Agents from Object
Agents

- We made Stateful Agents from
  - State-transition functions
    - State x Message $\rightarrow$ State
  - Ports

- Now we make agents out of objects
Object-based Agents

- A class will define the agent behavior
  - The local attributes will be the local state variables
  - The method heads will be the messages
  - The methods definitions will be the message handlers (corresponds to the transition function)

- Each object will have an associated thread that reads message from the stream and invoke the method of the object
Example Agent Behavior

class BankAccount
    attr balance:0
    meth init(N) balance := N end
    meth withdraw(N)
        if @balance >= N then
            balance := @balance - N
        end
    end
    meth deposit(N)
        balance := @balance + N
    end
    meth balance($)
        @balance
    end
    meth otherwise(M) {Browse M} end
end
Example: BankAccount Agent

BP = {NewAgent BankAccount init(0)}
{BP deposit(100)}
{BP deposit(50)}
{BP withdraw(50)}
{Browse {BP balance($)}}

- Agent Sending message m1(...) to itself
  
  meth m(...)
  
  {@this m1(...)}

  ...

end
Circulating Ball Agents

- Agents that send a ball among each other
Example: Circulating Ball Agents

class Ball

attr n:0 i:0 agents:nil

meth init(N I Agents)
    n := N i := I agents := Agents
end

meth ball
    if @n > 0 then
        {Inspect gotBall(agent(@i) @n)}
        n := @n-1 {Delay 1000}
        {{Nth @agents (@i mod {Length @agents}) + 1} ball}
    end
end
end
Example: Starting the whole thing

Bs = for I in 1..4 collect: C do
    {C {NewAgent Ball init(10 I Bs)}}
end
{Bs.1 ball}

gotBall(agent(1) 10)
gotBall(agent(2) 10)
gotBall(agent(3) 10)
gotBall(agent(4) 10)
gotBall(agent(1) 9)
gotBall(agent(2) 19)
The new NewAgent Abstraction - Simplified

- This abstract is simple

```lambdabang
fun \{NewAgent V I\}
    if \{Value.type V\} == procedure then
        \{NewAgentF V I\}
    elseif \{Value.type V\} == 'class' then
        \{NewAgentC V I\}
    else
        V
    end
end
```
The *new* NewAgent Abstraction - Simplified

- This abstract is simple

```plaintext
fun {NewAgentC Class Init}
  P Obj = {New Class Init} in
  thread S in
    P = {NewPort S}
    for M in S do {Obj M} end
  end
end
proc {$ M} {Send P M} end
end
```
Sending Messages to Yourself

- In objects invoking a method locally
  - `{self m(... ) }`

- So how can an agent send a message to itself
  - We need to store the agent identity as part of its object state
  - Accessed by: `@this`
  - Sending a message by: `{@this m(... ) }`
The *new* NewAgent Abstraction with @this

- This abstract is simple

```plaintext
fun {NewAgentC Class Init}
  P
  Agent = proc {$ M} {Send P M} end
This = class $ attr this: Agent end
MyClass = class $ from Class This end
Obj = {New MyClass Init}

in
  thread S in ... end
Agent
end
```
Summary

- Concurrent Agents can be built out
  - Objects
  - Transition state-functions
- We used abstractions and encapsulation
  - Ports
  - Implementation is not observable from outside
- Agents communicate by messages
- Internal protocol sessions can use data-flow variables as single-value channels
Further Readings

Programming Erlang
Software for a Concurrent World
Second Edition
Joe Armstrong
Edited by Susanah Davidson Pflaizer

Akka Concurrency
Building reliable software in a multi-core world
artima
Derek Wyatt