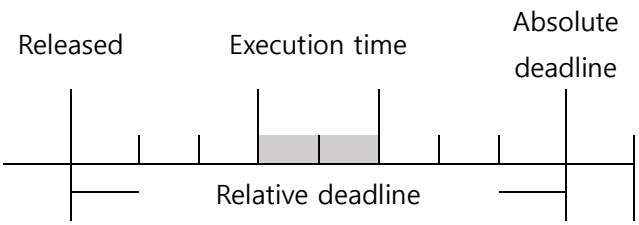
- **Interrupt**

  Device informs exception to the microprocessor. Efficient implementations.

- **USB**

  Serial or parallel interfaces have low performance to process a wide variety of multimedia data.

- **Real-Time Workload**



- **Real-Time task**

  Task is a sequence of similar jobs. Jobs repeat regularly.

$$\text{Periodic task}(p, e), \qquad 0 < e < p$$

  Where p is period of jobs, that is, inter-release time. e is execution time, that is, maximum execution time.

$$\text{Utilization } U = e/p$$

- **Admission Control**

  The basis of all HRT systems is that, for processes or threads to be created, need to pass admission control.

  In NRT systems admission control typically is concerned with starvation of processes due to lack of resources.

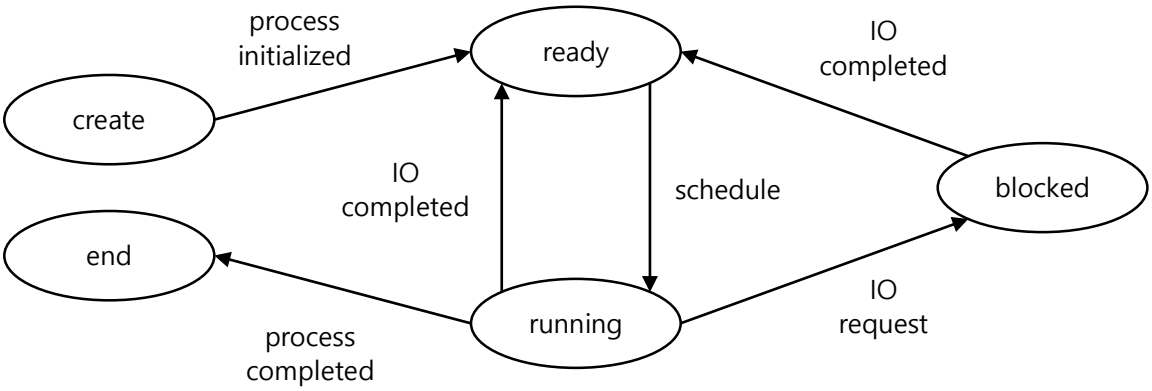
- **Dynamic HRT System**

  Dynamic schedule computed at run-time based on tasks really executing. 프로세스의 우선순위가 런타임에 바뀜.

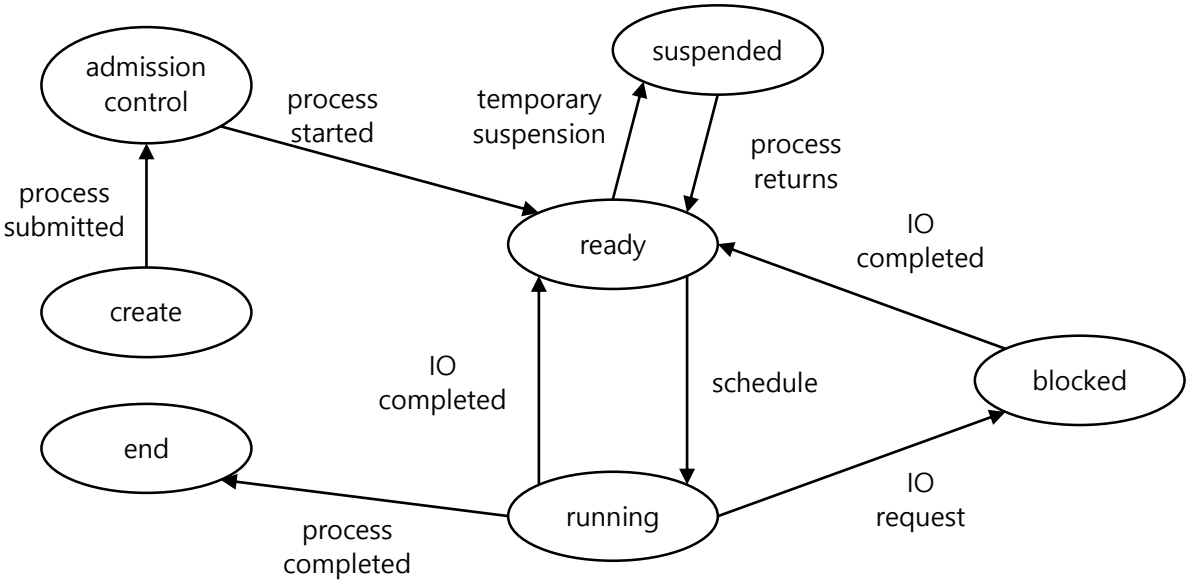
- **Static HRT System**

  Static schedule one at compile time for all possible tasks.

- **NRT System State Diagram**



- **RT System State Diagram**



- **Dynamic Scheduling Notation**

  Assume non-preemptive system / Tasks $\{T_i\}$ are periodic, with hard deadlines and no jitter / Tasks and instances are completely independent / Deadline = period ($p_i = d_i$) / WCET is known and constant / Context switching is free (zero cost)

- **Schedule-ability**

  Property indicating whether a real-time system can meet their deadlines.

- **Rate Monotonic (RM)**

  Optimal static-priority scheduling. It assigns priority according to period. A task with a shorter period has a higher priority. Executes a job with the shortest period.

- **Response Time**

  Duration from released time to finish time.

$$r_i = e_i + \sum_{T_k \in HP(T_i)} \left\lceil \frac{r_i}{p_k} \right\rceil \cdot e_k$$

  where $HP(T_i)$ is a set of higher-priority tasks than T_i.

  Real-time system is schedulable under RM if and only if $r_i \le p_i$ for all task $T_i(p_i, e_i)$.

  or if

$$\sum U_i \le n\left(2^{1/n} - 1\right)$$

  Where U is utilization and n is number of tasks.

  Ex) $T_1(4,1), T_2(5,1), T_3(10,1)$

$$\sum U_i = \frac{1}{4} + \frac{1}{5} + \frac{1}{10} = 0.55$$

$$3\left(2^{1/3} - 1\right) \cong 0.779$$

  Thus, $\{T_1, T_2, T_3\}$ is schedulable under RM.

  Utilization Bound is

$$U(n) = n\left(2^{1/n} - 1\right)$$

  If $\sum U_i < U(n)$ then the set of tasks is schedulable.
  if $\sum U_i > 1$ then the set of tasks is un-schedulable.
  if $U(n) < \sum U_i < 1$ then the test is inconclusive.

  Ex) $T_1(10,4), T_2(15,4), T_3(35,10)$

$$\sum U_i = \frac{4}{10} + \frac{4}{15} + \frac{10}{35} = 0.953$$

$$U(3) = 3\left(2^{1/3} - 1\right) \cong 0.779$$

  $U_{1+2} = 0.667$, schedulable. However, $0.779 < 0.953 < 1$. Therefore, inconclusive for t_3.

- **Earliest Deadline First (EDF)**

  Optimal dynamic priority scheduling. A task with a shorter deadline has a higher priority. Executes a job with the earliest deadline.

  Demand Bound Function $\mathrm{dbf}(t)$ is the maximum processor demand by workload over any interval of length t.

  Real-time system is schedulable under EDF if and only if

$$\mathrm{dbf}(t) \le t \text{ for all interval } t$$

  or if

$$\sum U_i \le 1$$

- **RM vs EDF**

  RM is simpler implementation, even in systems without explicit support for timing constraints (periods, deadlines). Predictability for the highest priority tasks.

  EDF is full processor utilization. Misbehavior during overload conditions.

- **Priority Inversion**

  Assume that there are tasks that has

  $$\text{priority } p_L < \cdots \leq p_{M_i} \leq \cdots < p_H$$

  T_L locks mutex m.

  T_H preempt then require m but blocked

  T_{M_i}s continue to preempt.

  T_H fail.

- **Priority Inheritance**

  T_L get p_H when T_H blocked by T_L.

- **Deadlock on Priority Inheritance**

  T_L locks mutex m_1.

  T_H preempt then locks mutex m_2 and blocked by T_L.

  T_L requires m_2 but blocked. DEADLOCK!

- **Priority Ceiling**

  Assume that p_T is highest priority.

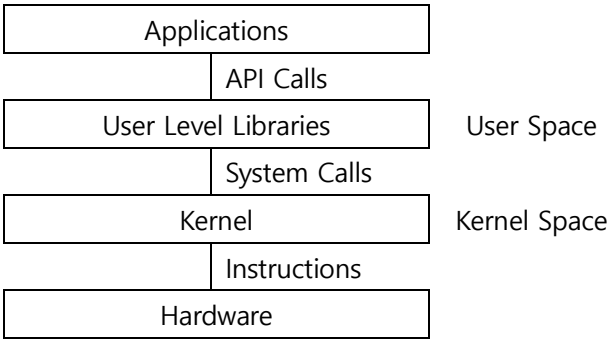
  T_L get p_T when locks mutex.

- **Virtualization**

  A frame work that combines or divides computing resources to present a transparent view of one or more environments.

  Ex) HW or SW partitioning / Partial or complete machine simulation / Emulation / Time-sharing / In general, can be M-to-N mapping. (M is real resources; N is virtual resources) / VM (M-N) / Grid Computing (M-1) / Multitasking (1-N)

- **Machines Stacked Architecture**



- **Possible Abstraction Levels**

  Instruction Set Architecture (ISA)
  Hardware Abstraction Layer (HAL)
  Operating System Level
  Library (user-level API) Level
  Application (Programming Language) Level

	ISA	HAL	OS	Lib	PL
Performance	1	4	4	3	2
Flexibility	4	3	2	2	2
Ease of Impl.	2	1	3	2	2
Degree of Isolation	3	4	2	2	3

(higher number is better)

- **Virtualized Machines Stacked Architecture**

- **Virtualization Types**

  - Full virtualization

    Virtualize without OS modification. Ex) VMWare

  - HW-assisted virtualization

    HW provides virtualization.

  - Partial virtualization

    Partially modifying feature of OS for virtualization.

  - Para-virtualization

    Specially modified OS for virtualization.

	Full	HW	Para
Performance	1	3	2
Flexibility	3	1	2
Ease of Impl.	1	3	2

(higher number is better)

- **Host Operating System**

  The one that is installed on your physical machine and runs VMware Workstation.

- **Guest Operating System**

  The virtual OS that gets installed on top of the Host OS.

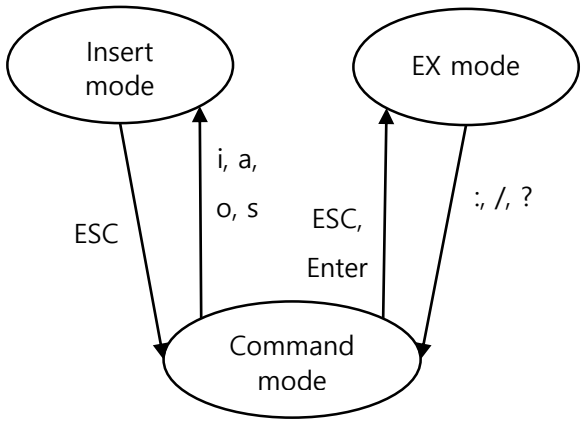
- **Networking Options**
  - Bridged networking

    Uses host NIC and VM gets its own IP address from host's DHCP server. 호스트와 다른 IP.
  - NAT networking

    Uses host NIC and shares IP address with host. 호스트와 같은 IP 사용.
  - Host-only networking

    VMware acts as a DHCP server and provides IP address to VMs. VMs can only communicate with each other and the host. 다른 VMware와 호스트 사이에서만 통신 가능.
- **Vi / Vim Editor**

  A classic text based editor on Linux system. It has powerful editing capabilities. Also, if using separate tools, it can easily trace source tree, code regions, etc.



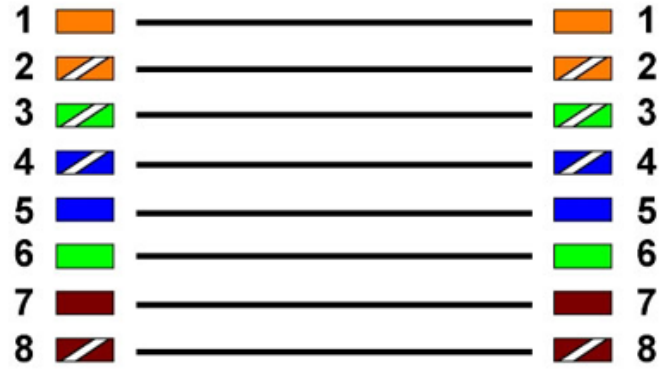
  Command mode: supplies a large set of single-key commands such as move, copy, delete, attach, etc.

  Insert mode: enter insert mode for editing file with the `i`, `a`, `o`, or `s` key

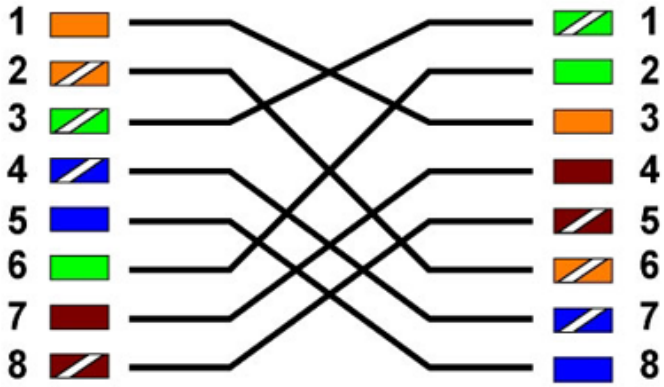
  EX mode: use ex mode for searching pattern, replacing strings, file open, file close, etc.
- **Development Environment of an Embedded System** is Host system, Back-end, Target system.

  `$ apt-get install libsub-dev`
- **Cross Compiler** compiles code in host for target system.
- **Cross Debugger** displays target state, allows target system to be controlled.
- **Ethernet Cable**

  Straight Through Diagram



  Crossover Diagram



- **Daemon**

  Computer program that runs as a background process, rather than being under the direct control of an interactive user.

  Typically daemon names end with the letter d. We can see what daemons are running by `pstree`. Adjust the running of daemons by system-config-services or `ntsysv`.
- **Stand-Alone Daemon**

  Each started via rc script than always running. Listens (binds) to the service port. User resources even when idle.

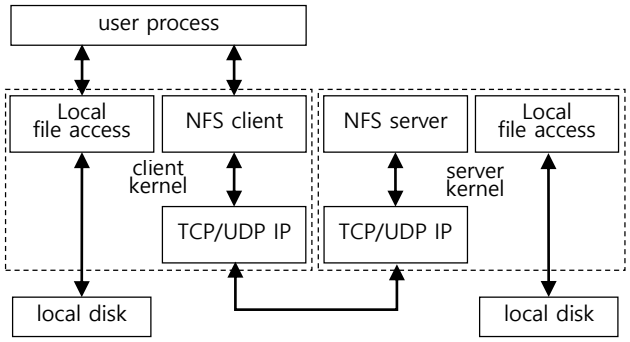
- **Super Daemon aka inetd**

  Listens to many ports. Starts daemon when request is received. Daemon shuts down when finished.

  `/etc/inetd.conf`
  `/etc/init.d/inetd start|stop|restart`

- **Network File System (NFS)** allowing a user on a client computer to access files over a computer network much like local storage is accessed.

- **NFS Structure**



- **NFS Installation**

  /[SRCDIR] [IPADDR]([OPT,]…)

  `SRCDIR` is directory to make NFS root.

  `IPADDR` is permitted IP address. It can be *.

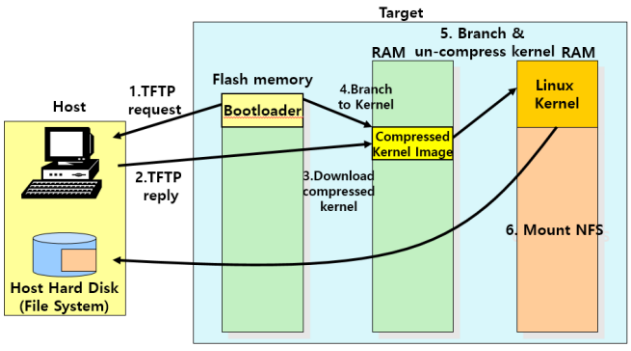
  `OPT` for…

  `rw` give read/write permission.

  `no_root_squash` admit root permission.
  `no_subtree_check`

  ```
  $ service nfs-kernel server start|stop
  $ mount –t nfs HOST_IP:REMOTE_DIR SRC_DIR
  $ unmount SRCDIR
  ```

- **Booting Sequences by TFPT/NFS**



- **TFTP Installation**

  ```
  $ apt-get install tftp-hpa tftpd-hpa openbsd-inetd
  $ vim /etc/default/tftpd-hpa
  $ service openbsd-inetd restart
  $ service tftpd-hpa restart
  ```

- **Boot Loader** is a computer program that loads an OS or some other system software for the computer after completion of the power-on self-test; it is loader for the OS system itself. [wikipedia]

- **Toolchain** is a set of programs which includes compiler, libraries and binary utilities, in order to generate source files into a binary file. Components are: GCC: compiler / binutils: assembler, loader and a utility to edit a binary file. / Glibc: libraries for compilation, including cross-compilation. / Linux kernel: Linux kernel source.

- **Loader** moves bits from non-volatile memory to memory and then transfers CPU control to the newly "loaded" bits (excitable).

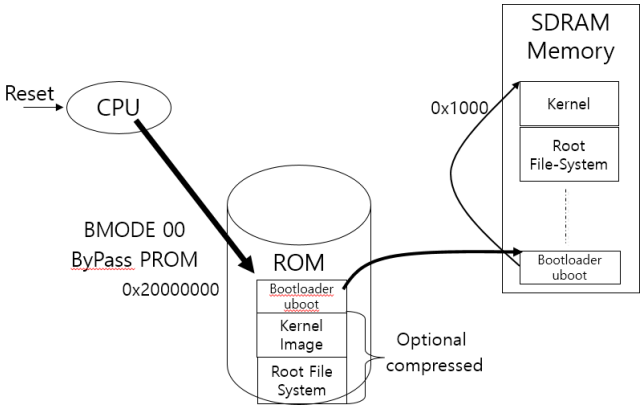
- **Bootloader** loads the "first program" (the kernel). Initializes CPU registers, device controllers and contents of the main memory. After this, it loads the OS.

- **Boot PROM**

  Persistent code that is "already loaded" on power-up.

- **Boot Manager** lets you choose the "first program" to load.

- **Loading through Das U-boot**



  CPU → move Bootloader in ROM to RAM → Bootloader config and execute kernel

- **Overall Flow of U-Boot**

1. [ASM] startup code (`cpu/pxa/start.s`)
   CPU initialization, sdram initialization, relocation to sdram, call `start_armboot()`
2. [C] code `start_armboot()` (`lib_arm/board.c`)
   `flash_init`, `eth_init` console initialization
3. `main_loop()` (`common/main.c`)
   Routine to handle a command autoboot
4. Linux booting (`lib_arm/armlinux.c`)

- **Task Performed at Boot Up**

1. Run diagnostics to determine the state of machine. If diagnostics pass, booting continues.
2. Runs a Power-On Self-Test (POST) to check the devices that the computer will rely on, are functioning.
3. BIOS goes through a preconfigured list of devices until it finds one that is bootable. If it finds no such device, an error is given and the boot process stops.
4. Initializes CPU registers, device controllers and contents of the main memory. After this, it loads the OS.
5. On finding a bootable device, the BIOS loads and executes its boot sector. In the case of a hard drive, this is referred to as the Master Boot Record (MBR) and is often not OS specific.
6. The MBR code checks the partition table for an active partition. If one is found, the MBR code loads that partition's boot sector and executes it.
7. The boot sector is often OS specific, however in most OS its main function is to load and execute a kernel, which continues startup.

- **U-Boot Commands: Information Commands**

`bdinfo` list board information on console.

`coninfo` list console information.

`flinfo` list flash memory information.

`iminfo` list application image information.

`help [CMND]` lists commands, for help on specific command type.

- **U-Boot Commands: Memory Commands**

`base` print or set base address

`crc32` calculate the crc32 checksum over an address range

`cmp ADDR1 ADDR2 SIZE` compare two memory ranges.

`cp SRC_ADDR DST_ADDR SIZE` copy memory.

`md [.b|.w|.l] ADDR SIZE` display memory

`mm [.b|.w|.l] ADDR` modify memory will prompt for new value.

`mtest [START [END [PATTERN]]]` simple memory test.

`mw [.b|.w|.l] ADDR VAL` memory write.

`nm [.b|.w|.l] ADDR` memory modify.

`loop [.b|.w|.l] ADDR NUM_OF_OBJ` infinite loop on address range.

- **U-Boot Commands: Flash Memory Commands**

`erase` erase flash memory.

`protect` enable or disable flash protection.

`mtdparts` define a Linux compatible Memory Technology Device (MTD) partition scheme.

- **U-Boot Commands: Execution Control Commands**

  `autoscr` run script from memory

  `bootm` moot application image from memory

  go start application at address

- **U-Boot Commands: Download Commands**

  `bootp` Boot image via network using BOOTP/TFTP protocol

  `dhcp` Invoke DHCP client to obtain IP/boot params

  `loadb` Load binary file over serial line.

  `laods` Load S-Record file over serial line.

  `rarpboot` Boot image via network using RARP/TFTP protocol

  `tftpboot` Boot image via network using TFTP protocol.

- **U-Boot Commands: Env. Variables Commands**

  `printenv` print environment variables.

  `saveenv` save environment variables to persistent storage.

  `setenv NAME VAL` set environment variable.

  `run` run commands in an environment variable

  `bootd` boot default, i.e., run bootcmd

- **U-Boot Commands: microSD(NAND)**

  `movi init` Initialize moviNAND and show card info

  `movi read u-boot|kernel ADDR` Read data from sd/mmc.
  `$ movi read u-boot S5P_MSHC2 50000000`
  Read U-Boot, and kernel image from SD and then load to SDRAM.

  `movi read rootfs ADDR [BYTE]` Read rootfs data from sd/mmc by size.
  `$ movi read rootfs S5P_MSHC2 50000000 100000`

  `movi write u-boot|kernel ADDR` Write data to sd/mmc.
  `$ movi read u-boot S5P_MSHC2 50000000`

  `movi write rootfs ADDR [BYTE]` Write rootfs data to sd/mmc by size.
  `

- **Kernel** is the central components of operating system. Process management, CPU scheduling, I/O control, etc. System management such as memory, file, and peripherals.

- **Monolithic Kernel**

  Entire operating system is working in kernel space. Access to kernel functionalities by using function call. Simple and effective, but porting and extensions are difficult.

- **Micro Kernel**

  Necessary functionalities are divided into small server modules that are essentially like daemon program.

  Function extension and reconstitution are easy, but multiple message transfer and context switching happens when using services.

  Inter Process Communication (IPC) is needed. There are two methods for transfer data; shared memory, message pass.