Lecture 8

Program Implementation in System Operating System

Process

Process implementation

Operating system (OS)

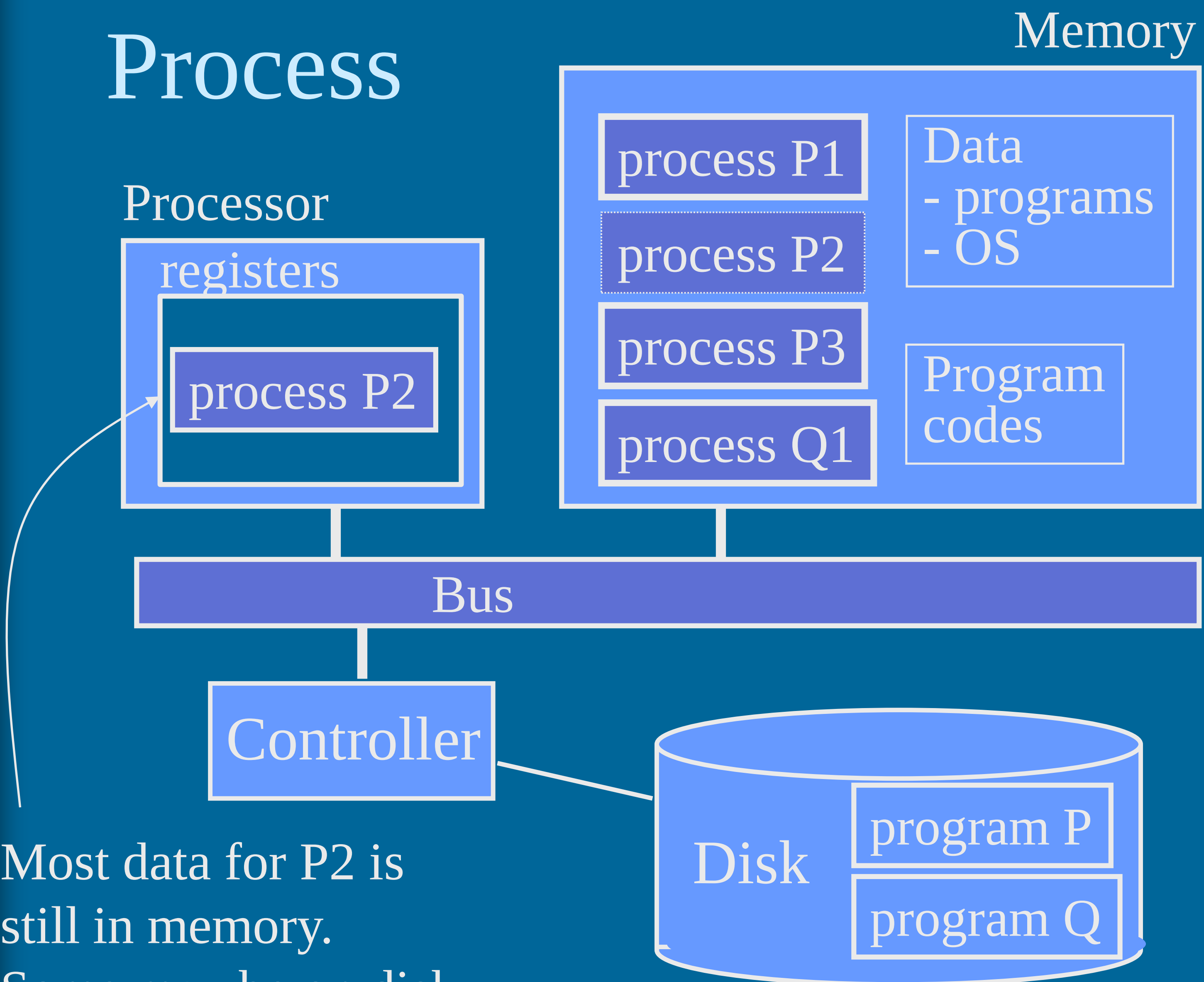
OS processes

I/O implementation with device drivers

Process = Program in System

- System may have ”at the same time” **many processes** from the same or different programs
  - User (human) point of view and time scale (1 min, 1 sek, 10 ms)
- The processor has **in execution only one** process at a time
  - Assume: 1 core processor
  - Hardware (processor, system) point of view and time scale (1 ns, 1 µs, 1 ms)
- **All other processes are waiting** for something
  - Processor? I/O? Message from another process?
  - Available memory space?

Process



Most data for P2 is
still in memory.
Some may be on disk.

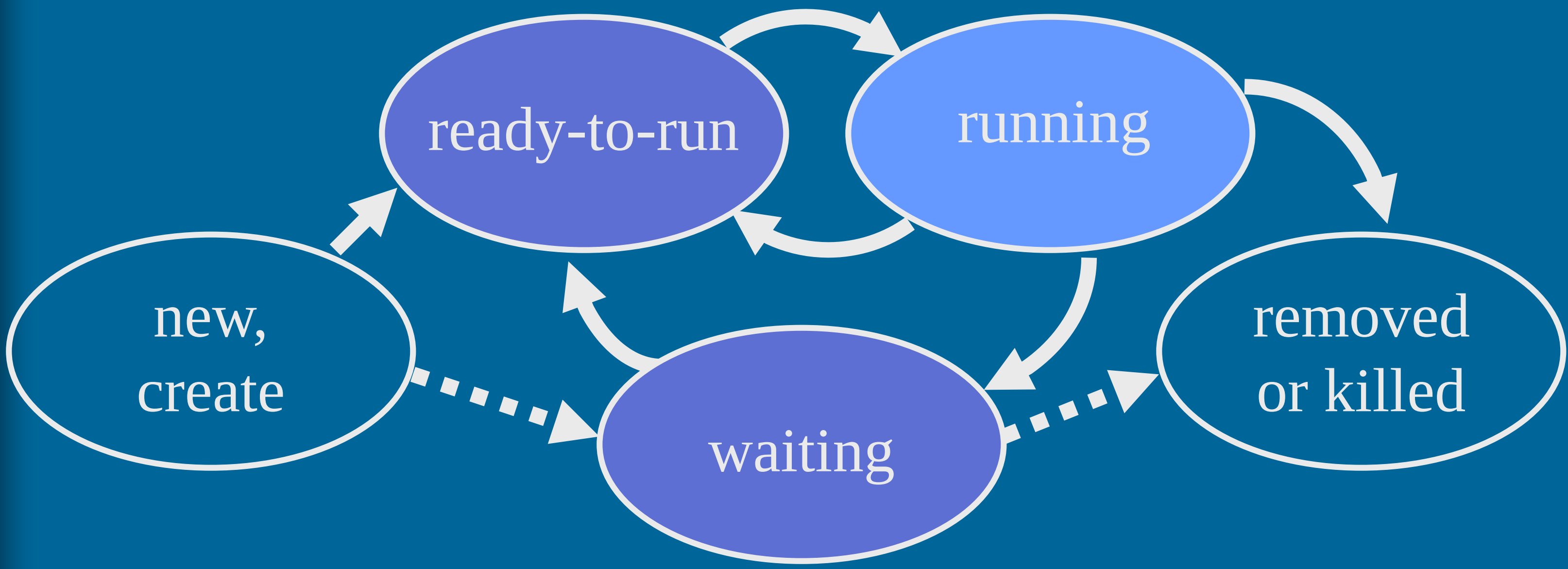
Process switch

prosessin vaihto

- Changing the process executing on the processor
- Happens quite often (in human time scale)
  - E.g., every 2000-3000 machine instruction executed?
  - E.g., 50-500 time per sec? Every 10 ms?
  - Why?
    - Current process cannot continue execution
    - Current process does not want to continue execution
    - OS decides, that it is time to change executing process
      - After OS got to execute via some interrupt
- Big operation – lots of copying data
  - How many instructions are needed?

50-500?
0?

5-state Prosess Model



- 5 states of a process
  - When does any give state change happen?
  - What event causes the state change?
  - What happens in the state change?
  - Who continues execution after the state change?

Discuss

Process Descriptor (PCB)

Process Control Block

prosessin kuvaaja

- Process id
- Priority for processor scheduling
- Process state and/or reason for waiting
- Processor context, saved while waiting
  - Work registers R0-R5, SP, FP, state registers, …
  - PC (address of next instr to execute)
    - Prosess switch occurs when this is set
- Interrupt handler addresses (unless default)
- Time slice (after which time loser CPU turn)
- Used memory areas, open files
- OS admin data (compute time, cpu time, etc.)

14023

143

R-to-R

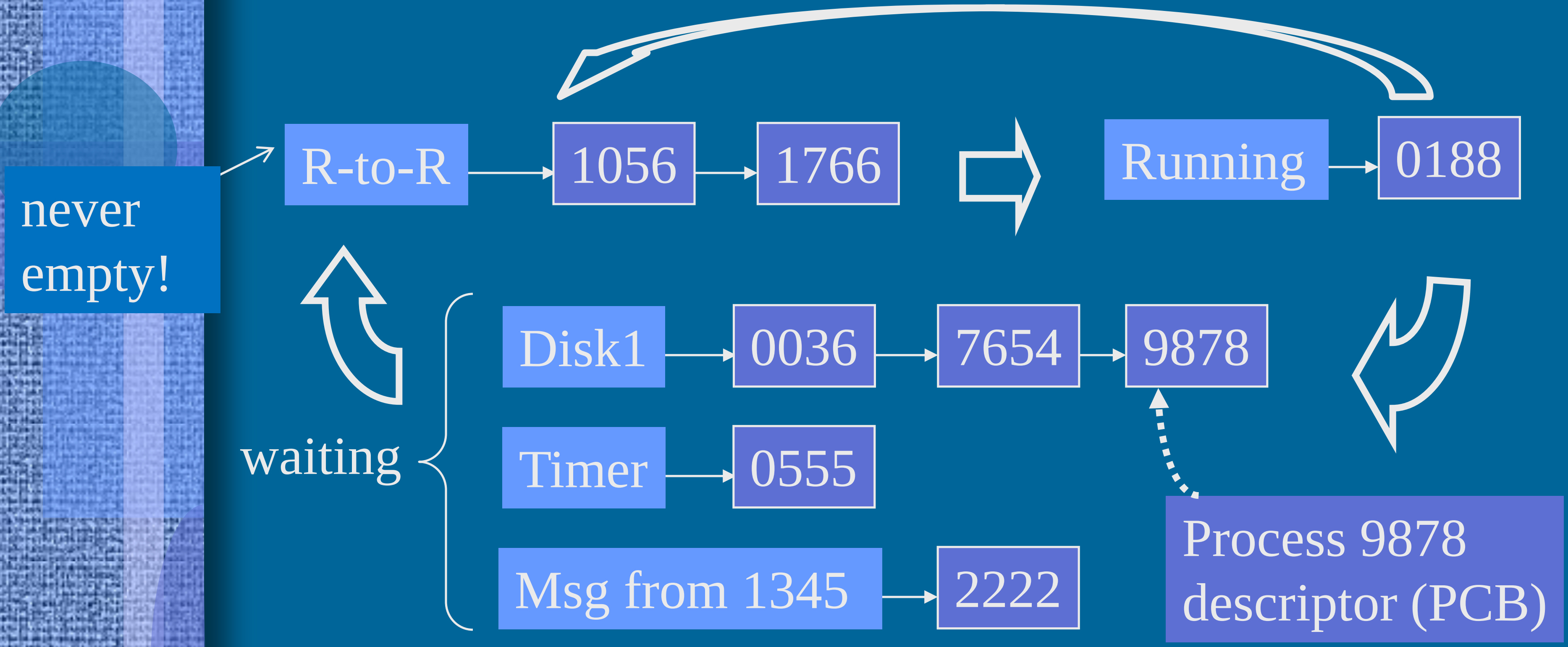
initially main {}

suoritin ympäristö

aika-viipale

Discuss

Processes in Queues



Scheduling:
select next process from Ready-to-Run queue and
move (dispatch) it to execute in CPU
(copy the **processor context** of selected process to CPU registers)

Process Switch

- Done by an OS routine executing in currently running process (e.g., called from interrupt handler)
- Save processor context of previous process to its PCB in memory (with normal memory store instructions?)
  – Registers, also PC (execution continues from here, if continues)
  – No need to do, if previous process is going to be terminated
- Load processor context of new process from its PCB
  – Load all (processor context) registers, last one is PC
- Execution of new process continues exactly where it was interrupted (or from 1st instruction if new process)
  – Same machine instruction, same execution environment
  – Usually in the middle process switch OS routine, which was called from interrupt handler
  – Next return from OS services and interrupt handler, and proceed with normal computation

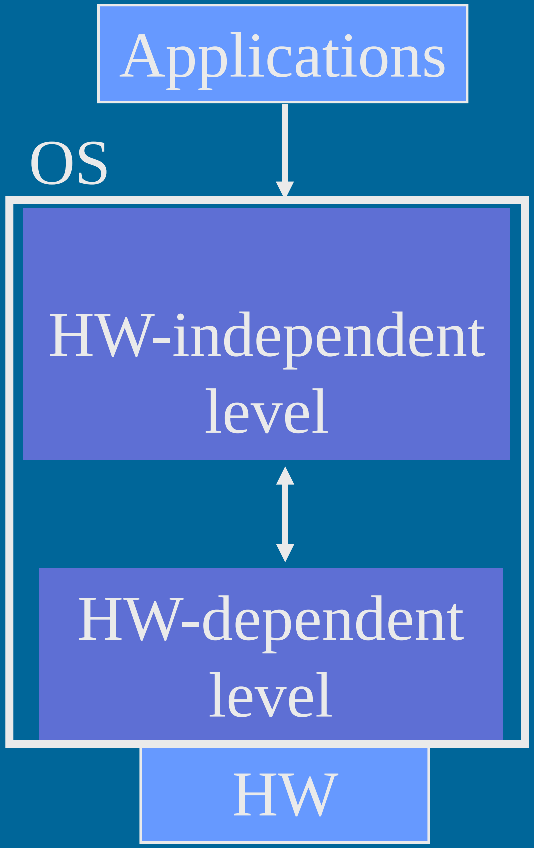
Process Priority

- Process importance in processor (not for I/O, e.g.)
  - E.g., small number $\Rightarrow$ large (better) priority (or vice versa)
- Each priority (class) has its own R-to-R queue
  - OS processes have higher priority than user processes
  - Real time processes have higher priority than OS processes
    - They must give OS some time to do its work!
- Priority may vary during process life time
  - Lots of CPU time $\Rightarrow$ lower priority
  - Long time in R-to-R queue $\Rightarrow$ higher priority
    - Process is moved to higher priority R-to-R queue?

Discuss

Operating System (OS)

- HW-independent application interface to HW
  - Makes it easier to use HW
  - Give fair service to all
  - Resource management and control
  - Applications are easier to implement and port elsewhere
- Resource management
  - See next slide



Resource Management and Control

- Process management, processor scheduling
  - Schedule CPU fairly, no-one waits forever
  - Critical processes are scheduled to run in time
- Memory management
  - How much main memory to each process?
  - Which memory areas are allocated to which process?
  - Easy use of shared memory areas and good data protection
- File management
  - HW and location independent use of files and devices
  - Easy use of shared files and good data protection
- Network management
  - HW independent use of networks
  - Easy use and good data protection

OS Structure

- Process management
- Memory management
- File management
- Device management
- Network management

OS Implementation

- Set of processes and/or subroutines
  - Processes live their own lives
    - User level process
    - Privileged process, root-process
    - E.g., device driver
  - Subroutines are executed interrupted process's environment (in privileged state?)
    - E.g., interrupt handler, device driver
  - Get control whenever needed
    - Subroutine calls, SVC, viestit
    - Timers and other interrupts
    - OS does <u>nothing</u>, unless code for it is in execution!

daemon
windows service

Discuss

Return from OS Service

Explicit OS-service request

- Subroutine calls
  - CALL → RETURN
- SVC
  - SVC → IRET
- Messages
  - message → reply message (sender waits for reply in RECEIVE op)

Implicit OS-service request

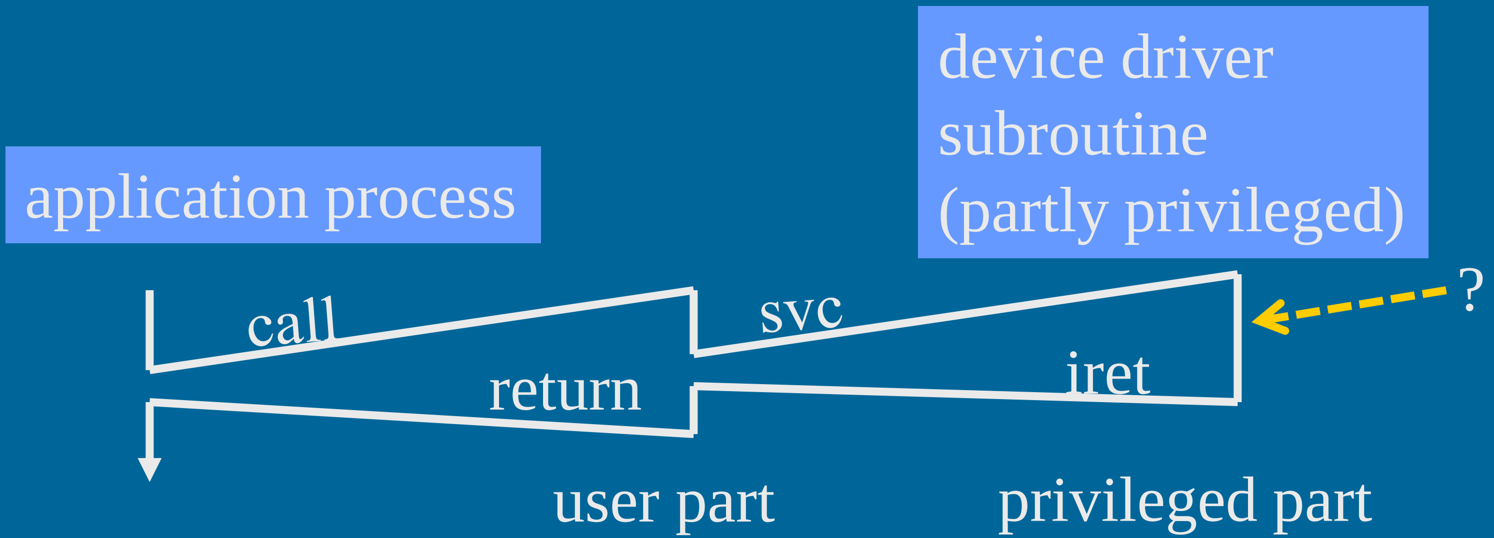
- Timers and other interrupts
  - interrupt → IRET (next to execute is the interrupted process, or the process selected by scheduler)

OS Example: Device Driver

- As a privileged subroutine (procedure)
  - Device driver is executed as an OS routine (called with SVC) in privileged state
    - Only one call in execution at a time? How can you supervise it?
      - What if process switch at this point?

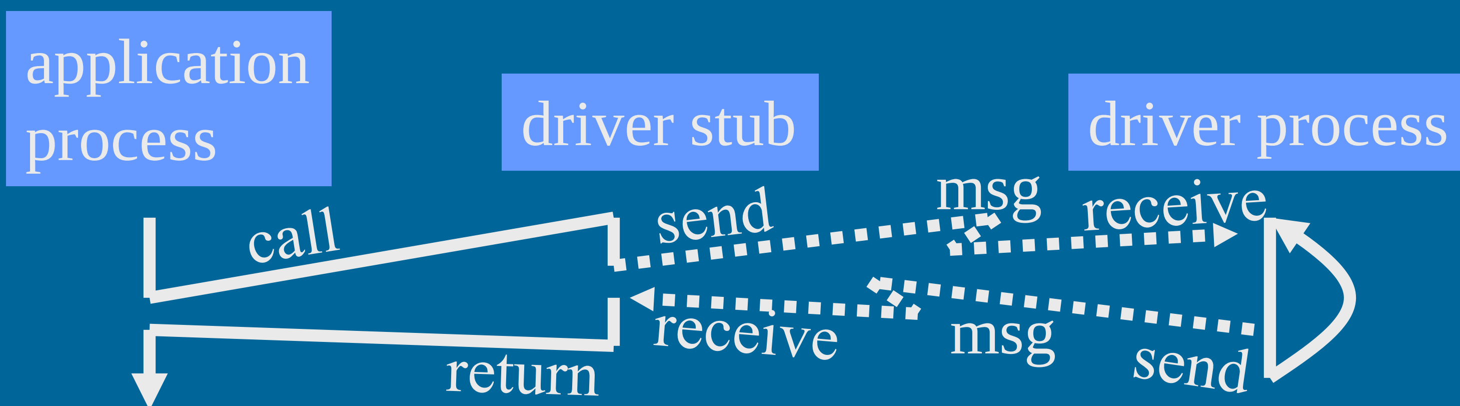
OS Example: Device Driver (contd)

- As a partly privileged subroutine
  - Device driver is executed as an OS routine (called with CALL) partly in privileged state
    - Part or all of the code may be privileged
    - Only one call in execution at a time?
      How can you supervise it?



OS Example: Device Driver (contd)

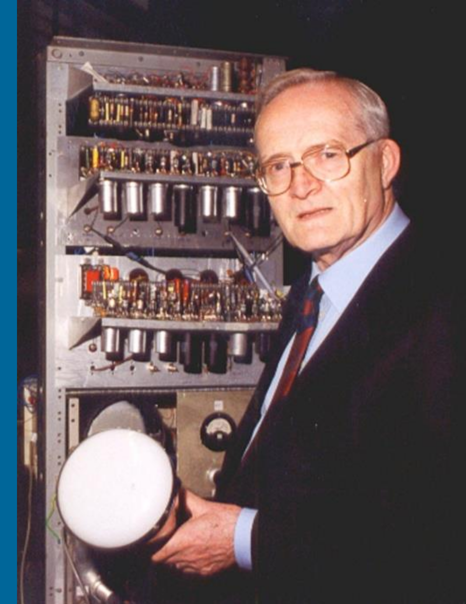
- As a process
  - Device driver stub call as a subroutine sends I/O request as a message to the device driver process and waits for reply
    - Stub may be is user state
    - Driver may be (partly) privileged
    - Based on IPC (Inter-Process Messaging)



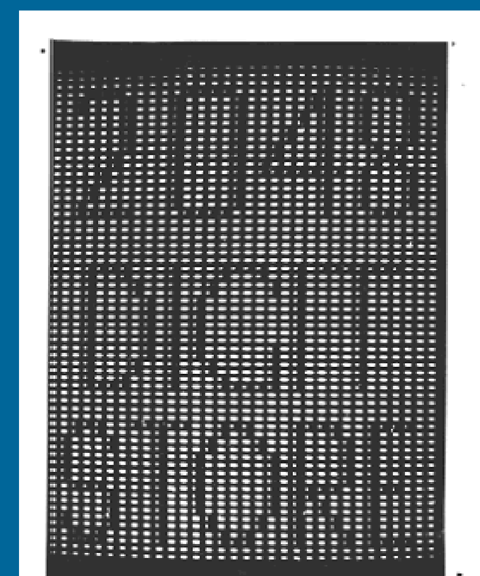
Why is it no problem, if many calls happen at the same time?

-- End --

- Williams Tube
  - 1946, Williams & Kilburn
  - Cathode Ray Tube (CRT)
  - First large ”RAM” memory
  - expensive: $1000 / tube / month
  - Small Scale Experimental Machine (”Baby”), 1947
  - Ferranti Mark I, 1st general use commercial computer system, 1951 (10000 bit memoryi)



Tom Kilburn holding a Cathode Ray Tube



Storing 2048 bits on a CRT in 1947