SCOTT83;
Messages vs. Remote Procedures is a False Dichotomy;
Sigplan Notices, 18, 5 (May 1983).

Concurrent languages are sometimes classified as either ’message based’ or ’procedure based’. Scott argues that while some languages can be identified in this way, several languages are actually a mixture of the two and that the terms are ambiguous. Classifying a language as ’message’ or ’procedure’ based, implies a relationship between message sending and message reception. In fact, these two activities are independent.

The sender of a message can behave in essentially one of three ways. He can send a message and continue execution immediately (no-wait send). He can send a message and continue only after it has been received (synchronization send). Finally, he can send a message and wait for a reply (remote invocation send). Languages that support either of the first woe approaches would be classified as ’message based’. The third approach because of its similarity to a procedure or co-routine call is classified as ’procedure based’

Messages can be received in 2 ways - either implicitly (by an entry point) or explicitly (through a receive statement). Implicit receipt is considered to be ’procedure based’ while explicit receipt is ’message based’.

Most monitor languages (eg. Concurrent Pascal, Mesa) can be classified as ’procedure based’ for both sending and receiving messa
the parameters of the monitor procedure calls).

Several languages use the ’message based’ mechanisms for both sending and receiving messages. Examples of some ’message based’ languages are CSP and Extended CLU.

The naming convention breaks down however, with languages such as Ada. The

So is Ada a ’messages based’ or ’procedure based’ language? This illustrates the false dichotomy of the two terms.

Scott foes on to examine the different send and receive mechanisms to determine which is best. His conclusion is that each one has its place. He then suffests that the best solution might be to include both ’message based’ and ’procedure based’ mechanisms in the language and let the programmer decide!

The article is clearly written and would be of interest to anyone wishing to develop and understqanding of some of the issues in task synchronization and communication. Scott’s suggestion to include all the alternatives in a language and then let the programmer select the most appropriate would probably only serve to increase the complexity of the language.