CS684 – Embedded Systems (Software)

Introduction to Realtime Systems – I (Characteristics)

Kavi Arya

IIT Bombay

Embedded Systems?

Plan

- **Realtime Embedded Systems**
  - Introduction
  - Application Examples
- **Real-time support for Embedded Software**

Anatomy of Embedded Applications

- CC : continuous control, signal processing
  - Differential equation solving, filters
  - Specs and simulation in Matlab / Scilab, manual or automatic code
- FSM : finite state machines, state transition systems
  - Discrete control, protocols, networking, drivers, security, etc.
  - Flat or hierarchical state machines, manual or automatic code
- Calc : calculation intensive
  - Navigation, security, etc.
  - C, manual + libraries
- Web : web-like navigation, audio / video streaming
  - Consumer electronics, infotainment systems
  - Data-flow networks, embedded Java

1. BMW 745i : Prelude To Complexity

A Life Cycle Example :

The Software Error

External view

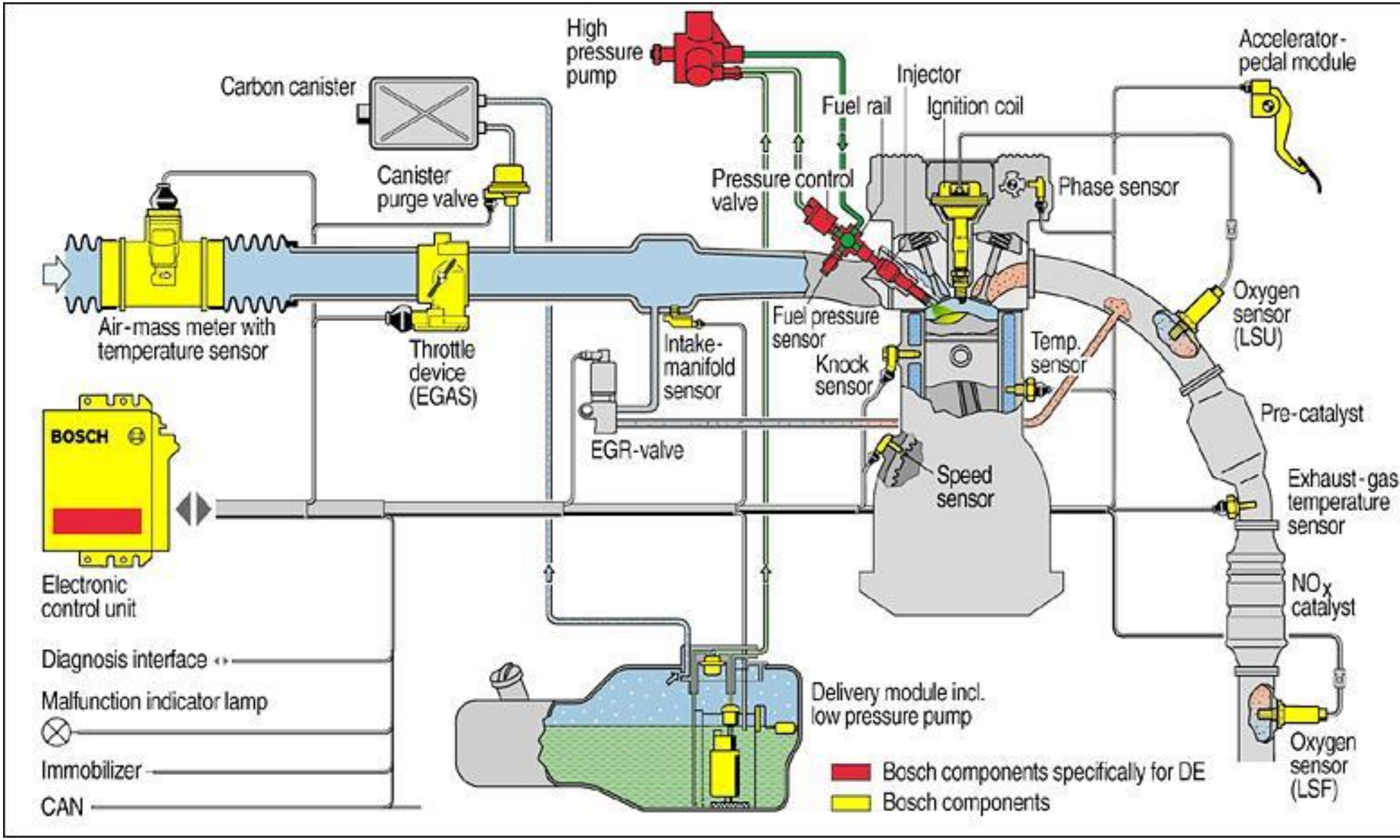
The problem: software error, a desynchronization of the valvetronic motors

- **Rough running engine, possibly stall**
- **Severity: 6 incidents in 5,470 cars with 2 rear endings**
  - "alleged injury" of BMW passengers
  - Fault of drunk or inattentive following drivers

BMW Cost

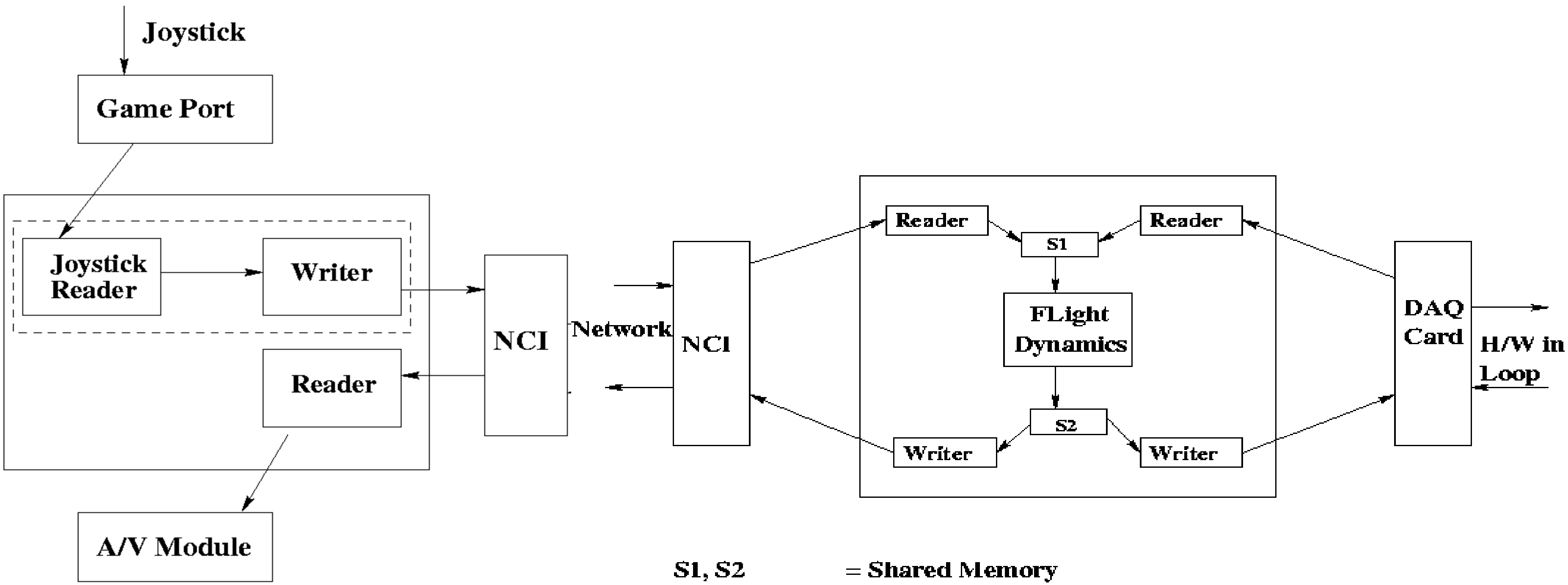
- **To repair: Reprogram ECU**
- **Recalls not uncommon in industry**
  - BMW 5,470 cars @ $68,500 = Rev $372 mil
- **Compare Cost: Recall BMW X5**
  - 164,000 units @ $66,800 = Rev $10 bil.
  - ~$5 Million
  - ~$30 per SUV

Bosch EMU For Four Wheeler (Multi Cylinder)



Source: Bosch Brochure : Ref 6

2. Flight Simulator

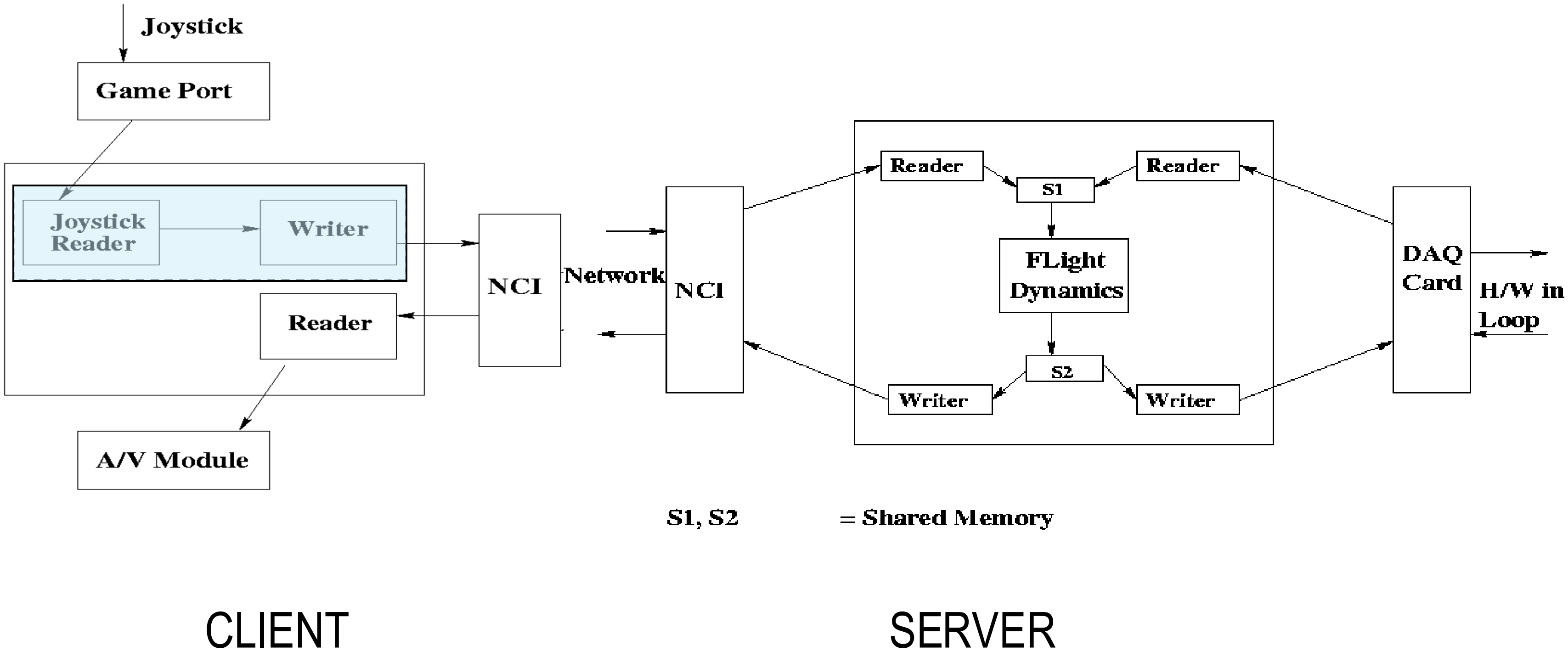


CLIENT - pilot

SERVER - simulator

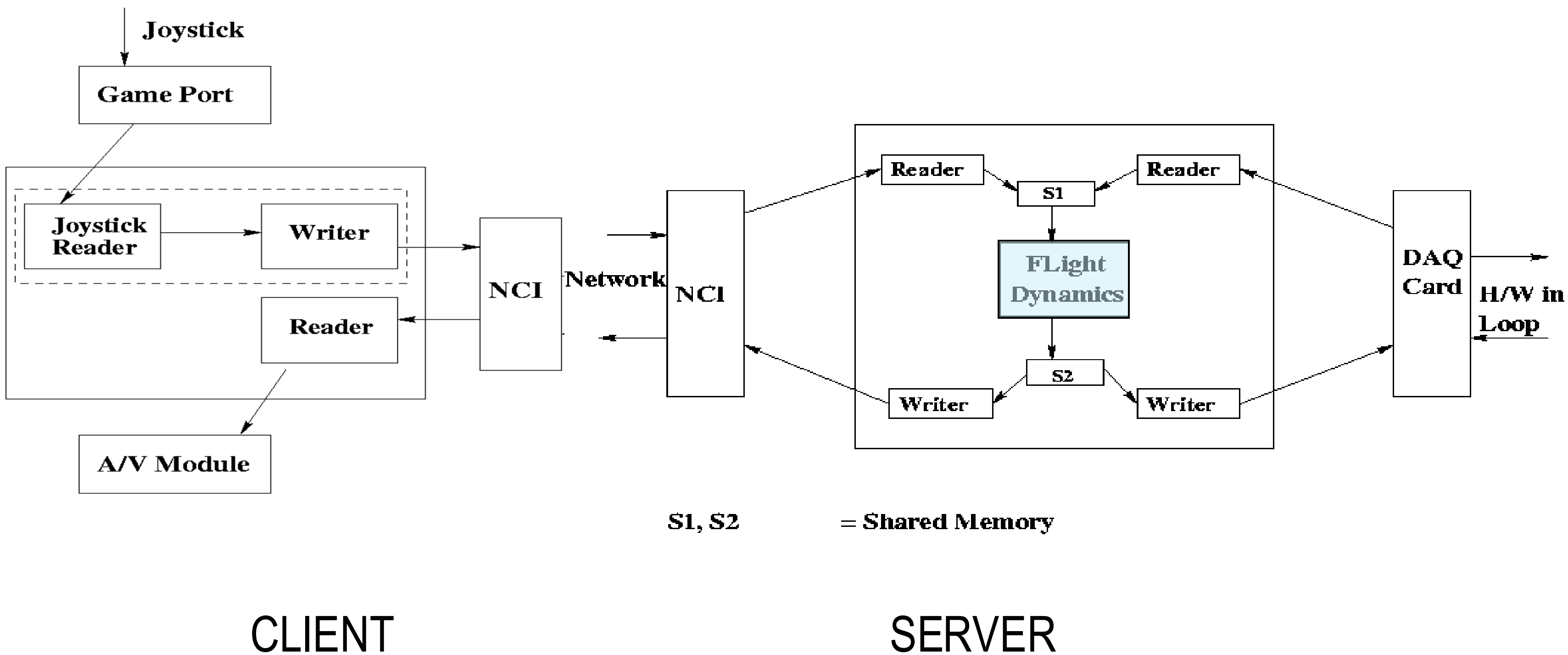
Constraints on responses to pilot inputs, aircraft state updates

Time Periods to meet Timing Requirements



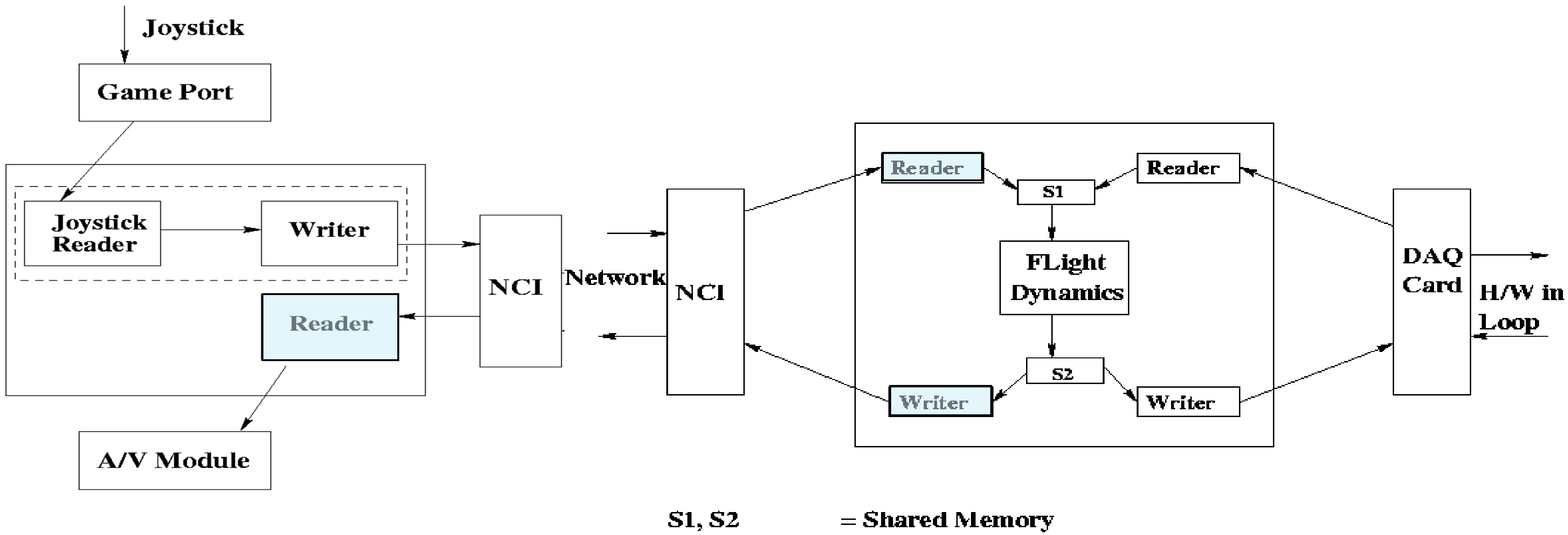
Requirement	Choice Made	Rationale
Continuous pilot inputs should be polled at rates greater than **62.5** *Hz*	The time period of the writer on Client should be less than **16** *ms*	The writer thread on the Client polls for the pilot inputs from the joystick

Time Periods to meet Timing Requirements…



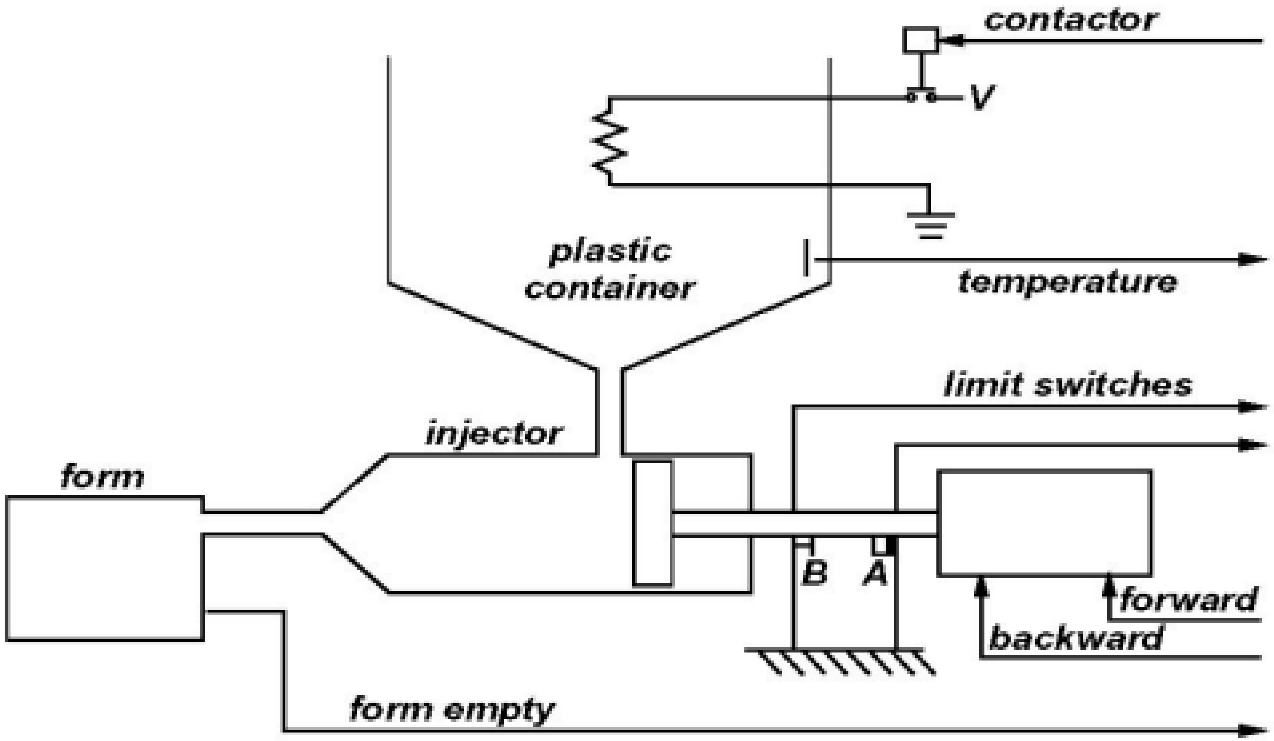
Requirement	**Choice Made**	**Rationale**
The state of the aircraft is to be advanced at **12.5** *ms* time steps	The time period of the Flight Dynamics thread on the Server is **12.5** *ms*	The flight dynamics thread on the Server advances the state of the system

Time Periods to meet Timing Requirements…



Requirement	**Choice Made**	**Rationale**
Response time for pilots should be less than **150** *ms* for commercial aircrafts and **100** *ms* for fighter aircrafts	Reader and Writer threads on Server, and the Reader thread on the Client should be as fast as the system permits. (Time period of **4** *ms* in our case)	• Delay in data transfer at these threads increases the response time • These threads should be interrupt driven in order to minimize the response time

Example 3: Injection Molding



–Keep plastic at proper temperature (liquid, not boiling)

–Control injector solenoid (make sure that the motion of the solenoid terminates before the piston reaches the end of its travel.

Source: “Laboratory for Perceptual Robotics, UMass” Copyright 1996 by Roderic A. Grupen

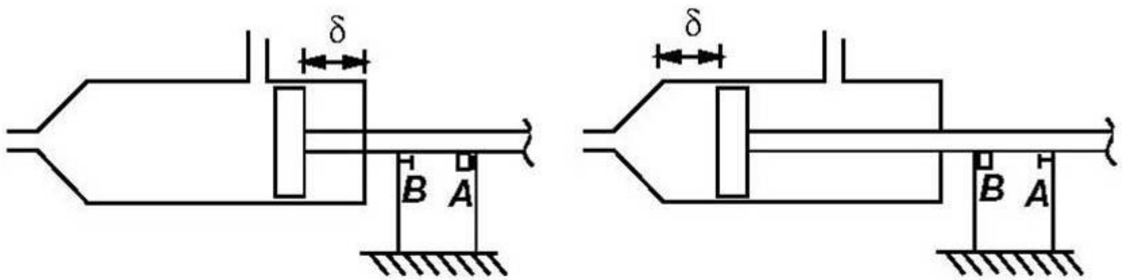
Controlling a reaction

- **We know:**
  - If temperature too high, it explodes
  - Maximum rate of temperature increase
  - Rate of cooling
- **Events:**
  - Temperature change
  - Temperature > safe threshold
- **We can derive:**
  - How often we have to check temperature
  - When we have to finish cooling

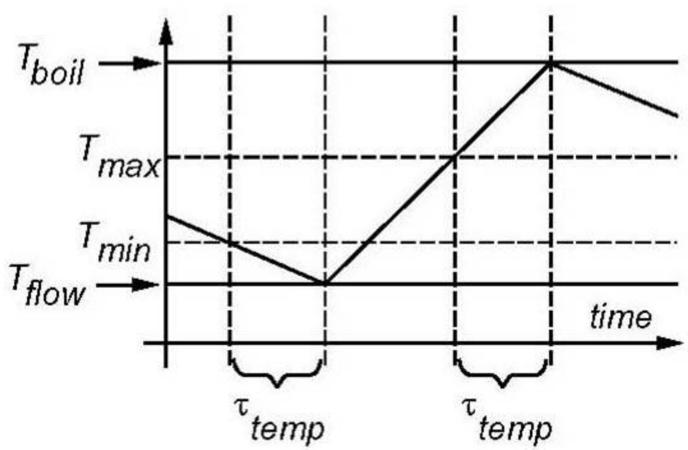
Example - Injection Molding (cont.)

– Timing constraints

- the injector must be off within τ_{inj} seconds after A or B limit signals, so that $v_{inj}\tau_{inj} < \delta$



- the temperature control contactor must be activated within τ_{temp} seconds of a temperature event, so that $T_{flow} < T < T_{boil}$



Example – Injection Molding (cont.)

– Concurrent control tasks

injector control:

```
in position A;
while(1) {
    wait_until(form_empty);
    on(forward);
    wait_until(B);
    off(forward);
    on(backward);
    wait_until(A);
    off(backward);
}
```

temperature control:

```
while(1) {
    analog_in(Temp);
    if (Temp > Tmax) {
        off(contactor);
    }
    else if (Temp < Tmin) {
        on(contactor);
    }
}
```

Plan

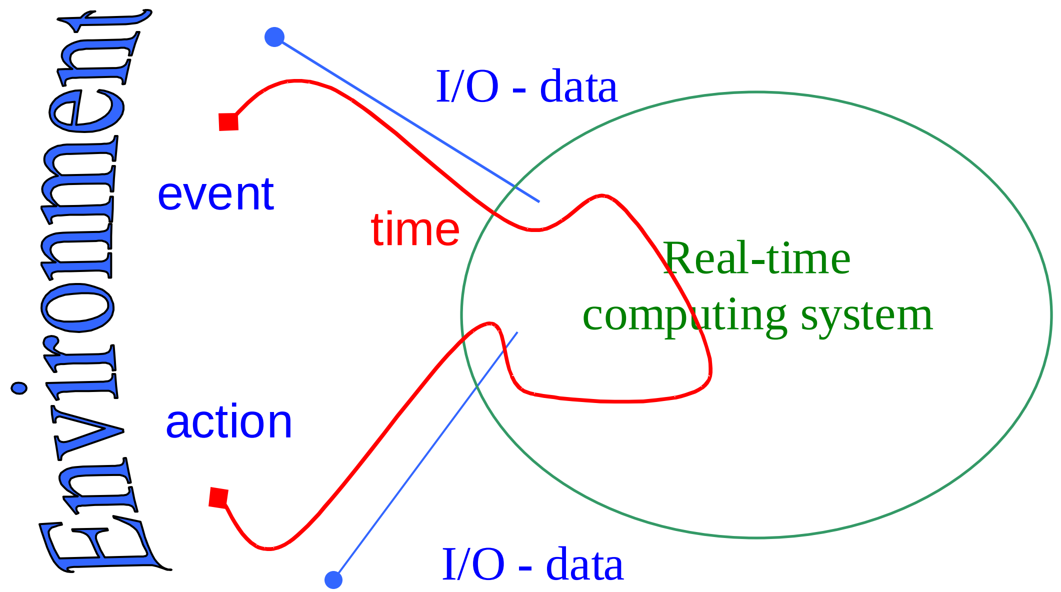
Real-Time Support

- Special Characteristics of Real-Time Systems
- **Real-Time Constraints**
- Canonical Real-Time Applications
- Scheduling in Real-time systems
- Operating System Approaches

What is "real" about real-time?

Computer world e.g., PC	*Real world* industrial system, airplane
Average response for user, interactive	Events occur in environment at own speed
Occasionally longer	Reaction too slow: deadline miss
Reaction: user annoyed	*Reaction:* damage, pot. loss of human life
Computer controls speed of user	*Computer must follow speed of environment*
"Computer time"	"Real-time"

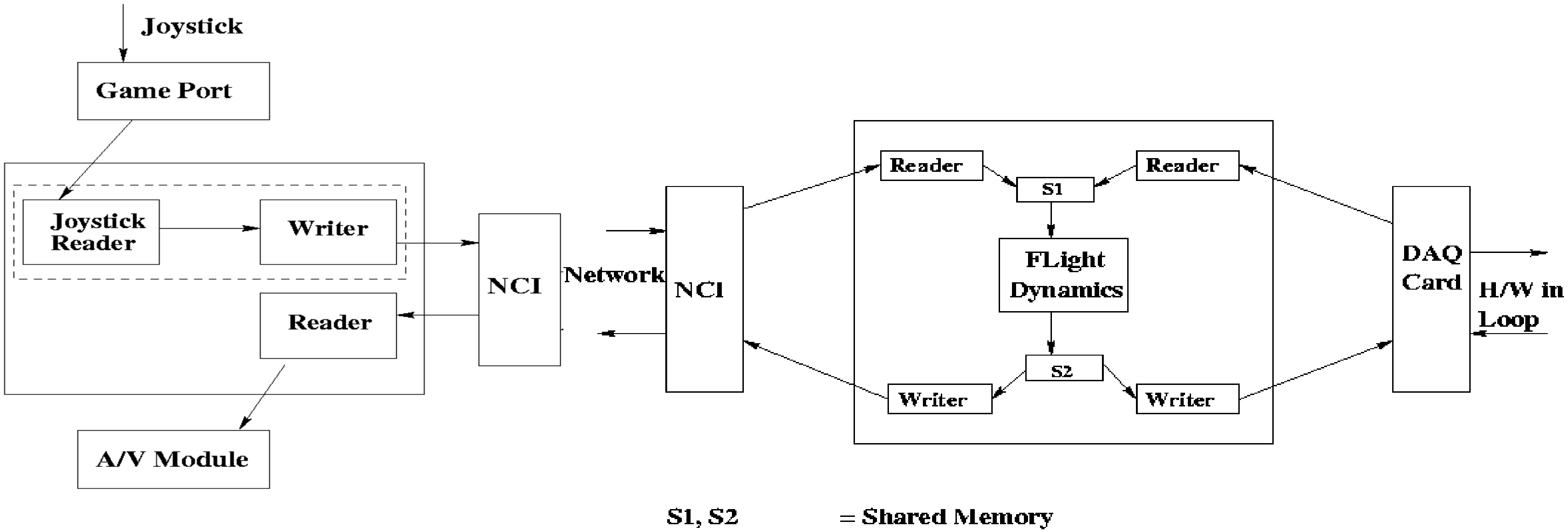
Real-Time Systems



A real-time system is a system that reacts to events in the environment by performing predefined actions

<u>within specified time intervals.</u>

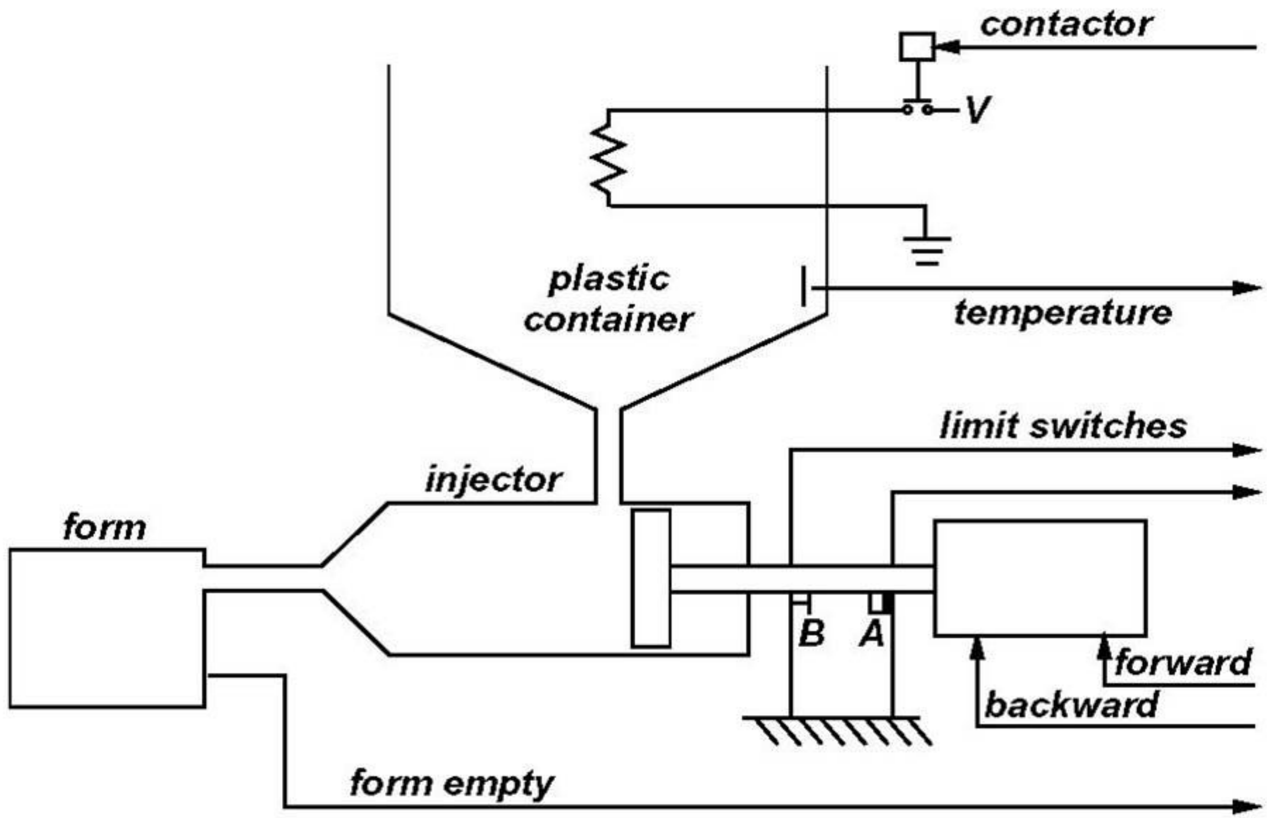
Flight Avionics



CLIENT SERVER

Constraints on responses to pilot inputs, aircraft state updates

Example: injection molding



Constraints:

– **Keep plastic at proper temperature** (liquid, but not boiling)

– **Control injector solenoid**
(make sure motion of piston reaches end of its travel)

Real-Time Systems: Properties of Interest

- ***Safety***: Nothing bad will happen
- ***Liveness***: Something good will happen
- ***Timeliness***:
  Things will happen on time -- by their deadlines, periodically, ...

In a Real-Time System….

Correctness of results depends on value
*and its **time** of delivery*

Correct value **delivered *too late* is *incorrect***

e.g., traffic light: light must be green *when crossing*, not enough before

***Real*-time:**

(Timely) reactions to events *as they occur*, at their pace:

(real-time) system (internal) time same time scale as environment (external) time

Performance Metrics in Real-Time Systems

- Beyond minimizing response times and increasing the throughput:
  - *achieve timeliness*.
- More precisely, how well can we predict that deadlines will be met?

Types of RT Systems

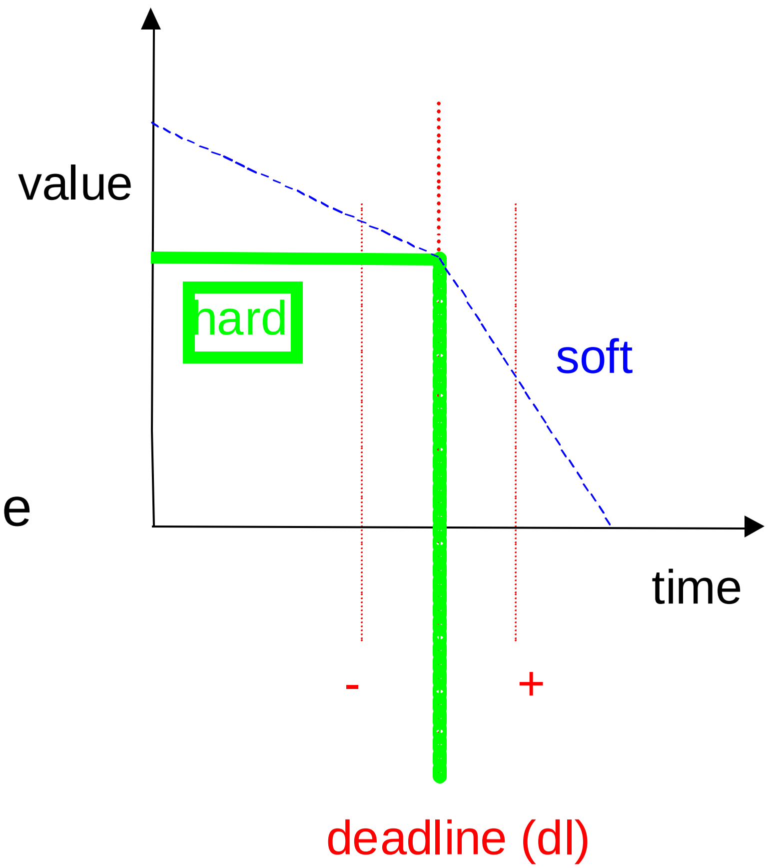
Dimensions along which real-time activities can be categorized:

- *How tight are the deadlines?*
  --deadlines are tight when **laxity** (deadline -- computation time) is small.
- *How strict are the deadlines?*
  --what is the value of executing an activity after its deadline?
- *What are the characteristics of the environment?*
  --how static or dynamic must the system be?

Designers want their real-time system to be *fast*, *predictable*, *reliable*, *flexible*.

Hard, soft, firm

- **Hard**
  result useless or dangerous
  if deadline exceeded
- **Soft**
  result of some - lower -
  value if deadline exceeded
- **Firm**
  If value drops to zero at deadline



Deadline intervals:
result required not later
and not before

Examples

- **Hard real time systems**
  - Aircraft
  - Airport landing services
  - Nuclear Power Stations
  - Chemical Plants
  - Life support systems
  - …
- **Soft real time systems**
  - Multimedia
  - Interactive video games
  - ATM response
  - …

Real-Time: Items and Terms

Task

- program, perform service, functionality
- requires resources, e.g., execution time

Deadline

- specified time for completion of, e.g., task
- time interval or absolute point in time
- value of result may depend on completion time

Plan

- Special Characteristics of Real-Time Systems
- Real-Time Constraints
- **Canonical Real-Time Applications**
- Scheduling in Real-time systems
- Operating System Approaches

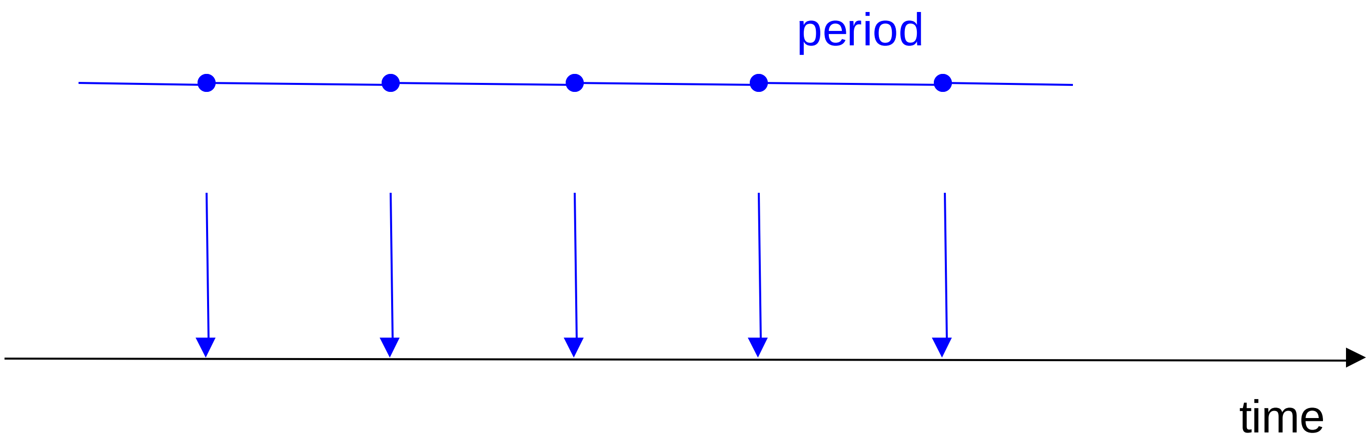
Timing Constraints

Real-time means to be in time ---

How do we know something is "in time"?
How do we express that?

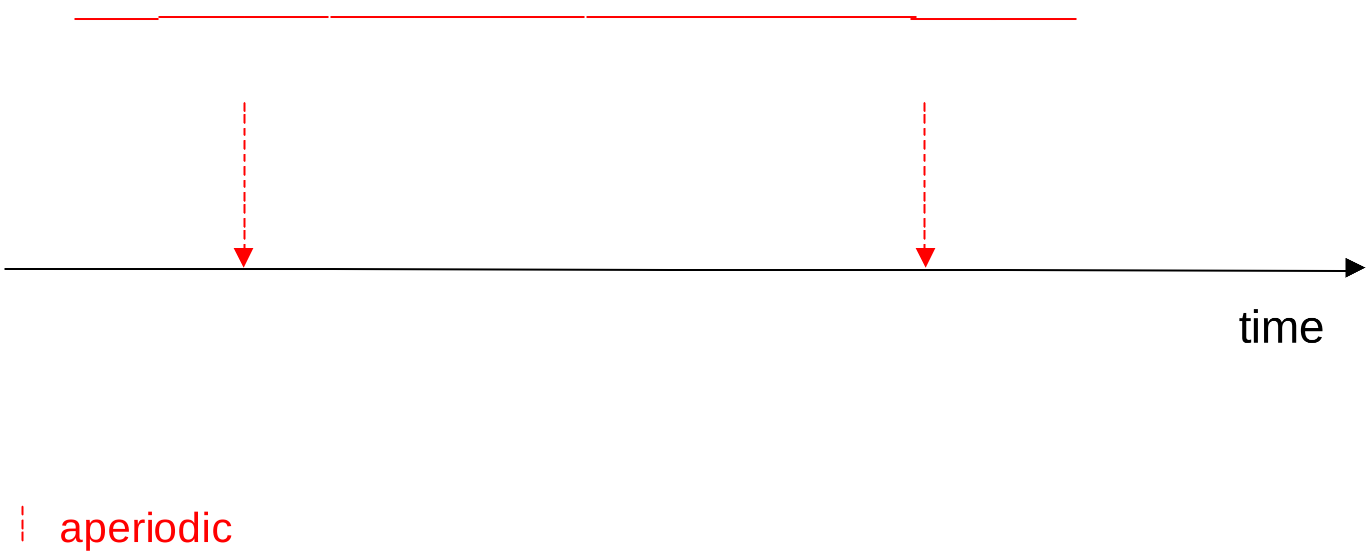
- *Timing constraints* **to specify temporal correctness**
  e.g., "finish assignment by 2pm",
  "be at station before train departs".
- **System said to be *(temporally) feasible*,**
  if it meets all specified timing constraints.
- **Timing constraints do not come out of thin air:**
  Design process identifies *events*, derives models, and finally *specifies* timing constraints

- **Periodic**
  - Activity occurs repeatedly
  - e.g., to monitor environment values, temperature, etc.

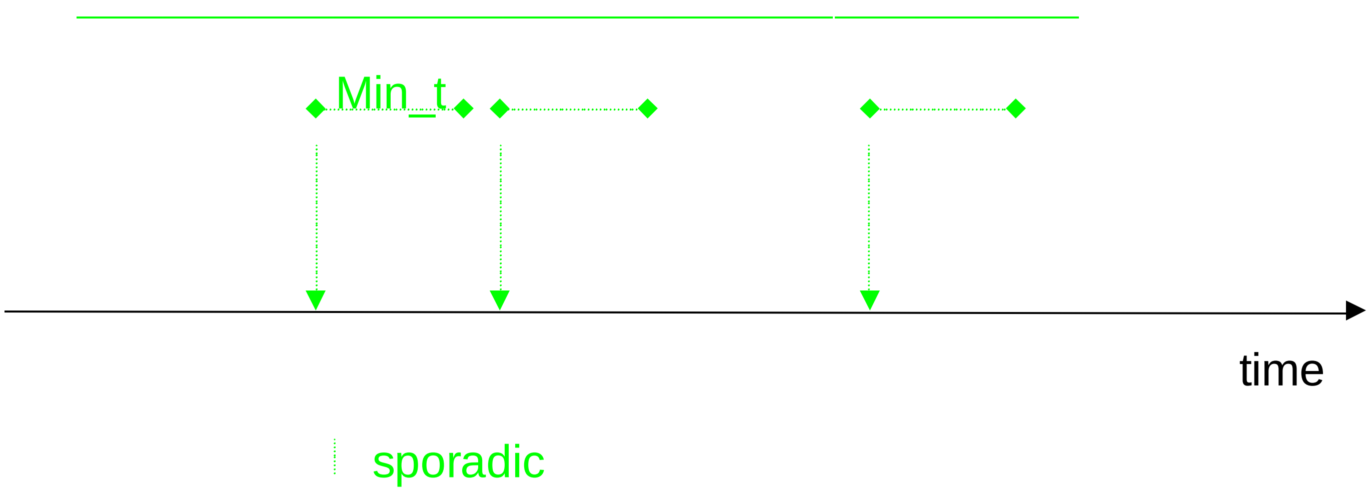


periodic

- **Aperiodic**
  - Can occur any time
  - No arrival pattern given



- **Sporadic**
  - Can occur any time, but
  - Minimum time between arrivals



Who initiates (triggers) actions?

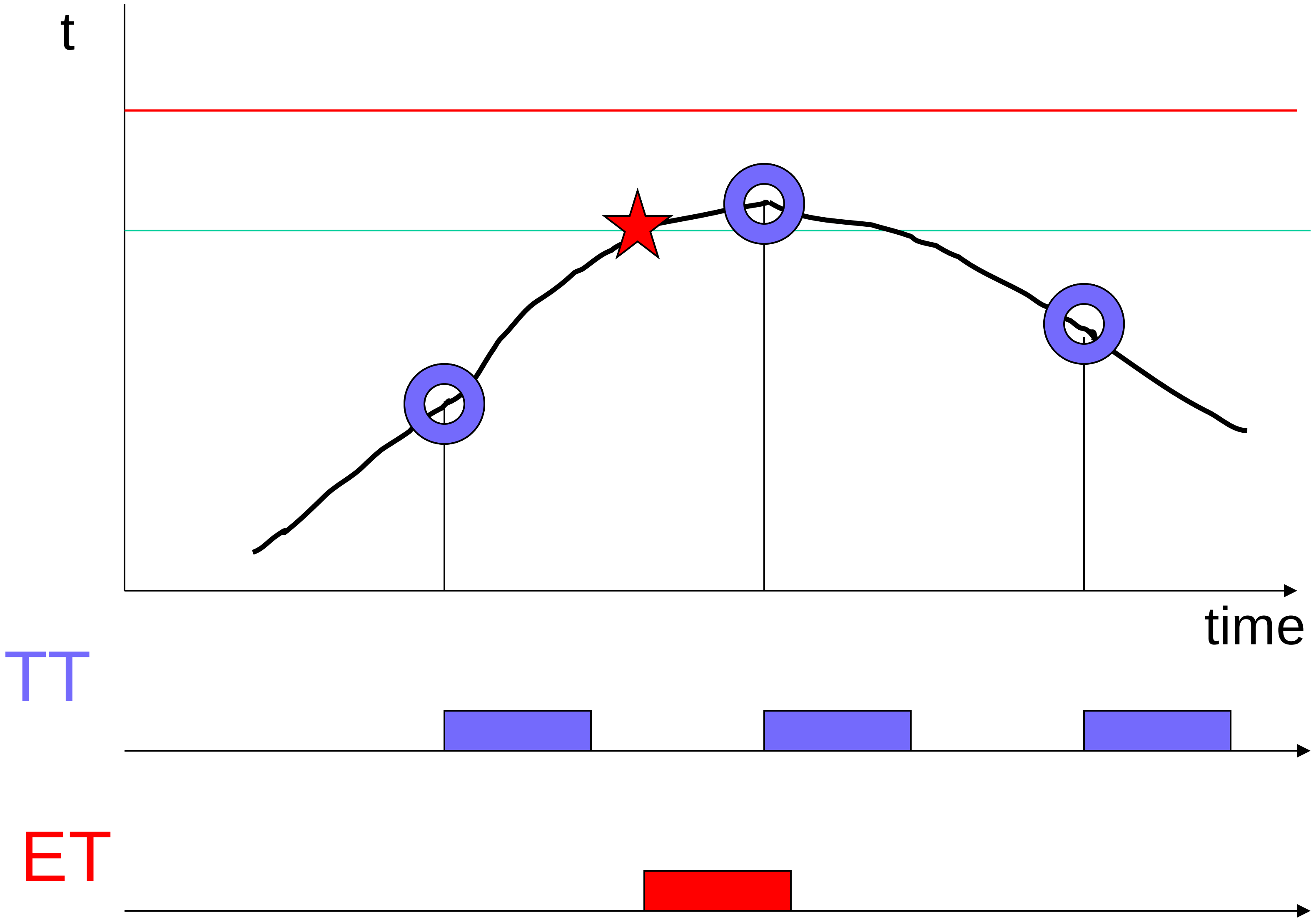
Example: Chemical process

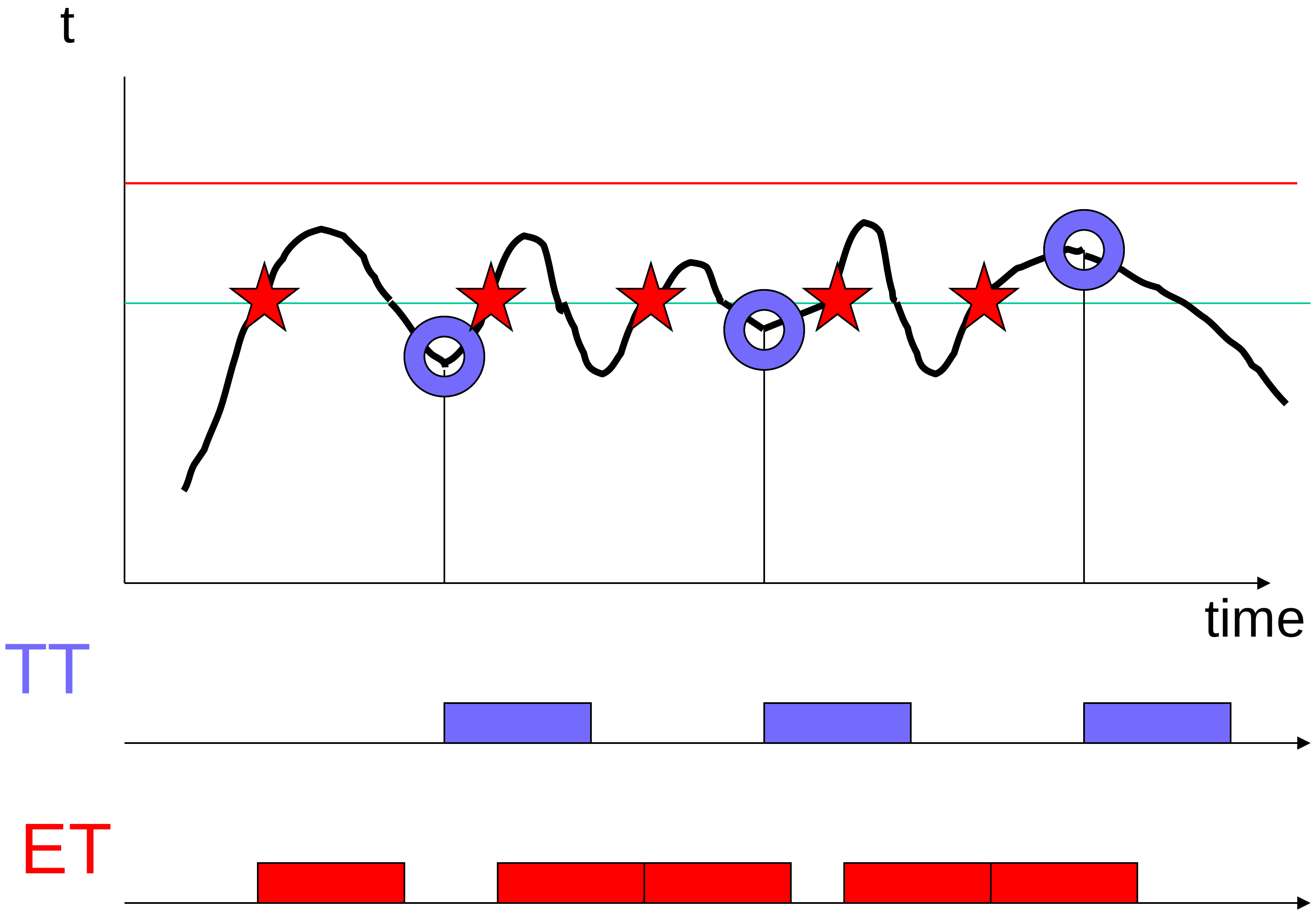
- Controlled so temperature stays below *danger* level
- Warning triggered before danger point

  …… so cooling can still occur

Two possibilities:

- Action whenever temp raises above *warn;*
  event triggered
- Look every *int* time intervals;
  action when temp if measures above *warn*
  time triggered


t
time
TT
ET


t
time
TT
ET

ET vs TT

- **Time triggered**
  - Stable number of invocations
- **Event triggered**
  - Only invoked when needed
  - High number of invocation and computation demands if value changes frequently

Slow down the environment?

- **Importance**
  - Which parts of the system are important?
  - Importance can change over time
    e.g., fuel efficiency during emergency landing
- **Flow control**
  Who has control over speed of processing?
  Who can slow partner down?
  - Environment
  - Computer system

  RT: environment cannot be slowed down

Other Issues to worry about

- **Meet requirements -- some activities may run only:**
  - After others have completed - *precedence constraints*
  - While others are not running - *mutual exclusion*
  - Within certain times - *temporal constraints*
- **Scheduling**
  - Planning of activities, such that required timing is kept
- **Allocation**
  - Where should a task execute?

In Summary

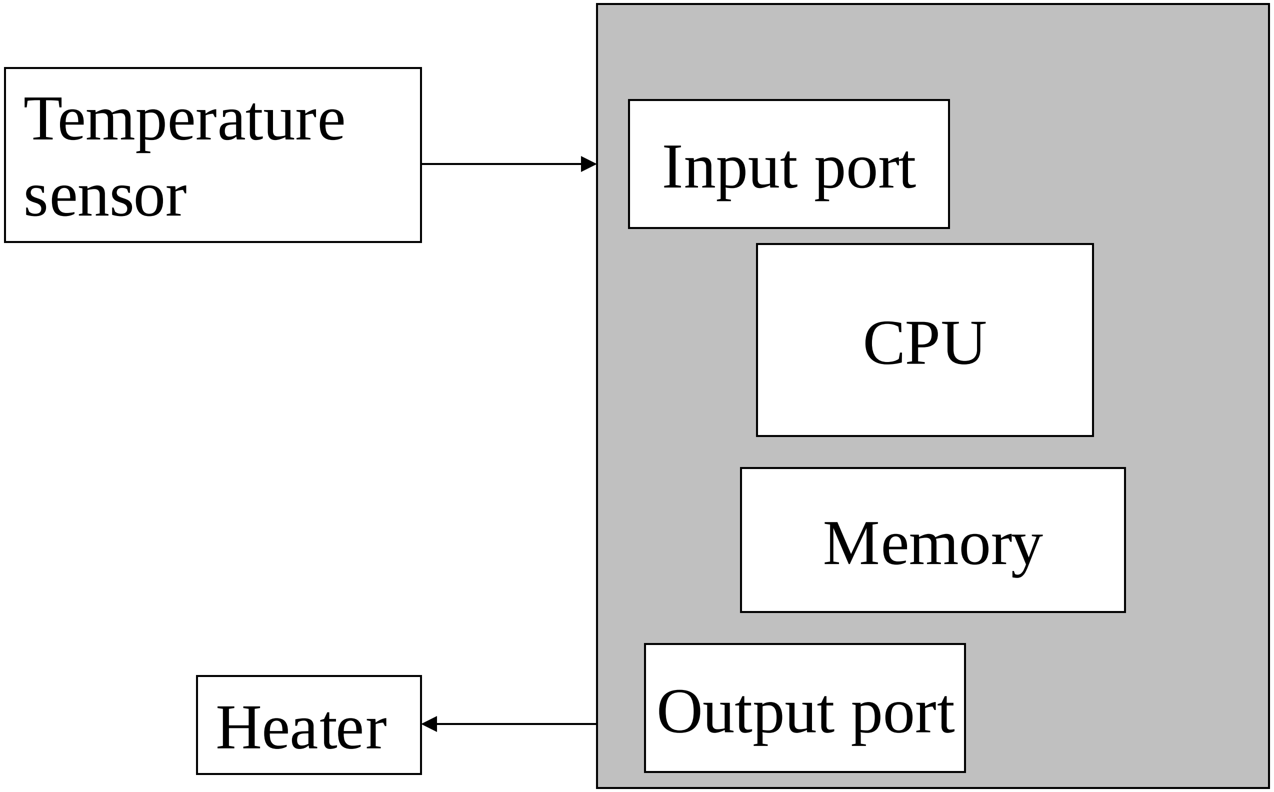
- **Examples:**
  - Engine ECU
  - Flight simulator
  - Injection molding
- **Definitions:**
  - What is “real” about realtime systems
  - Performance metrics in realtime systems: “timeliness”
  - Types of RT systems: nature of deadlines, hard, soft, firm
- **Timing constraints**
  - Periodic, aperiodic, sporadic
  - Event driven vs time-triggered systems
  - Other issues: requirements, scheduling, resource allocation

Plan

- Special Characteristics of Real-Time Systems
- Real-Time Constraints

- **Canonical Real-Time Applications**
- **Scheduling in Real-time systems**
- **Operating System Approaches**

A Typical Real time system



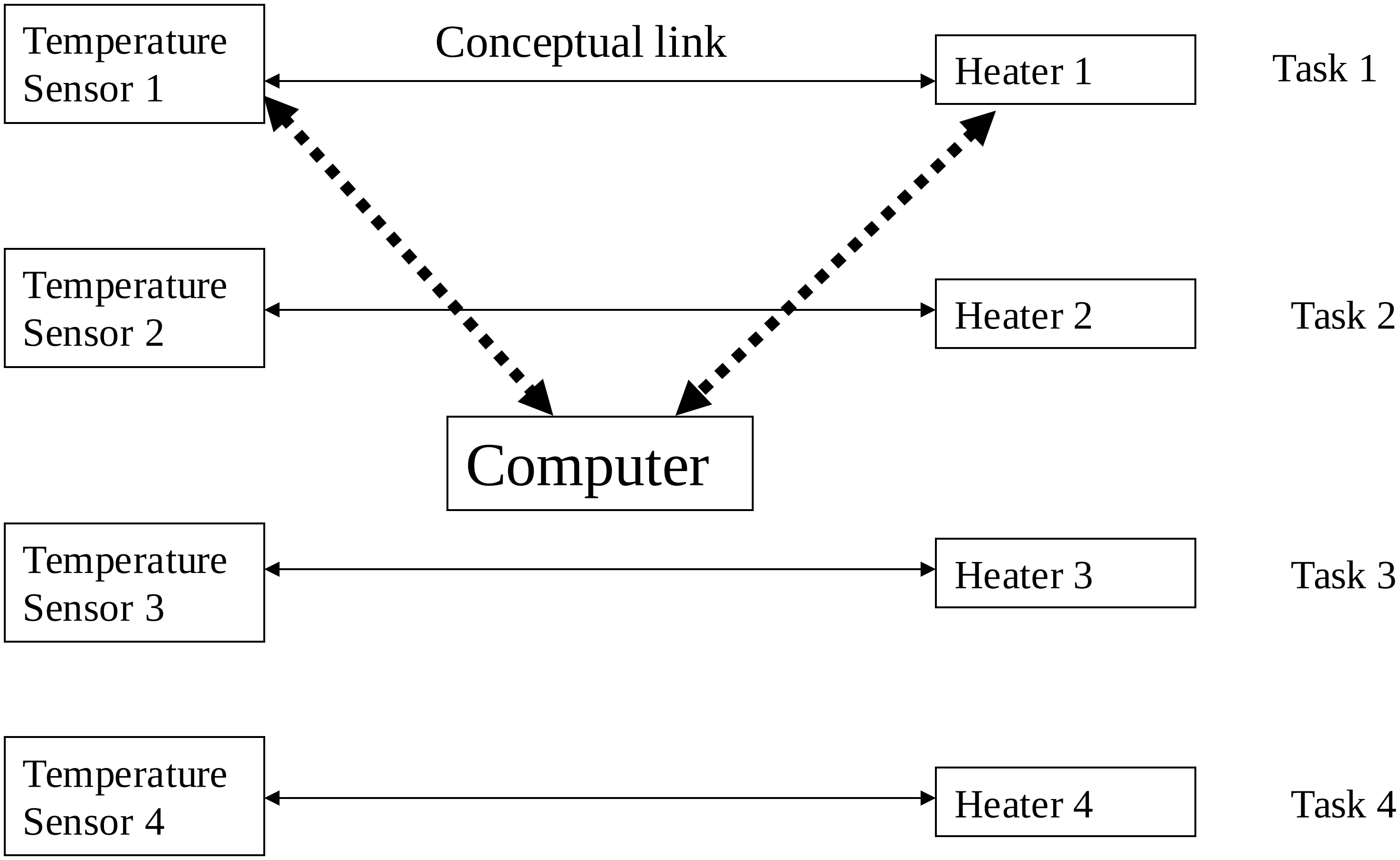
Code for example

```
While true do
        {
        read temperature sensor
        if temperature too high
            then turn off heater
            else if temperature too low
                  then turn on heater
                   else nothing
        }
```

Comment on code

- **Code is by** Polling **device (temperature sensor)**
- **Code is in form of infinite loop**
- **No other tasks can be executed**
- **Suitable for dedicated system or sub-system only**

Extended polling example



Polling

- Problems
  - Arranging task priorities
  - Round robin is usual within a priority level
  - Urgent tasks are delayed

Interrupt driven systems

- **Advantages**
  - Fast
  - Little delay for high priority tasks
- **Disadvantages**
  - Programming
  - Code difficult to debug
  - Code difficult to maintain

How can we monitor a sensor every 100 ms

Initiate a task T1 to handle the sensor

T1:

Loop

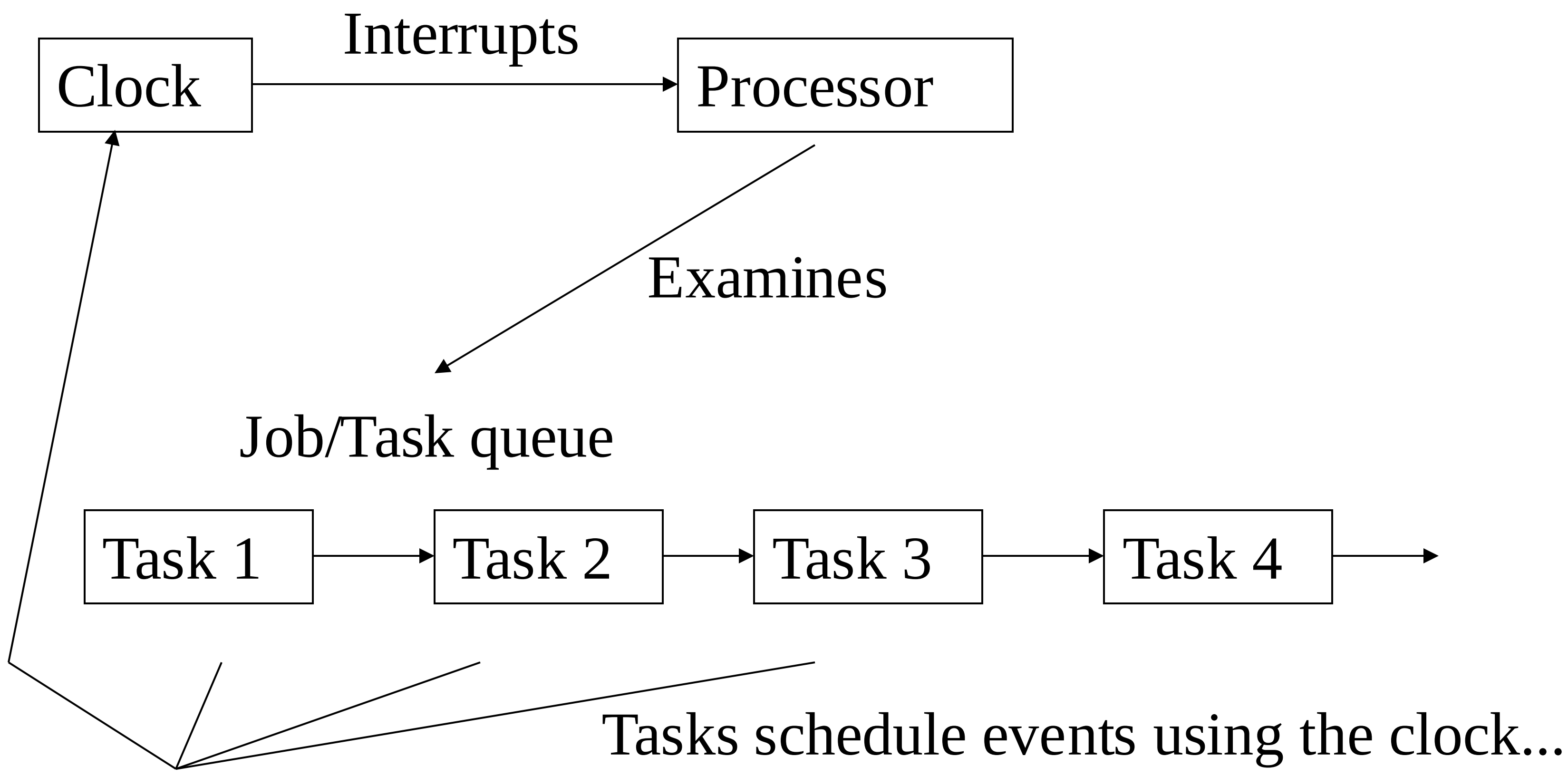
{Do sensor task T2

Schedule T2 for +100 ms

}

Note: time could be **relative** (as here) or an **actual** time - there would be slight differences between the methods, due to the additional time to execute the code.

Clock, interrupts, tasks



Average vs. Worst Case

"There was a man who drowned crossing a river with an average depth of 20cm."

Standard computing is concerned with *average* behavior of system

- e.g., if it responds fast enough on average, it is acceptable
- **Word processing, etc.**

Real-time computing requires timely behavior *in all given situations*

- e.g., a *single* value delivered too late can cause the system to *fail*, even when the average timing is kept
- **Air planes, power plants, etc.**

Performance Metrics in Real-Time Systems

- **Beyond minimizing response times and increasing the throughput:**
  - *Achieve timeliness*.
- **More precisely, how well can we predict that deadlines will be met?**

What is Predictability

A simplified definition:

Ability to reliably determine whether activity will meet its deadline

A complete definition can be quite complex and -- subject to assumptions about failure, etc.

Achieving Predictability

- ***Layer by layer*:**
  - Predictability requirement of an activity percolates down/up the layers.

- ***Top-layer:***
  - Do your best to meet the deadline of an activity
  - If deadline cannot be met, handle the exception predictably

Layer-by-layer Predictability

***Applications*:**	**Activities having deadlines.**
Processes:	**Processes with deadlines.** **One or more processes make up an activity.**
Tasks:	**Tasks with deadlines** **Multiple (precedence-related) tasks make up a process.**
***Operating System*:**	**Bounded code execution time.** **Bounded Operating System primitives.** **Bounded synchronization costs.** **Bounded scheduling costs.**
Architecture:	**Bounded memory access times.** **Bounded instruction execution times.** **Bounded inter-node communication costs.**

Issues to worry about

- **Meet requirements -- some activities may run only:**
  - After others have completed - *precedence constraints*
  - While others are not running - *mutual exclusion*
  - Within certain times - *temporal constraints*
- **Scheduling**
  - Planning of activities, such that required timing is kept
  - Models
  - Methods

Issue: Running time of programs

- Depends on hardware, memory wait cycles, clock etc.
- Depends on virtual memory and caching
- Depends on data
- Depends on program control
- Depends on pipelining
- Depends on memory management

More Issues…

- **Operating Systems**
  - Off-the shelf (COTS)
  - Application-specific

- **Dataflow**
  - Data read from environment - sensors
  - Data processed by tasks
  - Results processed by other tasks…
  - Data transferred to environment
  - Actuators

Further Issues

- **Programming languages**
  - Maximum execution times
  - Difficult with recursions, unbounded loops, etc
- **Networks**
  - Bounded transmission times
  - e.g., Ethernet, CSMA/CD large number of collisions
- **Synchronization of clocks**
  - Clock drift apart
  - Faulty clocks can cause wrong synchronization, e.g., for average

Even more issues…

- **Fault tolerance**
  - Deadlines cannot be kept if computer or network has hardware errors
  - Tolerate certain faults within timing
- **Real-Time Databases**
  - maintain data consistent and fresh
- **Design**
  - derive and specify timing

In Summary

- **Polling or Interrupt driven systems**
- **Average vs Worst case scenario**
  - Man who drowned crossing a river with average depth of 20cm
- **Performance metrics in RT Systems**
  - Notion of timeliness
- **Predictability**
  - Whether we can meet deadlines
  - Building predictability layer-by-layer
- **Other issues:**
  - OS, networks, clock synchronisation, fault tolerance, RT-databases, etc.