```
process type chopstick;
    entry pickup;
    entry putdown;

    begin
    repeat
        select
            accept pickup do
                null;
            accept putdown do
                null;
        or
            terminate;
        end;
    forever;
    end;
```

```
var
chopsticks : array[1..N] of chopstick;
```

```
process chairs;
    entry getchair;
    entry replacechair;

    var
    freechairs : integer;

    begin
    freechairs := N - 1;
    repeat
        select
            when freechairs > 0 =>
            accept getchair do
                null;
            freechairs := freechairs - 1;
        or
            accept replacechair do
                null;
            freechairs := freechairs + 1;
        or
            terminate;
        end;
    forever;
    end;
```

```
process type philosopher(name : integer);

    var
    i, chop1, chop2 : integer;

    begin
```

```
process type philosopher (name : integer);
    var
    i, chop1, chop2 :
    integer;

    begin
    chop1 := name;
    if (name = N) then
        begin
        chop2 := 1;
        end
    else begin
        chop2 := name + 1;
        end;

    for i := 1 to 10 do
        begin
        sleep(random(5)); { Thinking. }
        chairs.getchair;
        chopsticks[chop1].pickup;
        chopsticks[chop2].pickup;
        sleep(random(5)); { Eating. }
        chopsticks[chop2].putdown;
        chopsticks[chop1].putdown;
        chairs.replacechair;
        end;
    end;
```

6 Symmetric vs Asymmetric select

Synchronous message-passing has a **symmetric select** — both channel reads and channel writes are legal select alternatives.

With remote invocation this is unnecessary — an **asymmetric select** suffices.

Using remote invocation over synchronous message-passing can actually improve reliability. Consider a client that makes a request and receives a reply from a server process:

(i) Without remote invocation:

```
process server
    entry request(D : DataType);
    entry reply(var D : ResultType);

    { ... }

    repeat
        select
            accept request(D : DataType) do
                { Some computation using D. }

            accept reply(var D : ResultType) do
                { Return the results to the client here. }
        or
            terminate;
        end;
    forever;
```