ELE6953E : Cyber-Physical Systems and the Internet of Things

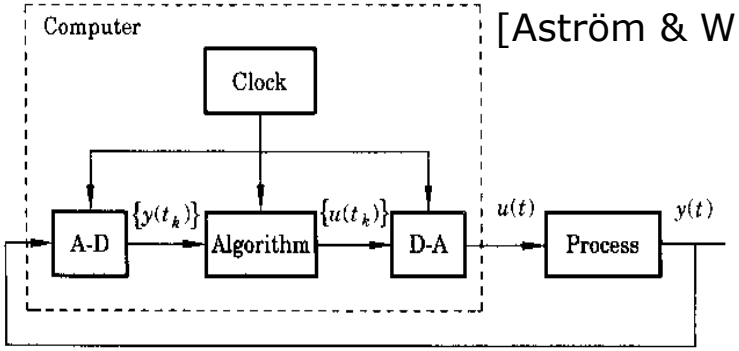
Lecture 3-1: NECS Implementation Examples & Simulation

Jérôme Le Ny
Department of Electrical Engineering, Polytechnique Montreal

GERAD
GROUP FOR RESEARCH IN DECISION ANALYSIS

POLYTECHNIQUE MONTRÉAL

CLASSICAL IMPLEMENTATION OF CONTROL TASKS



[Aström & Wittenmark, 1997]

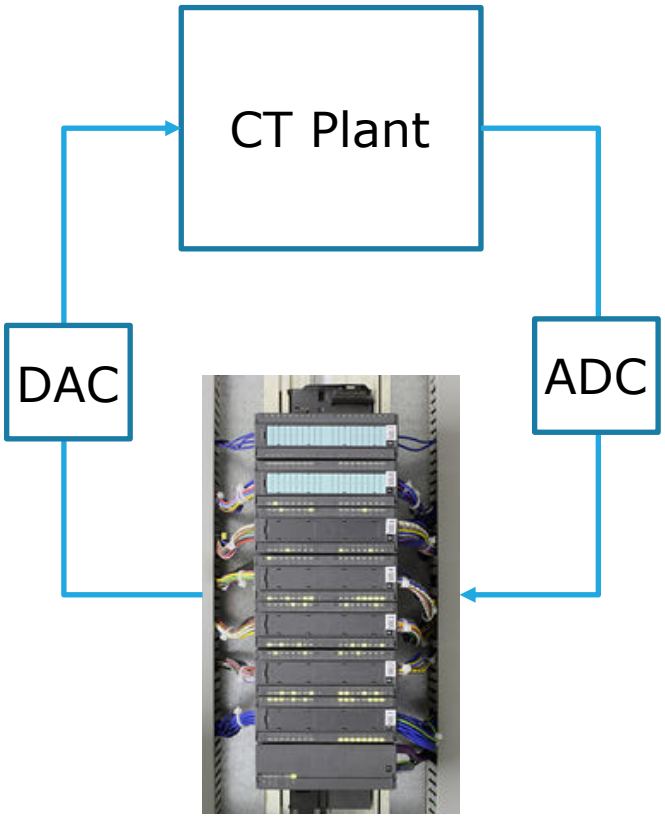
DT state-space controller example

Implement algorithm:

$$x_{k+1} = Fx_k + Gy_k + G_c u_{c,k}$$

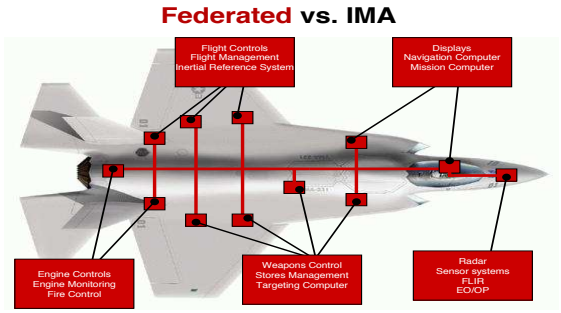
$$u_k = Cx_k + Dy_k + D_c u_{c,k}$$

```
Procedure Regulate    at clock interrupt
begin
1     Adin y uc
2     u:=u1+D*y+Dc*uc     output u asap
3     Daout u
4     x:=F*x+G*y+Gc*uc  | precompute for
5     u1:=C*x           | next period
end
```

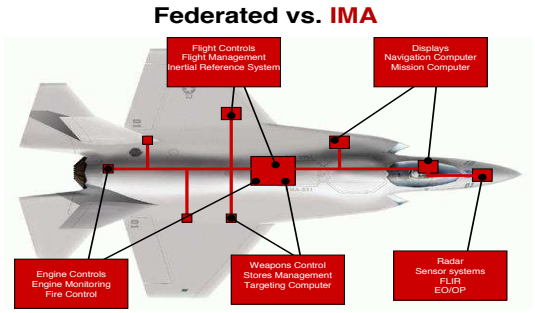


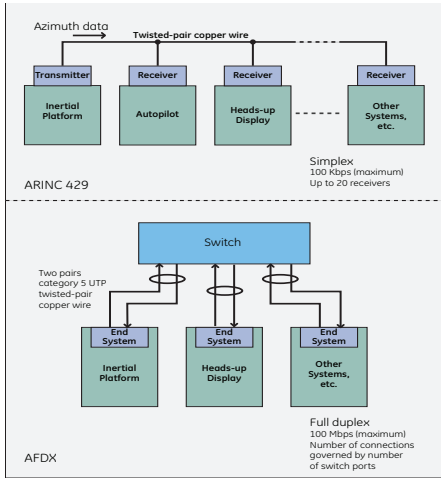
y: sensor outputs
u_c: ext. command sig.
u: controller output
x: controller state

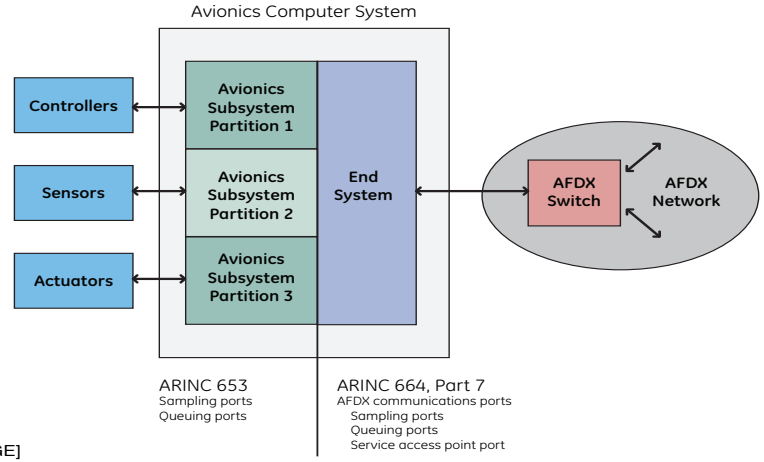
CURRENT REALITY: AVIONICS EXAMPLE



[© Wind River]



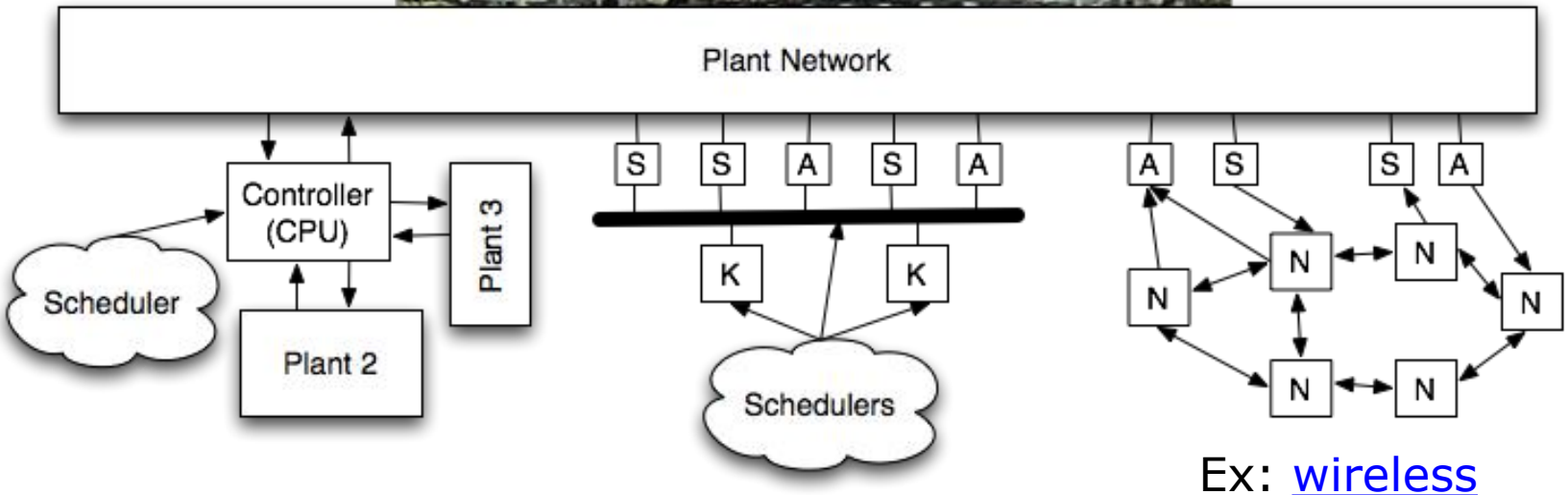




[© GE]

- AFDX/ARINC 664 Part 7: a type of switched Ethernet communication network
- ARINC 653: Partitioning for safety-critical avionics RTOS (real-time operating system)

VARIETY OF CONTROL SYSTEM CONFIGURATIONS TO CONSIDER



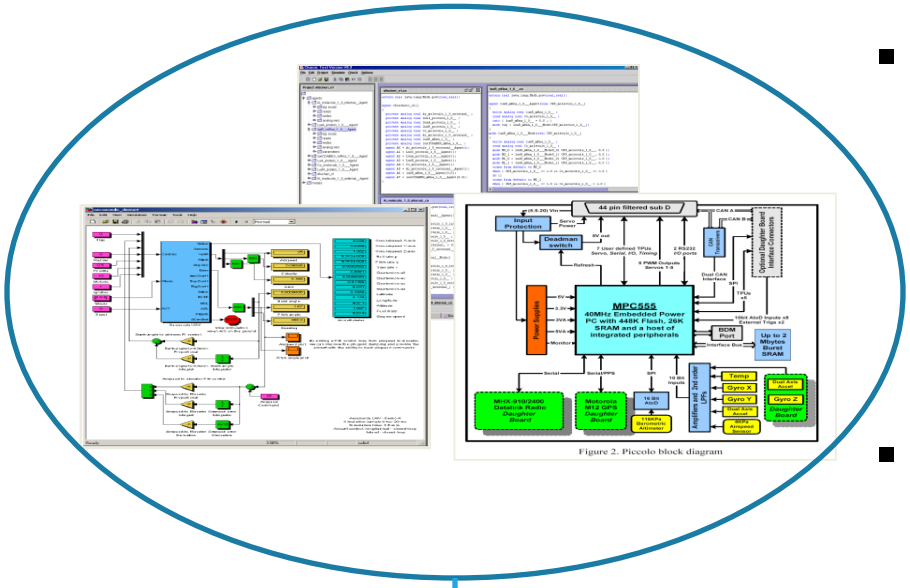
Ex: wireless control networks

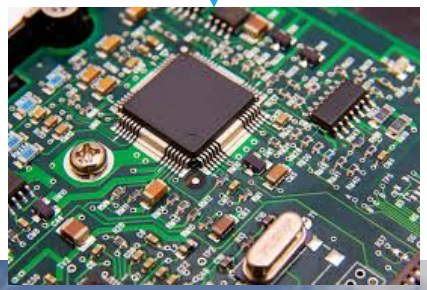
ISSUES

- Classical digital control is concerned with algorithms implementable on essentially dedicated resources: little need to worry about implementation at the algorithm design stage
- Modern implementation platforms are significantly more problematic for control algorithms, due to shared resources, unreliable communication, etc.: now need to model implementation artefacts already at algorithm design stage
- Examples:
  - Networked sensors with incorporated sampling circuitry: sampling is not synchronous among sensors, not decided by clock of controller
  - When data samples arrives at controller, might not be ready to execute the control task (multitasking with RTOS)
  - Output produced by controller needs to travel back on a network to reach actuators, DAC is performed there
  - How tightly are clocks on the network synchronized (if at all)?
  - What are protocols & resource scheduling policies used ? Etc.

HOLY GRAIL OF MODEL-BASED SYSTEMS ENGINEERING (MBSE)

POLYTECHNIQUE MONTRÉAL





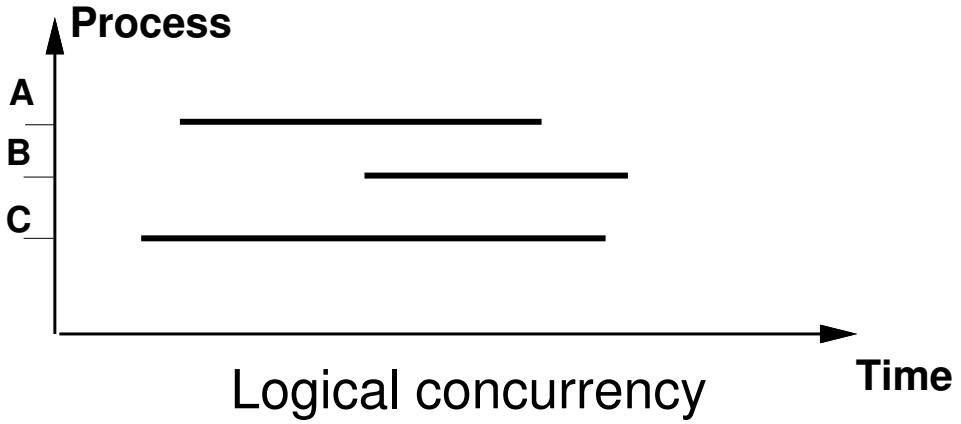
- Provide
  - high-level application specifications (stability, performance)
  - model of the physical plant and uncertainty
  - description of available implementation platform (network, computers, RTOS, etc.)
- Output:
  - Generated code for target platform, scheduling policies, etc.
  - Proof / certificate that algo + platform + plant model satisfy specs or that it is impossible to satisfy
- More realistically?
  - Generate code implementing a given control law on a specific platform that preserves the certification/proofs provided at the control design stage
- Current efforts in MBSE: AADL+Ocarina, Papyrus(-RT) & SysML, …

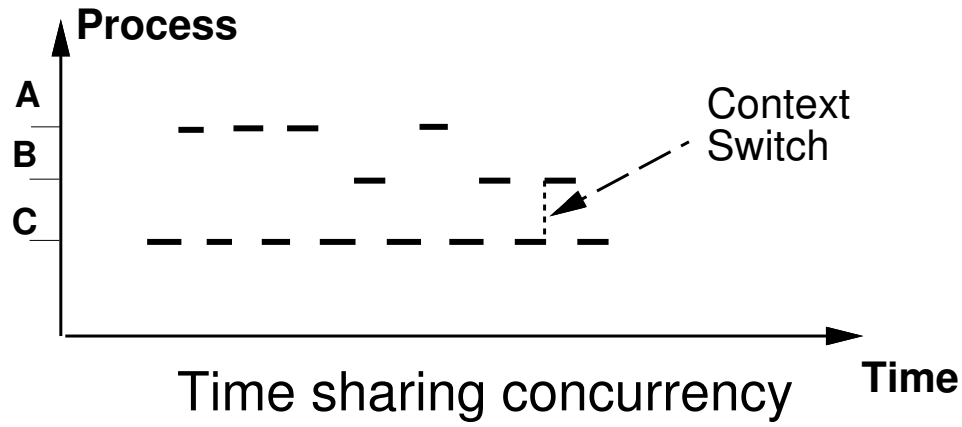
AGENDA

1. Sharing of computational resources: RTOS task scheduling
2. Networking
3. Simulation with TrueTime

SOME TERMINOLOGY

- In computer science (CS), reactive systems are computer systems that interact with the environment via inputs and outputs
  - Focus different from traditional computing (logic+speed, ex: sorting)
- *Real-time systems* are reactive systems that must provide timing guarantees when reacting to external events
  - Execution timing must be **predictable**, not necessarily fast
  - Ex: control systems. Execute a control task every 10 ms
  - Hard/firm/soft real-time constraints. Ex: Patriot missile failure
- Often (but not always), RTS are *embedded systems*: HW/SW integrated with machines (mechanical, electrical, etc.) for a *specific purpose* (vs. general-purpose computing)
  - Ex. of embedded systems: cell phone, smart toy, cruise controller, flight control system, robot controller, etc.
- Many (but not all) embedded systems are real-time systems
- How can we rigorously design and program real-time systems to satisfy these timing (and performance) constraints?





[K-E Arzén, LTH]

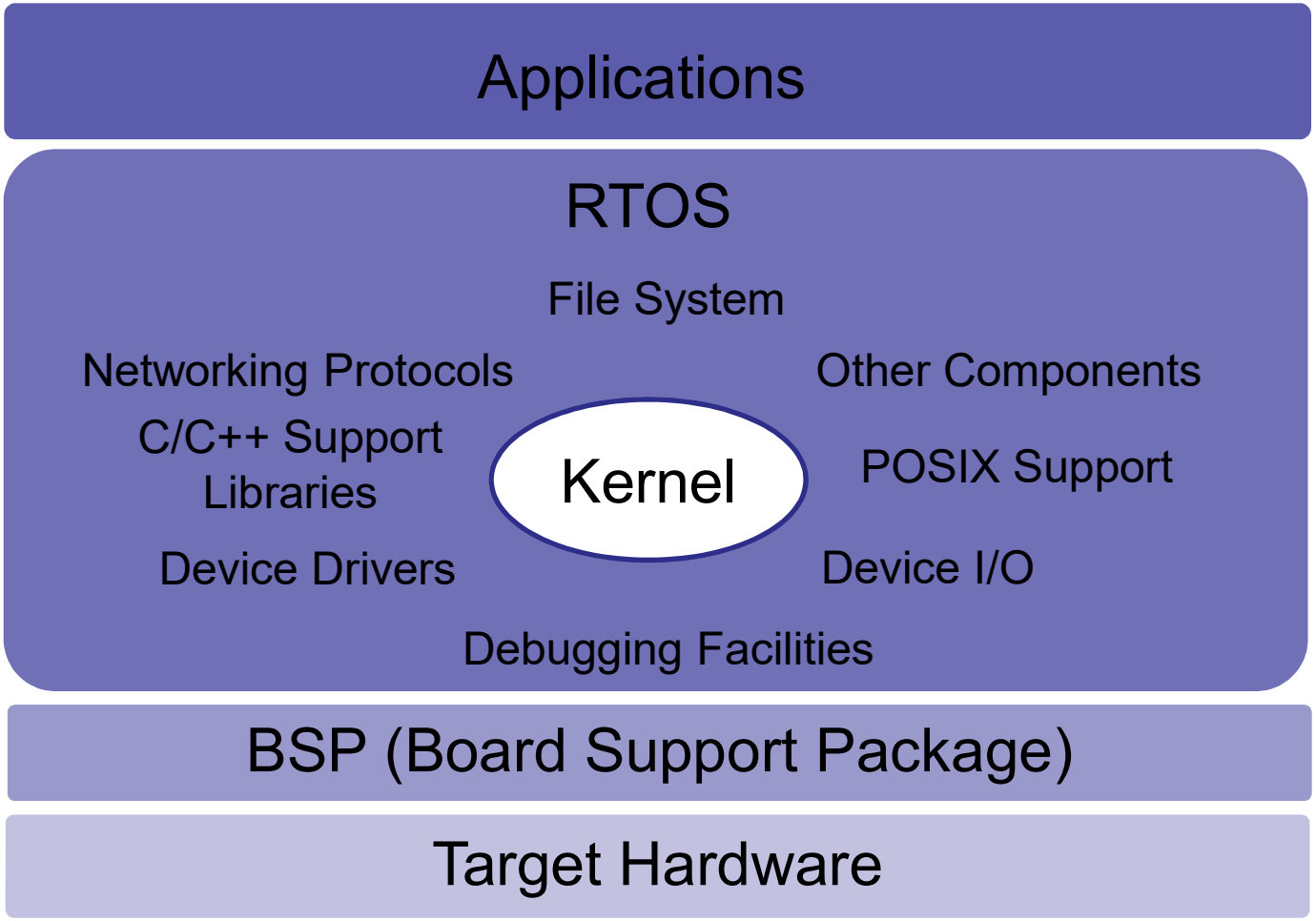
- Ex: use single computer to close multiple control loops
- Sharing computing time on single CPU to achieve pseudo-parallelism
- ≠ true parallelism with multi-core or networked computers, etc. See later in the course for distributed computations.

REAL-TIME OPERATING SYSTEMS

POLYTECHNIQUE MONTRÉAL

- An RTOS provides abstraction/interface to facilitate programming / understanding / maintaining / certifying RT software
  - Multiple functions/applications sharing same processor
  - Alternative to "bare-metal programming" using assembly, timers, interrupts, low-level drivers, etc. directly
- Want formal guarantees to avoid in particular occurrence of rare but possible timing bugs, very hard to find otherwise via testing
  - Ex: Patriot missile accumulated 57 µs per minute, 343 ms after 100 hr
- RTOS provides some or all of the following:
  - Real-time kernel, allowing switches between processes, preemption with timing guarantees (vs. kernel of standard OS)
  - Scheduler (periodic and non-periodic tasks) and (possibly) API to specify timing constraints on tasks
  - Handling interrupts, I/O
  - Context switching for persistent state between task activations
  - Communication between processes, memory partitioning / management
  - Possibly support for time triggered architecture, etc.
- OS standards to facilitate programming portable applications
  - (RT-)POSIX (gen. purpose), OSEK/VDX/AUTOSAR OS (automotive), ARINC 653/APEX (avionics), µITRON (small embedded systems)

SYSTEM ARCHITECTURE



[ELE8200, G. Zhu]

PROGRAMMING WITH AN RTOS

POLYTECHNIQUE MONTRÉAL

```
Task(x) {
    int local;
    initialization();

    for (;;) {
        do_instance();
        end_instance();
    }
}
```

suspend task until next activation

Ex: Static thread in ChibiOS

```
#include <ch.h>

/*
 * Working area for the LED flashing thread.
 */
static THD_WORKING_AREA(myThreadWorkingArea, 128);

/*
 * LED flashing thread.
 */
static THD_FUNCTION(myThread, arg) {

  while (true) {
    LED_ON();
    chThdSleepMilliseconds(500);
    LED_OFF();
    chThdSleepMilliseconds(500);
  }
}

int main(int argc, char *argv[]) {

  /* Starting the flashing LEDs thread.*/
  (void)chThdCreateStatic(myThreadWorkingArea, sizeof(myThreadWorkingArea),
                          NORMALPRIO, myThread, NULL);
  .
  .
  .
}
```

- > 100 commercial RTOS
  - Ex: VxWorks, OSE, Windows CE, QNX, INTEGRITY, PikeOS, etc.
- Some open-source options and/or free for certain applications
  - Ex: FreeRTOS, RT-Linux, eCos, ChibiOS
- Typically restricted set of scheduling policies (preemptive, often static, etc.)
- Real-time research kernels
  - Ex: Erika, Shark, Marte OS
  - Support for more advanced features studied in RT scheduling research

CHOICE OF TASK ORGANIZATION

POLYTECHNIQUE MONTRÉAL

```
void task(void * pdata)
{
   servo_setup(pdata);
   sensor_setup(pdata);
   for ( ; ; ) {
      y = read_measurement();
      r = read_reference();
      x = Ac*x + Bc*(r-y);
      u = Cc*x + Dc*(r-y);
      write_control_signal(u);
   }
}
```

- Shorter tasks can improve schedulability (better for RT design, cost, etc.)
- But can reduce determinisn, ex: ordering of operations (more problems for control and algorithm design)

[ELE8200, G. Zhu]

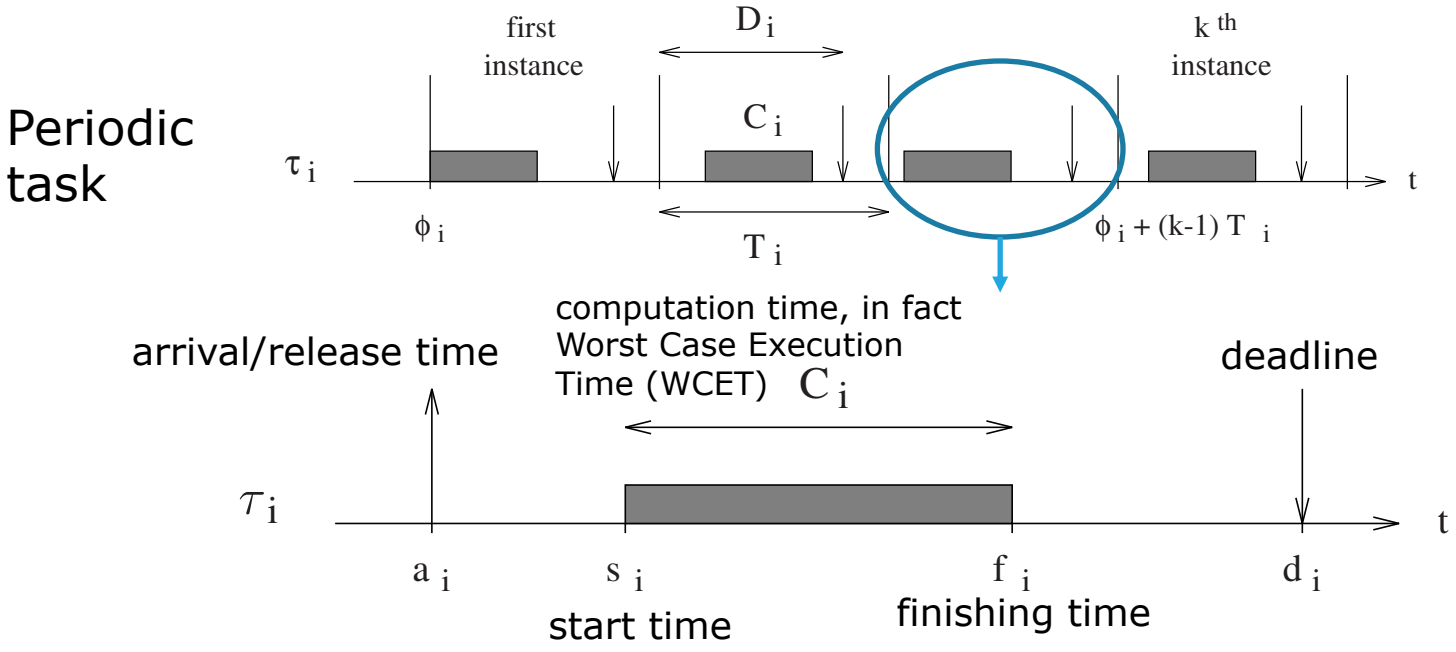
```
void task1(void * pdata)
{
   sensor_setup(pdata);
   for ( ; ; ) {
      time_out();
      y = read_measurement();
   }
}
```

```
void task2(void * pdata)
{
   servo_setup(pdata);
   for ( ; ; ) {
      time_out();
      r = read_reference();
      x = Ac*x + Bc*(r-y);
      u = Cc*x + Dc*(r-y);
      write_control_signal(u);
   }
}
```

BASICS OF REAL-TIME SCHEDULING: TERMINOLOGY

POLYTECHNIQUE MONTRÉAL

- Task: code that can execute repetitively on the CPU (seq. of *jobs*)
  - Periodic or aperiodic (sporadic if minimum inter-arrival time)



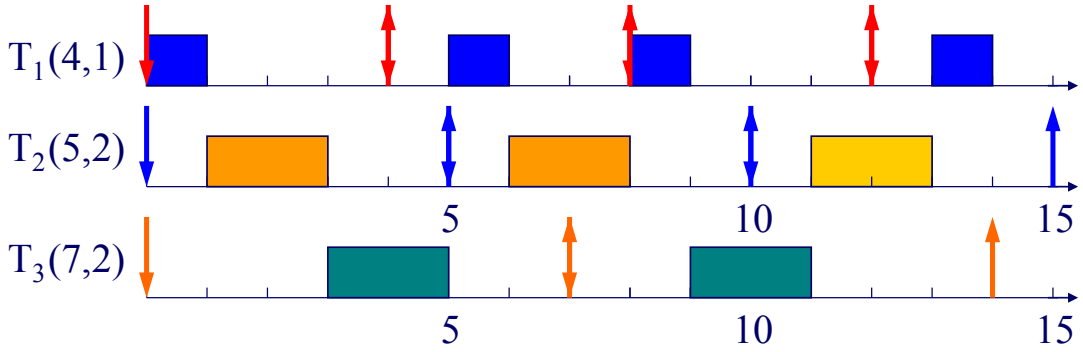
[Buttazzo, 2011]

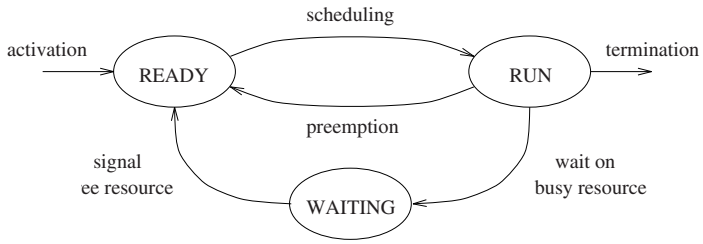
- Goal of scheduling: guarantee that a set of tasks execute on CPU and satisfy their (timing, precedence, mutual exclusion) constraints
- Scheduling: Preemptive, nonpreemptive or using preemption points
  - Most RT systems allow preemption (execute critical tasks asap, higher efficiency)

SCHEDULING WITH PREEMPTION

POLYTECHNIQUE MONTRÉAL

[Insup Lee, Upenn]





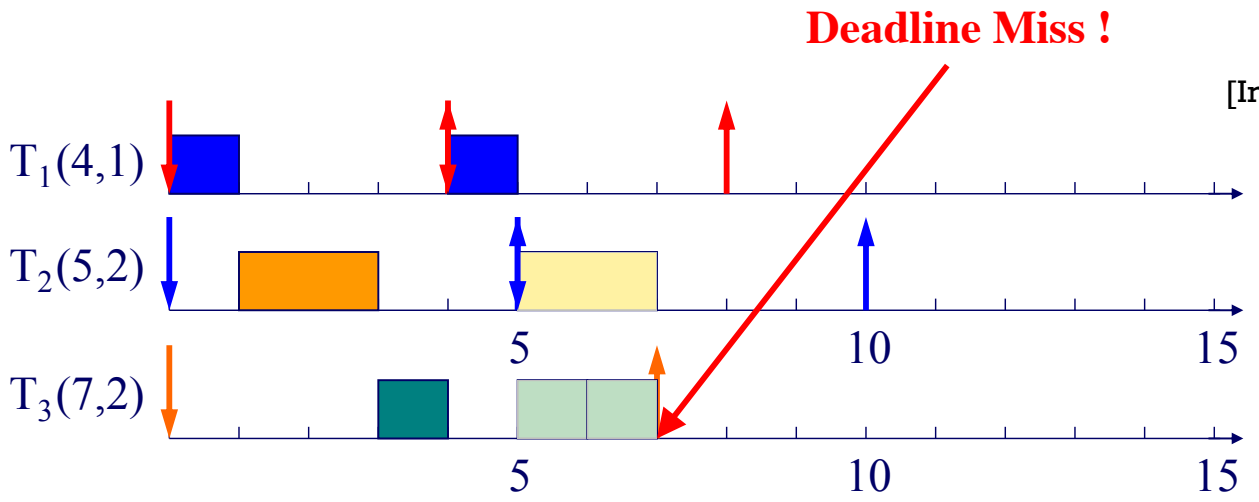
3 periodic tasks scheduled with EDF

- Tasks with higher priority can preempt those with lower priority
- Priorities can be set **statically** (at design time, ex: rate monotonic - RM) or **dynamically** (based on their *current state*, ex: Earliest Deadline First – EDF: schedule task with current earliest deadline)
  - EDF is more efficient but not typically available commercially. RM is more predictable for high priority tasks, can be implemented even if no explicit support for timing constraints in RTOS
  - Given a set of tasks if the EDF algorithm fails to find a feasible schedule, then there is no feasible schedule (meeting releases/deadlines)

RATE MONOTONIC SCHEDULING (RM)

POLYTECHNIQUE MONTRÉAL

- Jobs with smaller period have higher priority

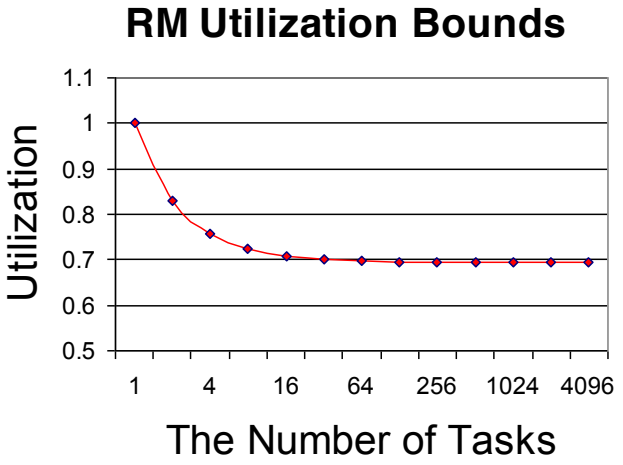


- A set of n tasks is schedulable if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

[Liu and Layland, 1973]

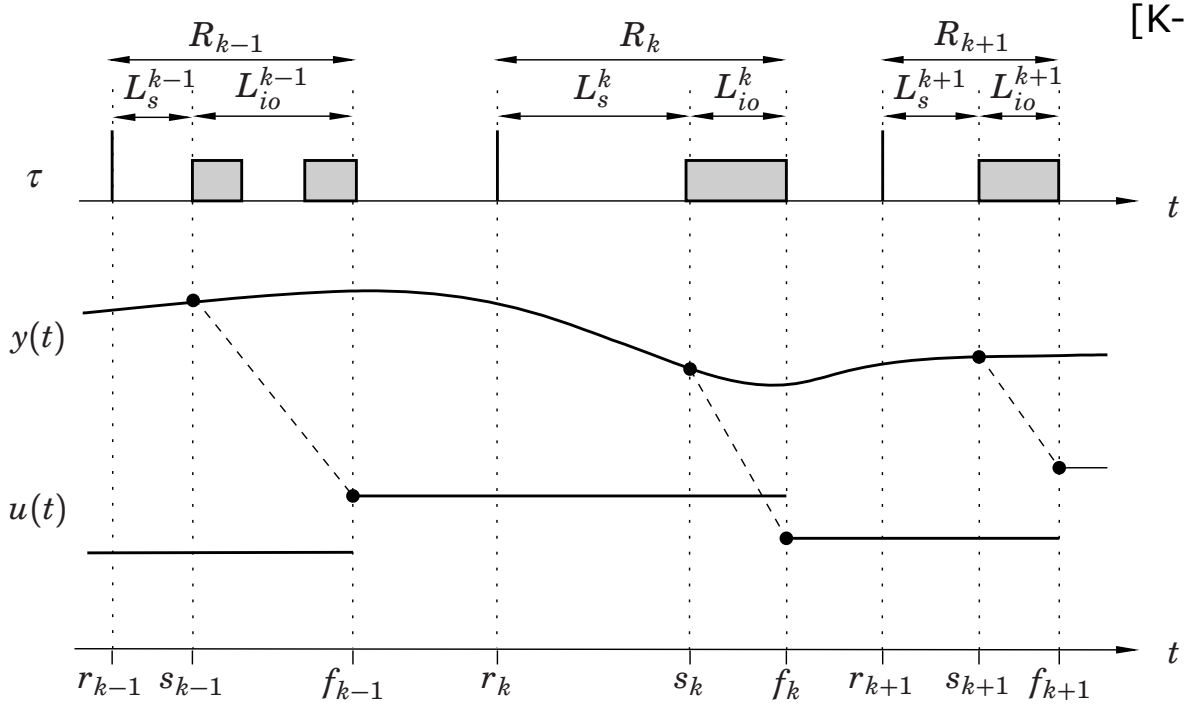
- Sufficient condition (bound = 1 for EDF)



CONTROLLER TIMING EXAMPLE WITH AN RTOS

POLYTECHNIQUE MONTRÉAL

[K-E Arzén, LTH]



- Control task τ released periodically at time instances $r_k = kh$
- Output $y(t)$ sampled after time-varying **sampling latency** L_s
- Control $u(t)$ generated after time-varying **input-output latency** L_{io}

AGENDA

1. Sharing of computational resources: RTOS task scheduling
2. Networking
3. Simulation with TrueTime

NETWORKING ISSUES

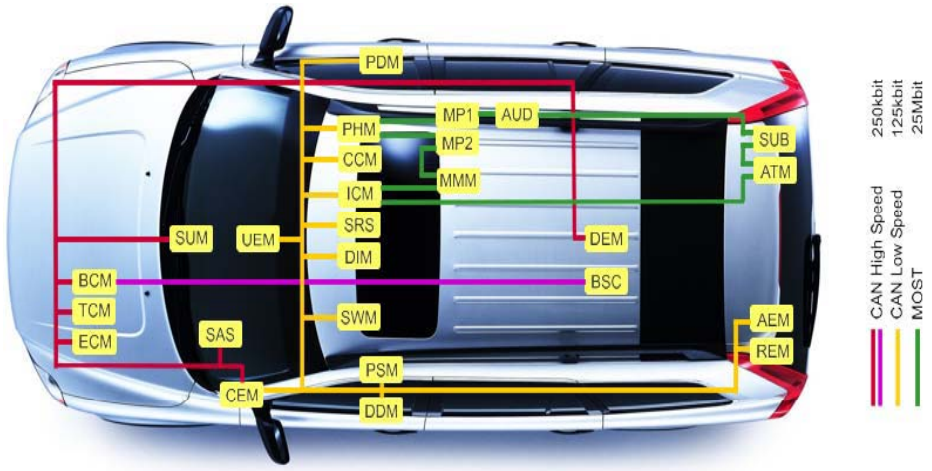
- Various components of control loops now often connected via digital communication network
  - Communication delays (generally due to congestion, not transmission time), lost packets
- Communication medium tends to be shared by several applications (as for computing resources)
  - Need management of communication resources as well
- The system design choices for communication and computation (hardware, protocols, etc.) not typically dictated by the control engineers (cost, in many industries, is a big factor)
  - E.g. CAN networks in automotive application introduce time-varying delays, bad from control perspective
  - But control and other timing concerns tends to slowly influence choices (TT-CAN, FlexRay, AFDX…)
  - Push to use Commercial Off-the-shelf (COTS) components, reuse components and software from other/previous systems

COMMUNICATION NETWORKS

POLYTECHNIQUE MONTRÉAL

- Wired point-to-point (no real network)
- Wired data bus (with or without bus controller), e.g., CAN
- Switched network (ex: AFDX)
- Wireless networks (e.g., for industrial applications, process control. With added reliability mechanisms (ex: WirelessHART)
- Delays, reliability (packet losses), jitter, etc. are network and protocol dependent. Impacts performance of control systems.

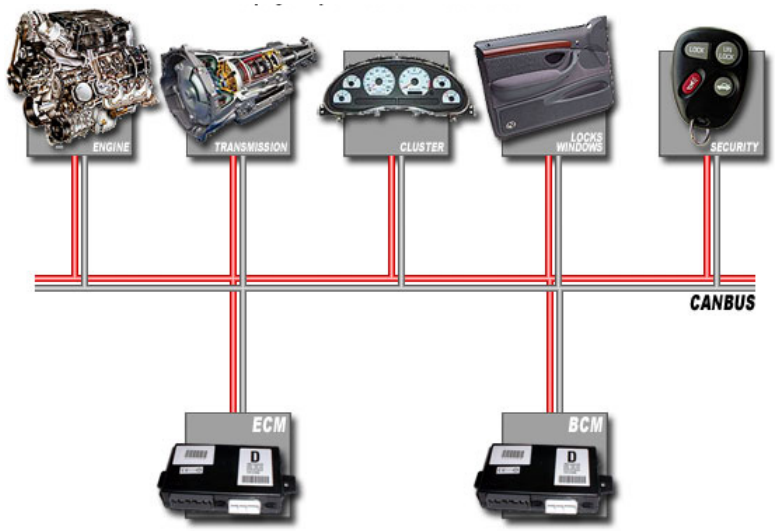
Volvo XC 90 network topology

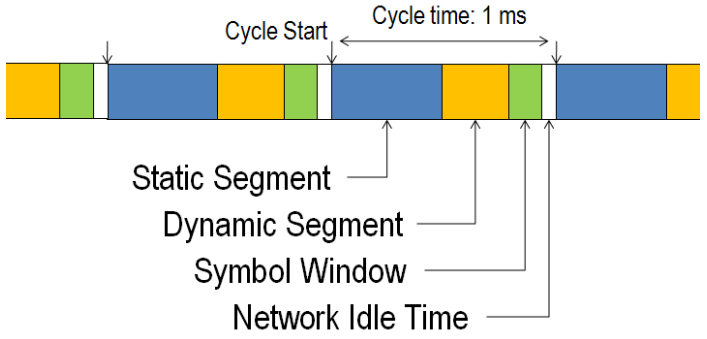


AUTOMOTIVE INDUSTRY EXAMPLE: CAN AND FLEXRAY (BUS TECHNOLOGY)

POLYTECHNIQUE MONTRÉAL

- De facto Standard in Automotive Industry: CAN (Controller Area Network)
  - Messages have an identifier used to set priorities in case of collision
  - Unsuccessful node retries later
- Issues with non-determinism
  - Move toward time-triggered architectures (TDMA: Time-division multiple access, e.g. TT-CAN), for timing predictability
  - But wastes bandwidth
- FlexRay: part TT, part not to accommodate different types of traffic (video feed vs. engine rpm…)





AVIONICS EXAMPLE: AFDX (SWITCHED NETWORK TECHNOLOGY)

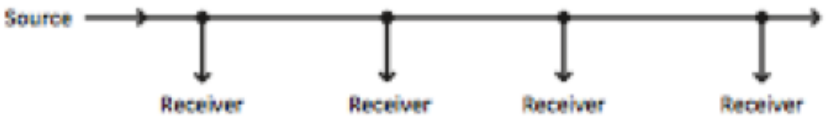
POLYTECHNIQUE MONTRÉAL

- All electronic fly-by-wire now only type of control systems used on new airliners
- Other on-board safety-critical system systems also rely on timely delivery of data: communications, inertial platforms, etc.
- AFDX: "Avionics Full-DupleX, switched Ethernet". Original concept by Airbus. Evolved into a standard: IEEE 802.3 (Ethernet frame format) and ARINC 664, Part 7.
- Used by Airbus for A380, by Boeing for 787 Dreamliner
- Replaces ARINC 429 (point-to-point technology), and also MIL-STD 1553 (bus technology). Higher data rate: 100 Mbps, 1000 times faster than ARINC 429.
- Reduces wiring, hence weight

AVIONICS NETWORKS

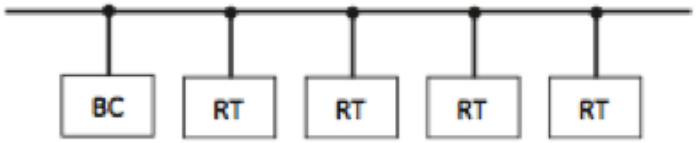
POLYTECHNIQUE MONTRÉAL

ARINC 429



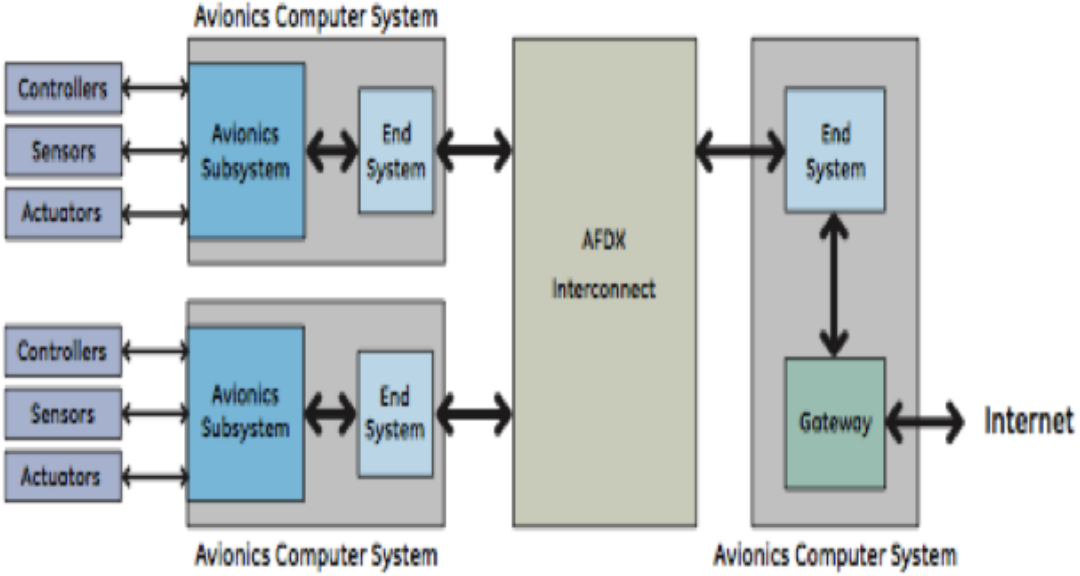
Bit rates are either 100 Kbps or 12.5 Kbps
32-bit messages

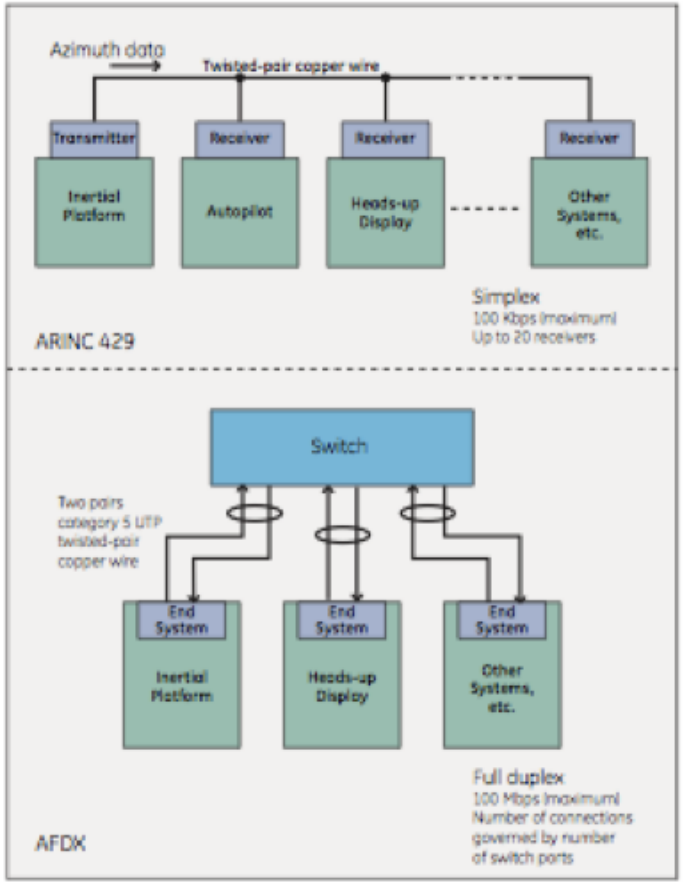
MIL-STD 1553 DATA BUS



Bit-rate 1 Mbps
20-bit data word

AFDX





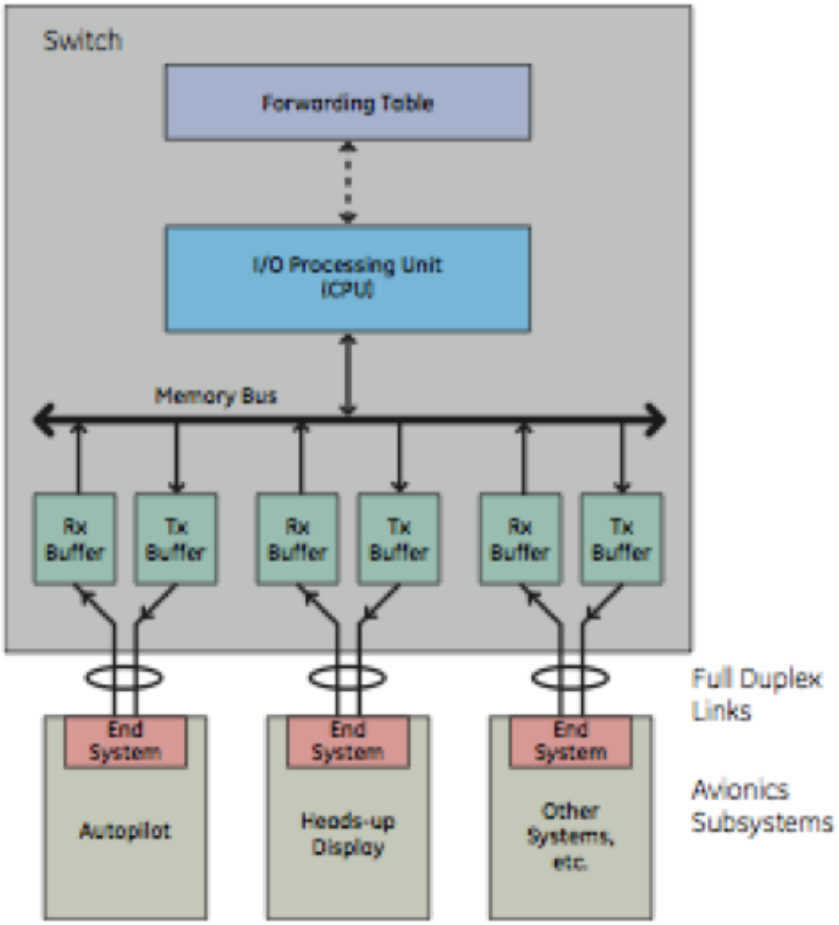
AFDX IS BASED ON ETHERNET

- Benefits from many investments and advancements since 1972
- Ethernet has no centralized bus control: transmissions can collide. "CSMA/CD" protocol (Carrier Sense, Multiple Access, and Collision Detection)
  - If you have a message to send and the medium is idle, send the message.
  - If the message collides with another transmission, try sending the message later using a suitable back-off strategy → non-deterministic behavior, possibility of repeated collisions
- Ethernet frame between 64 and 1518 bytes
- Ethernet comm. is connectionless. ACK must be handled at higher levels in the protocol stack

AFDX: FULL-DUPLEX, SWITCHED ETHERNET

- With Ethernet, very large transmission delays are theoretically possible
- AFDX bounds the max. transmission time between a Tx and a Rx
- Moves from Half-duplex Ethernet to Full-duplex Switched Ethernet to eliminate possibility of collisions
- Now switch needs to move packets from Rx to Tx buffers through memory bus (store and forward architecture). Delays possible due to congestion at the switch
- Each buffer can store multiple packets in FIFO order. Requirements on avionics subsystems to avoid overflow.
- Jitter introduced in packet transmission times when waiting for other packets to be transmitted
- Multiple switches can be connected
- Redundancy: an AFDX system consists in fact of two independent networks sending the same packets between end systems

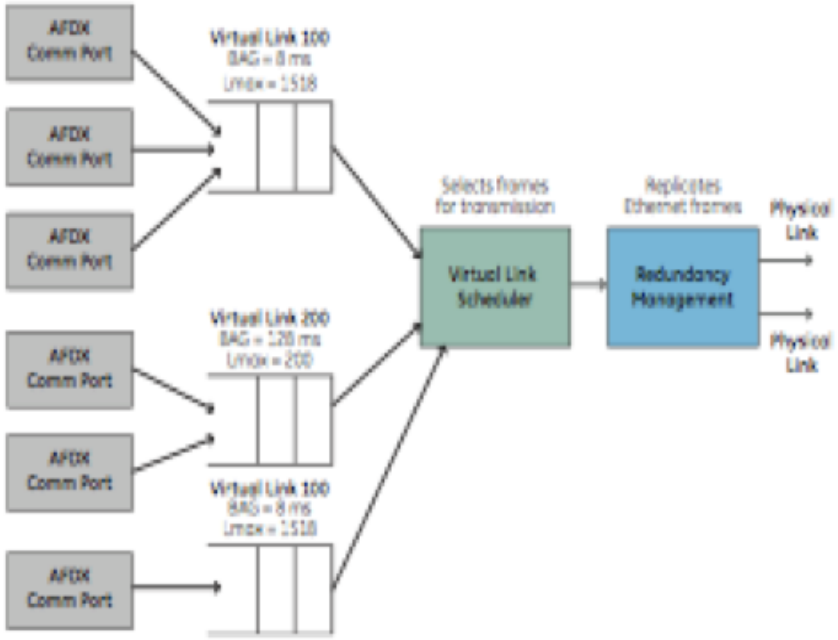
AFDX Switch



MESSAGE FLOWS

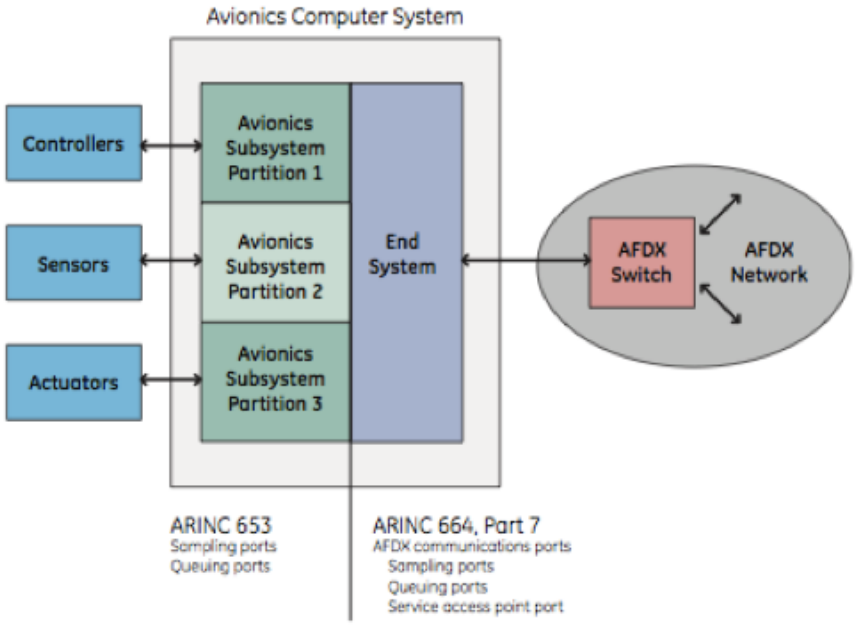
POLYTECHNIQUE MONTRÉAL

- Packets (AFDX frames, almost identical to Ethernet frames) are sent between End Systems using "Virtual Links" (VLs)
- Total 100 Mbps bandwidth at each end system is shared between VLs
- The bandwidth of each VL is limited: mandatory gaps between messages, max. size of frames
  - bandwidth choice depends on applications connecting to end system via comm. ports
  - bandwidth restrictions enforced by source End Systems, using VL scheduling algorithms.
  - VL scheduler also multiplexes the VL transmission to minimize jitter



INTERFACE RTOS / NETWORK
ARINC 653 / AFDX

- ARINC 653 RTOS: One computer system partitioned in multiple subsystems
  - restrict address space of each partition
  - limits on amount of CPU time for each partition
- Avionics applications send messages using communication ports
  - communication ports are part of OS API for portable avionics applications described in ARINC 653
  - Sampling ports and Queuing ports
  - AFDX end system has corresponding sampling and queuing ports, + Service Access Point (SAP) ports for comm. with non-AFDX systems
  - Each sending AFDX comm. ports is associated with a single VL transporting its messages



ARINC 653/AFDX COMMUNICATION PORTS

- Sampling Ports
  - buffer storage for a single message.
  - a message stays in the buffer until overwritten by new message. Can be read multiple times
  - must provide time-stamped messages. Indication of the freshness of the message in the buffer
  - Adequate for data sampled for control applications
- Queuing Ports
  - buffer can store a fixed number of message (config. param.)
  - reading a message removes it from the queue (FIFO)

AGENDA

1. Sharing of computational resources: RTOS task scheduling
2. Networking
3. Simulation with TrueTime

USING SIMULATION TOOLS

POLYTECHNIQUE MONTRÉAL

- TrueTime [Henriksson, Cervin, Ohlin, Eker 1999-2008]
  - Matlab/Simulink toolbox, can be interfaced with other Simulink blocks. Allows co-simulation of global system (implementation platform (RTOS+network) + model of physical plant)
  - Focus on control systems
  - Used in HW1
- Other simulators, with focus RT computer systems only: RTSim, ChronSIM, …

FEATURE

How Does Control Timing Affect Performance?

Analysis and Simulation of Timing Using Jitterbug and TrueTime

Control systems are becoming increasingly complex from both the control and computer science perspectives. Today, even seemingly simple embedded control systems often contain a multitasking real-time kernel and support networking. At the same time, the market demands that the cost of the system be kept at a minimum. For optimal use of computing resources, the control algorithm and the control software designs need to be considered at the same time. For this reason, new computer-based tools for real-time and control codesign are needed.

Many computer-controlled systems are distributed systems consisting of computer nodes and a communication network connecting the various systems. It is not uncommon for the sensor, actuator, and control calculations to reside on different nodes, as in vehicle systems, for example. This gives rise to networked control loops (see [1]). Within the individual nodes, the controllers are often implemented as one or several tasks on a microprocessor with a real-time operating system. Often the microprocessor also contains tasks for other functions (e.g., communication and user interfaces). The operating system typically uses multiprogramming to multiplex the execution of the various tasks. The CPU time and the communication bandwidth can hence be viewed as shared resources for which the tasks compete.

Digital control theory normally assumes equidistant sampling intervals and a negligible or constant control delay from sampling to actuation. However,

©MASTERFILE/RES

By Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén

Cervin (anton@control.lth.se), Henriksson, Lincoln, Eker, and Årzén are with the Department of Automatic Control, Lund Institute of Technology, Box 118, SE-221 00 Lund, Sweden.

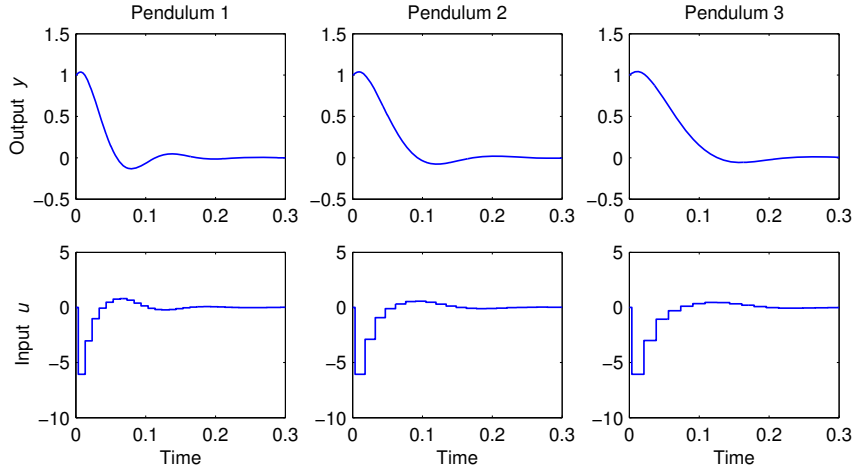
0272-1708/03/$17.00©2003IEEE

[Cervin et al. IEEE CSM, 2003]

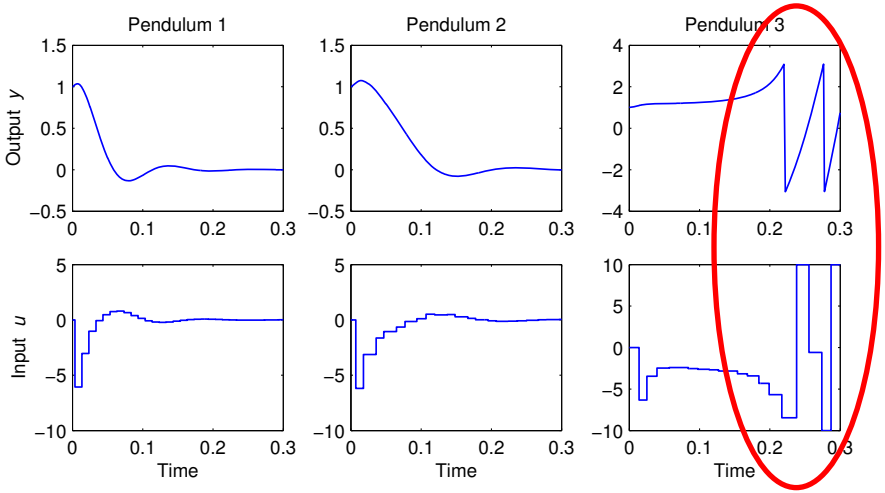
THREE PENDULUM EXAMPLE

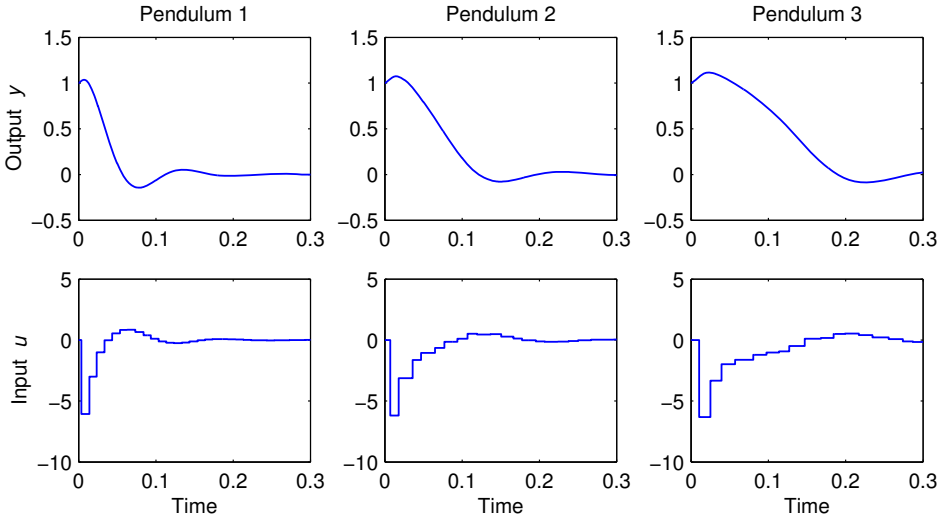
POLYTECHNIQUE MONTRÉAL

3 CPUs



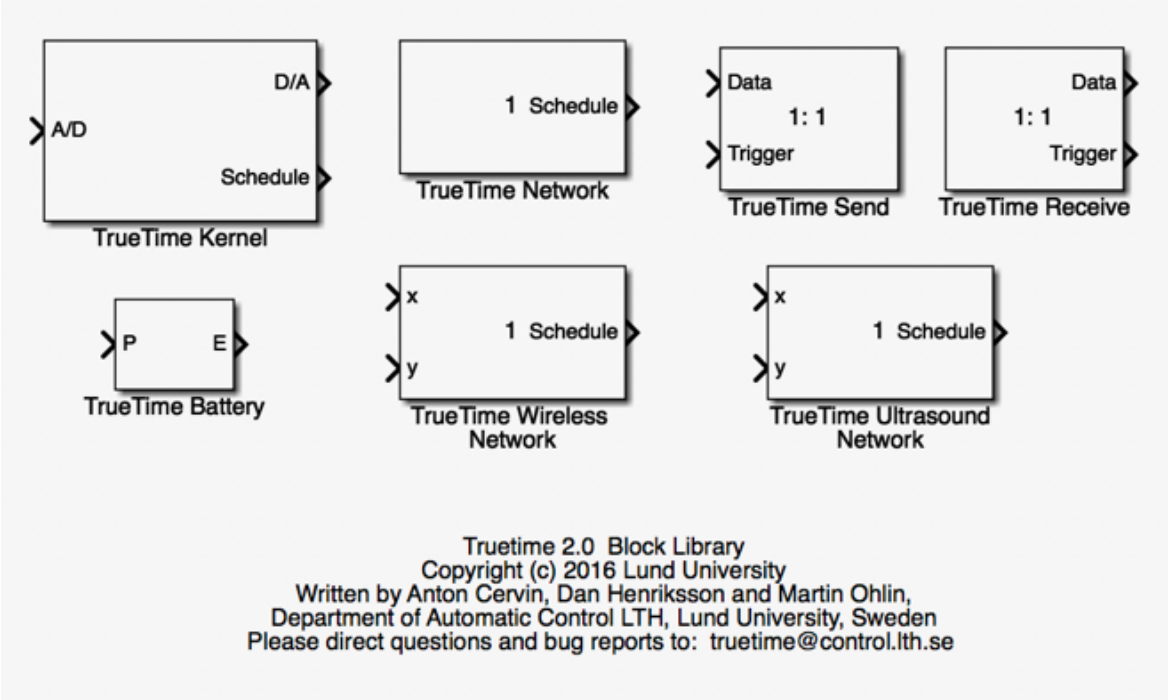
RM scheduling (schedulable)





EDF scheduling (schedulable)

TRUETIME LIBRARY



- Kernel block (RTOS) to simulate task scheduling policies
- Networking blocks to simulate communication protocols
- Battery block (and dynamic voltage scaling)

QUESTIONS RAISED BY THESE EXAMPLES

Possible NECS design approaches?

- Design control loops as usual, ignoring implementation issues
  - Research question: analyze the impact of implementation choices on stability and performance
- Control and embedded system co-design: design control loops together with the scheduling policies, communication protocols, etc.
  - Popular research topic, perhaps not a viable due to other influences than control system on implementation choices
- Design implementation aware control loops
  - Add compensating mechanisms within the control laws, for delays, jitter, packets losses.
  - Control laws more complex to implement, but we are (as much as possible) only modifying the control systems, not the global system design.
  - Expect potential loss in performance with respect to co-design, but should be better than ignoring issues. Also, in this approach, we would like to certify against sets of possible choices of implementations, to leave more design freedom, possible subsequent changes of design without recertification, increase possibilities of software component reuse, etc.