Distributed Programming

- Processes don't share memory.
- May be connected by arbitrary network.
- *Message-passing primitives* provide means of communicating.
  - Blocking or non-blocking
- Processes share *channels* over which messages are passed (`send` and `receive`).
  - Global, receiver specific, or sender & receiver specific.
  - One or two way.

Distributed Paradigms

Filter Data translator— Read input stream, write to output stream.

Client Active (triggering) process— Request service, often wait for response.

Server Reactive process— Wait for request, respond.

Peer Co-operating process.

Asynchronous Message Passing

```
chan ch(type1 v1, type2 v2 ...);

send ch(x1, x2 ...);

receive ch(y1, y2 ...);

empty(ch);
```

- Channels are unbounded queues (model as sequences).
- Non-blocking `send`.
- Blocking `receive`.

Since `receive` is the only blocking call, deadlock can only occur there (when channel is empty — use `empty` to check).

Filter: Mergesort

Problem: Sort a list of values

Filter Process:

- `receive` two sorted lists from two channels
- `send` a sorted combined list to another channel

Solution: Network filters in a tree structure

Mergesort

```
chan in1(int), in2(int), out(int);

process Merge {
  int v1, v2;
  receive in1(v1);
  receive in2(v2);

  while (v1 != EOS and v2 != EOS) {
    if (v1 <= v2) {
      send out(v1); receive in1(v1);
    } else {
      send out(v2); receive in2(v2);
    }
  }
  if (v1 == EOS) {
    while (v2 != EOS) {
```



```
      send out(v2); receive in2(v2);
    }
  }
  if (v2 == EOS) {
    while (v1 != EOS) {
      send out(v1); receive in1(v1);
    }
  }
  send out(EOS);
}
```


Client-Server: Monitor

Simulating a monitor using AMP.

```
chan request(int clientID, op_kind, arg_type);
chan reply[n] (res_type);

process Server { # Montor
  while (true) {
    receive request(clientid, op, args);
    switch (op) {
      case OP1: # monitor methods
        ...
    }
    send reply[clientid](results);
  }
}
```



```
process Client { # Monitor user
  ...
  send request(i, op, args);
  receive reply[i](results);
}
```

Conditions

Each condition c becomes a queue, q_c, local to the server.

`wait`(c) adds `clientID`, op etc. to q_c

`signal`(c) removes front from q_c, sends results

Self Scheduling Disk Server

```
chan request(Request r);
chan reply[n](Results r);

process Disk_Driver {
  Queue pending; # pending requests
  while (true) {
    while (!empty(request) or empty(pending)) {
      receive request(req);
      pending.insert(req);
    }
    pending.getNext(req); # retrieve task to service
    access disk
    send reply[req.Id](results);
  }
}
```


Interacting Peers: Exchanging Values

Task: Determine the largest and smallest value held by processes.

Centralized: Coordinator gathers all, and sends results.

- Asymetric — coordinator does all the work
- $2(n-1)$ messages, n channels

Symmetric: Each sends data to all others, receives from all others, then computes results.

- $n(n-1)$ messages, $2n$ channels

Logical Ring: Recv local max, min from prev; Send local max, min to next; Recv global max, min from prev; Send global max, min to next.

- $2(n-1)$ messages, n channels

AMP in Java – Sockets

- Two-way channels for `Strings`.
- `ServerSocket` – allocates a port for the channel.
- `Socket` – opens a channel on the port.
  - `inputStream`
  - `outputStream`

Server

```
ServerSocket listen = new ServerSocket(0); // any available socket

Socket socket = listen.accept();
BufferedReader from_client =
    new BufferedReader(
      new InputStreamReader(socket.getInputStream()));
PrintWriter to_client =
    new PrintWriter(socket.getOutputStream());
// use socket
//
to_client.close();
from_client.close();
socket.close();
```

Client

```
Socket socket = new Socket(host, port);
BufferedReader from_server =
  new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter to_server = new PrintWriter(socket.getOutputStream());

// use socket

socket.close();
```


Synchronous Message Passing

- Non-buffered communication
- `sync_send` blocks until message is received
- Combined communication and synchronization
- Can be viewed as distributed assignment statement.
- Often reduces concurrency — sender or receiver waiting.
- More prone to deadlock.

Examples

- Pipelined seive of Eratosthenes
  - First number received, p_i, is prime
  - From remaining values, pass on only if $\mathrm{x}\%\mathrm{p}_i \neq 0$
- Heartbeat compare and exchange sort
  - sort my n/k elements
  - Odd rounds: if i is odd, $\mathtt{P}[\mathrm{i}]$ send largest to $\mathtt{P}[\mathrm{i}+1]$, receive from $\mathtt{P}[\mathrm{i}+1]$ its smallest.
  - Even rounds: if i is even, $\mathtt{P}[\mathrm{i}]$ send largest to $\mathtt{P}[\mathrm{i}+1]$, receive from $\mathtt{P}[\mathrm{i}+1]$ its smallest.