2000 IARE EDUCATION FOR LIBERATION

Presentation on
Principles of Distributed Embedded Systems
(Embedded Systems - ECE)

I -M.TECH I -Semester

(AUTONOMOUS-R18)

Prepared by,

Dr. S. Vinoth

Associate Professor

UNIT - I
REAL TIME ENVIRONMENT

UNIT - I

REAL-TIME ENVIRONMENT

- Real-time computer system requirements
- classification of real time systems
- simplicity, global time
- internal and external clock synchronization
- real time model. Real time communication
- temporal relations, dependability
- power and energy awareness
- real time communication
- event triggered
- rate constrained
- time triggered.

What is an Embedded system?

Embedded systems (ES) = **information processing systems embedded into a larger product**

Examples:

Main reason for buying is **not** information processing

What is a real-time system?

- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period

  –the correctness depends not only on the logical result but also the time it was delivered

  –failure to respond is as bad as the wrong response!

- **The computer is a component in a larger engineering system => EMBEDDED COMPUTER SYSTEM 99% of all processors are for the embedded systems market**

Terminology

- **Hard real-time** — systems where it is absolutely imperative that responses occur within the required deadline. E.g. Flight control systems.
- **Soft real-time** — systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. E.g. Data acquisition system.
- **Real real-time** — systems which are hard real-time and which the response times are very short. E.g. Missile guidance system.
- **Firm real-time** — systems which are soft real-time but in which there is no benefit from late delivery of service.

A single system may have all hard, soft and real real-time subsystems In reality many systems will have a cost function associated with missing each deadline.

Characteristics of a RTS

- Large and complex — vary from a few hundred lines of assembler or C to 20 million lines of Ada estimated for the Space Station Freedom

- Concurrent control of separate system components — devices operate in parallel in the real-world; better to model this parallelism by concurrent entities in the program

- Facilities to interact with special purpose hardware — need to be able to program devices in a reliable and abstract way

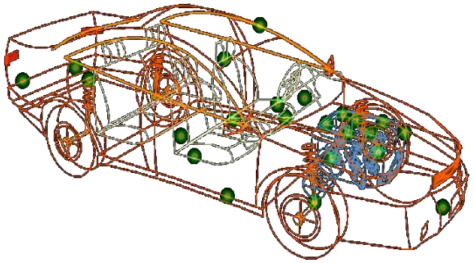
Characteristics of a RTS cont……

- **Extreme reliability and safe** — embedded systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss

- **Guaranteed response times** — we need to be able to predict with confidence the worst case response times for systems; efficiency is important but predictability is essential

Embedded computing systems are becoming pervasive in our society (more than 10^9 units/year):

- **Robotics**
- **Flight control systems**
- **Plant control**
- **Automotive**
- **Consumer electronics**
- **Multimedia systems**
- **Sensor/Actor**



Criticality

QoS management

High performance

Safety critical

firm

hard

Timing contraints

Common features

- In these diversified domains some shared features can be identified:
- Dedicated function (vs general-purpose computers)
- Reactive / Interactive
- Real-time
- Constraints on several metrics:
  cost, power, performance, noise, weight, size, flexibility, maintainability, correctness, safety, time-to-market

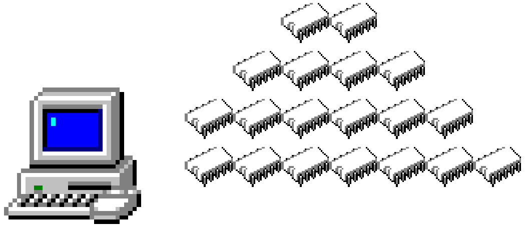
Embedded systems overview

- Embedded computing systems
- Computing systems embedded within electronic devices
- Hard to define. Nearly any computing system other than a desktop computer
- Billions of units produced yearly, versus millions of desktop units
- Perhaps 50 per household and per automobile

Computers are in here...

and here...

and even here...


Lots more of these, though they cost a lot less each.

A “short list” of embedded systems

- Anti-lock brakes
- Auto-focus cameras
- Automatic teller machines
- Automatic toll systems
- Automatic transmission
- Avionic systems
- Battery chargers
- Camcorders
- Cell phones
- Cell-phone base stations
- Cordless phones
- Cruise control
- Curbside check-in systems
- Digital cameras
- Disk drives
- Electronic card readers
- Electronic instruments
- Electronic toys/games
- Factory control
- Fax machines
- Fingerprint identifiers
- Home security systems
- Life-support systems
- Modems
- MPEG decoders
- Network cards
- Network switches/routers
- On-board navigation
- Pagers
- Photocopiers
- Point-of-sale systems
- Portable video games
- Printers
- Satellite phones
- Scanners
- Smart ovens/dishwashers
- Speech recognizers
- Stereo systems
- Teleconferencing systems
- Televisions
- Temperature controllers
- Theft tracking systems
- TV set-top boxes
- VCR’s, DVD players
- Video game consoles

And the list goes on and on

Common characteristics of Embedded systems

Some common characteristics of embedded systems

- **Single-functioned**

–Executes a single program, repeatedly

- **Tightly-constrained**

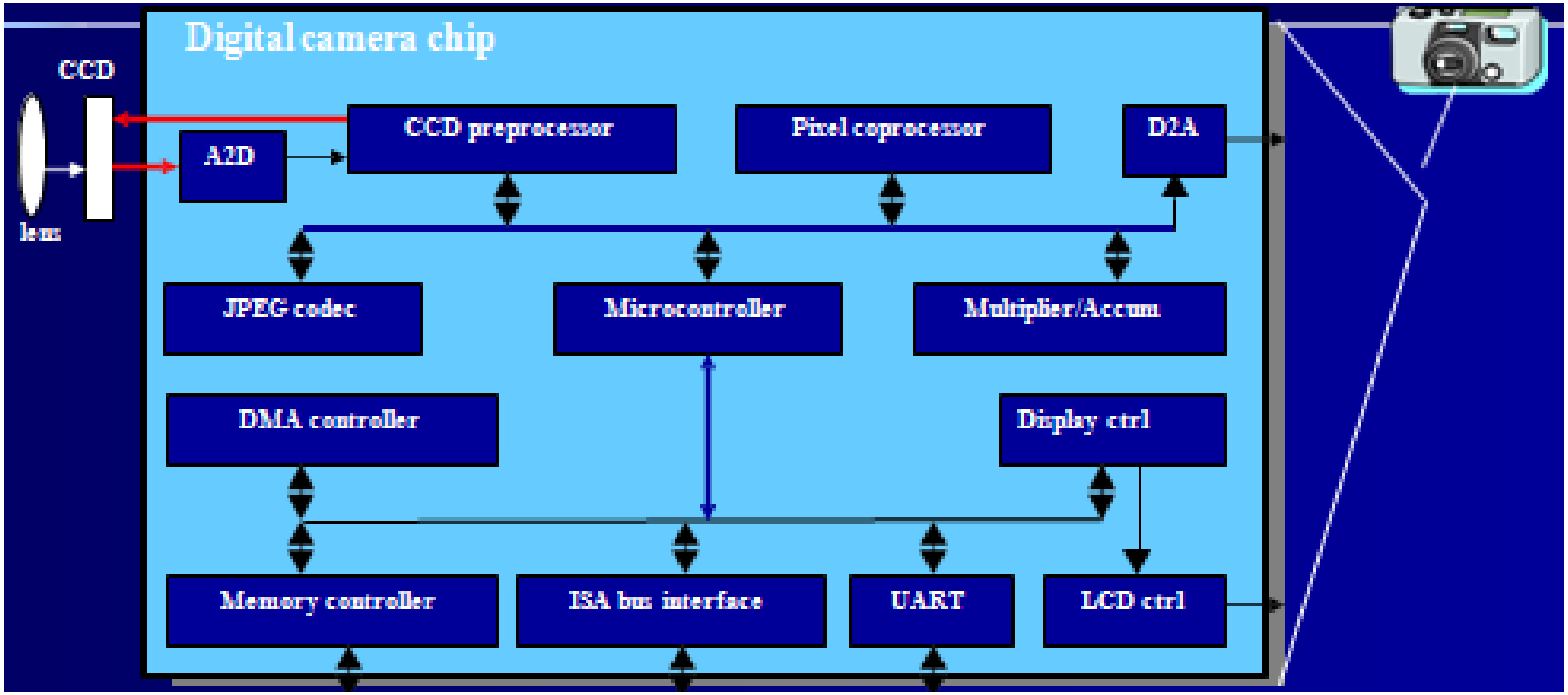
–Low cost, low power, small, fast, etc.

- **Reactive and real-time**

–Continually reacts to changes in the system’s environment

–Must compute certain results in real-time without delay

An embedded system example – Digital camera



- Single-functioned -- always a digital camera
- Tightly-constrained -- Low cost, low power, small, fast
- Reactive and real-time -- only to a small extent

What is real-time?

What is real-time? Is there any other kind?

- A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced.

- By system behavior we mean the sequence of outputs in time of a system.

Real-time means reactive

- A real-time computer system must react to stimuli from its environment

- The instant when a result must be produced is called a deadline.

- If a result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm.

- If severe consequences could result if a firm deadline is missed, the deadline is called hard.

Example: Consider a traffic signal at a road before a railway crossing. If the traffic signal does not change to red before the train arrives, an accident could result.

Reliability

- The Reliability R(t) of a system is the probability that a system will provide the specified service until time t, given that the system was operational at the beginning (t-t0).

- The probability that a system will fail in a given interval of time is expressed by the failure rate, measured in FITs (Failure In Time).

- A failure rate of 1 FIT means that the mean time to a failure (MTTF) of a device is 10^9 h, i.e., one failure occurs in about 115,000 years.

- If a system has a constant failure rate of λ failures/h, then the reliability at time t is given by,

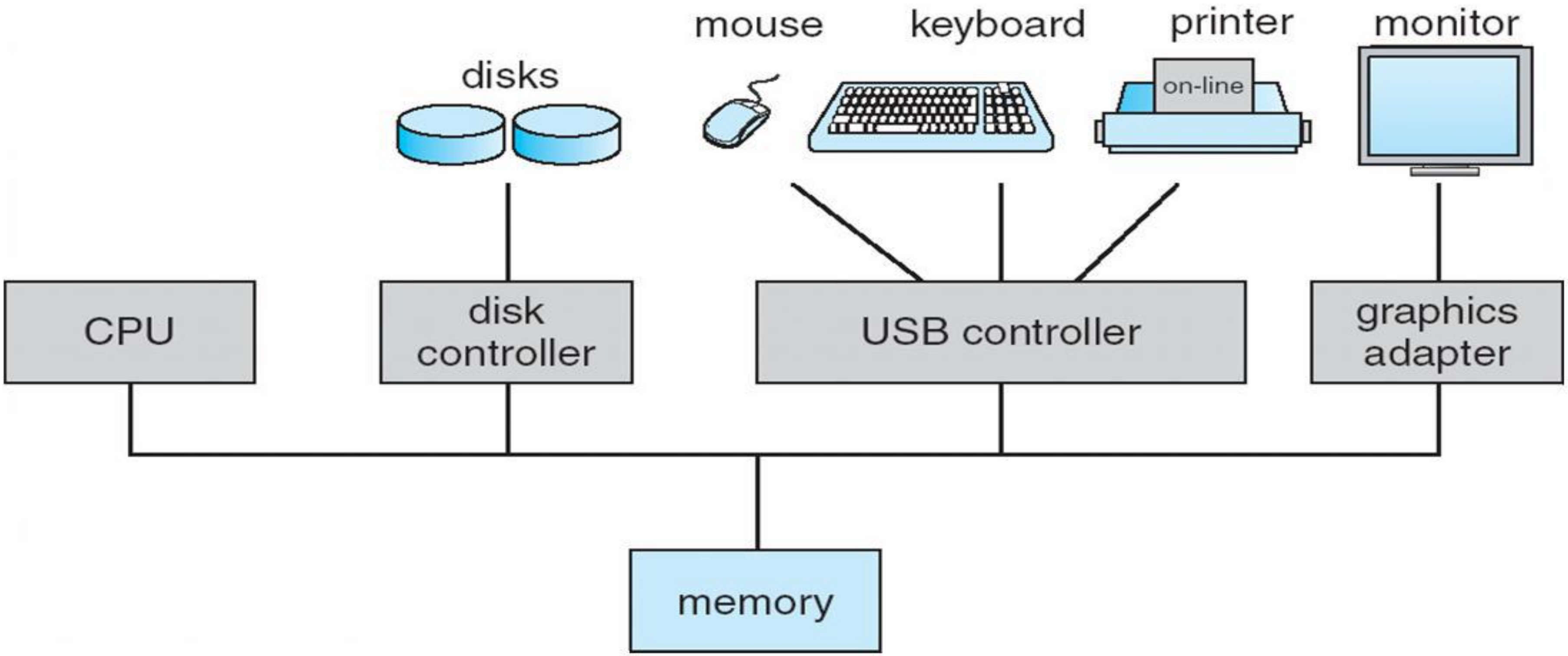
$$R(t)= \exp(-\lambda(t-to))$$

$$MTTF = 1/\lambda$$

Computer System Organization

Computer-system operation

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles



Distributed Systems

- Computation is distributed among several processors. In contrast to tightly-coupled systems, the processors do not share a clock or memory. Each has its own local memory.

- Communication is via a network. These systems are termed loosely-coupled or distributed systems. The processors vary in size and function and are called nodes.

Advantages of distributed systems:

- **Reliability:** If one node fails, the remaining nodes can continue operating. So by building enough redundancy, the system will not fail if one or more nodes fail (e.g. redundant web servers).
- **Computation speedup:** Computation can be distributed among various nodes to run concurrently (e.g. load balanced web servers).

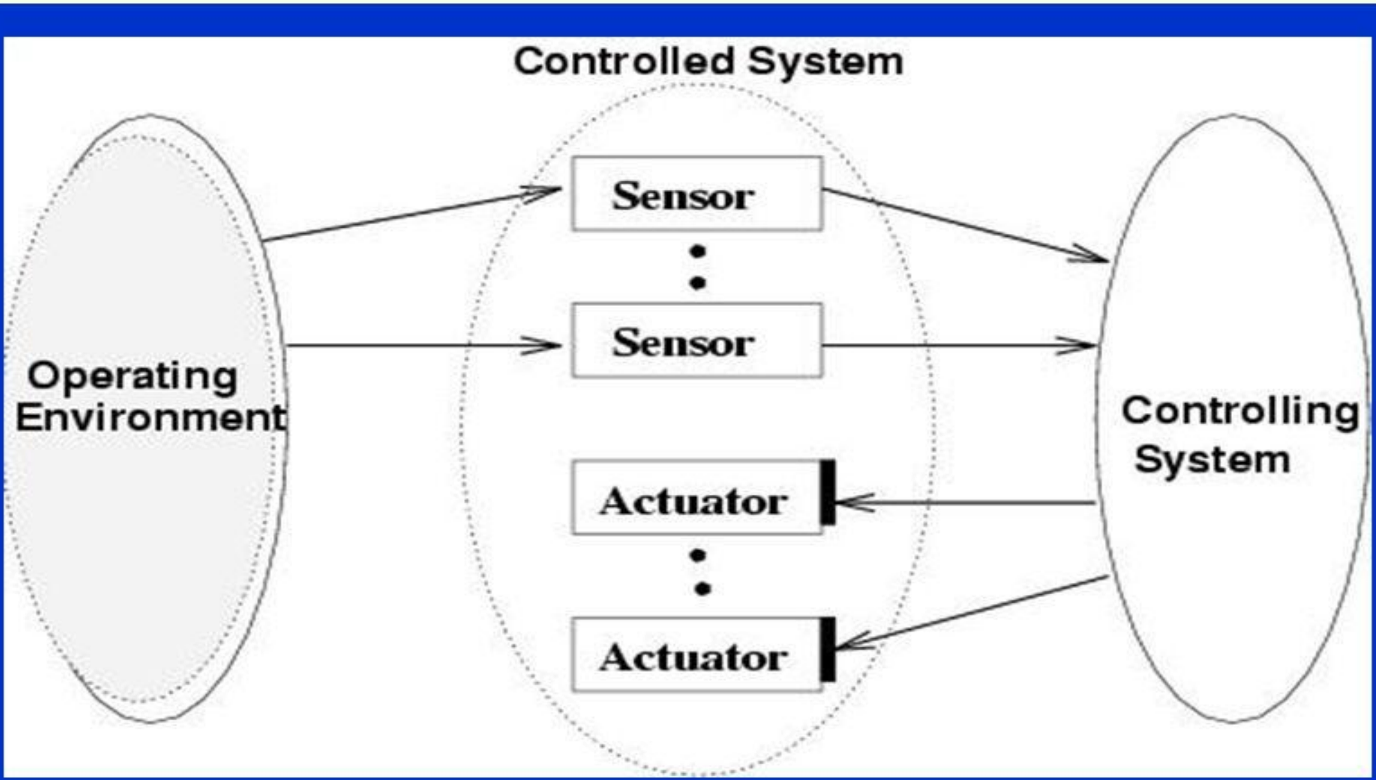
Distributed Systems (contd.)

- Resource Sharing: Software, data, and hardware resources can be shared. E.g. data files in node A can be accessed by a user at node B. Files can be printed at a shared laser printer.

- Communication: Processes at various nodes can exchange information.

Special Purpose Systems

A typical real-time embedded system



4

WHEN IS A COMPUTER SYSTEM REAL-TIME?

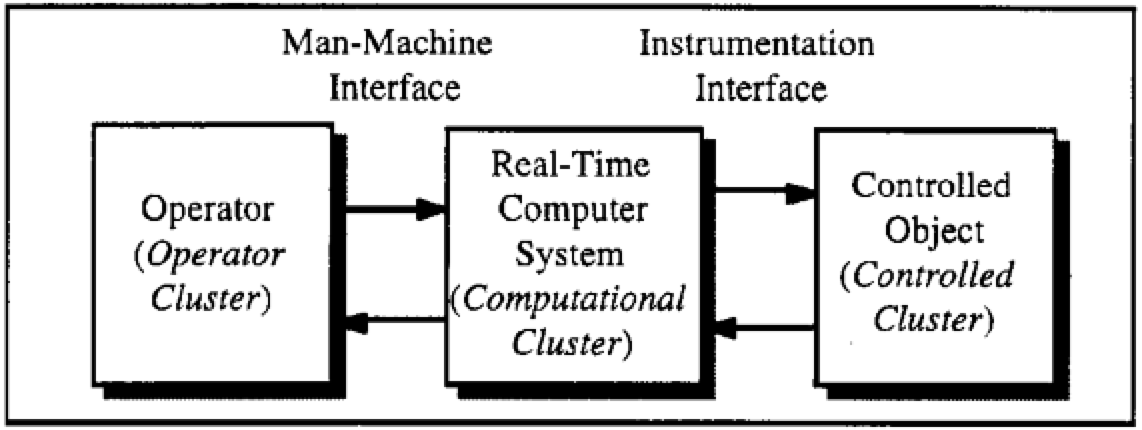
- A real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced.

- A real-time computer system is always part of a larger system–this larger system is called a real-time system.

- A real-time system changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped.

- It is reasonable to decompose a real-time system into a set of sub-systems called clusters.

Real- Time system cont…..



Man-Machine Interface
Instrumentation Interface
Operator (Operator Cluster)
Real-Time Computer System (Computational Cluster)
Controlled Object (Controlled Cluster)

Figure 1.1: Real-time system.

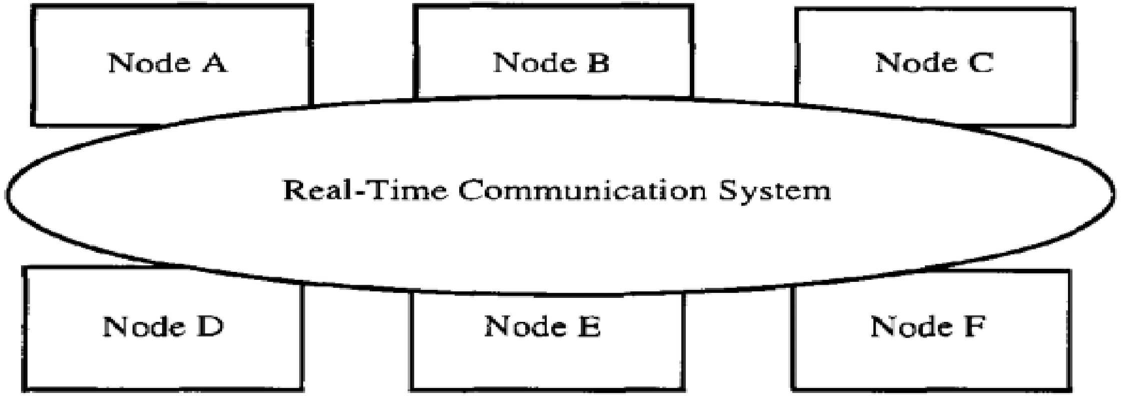


Figure 2.1: Distributed computer system.

- We refer to the controlled object and the operator collectively as the environment of the real-time computer system.

- If the real-time computer system is distributed, it consists of a set of computer) nodes interconnected by a real-time communication network

Real- Time system cont…..

- The interface between the human operator and the real-time computer system is called the man-machine interface, and the interface between the controlled object and the real-time computer system is called the instrumentation interface.

- The man-machine interface consists of input devices (e.g., keyboard) and output devices (e.g., display) that interface to the human operator.

- The instrumentation interface consists of the sensors and actuators that transform the physical signals (e.g., voltages, currents) in the controlled object into a digital form and vice versa.

- A node with an instrumentation interface is called an interface node.

Real- Time system cont…..

- A real-time computer system must react to stimuli from the controlled object (or the operator) within time intervals dictated by its environment.

- The instant at which a result must be produced is called a deadline. If a result has utility even after the deadline has passed, the deadline is classified as soft, otherwise it is firm. If a catastrophe could result if a firm deadline is missed, the deadline is called hard.

Real- Time system cont…..

Consider a railway crossing a road with a traffic signal.

- If the traffic signal does not change to "red" before the train arrives, a catastrophe could result.
- A real-time computer system that must meet at least one hard deadline is called a hard real-time Computer system or a safety-critical real-time computer system.
- If no hard real-time deadline exists, then the system is called a soft real-time computer system.

Real- Time system cont…..

- The design of a hard real-time system is fundamentally different from the design of a soft real-time system.

- While a hard real-time computer system must sustain a guaranteed temporal behavior under all specified load and fault conditions, it is permissible for a soft real-time computer system to miss a deadline occasionally.

FUNCTIONAL REQUIREMENTS

- The functional requirements of real-time systems are concerned with the functions that a real-time computer system must perform.

- They are grouped into data collection requirements, direct digital control requirements, and man-machine interaction requirements.

Data Collection

- A controlled object, e.g., a car or an industrial plant, changes its state as a function of time.
- If we freeze time, we can describe the current state of the controlled object by recording the values of its state variables at that moment.
- Possible state variables of a controlled object "car" are the position of the car, the speed of the car, the position of switches on the dash board, and the position of a piston in a cylinder.
- We are normally not interested in all state variables, but only in the subset of state variables that is significant for our purpose.
- A significant state variable is called a real-time (RT) entity.

Data Collection cont..

- Every RT entity is in the sphere of control (SOC) of a subsystem, i.e., it belongs to a subsystem that has the authority to change the value of this RT entity.

- Outside its sphere of control, the value of an RT entity can be observed, but cannot be modified.

- **For example,** the current position of a piston in a cylinder of the engine of a controlled car object is in the sphere of control of the car. Outside the car, the current position of the piston can only be observed.

Data Collection cont….

- The first functional requirement of a real-time computer system is the observation of the RT entities in a controlled object and the collection of these observations.

- An observation of an RT entity is represented by a real-time (RT) image in the computer system.

- Since the state of the controlled object is a function of real time, a given RT image is only temporally accurate for a limited time interval.

- The length of this time interval depends on the dynamics of the controlled object.

- If the state of the controlled object changes very quickly, the corresponding RT image has a very short accuracy interval.

Temporal accuracy of the traffic light

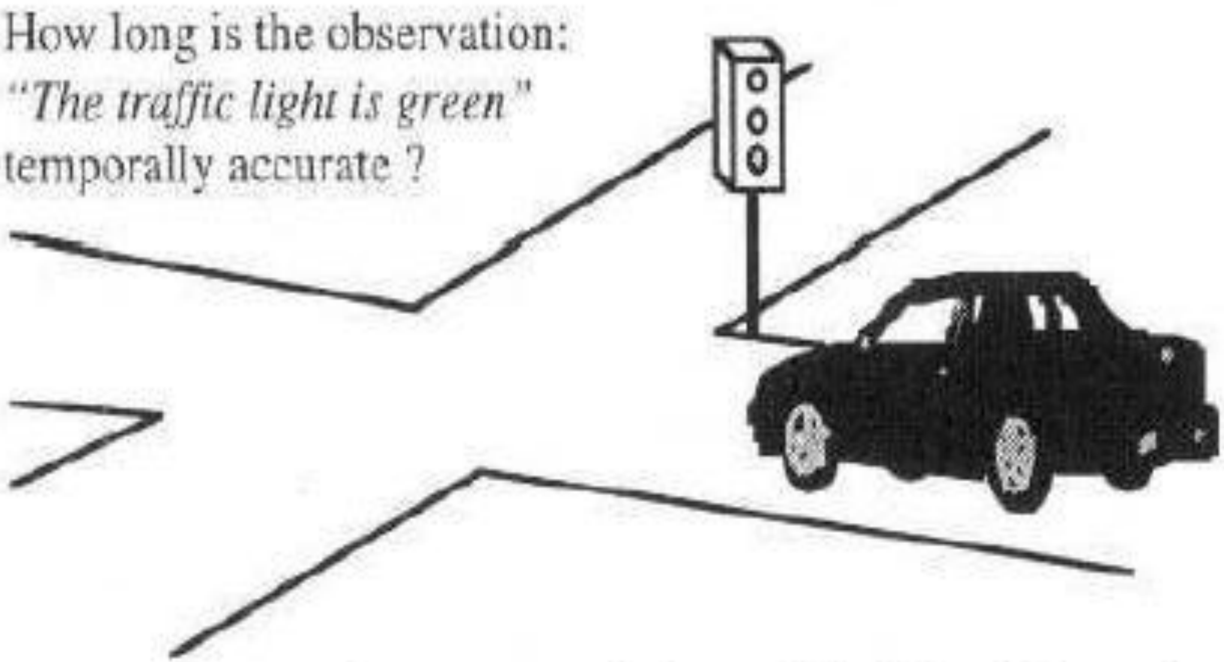


Figure 1.2: Temporal accuracy of the traffic light information.

Example: Consider the example of Figure 1.2, where a car enters an intersection controlled by a traffic light. How long is the observation "the traffic light is green" temporally accurate? If the information "the traffic light is green" is used outside its accuracy interval, i.e., a car enters the intersection after the traffic light has switched to red, a catastrophe may occur. In this example, an upper bound for the accuracy interval is given by the duration of the yellow phase of the traffic light.

Data Collection

- The set of all temporally accurate real-time images of the controlled object is called the real-time database.

- The real-time database must be updated whenever an RT entity changes its value.

- These updates can be performed periodically, triggered by the progression of the real-time clock by a fixed period (time-triggered (TT) observation), or immediately after a change of state, which constitutes an event, occurs in the RT entity (event-triggered (ET) observation).

Signal Conditioning

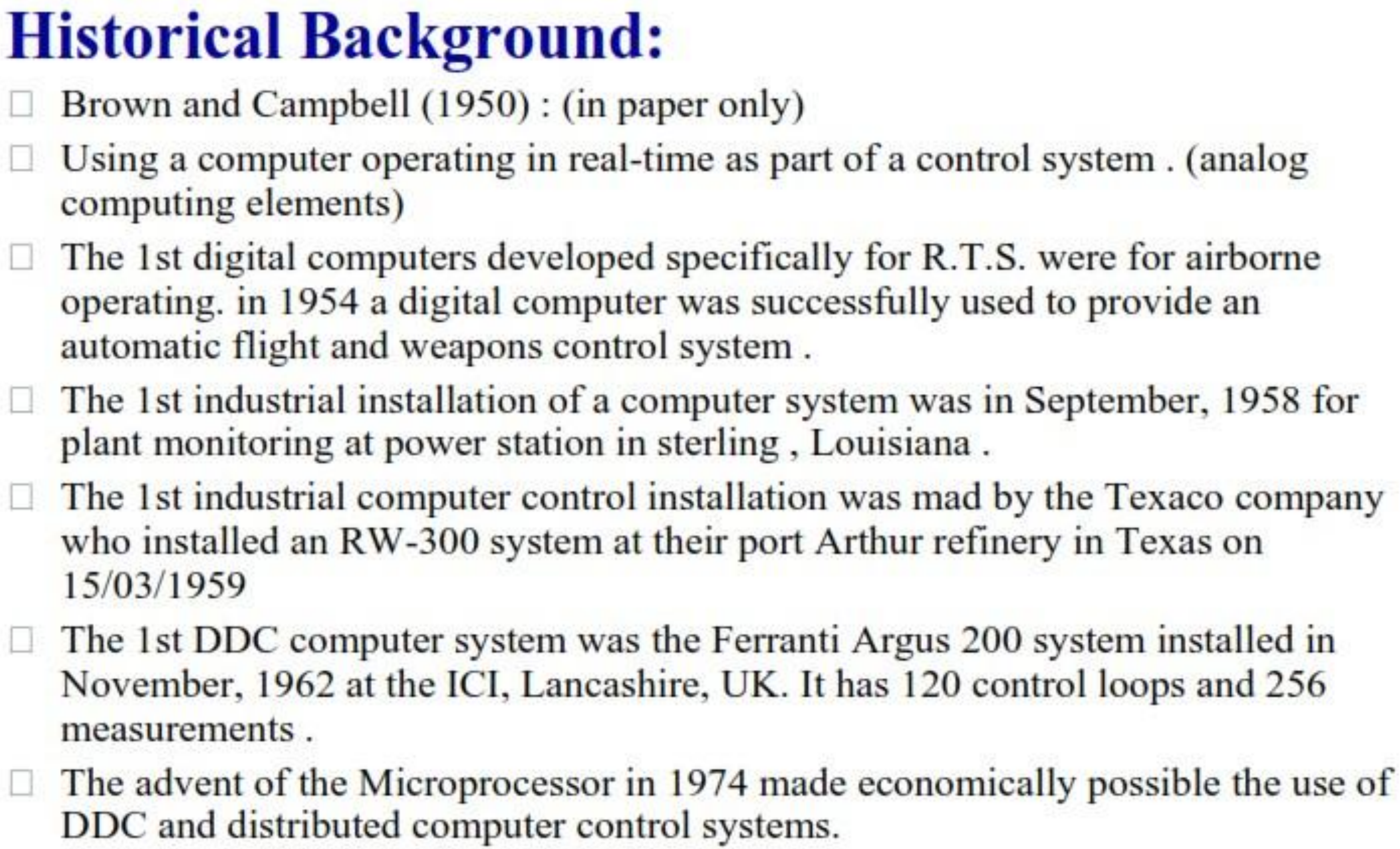
- A physical sensor, like a thermocouple, produces a raw data element (e.g., a voltage). Often, a sequence of raw data elements is collected and an averaging algorithm is applied to reduce the measurement error.

- In the next step the raw data must be calibrated and transformed to standard measurement units. The term signal conditioning is used to refer to all the processing steps that are necessary to obtain meaningful measured data of an RT entity from the raw sensor data.

- After signal conditioning, the measured data must be checked for plausibility and related to other measured data to detect a possible fault of the sensor.

- A data element that is judged to be a correct RT image of the corresponding RT entity is called an agreed data element.

Historical background



Historical Background:

□ Brown and Campbell (1950) : (in paper only)

□ Using a computer operating in real-time as part of a control system . (analog computing elements)

□ The 1st digital computers developed specifically for R.T.S. were for airborne operating. in 1954 a digital computer was successfully used to provide an automatic flight and weapons control system .

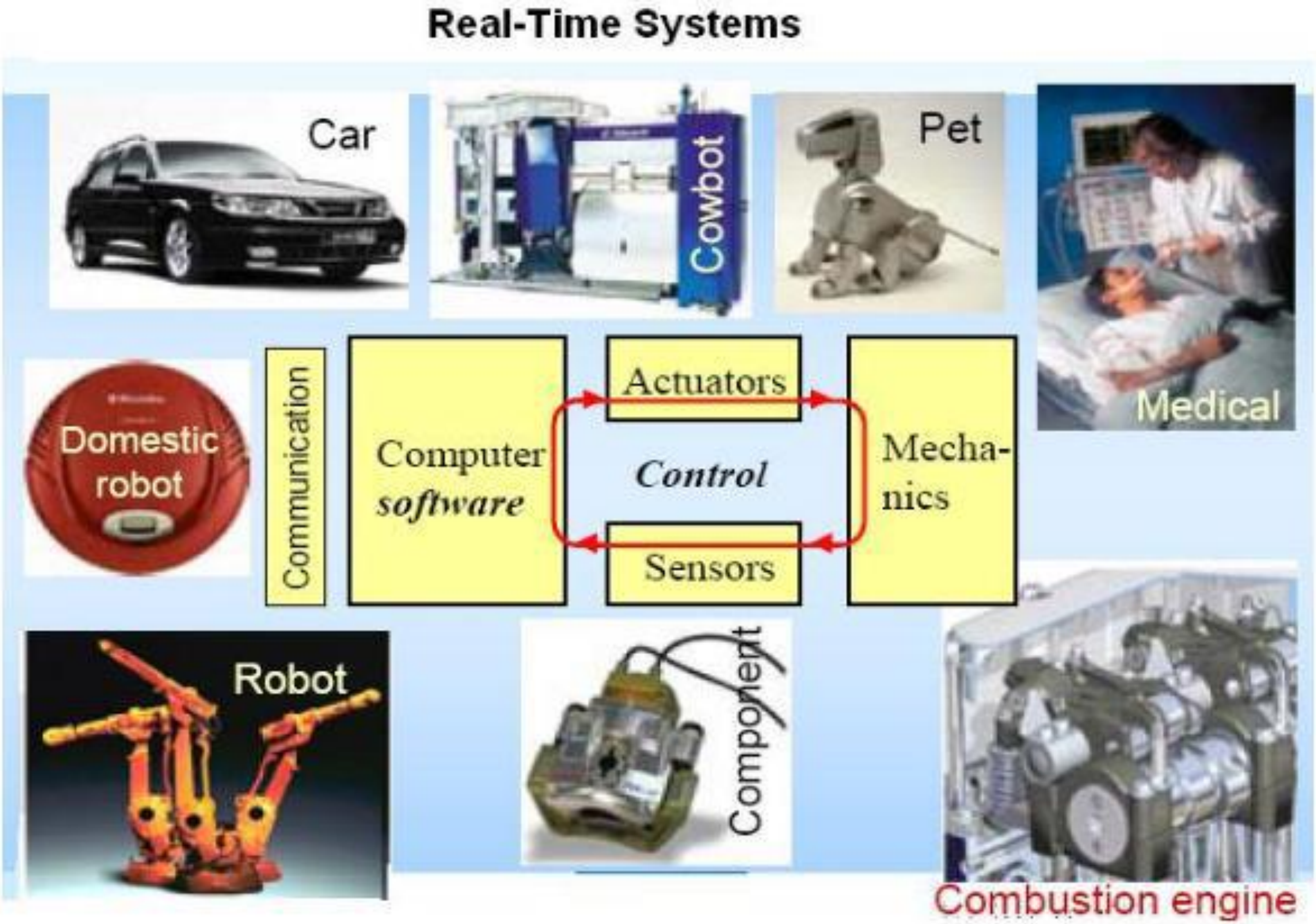
□ The 1st industrial installation of a computer system was in September, 1958 for plant monitoring at power station in sterling , Louisiana .

□ The 1st industrial computer control installation was mad by the Texaco company who installed an RW-300 system at their port Arthur refinery in Texas on 15/03/1959

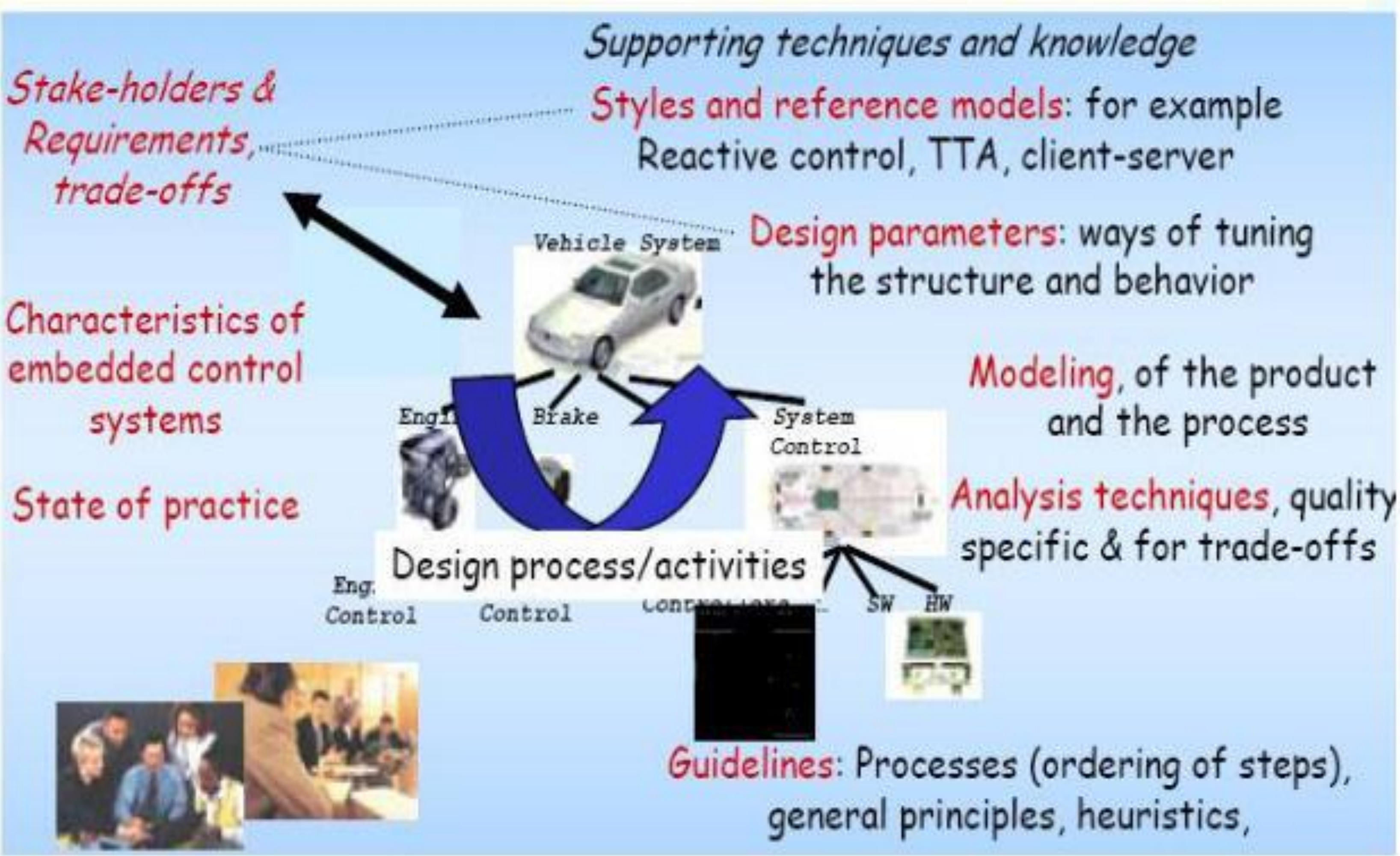
□ The 1st DDC computer system was the Ferranti Argus 200 system installed in November, 1962 at the ICI, Lancashire, UK. It has 120 control loops and 256 measurements .

□ The advent of the Microprocessor in 1974 made economically possible the use of DDC and distributed computer control systems.

Real time Systems



Real time Systems



Supporting techniques and knowledge

Stake-holders & Requirements, trade-offs

Characteristics of embedded control systems

State of practice

Styles and reference models: for example Reactive control, TTA, client-server

Design parameters: ways of tuning the structure and behavior

Modeling, of the product and the process

Analysis techniques, quality specific & for trade-offs

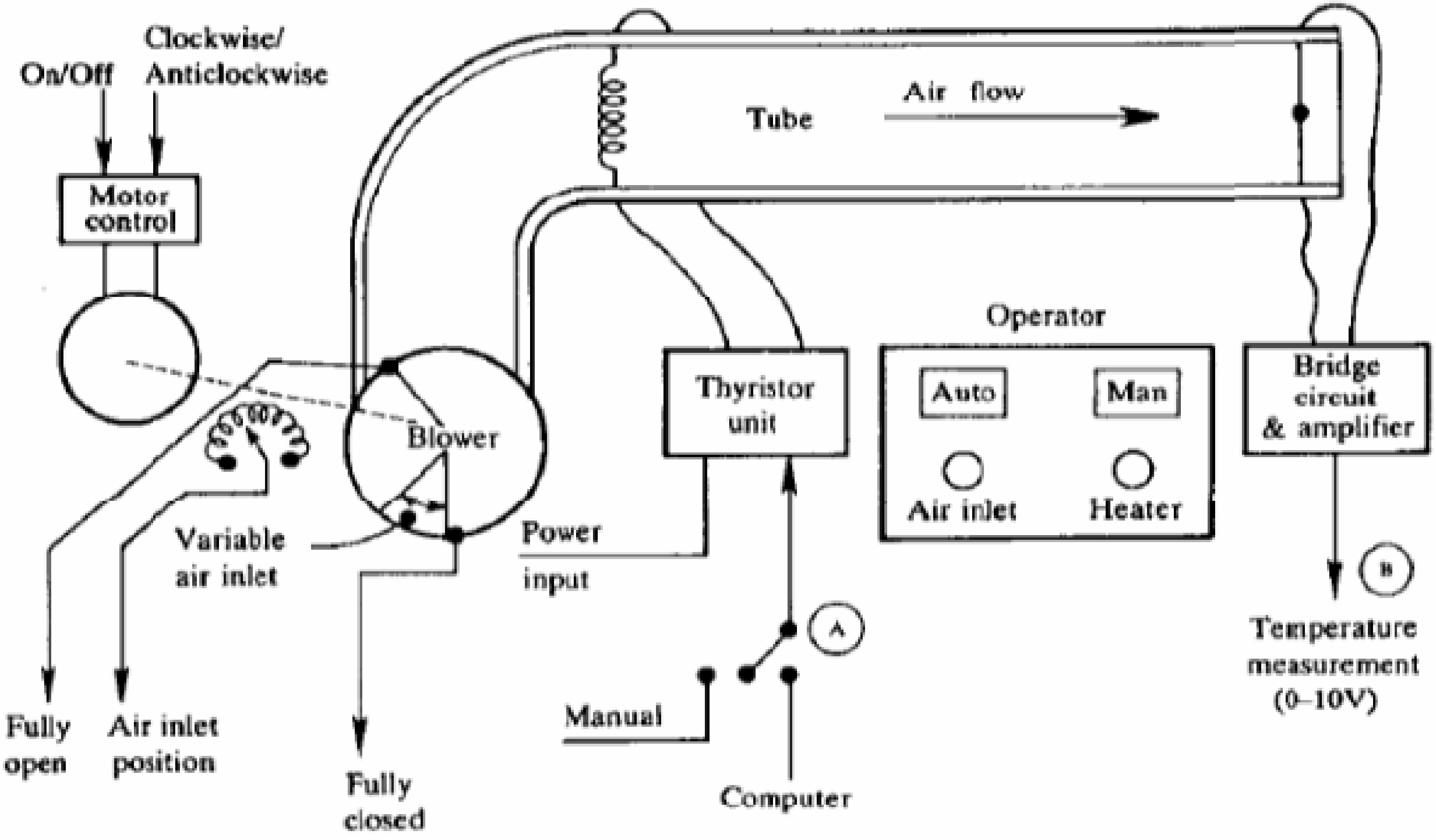
Design process/activities

Guidelines: Processes (ordering of steps), general principles, heuristics,

Elements of Real time systems

Elements of a Real-Time System:

A simple plant – a hot-air blower.

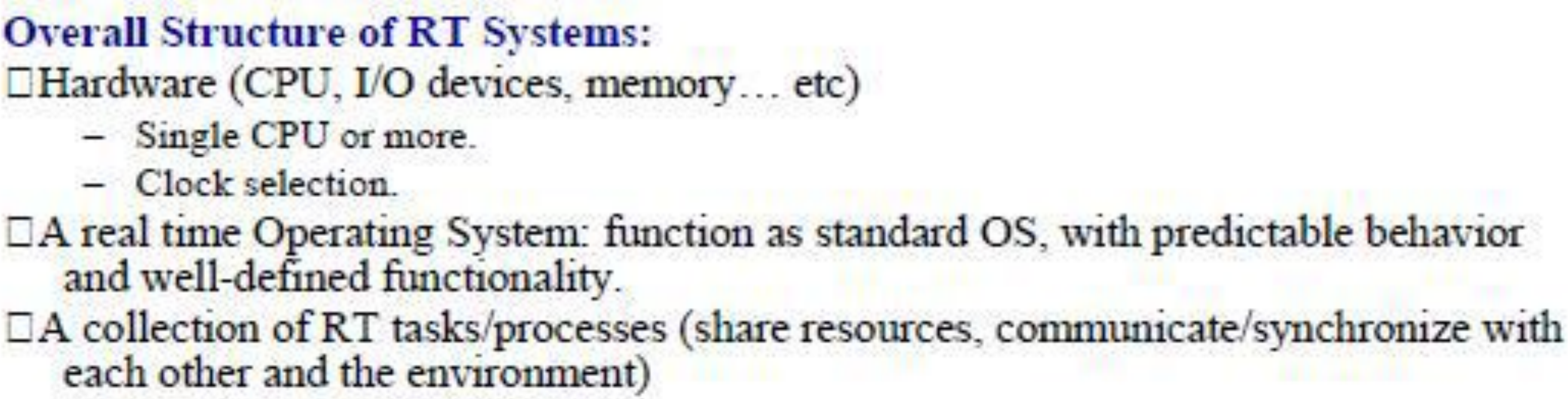


Elements of a computer control system

Elements of a Computer Control System:

- □ A centrifugal fan blows air over a heating element and into a tube.
- □ A thermostat is used to detect the temp.
- □ The position of the air-inlet cover to the fan is adjusted by a reversible DC motor (constant speed).
- □ A potentiometer is attached to the air-inlet cover .
- □ Two 8-switches are used to detect when the cover is fully open or fully closed .
- □ The operator is provided with a panel from which the control system can be switched from automatic to manual control panel. Lights indicate: Fan ON, Heater ON, Cover fully-open, Cover fully-closed, Auto/Manual status.
- □ The information is available from the plant instrument in the following two forms:
  - □ Analog signals : Air Temp., Fan-inlet cover position .
  - □ Digital signals : Fan-inlet cover position (Fully-open, Fully-closed)
  - □ Status signals : Auto/Manual , Fan motor ON, Heater ON

Overall structure of RT Systems



Overall Structure of RT Systems:

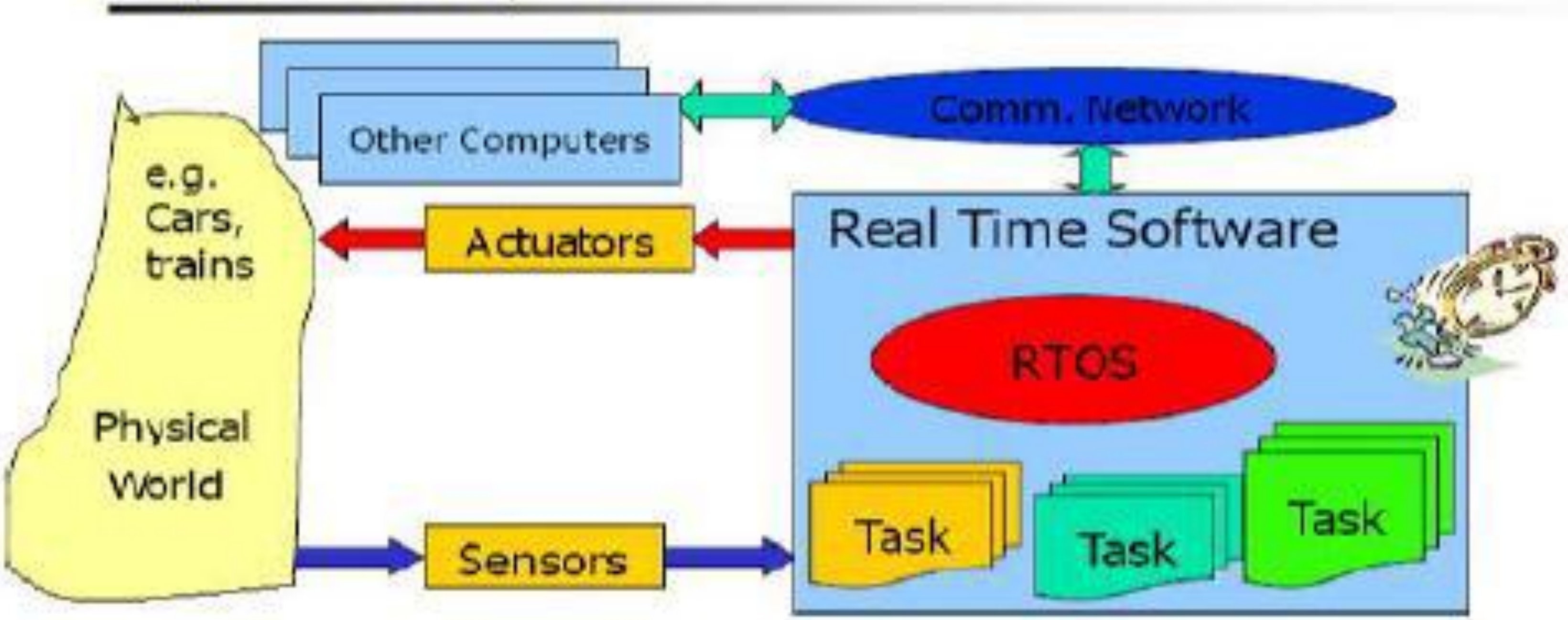
□Hardware (CPU, I/O devices, memory… etc)

- Single CPU or more.
- Clock selection.

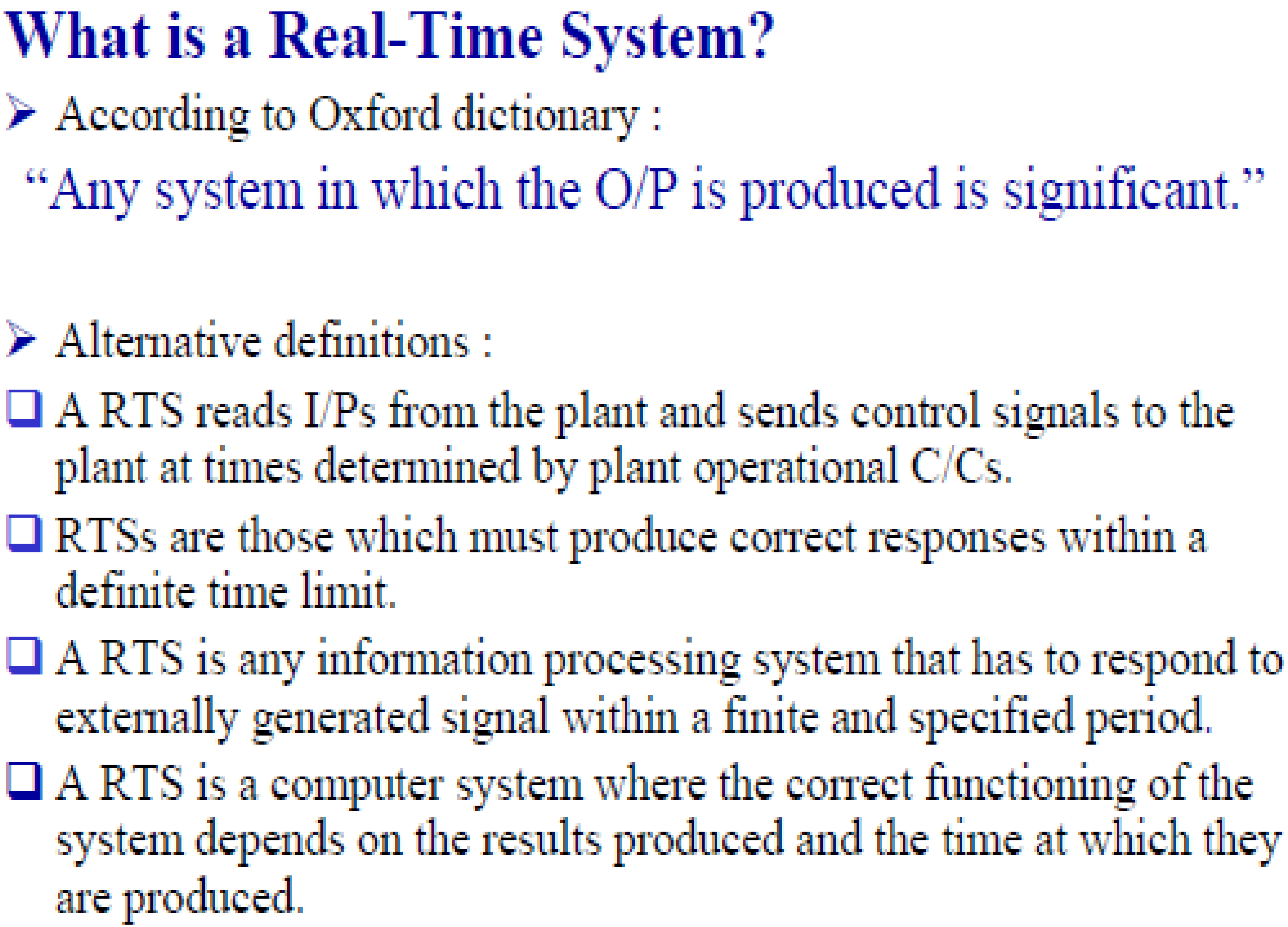
□A real time Operating System: function as standard OS, with predictable behavior and well-defined functionality.

□A collection of RT tasks/processes (share resources, communicate/synchronize with each other and the environment)

Components of RT Systems



What is a Real Time system?



What is a Real-Time System?

➢ According to Oxford dictionary :

"Any system in which the O/P is produced is significant."

➢ Alternative definitions :

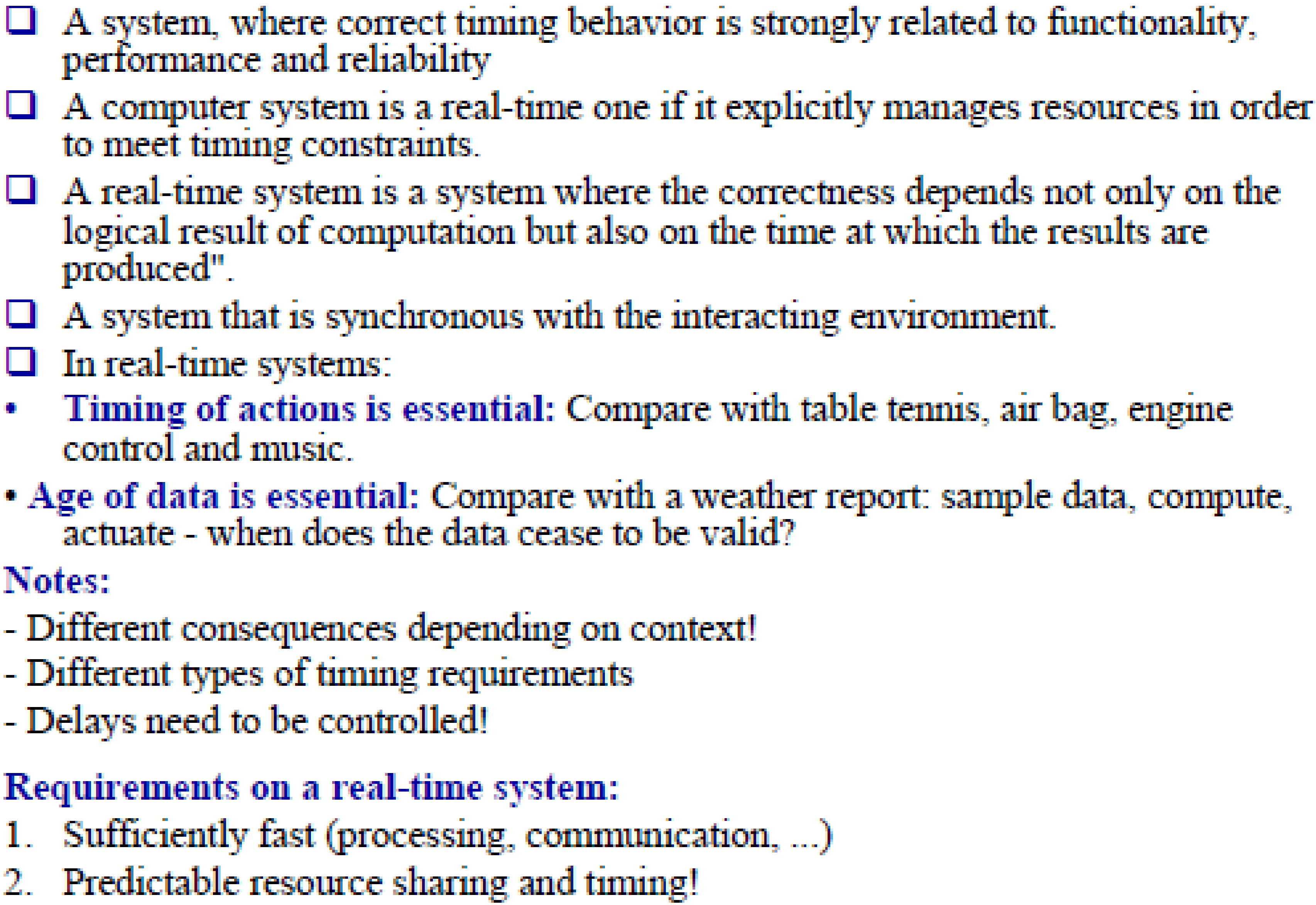
❑ A RTS reads I/Ps from the plant and sends control signals to the plant at times determined by plant operational C/Cs.

❑ RTSs are those which must produce correct responses within a definite time limit.

❑ A RTS is any information processing system that has to respond to externally generated signal within a finite and specified period.

❑ A RTS is a computer system where the correct functioning of the system depends on the results produced and the time at which they are produced.

What is a Real Time system? Cont…..



- A system, where correct timing behavior is strongly related to functionality, performance and reliability
- A computer system is a real-time one if it explicitly manages resources in order to meet timing constraints.
- A real-time system is a system where the correctness depends not only on the logical result of computation but also on the time at which the results are produced".
- A system that is synchronous with the interacting environment.
- In real-time systems:

• **Timing of actions is essential:** Compare with table tennis, air bag, engine control and music.

• **Age of data is essential:** Compare with a weather report: sample data, compute, actuate - when does the data cease to be valid?

Notes:

- Different consequences depending on context!
- Different types of timing requirements
- Delays need to be controlled!

Requirements on a real-time system:

1. Sufficiently fast (processing, communication, ...)
2. Predictable resource sharing and timing!

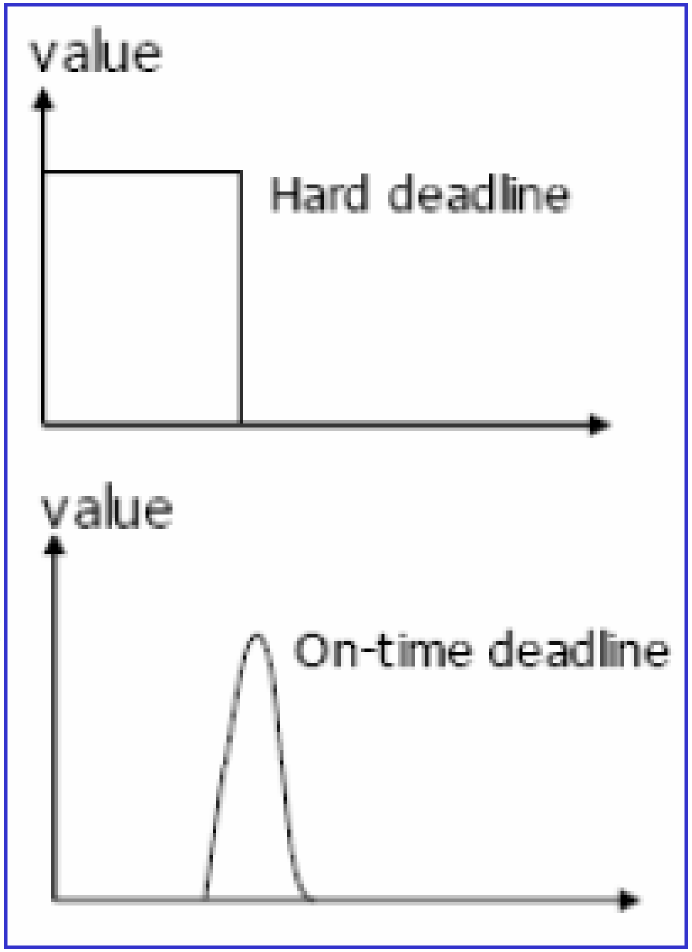
Classification of RTSs

Classification of RTSs:

Real-Time systems can be classified as:

1. HARD REAL-TIME SYSTEM:

- A system whose operation is degraded if results are not proceeded according to specified timing requirements.
- System response occur within a specified deadline. Failure to meet such a timing requirement can have catastrophic consequences.
- Systems where it is absolutely imperative that responses occur within the required deadline. Example: Flight control systems, automotive systems, robotics etc.

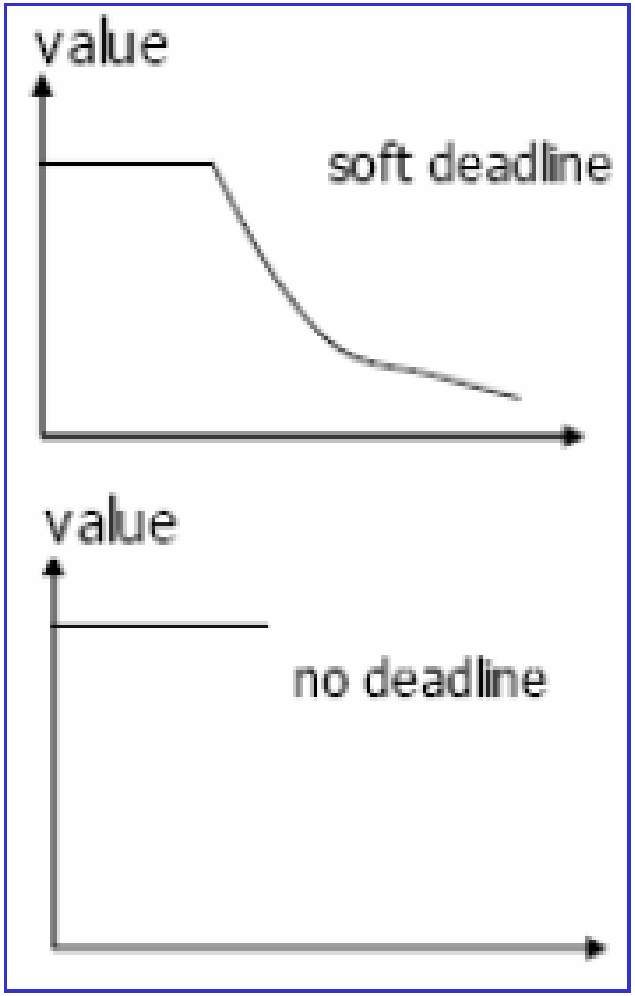


Classification of RTSs cont….

Classification of RTSs: (Cont)

2. SOFT REAL-TIME SYSTEMS:

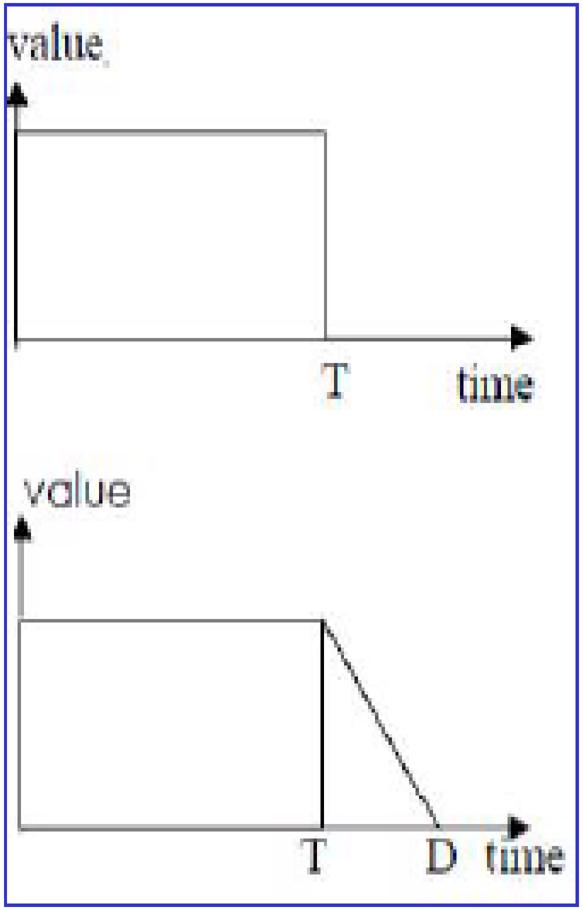
- A system whose operation is incorrect if results are not produced according to the timing constraints. Catastrophic results will happen then.
- The response times are important but not critical to the operation of the system . Failure to meet the timing requirements would not impair the system.
- Systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. Example: Banking system, multimedia etc.



Classification of RTSs cont……

Classification of RTSs: (cont.)

3. **FIRM REAL-TIME SYSTEMS:** There is no value for a response that occurs past a specific deadline. Failure to meet the timing requirements is undesirable .



Notes:

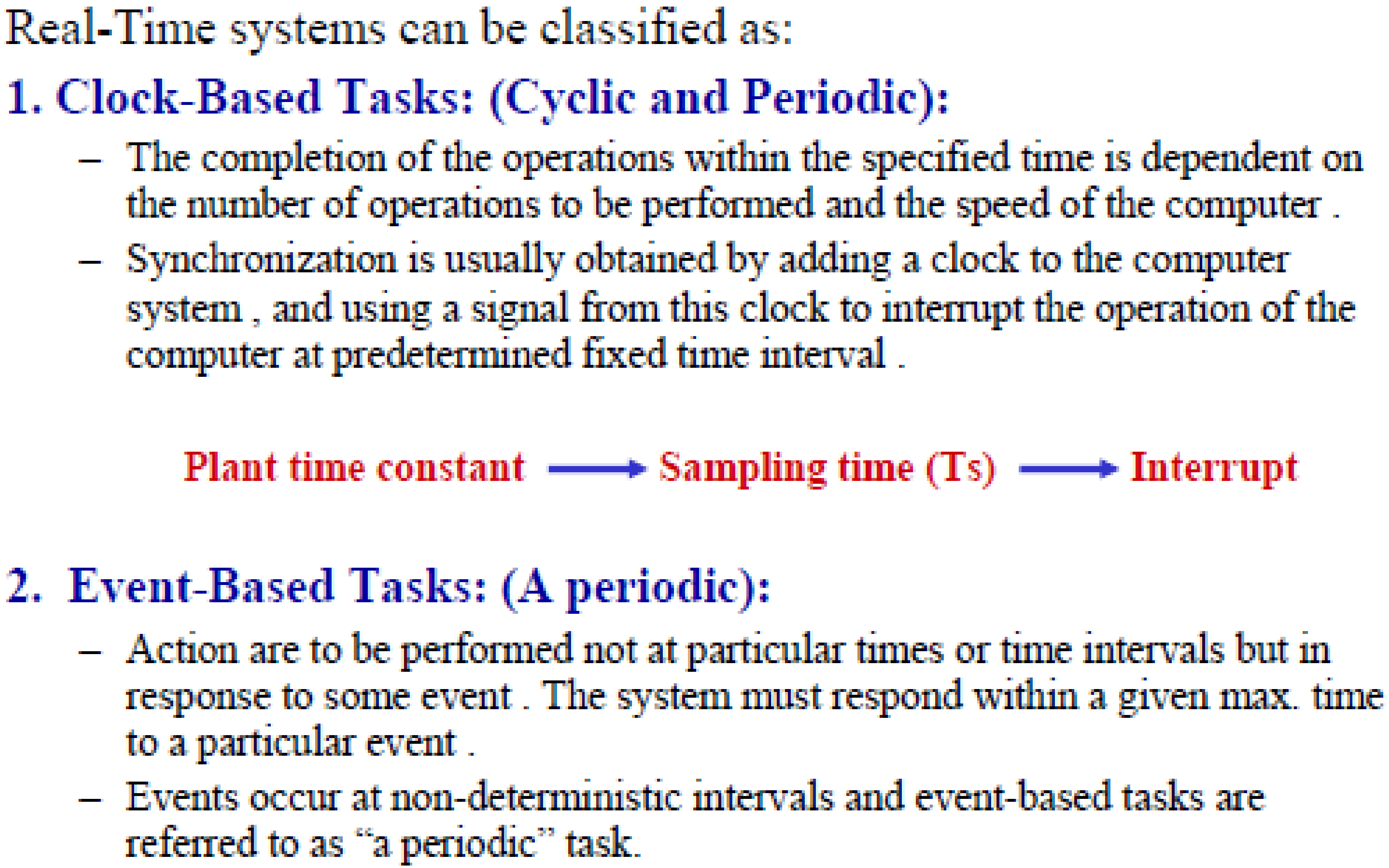
- ➢ A single system may have both hard and soft real-time Subsystems.
- ➢ In reality many systems will have a cost function associated with missing each deadline.

Classification of RTSs cont……

Clock-Based & Event-Based Systems:

- Synchronization between the external processes and internal actions (tasks) carried out by the computer may be defined in terms of the passage of time, or the actual time of day, in which case the system is said to be **“Clock-based system”** or it may be defined in terms of events, and the system is said to be **“Event-based system”**.
- If the relationship between the actions in the computer and the system is much more loosely defined, then the system is said to be **“interactive system”.**

Classification of Real time systems



Real-Time systems can be classified as:

1. Clock-Based Tasks: (Cyclic and Periodic):

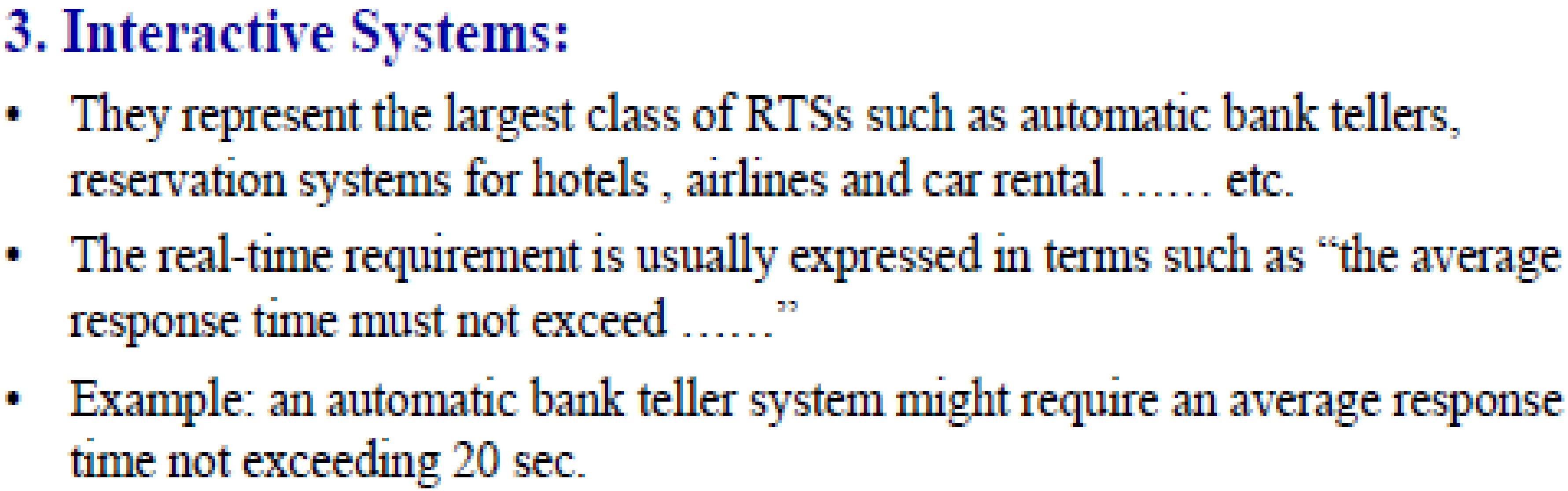
- The completion of the operations within the specified time is dependent on the number of operations to be performed and the speed of the computer .
- Synchronization is usually obtained by adding a clock to the computer system , and using a signal from this clock to interrupt the operation of the computer at predetermined fixed time interval .

Plant time constant → Sampling time (Ts) → Interrupt

2. Event-Based Tasks: (A periodic):

- Action are to be performed not at particular times or time intervals but in response to some event . The system must respond within a given max. time to a particular event .
- Events occur at non-deterministic intervals and event-based tasks are referred to as "a periodic" task.

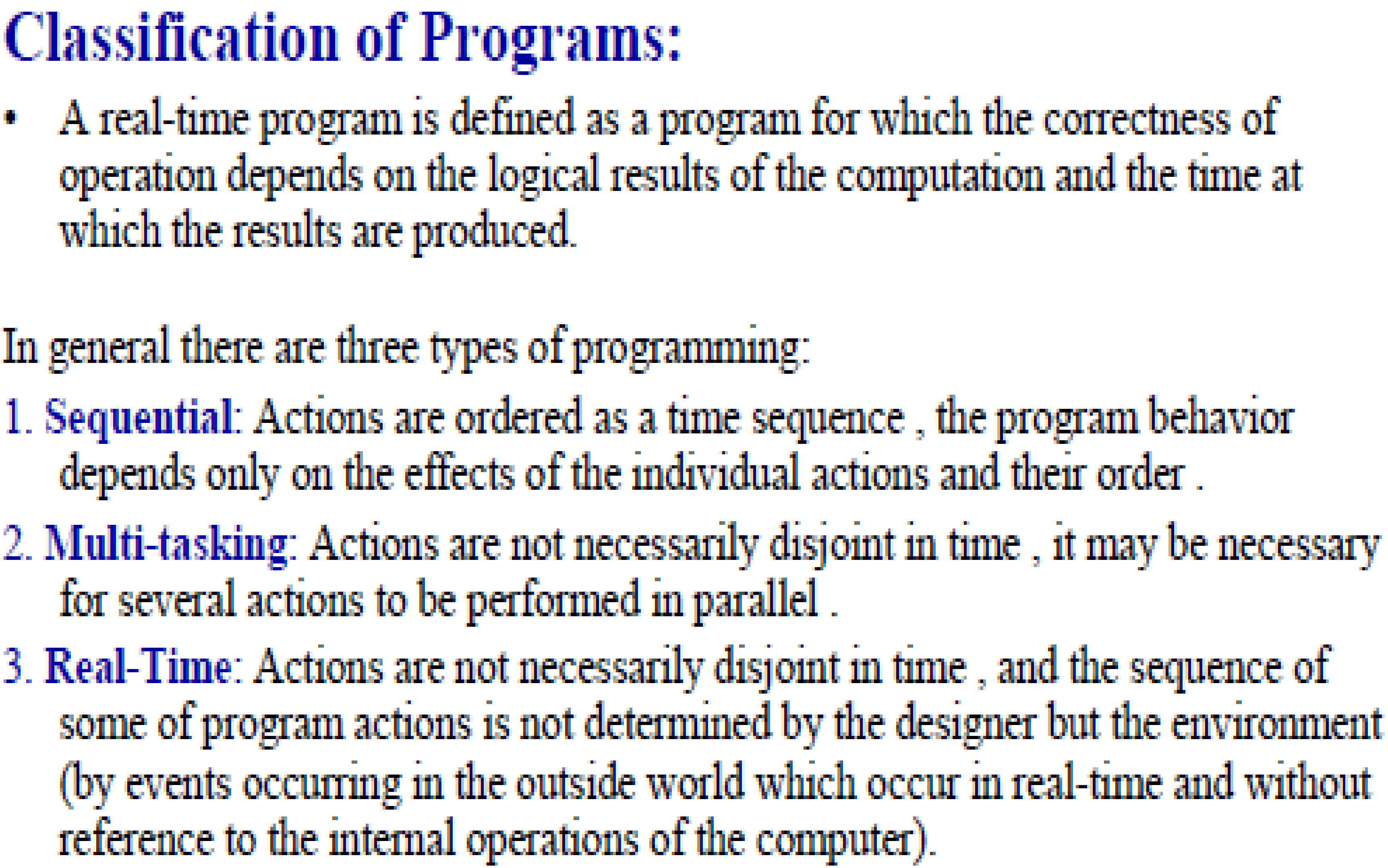
Classification of Real time systems cont…..



3. Interactive Systems:

- They represent the largest class of RTSs such as automatic bank tellers, reservation systems for hotels , airlines and car rental …… etc.
- The real-time requirement is usually expressed in terms such as "the average response time must not exceed ……"
- Example: an automatic bank teller system might require an average response time not exceeding 20 sec.

Classification of programs



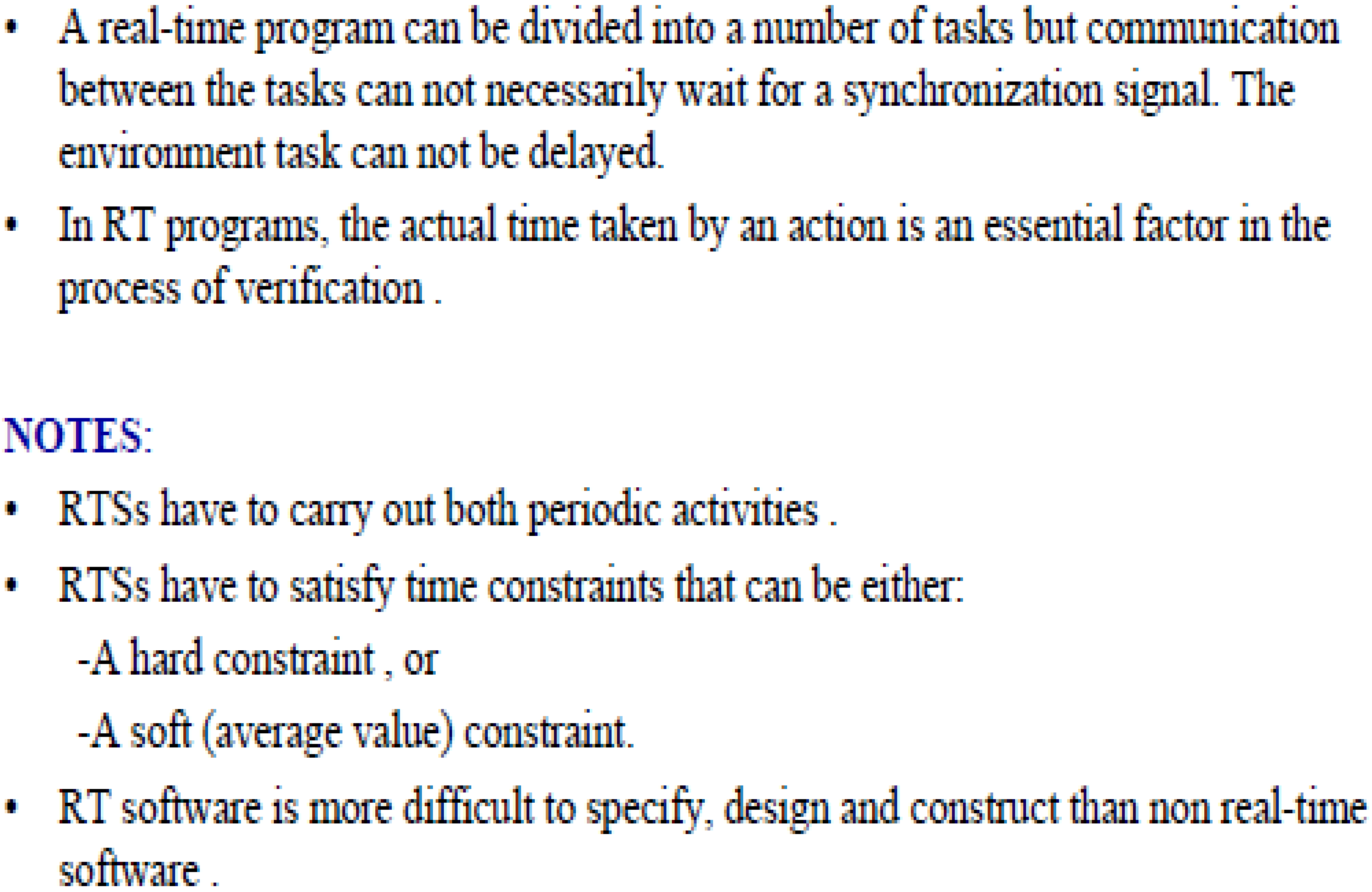
Classification of Programs:

- A real-time program is defined as a program for which the correctness of operation depends on the logical results of the computation and the time at which the results are produced.

In general there are three types of programming:

1. **Sequential**: Actions are ordered as a time sequence , the program behavior depends only on the effects of the individual actions and their order .
2. **Multi-tasking**: Actions are not necessarily disjoint in time , it may be necessary for several actions to be performed in parallel .
3. **Real-Time**: Actions are not necessarily disjoint in time , and the sequence of some of program actions is not determined by the designer but the environment (by events occurring in the outside world which occur in real-time and without reference to the internal operations of the computer).

Real Time programs



- A real-time program can be divided into a number of tasks but communication between the tasks can not necessarily wait for a synchronization signal. The environment task can not be delayed.
- In RT programs, the actual time taken by an action is an essential factor in the process of verification .

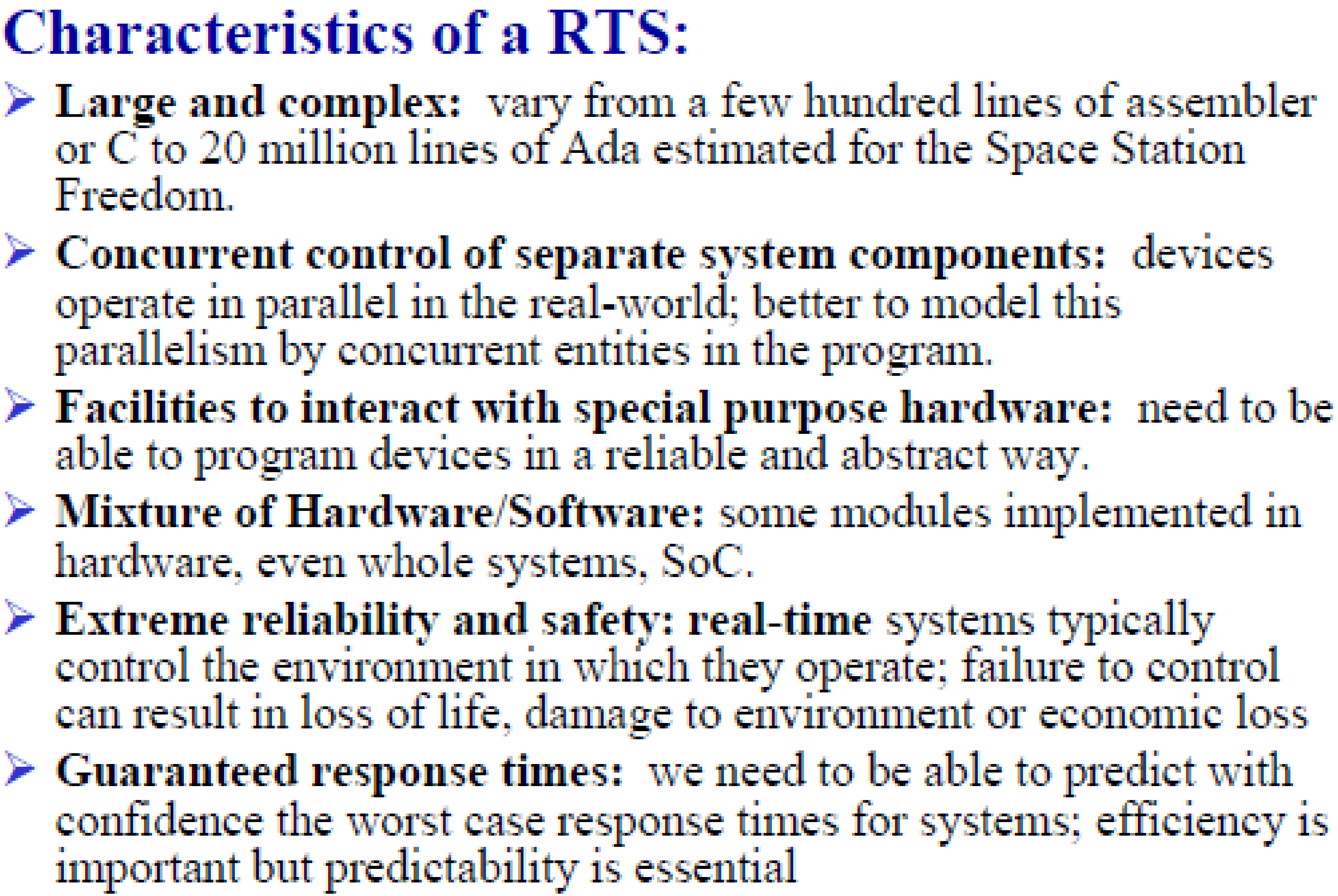
NOTES:

- RTSs have to carry out both periodic activities .
- RTSs have to satisfy time constraints that can be either:

  -A hard constraint , or

  -A soft (average value) constraint.
- RT software is more difficult to specify, design and construct than non real-time software .

Characteristics of a RTS



Characteristics of a RTS:

- **Large and complex:** vary from a few hundred lines of assembler or C to 20 million lines of Ada estimated for the Space Station Freedom.
- **Concurrent control of separate system components:** devices operate in parallel in the real-world; better to model this parallelism by concurrent entities in the program.
- **Facilities to interact with special purpose hardware:** need to be able to program devices in a reliable and abstract way.
- **Mixture of Hardware/Software:** some modules implemented in hardware, even whole systems, SoC.
- **Extreme reliability and safety: real-time** systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss
- **Guaranteed response times:** we need to be able to predict with confidence the worst case response times for systems; efficiency is important but predictability is essential

Global Time

1. The notions of causal order, temporal order, and delivery order and their interrelationships are elaborated.

2. The parameters that characterize the behavior and the quality of a digital clock are investigated. The positivist tradition by introducing an omniscient external observer with an absolute reference clock that can generate precise timestamps for all relevant events.

3. These absolute timestamps are used to reason about the precision and accuracy of a global time base, and to expose the fundamental limits of time measurement in a distributed real-time system.

4. The idea of a sparse time base is introduced to establish a consistent view of the order of computer-generated events in a distributed real-time system without having to execute an agreement protocol.

TIME AND ORDER

- The notion of time is fundamental to our existence. We can reflect on past events and on possible future events, and thus reason about events in the domain of time.

- In many models of natural phenomena (e.g., Newtonian mechanics), time is an independent variable that determines the sequence of states of a system.

- The basic constants of physics are defined in relation to the standard of time, the physical second. This is why the global time base in a distributed real-time system should be based on the metric of the physical second.

TIME AND ORDER cont….

- In a typical real-time application, the distributed computer system performs a multitude of different functions concurrently,

- e.g., the monitoring of real-time (RT) entities (both their value and rate of change), the detection of alarm conditions,

- Display of the observations to the operator, and the execution of control algorithms to find new set points.

- These different functions are normally executed at different nodes.

Different Orders

- **Temporal Order**
- **Causal Order**
- **Delivery Order**
- **Clocks:** Physical Clock, Reference Clock, Clock Drift, Failure Modes of a Clock
- **Precision and Accuracy:** Time Standards- International Atomic Time (TAI–Temps Atomique Internationale), Universal Time Coordinated (UTC)
- **Time Measurement:** Global Time, Reasonableness Condition: Interval Measurement, / -Precendence

INTERNAL CLOCK SYNCHRONIZATION

Internal and external clock synchronization

- The purpose of internal clock synchronization is to ensure that the global ticks of all correct nodes occur within the specified precision II , despite the varying drift rate of the local real-time clock of each node.

- Because the availability of a proper global time base is crucial for the operation of a distributed real-time system, the clock synchronization should be fault-tolerant.

Synchronization condition

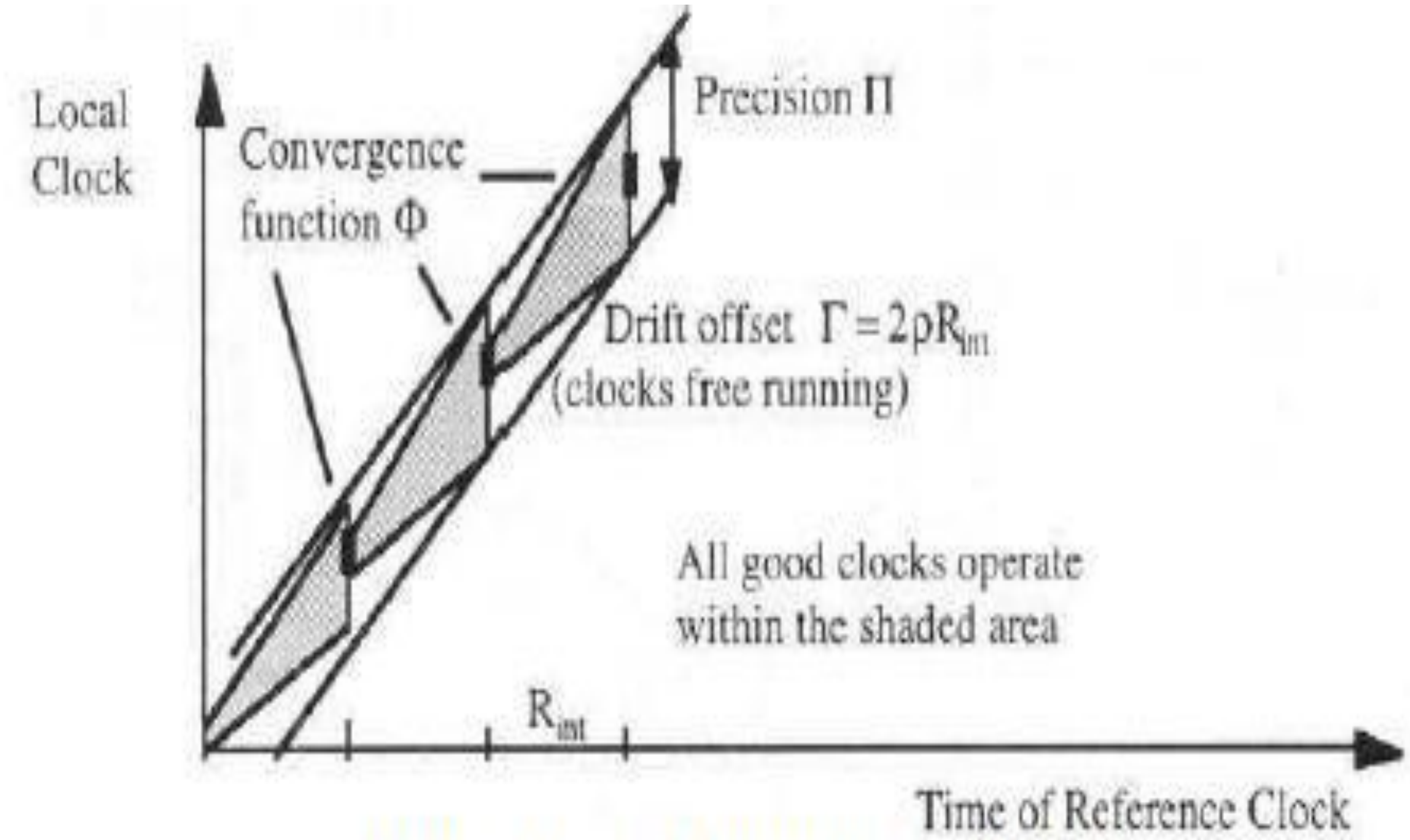

Local Clock
Convergence function Φ
Precision Π
Drift offset $\Gamma = 2\rho R_{int}$
(clocks free running)
All good clocks operate within the shaded area
R_{int}
Time of Reference Clock

Figure 3.9: Synchronization condition.

The Synchronization Condition

1. Every node of a distributed system has a local oscillator that (micro)ticks with a frequency determined by the physical parameters of the oscillator.

2. A subset of the local oscillator's microticks called the ticks (or macroticks), are interpreted as the global time ticks at the node. The global time ticks increment the node's local global time counter.

3. The global time ticks of each node must be periodically resynchronized within the ensemble of nodes to establish a global time base with specified precision.

4. The period of resynchronization is called the resynchronization interval.

The Synchronization Condition

1. At the end of each resynchronization interval, the clocks are adjusted to bring them into better agreement with each other.

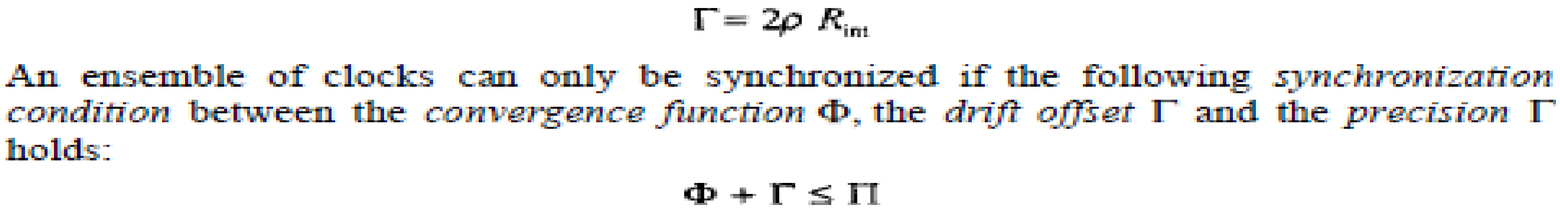
2. The convergence function denotes the offset of the time values immediately after the resynchronization.

3. Then, the clocks again drift apart until they are resynchronized at the end of the next resynchronization interval Rint.

4. The drift offset indicates the maximum divergence of any two good clocks from each other during the resynchronization interval Rint, where the clocks are free running.

5. The drift offset depends on the length of the resynchronization interval Rint and the maximum specified drift rate of the clock:

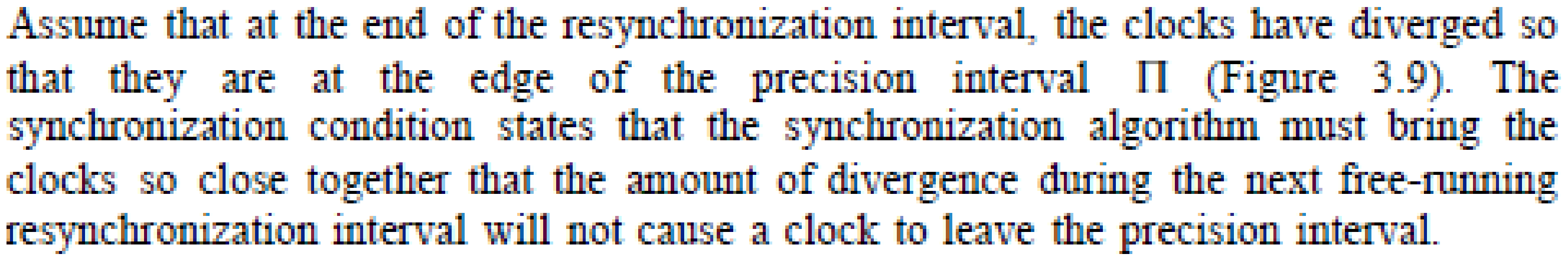
The Synchronization Condition



$$\Gamma = 2\rho\ R_{int}$$

An ensemble of clocks can only be synchronized if the following *synchronization condition* between the *convergence function* Φ, the *drift offset* Γ and the *precision* Γ holds:

$$\Phi + \Gamma \leq \Pi$$

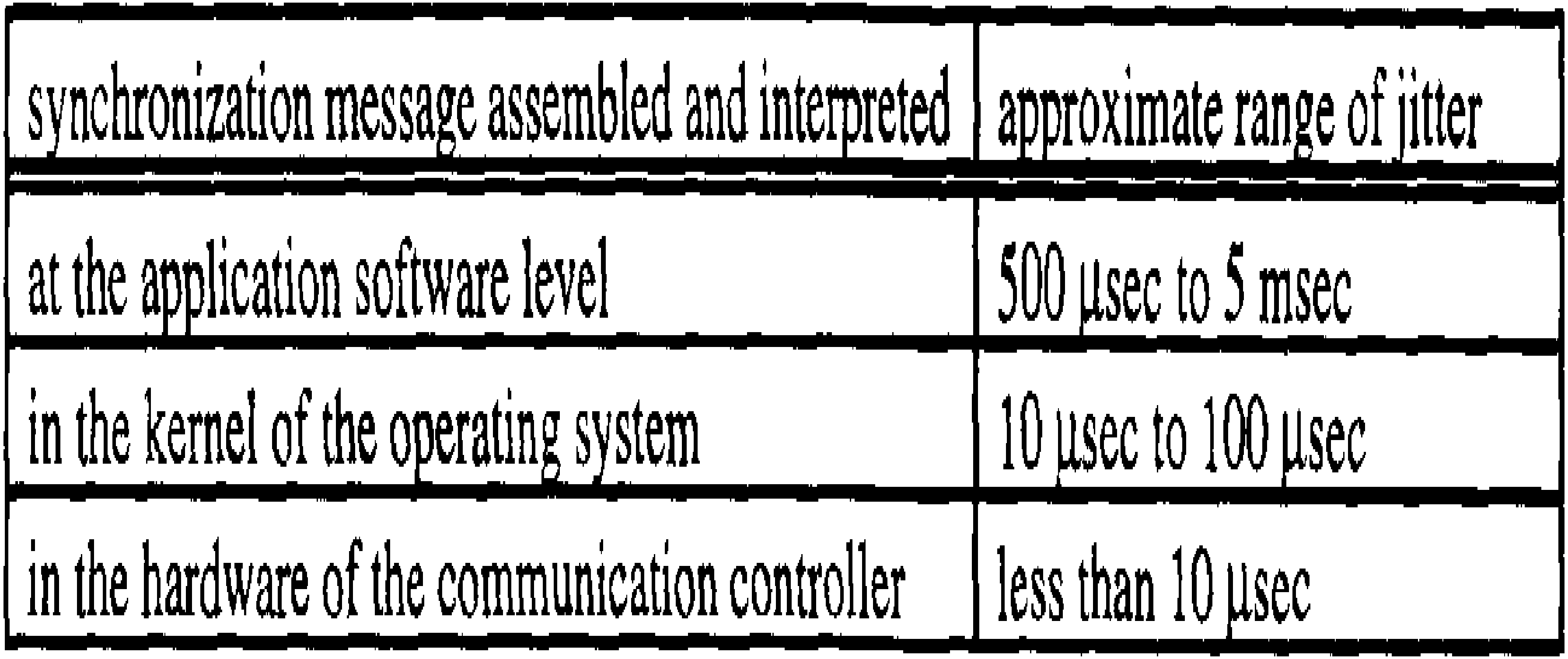


Assume that at the end of the resynchronization interval, the clocks have diverged so that they are at the edge of the precision interval Π (Figure 3.9). The synchronization condition states that the synchronization algorithm must bring the clocks so close together that the amount of divergence during the next free-running resynchronization interval will not cause a clock to leave the precision interval.

Observations and examples

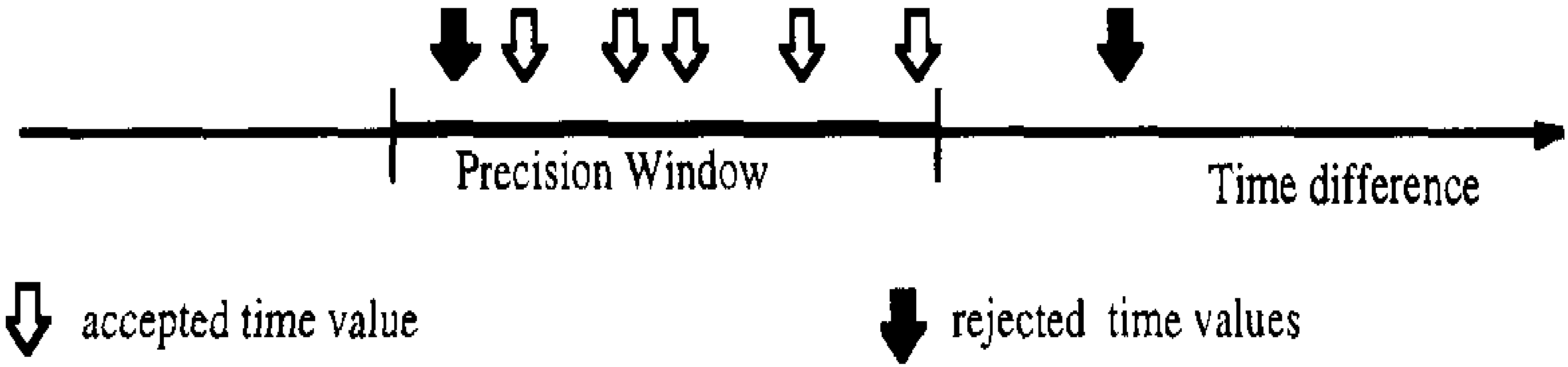
- Byzantine Error: Behavior of a malicious clock.
- Central Master Synchronization
- Distributed Synchronization Algorithms: Reading the Global Time Result
- The Convergence Function

Table 3.2: Approximate jitter of the synchronization messages



synchronization message assembled and interpreted	approximate range of jitter
at the application software level	500 μsec to 5 msec
in the kernel of the operating system	10 μsec to 100 μsec
in the hardware of the communication controller	less than 10 μsec

Figure : Accepted and rejected time values.



Example: Figure shows an ensemble of 7 nodes and one tolerated Byzantine fault. The FTA takes the average of the five accepted time values shown.

Figure : Worst possible behavior of a malicious

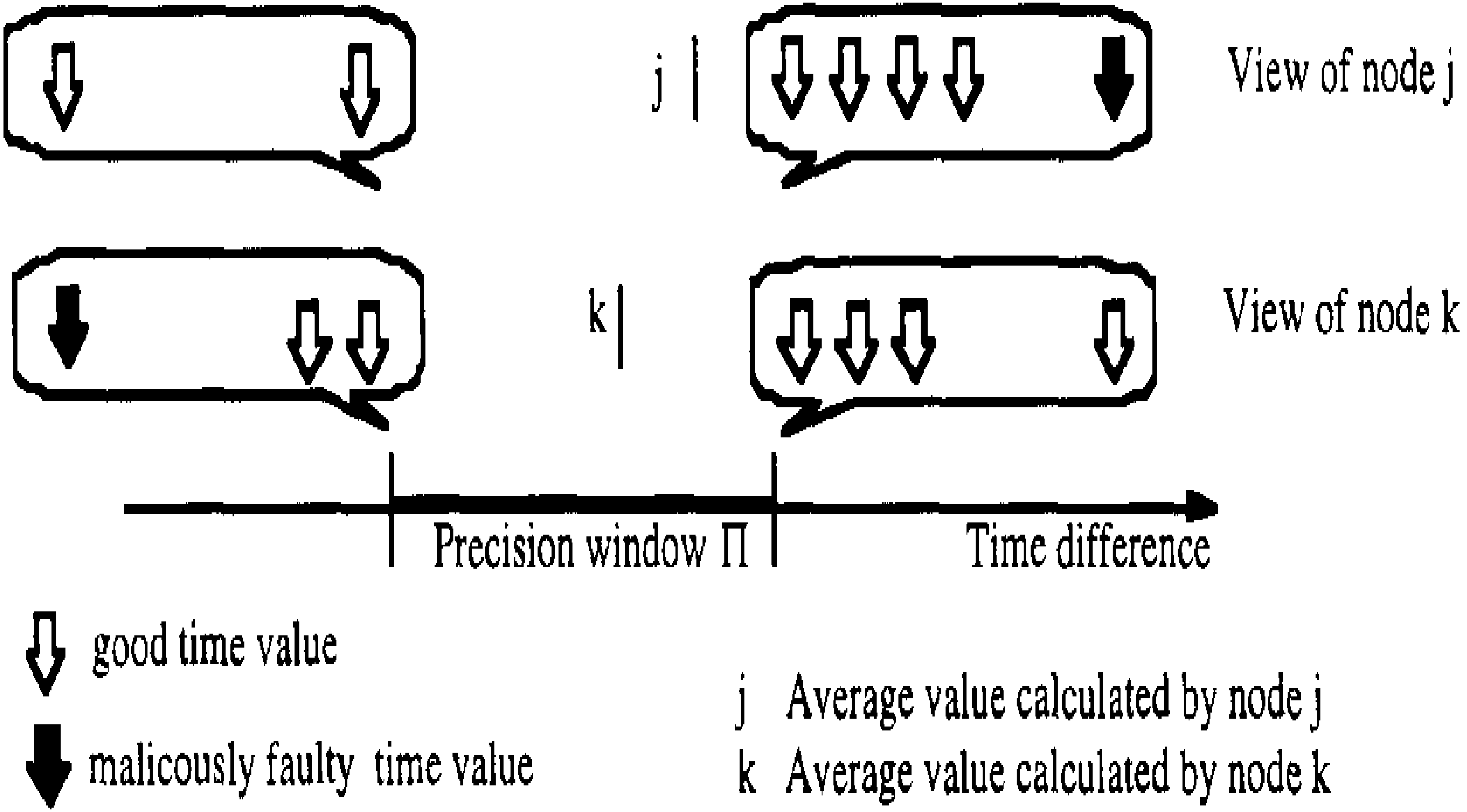
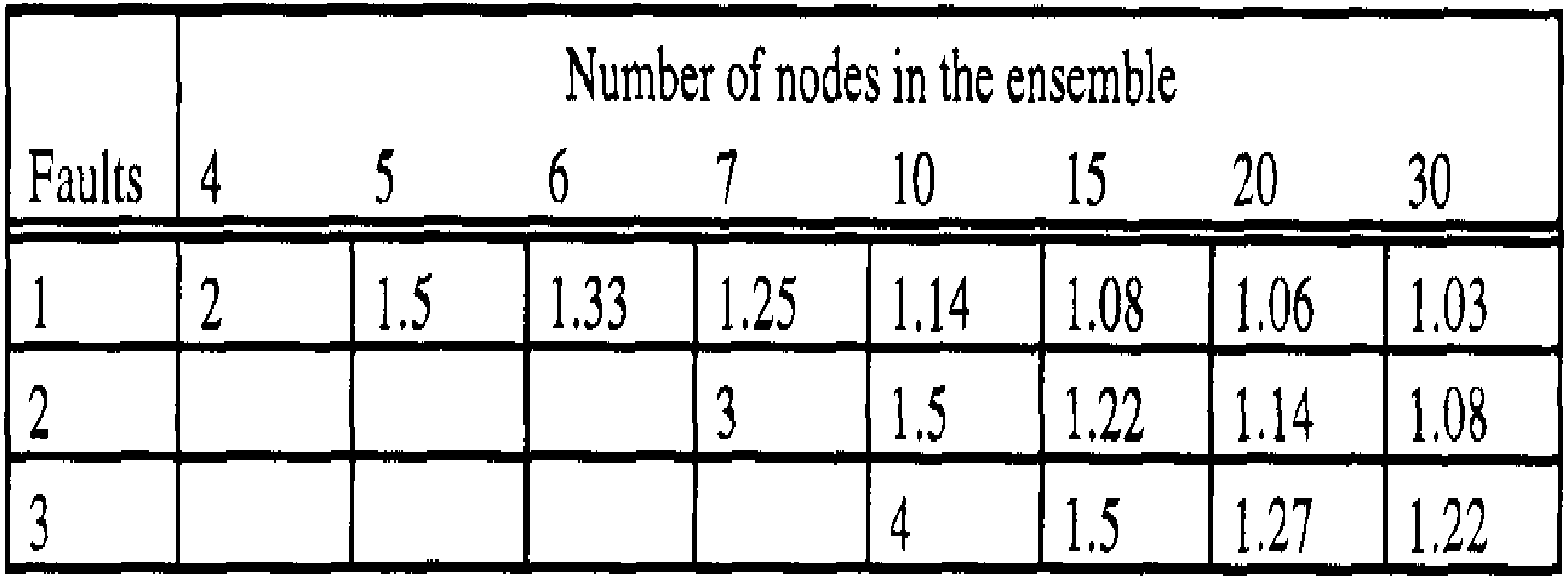

j
View of node j
k
View of node k
Precision window Π
Time difference
⇩ good time value
⬇ malicously faulty time value
j   Average value calculated by node j
k   Average value calculated by node k

Figure : Worst possible behavior of a malicious

- The worst-case scenario occurs if all good clocks are at opposite ends of the precision window , and the Byzantine clock is seen at different corners by two nodes.

- In the example of Figure 3.12, node j will calculate an average value of 4phi/5 and node k will calculate an average value of 3/phi5; the difference between these two terms, caused by the Byzantine fault, is thus pi/5.

Precision of the FTA: Table: Byzantine error term μ(N,k)



	Number of nodes in the ensemble							
Faults	4	5	6	7	10	15	20	30
1	2	1.5	1.33	1.25	1.14	1.08	1.06	1.03
2				3	1.5	1.22	1.14	1.08
3					4	1.5	1.27	1.22

State Correction versus Rate Correction

EXTERNAL CLOCK SYNCHRONIZATION

- External synchronization links the global time of a cluster to an external standard of time.

- For this purpose it is necessary to access a time server, i.e., an external time source that periodically broadcasts the current reference time in the form of a time message.

- This time message must raise a synchronization event (such as the beep of a wrist watch) in a designated node of the cluster and must identify this synchronization event on the agreed time scale.

- Such a time scale must be based on a constant measure of time, e.g., the physical second, and must relate the synchronization event to a defined origin of time, the epoch.

- The interface node to a time server is called a time gateway.

Principle of Operation

- Assume that the time gateway is connected to a GPS (Global Positioning System) receiver.

- This UTC time server periodically broadcasts time messages containing a synchronization event, as well as information to place this synchronization event on the TAI scale.

- The time gateway must synchronize the global time of its cluster with the time received from the time server.

- This synchronization is unidirectional, and therefore asymmetric.

Figure : Flow of external synchronization

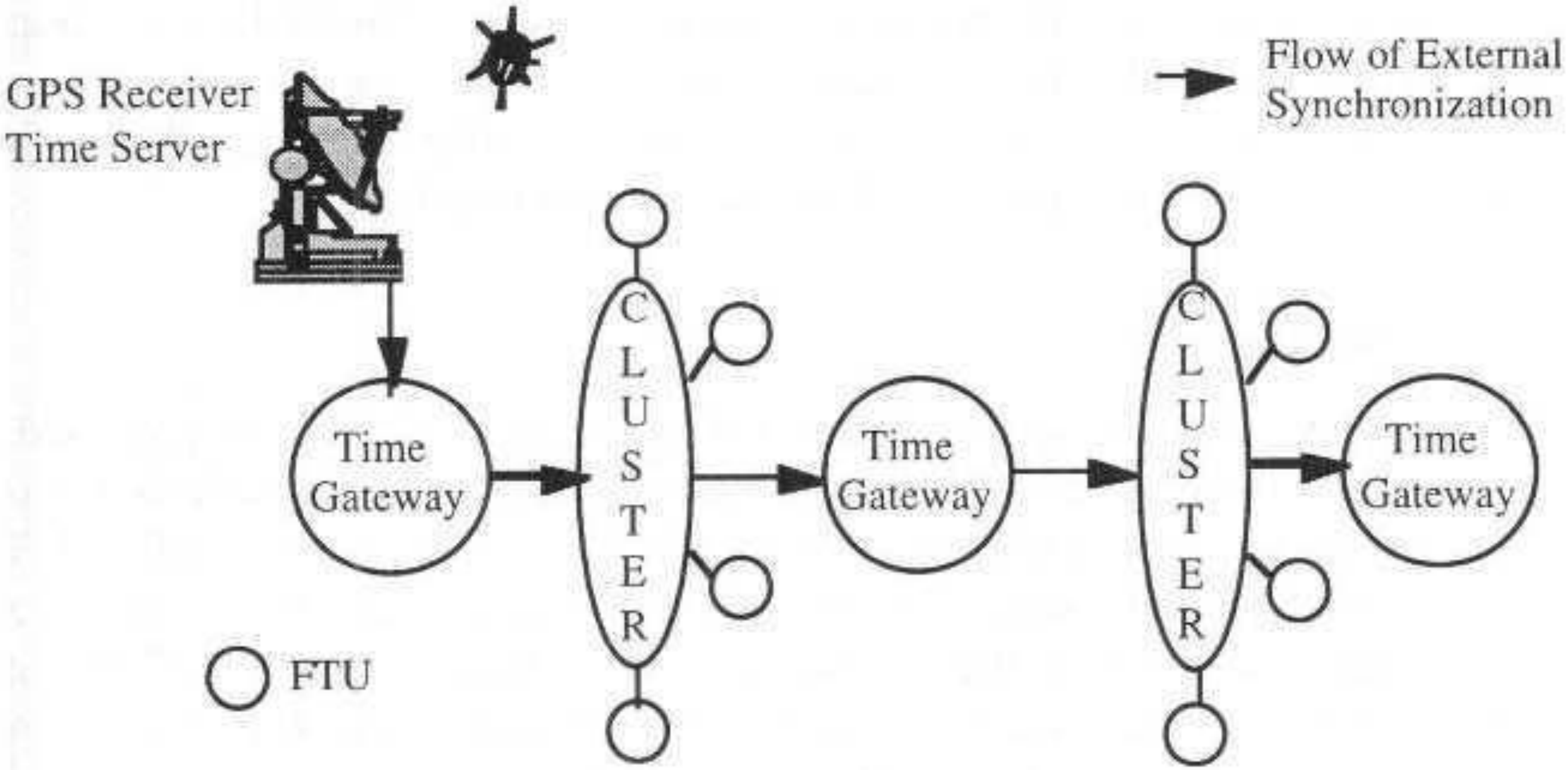


Figure : Flow of external synchronization

- If another cluster is connected to this "primary" cluster by a secondary time gateway, then, the unidirectional synchronization functions in the same manner.
- The secondary time gateway considers the synchronized time of the primary cluster as its time reference, and synchronizes the global time of the secondary cluster.
- While internal synchronization is a cooperative activity among all the members of a cluster, external synchronization is an authoritarian process: the time server forces its view of external time on all its subordinates.
- From the point of view of fault tolerance, such an authoritarian regime introduces a problem: if the authority sends an incorrect message, then all its "obedient" subordinates will behave incorrectly.
- However, for external clock synchronization, the situation is under control because of the "inertia" of time.

Figure : Flow of external synchronization

- Once a cluster has been synchronized, the fault-tolerant global time base within a cluster acts as a monitor of the time server.

- A time gateway will only accept an external synchronization message if its content is sufficiently close to its view of the external time.

- The time server has only a limited authority to correct the drift rate of a cluster.

- The enforcement of a maximum common-mode drift rate– we propose less than 10-4 sec/sec–is required to keep the error in relative time-measurements small.

- The maximum correction rate is checked by the software in each node of the cluster

Figure : Flow of external synchronization

- The implementation must guarantee that it is impossible for a faulty external synchronization to interfere with the proper operation of the internal synchronization, i.e., with the generation of global time within a cluster.

- The worst possible failure scenario occurs if the external time server fails maliciously.

- This leads to a common-mode deviation of the global time from the external time base with the maximum permitted correction rate.

- The internal synchronization within a cluster will, however, not be affected by this controlled drift from the external time base.

Time Formats

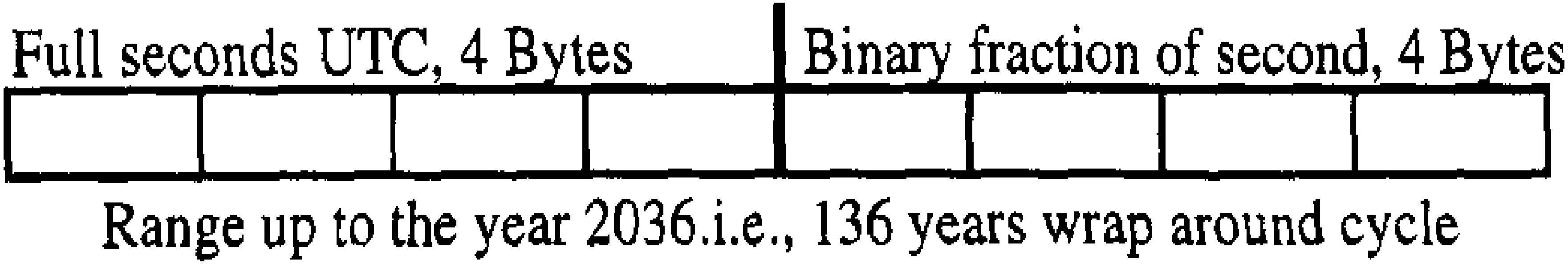
- Over the last few years, a number of external-time formats have been proposed for external clock synchronization.
- The most important one is the standard for the time format proposed in the Network Time Protocol (NTP) of the Internet [Mil91].
- This time format with a length of eight bytes contains two fields: a four byte full seconds field, where the seconds are represented according to UTC, and a fraction of a second field, where the fraction of a second is represented as a binary fraction with a resolution of about 232 pico second.

Time Formats

- On January 1, 1972, at midnight the NTP clock was set to 2,272,060,800.0 seconds, i.e., the number of seconds since January 1, 1900 at 00:00h.

Figure: Time format in the Network Time Protocol (NTP).

- **The NTP time is not chronoscopic because it is based on UTC. The occasional insertion of a leap second into UTC can disrupt the continuous operation of a time- triggered real-time system.**


Full seconds UTC, 4 Bytes
Binary fraction of second, 4 Bytes
Range up to the year 2036.i.e., 136 years wrap around cycle

Time Gateway

The time gateway must control the timing system of its cluster in the following ways:

(i) It must initialize the cluster with the current external time.

(ii) It must periodically adjust the rate of the global time in the cluster to bring it into agreement with the external time and the standard of time measurement, the second.

(iii) It must periodically send the current external time in a time message to the nodes in the cluster so that a reintegrating node can reinitialize its external time

Real time model/ Modeling Real-Time Systems

The Purpose of the Model:

- The limited information processing capacity of the human mind– compared to the large amount of information in the real world– requires a goal-oriented information reduction strategy to develop a reduced representation of the world (a model) that helps in understanding the problem posed.

- New concepts emerge and take shape if mental activity is focused on solving a particular problem. Reality can be represented by a variety of models:

- A physical-scale model of a building, a simulation model of a technical process, a mathematical model of quantum physics phenomena, or a formal logical model of the security in a computer system.

Real time model/ Modeling Real-Time Systems

- All these models are different abstractions of reality, but should not be mistaken for reality itself. A model that introduces a set of well-defined concepts and their interrelationships is called a conceptual model.

- When proceeding from informal to formal modeling, a certain order must be followed: a sound and stable conceptual model is a necessary prerequisite for any more formal model.

- Formal models have the advantage of a precise notation and rigorous rules of inference that support the automatic reasoning about selected properties of the modeled system.
- **Assumption Coverage**
- **What is Relevant?; Durations of Actions:**
- **Frequency of Activations: What Is Irrelevant? : Issues of Representation**

Temporal relations, dependability

- **DEPENDABILITY:** is a measure of a system’s availability, reliability and its maintainability, and maintenance support performance --in some cases, other characteristics such as durability, safety and security.

- **FUNCTIONAL REQUIREMENTS:** Data Collection, Signal Conditioning:, Alarm Monitoring:, Direct Digital Control, Man-Machine Interaction.

- **TEMPORAL REQUIREMENTS:** Where Do Temporal Requirements Come From?, A Simple Control Loop:, The Controlled Object:, Controlling Computer System:, Delay.

Temporal relations, dependability

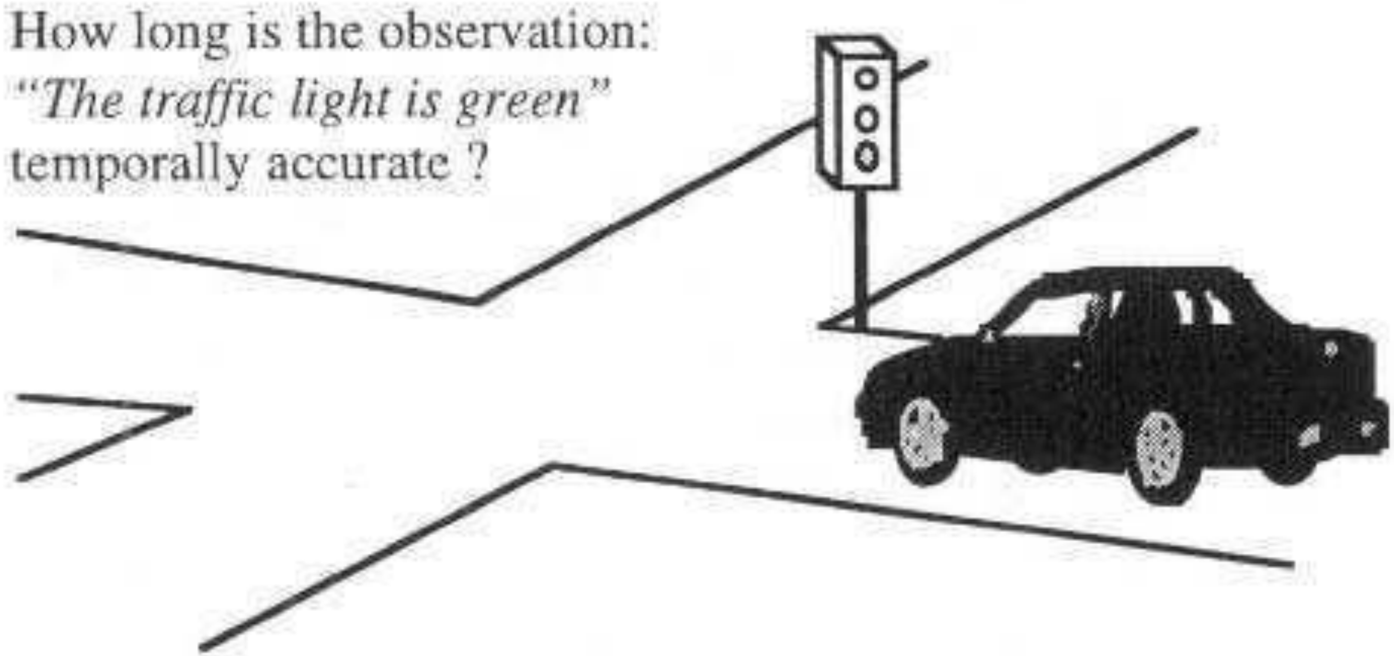


Figure 1.2: Temporal accuracy of the traffic light information.

Data Collection

- Controlled object, e.g., a car or an industrial plant, changes its state as a function of time.

- If we freeze time, we can describe the current state of the controlled object by recording the values of its state variables at that moment.

- Possible state variables of a controlled object "car" are the position of the car, the speed of the car, the position of switches on the dash board, and the position of a piston in a cylinder.

- We are normally not interested in all state variables, but only in the subset of state variables that is significant for our purpose. A significant state variable is called a real-time (RT) entity.

Data Collection

- Every RT entity is in the sphere of control (SOC) of a subsystem, i.e., it belongs to a subsystem that has the authority to change the value of this RT entity.

- Outside its sphere of control, the value of an RT entity can be observed, but cannot be modified.

- For example, the current position of a piston in a cylinder of the engine of a controlled car object is in the sphere of control of the car.

- Outside the car, the current position of the piston can only be observed.

Data Collection

- The first functional requirement of a real-time computer system is the observation of the RT entities in a controlled object and the collection of these observations.

- An observation of an RT entity is represented by a real-time (RT) image in the computer system. Since the state of the controlled object is a function of real time, a given RT image is only temporally accurate for a limited time interval.

- **The length of this time interval depends on the dynamics of the controlled object. If the state of the controlled object changes very quickly, the corresponding RT image has a very short accuracy interval.**

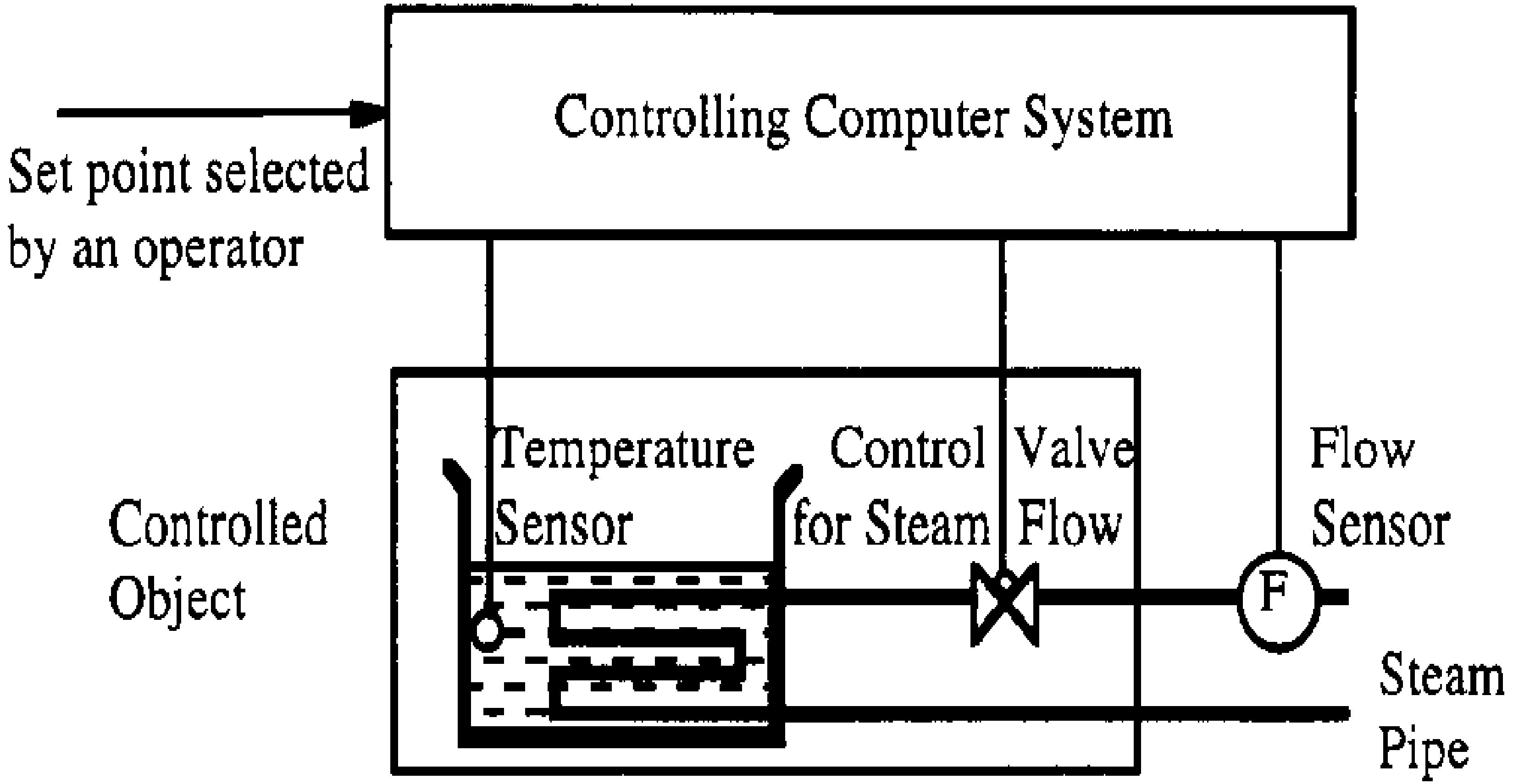
TEMPORAL REQUIREMENTS

Where Do Temporal Requirements Come From?:

- The most stringent temporal demands for real-time systems have their origin in the requirements of the control loops,

- e.g., in the control of a fast mechanical process such as an automotive engine. The temporal requirements at the man-machine interface are, in comparison, less stringent because the human perception delay, in the range of 50-100 msec, is orders of magnitudes larger than the latency requirements of fast control loops

Figure 1.3: A simple control loop.


Controlling Computer System
Set point selected by an operator
Controlled Object
Temperature Sensor
Control Valve for Steam Flow
Flow Sensor
F
Steam Pipe

A simple control loop.

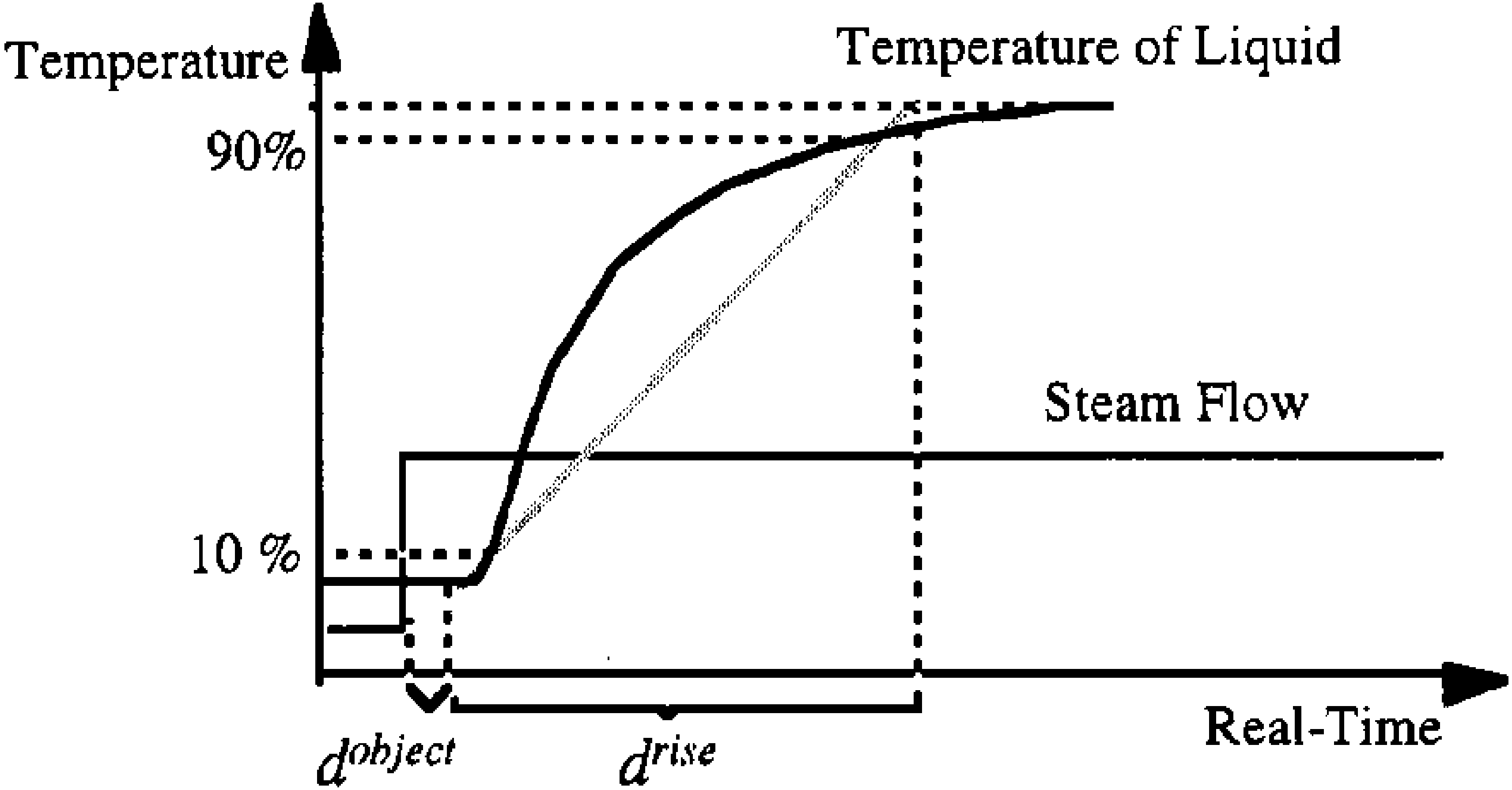
- **A Simple Control Loop: Consider the simple control loop depicted in Figure 1.3 consisting of a vessel with a liquid,**

- A heat exchanger connected to a steam pipe, and a controlling computer system.

- The objective of the computer system is to control the valve (control variable) determining the flow of steam through the heat exchanger so that the temperature of the liquid in the vessel remains within a small range around the set point selected by the operator.

- The focus of the following discussion is on the **temporal properties of this simple control loop consisting of a controlled object and a controlling computer system.**

Figure 1.4: Delay and rise time of the step response.



TEMPORAL REQUIREMENTS

- A hard real-time system must maintain synchrony with the state of the environment (the controlled object and the human operator) in all operational scenarios.

- It is thus paced by the state changes occurring in the environment.
- Because the state of the controlled object changes as a function of real-time,

- An **observation is temporally accurate only for a limited time interval.**
- Real-time systems have only small data files, the real-time database that is formed by the temporally accurate images of the RT-entities.

- **The key concern is on the short-term temporal accuracy of the real-time database that is invalidated by the flow of real-time.**

TEMPORAL REQUIREMENTS

- The most stringent temporal demands for real-time systems have their origin in the requirements of the control loops.

- The temporal behavior of a simple controlled object can be characterized by process lag and rise time of the step-response function.

- Example. 1. **Event-Triggered Communication Systems**

  2. Time-Triggered Communication Systems
- Temporal control in an ET system is a global issue, depending on the behavior of all nodes in the system. From the point of view of temporal behavior, ET systems are not composable.

TEMPORAL REQUIREMENTS

In a time-triggered communication system that transports only state messages, **temporal control resides within the communication system.**

Since system integration will not change the temporal properties of the CNI, a TT architecture is composable with respect to **temporal properties.**

Observations;

Global Time: A global time base helps to establish such a consistent temporal order on the basis of the time stamps of the events.

TEMPORAL REQUIREMENTS

- Event triggered, Rate constrained, Time triggered.

- Event-Triggered versus Time-Triggered

Event-Triggered Communication Systems:

- The decision of either settling for a time-triggered or an event-triggered architecture clearly deals with non-functional properties of a real-time system, and should therefore be postponed as far as possible.

Event-Triggered Communication Systems

- If a communication system transports event messages, the temporal control is external to the communication system.

- It is within the sphere of control of the host computers to decide when a message must be sent.

- Consider the case where a number of nodes decide to send a message to a particular receiving node at the same instant.

- If the communication system has dedicated channels between any two nodes, all messages will arrive simultaneously at the receiver, and overload the receiver.

Event-Triggered Communication Systems

- This does not solve the fundamental problem, namely that the temporal control at the CNI is not defined by an ET protocol.

- Temporal control in an ET system is thus a global issue, depending on the behavior of the application software in all nodes of the distributed system.

- From the point of view of temporal behavior, ET systems are not composable.

- The basic capability to talk to each other does not ensure a disciplined conversation.

Time-Triggered Communication Systems

- In a time-triggered (TT) communication system, temporal control resides within the communication system, and is not dependent on the application software in the nodes.

- State messages are transported from the sender CNI to the receiver CNI at predetermined points in time which are stored in message scheduling tables within the communication controllers.

- The host computers have no opportunity to influence the temporal behavior of the communication system.

Time-Triggered Communication Systems

- The CNI is strictly a data-sharing interface without any control signals crossing the interface.
- It thus acts as a temporal firewall, isolating the temporal behavior of the host computer from the temporal behavior of the communication system.
- There is no possibility for control-error propagation from the host to the communication system, and vice versa.
- The temporal properties of the CNI between a node and the communication system are fully defined at design time.
- It is thus possible to test each node individually with respect to the CNI.
- Since system integration will not change the temporal properties of the CNI, a TT architecture is composable with respect to communication timeliness.

Time-Triggered Communication Systems

Event-Triggered versus Time-Triggered are refined and extended beyond the communication system.

A temporal control signal for the activation of a task in a node can arise from one of the following two sources:

(i) The control signal is derived from a significant state change, an event, in the environment or within the computer system.

(ii) Examples of such significant state changes are the depressing of a push button by an operator, the activation of a limit switch, the arrival of a new message at a node, or the completion of a task within a node.

Time-Triggered Communication Systems

- We call a control signal that is derived from a significant state change an event trigger.

- A system where all the control signals are derived from event triggers is **called an event-triggered (ET) system.**

- The control signal is derived from the progression of real-time. Whenever the real-time clock within a node reaches a preset value specified in a scheduling table, a temporal control signal is generated. We call such a control signal that is derived from the progression of time a time trigger. A system where all the control signals are derived from time triggers is **called a time-triggered (TT) system.**

Event & Time-Triggered Communication Systems

- **EXAMPLE: The design of a computer system controlling a set of elevators in a high rise building can be event-triggered or time-triggered.**
- In an event-triggered implementation, every press of the lift-call button causes an interrupt in the computer system, and activates a task that reschedules the lifts to service the request.

- In a time-triggered implementation, every press of the lift-call button sets a local memory element in the lift-call button.

- The memory elements of all the lift-call buttons are periodically sampled with a sampling period of, say, 500 msec and then reset by the computer system.

- After a complete sampling cycle, the lift scheduler is activated to calculate a new schedule to service all requests.

Event & Time-Triggered Communication Systems

- If a user becomes impatient if the lift does not arrive and presses the lift-call button again, the different implementations will handle the redundant call-button pushes differently.

- The event triggered implementation will relay additional interrupts to the computer system, while the time-triggered implementation will not recognize the redundant call-button signals as long as the memory elements are set.

Time-Triggered Embedded Systems

Embedded system: a special-purpose computer designed to perform one or few dedicated functions, often with real-time computing constraints

Real-time adjective: bounded response time

Examples :Car Anti-Lock Braking System

- Avionics Pressure Cabin Control
- Mirowave Oven RF Controller
- Train Inter-Lock System
- GSM Subscriber Base Station
- Bluetooth Hands-Free Set
- Electronic Ignition System • Any many, many

Hard vs. Soft Real-Time Embedded Systems

- Hard deadline systems: missing a program/task execution time deadline can have catastrophic consequences (financial, human loss of life, etc.)
- Soft deadline systems: missing a dealine may not be critical and can be tolerated to a certain degree (e.g., VoIP)

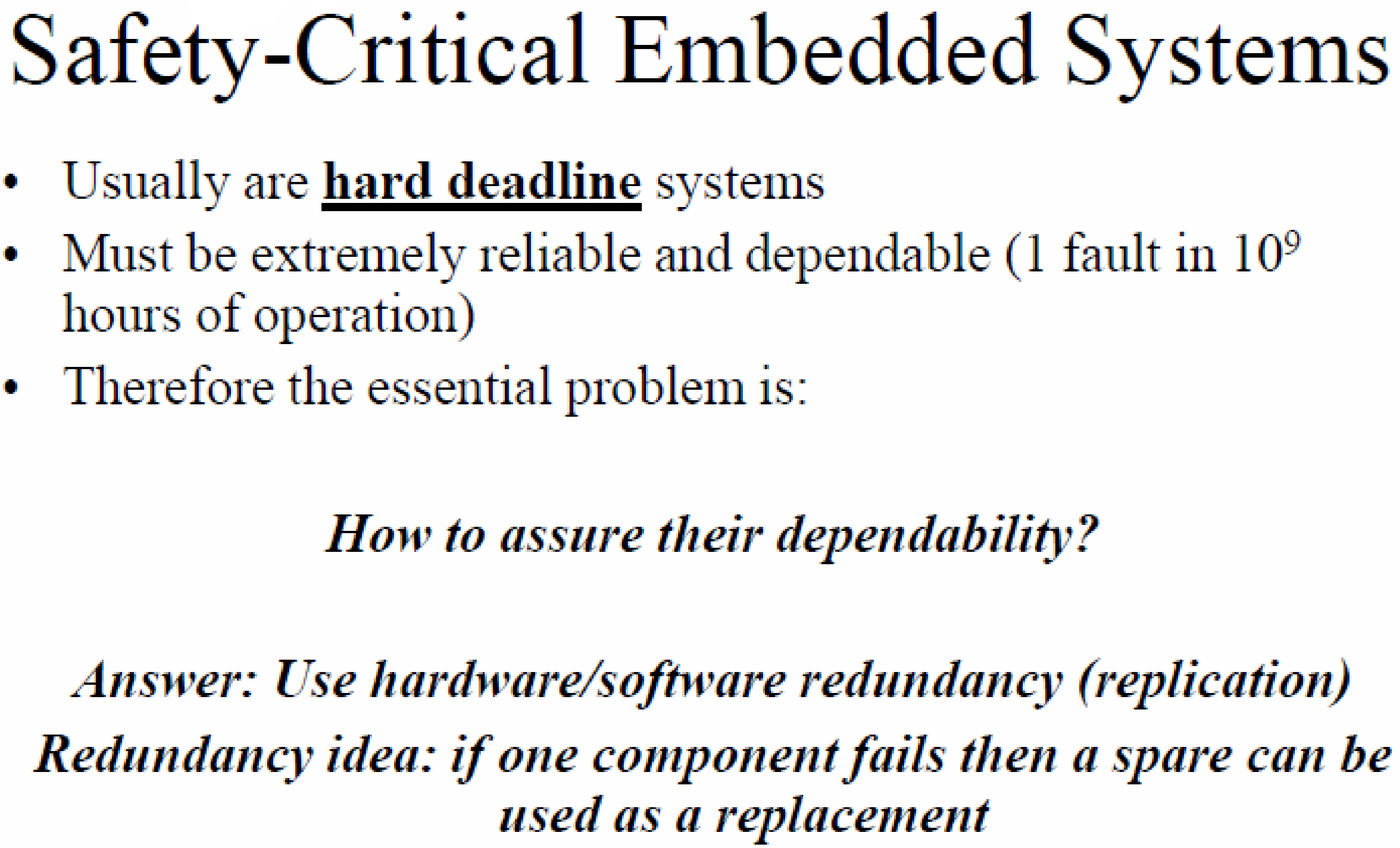
Safety-Critical Embedded Systems

- Usually are hard deadline systems
- Must be extremely reliable and dependable (1 fault in 109 hours of operation)
- Therefore the essential problem is: How to assure their dependability?

Answer: Use hardware/software redundancy (replication) Redundancy idea:

if one component fails then a spare can be used as a replacement

Safety- critical Embedded systems



Safety-Critical Embedded Systems

- Usually are **hard deadline** systems
- Must be extremely reliable and dependable (1 fault in 10^9 hours of operation)
- Therefore the essential problem is:

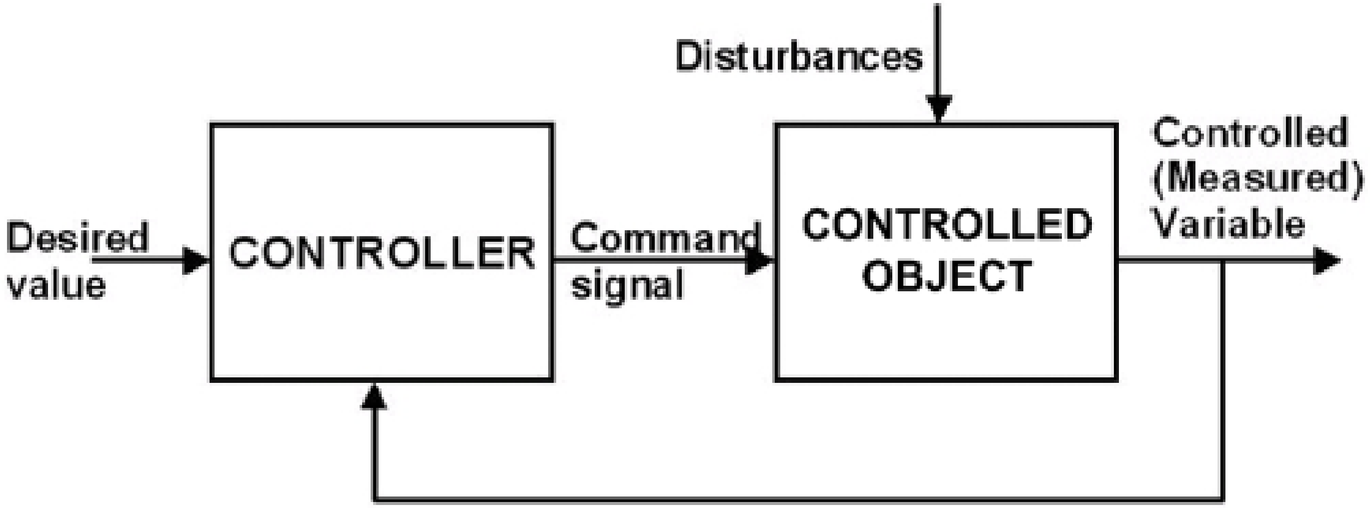
How to assure their dependability?

Answer: Use hardware/software redundancy (replication)

Redundancy idea: if one component fails then a spare can be used as a replacement

Embedded System's main components

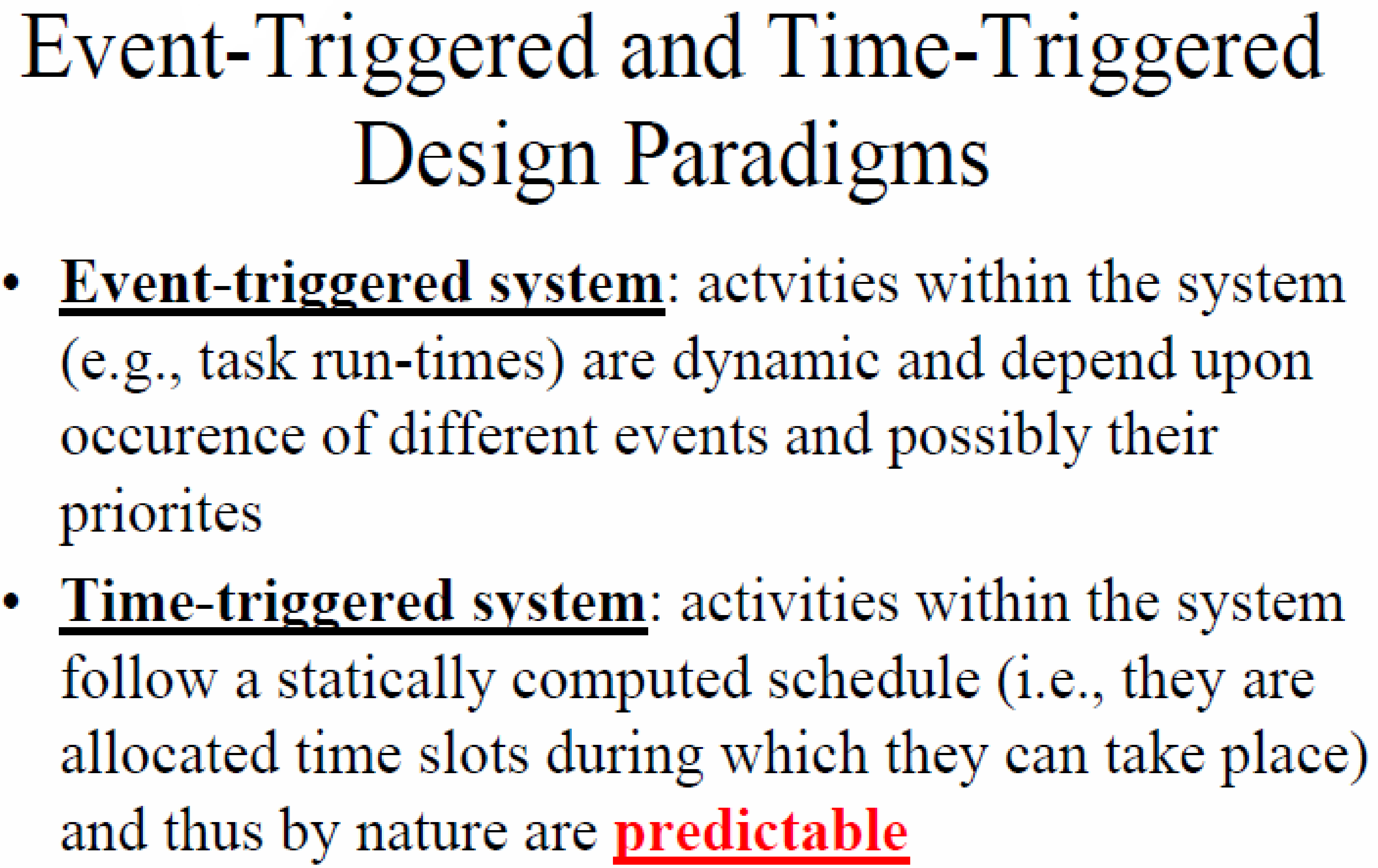
Embedded System's Main Components



Controller: local or a distributed if **distributed then it must be connected into a network (e.g., a data bus)**

Controlled object: physical, biological, or chemical process or object (e.g., combustion engine)

Event Triggered and Time Triggered Design Paradigms



Event-Triggered and Time-Triggered Design Paradigms

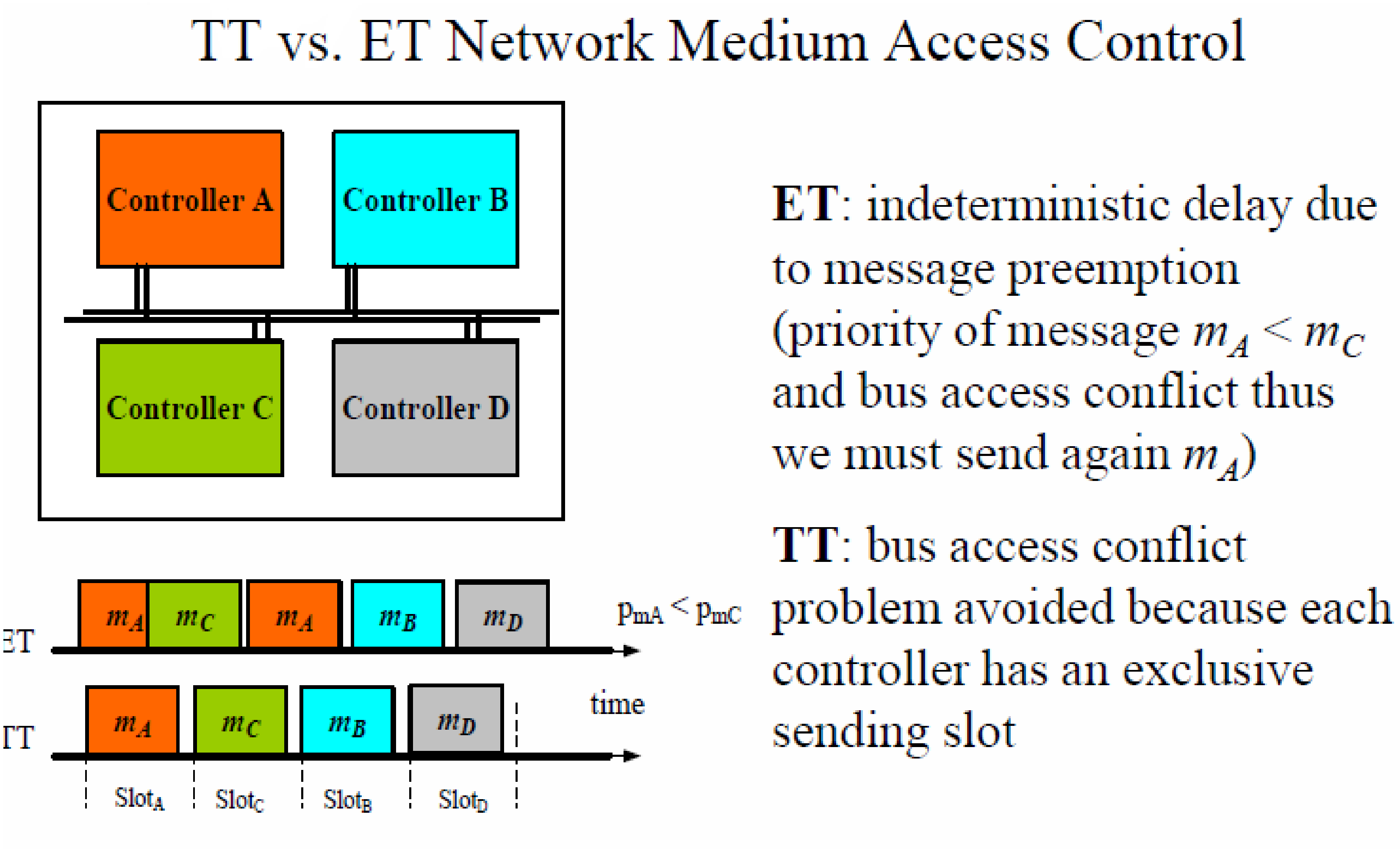
- **Event-triggered system**: actvities within the system (e.g., task run-times) are dynamic and depend upon occurence of different events and possibly their priorites
- **Time-triggered system**: activities within the system follow a statically computed schedule (i.e., they are allocated time slots during which they can take place) and thus by nature are **predictable**

Paradigm Relationship

Paradigm Relationship

- Event-triggered (ET) and time-triggered (TT) system design can relate to the operating system, application or communication (network) behavior mode (sporadic activity processing – ET or periodic activity processing –TT)
- Here we focus on the TT network's main component **known as medium access control (MAC) protocol layer**

TT vs ET Network Medium Access Control



TT vs. ET Network Medium Access Control

ET: indeterministic delay due to message preemption (priority of message $m_A < m_C$ and bus access conflict thus we must send again m_A)

TT: bus access conflict problem avoided because each controller has an exclusive sending slot

TT vs ET Network Medium Access Control

- Time-triggered (TT) technology relates to the overall system architecture (operating system, network, application)
- TT paradigm more dominant in the safety-critical sector
- TTA/TTP – fault tolerant TT network architecture and communication protocol with over 20 years of research behind it (Tech. Univ. of Vienna, TTTech, and others) – **FGCU also uses it!**
- Applicability assessment of time-triggered and event-triggered technology is not trivial and depends on application’s time of activity (periodic, aperiodc, hybrid)

UNIT II
REAL TIME OPERATING SYSTEMS

REAL-TIME OPERATING SYSTEMS

- Inter Component Communication
- Task Management and dual role of time
- Inter task interactions
- process input/ouput
- Agreement protocols
- Error detection

Real Time Operating System Definition

- **Real Time:** A **real time** is the time which continuously increments at regular intervals after the start of the system and time for all the activities at difference instances take that time as a reference in the system.

- **RTOS: A real time operating system (RTOS) is** multitasking operation system for the applications with hard or soft real time constraints.

- **Real-time constraint** means constraint on occurance of an event and system expected response and latency to the event.

Introduction

- Embedded systems are tightly coupled to their environment.

- This imposes real-time constraints by the need to interact with the environment.

–required speeds of motion,
– required precision of measurement,
– required time durations.

Examples of Embedded Devices

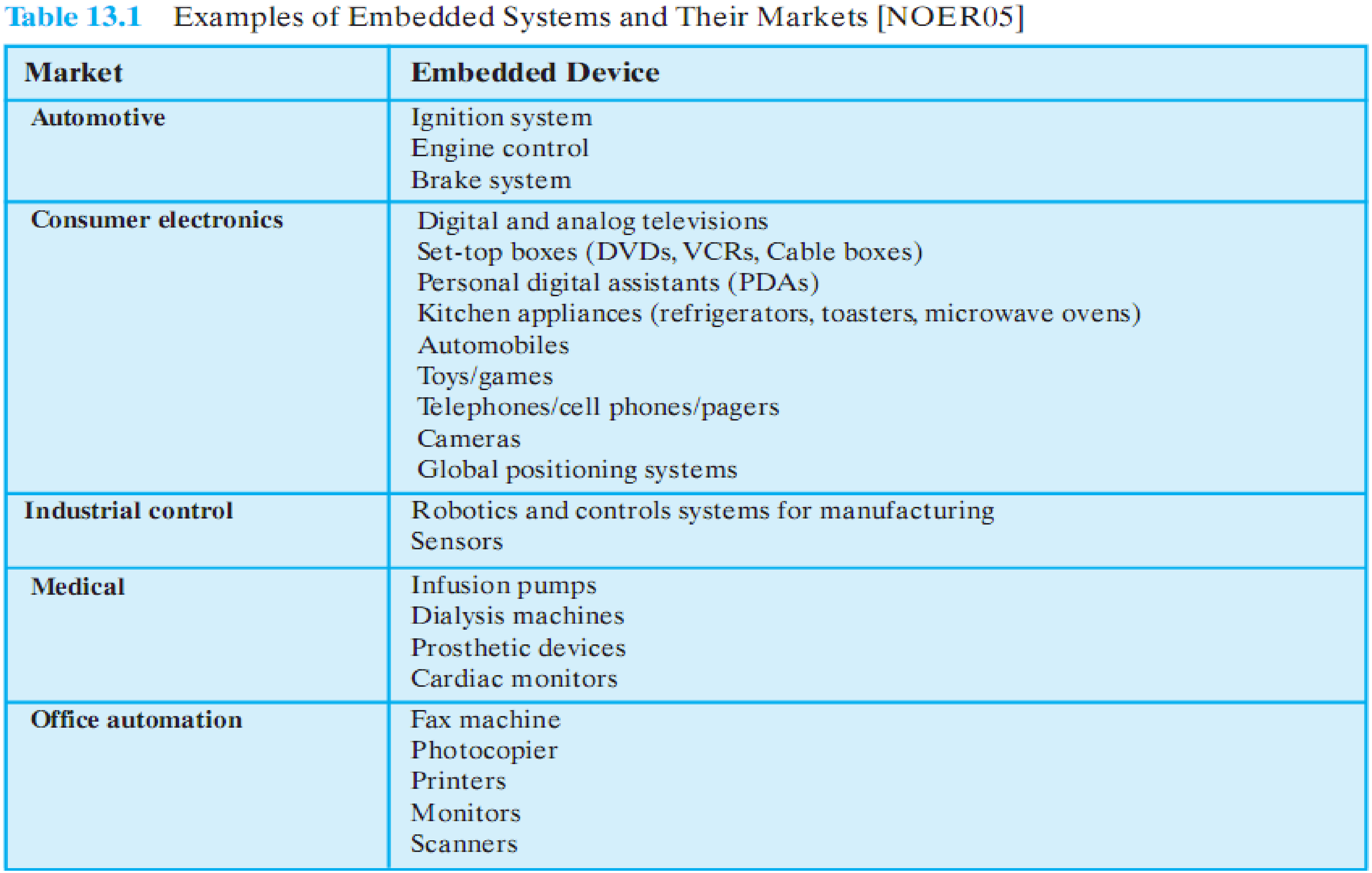


Table 13.1 Examples of Embedded Systems and Their Markets [NOER05]

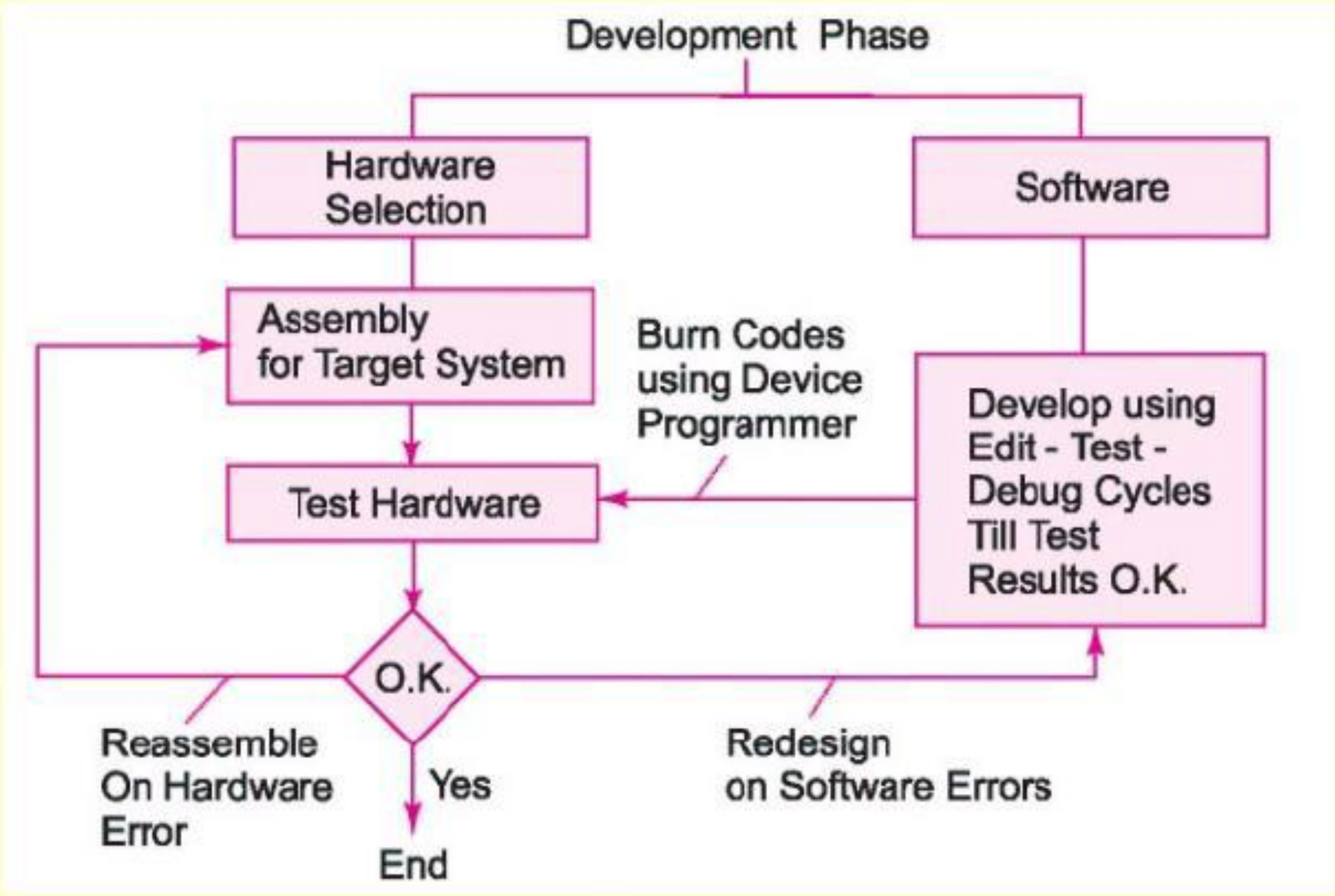
Market	Embedded Device
Automotive	Ignition system Engine control Brake system
Consumer electronics	Digital and analog televisions Set-top boxes (DVDs, VCRs, Cable boxes) Personal digital assistants (PDAs) Kitchen appliances (refrigerators, toasters, microwave ovens) Automobiles Toys/games Telephones/cell phones/pagers Cameras Global positioning systems
Industrial control	Robotics and controls systems for manufacturing Sensors
Medical	Infusion pumps Dialysis machines Prosthetic devices Cardiac monitors
Office automation	Fax machine Photocopier Printers Monitors Scanners

RTOS

- RTOS is an OS for response time controlled and event controlled processes.
- The processes have predicable latencies and execute by preemptive scheduling
- An RTOS is an OS for the systems having the hard or soft real timing constraints and deadline on the tasks, ISTs and ISRs

Development process of an embedded system

Development process of an embedded system

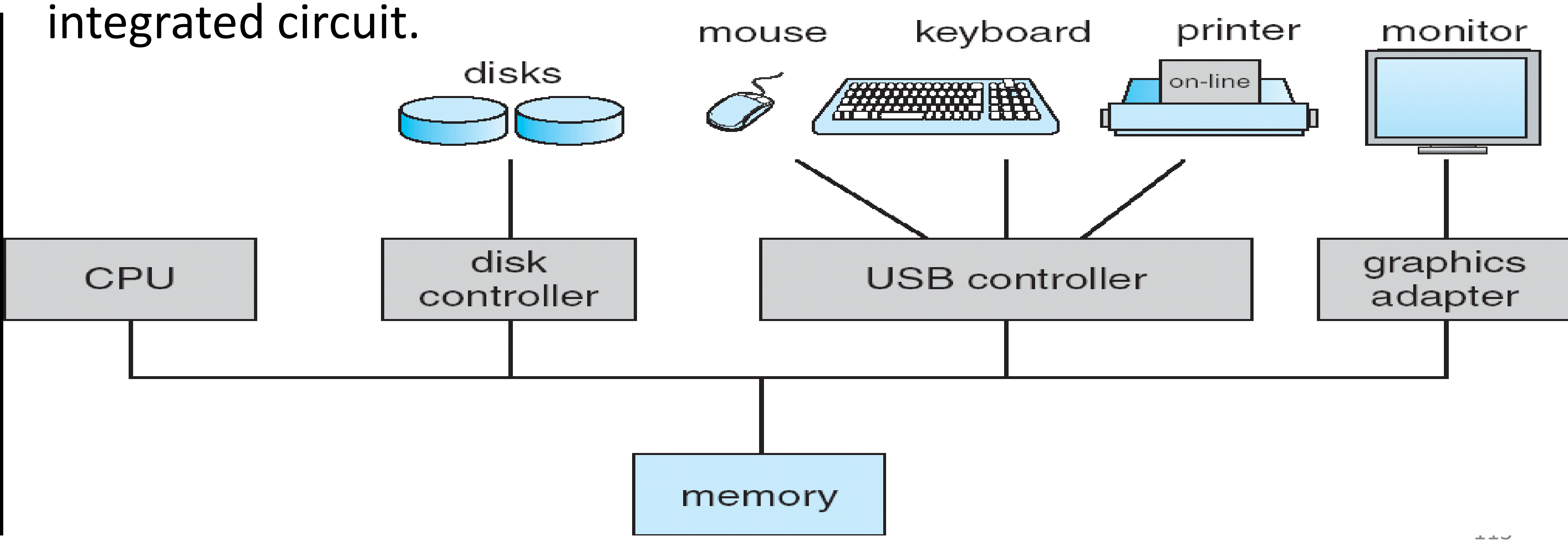

Development Phase
Hardware Selection
Software
Assembly for Target System
Burn Codes using Device Programmer
Develop using Edit - Test - Debug Cycles Till Test Results O.K.
Test Hardware
O.K.
Reassemble On Hardware Error
Yes
Redesign on Software Errors
End

System Characteristics

- Single purpose, Small size, Inexpensively mass-produced, Specific timing requirements

System-on-a-Chip ;

- Many real-time systems are designed using system-on-a-chip (SOC) strategy.
- SOC allows the CPU, memory, memory-management unit, and attached peripheral ports (I.e. USB) to be contained in a single integrated circuit.



Features of Real-Time Kernels

- Most real-time systems do not provide the features found in a standard desktop system.

Reasons include,

–Real-time systems are typically single-purpose.

–Real-time systems often do not require interfacing with a user.

–Features found in a desktop PC require more substantial hardware that what is typically available in a real-time system.

Virtual Memory in Real-Time Systems

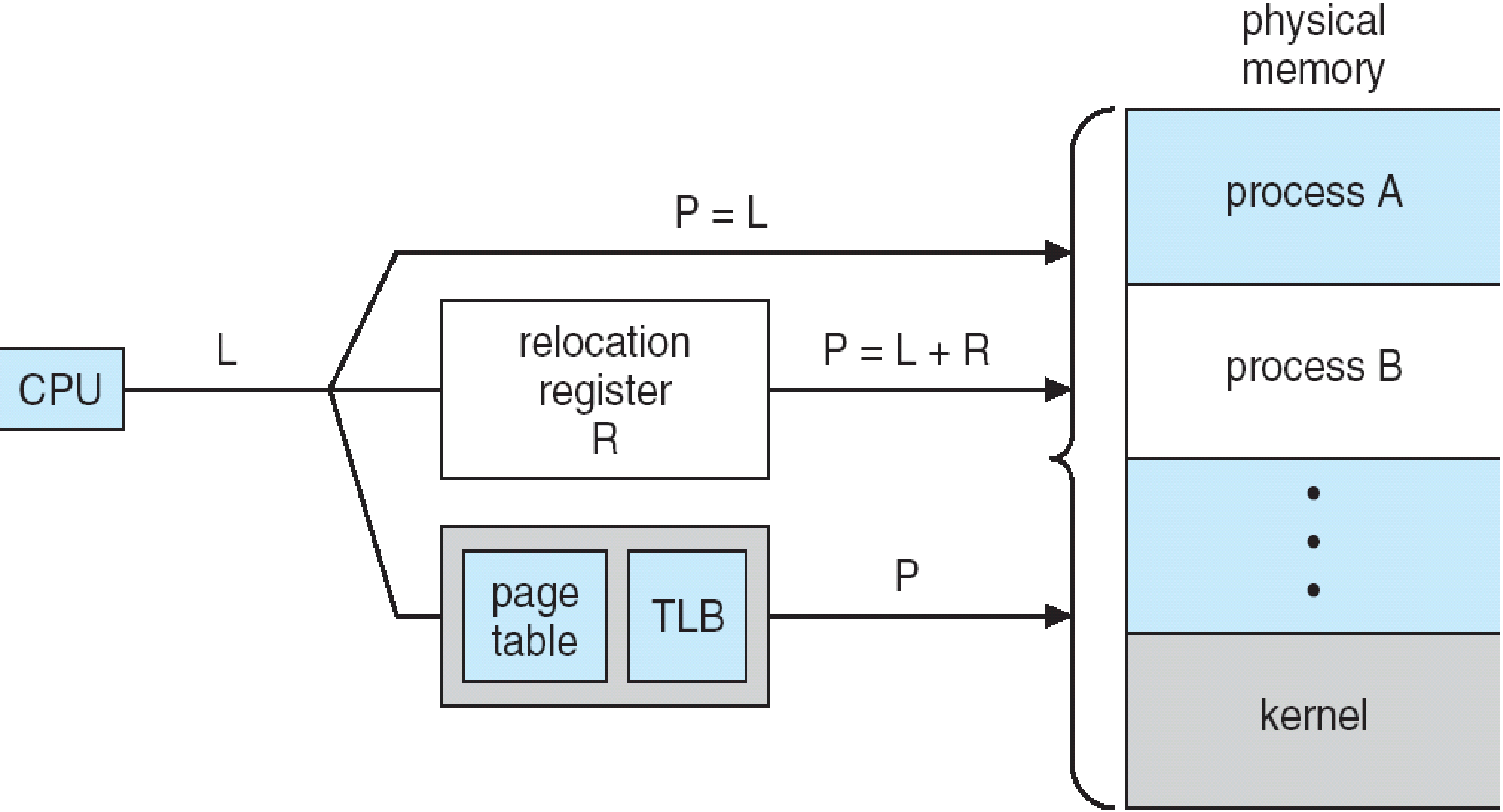
Address translation may occur via:

(1) **Real-addressing mode where programs** generate actual addresses.

(2) **Relocation register mode.**

(3) Implementing full **virtual memory.**

Address Translation



Implementing Real-Time Operating Systems

In general, real-time operating systems must provide:
(1) Preemptive, priority-based scheduling
(2) Preemptive kernels
(3) Latency must be minimized

- RTOS is an OS for response time controlled and event controlled processes. The processes have predicable latencies and execute by pre-emptive scheduling

- An RTOS is an OS for the systems having the hard or soft real timing constraints and deadline on the tasks

Real-Time Operating Systems

Motivation

- Most current operating systems are not suitable for developing or deploying applications with real-time constraints i.e. scheduling policies, process synchronization, system architecture.

Goals

- Create an environment suitable for developing applications with hard real-time constraints on task execution in a reactive environment. Two key ideas:

Predictability: Predict direct consequences of any scheduling decision.

Schedulability Guarantees: Verify the schedulability of a given set of tasks.

Design Issues

- How can I guarantee predictability and schedulability with the different components of my system?
- Architecture, Memory Management, Task structure, Kernel, etc...

Priority-based kernels

- **Objective:**
  High performance for average response time to external events.

- **Characteristics**:
  Fast context switching - small footprint - efficient interrupt handling – pre-emptable primitives - fast communication mechanisms

- Driven by a real-time clock Task Scheduling with fixed priorities Synchronization tools, limitations Communication protocols Problem?

Real-Time extensions

- **Real-Time extensions of timesharing OS Objective:**

  Extend current (commerical) time-sharing systems to satisfy real-time constraints.

- **Characteristics:**

  Reuse standard peripherals and interfaces - Speedier development

- **Reused Kernel**

  Task Scheduling with fixed priorities

- **Tasks vs Threads?**

  Re-implemented User-Level Threads

  Non pre-emptable syscalls, interrupts problems?

Research Operating Systems

- **Objective:**

  The ability to treat tasks with explicit timing constraints, such as periods and deadlines

- **Characteristics:**

  Scheduling guarantee mechanisms - Characterize tasks with additional parameters - Avoidance of nondeterministic blocking time

- **General Purpose Operating System**

  An interface between users and hardware
  Controlling and allocating memory
  Controlling input and output devices
  Managing file systems
  Facilitating networking

Non-Real-Time systems

- Non-Real-Time systems are the operating systems most often used.
- No guaranteed worst case scheduling jitter.
- System load may result in delayed interrupt response.
- System response is strongly load dependent.
- System timing is a unmanaged resource.

What is a RTOS ?

- RTOS is a pre-emptive multitasking operating system intended for real-time applications.
- Predictable OS timing behavior.
- Able to determine task’s completion time.
- A system of priority inheritance has to exist.
- Guarantees task completion at a set deadline.

Types of RTOS

Soft Real-Time system , Hard Real-Time system

- A **real-time system** requires that results be produced within a specified deadline period.
- An **embedded system** is a computing device that is part of a larger system (I.e. automobile, airliner.)
- A **safety-critical system** is a real-time system with catastrophic results in case of failure.
- A **hard real-time system** guarantees that real-time tasks be completed within their required deadlines.
- A **soft real-time system** provides priority of real-time tasks over non real-time tasks.

Soft Real-Time system

- The soft real-time definition allows for frequently missed deadlines

- If the system fails to meet the deadline, possibly more than once ,the system is not considered to have failed

- Example : Multimedia streaming , Video games

Inter Component Communication/ Interprocess Communication

- Interprocess communication is needed to exchange information among concurrently executing tasks so progress towards the common goal can be achieved.

- There are possible types of information exchange: the direct exchange of messages among the involved tasks and the indirect exchange of information via. a common region of data.

- **Messages: If interprocess communication is based on messages, a choice must be made between event-message semantics and state-message semantics**

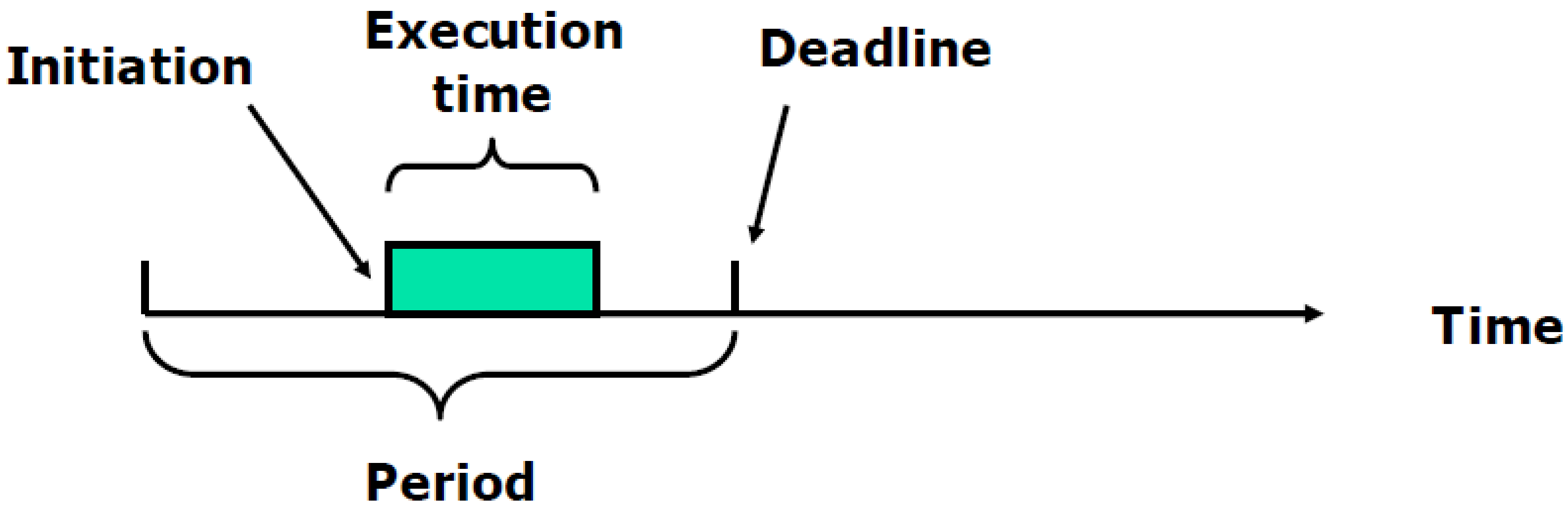
- In many real-time systems the sender and receiver tasks are periodic with differing periods.

Inter Component Communication/ Interprocess Communication

- In these systems the one-to-one synchronization requirement of event messages is not satisfied.
- Because state messages support the information exchange among tasks of differing periods, state-message semantics matches better the needs of real-time applications.
- The operating system must implement the atomicity property of a state message: a process is allowed to see only a complete version of a state message.
- The intermediate states that occur during a state message update must be hidden by the operating system.

Task Management

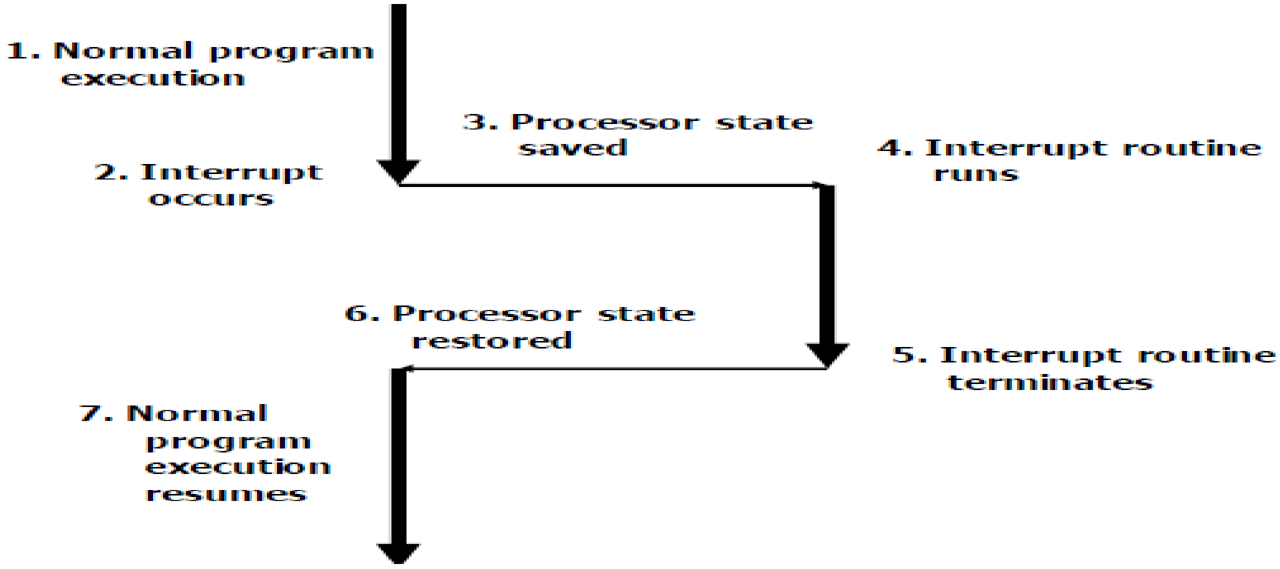
- Tasks are implemented as threads in RTOS.
- Have timing constraints for tasks
- Each task a triplet: (execution time, period, deadline)
- Can be initiated any time during the period



Task States

- **Idle :** task has no need for computer time
- **Ready :** task is ready to go active, but waiting for processor time
- **Running :** task is executing associated activities
- **Waiting :** task put on temporary hold to allow lower priority task chance to execute
- **Suspended:** task is waiting for resource

Interrupt handling


1. Normal program execution
2. Interrupt occurs
3. Processor state saved
4. Interrupt routine runs
5. Interrupt routine terminates
6. Processor state restored
7. Normal program execution resumes

Types of interrupts

- Asynchronous or hardware interrupt
- Synchronous or software interrupt
- Very low Interrupt latency
- The ISR of a lower-priority interrupt may be blocked by the ISR of a high-priority

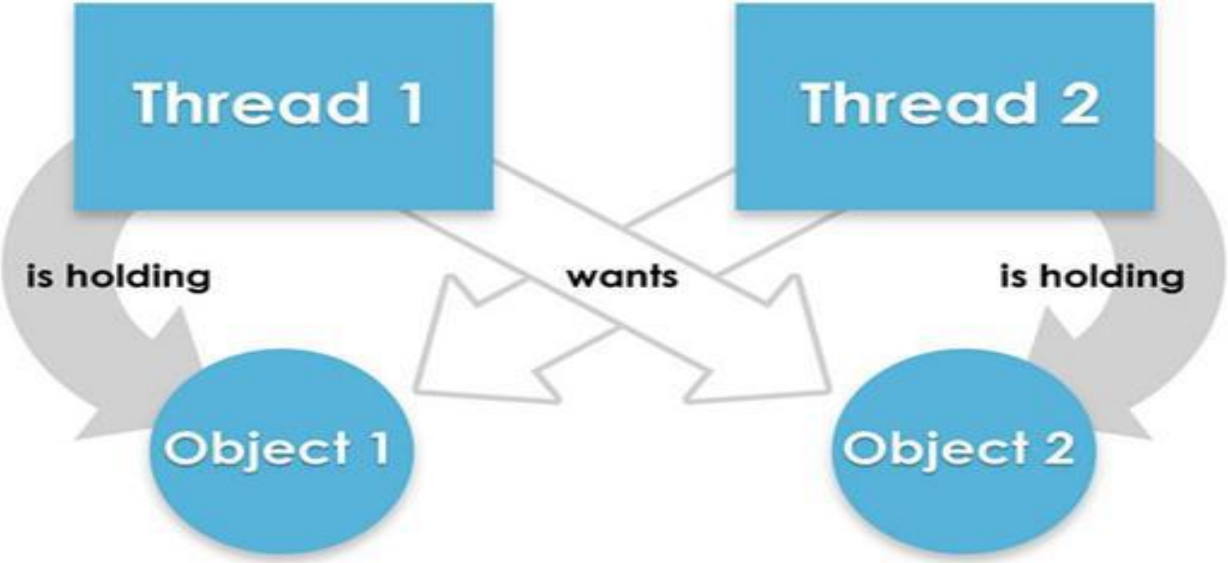
Memory management

- RTOS may disable the support to the dynamic block allocation
- When a task is created the RTOS simply returns an already initialized memory location
- when a task dies, the RTOS returns the memory location to the pool
- No virtual memory for hard RT tasks

Exception handling

- Exceptions are triggered by the CPU in case of an error
- E.g. : Missing deadline, running out of memory, timeouts, deadlocks, divide by zero, etc.
- Error at system level, e.g. deadlock
- Error at t

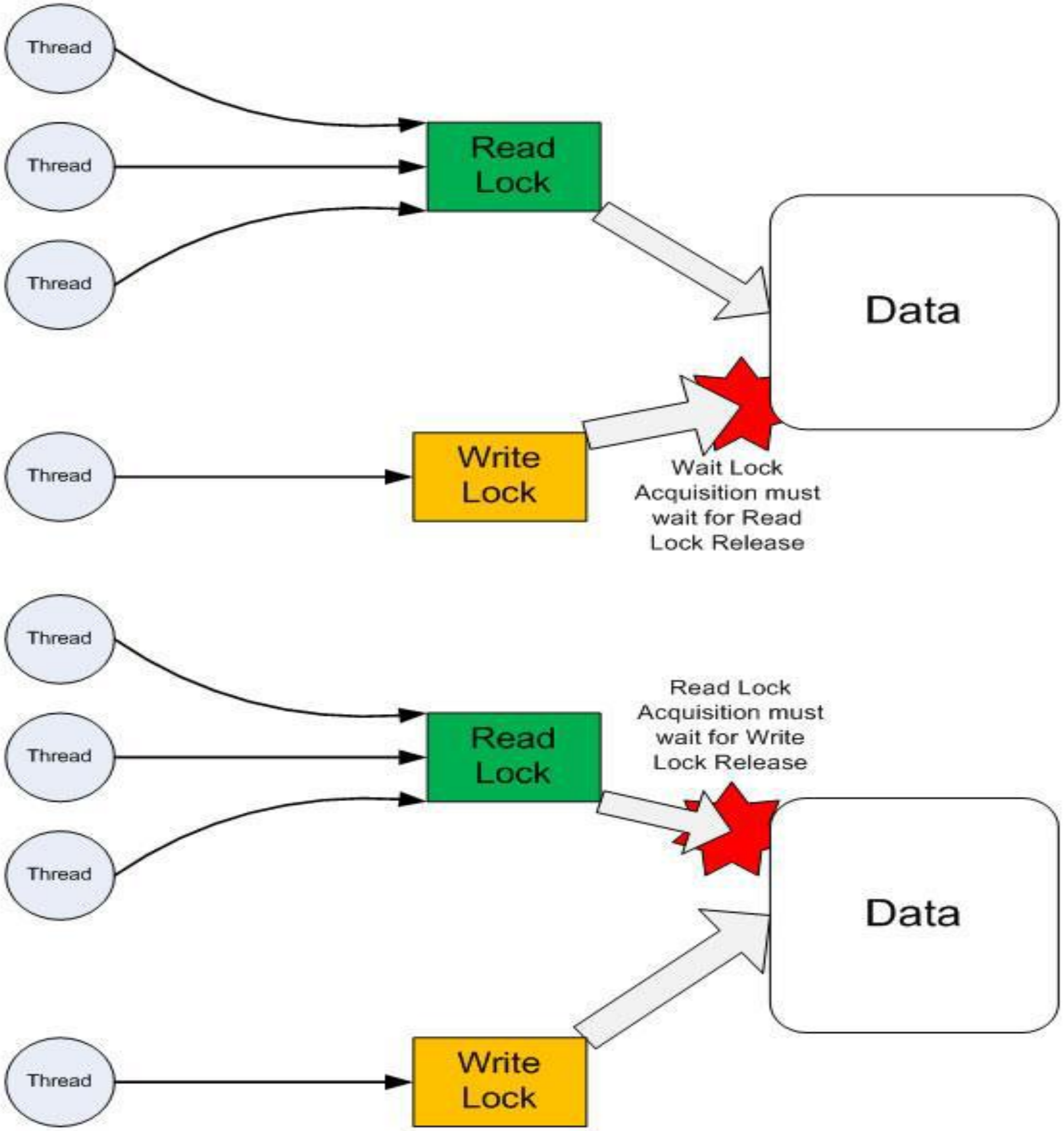
- Standard techniques:
- System calls with error code
- Watch dog
- Fault-tolerance
- Missing one possible case may result in disaster



Task Synchronization

Task Synchronization

- Semaphore
- Mutex
- Spinlock
- Read/ write locks


Thread
Thread
Thread
Read Lock
Data
Thread
Write Lock
Wait Lock Acquisition must wait for Read Lock Release
Thread
Thread
Thread
Read Lock
Read Lock Acquisition must wait for Write Lock Release
Data
Thread
Write Lock

Task scheduling

- Scheduler is responsible for time-sharing of CPU among tasks.
- Priority-based Preemptive Scheduling.
- Rate Monotonic Scheduling.
- Earliest Deadline First Scheduling.
- Round robin scheduling.

Priority-based Preemptive Scheduling & Earliest Deadline First Scheduling

Priority-based Preemptive Scheduling

- Assign each process a priority
- At any time, scheduler runs highest priority process ready to run
- Rate Monotonic Scheduling
- A priority is assigned based on the inverse of its period
- Shorter execution periods = higher priority
- Longer execution periods = lower priority

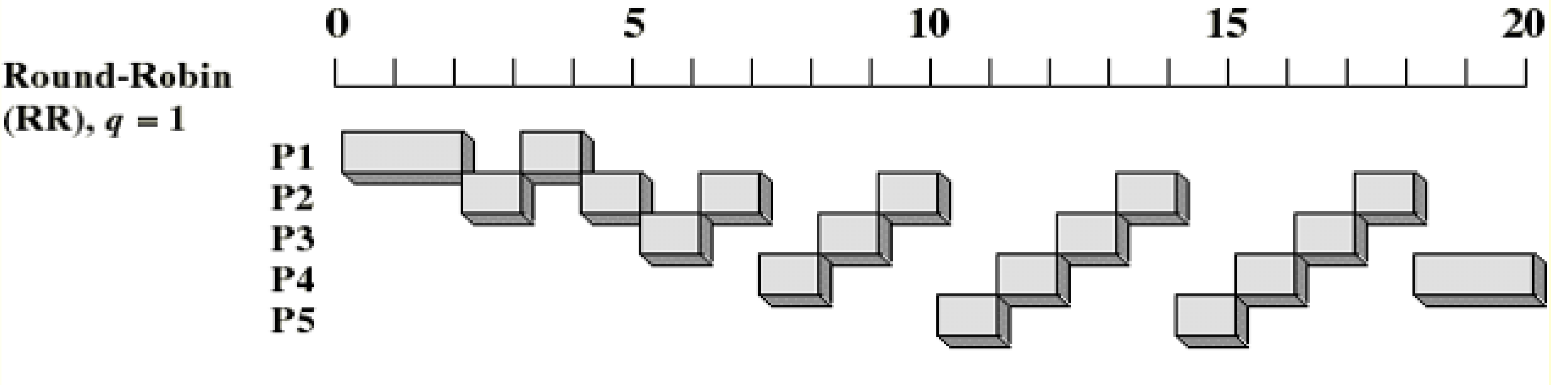
Priority-based Preemptive Scheduling & Earliest Deadline First Scheduling

Earliest Deadline First Scheduling

- Priorities are assigned according to deadlines
- The earlier the deadline, the higher the priority
- Priorities are dynamically chosen
- Round robin scheduling
- Designed for time-sharing systems
- Jobs get the CPU for a fixed time
- Ready queue treated as a circular buffer
- Process may use less than a full time slice

Example of Task scheduling (RR)

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time management

- Time interrupt: A high resolution hardware timer is programmed to interrupt the processor at fixed rate

- Each time interrupt is called a system tick The tick may be chosen according to the given task parameters

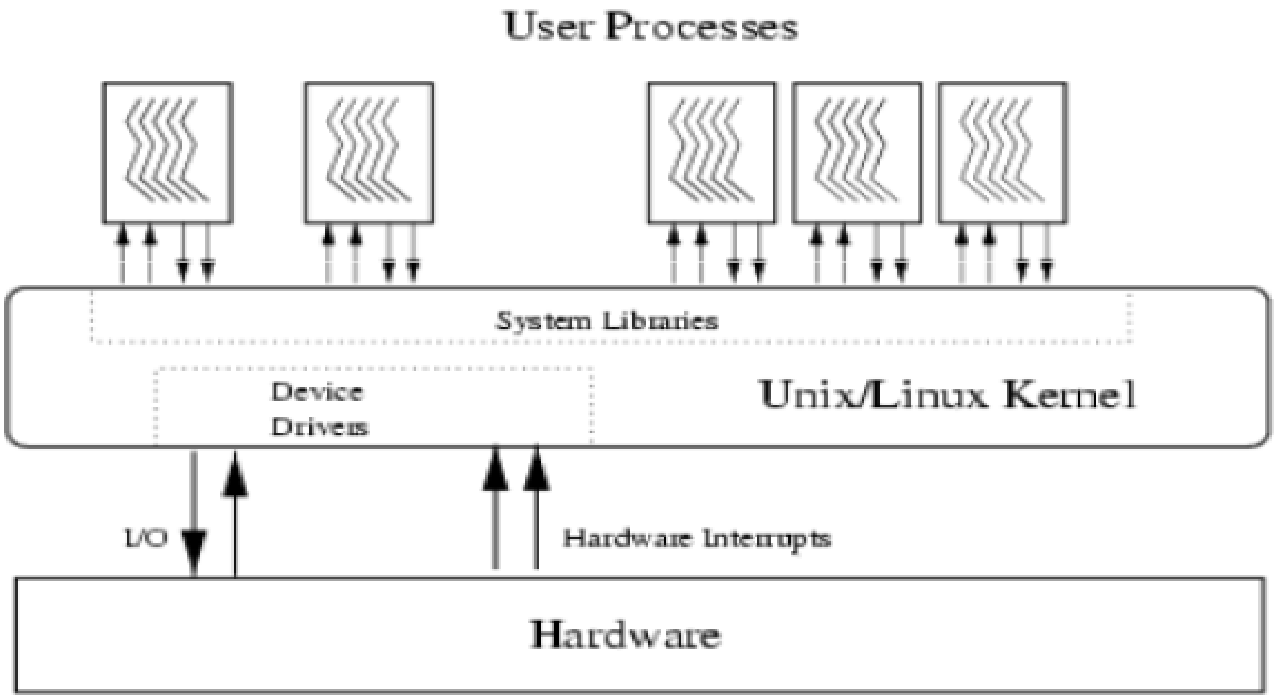
Existing RTOS categories

- Priority based kernel for Embbeded applications, VxWorks, OSE, QNX, Real Time Extensions of existing time-sharing OS, Real time Linux , Real time NT, Research RT Kernels MARS, Spring

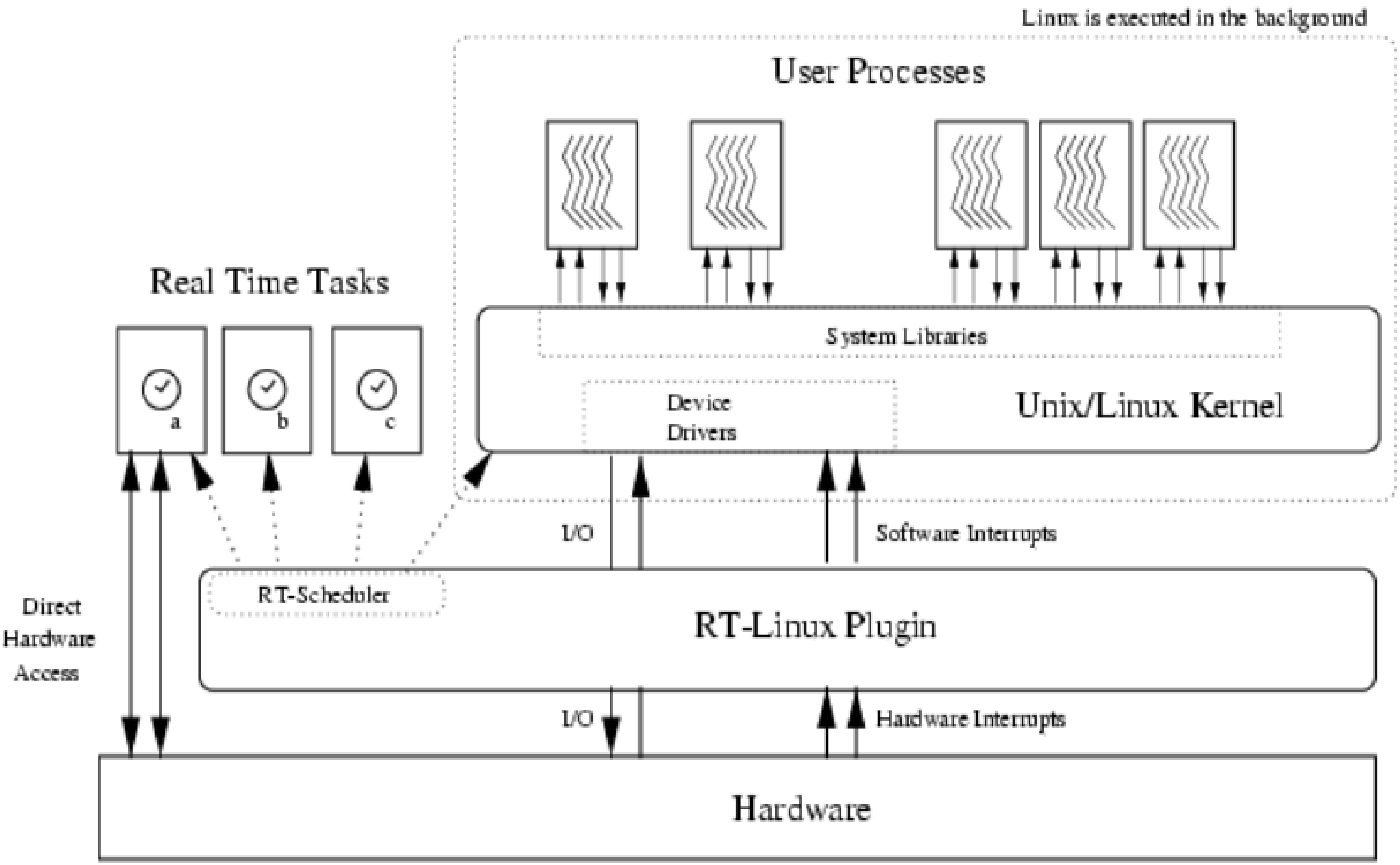
RT Linux

RT Linux: an example

RT-Linux is an operating system, in which a small real-time kernel co-exists with standard Linux kernel



RT Linux Kernel



TASK MANAGEMENT

- Task management is concerned with the provision of the dynamic environment within a host for the initialization, execution, and termination of application tasks.

Background: What's a "Real Time" System?

- When correctness of results depend on content and time
- Hard or Soft: indicates how for giving the system is

What makes an OS Real-Time?

- Predictable (possibly deterministic behavior), that's all
- Not necessarily fast
- Byproduct: mediocre throughputs

How do they work?

- Tasks are scheduled by OS according to fixed priority (typically)
- Tasks can't directly interact; they use messages or shared memory & semaphores to communicate

Designing an RTOS

Designing an RTOS: Typical Features

- Many POSIX Specs Exist
- No virtual memory (swap file)
- Shared memory capabilities
- High-resolution timer(s)
- Real-time signals/QoS

Designing an RTOS: End Goal

- Have known switching & scheduling overhead
- Avoid common problems like priority inversion and deadlock

Designing an RTOS cont…..

Designing an RTOS: Common Problems (can’t) Deadlock

- Two semaphores locked out of order by two tasks and circularly block each other
- Solution: “Instant Inheritance” implementation of Priority Ceiling Protocol – semaphores possibly needed by higher processes become priority tokens

Designing with an RTOS: What do you need?

Task information

- Priorities for each task
- Worst-case runtime
- Best-case period

Interference information

- Deadline Monotonic Analysis (DMA) calculations

Real Time Operating System (RTOS)

Fundamental requirements for an RTOS

– The OS behavior must be predictable

– The OS must be multithreaded and preemptive.

– The OS must support thread priority.

– The OS must support predictable thread synchronization mechanisms.

Additional Requirements:

–The maximum time that device drivers use to process an interrupt, and specific IRQ information relating to those device drivers, must be known.

–The interrupt latency (the time from interrupt to task run) must be predictable and compatible with application requirements.

Real Time Operating System (RTOS) cont…..

Three groups,

–Small, fast, proprietary kernels

–Real-time extensions to commercial operating systems

–Research operating systems

Small, fast, proprietary kernels

–homegrown

–commercial offerings: QNX, PDOS, pSOS, VCOS, VRTX32, VxWorks

–To reduce the run-time overheads incurred by the kernel and to make the system fast, the kernel has a small size responds to external interrupts quickly minimizes intervals during which interrupts are disabled provides fixed or variable sized partitions for memory management as well as the ability to lock code and data in memory provides special sequential files that can accumulate data at a fast rate

Real Time Operating System (RTOS) cont…..

RTOS

–To deal with timing constraints, the kernel provides bounded execution time for most primitives maintains a real-time clock provides primitives to delay processing by a fixed amount of time and to suspend/resume execution

–Also, the kernel performs multitasking and intertask communication and synchronization via standard primitives such as mailboxes, events, signals, and semaphores.

– For complex embedded systems, these kernels are inadequate as they are designed to be fast rather than to be predictable in every aspect.

Examples of RTOS

Open source:

–eCos

–Fiasco (L4 clone) [1]

–FreeRTOS

–Linux as of kernel version 2.6.18

–Phoenix-RTOS

–Nut/OS [2]

–Prex

–RTAI

–RTEMS

–RTLinux

–SHaRK [3]

–TRON Project

–Xenomai [4]

Proprietary:

–Ardence RTX - BeOS

–ChorusOS - DNIX

–DSOS - embOS (Segger)

–ITRON

–LynxOS - MicroC/OS-II

–MQX RTOS [5] - Nucleus

–OS-9 - OSE

–OSEK/VDX - OSEKtime

–PDOS - Phar Lap ETS

–PikeOS - Portos

–pSOS - QNX

–RMX - RSX-11

–RT-11 - RTOS-UH

–RTXC - Salvo RTOS [6]

–SINTRAN III - Symbian OS

–ThreadX - VRTX

–VxWorks - Windows CE

–µnOS - UNIX-RTR

RT Linux

- Real-Time Linux (now part of FSMLabs Inc.) that is suitable for Real-Time applications.
- Its view is that the application system can be split into two parts
  - Real-time
  - Non real-time

- With this approach, it splits the applications to run on either the Linux kernel, or a real-time kernel!

- Between the real and non-real parts communication is performed through FIFOs called RT-FIFOs.

These FIFOs are:

–Locked to memory in kernel space.

–FIFOs appear as devices to Linux user processes.

–Reads and writes are non-blocking and atomic.

Vx Works

Monolithic Kernel

- Reduced run-time overhead, but increased kernel size compared to Microkernel designs
- Supports Real-Time POSIX standards

Common in industry

–Mars missions

–Honda ASIMO robot

–Switches

–MRI scanners

–Car engine control systems

RTOS overview

- The worst-case administrative overhead (WCAO) of every operating system call of a real-time operating system must be known a priori, so that the temporal properties of the behavior of the complete host can be determined analytically.

- The a priori designed task schedule of a TT system must consider the required precedence and mutual exclusion relationships between the tasks such that an explicit coordination of the tasks by the operating system at run time is not necessary.

- The simplest application program interface (API) is the API of a time-triggered S-task.

- The coupling between the application program and the operating system increases with the number and variety of the operating system calls.

ERROR DETECTION

- **ERROR DETECTION**

  A real-time operating system must support error detection in the temporal domain and error detection in the value domain by generic methods.

- **Monitoring Task Execution Times**
- **Monitoring Interrupts**
- **Double Execution of Tasks**

- **Watchdogs**

  ERCOS(Embedded Real-Time Control Operating System)

- **Monitoring Task Execution Times**

  A tight upper bound on the worst-case execution time (WCET) of a real-time task must be established during software development

Embedded Real-Time Control Operating System

ERCOS(Embedded Real-Time Control Operating System)

- Task Model
- Scheduling
- Interprocess Communication
- Error Detection
- Off-line Software Tools(OLT)

- The high production volume of embedded systems demands reliable system solutions that minimize the hardware resource requirements. Many design decisions in ERCOS (Embedded Real-Time Control Operating System)

- An operating system for embedded real-time applications in the automotive industry, have been influenced by this quest for optimum performance and utmost reliability.

ERCOS(Embedded Real-Time Control Operating System)

- ERCOS is used for the implementation of embedded systems, such as engine control or transmission control, in vehicles.

- A typical state-of-the-art engine controller has a memory consisting of 256 kbyte ROM and 32 kbyte RAM.

- It interfaces to about 80 external sensors and actuators, and is connected to the other system by a real-time communication network, such as a CAN bus

- The software is organized into about 100 concurrently executing tasks. The most demanding task, the injection control, must be precise within a few microseconds.

Task Model

- The basic task model of ERCOS consists of S-tasks.
- A set of S-tasks that follow one another in sequence forms a schedule sequence.
- A schedule sequence is built offline during the static analysis of the application software.
- Each schedule sequence is assigned a given priority level, and is treated as a single unit of scheduling by the operating system.
- Whenever the activation event of a schedule sequence occurs, the whole schedule sequence is executed.

Scheduling

ERCOS supports static and dynamic scheduling of schedule sequences.

- The time triggered static schedules are developed off-line such that the required dependency relations, such as mutual exclusion and precedence between the tasks, are integrated into the off-line schedules and no explicit synchronization is needed.

Dynamic scheduling decisions are based on the priorities of ready schedule sequences.

- Two different scheduling strategies, cooperative scheduling and preemptive scheduling, are distinguished.
- Cooperative scheduling is non-preemptive at the task level.
- A context switch may only take place between the tasks of a schedule sequence. This simplifies the maintenance of data consistency, since a complete critical section is encapsulated in a single task.

Error Detection

ERCOS provides many mechanisms for run-time error detection, such as:

(i) A deadline checking service is provided by the operating system to detect late system responses, and to make it possible for an exception handler to react to such a failure.

(ii) The occurrence of interrupts originating from the controlled object is continuously monitored. After each interrupt occurrence, the interrupt line is disabled for the duration of the minimum inter arrival period.

(iii) The actual number of active instances of a task is monitored by the operating system at run time and compared with the permitted maximum number of concurrently active instances of a task that has been determined off-line.

(iv) A watchdog process generates a life-sign message with a known period so that an outside observer is continuously informed of the proper operation of a node.

OLST/ OLT(Off-line Software Tools)

- An extensive off-line software development tool (OLT) supports the design and implementation of application code for the ERCOS run-time system.
- The OLT performs a static source code analysis of the application code and generates the necessary interface code to link the application to the run-time kernel.

An overview on the OLT's functionality is given in the following:

- Support for object-based software construction and software reuse: The OLT provides the functions to structure the application software according to the ERCOS real-time object model.
- This object model supports autonomously active objects and concurrent activity within and between objects.
- To support the reuse of software in widely varying contexts, the OLT generates the necessary code to ensure data consistency in the presence of preemptive scheduling.
- Object interfaces are checked for consistency, completeness and conformance to visibility rules.

OLST/ OLT(Off-line Software Tools) cont……

(ii) **Automatic operating system configuration:**

- The ERCOS kernel configured and generated automatically by the OLT for each individual application.
- All the necessary RAM and ROM data structures are reserved by the OLT based on the static source code analysis.
- This avoids the effort for dynamic memory handling and ensures that only a minimal amount of memory is configured.

(iii) **Optimization of operating system functions:**

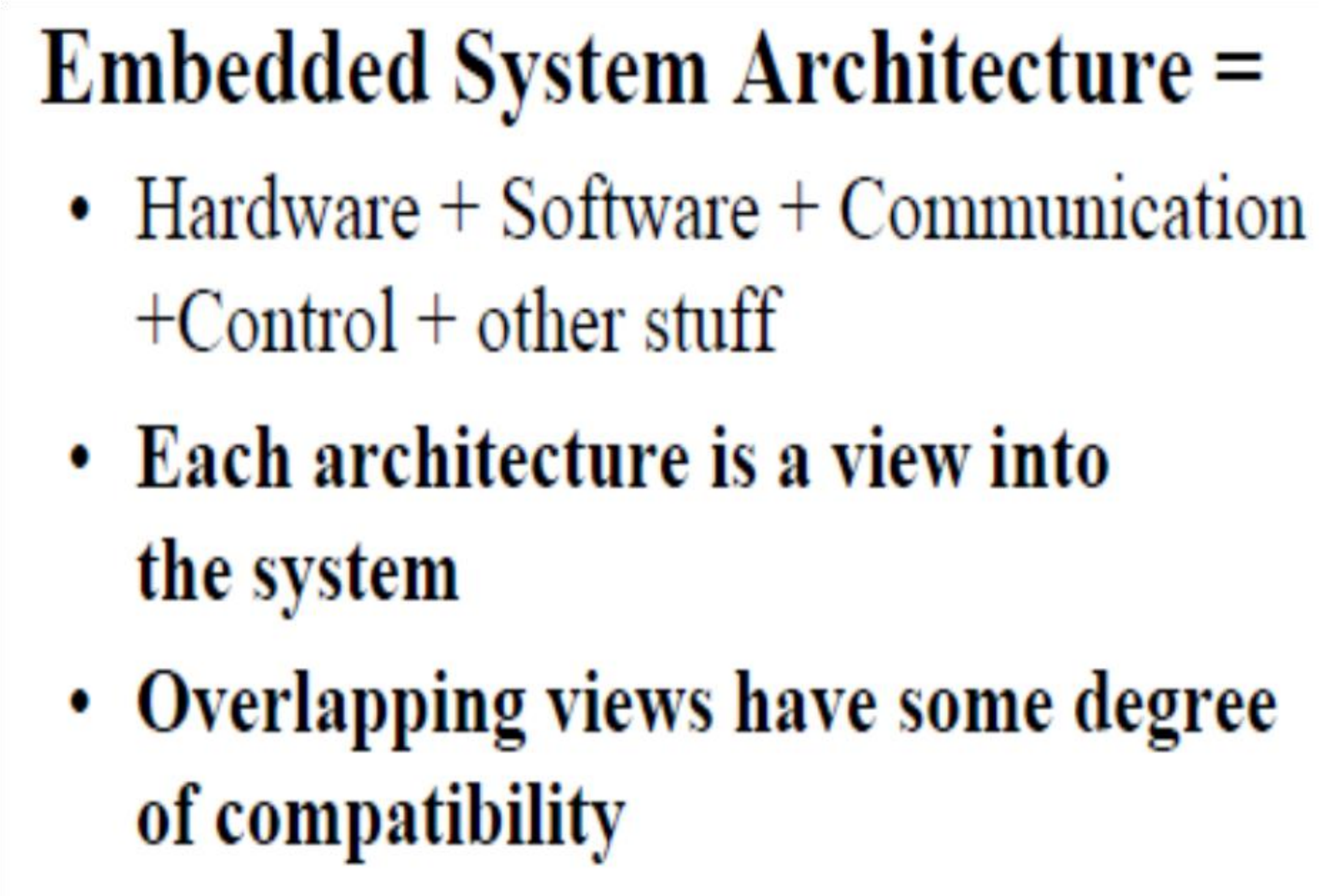
- Based on the static analysis of the source code, the OLT selects optimized implementations for operating system functions.
- For example, the static source code analyzer detects the situations where concurrency conflicts cannot arise during execution.

UNIT - III
SYSTEM DESIGN

UNIT – III

- **Scheduling problem**
- **Static and dynamic scheduling**
- **System Design**
- **Validation**
-
- **Time-triggered Architecure**

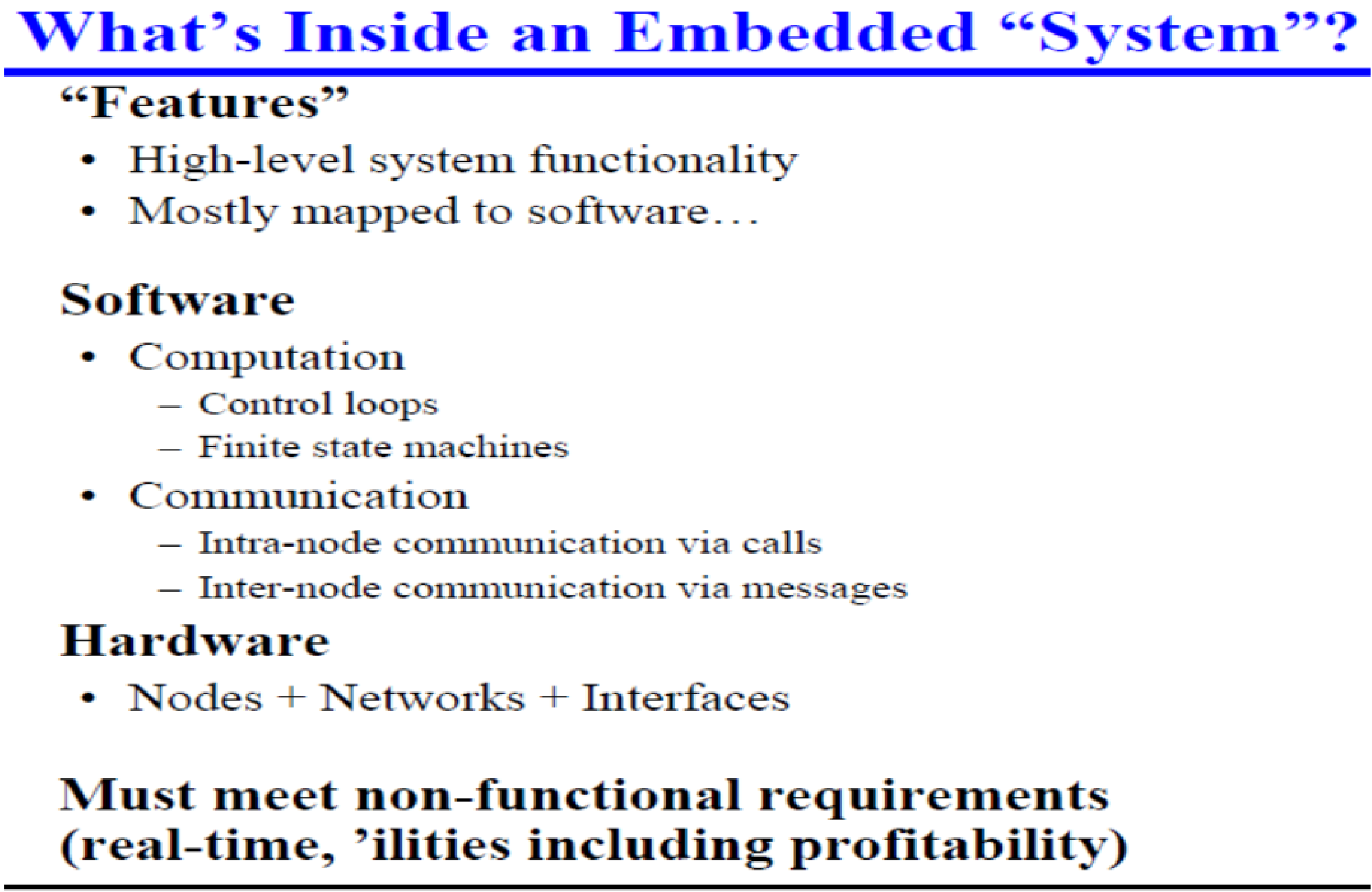
Embedded system



Embedded System Architecture =

- Hardware + Software + Communication +Control + other stuff
- **Each architecture is a view into the system**
- **Overlapping views have some degree of compatibility**

Embedded system



What’s Inside an Embedded “System”?

“Features”

- High-level system functionality
- Mostly mapped to software…

Software

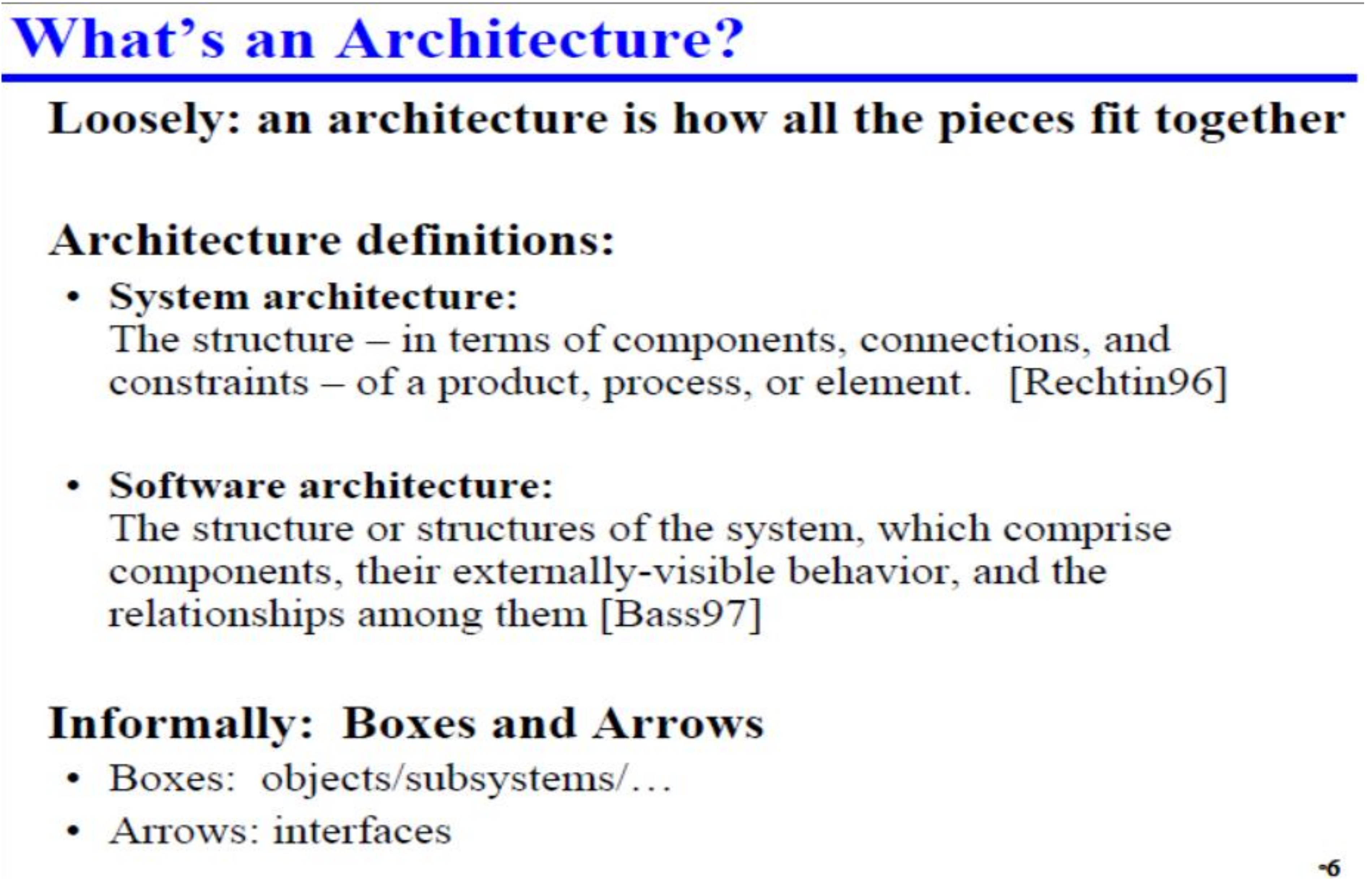
- Computation
  - Control loops
  - Finite state machines
- Communication
  - Intra-node communication via calls
  - Inter-node communication via messages

Hardware

- Nodes + Networks + Interfaces

Must meet non-functional requirements (real-time, ’ilities including profitability)

Embedded system Architecture



What's an Architecture?

Loosely: an architecture is how all the pieces fit together

Architecture definitions:

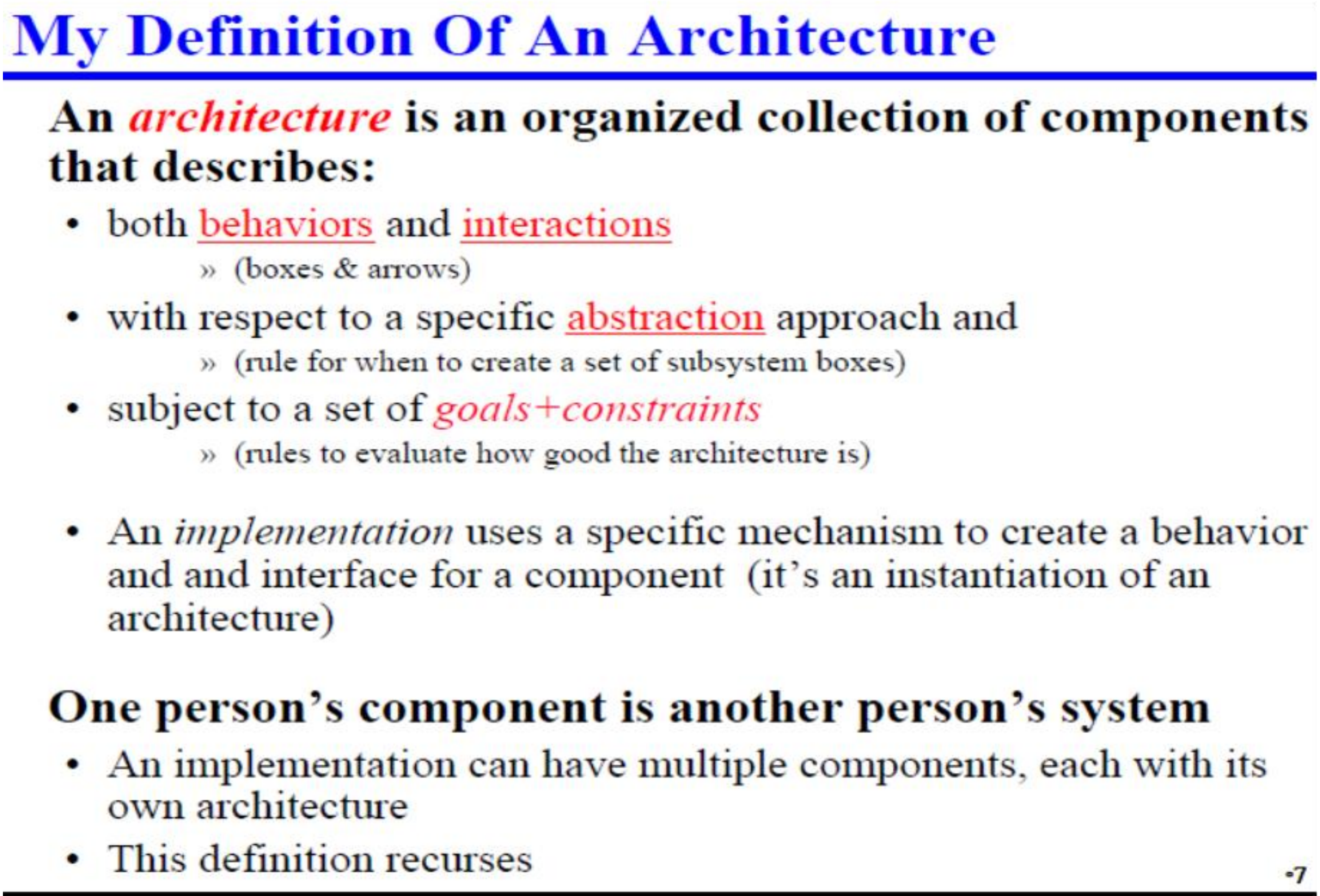
- **System architecture:**
  The structure – in terms of components, connections, and constraints – of a product, process, or element. [Rechtin96]

- **Software architecture:**
  The structure or structures of the system, which comprise components, their externally-visible behavior, and the relationships among them [Bass97]

Informally: Boxes and Arrows

- Boxes: objects/subsystems/…
- Arrows: interfaces

Embedded system Architecture



My Definition Of An Architecture

An *architecture* is an organized collection of components that describes:

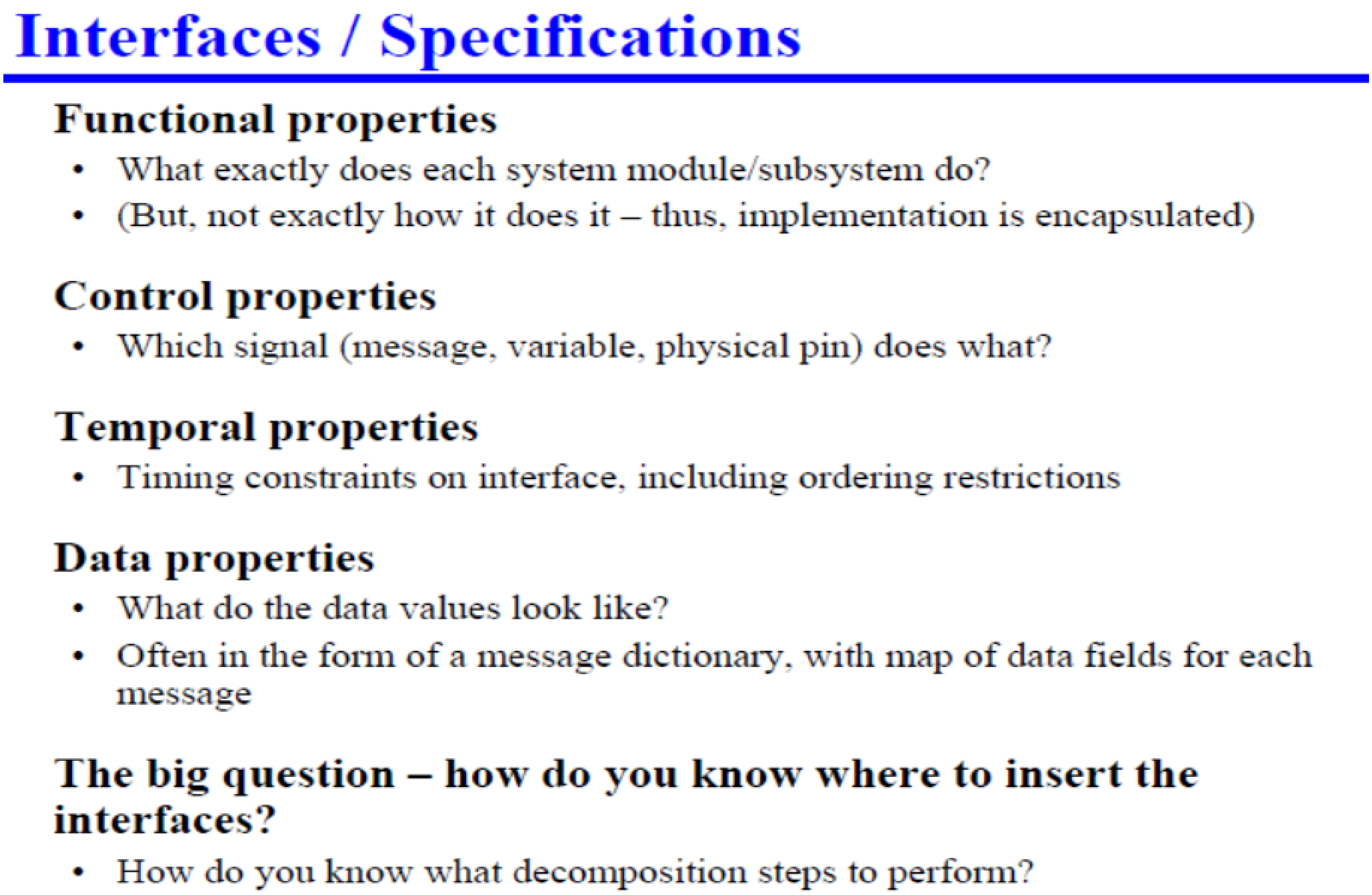
- both behaviors and interactions
  - » (boxes & arrows)
- with respect to a specific abstraction approach and
  - » (rule for when to create a set of subsystem boxes)
- subject to a set of *goals+constraints*
  - » (rules to evaluate how good the architecture is)

- An *implementation* uses a specific mechanism to create a behavior and and interface for a component (it's an instantiation of an architecture)

One person's component is another person's system

- An implementation can have multiple components, each with its own architecture
- This definition recurses

Interfaces / Specifications



Interfaces / Specifications

Functional properties

- What exactly does each system module/subsystem do?
- (But, not exactly how it does it – thus, implementation is encapsulated)

Control properties

- Which signal (message, variable, physical pin) does what?

Temporal properties

- Timing constraints on interface, including ordering restrictions

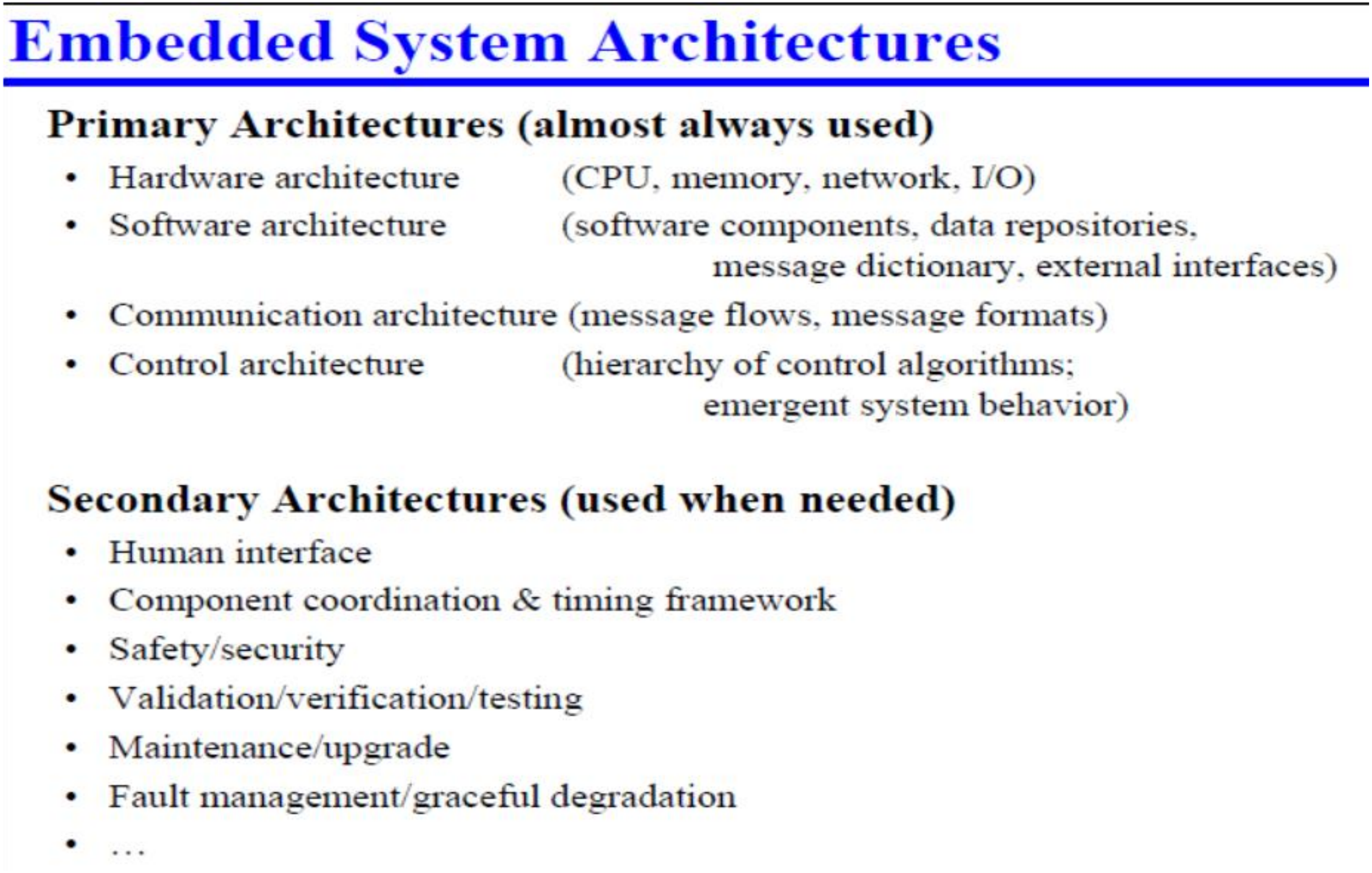
Data properties

- What do the data values look like?
- Often in the form of a message dictionary, with map of data fields for each message

The big question – how do you know where to insert the interfaces?

- How do you know what decomposition steps to perform?

Embedded System Architectures



Embedded System Architectures

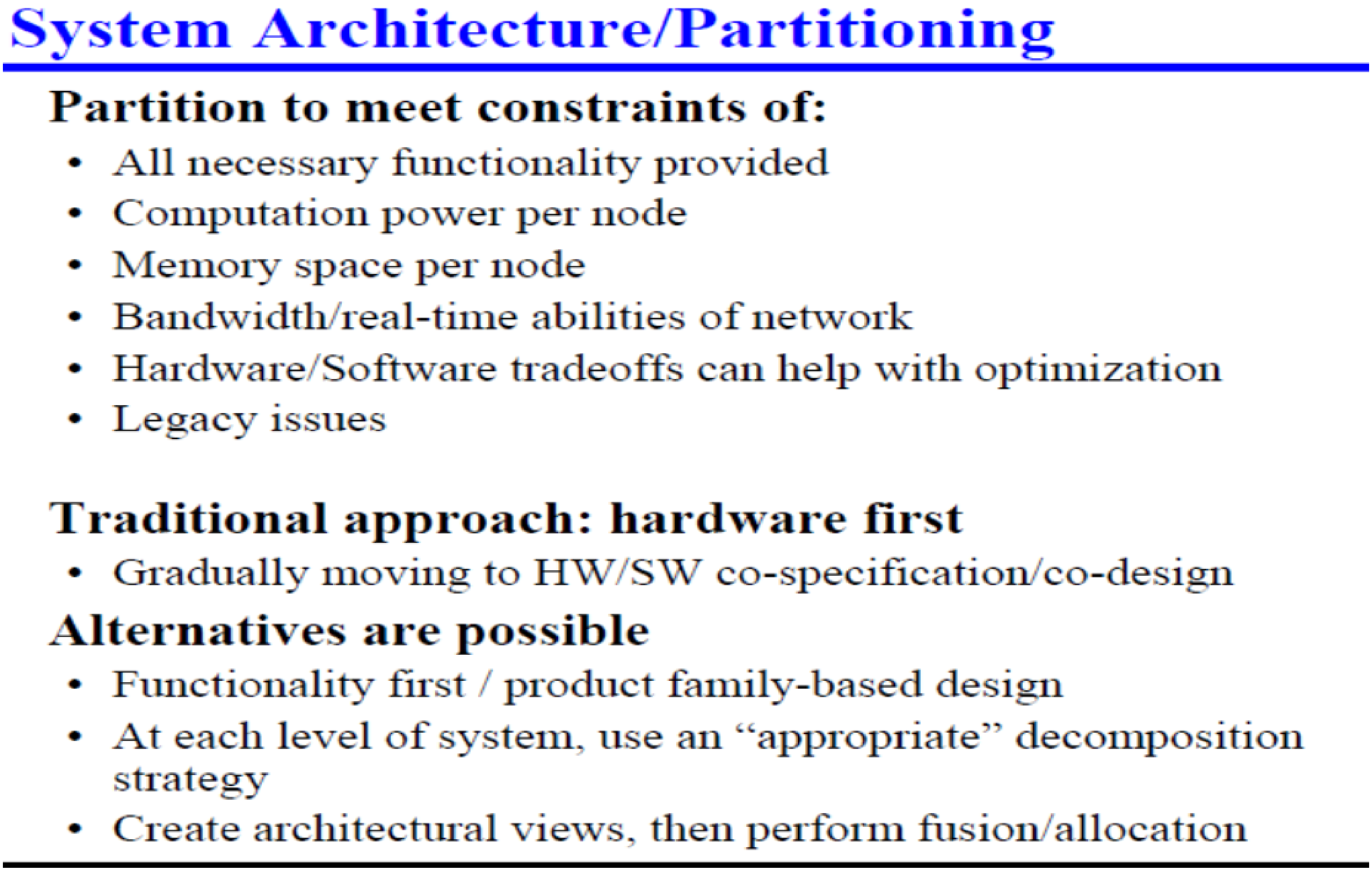
Primary Architectures (almost always used)

- Hardware architecture (CPU, memory, network, I/O)
- Software architecture (software components, data repositories, message dictionary, external interfaces)
- Communication architecture (message flows, message formats)
- Control architecture (hierarchy of control algorithms; emergent system behavior)

Secondary Architectures (used when needed)

- Human interface
- Component coordination & timing framework
- Safety/security
- Validation/verification/testing
- Maintenance/upgrade
- Fault management/graceful degradation
- …

System Architecture/ Partitioning



System Architecture/Partitioning

Partition to meet constraints of:

- All necessary functionality provided
- Computation power per node
- Memory space per node
- Bandwidth/real-time abilities of network
- Hardware/Software tradeoffs can help with optimization
- Legacy issues

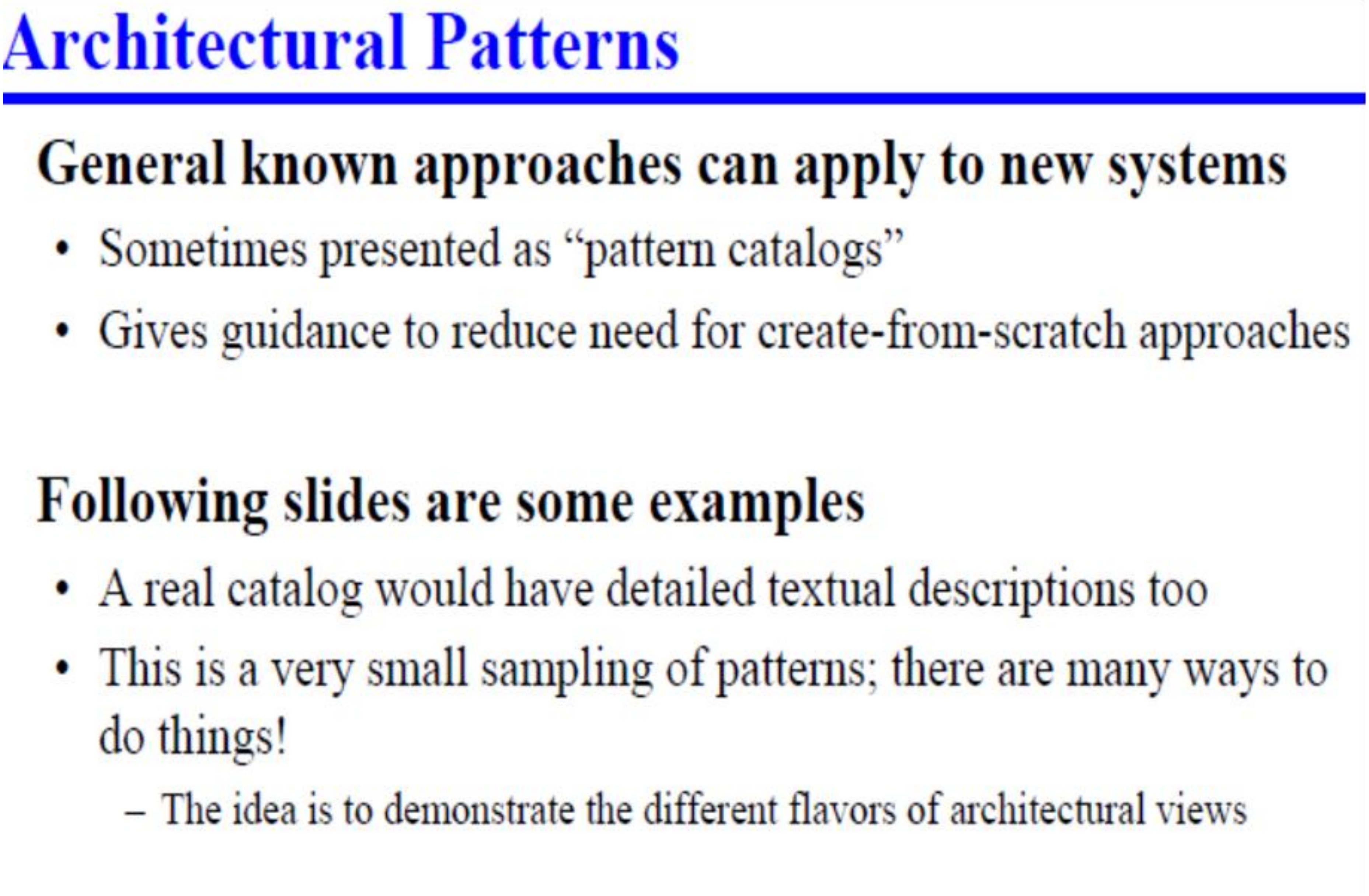
Traditional approach: hardware first

- Gradually moving to HW/SW co-specification/co-design

Alternatives are possible

- Functionality first / product family-based design
- At each level of system, use an "appropriate" decomposition strategy
- Create architectural views, then perform fusion/allocation

Architectural Patterns



Architectural Patterns

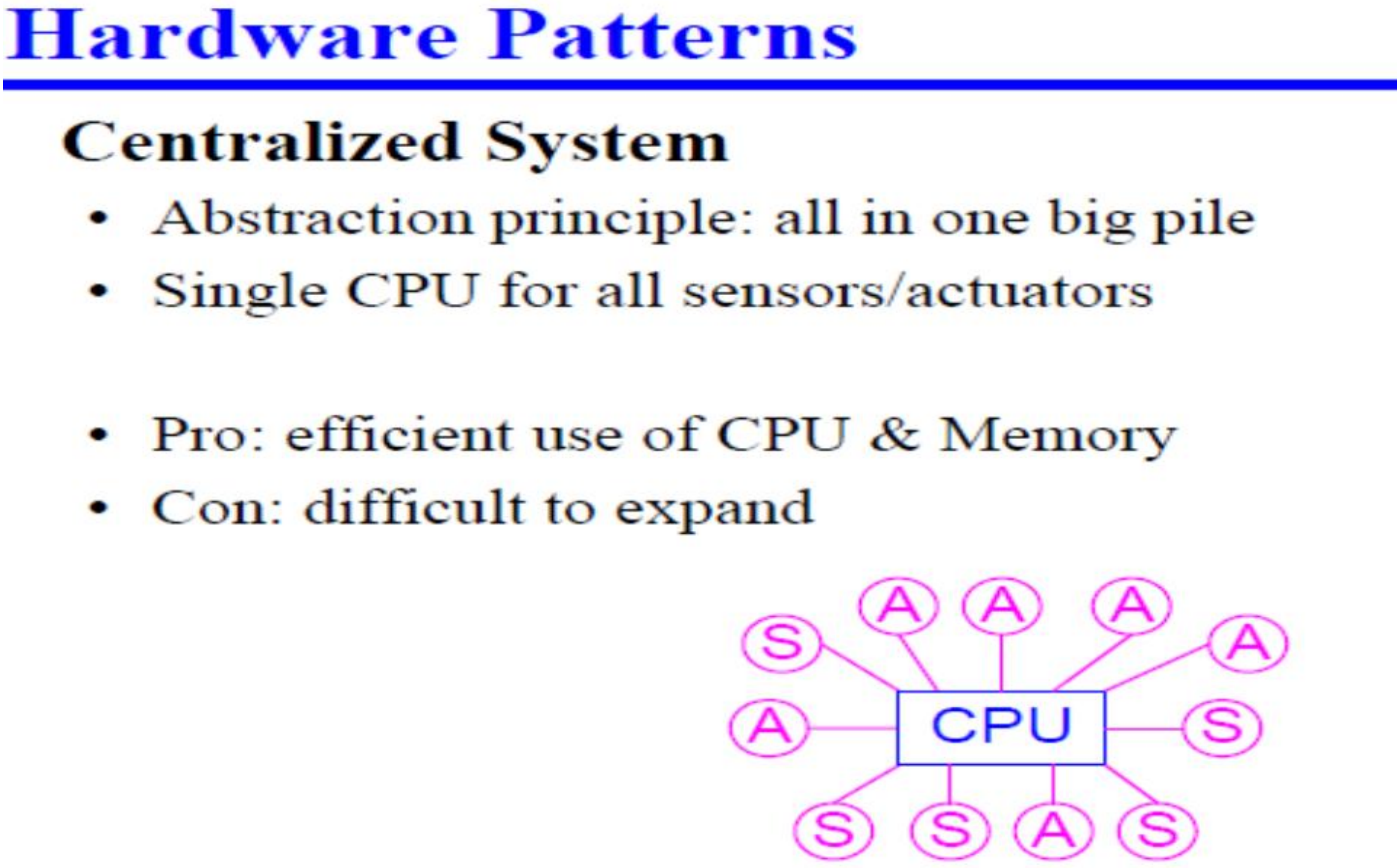
General known approaches can apply to new systems

- Sometimes presented as "pattern catalogs"
- Gives guidance to reduce need for create-from-scratch approaches

Following slides are some examples

- A real catalog would have detailed textual descriptions too
- This is a very small sampling of patterns; there are many ways to do things!
  - The idea is to demonstrate the different flavors of architectural views

Hardware Patterns



Hardware Patterns

Centralized System

- Abstraction principle: all in one big pile
- Single CPU for all sensors/actuators

- Pro: efficient use of CPU & Memory
- Con: difficult to expand

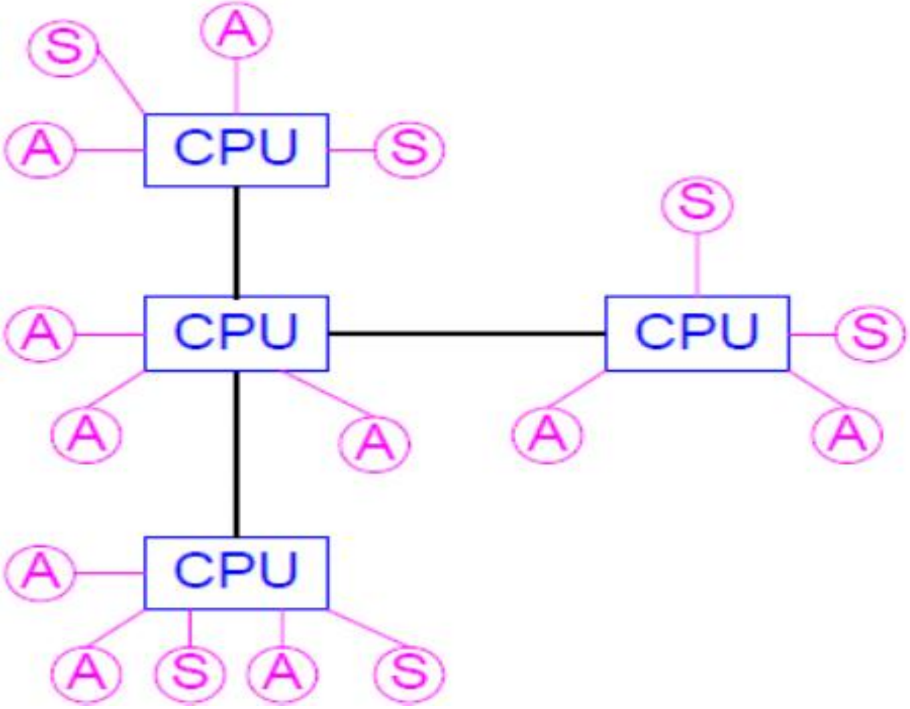
S
A
A
A
A
A
CPU
S
S
S
A
S

Hardware patterns

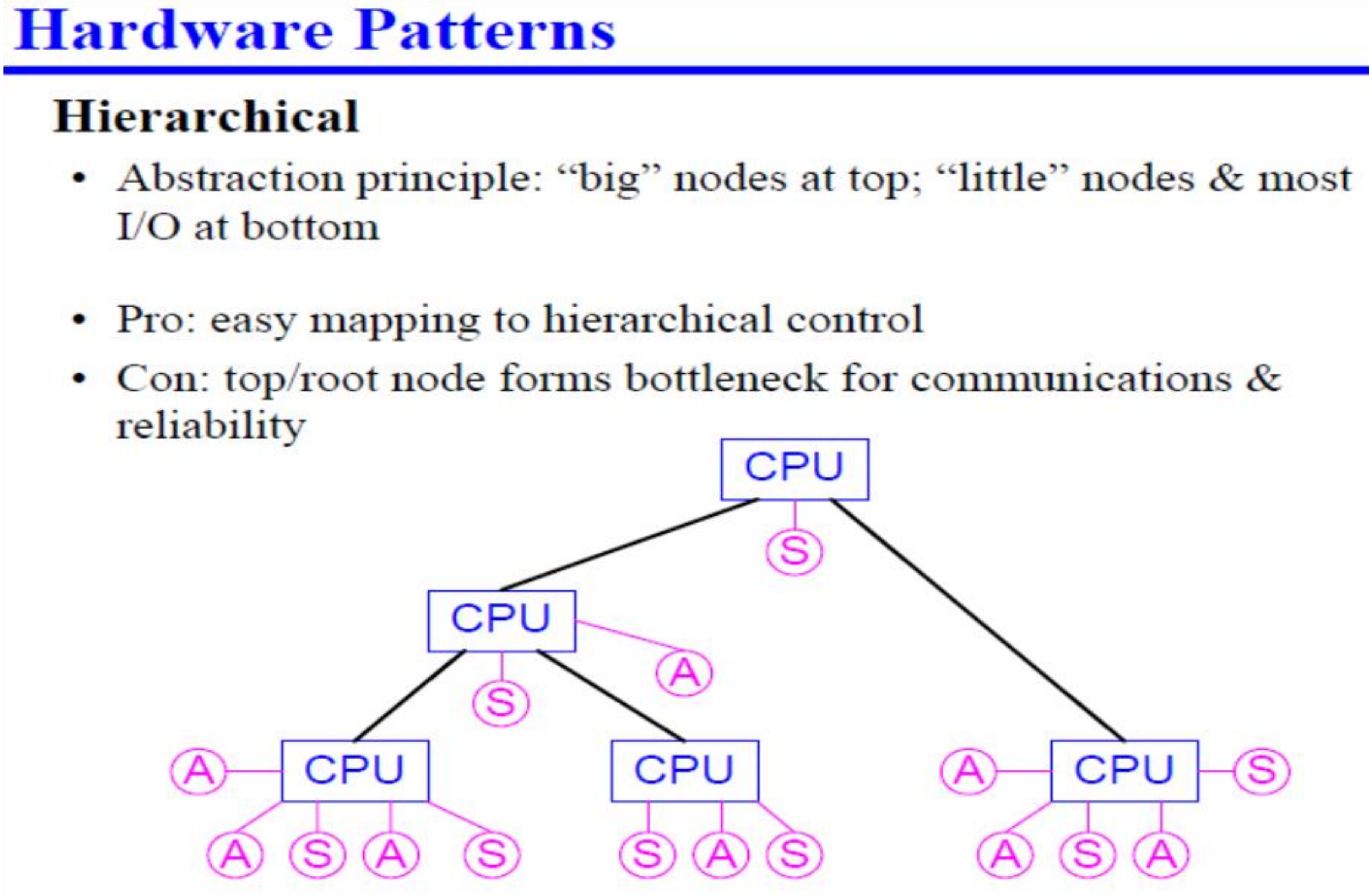
Hardware Patterns

Ad Hoc

- Abstraction principle: paste extra boxes on as system evolves
- Pro: easy way to tack on patches in evolving system
- Con: inefficient mapping of most architectural approaches



Hardware patterns



Hardware Patterns

Hierarchical

- Abstraction principle: “big” nodes at top; “little” nodes & most I/O at bottom

- Pro: easy mapping to hierarchical control
- Con: top/root node forms bottleneck for communications & reliability

CPU
S
CPU
A
S
A
CPU
CPU
A
CPU
S
A
S
A
S
S
A
S
A
S
A

SYSTEM DESIGN

- On system design starts with a philosophical discussion on design in general.

- In computer system design, the most important goal is controlling the complexity of the solution by introducing structure.

- This introduction of structure restricts the design phase and has a negative impact on the performance of the system.

- In the context of real-time systems, these performance penalties must be carefully evaluated.

SYSTEM DESIGN cont……

The architecture design phase starts with analyzing the requirements.

There are two opposing views on how to proceed in this phase:

(i) To complete an unbiased and consistent capture of all requirements before starting the "real" design work, or

(ii) To learn about the requirements by starting a rapid prototype implementation of key system functions at an early stage.

SYSTEM DESIGN cont……

- In any case, the designer must get a deep insight into all the different aspects of the problem domain before she/he can design the application architecture.

- The crucial step is the development of the system structure, the clustering of the functions into nearly decomposable subsystems of high internal cohesion with simple external interfaces.

- In distributed systems, a complete node forms such a subsystem of defined functionality. The node interfaces define the boundaries of the error- containment regions.

SYSTEM DESIGN cont……

Design is an iterative process.

- As more is learned about the problem domain, with different design alternatives being explored, there is the need to start all over again more often than once.

- At the end of the design phase the alternate solutions must be evaluated and compared. TEST OF A DECOMPOSITION- contains checklists that can assist the designer in evaluating a design.

- After the architecture design is completed and frozen, the detailed design and implementation of the node software can be performed by a number of teams in parallel

THE DESIGN PROBLEM

- **Complexity:** Horizontal Structuring, Vertical Structuring.

- **Grand Design versus Incremental Development:** Grand Design, Rapid Prototyping, A Compromise.

- **Legacy Systems**

- **Design Problems are Wicked**

- **REQUIREMENTS ANALYSIS:** Developing Project Standards Information Representation, Naming, Message Interfaces, Documentation, Software Development Tools, Change Control

THE DESIGN PROBLEM cont..

- **Exploring the Constraints:** Minimum Performance Criterion, Dependability Constraints, Cost Constraints.

- **DECOMPOSITION OF A SYSTEM INTO SUBSYSTEMS:** Identification of the Subsystems, The Communication Network Interface, Development of the Message Schedules, Result of the Architecture Design Phase

- **TEST OF A DECOMPOSITION:** Functional Coherence, Testability, Dependability

Physical Characteristics of System design

Physical Characteristics:

- **DETAILED DESIGN AND IMPLEMENTATION:** Definition of the I/O Interfaces, Task Development, Task Scheduling.

- **REAL-TIME ARCHITECTURE PROJECTS:** SPRING, MAFT, FTPP.

Important Reviews of system design

Important reviews –system design

- Design is a creative holistic human activity that cannot be reduced to following a set of rules out of a design rule book.

- Design is an art, supplemented by scientific principles.

- In every project, there is an ongoing conflict between what is desired and what can be done within the given technical and economic constraints.

- A good understanding and documentation of these technical and economic constraints reduces the design space, and helps to avoid exploring unrealistic design alternatives.

System design cont….

- Two kinds of structuring of a computer system can be distinguished to reduce the system complexity: horizontal versus vertical structuring.

- Horizontal structuring (or layering) is related to the process of stepwise abstraction.

- Vertical structuring is related to the process of partitioning a large system into a number of nearly independent subsystems.

- The analysis and understanding of a large problem is never complete and there are always good arguments for asking more questions concerning the requirements before starting with the "real" design work.

System design cont….

- Often it is easier to work on a well-specified detailed side problem than to keep focus on the critical system issues.

- It requires an experienced designer to decide what is a side problem and what is a critical system issue.

- Every requirement must entail an acceptance criterion that allows to measure, at the end of the project, whether the requirement has been met.

- If it is not possible to define a distinct acceptance test for a requirement, then the requirement cannot be very important:

- It can never be decided whether the implementation is meeting this requirement or not.

System design cont….

- The minimum performance criteria establish a borderline between what constitutes success and what constitutes failure.

- A precise specification of the minimum performance, both in the value domain and in the temporal domain is necessary for the design of a fault-tolerant system architecture that does not demand excessive resources.

Dependability

- The dependability constraints of the application are often design drivers.

- A precise specification of the minimal dependability requirements helps to reduce the design space, and guides the designer in finding acceptable technical solutions.

- In the context of distributed real-time systems, a node with an autonomous temporal control can be considered a stable intermediate form.

- The specification of the interface between the nodes and the communication system, the CNI, is thus of critical importance.

System design Features

- The introduction of structure restricts the design space, and may have a negative impact on the performance of a system.

- The key issue is to find the most appropriate structure where the performance penalties are outweighed by the other desirable properties of the structure.

- The allocation of functions to nodes must be guided by the desire to build functional units (nodes) with a high inner connectivity and small external interfaces.

System design Features

- It can be expected that there will be misfits, that some requirements cannot be accommodated in any sensible way.

- It is good practice to challenge these clashing requirements and to reexamine their economic utility.

- Extreme requirements should never drive a design process, and determine an architecture.

- The CNI determines the complexity at the cluster level, and acts as an error detection interface that defines the error-containment regions within a cluster.

- Any error that is not detected at the CNI has the potential to cause a total system failure.

THE TIME-TRIGGERED ARCHITECTURE

Time-triggered architecture (TTA).

- This architecture is being implemented at the Technische Universität Wien with industrial support, taking advantage of the lessons learned during the more than fifteen years of research on dependable distributed real-time systems.

The MARS (Maintainable Real-time System) project

- The MARS (Maintainable Real-time System) project.
- It then gives an overview of the time-triggered architecture (TTA) and emphasizes the essential role of the real-time database in this architecture.
- The building blocks of a TTA prototype implementation are described. The only nonstandard hardware unit is the TTP/C communication controller.
- The TTP/C controller implements all functions of the TTP/C protocol and interfaces to the host via a dualported memory.
- The TTP controller contains ndependent bus guardians to protect the bus against "babbling idiot" failures of the nodes.

MARS project

- Is devoted to the software support tools that are being implemented and planned for the development of software in the TTA.

- The time-triggered operating system that has been developed for MARS has been ported to the TTA host, and adapted to the Communication Network Interface of the TTP controller.

- The generation of the message descriptor lists for the TTP controller is supported by a "cluster compiler".

- The fault-tolerance strategy of the TTA TTA supports the implementation of replicated communication channels and fault-tolerant units consisting of replicated fail-silent nodes, TMR nodes, and other FTU organizations.

LESSONS LEARNED FROM THE MARS PROJECT

- The time-triggered architecture evolved out of the many years of university research centered on the topic of distributed fault-tolerant real-time systems, and carried out in the context of the MARS project.

- **The MARS Project: Project Goals, The MARS Architecture, Building Fail-Silent Nodes**

- **The High Error Detection Coverage Mode (HEDC): Time Redundant Task Execution, End-to-End CR**

TIME-TRIGGERED ARCHITECTURE

- **THE TIME-TRIGGERED ARCHITECTURE:** Economy of Concepts, The Real-Time Database
- **The Hardware Building Blocks:** TTP Controller, TTA-Nodes, Fieldbus Nodes
- **SOFTWARE SUPPORT:** Operating System, Time-Triggered Operating System, Event-Triggered Operating System, interrupts
- **The Cluster Compiler:** Testing
- **FAULT TOLERANCE:** Fault-Tolerant Units, Redundant Sensors
- **WIDE-AREA REAL-TIME SYSTEMS:** The Emergence of ATM Technology, An ATM Gateway

TIME-TRIGGERED ARCHITECTURE

Important Points

- The time-triggered architecture is based on the vision that a node can be built on an inexpensive single chip.

- The system architect is then free to use as many nodes as necessary to implement the given application requirements within a clean functional structure.

- In the TTA a hardware node is considered a unit of failure with a single external failure mode: fail-silence.

- The TTA is based on a small number of orthogonal concepts that are used over again to simplify the understanding of a design.

TIME-TRIGGERED ARCHITECTURE

- The distributed real-time database, formed by the temporally accurate images of all relevant RT entities, is at the core of the time-triggered architecture.

- The real time database contains a temporally valid "snapshot" of the current state of the cluster and the cluster environment.

- In the time-triggered architecture the communication system controls autonomously the exchange of information among the nodes and provides the distributed services for node coordination, such as clock synchronization, membership, and redundancy management.

TIME-TRIGGERED ARCHITECTURE

- The cluster compiler generates the message schedules and tries to make the real time images parametric by selecting appropriate update frequencies.

- At the end it produces the MEDL for each node.

- It is proposed to build wide-area time-triggered real-time systems by making use of the emerging ATM technology.

UNIT- IV

CAN

UNIT - IV

- CANBUS Introduction
  - What is CANBUS?
  - Who uses CANBUS?
  - CANBUS history
  - CANBUS timeline
- CANBUS Characteristics
  - OSI Model
  - Physical Layer
  - Transmission Characteristics
- Message Oriented Communication
- Message Format
- Bus Arbitration

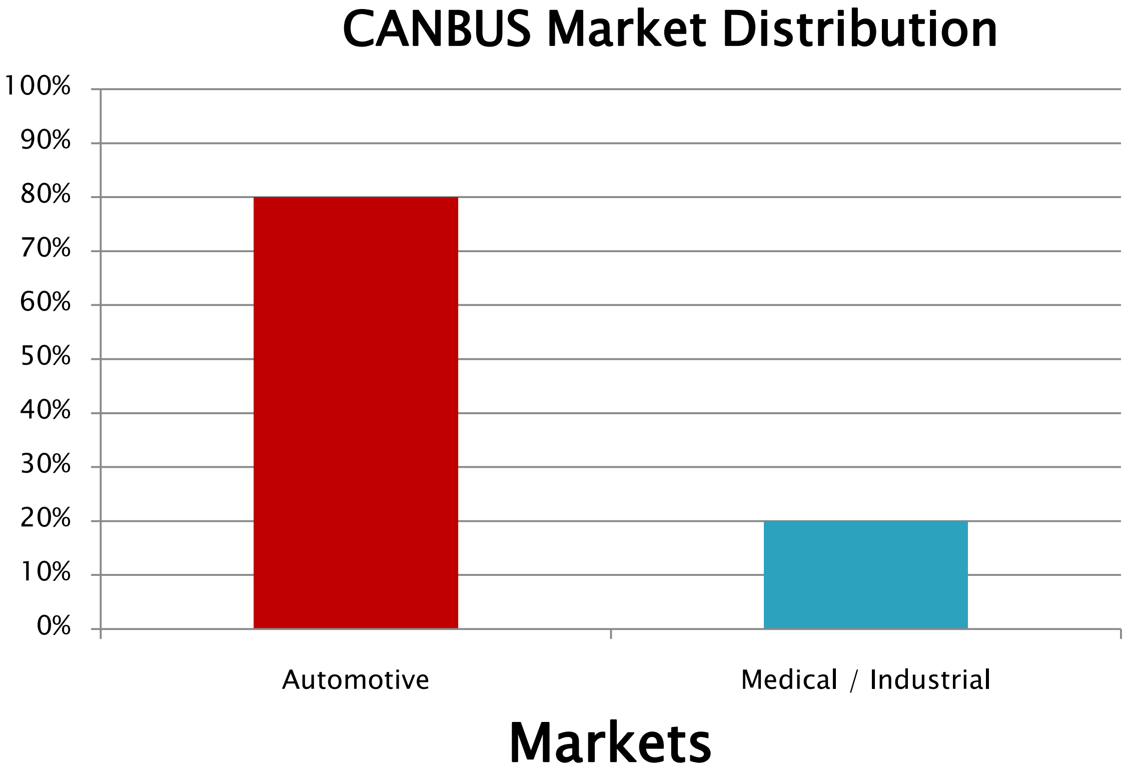
What is CANBUS?

- CANBUS or CAN bus – **C**ontroller **A**rea **N**etwork **bus**

- An automotive serial bus system developed to satisfy the following requirements:
  - ➢ Network multiple microcontrollers with 1 pair of wires.
  - ➢ Allow microcontrollers communicate with each other.
  - ➢ High speed, real-time communication.
  - ➢ Provide noise immunity in an electrically noisy environment.
  - ➢ Low cost

Who uses CANBUS?

- Designed specifically for automotive applications
- Today - industrial automation / medical equipment



History

- First idea - The idea of CAN was first conceived by engineers at Robert Bosch Gmbh in Germany in the early 1980s.

- Early focus - develop a communication system between a number of ECUs (**electronic control units**).

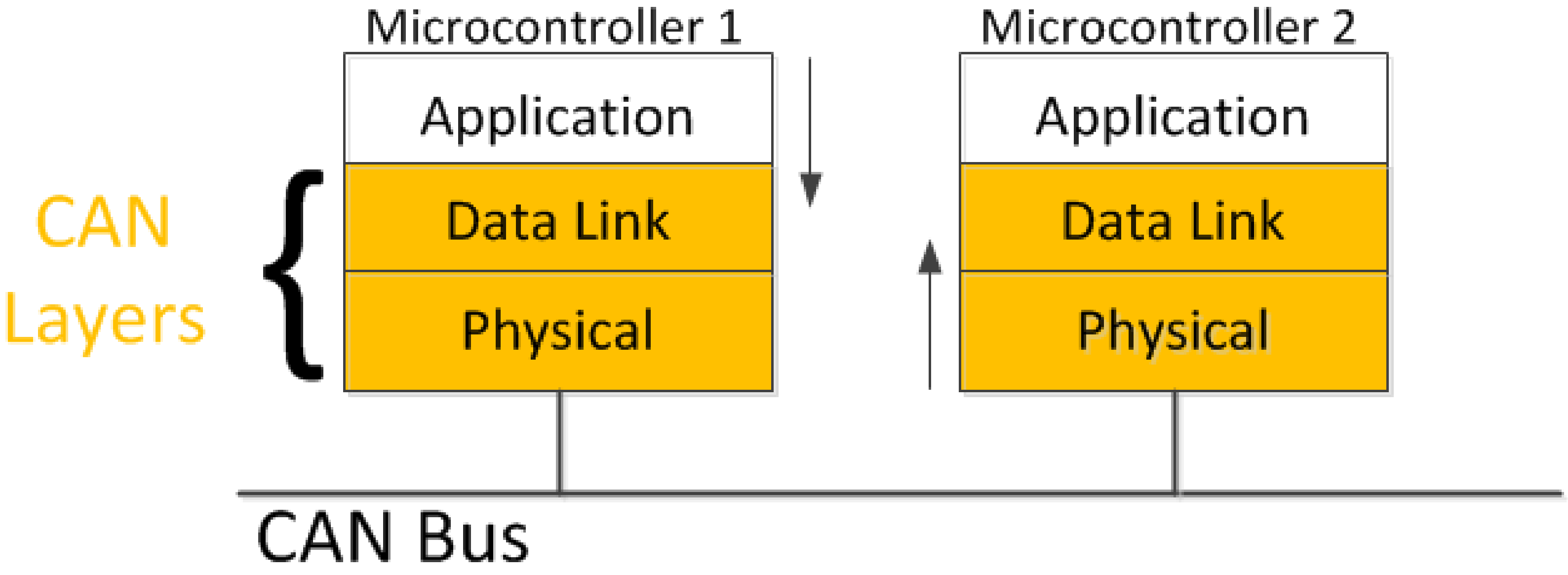
- New standard - none of the communication protocols at that time met the specific requirements for speed and reliability so the engineers developed their own standard.

CANBUS TIMELINE

- 1983 : First CANBUS project at Bosch
- 1986 : CAN protocol introduced
- 1987 : First CAN controller chips sold
- 1991 : CAN 2.0A specification published
- 1992 : Mercedes-Benz used CAN network
- 1993 : ISO 11898 standard
- 1995 : ISO 11898 amendment
- Present : The majority of vehicles use CAN bus.

INTRODUCTION

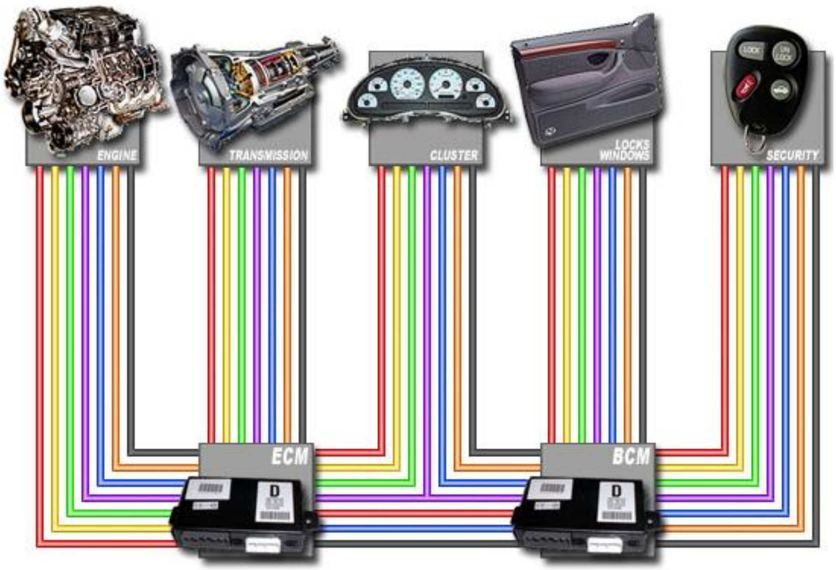
- CAN is a closed network
  - – no need for security, sessions or logins.
  - - no user interface requirements.
- Physical and Data Link layers in silicon.



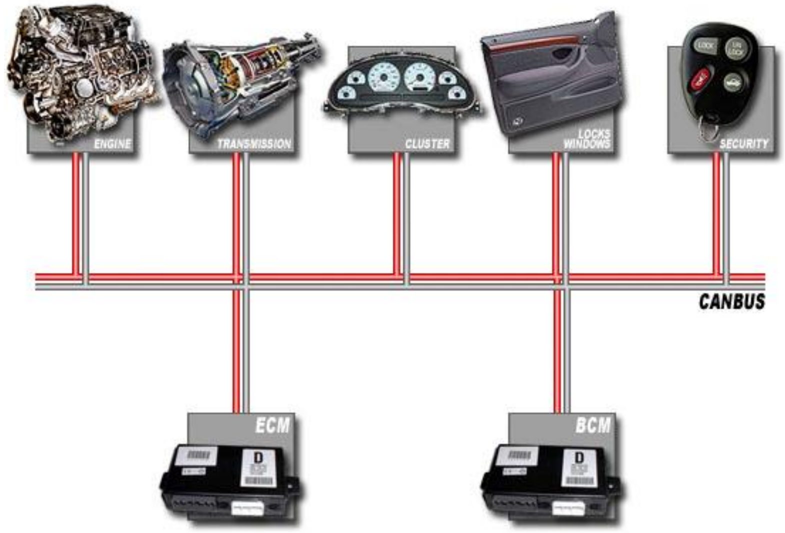
CANBUS and the OSI Model

- Physical medium – two wires terminated at both ends by resistors.
- Differential signal - better noise immunity.
- Benefits:
  - Reduced weight, Reduced cost
  - Fewer wires = Increased reliability

Conventional multi-wire looms

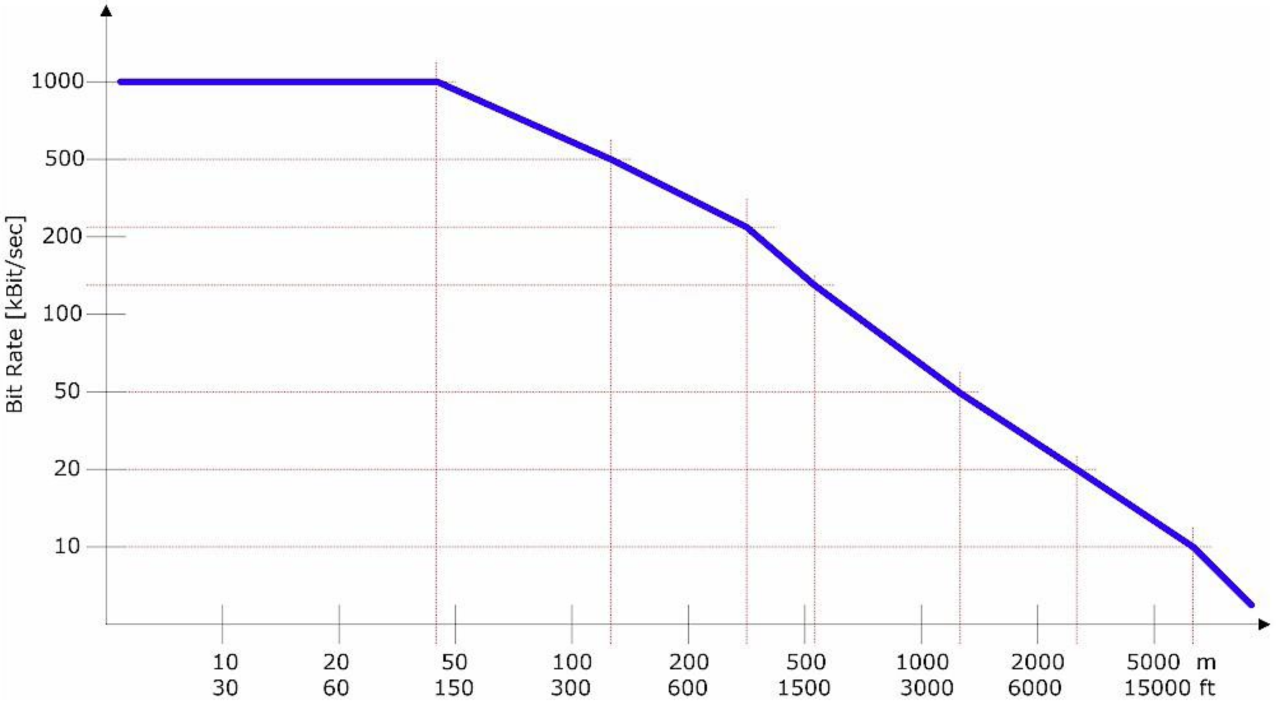


CAN bus network



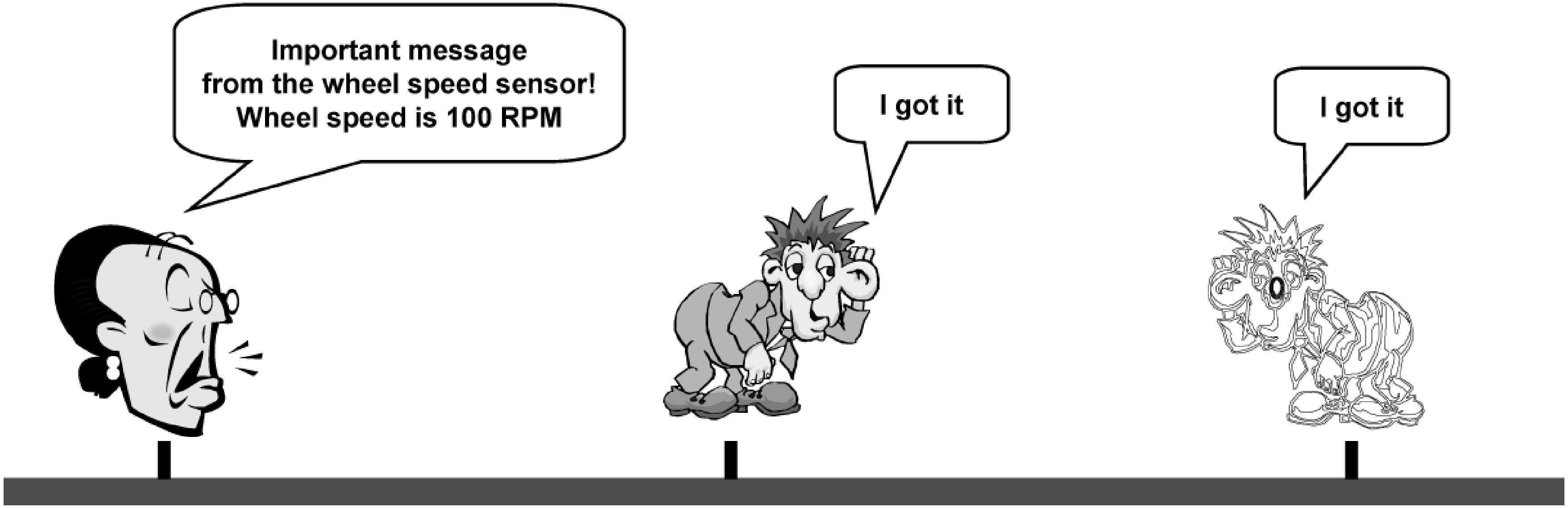
Transmission Characteristics

- Up to 1 Mbit/sec.
- Common baud rates: 1 MHz, 500 KHz and 125 KHz
- All nodes – same baud rate
- Max length:120’ to 15000’ (rate dependent)



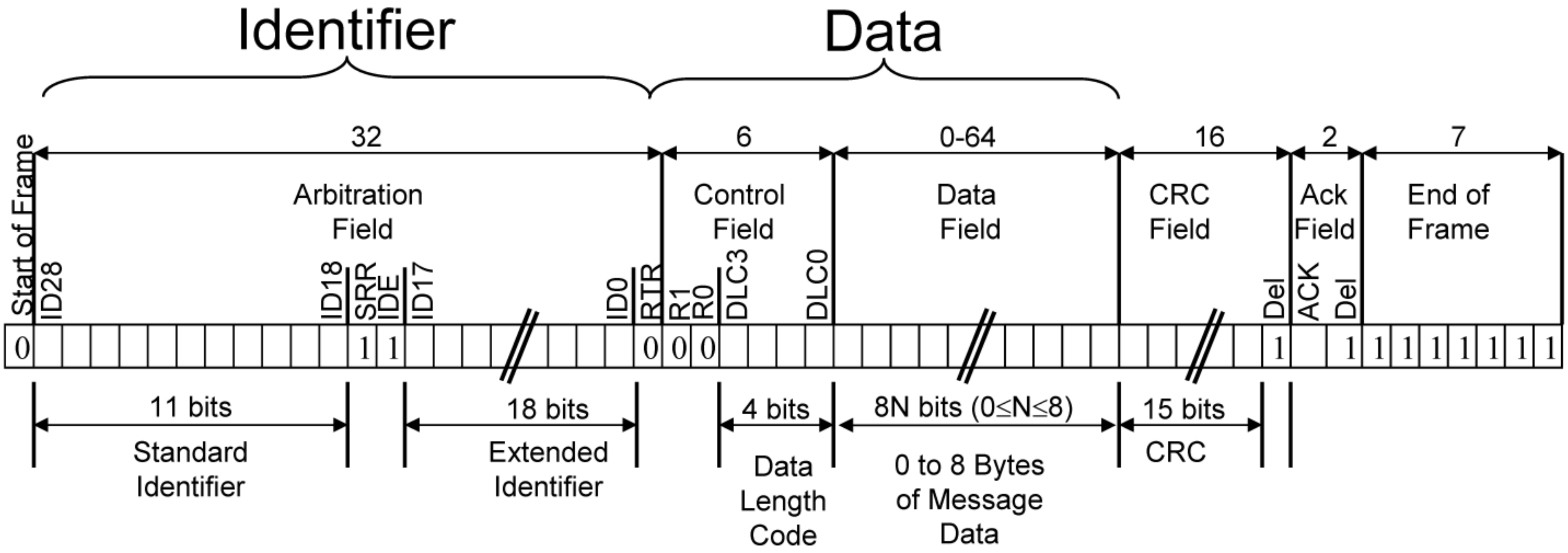
Message Oriented Transmission Protocol

- Each node – receiver & transmitter
- A sender of information transmits to all devices on the bus
- All nodes read message, then decide if it is relevant to them
- All nodes verify reception was error-free
- All nodes acknowledge reception



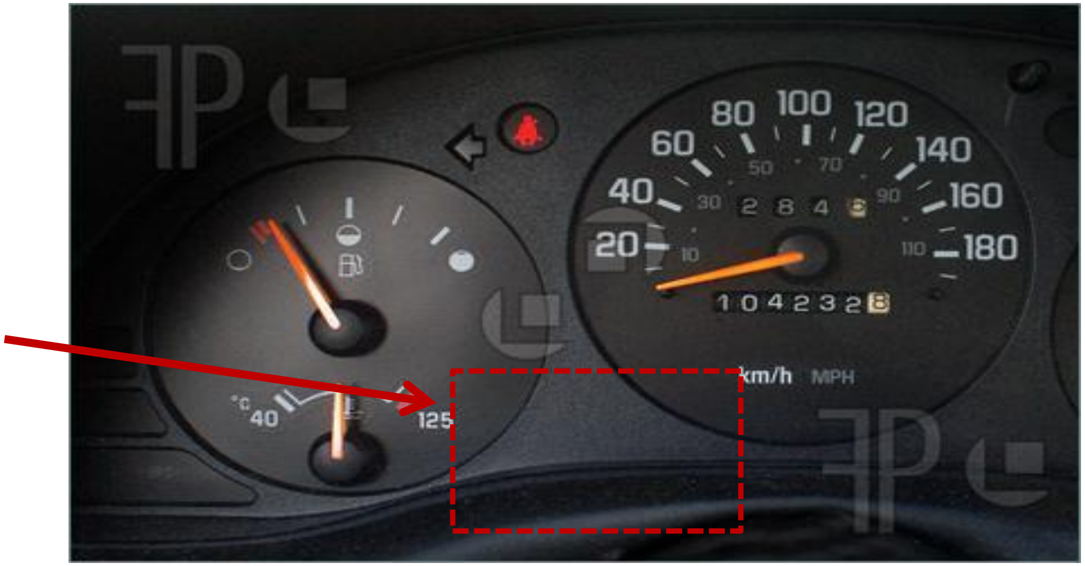
Message Format

- Each message has an ID, Data and overhead.
- Data –8 bytes max
- Overhead – start, end, CRC, ACK



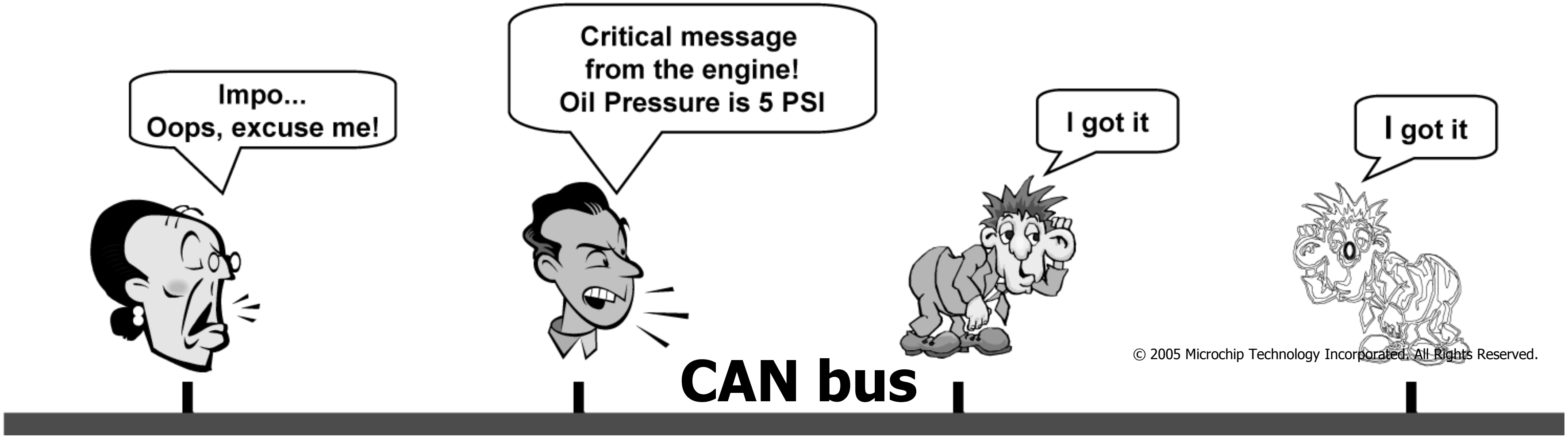
Example of Message Transaction

	ID	Data
Instrument panel ECU says "can anyone tell me what the block temperature is?	400	
Block ECU sees this message and issues a message "block temperature is 76 Celsius"	400	076
Instrument panel ECU sees block temperature message and displays it on console		



Bus Arbitration

- Arbitration – needed when multiple nodes try to transmit at the same time
- Only one transmitter is allowed to transmit at a time.
- A node waits for bus to become idle
- Nodes with more important messages continue transmitting



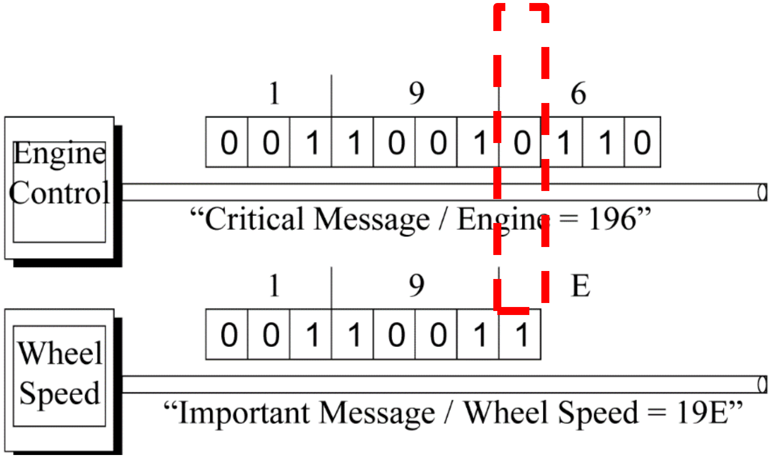
© 2005 Microchip Technology Incorporated. All Rights Reserved.

Bus Arbitration

- Message importance is encoded in message ID.

  Lower value = More important

- As a node transmits each bit, it verifies that it sees the same bit value on the bus that it transmitted.
- A “0” on the bus wins over a “1” on the bus.
- Losing node stops transmitting, winner continues.

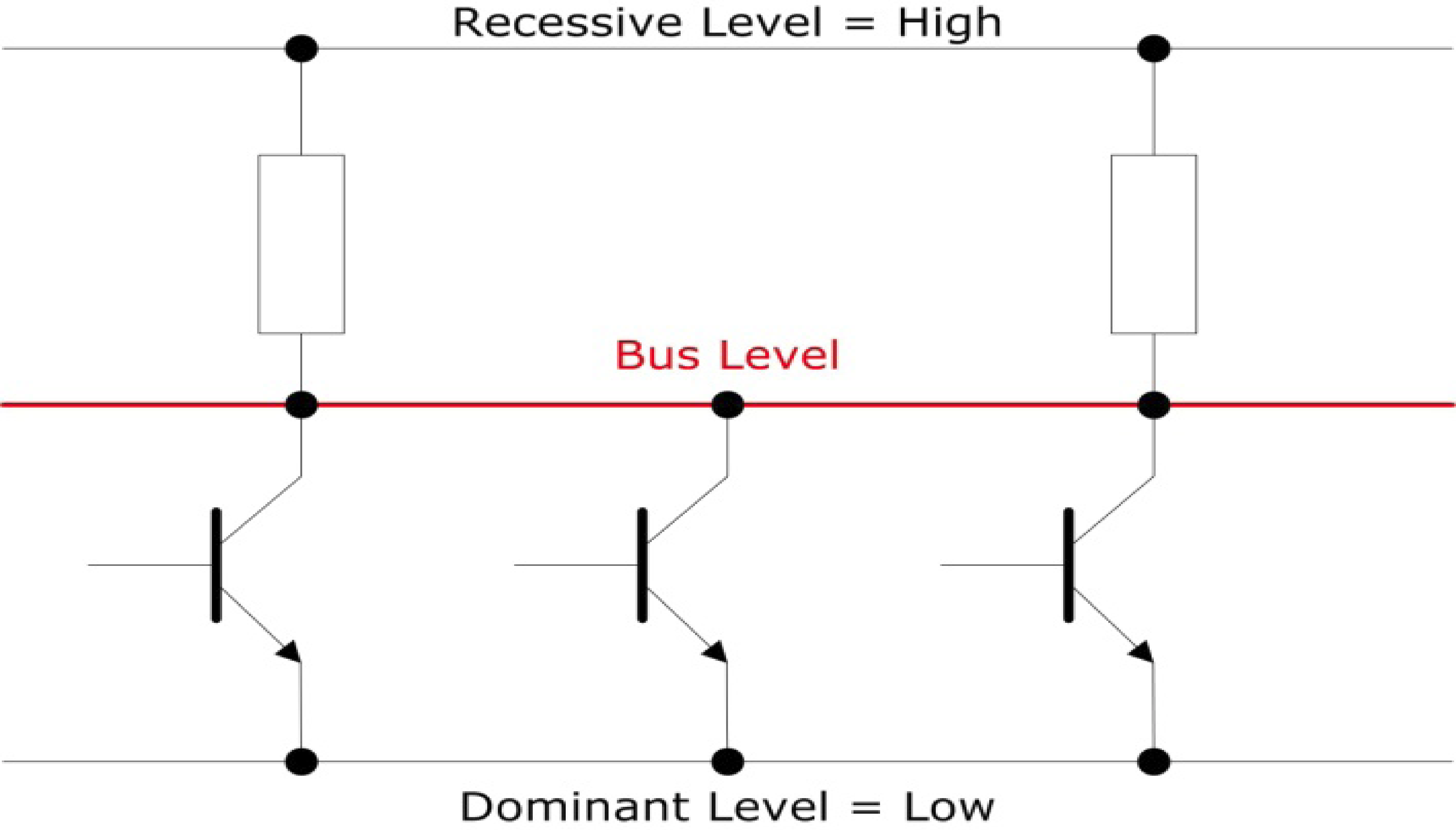

Engine Control
1
9
6
0 0 1 1 0 0 1 0 1 1 0
“Critical Message / Engine = 196”
Wheel Speed
1
9
E
0 0 1 1 0 0 1 1
“Important Message / Wheel Speed = 19E”

Summary

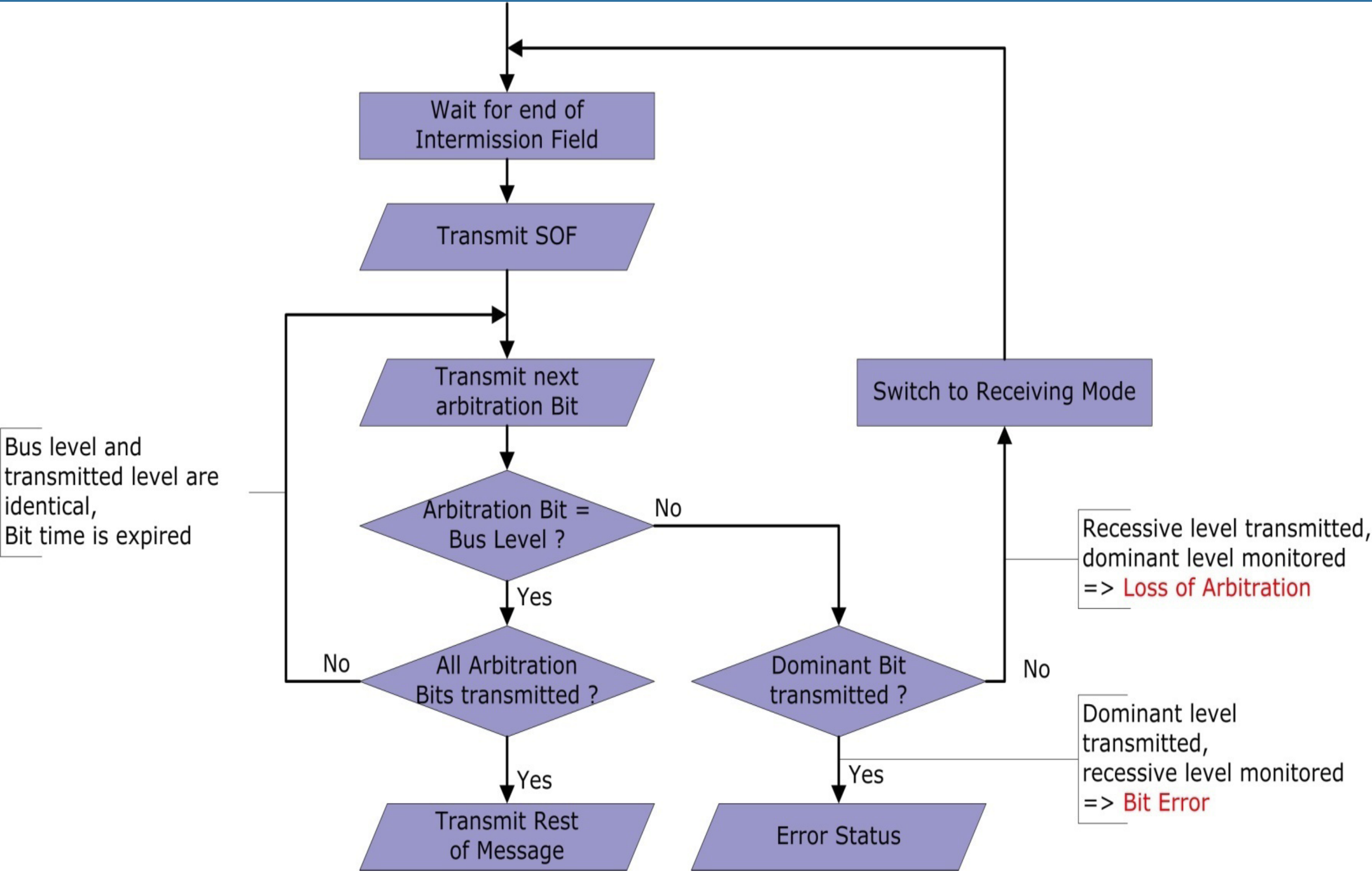
- CAN bus – Controller Area Network bus
- Primarily used for building ECU (electronic control units). Networks in automotive applications.
- Two wires
- OSI - Physical and Data link layers
- Differential signal - noise immunity
- 1Mbit/s, 120
- Messages contain up to 8 bytes of data

Bus arbitration

A “0” (low voltage) on the bus by 1 node wins over a “1” (high voltage) on the bus.



Bus Arbitration Flowchart


Wait for end of Intermission Field
Transmit SOF
Transmit next arbitration Bit
Arbitration Bit = Bus Level ?
No
Yes
All Arbitration Bits transmitted ?
No
Yes
Transmit Rest of Message
Dominant Bit transmitted ?
No
Yes
Error Status
Switch to Receiving Mode

Bus level and transmitted level are identical,
Bit time is expired

Recessive level transmitted,
dominant level monitored
=> Loss of Arbitration

Dominant level transmitted,
recessive level monitored
=> Bit Error

UNIT- V

CAN STANDARDS

UNIT- V

- Configuration files,
- Service data objectives,
- Network management CAN open messages,
- Device profile encoder

INTRODUCTION TO CANOPEN

- The Controller Area Network, commonly known as CAN, was originally designed for use in automobiles.

- By virtue of its massive adoption by automakers worldwide,

- low-cost microcontrollers with CAN controller interfaces are available from over twenty manufacturers, making CAN a mainstream network technology

INTRODUCTION TO CANOPEN

Moreover,

- CAN has migrated into many non-automobile applications over the last ten years creating a requirement for an open,
- Standardized higher-layer protocol that provides a reliable message exchange system along with a means to detect,
- Configure and operate nodes
- Several higher-layer CAN protocols emerged such as SAE J1939, DeviceNet and CANopen.

INTRODUCTION TO CANOPEN

- While each protocol has its own special purpose, CANopen is the most popular higher-layer protocol for embedded networking applications .
- Those networks that are completely hidden within a machine cell and is found in over twenty vertical markets such as,
- Transportation, medical, industrial machinery, building automation and military, just to name a few.

History of CAN and CANopen

- In February of 1986, Robert Bosch introduced the CAN (Controller Area Network) serial bus system at the SAE congress in Detroit.
- In mid-1987, Intel delivered the first stand-alone CAN controller chip, the 82526. Shortly thereafter, Philips Semiconductors introduced the 82C200.
- Today, almost every new passenger car manufactured in Europe is equipped with at least one CAN network. Also used in other types of vehicles, from trains to ships, as well as in industrial controls, CAN is one of the most dominating bus protocols.
- To date, chip manufacturers have produced and sold more than 500 million CAN devices in total.

History of CAN and CANopen

- CAN was originally developed to be used in passenger cars,
- the first applications came from other market segments.
- Especially in northern Europe,
- CAN was already very popular even in its early days.
- At the beginning of 1992, users and manufacturers established the CAN in Automation (CiA) international users and manufacturers association.

History of CAN and CANopen

- Although the CAL approach was academically correct and it was possible to use it in industrial applications, every user needed to design a new profile because CAL was a true application layer.
- Since 1993 and within the scope of the Esprit project ASPIC,
- a European consortium led by Bosch had been developing a prototype of what would become CANopen, the CAL-based profile for embedded networking in production cells.

Cont….

- In 1995, CiA released the completely revised CANopen communications profile.
- The CANopen profile family defines a framework for programmable systems as well as different device,
- Interface and application profiles. This is an important reason why whole industry segments
- (e.g. printing machines, maritime applications, medical systems, etc.) decided to use CANopen during the late 1990s.

Cont….

- In the early 1990s, engineers at the US mechanical engineering company Cincinnati Milacron started a joint venture together with Allen-Bradley and Honeywell Microswitch regarding a control and communications project based on CAN.

- However, after a short while important project members changed jobs and the joint venture fell apart.

- But Allen-Bradley and Honeywell continued the work separately.

Cont….

- This led to the two higher layer protocols ‘DeviceNet’ and ‘Smart Distributed System’ (SDS), which are quite similar, at least in the lower communication layers.

- In early 1994, Allen-Bradley turned the DeviceNet specification over to the Open DeviceNet Vendor Association (ODVA), which boosted the popularity of DeviceNet.

- Honeywell failed to go a similar way with SDS, which makes SDS look more like an internal solution by Honeywell Microswitch.

Cont….

- DeviceNet was developed especially for factory automation and therefore presents itself as a direct opponent to protocols like Profibus-DP and Interbus.

- Providing off-the-shelf plug-and-play functionality, DeviceNet has become the leading bus system in this particular market segment in the US.

Cont....

- With DeviceNet and CANopen, two standardized (EN 50325) application layers are now available, addressing different markets.

- DeviceNet is optimized for factory automation and CANopen is especially well suited for embedded networks in all kinds of machine controls.

- This has made proprietary application layers obsolete; the necessity to define application-specific application layers is history (except, perhaps, for some specialized high-volume embedded systems).

Cont….

- Of course, the more than 50 semiconductor vendors who have implemented CAN modules into their micro-controllers and ASICs are mainly focused on the automotive industry.

- Since the mid-1990s, Infineon Technologies (formerly Siemens) and Motorola have shipped large quantities of CAN controllers to European passenger car manufacturers.

- As a next wave, Far Eastern semiconductor vendors have also offered CAN controllers since the late 1990s.

Cont….

- Since 1992, Mercedes-Benz has been using CAN in their high-end passenger cars.

- Now nearly all new European passenger cars are equipped with several networks, with some high-end cars implementing up to five CAN networks.

- Although the CAN protocol is now 15 years old, it is still being enhanced.

- In the last two years an ISO task force defined a protocol for a time-triggered transmission of CAN messages.

Cont….

- The TTCAN extension will add about five to ten years to the lifetime of CAN.

- Considering CAN is still at the beginning of a global market penetration, even conservative estimates show further growth for this bus system for the next ten to fifteen years.

- This is underlined by the fact that the US and Far Eastern car manufacturers are just starting to use CAN in the production of their vehicles.

Cont….

- Furthermore, new potentially high-volume applications are in the pipeline – not only in passenger cars but also entertainment, domestic appliances and automatic building doors, among many others
- Several enhancements regarding the approval for different safety-relevant and safety-critical applications can be expected for the higher-layer protocols (HLP).

Cont….

- The German professional association BIA and the German safety standards authority TÜV have already certified some of the proprietary CAN-based safety systems.
- CANopen-Safety and DeviceNet Safety are the first standardized CAN solutions to earn a tentative TÜV approval.
- Approval of the CANopen framework for maritime applications by one of the leading classification societies worldwide, Germanischer Lloyd, is in preparation.
- Among other things, this specification defines the automatic switchover from a CANopen network to a redundant bus system.

Cont….

- In the future, CiA members will define several CANopen application profiles.
- An application profile specifies all device interfaces used in a specific application.
- This includes direct communication between dedicated devices overcoming the master/slave PDO communication as usual in standard device profiles.
- The first CANopen application profiles will be for automatic building doors, lift control systems, road construction machinery and light railways.

Communication in the Automation Pyramid

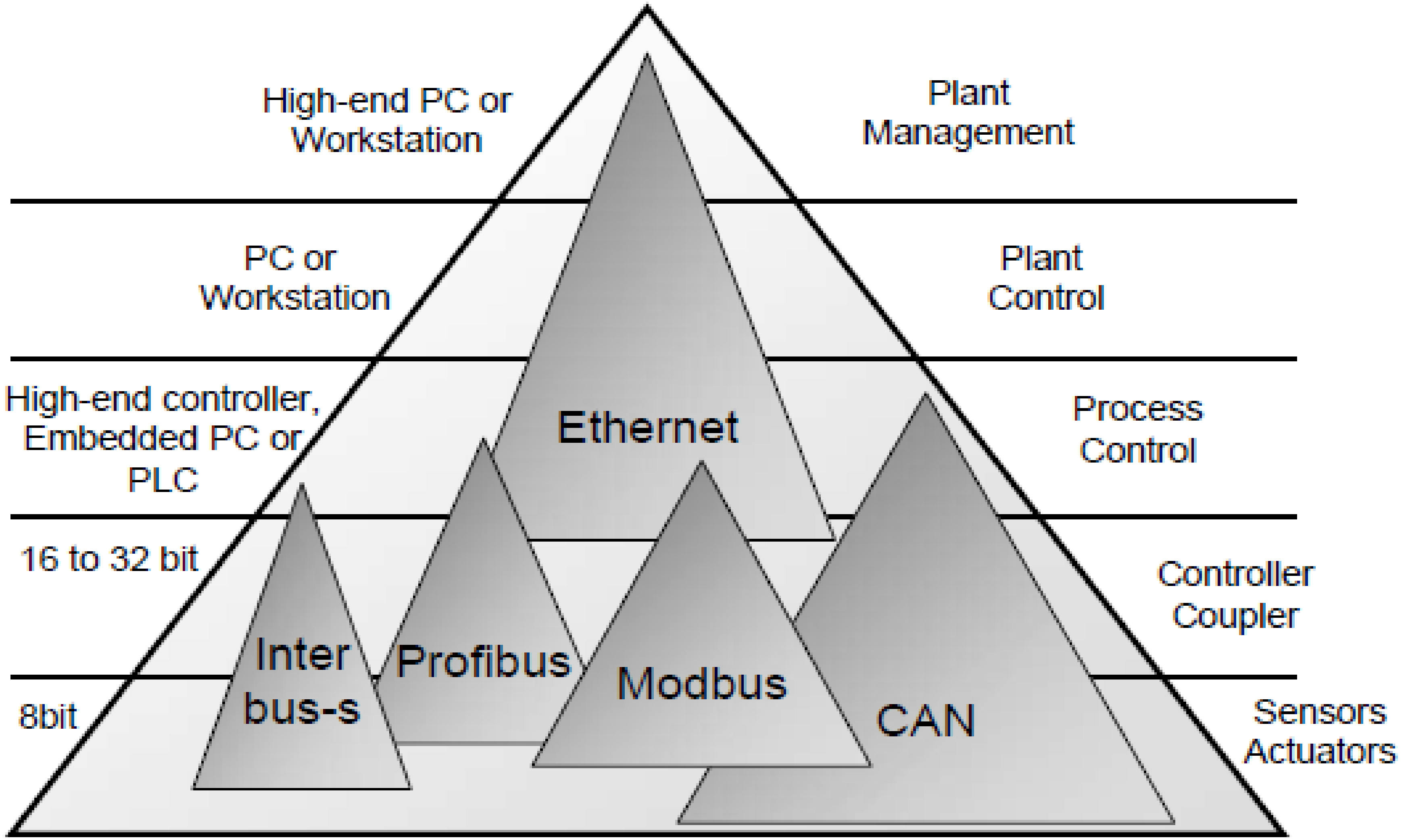


Figure 1.1 Communication in the Automation Pyramid

Inputs and Outputs- Traditional

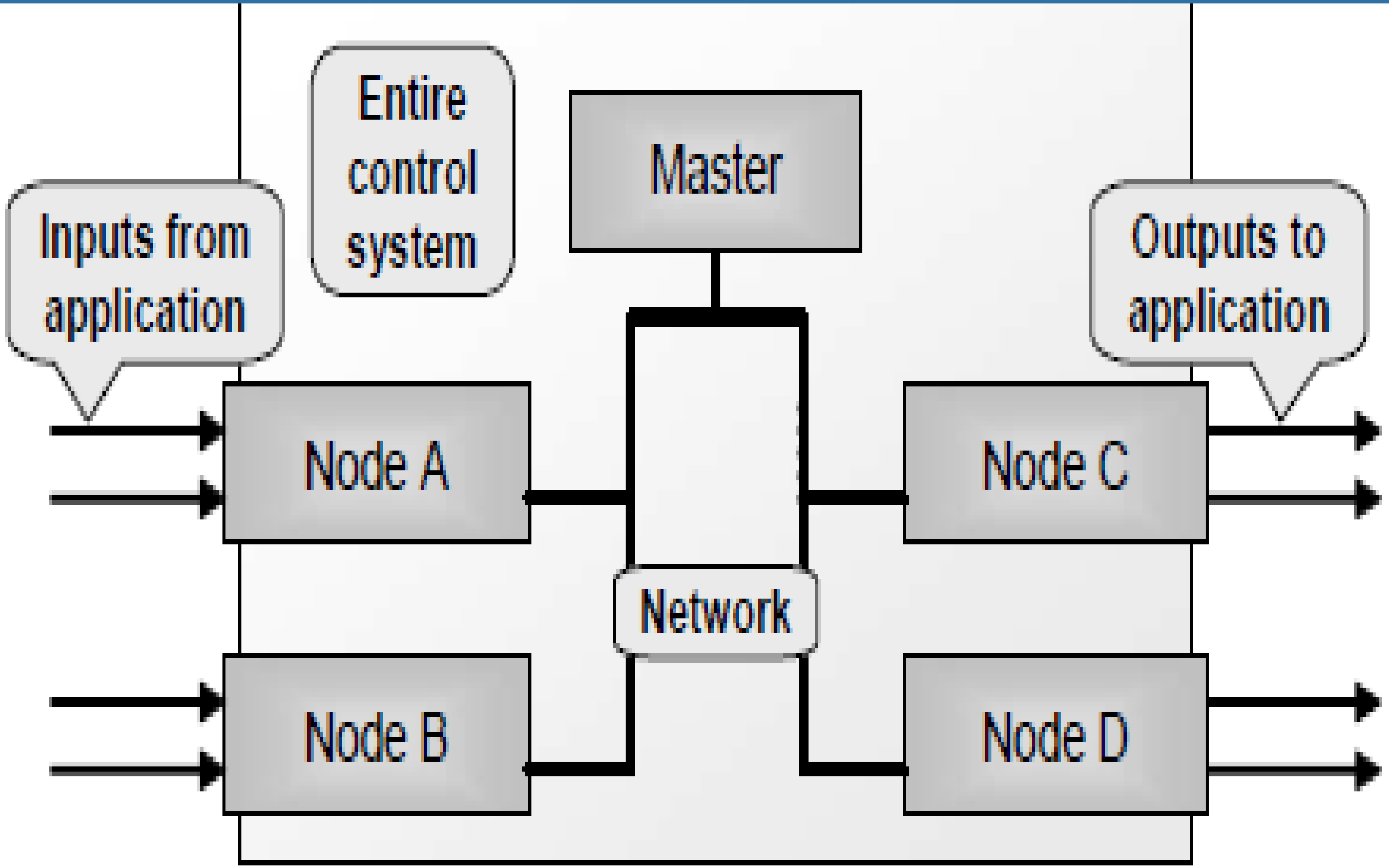


Figure 1.2 Inputs and Outputs - Traditional

Inputs and Outputs - Embedded

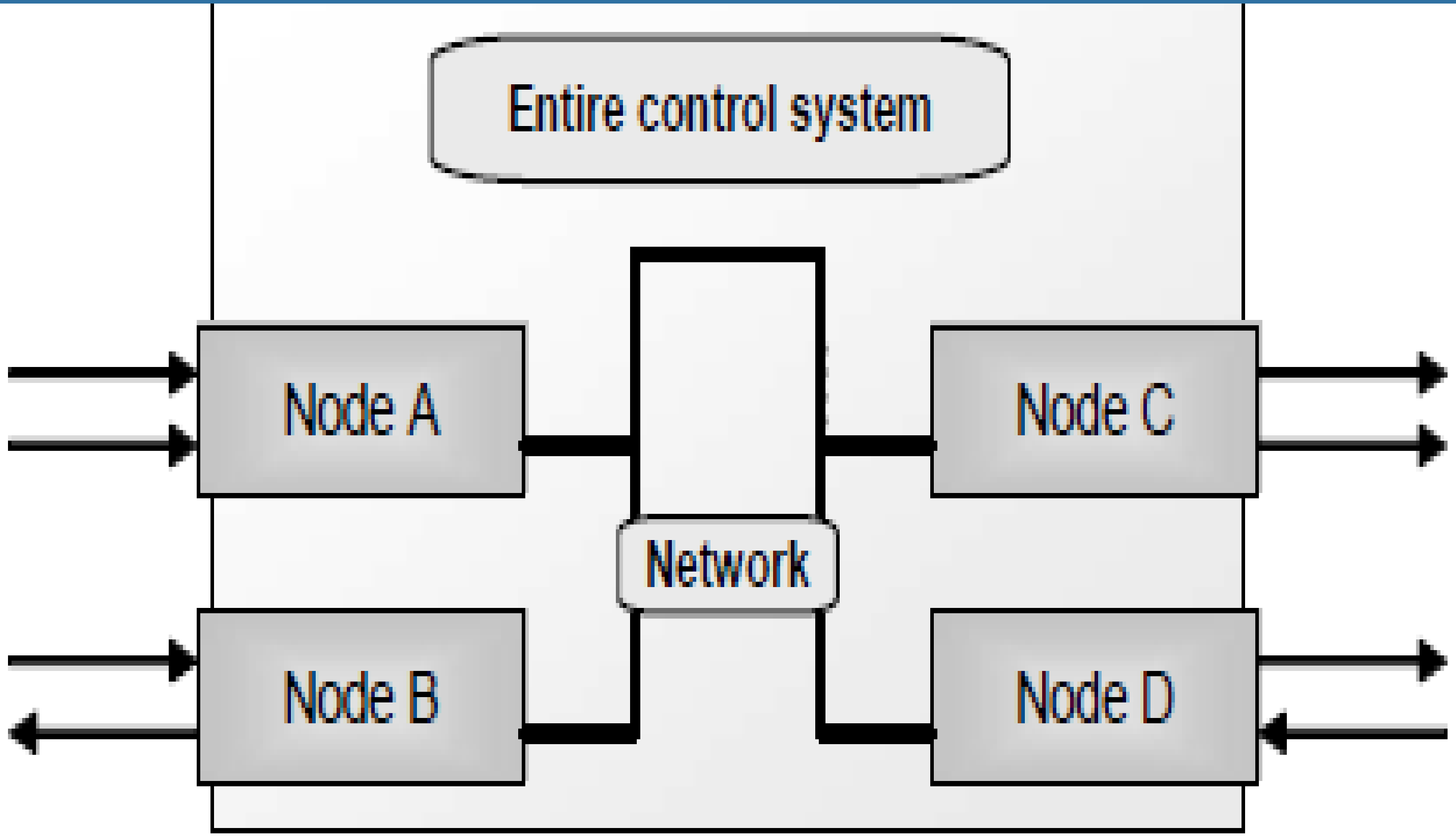


Figure 1.3 Inputs and Outputs - Embedded

Master/ Slave Communication Model

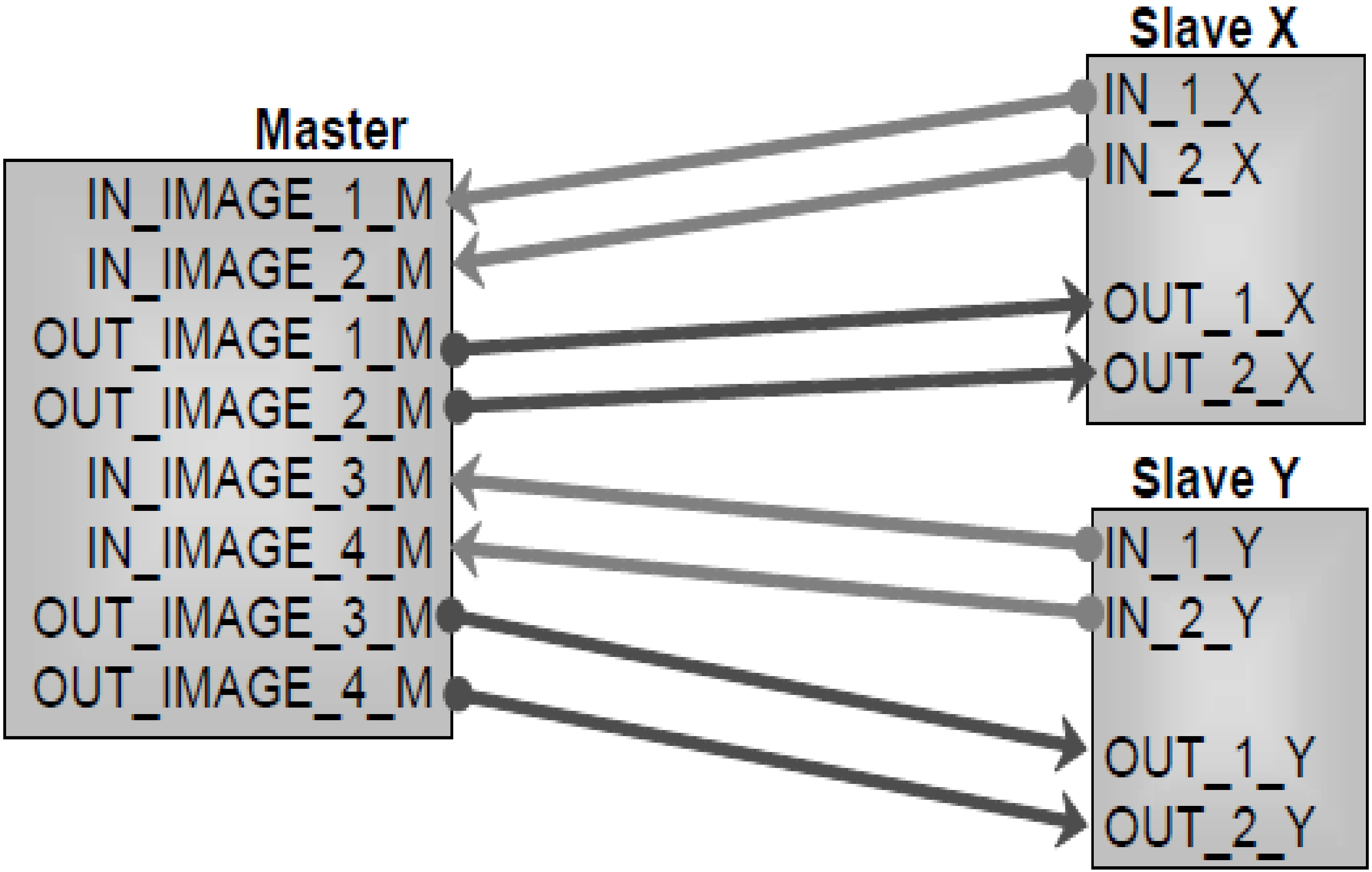


Figure 1.4 The Master/Slave Communication Model

Direct Communication Model

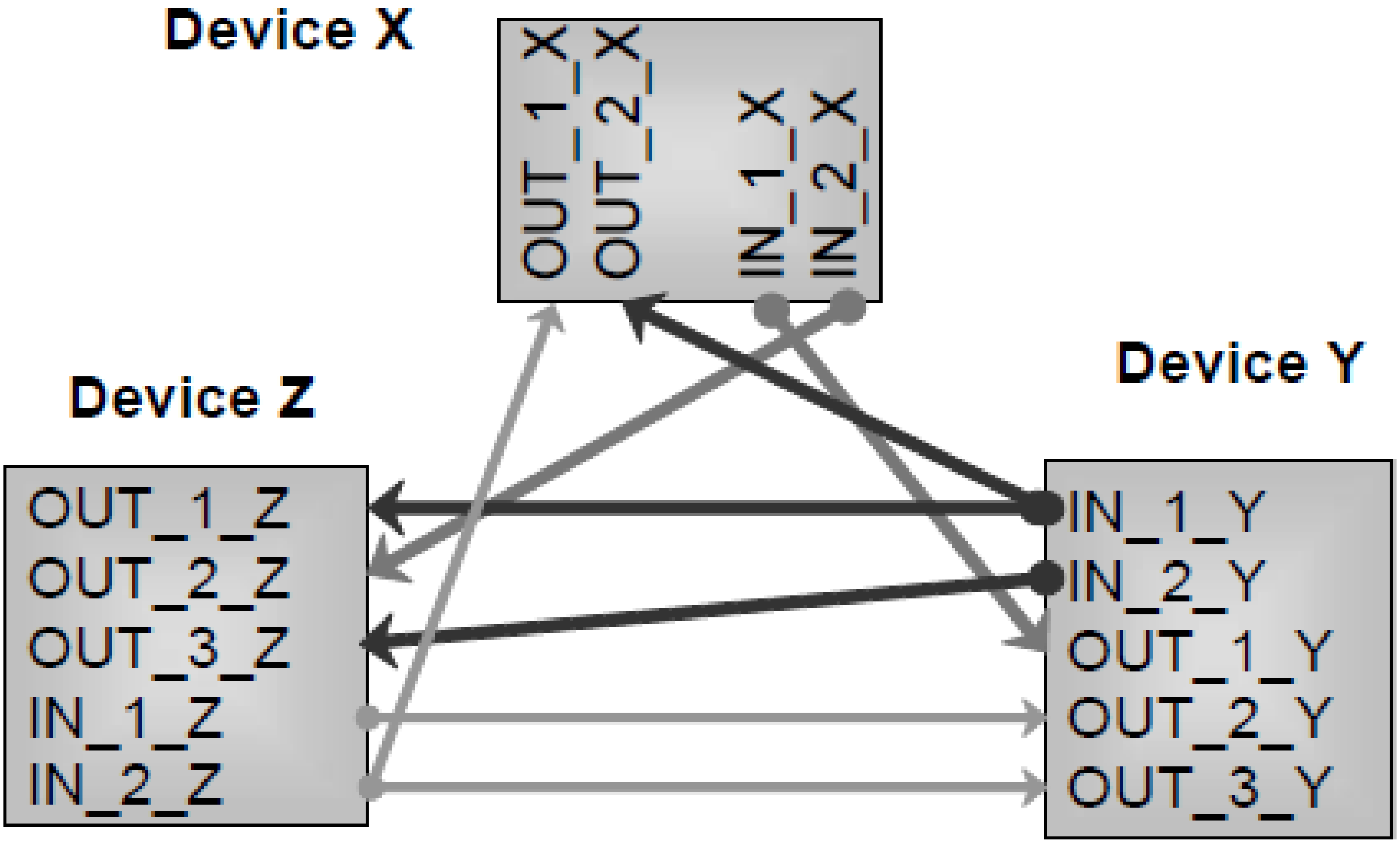


Figure 1.5 Direct Communication Model

Sample Network Layout

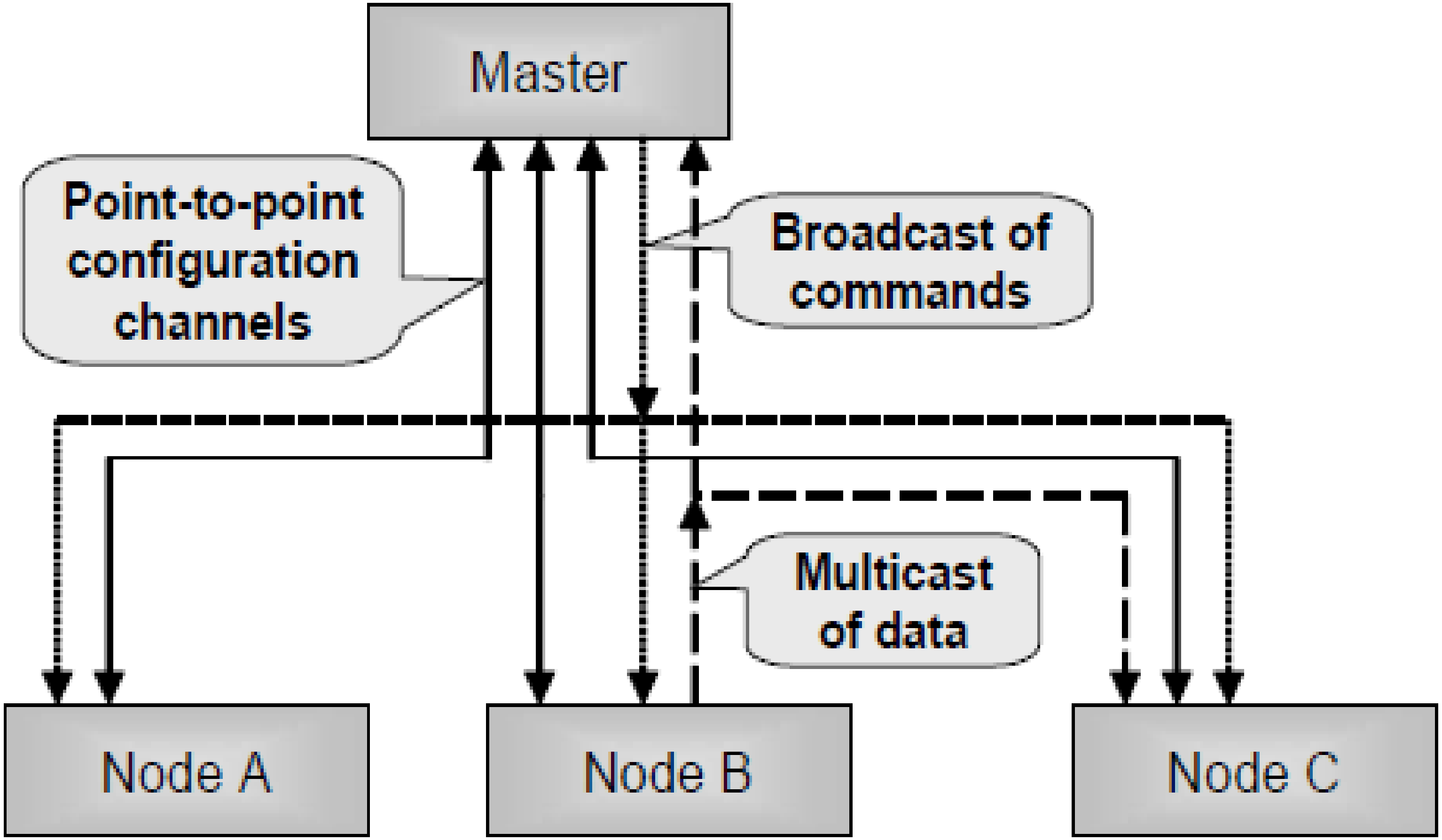


Figure 1.6 Sample Network Layout

OSI MODEL

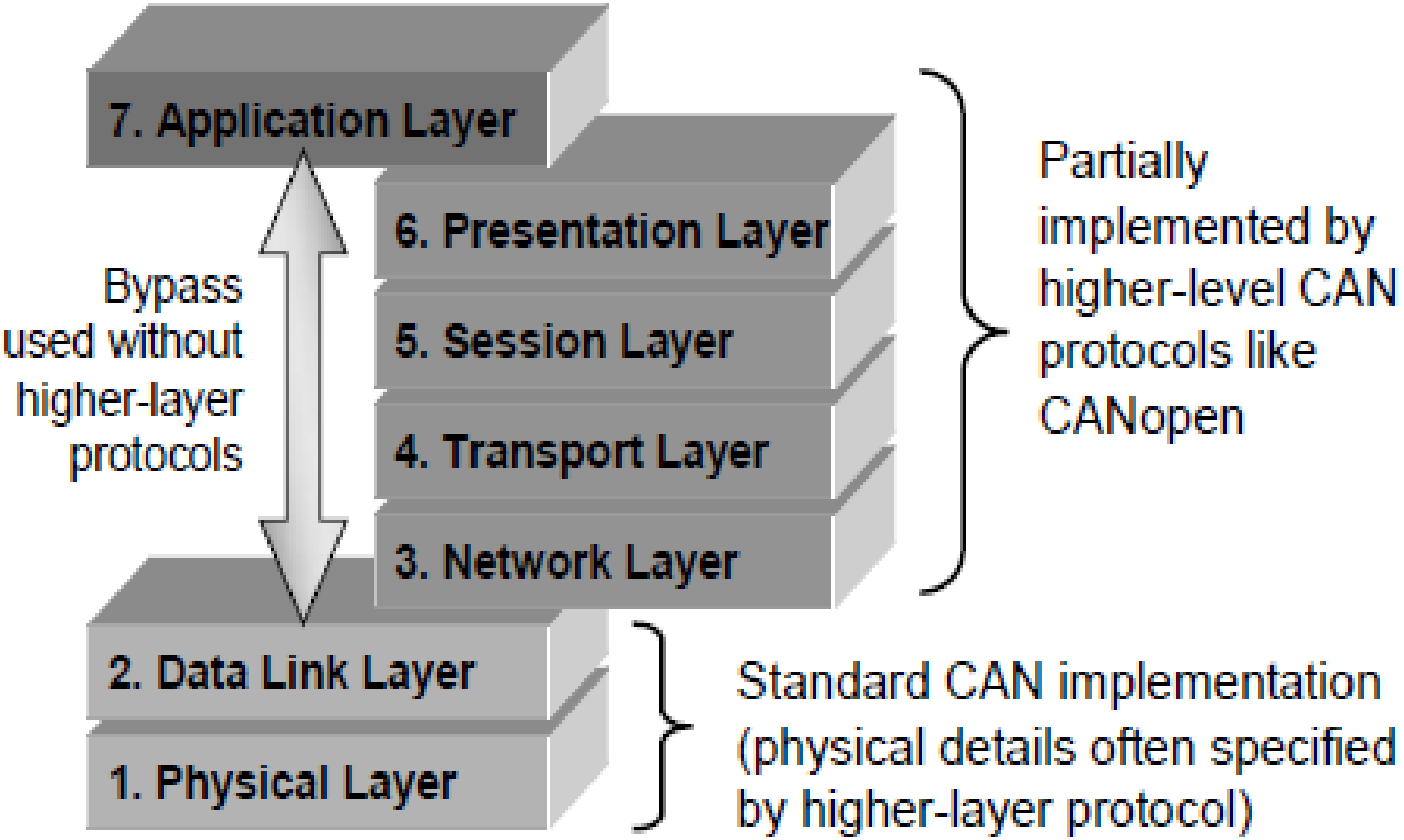
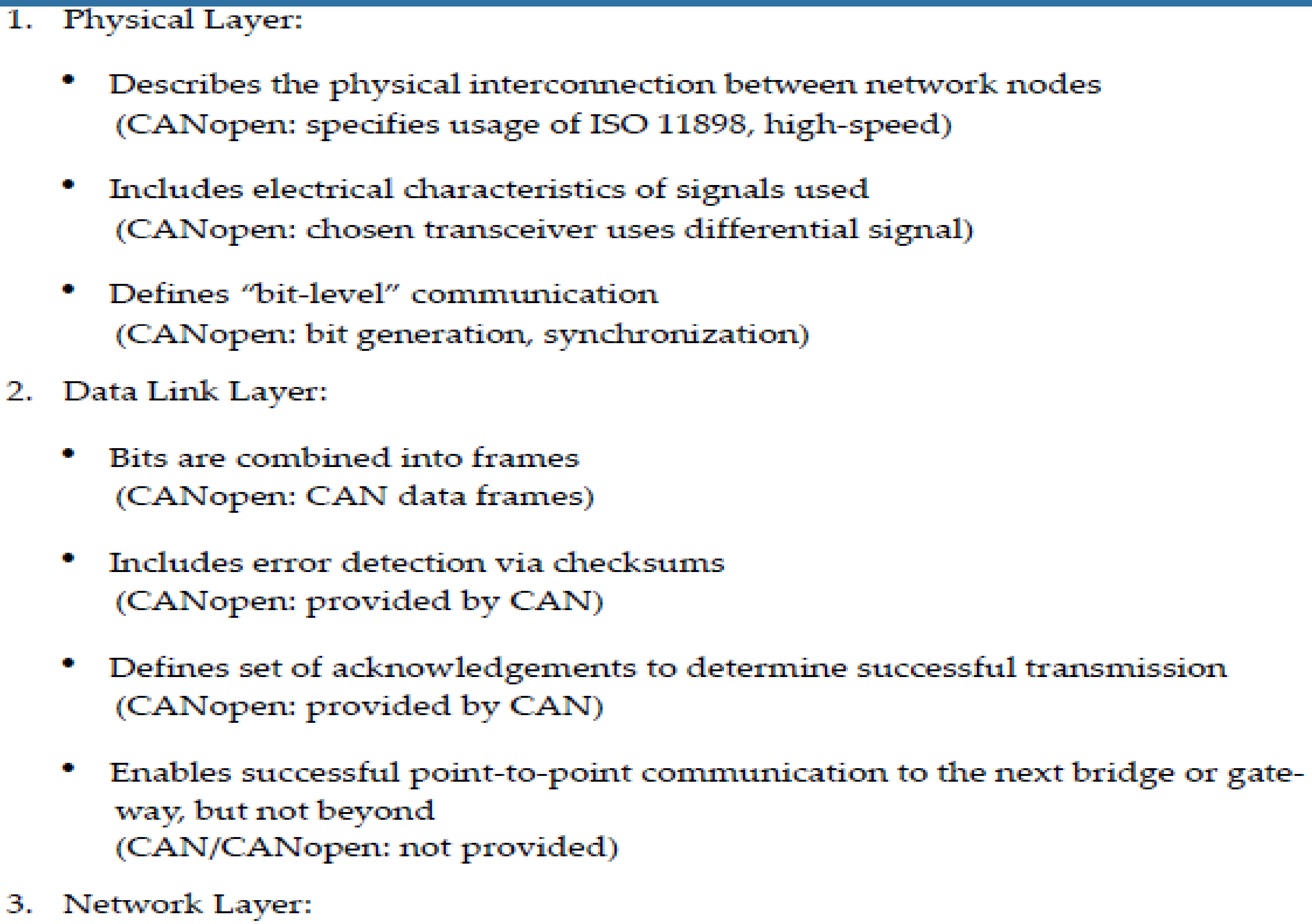

7. Application Layer
6. Presentation Layer
5. Session Layer
4. Transport Layer
3. Network Layer
2. Data Link Layer
1. Physical Layer
Bypass used without higher-layer protocols
Partially implemented by higher-level CAN protocols like CANopen
Standard CAN implementation (physical details often specified by higher-layer protocol)

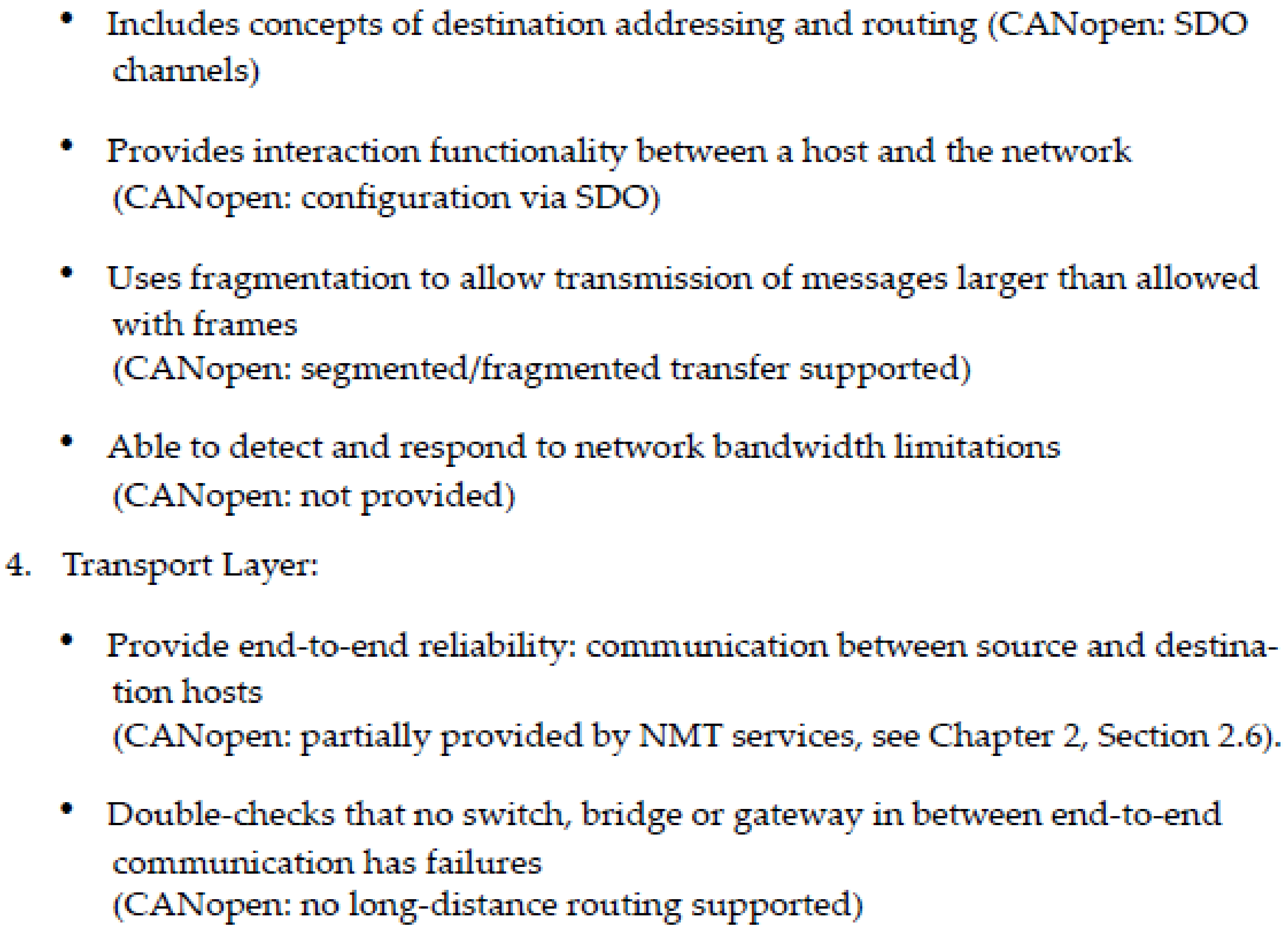
Figure 1.7 The ISO 7-layer Reference Model

OSI MODEL



1. Physical Layer:
    - Describes the physical interconnection between network nodes
      (CANopen: specifies usage of ISO 11898, high-speed)
    - Includes electrical characteristics of signals used
      (CANopen: chosen transceiver uses differential signal)
    - Defines "bit-level" communication
      (CANopen: bit generation, synchronization)
2. Data Link Layer:
    - Bits are combined into frames
      (CANopen: CAN data frames)
    - Includes error detection via checksums
      (CANopen: provided by CAN)
    - Defines set of acknowledgements to determine successful transmission
      (CANopen: provided by CAN)
    - Enables successful point-to-point communication to the next bridge or gateway, but not beyond
      (CAN/CANopen: not provided)
3. Network Layer:

OSI MODEL cont….



- Includes concepts of destination addressing and routing (CANopen: SDO channels)

- Provides interaction functionality between a host and the network (CANopen: configuration via SDO)

- Uses fragmentation to allow transmission of messages larger than allowed with frames
  (CANopen: segmented/fragmented transfer supported)

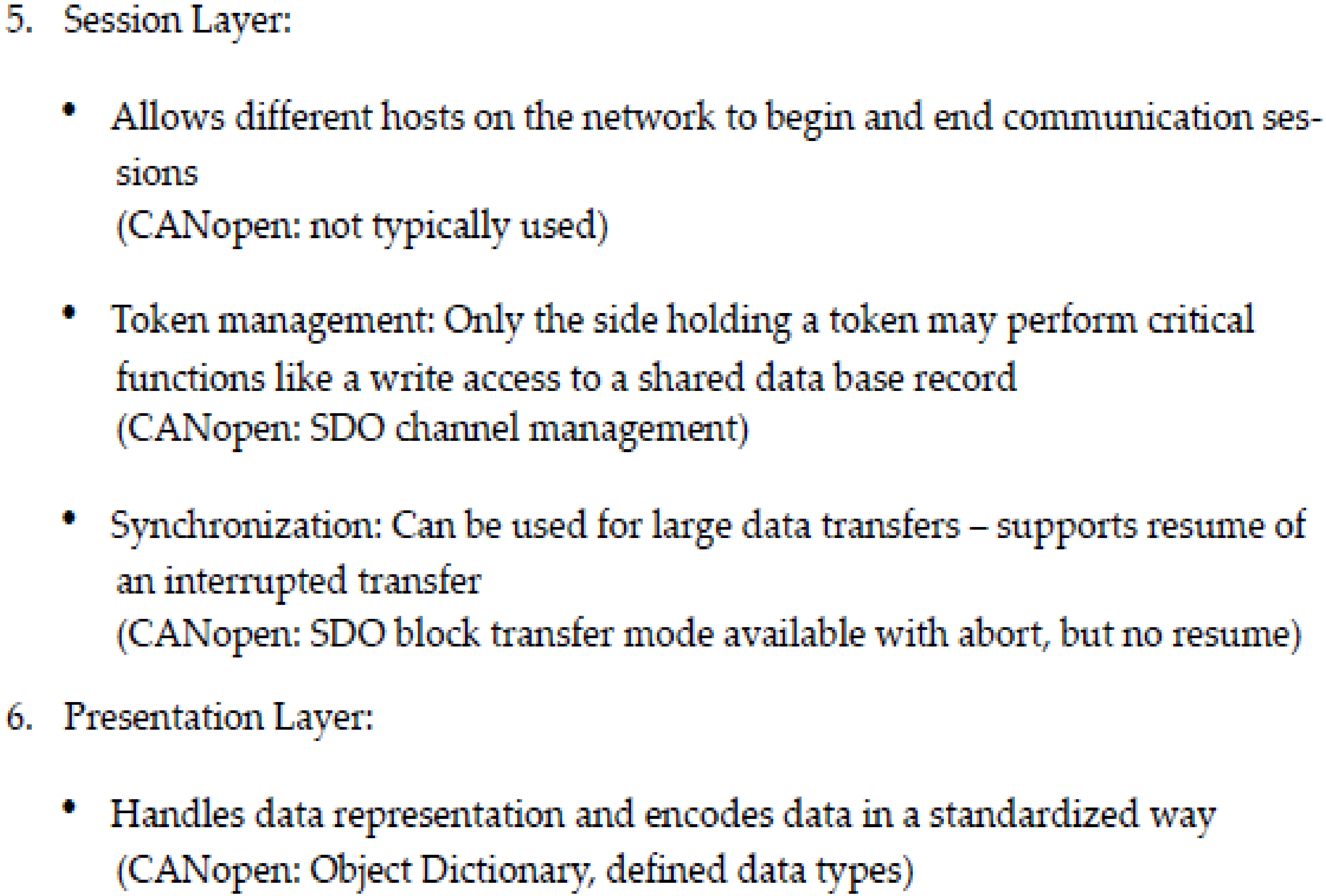
- Able to detect and respond to network bandwidth limitations
  (CANopen: not provided)

4. Transport Layer:

- Provide end-to-end reliability: communication between source and destination hosts
  (CANopen: partially provided by NMT services, see Chapter 2, Section 2.6).

- Double-checks that no switch, bridge or gateway in between end-to-end communication has failures
  (CANopen: no long-distance routing supported)

OSI MODEL cont...



5. Session Layer:

- Allows different hosts on the network to begin and end communication sessions
  (CANopen: not typically used)

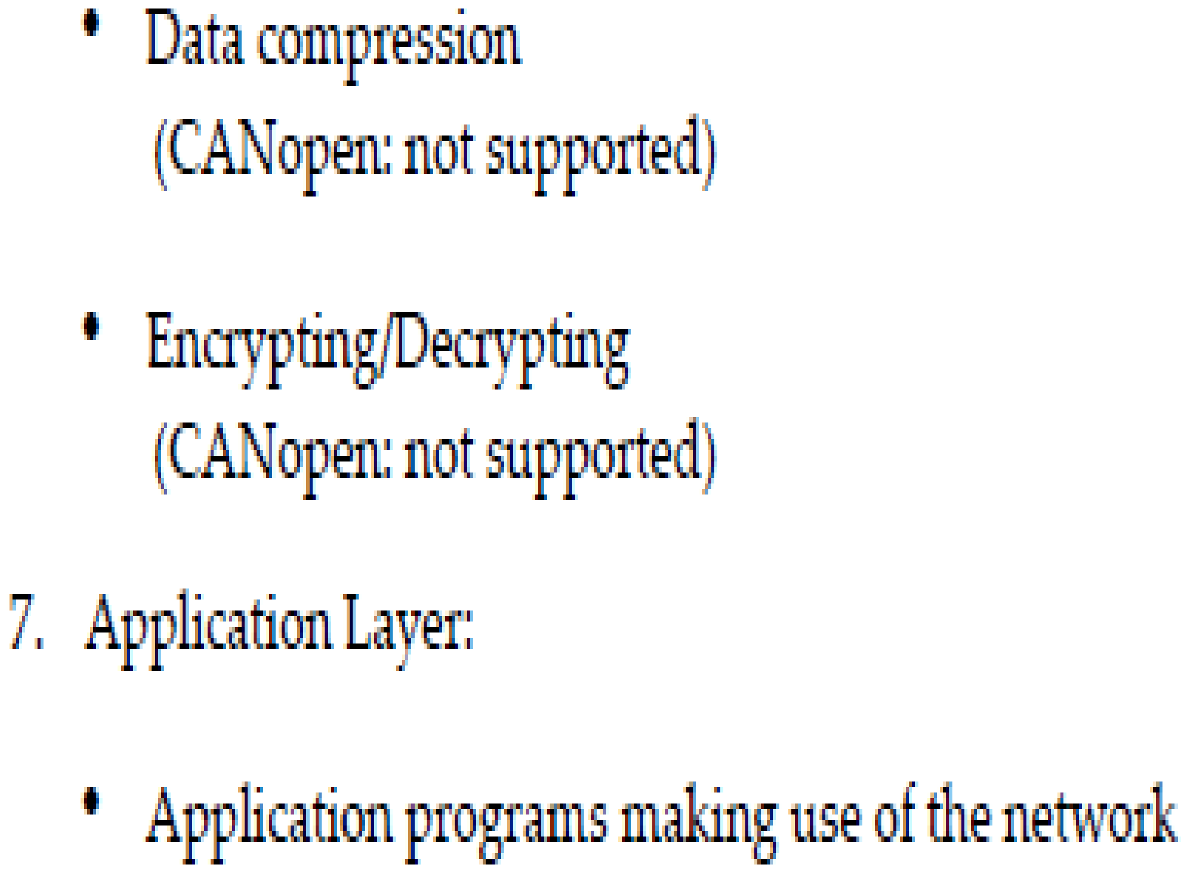
- Token management: Only the side holding a token may perform critical functions like a write access to a shared data base record
  (CANopen: SDO channel management)

- Synchronization: Can be used for large data transfers – supports resume of an interrupted transfer
  (CANopen: SDO block transfer mode available with abort, but no resume)

6. Presentation Layer:

- Handles data representation and encodes data in a standardized way
  (CANopen: Object Dictionary, defined data types)

OSI MODEL cont…



- Data compression
  (CANopen: not supported)

- Encrypting/Decrypting
  (CANopen: not supported)

7. Application Layer:

- Application programs making use of the network

Electronic Data Sheets

- Electronic Data Sheets (EDS) offer a standardized way of specifying supported Object Dictionary entries.
- Any manufacturer of a CANopen module delivers such a file with the module, which in layout is similar to the “.ini” files used with Microsoft Windows operating systems.
- (Note: a future standard for EDS files based on XML is currently in development.)

Electronic Data Sheets

- An example of an Object Dictionary entry in an EDS file is:

```
[1000]
ParameterName=DeviceType
ObjectType=0x07
DataType=0x0007
AccessType=ro
DefaultValue=0x00030191
PDOMapping=0
```

- The example above shows the EDS definition of the Object Dictionary entry [1000h,00h]. The data type is 7 (UNSIGNED32, see Table 1.1).

Electronic Data Sheets

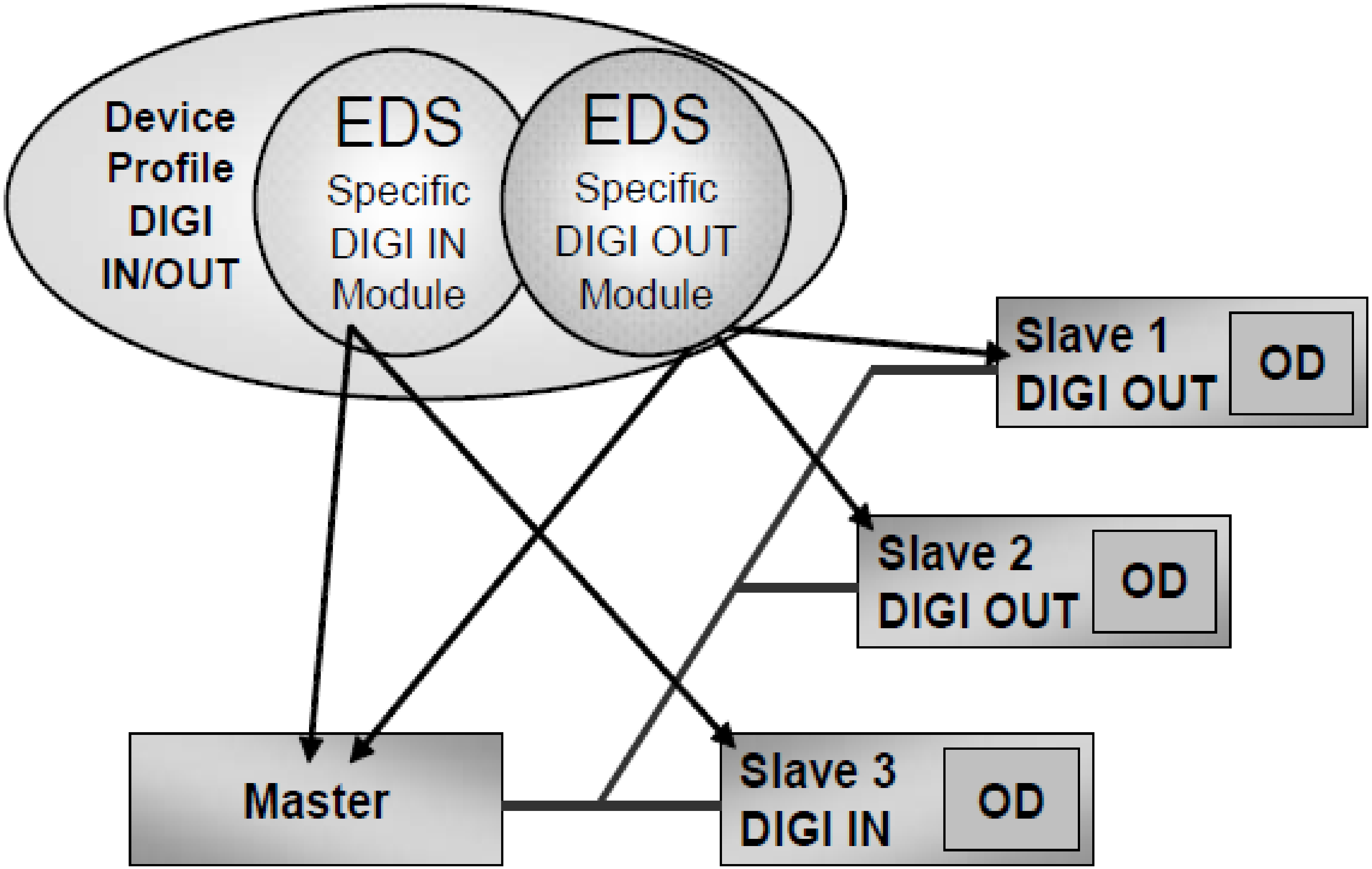


Figure 1.9 Electronic Data Sheets (EDS) Specify the Contents of Object Dictionaries