Message-Based Process Synchronization

Alexandre David

1.2.05

adavid@cs.aau.dk

AALBORG UNIVERSITY DENMARK

Aims

- Understand concepts related to process synchronization.
  - synchronous
  - asynchronous
  - general mechanisms
- Map those concepts to a few target languages.

Types of synchronization

- Via shared memory and related mechanisms
  - semaphore
  - mutex
  - pipes (can be classified as message)
- Via messages
  - send/receive messages
  - synchronous
  - asynchronous
  - group communication

Message-based – classification

- Asynchronous ✓
  - sender (or receiver) does not block/wait
    → light-weight, the catch: extra logic.
- Synchronous
  - sender (or receiver) blocks/waits
    → easier to use, the catch: heavier.
- Remote invocation
  - caller has the illusion that a call is local
    → abstract from message, the catch: very heavy.
  - Sender/receiver are not good names in this case.

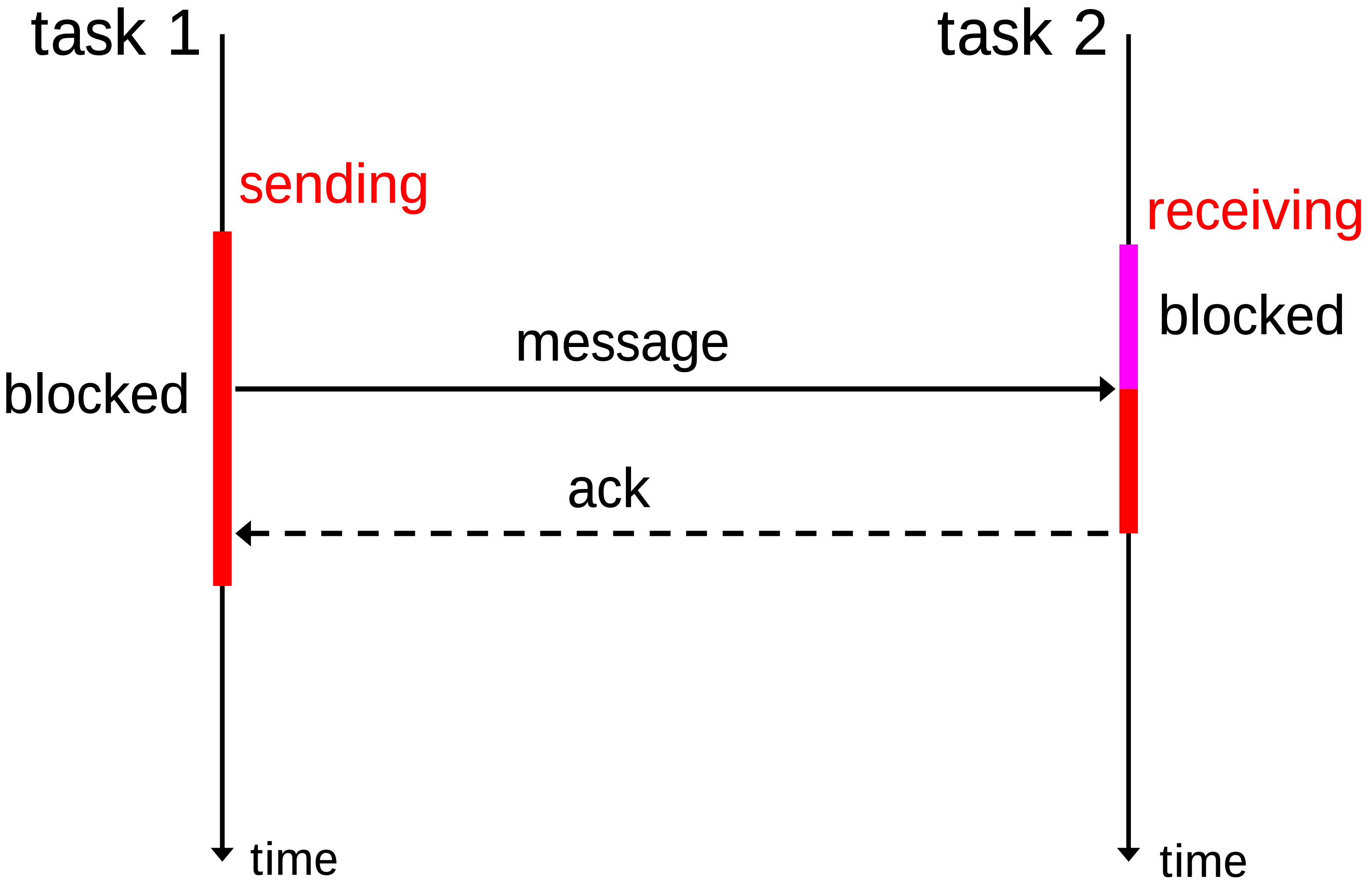
Asynchronous vs synchronous

- Analogy:
  - asynchronous = postcard, may be delayed, out-of-date.
  - synchronous = phone call, often referred as **rendezvous**.
- Asynchronous:
  - buffers are needed, additional logic for acknowledgments, maybe more communication, more complex.
- Synchronous:
  - simpler to use but no concurrency.

Synchronous

- send
  - transfer control to sending implementation (library/driver)
  - wait for interrupt from driver, or time-out
  - read answer
  - re-send if necessary (nack, time-out)
  - return control if success
- receive
  - transfer control to recv implementation
  - wait for interrupt from driver
  - send ack, or nack and wait again
  - return control

Synchronous ✓



Asynchronous

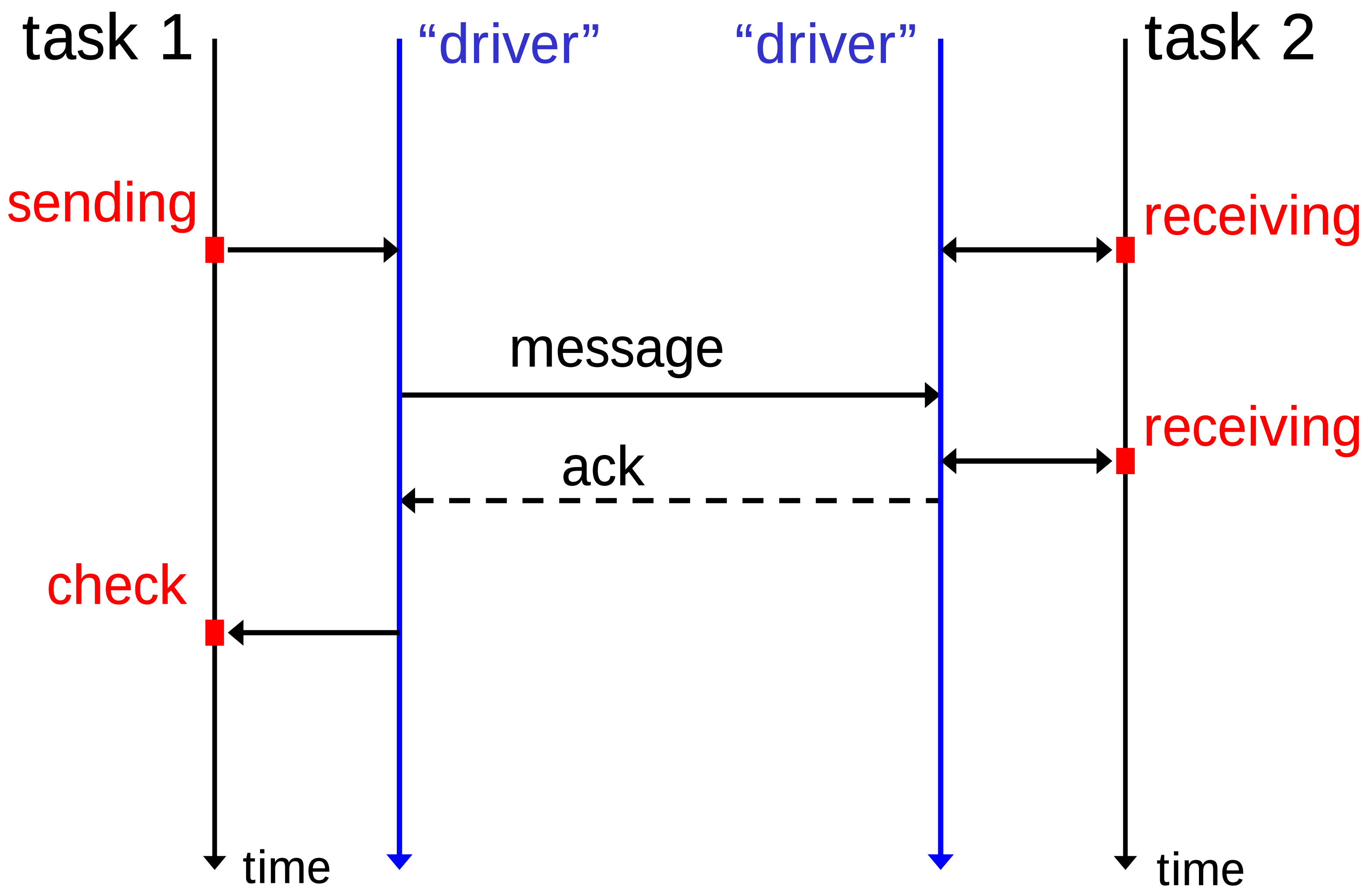
- Buffered or not buffered?
- Not buffered:
  - invoke library call with a pointer
  - return **while the transfer is being done**
  - check later when it's finished to reuse memory
- Buffered:
  - the call will copy the data before returning so it can be reused immediately, no need to check later.

Asynchronous ✓

- send
    - call library
        - concurrent thread/task runs
    - return
        - sending finishes at some point
    - check status
- receive
    - call library
        - concurrent thread/task runs
    - return status
        - may be finished if message was arrived, maybe not
    - may try again later

Asynchronous

✓



Remote invocation – principle

- synchronous send query
- wait reply

- wait query
- process query
- synchronous send reply

- There more to it:
  - illusion of local call
  - passing data across the network

Naming ✓

- Who do you send to?
- Direct or explicit:
  - give task/process as argument
- Indirect:
  - give channel/mailbox as argument
    → interface between communicating processes.
- Apply to sender:
  - send to ID or mailbox
  - broadcast to group
- Apply to receiver:
  - receive from ID or mailbox
  - receive from any

Message passing in Ada

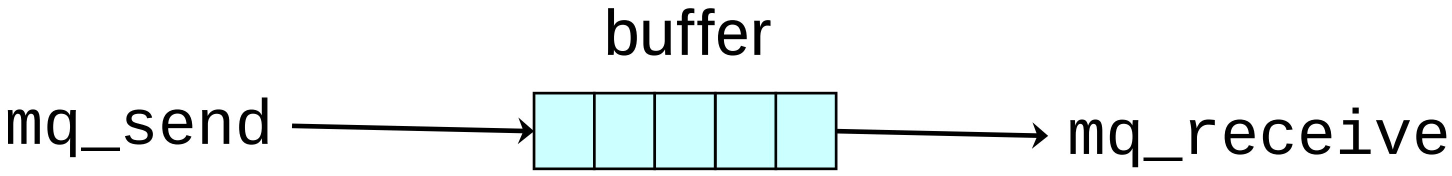
- Tasks declare an entry.
  - Defines interface for receiving messages.
  - Entry family = array of entries.

```
task type Foo is
  entry Family(number)(Data: Type);
  entry Recv(Data: Type);
end Foo
```

- Actual reception: accept.
- Exception handling
  -
```
exception
  when BadException =>
    something;
end
```

Message passing in POSIX

- C/Real-time POSIX message queues
  - type `mqd_t`
  - Named when opened with `mq_open`.
  - Send/receive from/to a buffer with `mq_send` and `mq_receive`.
    - Buffer full → block.
  - Error codes returned, no exception.



Guarded commands ✓

- Dijkstra 1975

  -
    ```
    if x <= y -> m := x
    □  x >= y -> m := y
    fi
    ```
  - Guarded commands by a boolean expression.
  - Choice non-deterministic if several evaluate to true.
  - Not an if-then-else.
  - If the command is a message operator, it is a <u>selective waiting</u> (Hoare 1978).

Ada select

- Map selective waiting concept.

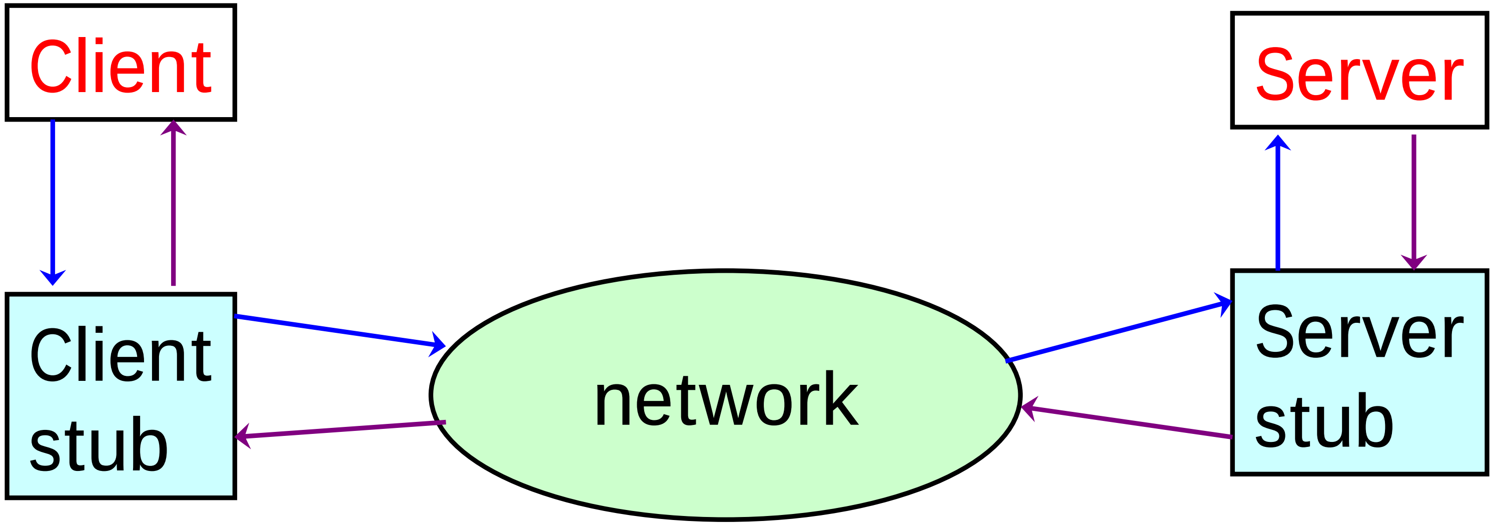
  -
    ```
    task Server is
      entry S1(…);
      entry S2(…);
    end Server;
    task body Server is
     …
    begin
      loop
        select
          accept S1(…) do
            …
          end S1;
        or
          accept S2(…) do
            …
          end S2;
        end select;
      end loop;
    end Server;
    ```

If none → Program_Error

If several → choose one

Remote procedure calls RPC

- Abstraction from messages and communication protocol.
  - Similar to a “standard” procedure call.
- Principle:



Steps of RPC

✓

- Client stub:
  - find address of remote procedure (like DNS)
  - convert parameter for transmission – marshalling
  - send request
  - wait for reply
  - unmarshal the result
  - return result or raise exception
- Server stub:
  - receive requests
  - unmarshal paramaters
  - execute, catch exceptions
  - marshal the result or exceptions
  - send the result back

Distributed object model

- Distributed or remote objects:
  - created remotely and dynamically
  - identified remotely
  - methods transparently invoked
    - transparent run-time dispatching across the network
- Support
  - Ada – static allocation, identification or remote Ada objects, remote execution.
  - Java – send code & create instances remotely, remote execution, via remote method interface.
  - C – CORBA implementation (common object request broker architecture) as library, skeleton code to fill for client and server, has a special interface language: IDL – interface definition language.