Robert Virding

Principle Language Expert
at Erlang Solutions Ltd.

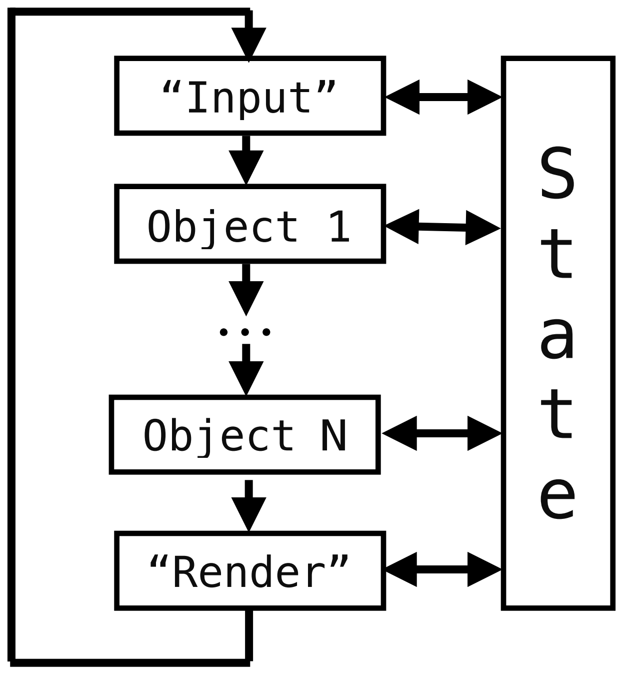
Erlang Solutions Ltd.

Synchronising Game Components

Overview

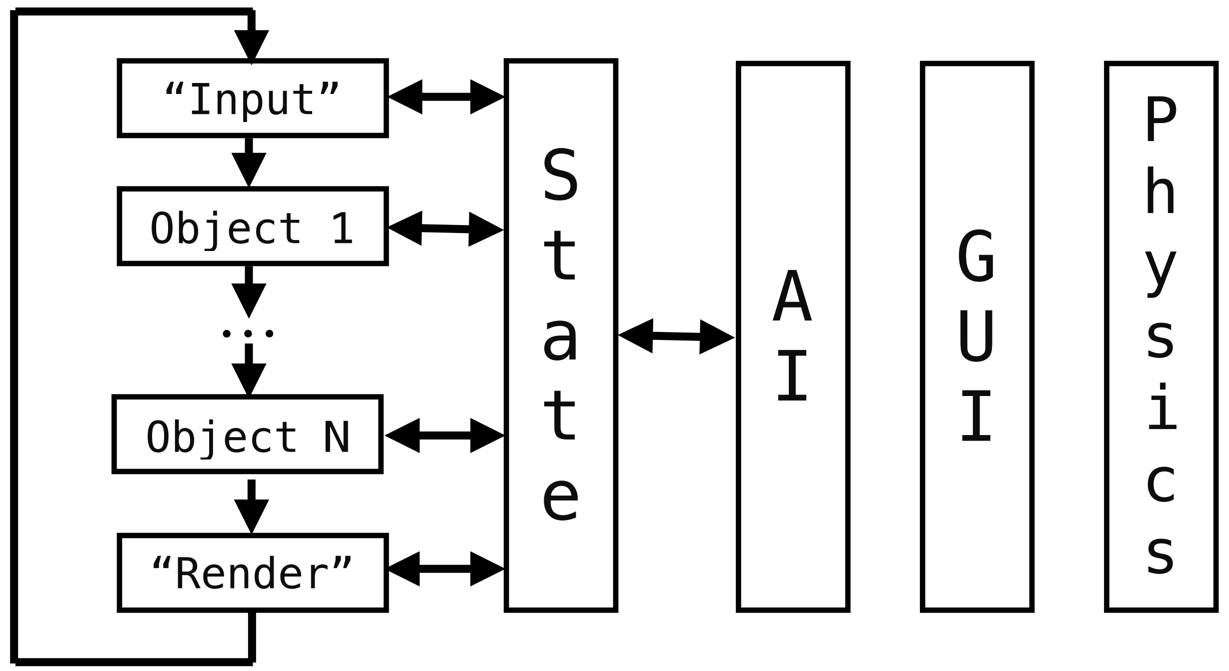
- The classic way
- A slightly improved classic way
- The scalability problem
- A way of solving the problem
- Messages and/or shared state
- A solution
- All is not rosy

The classic way



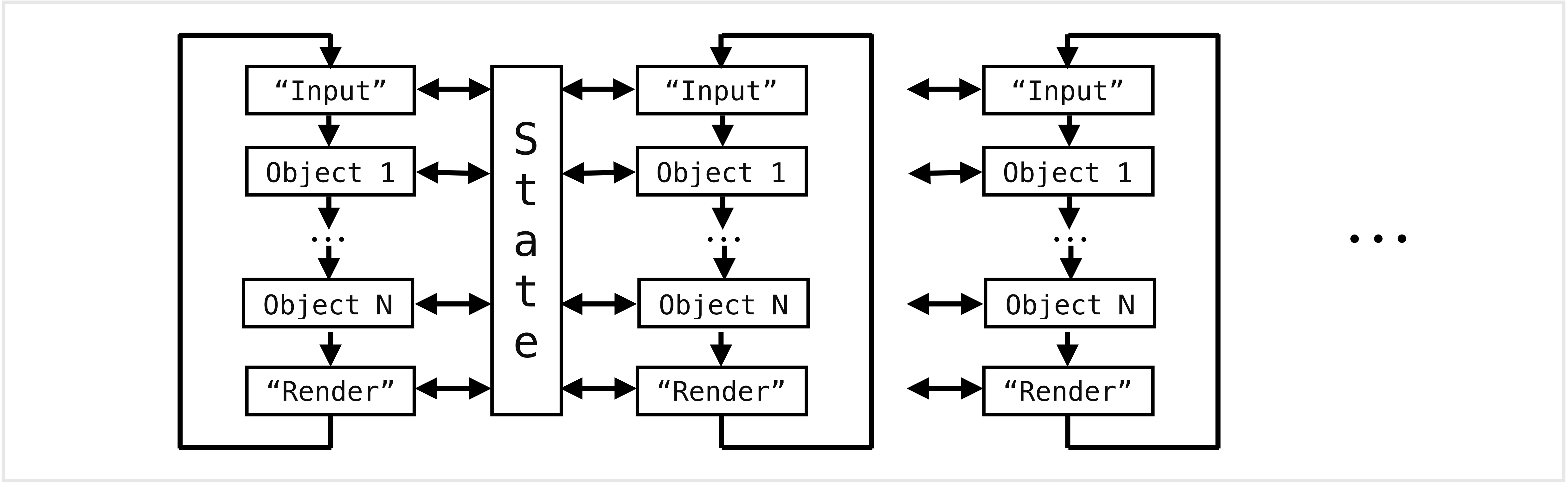
+ Complete control over the system
  + Complete control of state
  + Complete control of timing
– Problems with execution time
– Does not naturally scale to parallel systems

A slightly improved classic way



+ Slightly better parallelism
+ Complete control over the system
– Problems with execution time
– Only limited parallelism

The scalability problem



+ Scalable
– We have lost complete control over the system
– How do we synchronise?
– How do we communicate between the loops?

The scalability problem

- Sharing mutable state does not scale
  - Not feasible for central tables with state
  - The more parallelism we have the worse it gets
- Need another way to communicate and synchronise
  - Messages

Messages and/or shared state

- Synchronise with messages, share data (Go)
  - Difficult to get right and not safe
- STM
  - Doesn’t scale well with mutable data
  - Can have memory locality problems
  - GC
- Pure copying messages

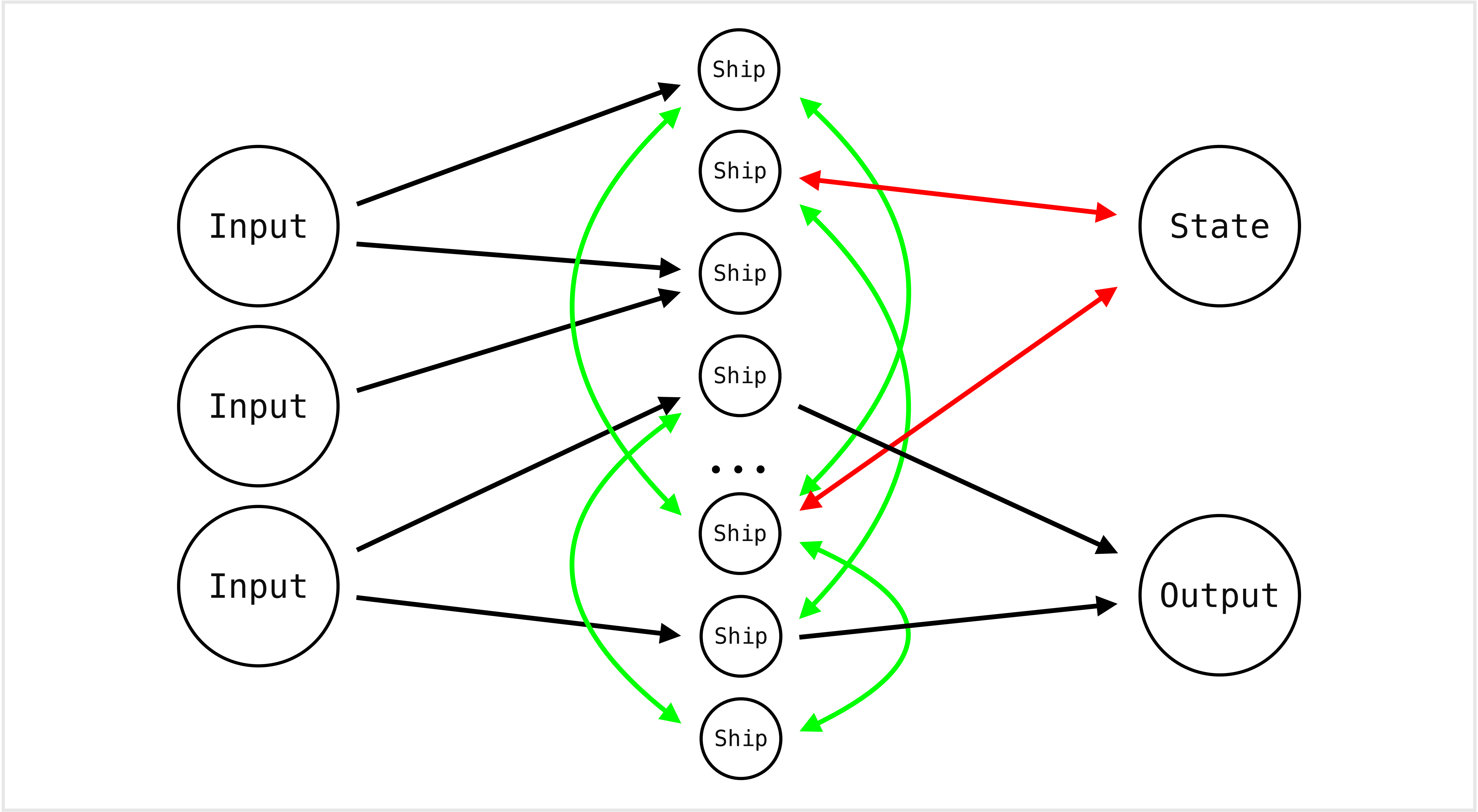
A solution

- Go fully parallel
- Everything in processes
- Communicate/synchronise with messages
- Reduce all central state to a minimum
- This scales quite well
  - The shared mutable state limits this
- Processes can be implemented in different languages
  - Interface is through messages

A solution: example

- Concurrent space ships
- Each an Erlang process
- All communication using messages
- Very limited shared state (which ships in a sector)
  - Managed by a process accessed with messages
- Ship logic in Lua (and Erlang)
- “Devices” behave like processes
  - Receive messages from input
  - Send messages to control output

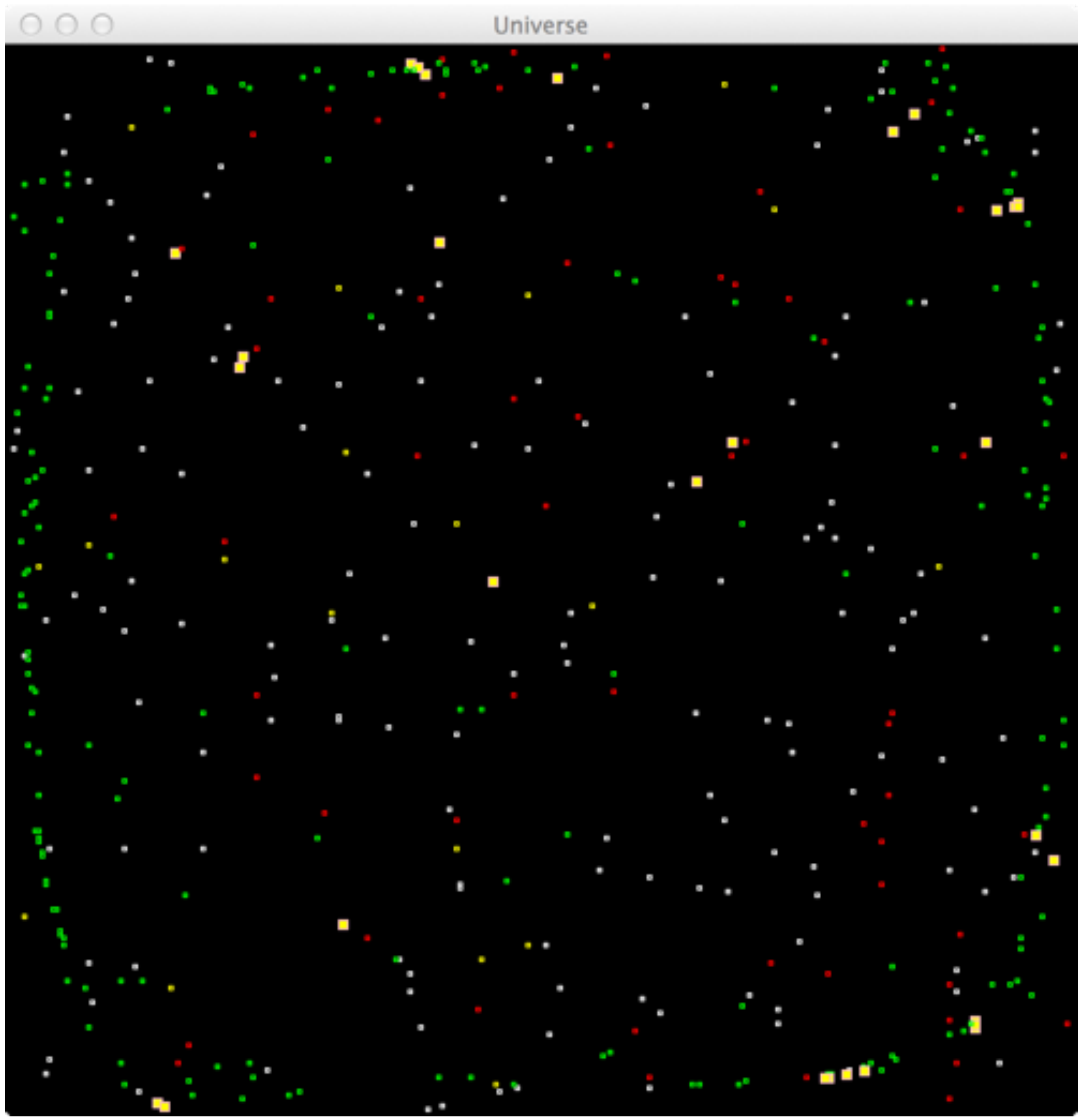
A solution: example



A solution: demo

- Start up the system
- Change ship code on the fly
- Show interacting ships
- Show ships programmed in different languages

A solution: demo



A solution: example code

```
local function move(x, y, dx, dy)
   local nx,ny,ndx,ndy = move_xy_bounce(x, y, dx, dy,
                                     universe.valid_x, universe.valid_y)
   -- Where we were and where we are now.
   local osx,osy = universe.sector(x, y)
   local nsx,nsy = universe.sector(nx, ny)
   if (osx ~= nsx or osy ~= nsy) then
      -- In new sector, move us to the right sector
      universe.rem_sector(x, y)
      universe.add_sector(nx, ny)
      -- and draw us
      display.set_ship(type, colour, nx, ny)
   end
   return nx,ny,ndx,ndy
end
```

A solution: example code

- Attack ships communication
- Output messages for video and sound
- Input messages for controlling ship

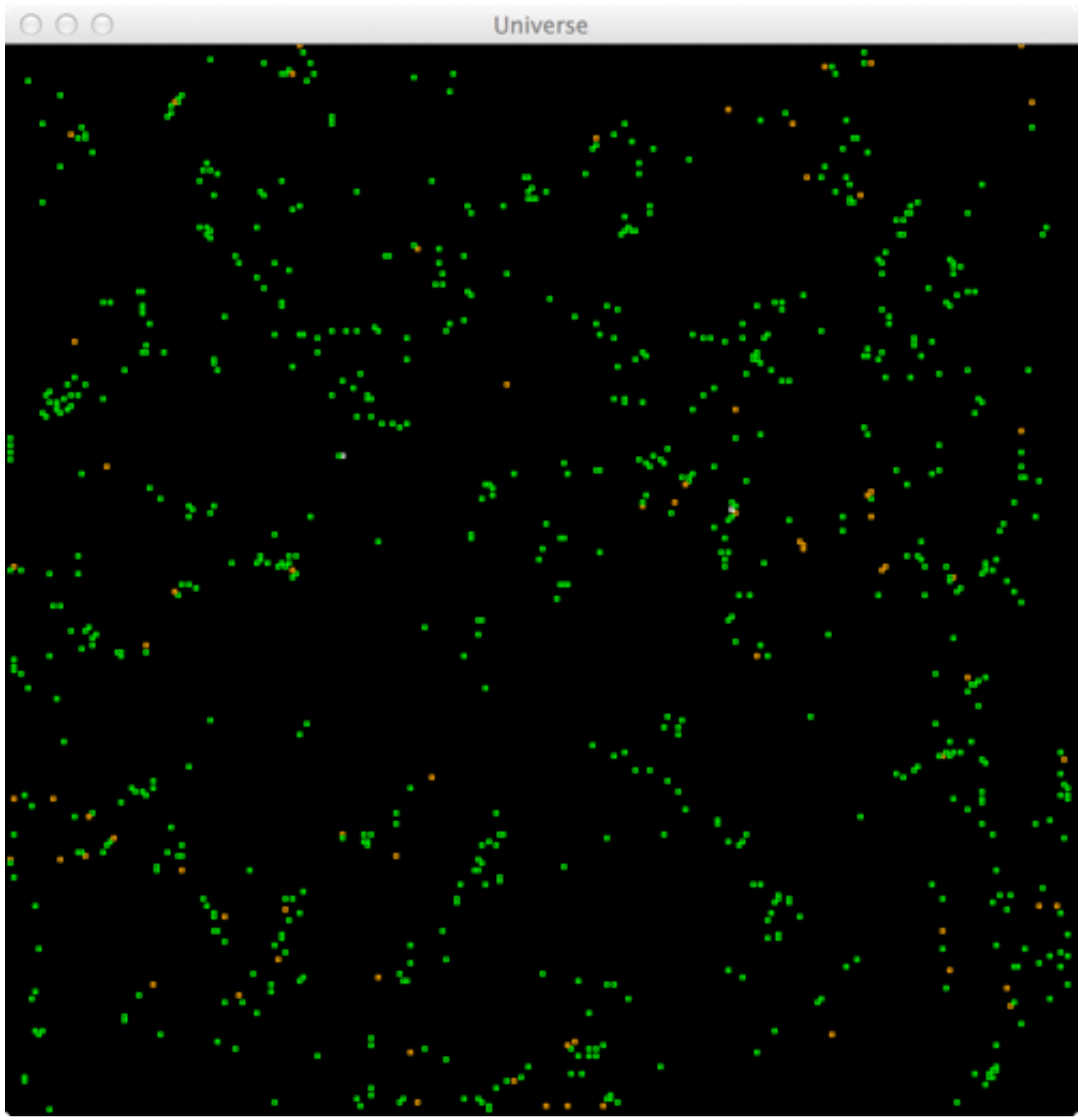
A solution: example code

```
local function zap_ships(osx, osy, nsx, nsy)
   local lsx,lsy,rsx,rsy = move_lr_sectors(osx, osy, nsx, nsy)
   local f = universe.get_sector(nsx, nsy)
   if (f and f ~= me) then      -- Always zap ship in front
      ship.zap(f, power)
   end
   f = universe.get_sector(lsx, lsy) or
      universe.get_sector(rsx, rsy)
   if (f and f ~= me) then      -- Zap ship either left or right
      ship.zap(f, power)
   end
end
```

All is not rosy

- Some traditionally standard things cause problems
- Synchronous communication is a killer
  - It blocks the caller
- Must be non–blocking
  - Use asynchronous communication

All is not rosy: demo



The Erlang concurrency model scales!

Thank you

robert.virding@erlang-solutions.com

@rvirding