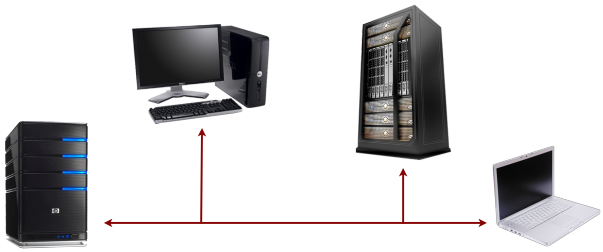
Erlang Solutions Ltd.

Distributed Programming

Distributed Programming

- Distributed Systems
- Erlang Distribution
- Node Connections
- Distributed BIFs
- Net Kernel
- Hidden Nodes
- Firewalls and Erlang Distribution

Distributed Systems



- A distributed system consists of a program running on a collection of loosely coupled processors

Distributed Systems

There are many reasons for building distributed systems

- Performance
- Reliability and fault tolerance
- Accessing remote resources
- Distributed applications (possibly heterogeneous)
- Extendibility and Scalability

Erlang Distribution: example

```
$ erl -sname foo
...
(foo@node)1> S = self().
<0.37.0>
(foo@node)2> spawn('bar@node', fun() -> S ! node() end).
<5827.42.0>
(foo@node)3> flush().
Shell got 'bar@node'
```

`fun() -> S ! node() end` runs on **bar@node**

```
$ erl -sname bar
...
(bar@node')1>
```


Erlang Distribution

- A **node** is an executing instance of the Erlang VM
- A node is said to be **alive** if it can communicate with other nodes
- Nodes that are alive have a unique name on that host
- **Name@Host** is the identifier of an Erlang node
- A distributed Erlang node is started using the **-name** and **-sname** flag
  - In Windows, use the DOS prompt to run the command

Erlang Distribution

- Erlang nodes can communicate only if they share the same cookie or if they know each other's cookies
- Simple on trusted networks and local clusters
- Harder on untrusted networks

Erlang Distribution: starting

```
cesarini@caen> erl -sname foo  -setcookie hello
will start a node with the short name foo@caen

cesarini@caen> erl -name foo
will start a node with the long name foo@caen.ericsson.se
```

- Nodes started with long names may not communicate with nodes using short names
- Nodes using short names do not require a DNS server
- If no cookie is defined when starting up the system, then one will be created and stored in the **~/.erlang.cookie** file

Messages: example

```
(foo@node)4> spawn('bar@node', s, s, []).
<5827.47.0>
(foo@node)5> {server, 'bar@node'} ! {hi, self()}.
{hi, <0.37.0>}
(foo@node)6> flush().
Shell got hi
```

hi sent from **foo@node** to **bar@node**, which replies with **hi**

```
-module(s).
-export([s/0]).

s() -> register(server, self()), l().

l() -> receive {M, Pid} -> Pid ! M end, l().
```


Erlang Distribution: messages

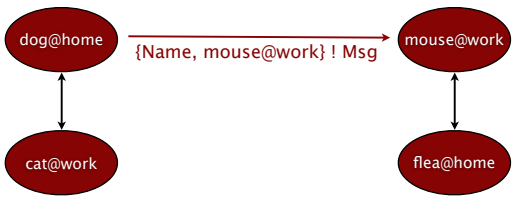
```
Pid ! Message
{Name, Node} ! Msg
```

- Send messages to processes on remote nodes transparently using the **!** construct
- Messages will be delivered in the same order they are sent.
- Only difference is that the remote node might go down.
- WIth little changes (or doing it right from the start), programs running on a single processor can easily become distributed

Node Connections

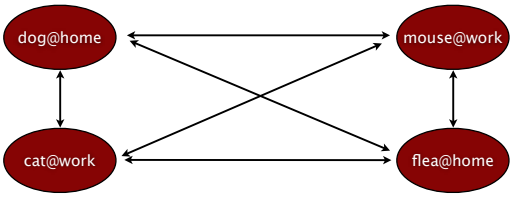
- Nodes are loosely connected to each other
- Connections are explicitly set up by the run time system when a node is first referred to
  - They are not set up by the programmer
- Information on new nodes is shared among the connected nodes
- Nodes monitor each other checking if their peers are alive
- Nodes can come and go dynamically in a similar manner as processes

Node Connections



- Dog is connected to Cat and Mouse to Flea
- A process in Dog sends a message to a registered process in Mouse

Node Connections



- Dog sets up a connection to mouse and is informed of flea
- Dog tells cat of mouse and flea, and tells mouse of cat
- This goes on and they all get happily connected

Distributed Example

```
call(Message, ServerNode) ->
    {frequency, ServerNode} ! {request,self(),Message},
    receive
        {reply, Reply} -> Reply
    end.
reply(Pid, Message) ->
    Pid ! {reply, Message}.
```

- Recall the resource server call? We can now receive requests to safely allocate resources from other nodes

Distributed Example

```
call(Message, ServerNode) ->
    monitor_node(ServerNode, true),
    {resource, ServerNode} ! {request,self(),Message},
    receive
        {nodedown, ServerNode}->
           {error, node_down};
        {reply, Reply} ->
           monitor_node(ServerNode, false),
           Reply
    end.
```

The node might go down before we have received the reply!

Distributed BIFs

- `Pid = spawn(Node, Mod, Func, Args)`
- `Pid = spawn_link(Node, Mod, Func, Args)`
- `monitor_node(Node, Flag)`
- `MyName@Host = node()`
- `[Node|Nodes] = nodes()`
- `Node = node(Item)` % Item is a PID, Ref or Port
- `disconnect_node(Node)`
- `erlang:set_cookie(Node, Cookie)`

Net Kernel

- The net_kernel is an Erlang process that coordinates operations in a distributed node
- BIFs such as **spawn/4** are converted by net_kernel to messages, and sent to the net_kernel on the remote node
- Handles authentication and rejects bad cookies
- You can change the net_kernel with a user-defined process giving a specific behaviour
  - Changing the authentication scheme,
  - not allowing remote nodes to spawn processes, etc

Hidden Nodes: example

```
erl -sname a
...
(a@node)1>
 net_kernel:connect('b@node'),
 net_kernel:connect('c@node').
true
(a@node)1> nodes()
['b@node']
(a@node)1> nodes(hidden)
['c@node']
(a@node)1> nodes(connected)
['b@node','c@node']
```

```
erl -sname b
...
(b@node)1> nodes().
['a@node']
(b@node)2> nodes(connected)
['a@node']
```

```
erl -sname c -hidden
...
(c@node)1> nodes().
[]
(c@node)2> nodes(hidden).
['node']
```

Hidden Nodes

```
erl -sname foo -hidden
will start a hidden node
```

- Hidden nodes have to be connected individually
- Typically used for Operation and Maintenance
- Can be used to create hierarchies of nodes

Firewalls and Erlang Distribution

- Poke a hole in the firewall, port 4369
  - Erlang Port Mapper Daemon (EPMD), listens to incoming connection requests
- Poke a hole in the firewall, ports 9100 to 9105
  - Set undocumented kernel variables:
    - **application:set_env(kernel, inet_dist_listen_min, 9100)**
    - **application:set_env(kernel_inet_dist_listen_max, 9105)**
    - Forces Erlang to use ports 9100 to 9105 for distribution
- Tunnel over using SSH
- Roll your own distribution handler

Distributed Programming

- Distributed Systems
- Erlang Distribution
- Node Connections
- Distributed BIFs
- Net Kernel
- Hidden Nodes
- Firewalls and Erlang Distribution