Language Constructs

Insup Lee, Vujay Gehlot

University of Pennsylvania

Prepared by Prof. Moonkun Lee

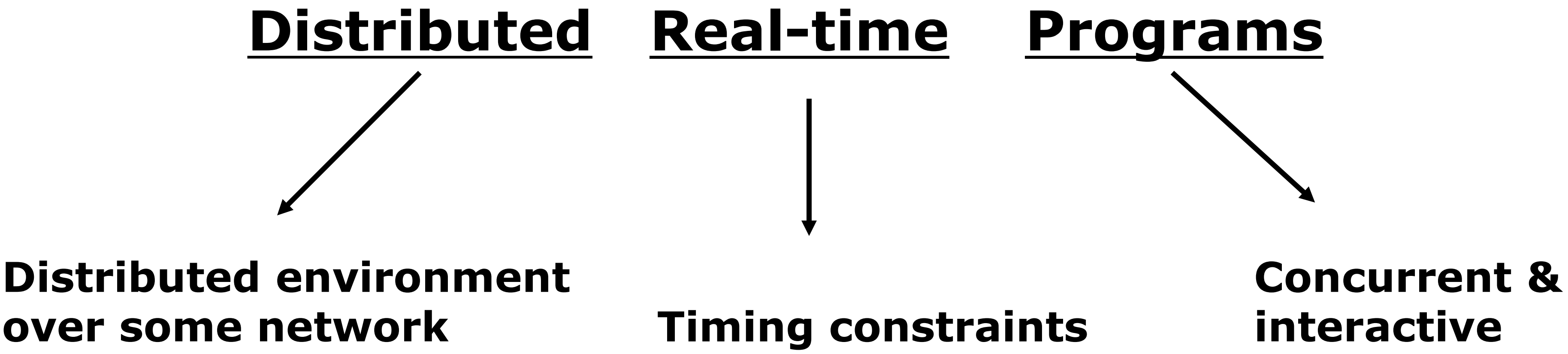
Chonbuk National University

Outline

- Introduction
- Motivation
- Assumptions and Basic Model
- Timing Specification
- Communication
- Exception Handling
- An Example

1. Introduction

Characteristics



Distributed Real-time Programs

Distributed environment over some network

Timing constraints

Concurrent & interactive

- **Applications:**
  - robot arm control, missile control, on-line process control, etc.
- **Requirements:**
  - Logical correctness & timing constraints satisfaction.
- **Conventional approach:** concurrent program w/ time.
  - Scheduling primitives.
  - Scheduler.
  - **Con:** responsibility of programmer.
- **Languages:** limited time spec(delay, sleep, timeout).
  - **Con**: Timing verification problem.

(=>) Need a **new language** for distributed real-time program:

  - Timing constraints.
  - Scheduled by the underlying system.

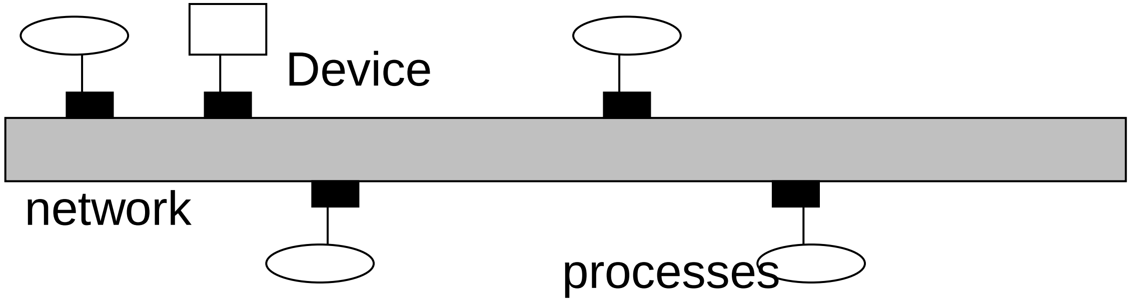
2. Motivation

- **Distributed Programming System (DPS) :**
  - Easy programming environment for distributed real-time programs.
- **Distributed Configuration Specification Language (DICON):**
  - Backbone of DPS
  - Distributivity:
    - Resource requirements.
    - Process interconnection.
    - Process assignment
  - Modularity
- **Design Goals:**
  - Timing constraints: code execution & IPC.
  - Exception handing of timing constraints.
  - Process scheduling by the underlying system.

- Other languages
  - PEARL :
    - single processor system.
    - Con:
      - Non-extensibility for distributed systems.
      - No exception handling.
  - ESTEREL :
    - event-based temporal constructs & an exception handing
    - Con:
      - Under process scheduling.
      - Instantaneous message transmission

3. Assumption & Basic Model

- Environment: processors interconnected by a network.
  - Communication: message passing, no shared memory.
  - Clock: synchronized within little time interval.
- Components:
  - Internal process: independent execution control thread.
    - timing constraints: code segments & communication.
  - External process: part of the external world.
  - Objects: instance of an abstraction of attached special purpose HW & interrupt and control routines.



- Distributed program: off-line & static process creation.
  - Phase of execution:
    - Initialization:
    - Operation:
  - Main process: idle & handling global timing exceptions.
  - Motivation: Verification of correctness.

4. Temporal Scope

- Definition:
  - Specification of timing constraints
- Attributes:
  - Deadline
  - Minimum delay
  - Maximum delay
  - Maximum execution time
  - Maximum elapsed time
- Types:
  - Global temporal scope
    - Encapsulate a whole process
    - Used to define a periodic process
  - Local temporal scope
    - The timing constraints within a process
    - How long or soon the execution of stmts takes
  - Communication temporal scope
    - The timing constraints with IPC
- If any constraint is violated:
  - An exception is raised
  - Handled by an exception handler

- Local Temporal Scope:
- Syntax:

```
start <d-part> [<e-part>] [<dl-part>] do
        <start-body>
      [<exception.]
end
<d-part> ::= now |
                 at <abs-time> |
                 after <rel-time>
<e-part> ::= execute <rel-time> |
                elapse <rel-time>
<dl-part> ::= by <abs-time> |
                 within <rel-time>
```

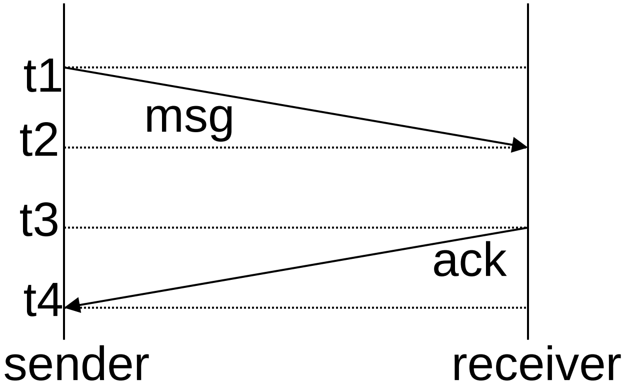
- Example:
  - A process can be put into sleep for 10 seconds:

```
start after 10 sec do end
```

  - A process with a delay-part and deadline-part:

```
start at (9h:00m) within 10 sec do
         /*  stmts  */
exception
         /* stmts   */
end
```

- **Communication Temporal Scope**:
- Spec contents:
  - Sending process:
    - Delivery and process time for a *msg* in a receiver.
    - Delivery and process time of a *msg* in a receiver and delivery time of an *ack* from receiver.
  - Receiving process:
    - Delivery time of a *msg* to receiver.
    - Process time of a received *msg.*



- **Repetitive Temporal Scope:**

Syntax:

```
from < start-time> to <end-time> every <period>
     execute <exec-time> within <dead-line> do
              /*     stmts    */
      [<exception>]
end
```

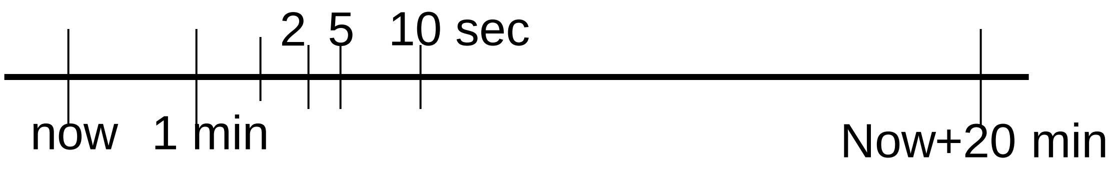
- **Consecutive Temporal Scope**: a composite temporal scope.

Syntax:

```
cstart <delay_1> [<execute-1>] [<deadline_1>] do
       <stmt_1>
   [<exception_1>]
cstart <delay_2> [<execute-2>] [<deadline_2>] do
       <stmt_2>
   [<exception_2>]
 …..
cstart <delay_n> [<execute-n>] [<deadline_n>] do
       <stmt_n>
   [<exception_n>]
```

- **Characteristics:**
  - Can be nested, but not overlapped.
  - Inconsistent deadline specification ignored at runtime:
    - Static deadline inconsistency: compile time.
    - Dynamic deadline inconsistency: runtime.
- **Sporadic process:**
  - Ready at any time.
  - Local and communication temporal scope.
- **Periodic Process**:
  - Ready at regular interval.
  - Insufficient with local & communication temporal scope.
  - Arguments: process name, start & end time, period,
    optional execution time, and deadline.
  - Example:

```
schedule stir at now+1min every 10sec
  execute 2sec within 5sec until now+20min
```



5. Communication

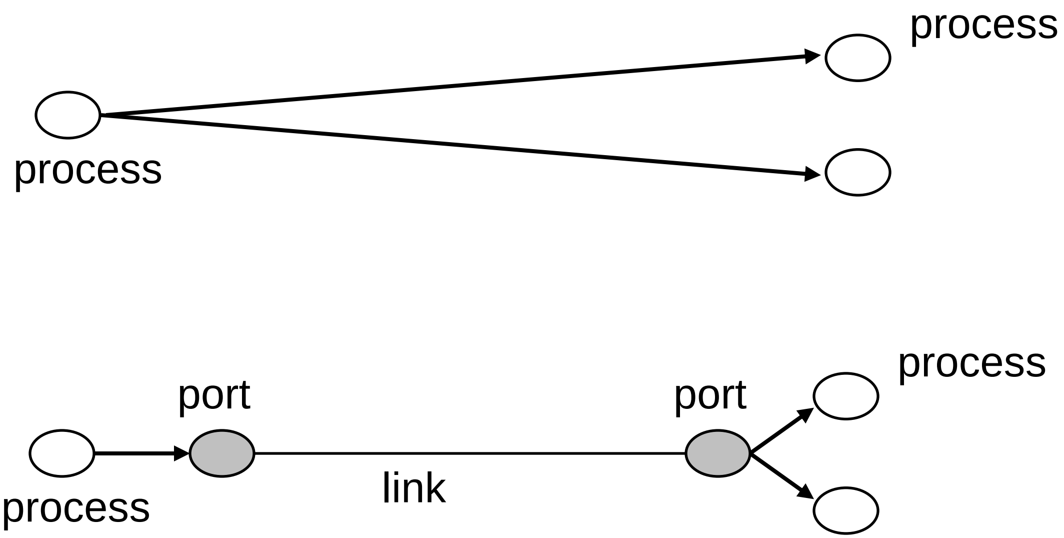
Introduction

- Communication method: message passing.
- Reasons for sending message:
  - To forward data or signal to another process.
  - To synchronize with other process.
  - To request action from other process.
- Types of communication:
  - send_no_wait :
  - synchronization :
  - send_ack :
- Communication model: two-way communication.
- Type of receives:
  - Explicit receive:
    - execute a receive operation;
    - how long wait for a msg,
    - what to do w/ a tardy msg.
  - Implicit receive:
    - corresponding code being executed;
    - no timeout

- Comparison:
  - no timing constraints in other primitives (except timeout).
- Design goals:
  - Timing constraints specification.
  - Exception handling.
  - Overflow control.
  - Msg type checking issue.

Naming & Buffer Control

- Static name creation at compile time.
- Type of naming (communication):
  - Direct: 1 or 2 ways.
  - Indirect: port or link



- Advantage:
  - Integration of modules without naming conflicts.
  - Buffer overflow control strategy (link).
- Kinds of real-time communication paradigm:
  - Asynchronous communication w/ non-queued msg.
  - Synchronization communication w/o msg loss.
  - Synchronous and asynchronous communication w/ possible loss of aged msg.
- Timing constraints:
  - Deadline of a msg.
  - Static size of msg buffers for each link.
  - Overflow control strategy.
- No blocking of a sending process when there is no available buffer.

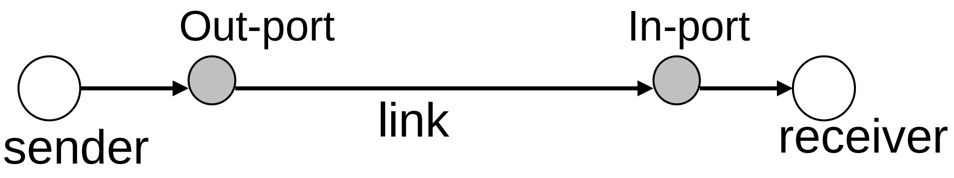
Unidirectional Communication

- **Def:**
  - Asynchronous communication using an one-way link.
- **Example:**
  - Sender:

```
send(OutPortId, var)
```

  - Receiver:

```
accept on <port-list> [within | by] <timeout>
   when port_1(arg):   /* stmts */
   when port_1(arg):   /* stmts */
       ….. 
   when port_n(arg):   /* stmts */
   when timeout:       /* stmts */
end
```



- **Characteristics:**
  - Accept construct: a communication temporal scope.
  - Handle the most time-critical msg.
  - Timing constraint inconsistency: check at compile time
- **Limitation:** No *ack* for timeout msg.
  - Two options to handle:

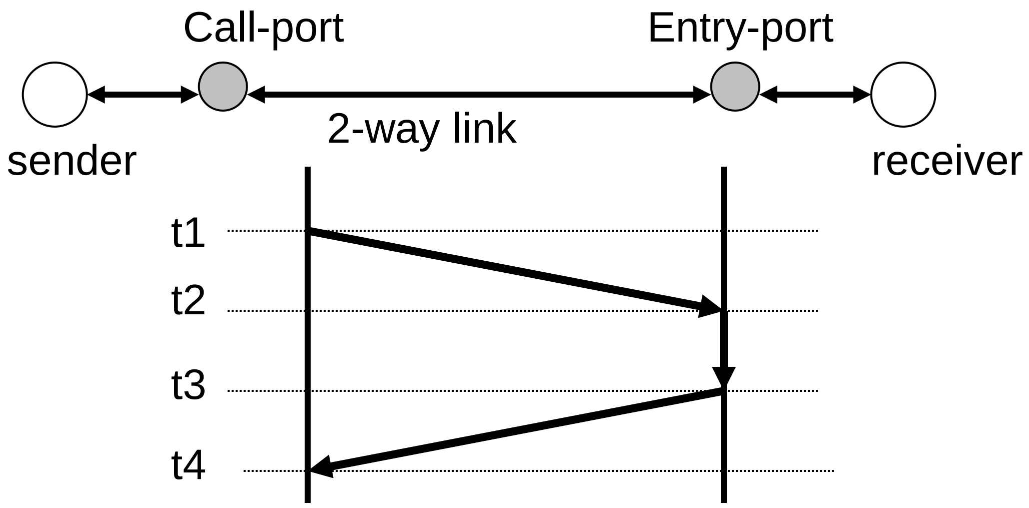
    1) In-port: receive only non-deadline msg.

    2) Keep deadline msgs and raise exceptions.
- **Future msg:** sending a msg at future time.

```
DelayedSend(OutPortId, time, var)
```

Bidirectional Communication

- **Def:**
  - A pair of asynchronous communication on a two-way link.



- **Timing constraints:**

  1) Deadline of a msg(call-port) : t4-t1.

  2) Processing time of msg(entry-port): t3-t2.

  3) Waiting time for a msg(entry-port): t2-t1.

- **Syntax:**
  - Sender:

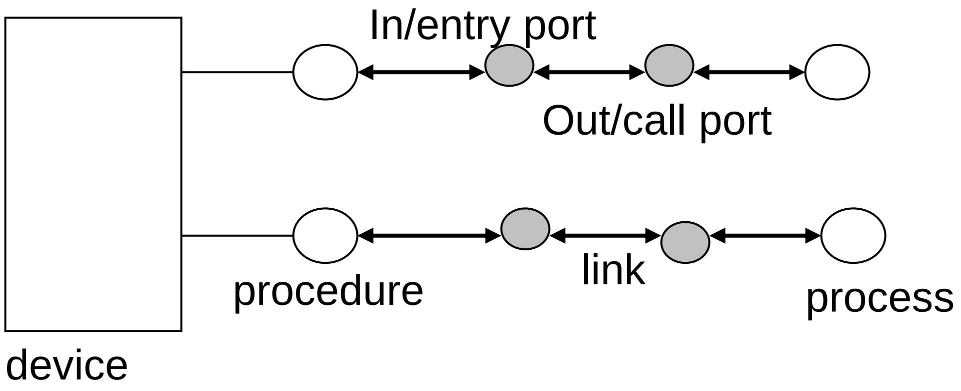
```
call(CallPortId, msg)
  /*  stmt  */
receive(CallPortId, ArrayVar, NumofReplies)
```

From receiver

Specify how many
replies are expected

Communication w/ shared Object

- Shared Object: Data, devices.
- Procedures defined to objects.
- Invoking of procedures:
  - by sending a msg to a port linked to procedures.
- A process is dedicated to handle remote procedure calls:
  - The most time critical request: deadline.
  - Maximum utilization.
  - Preemption: a list of preemptable processes.



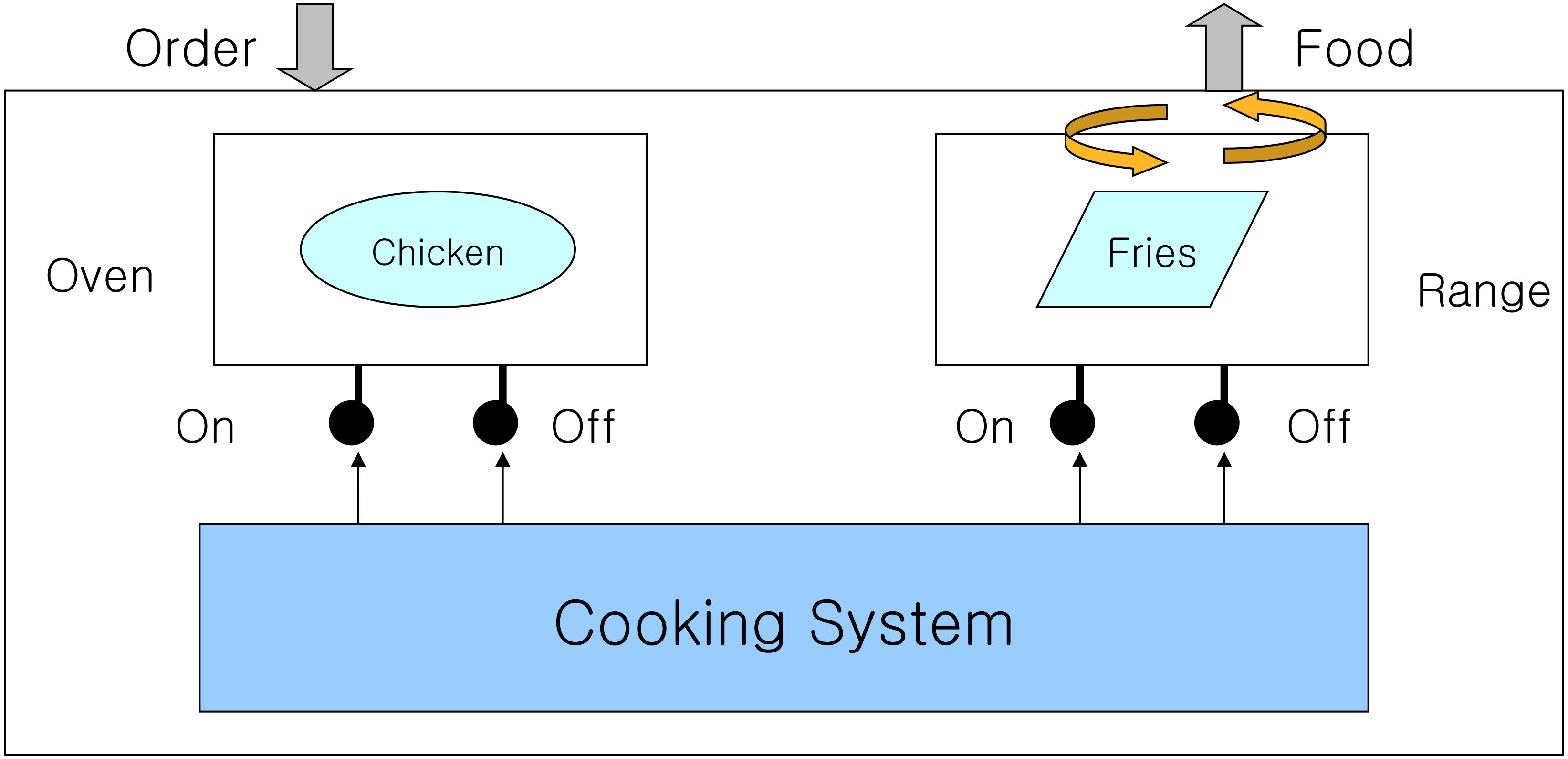
6. Exceptional Handling

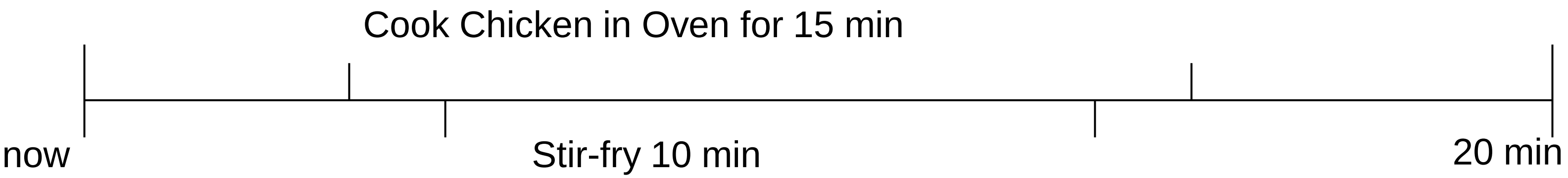
- An exception handling mechanism for timing error.
- Issues:
  1) Detection of exception: when
  2) Handling of exception: which process
  3) Recovery action:
  4) Time for exception handling:
- **Declaration:**
  1) End of local temporal scope.
  2) End of the body of the main process.
- **Syntax:**

```
start ...
   ...
exception
  when <exception list 1> within <deadline>: ...
  when <exception list 2> within <deadline>: ...
              …
end
```

7. Example

```
process cooking_robot;
 call-port RangeOn [deadline 2 sec], RangeOff [deadline 2
   sec];
 in-port: OvenOn [deadline 2 sec], OvenOff [deadline 2 sec];
 var ToBeDone : time
begin
   start now within 20 min do
      call (RangeOn, nil);
      send (OvenOn, nil);
      receive (RangeOn, nil, 1)l
      ToBeDone := now + 15 min ;
      delayedsend (OvenOff, ToBeDone, nil);
      from now to now+10min every 40 sec
         execute 10 sec within 10 sec do
      end;
      call ( RangeOff, nil);
      receive(RangeOff, nil, 1);
      start after (ToBeDone-now) do end;
      end;
end;
```


Order
Food
Oven
Chicken
Fries
Range
On
Off
On
Off
Cooking System


Cook Chicken in Oven for 15 min
now
Stir-fry 10 min
20 min