Australian National University

S03

Message-Based Synchronization

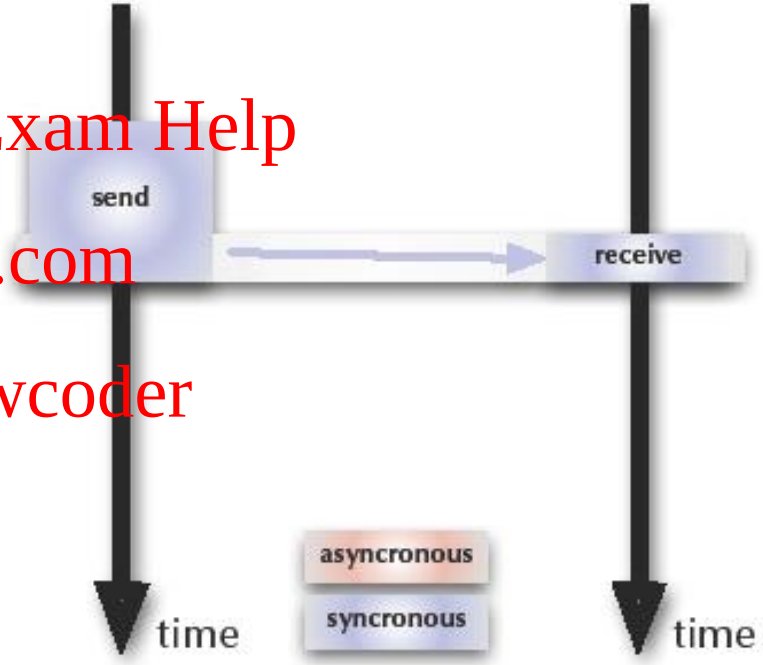
Message Protocols

Message-Based Synchronization

- Synchronization model
  - Asynchronous
  - Synchronous
  - Remote invocation
- Addressing (name space)
  - direct communication
  - mail-box communication
- Message structure
  - arbitrary
  - restricted to ‘basic’ types
  - restricted to un-typed communications

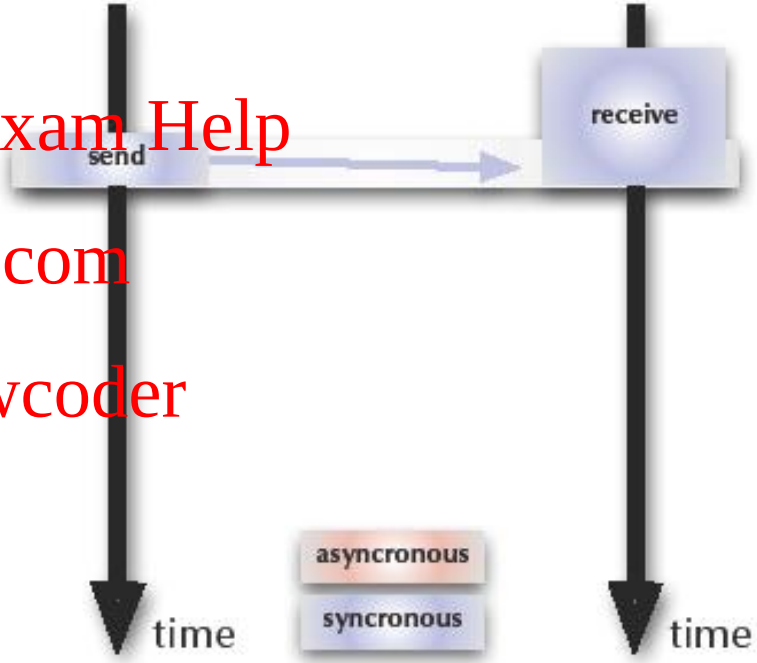
Message Protocols: Synchronous

- Delay the sender process until:
  - Receiver becomes available
  - Receiver acknowledges reception



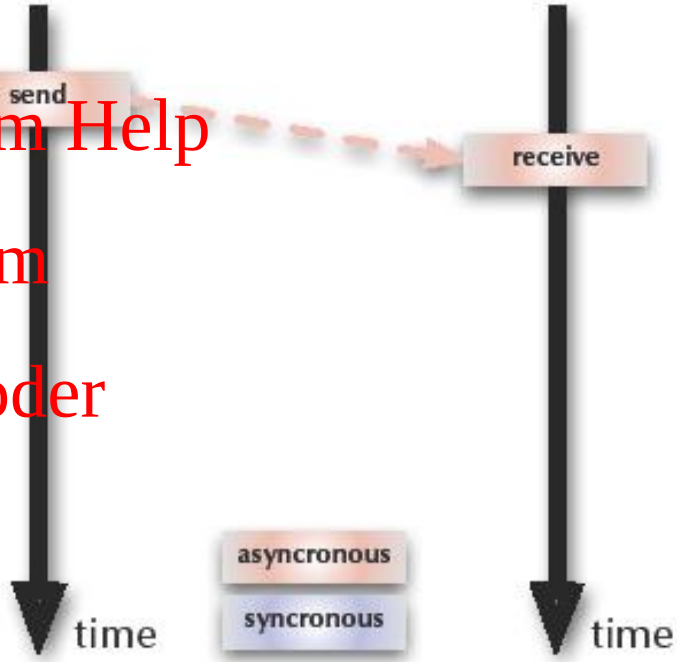
Message Protocols: Synchronous

- Delay the receiver process until:
  - Sender becomes available
  - Sender concludes transmission



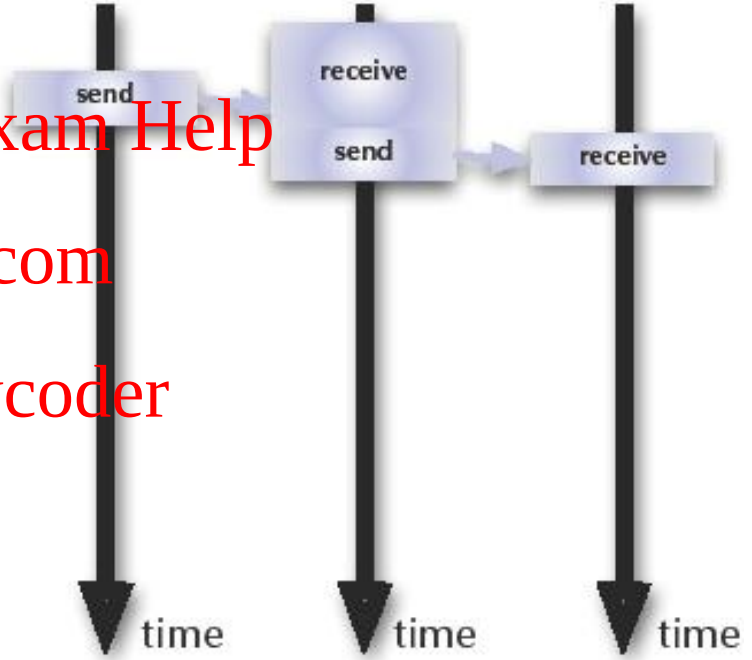
Message Protocols: Asynchronous

- Neither the sender nor the receiver is blocked:
  - Message is not transferred directly
  - A buffer is required to store the messages
  - Policy required for buffer sizes and buffer overflow situations



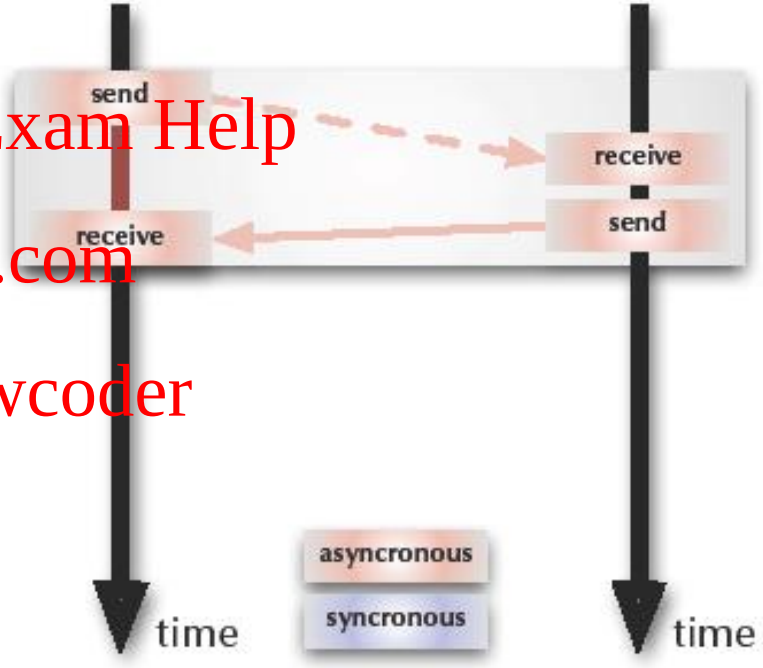
Message Protocols: Asynchronous (Emulated)

- Introducing an intermediate process:
  - Intermediate needs to be accepting messages at all times
  - Intermediate also needs to send out messages on request
  - While processes are blocked in the sense of synchronous message passing, they are not actually delayed as the intermediate is always ready



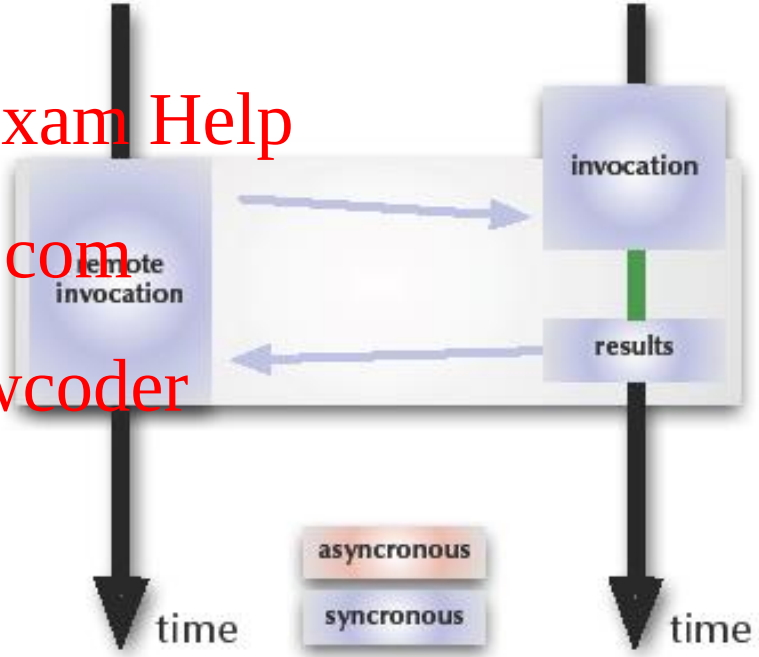
Message Protocols: Synchronous (Emulated)

- Introducing two asynchronous messages:
  - Both processes voluntarily suspend themselves until transaction is complete
  - As no immediate communication takes place, processes are never actually synchronized
  - Sender (but not receiver) process knows that transaction is complete



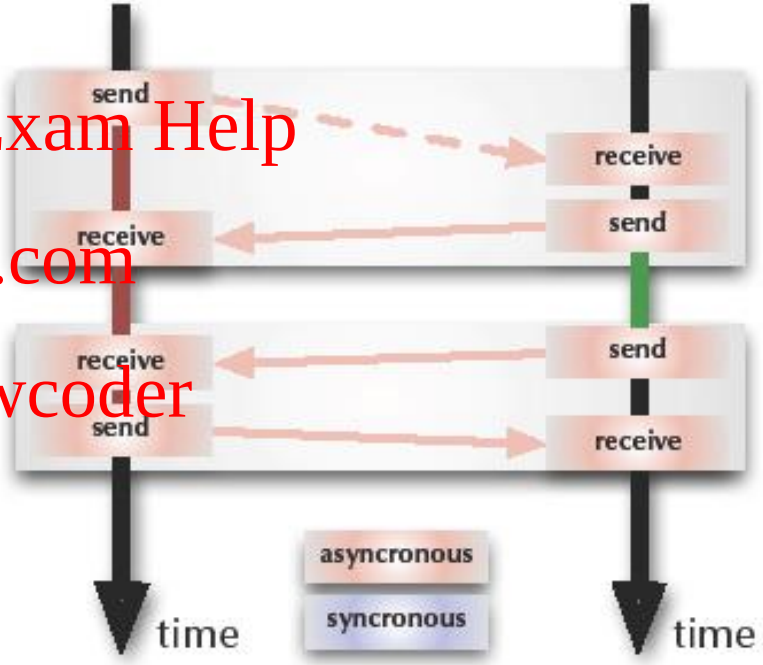
Message Protocols: Remote Invocation

- Delay sender or receiver until the first rendezvous point
- Pass parameters
- Keep sender blocked while receiver executes the local procedure
- Pass results
- Release both processes out of the rendezvous



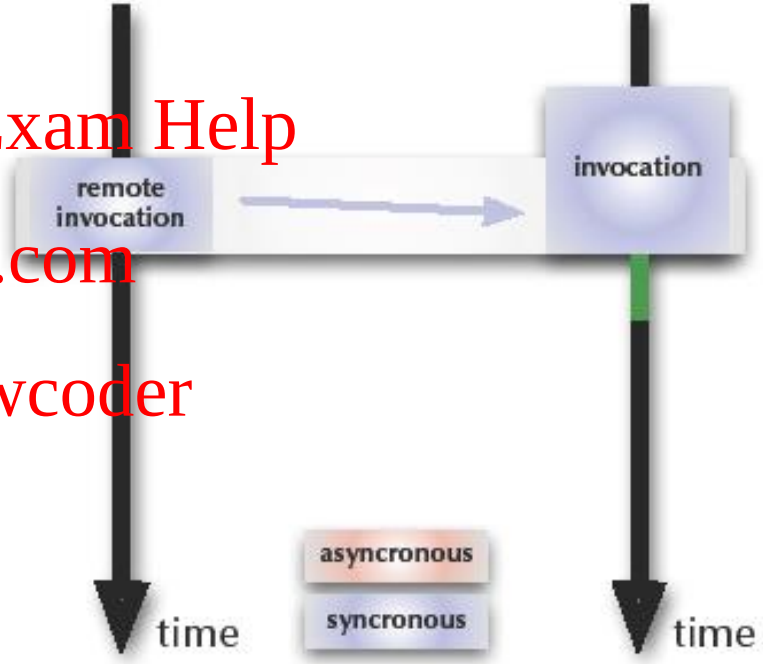
Message Protocols: Remote Invocation (Emulated)

- Emulate two synchronous messages
  - (Could combine middle two async)



Message Protocols: Remote Invocation (no result)

- Also called *active message*
  - Shorter form of remote invocation which does not wait for results to be passed back
  - Still both processes are actually synchronized at time of invocation



Synchronous vs. Asynchronous Communications

- Purpose ‘synchronization’:
  - synchronous messages / remote invocations
- Purpose ‘last message(s) only’:
  - asynchronous messages
- Synchronous message passing in distributed systems requires hardware support
  - emulated by asynchronous messages in some systems
- Asynchronous message passing requires buffers and overflow policies
  - emulated using synchronous messages by introducing a ‘buffer-task’ (de-coupling sender and receiver as well as allowing for broadcasts)

Addressing (Namespace)

- Direct:

```
send <message> to <process-name>
wait for <message> from <process-name>
```

- Indirect:

```
send <message> to <mailbox>
wait for <message> from <mailbox>
```

- Asymmetrical addressing:

```
send <message> to …
wait for <message>
```

  – Client-server paradigm

Addressing (Namespace)

Connections	Functionality
one-to-one	buffer, queue, synchronization
one-to-many	multicast
one-to-all	broadcast
many-to-one	local server, synchronization
all-to-one	general server, synchronization
many-to-many	general network- or bus-system

Message Structure

- How are complex types built on machine-dependent representations handled in a distributed environment?
- Communication system is often outside the typed language environment. Most communication systems only handle streams (packets) of a basic element type
- Conversion routines for data-structures other than the basic element type are supplied:
  - manually (POSIX, C)
  - semi-automatically (CORBA)
  - automatic (compiler-generated) and typed-persistent (Ada, CHILL, Occam2)

Message Structure (Ada)

```
package Ada.Streams is
   pragma Pure (Streams);
   type Root_Stream_Type is abstract tagged limited private;
   type Stream_Element is mod implementation-defined;
   type Stream_Element_Offset is range implementation-defined;
   subtype Stream_Element_Count is
     Stream_Element_Offset range 0..Stream_Element_Offset'Last;
   type Stream_Element_Array is
     array (Stream_Element_Offset range <>) of Stream_Element;
   procedure Read (…) is abstract;
   procedure Write (…) is abstract;
   private … -- not specified by the language
end Ada.Streams;
```

Message Structure (Ada)

Reading and writing values of any subtype S of specific type T to a Stream:


```
procedure S'Write       (Stream : access Ada.Streams.Root_Stream_Type'Class;
                         Item : in T);
procedure S'Class'Write (Stream : access Ada.Streams.Root_Stream_Type'Class;
                         Item : in T'Class);
procedure S'Read        (Stream : access Ada.Streams.Root_Stream_Type'Class;
                         Item : out T);
procedure S'Class'Read  (Stream : access Ada.Streams.Root_Stream_Type'Class;
                         Item : out T'Class)
```

Reading and writing values, bounds and discriminants:

```
procedure S'Output      (Stream : access Ada.Streams.Root_Stream_Type'Class;
                         Item : in T);
function S'Input (Stream : access Ada.Streams.Root_Stream_Type'Class) return T;
```

Message-Passing Systems Taxonomy

	ordered	symmetrical	asymmetrical	synchronous	asynchronous	direct	indirect	contents	one-to-one	many-to-one	many-to-many	method
POSIX MQ	✓	✓	✓		✓		✓	byte stream			✓	message queues
MPI	✓	✓	✓	✓	✓	✓	✓	basic types	✓	✓	✓	message passing
CHILL	✓	✓	✓	✓	✓		✓	basic types		✓	✓	message passing
Occam2	✓	✓		✓			✓	fully typed	✓			message passing
Ada	✓		✓	✓	✓	✓		fully typed		✓		remote invocation
Go	✓	✓		✓	✓	✓		fully typed	✓			channels
Erlang	✓	✓			✓	✓		fully typed	✓			message passing

Message-Based Synchronization in Ada

- Ada supports remote invocations ((extended) rendezvous) in form of:
  - entry points in tasks
  - full set of parameter profiles supported
- If local and remote task are on different architectures, or if an intermediate communication system is employed then:
  - parameters incl. bounds and discriminants are ‘tunnelled’ through byte-stream-formats.
- Synchronization:
  - Both tasks are synchronized at beginning of remote invocation (‘rendezvous’)
  - Calling task if blocked until remote routine is complete (‘extended rendezvous’)

Message-Based Synchronization in Ada

sender

receiver


```
<entry_name> [(index)] <parameters>
------ waiting for synchronization
…
```



```
-------synchronized
```

```
accept <entry_name> [(index)]
       <parameter_profile>;
```

Message-Based Synchronization in Ada

sender

receiver


```
                                    accept <entry_name> [(index)]
                                      <parameter_profile>;
                                    ------ waiting for synchronization
                                    ...
<entry_name> [(index)] <parameters>       -------synchronized
```

Message-Based Synchronization in Ada

sender

receiver


```
<entry_name> [(index)] <parameters>
------ waiting for synchronization
…

-------synchronized                    accept <entry_name> [(index)]
                                                <parameter_profile> do
                                          -- local operations
-------return results                  end <entry_name>
```

Task Entries in Ada

- In contrast to protected object entries, task entries can call other blocking operations
- Accept statements can be nested (but need to be different)
  - helpful e.g. to synchronize more than two tasks
- Accept statements can have a dedicated exception handler (like any other code-block)
  - Exceptions that are not handled during rendezvous phase propagate to all involved tasks

Task Entries in Ada

- Parameters cannot be direct ‘access’ parameters, but can be access-types
  - `'count` on task-entries is defined,
    but is only accessible from inside the task which owns the entry
- Entry families (arrays of entries) are supported
- Private entries (accessible for internal tasks) are supported