# Remote Procedure Call

- Ideally suited for Client-server structure.
- Combines aspects of monitors and synchronous message passing:
  - Module (class) exports operations, invoked with `call`.
  - `call` blocks (delays caller) until serviced.
- Operations are two-way communications channel (just like local procedure call).
- `call` causes a new process to be created on remote (server).
- Client-server synchronization and communication is implicit.





# Terminology / Notation

**module:** operations, (shared) varibles, local procedures and processes for servicing remote procedure calls.

**interface (specification):** describes the operations, parameter types and return types.
`op opname(`*param types*`)` `[returns` *return type*`]`

**server:** process created by call to service an operation.

**background process:** processes running in a module that aren't created in response to call.





# Issues

### Lookup and registration

- How does client find the server?
- Often server registers (binds) with a naming service (registry).





### Module Synchronization

Synchronization within a module (server) has to be programmed. Two approaches:

1) Assume mutual exclusion in server (only one server process/background process executing at a time).
   - Similar to monitors
   - Still need conditional synchronization (conditions, semaphores).
2) Program it explicitly (i.e., using semaphores, monitors etc.).

## Argument Passing

- Formats may be different on different machines (e.g., size, encodings, endianess).
- Address space is not shared (pointers & references can't be passed)
  - copy-in/copy-out: reference arguments converted to byte arrays and reconstructed on the other side.
  - proxy objects



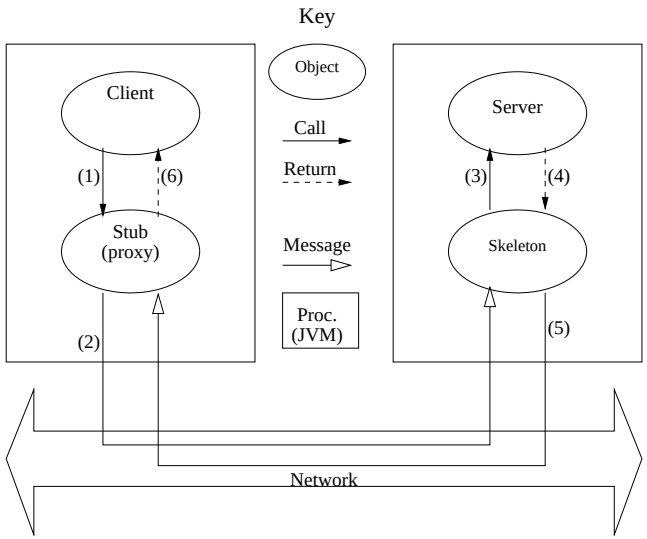


## Java Remote Method Invocation (RMI)

- Client objects and server objects are local to different JVM processes.
- Server objects (usually) extend `java.rmi.UnicastRemoteObject`
- Server objects registered by name with registry service (`Naming.bind`)
- Client objects obtain references to proxy objects (`Naming.lookup`)
- Calls to proxy objects communication with skeleton objects in server's machine.
- Skeleton objects call server objects.












- Multiple calls can be serviced at the same time.
- Format is not an issue since al JVMs follwo same data formats.
- Reference arguments (and subsidiary references) are serialize and passed by copy-in rather than reference. (Except RemoteObjects, in which case a stub is passed instead.)

# (Extended) Rendezvous

- Like RPC, except call is serviced by existing process.
- Mutual exclusion is implicit – only one call serviced at a time.

```
in op1(params) and B1 by e1 -> S1;
[] op2(params) and B2 by e2 -> S2;
[] ...
[] opn(params) and Bn by en -> Sn;
ni
```

- Blocks until one operation succeeds ($op_i$ has been called and $B_i$ is true).
- $e_i$ is a *scheduling expression* – invocation that minimizes $e_i$ is executed first.





## Example

```
module Bounded_buffer {
  op deposit(char);
  op fetch(char);
  body

  process Buffer {
    char buf[n];    # buffer
    int front = 0;  # first full slot
    int rear = 0;   # first empty slot
    int count = 0;  # number of full slots

    while (true) {
      in deposit(data) and count < n ->
          buf[rear] = data;
          rear = (rear+1)% n;
          count++;
```





```
      [] fetch(data) and count > 0 ->
          result = buf[front];
          front = (front+1)% n;
          count--;
      ni
    }
  }
}
```