CS 563
Concurrent Programming

Lecture 4: Message Passing (1)

Message Passing

- Abstract notions representing physical communication possibilities
- Many different names: message queues, pipes, mailboxes, ...
- Primitives:

send msg to dest

receive msg [from source]

Channel

- Unbounded queue of messages

$$\textbf{chan name(}\underbrace{\textbf{id1: type1}}_{\texttt{field}}\textbf{; ...; idN: typeN)}$$

Primitives

- `send name(expr1, ..., exprN)`
  - Types and number of fields must match
  - Effect:
    - Evaluate the expressions and produce a message M
    - Atomically append M to the end of the named channel
      - ➡ send is nonblocking (asynchronous)

Primitives

- `receive name(var1, ..., varN)`
  - Again, types and number of fields must match
    - Effect:
      - Wait for a message on the named channel
      - Atomically remove first message and put the fields of the message into the variables
        - ➡ receive is blocking (synchronous)

Example

```
chan ch(int)


    process A:                    process B:
            send ch(1)                    receive ch(x)
            send ch(2)                    receive ch(y)
```

- x will contain 1 and y will contain 2
- Order of messages from SAME source is the order of the sends

Example

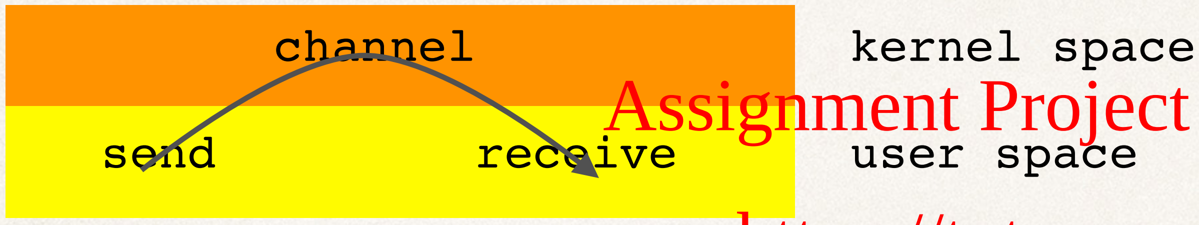
```
chan ch1(int), ch2(int)

            process A:                process B:
              send ch1(1)               receive ch1(x)
              send ch2(2)               receive ch1(y)

            process C:                process D:
              send ch1(3)               receive ch2(u)
              send ch2(4)               receive ch2(v)
```

What is received now?

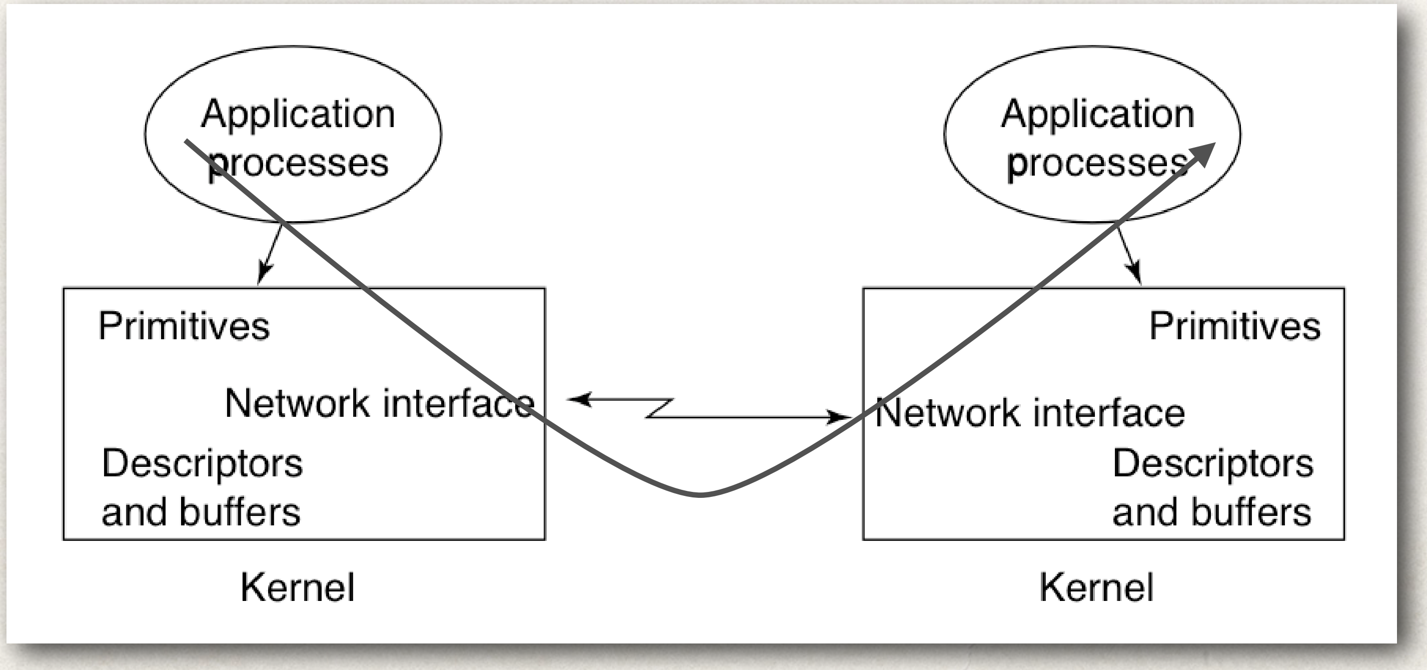
- x will get 1 or 3 and y will get 3 or 1
- u will get 2 or 4 and v will get 4 or 2

Implementation Sketch

With shared memory



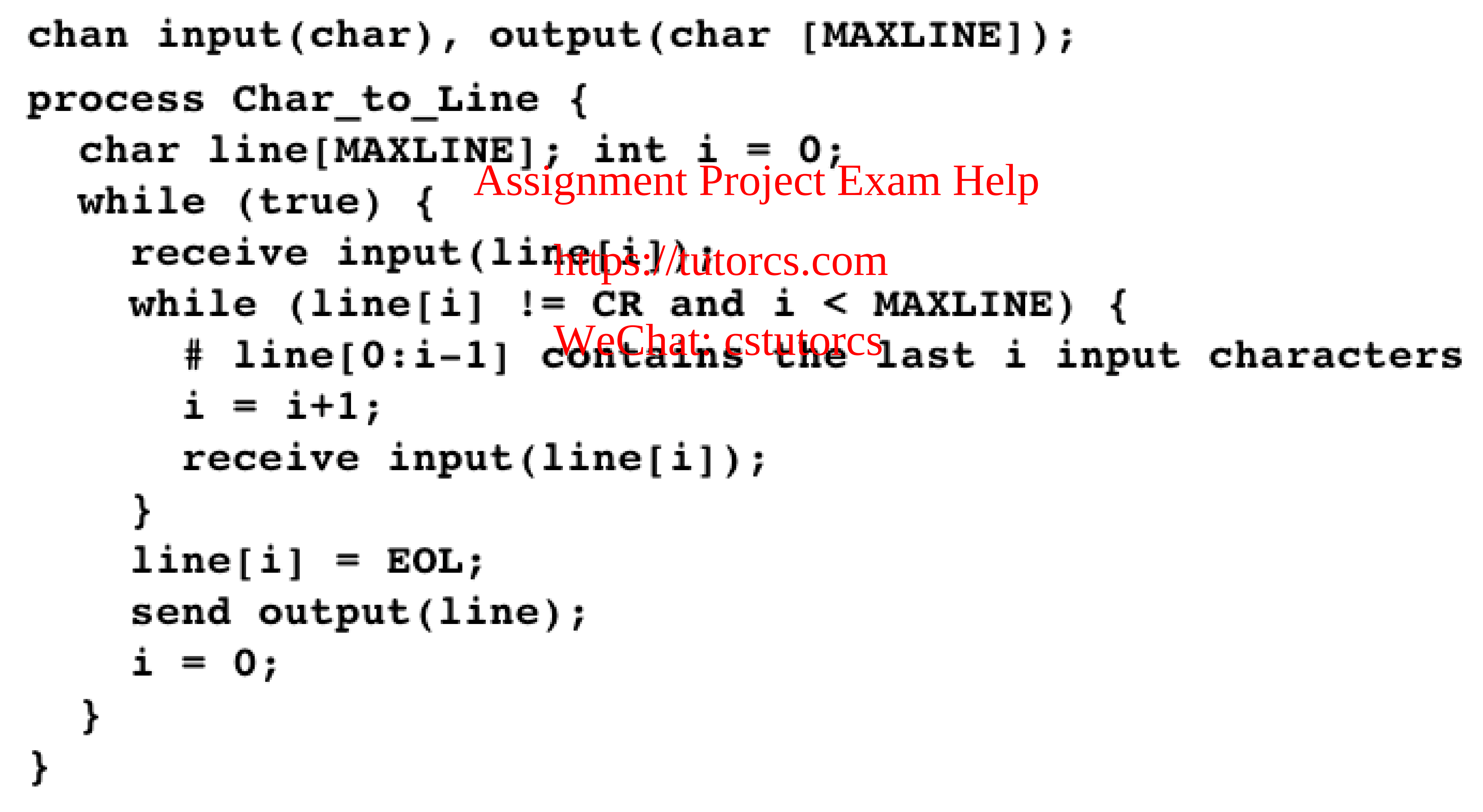
With distributed memory



Process Interaction Patterns

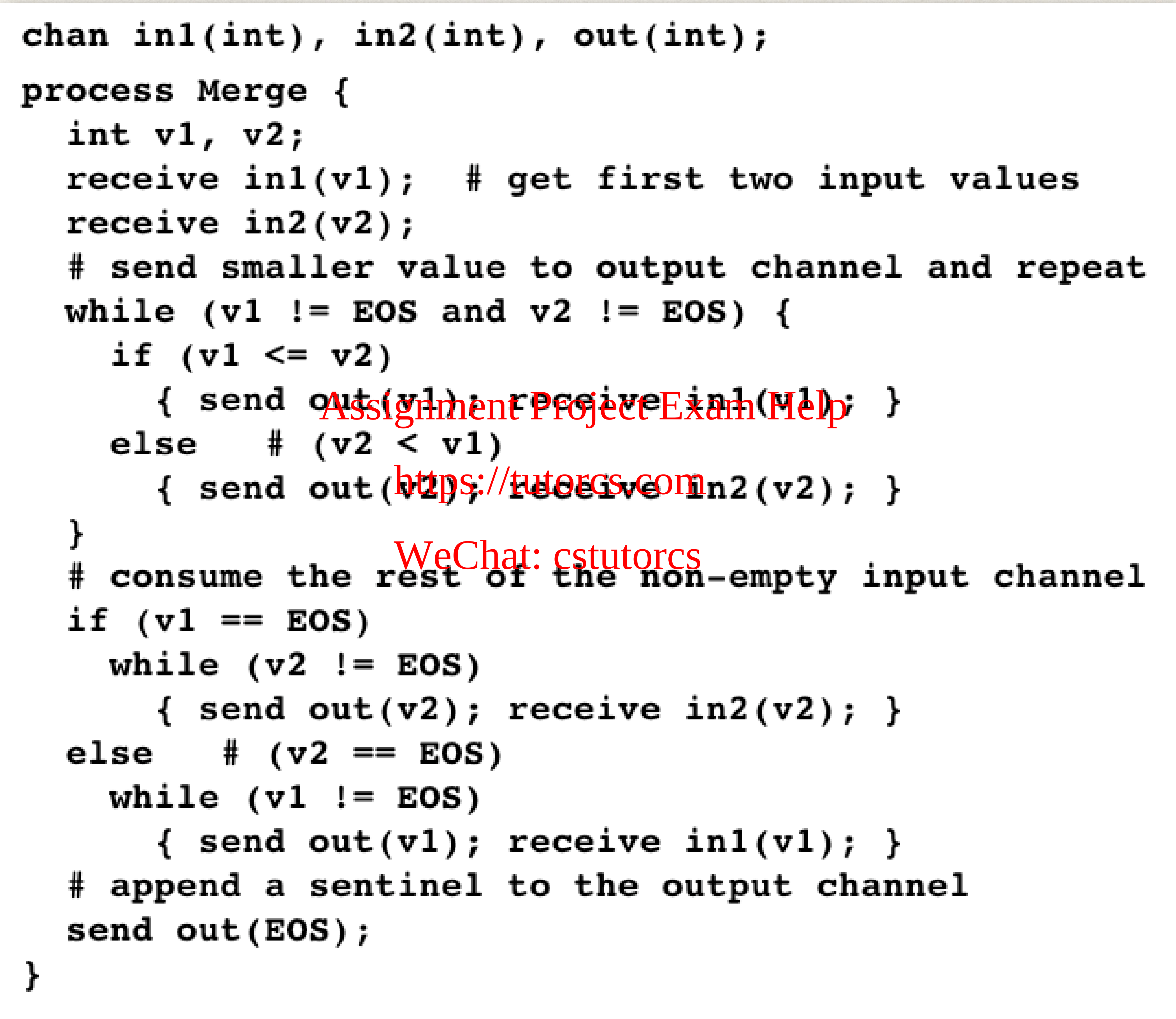
- Filters: one way
- Client/Server: two way as master/slave
- Interacting peers: two way as peers

Filter Process to Assemble Characters



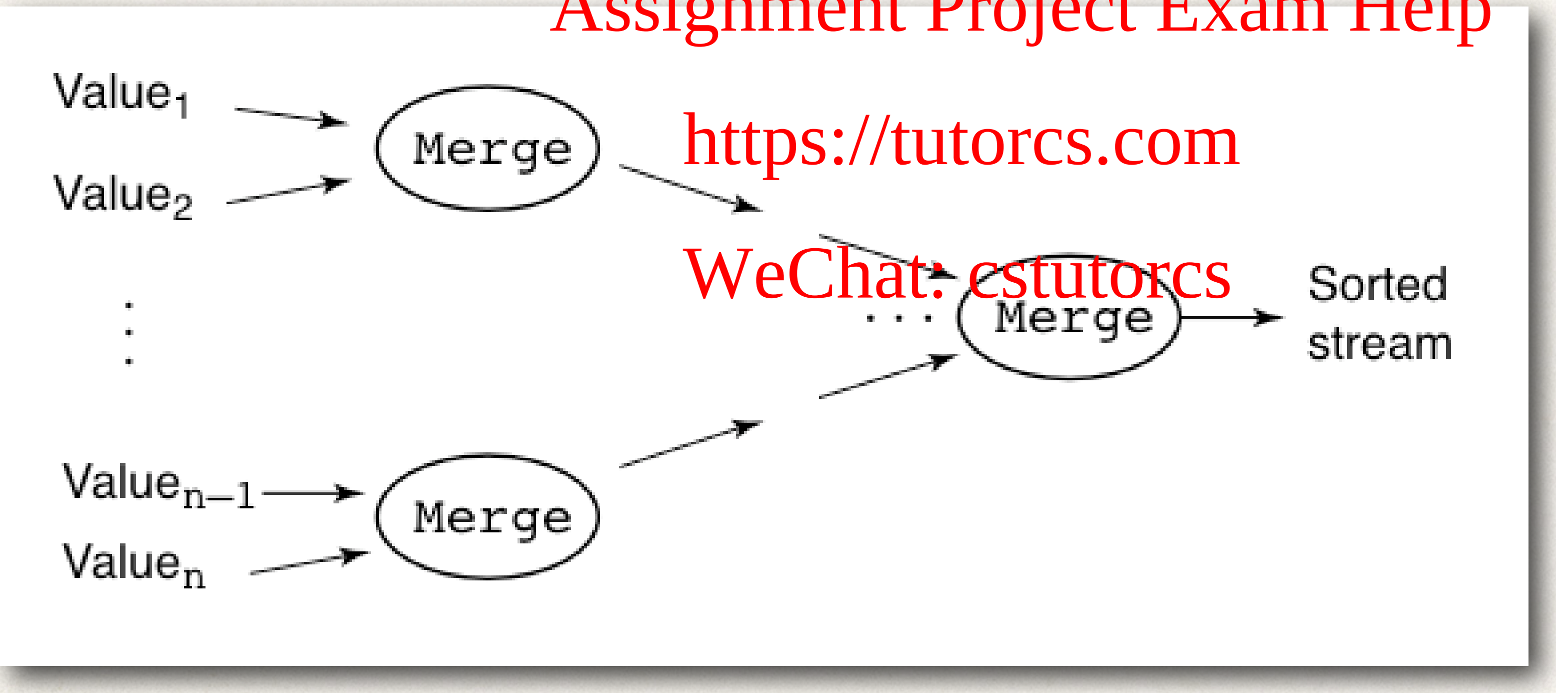
```
chan input(char), output(char [MAXLINE]);

process Char_to_Line {
  char line[MAXLINE]; int i = 0;
  while (true) {
    receive input(line[i]);
    while (line[i] != CR and i < MAXLINE) {
      # line[0:i-1] contains the last i input characters
      i = i+1;
      receive input(line[i]);
    }
    line[i] = EOL;
    send output(line);
    i = 0;
  }
}
```

?



```
chan in1(int), in2(int), out(int);
process Merge {
  int v1, v2;
  receive in1(v1);   # get first two input values
  receive in2(v2);
  # send smaller value to output channel and repeat
  while (v1 != EOS and v2 != EOS) {
    if (v1 <= v2)
      { send out(v1); receive in1(v1); }
    else    # (v2 < v1)
      { send out(v2); receive in2(v2); }
  }
  # consume the rest of the non-empty input channel
  if (v1 == EOS)
    while (v2 != EOS)
      { send out(v2); receive in2(v2); }
  else    # (v2 == EOS)
    while (v1 != EOS)
      { send out(v1); receive in1(v1); }
  # append a sentinel to the output channel
  send out(EOS);
}
```

Merge Process and Sorting Network


Value$_1$
Value$_2$
Merge
Merge
Merge
Value$_{n-1}$
Value$_n$
Sorted stream