

Sphactor

actor model concurrency for creatives



expertise centre creative technology



HKU expertisecentrum creatieve technologie



Background

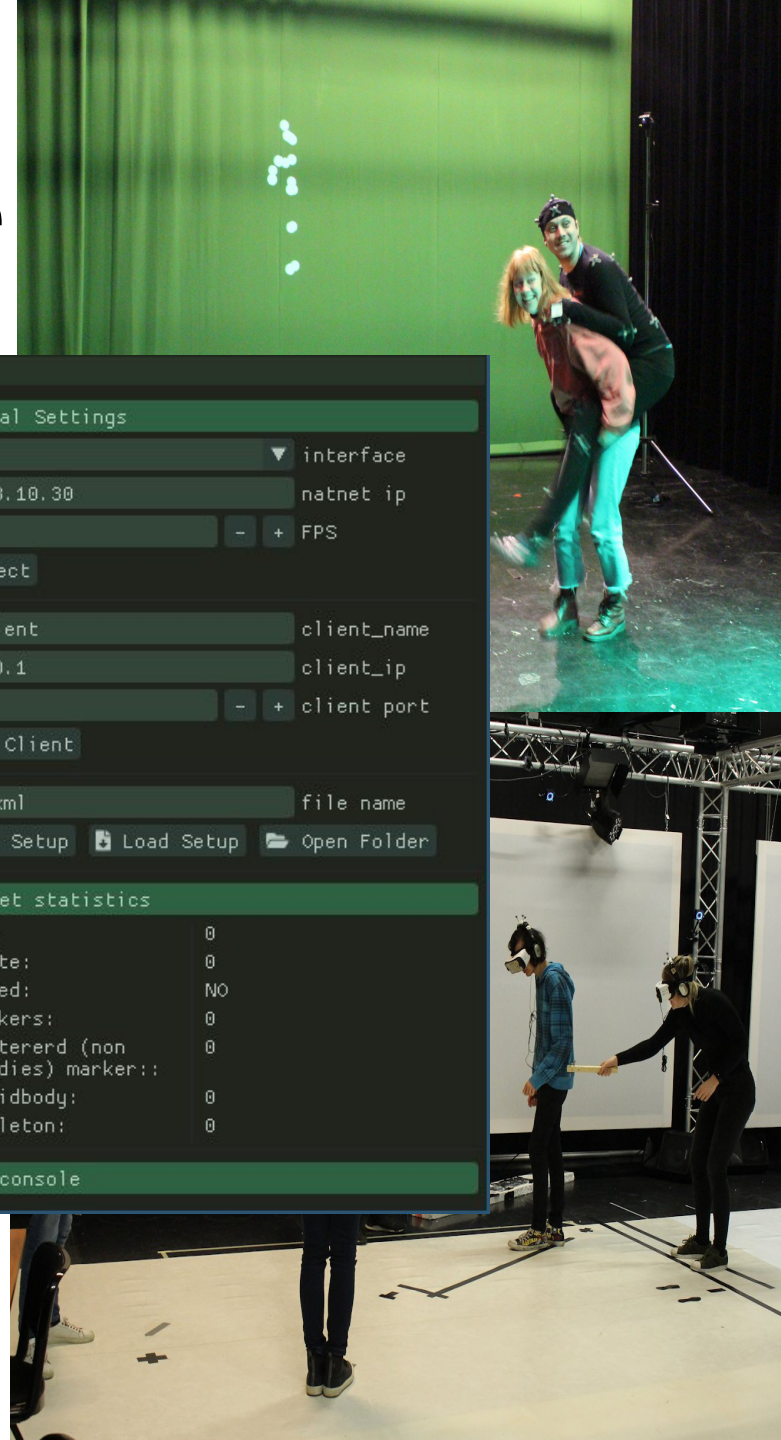
- >3900 students
- one of the largest culture-oriented institutes in Europe
- Expertise Centre Creative Technology

the art of



HKU



Context: Motion Capture

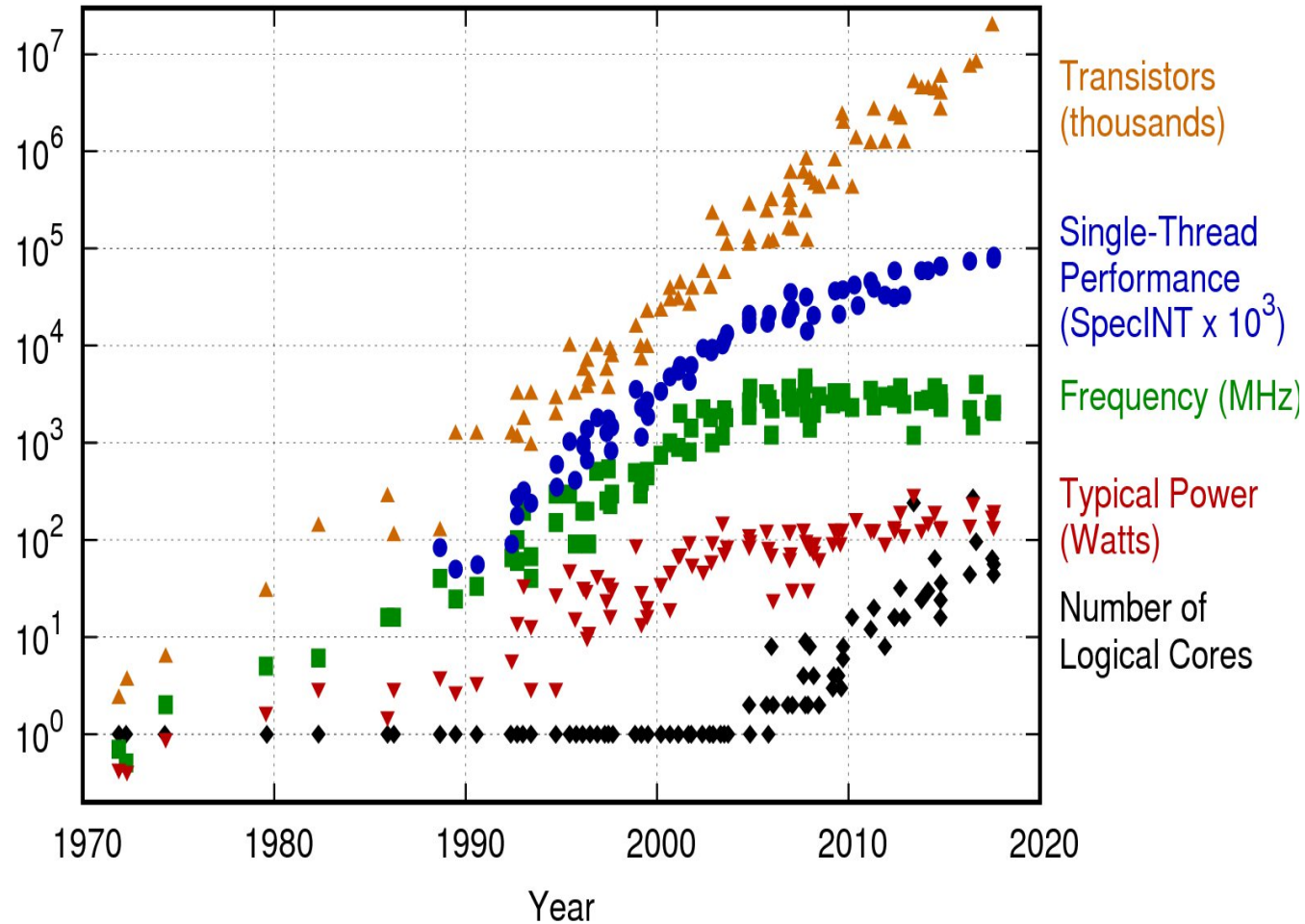


Programming Didactics

User	Operator	Scripter	Developer
Consuming technology	Combining technologies	Lego-ing with technologies	Final boss
	 <small>Photo by Tropical Press Agency/Getty Images</small>		

Multi Core?

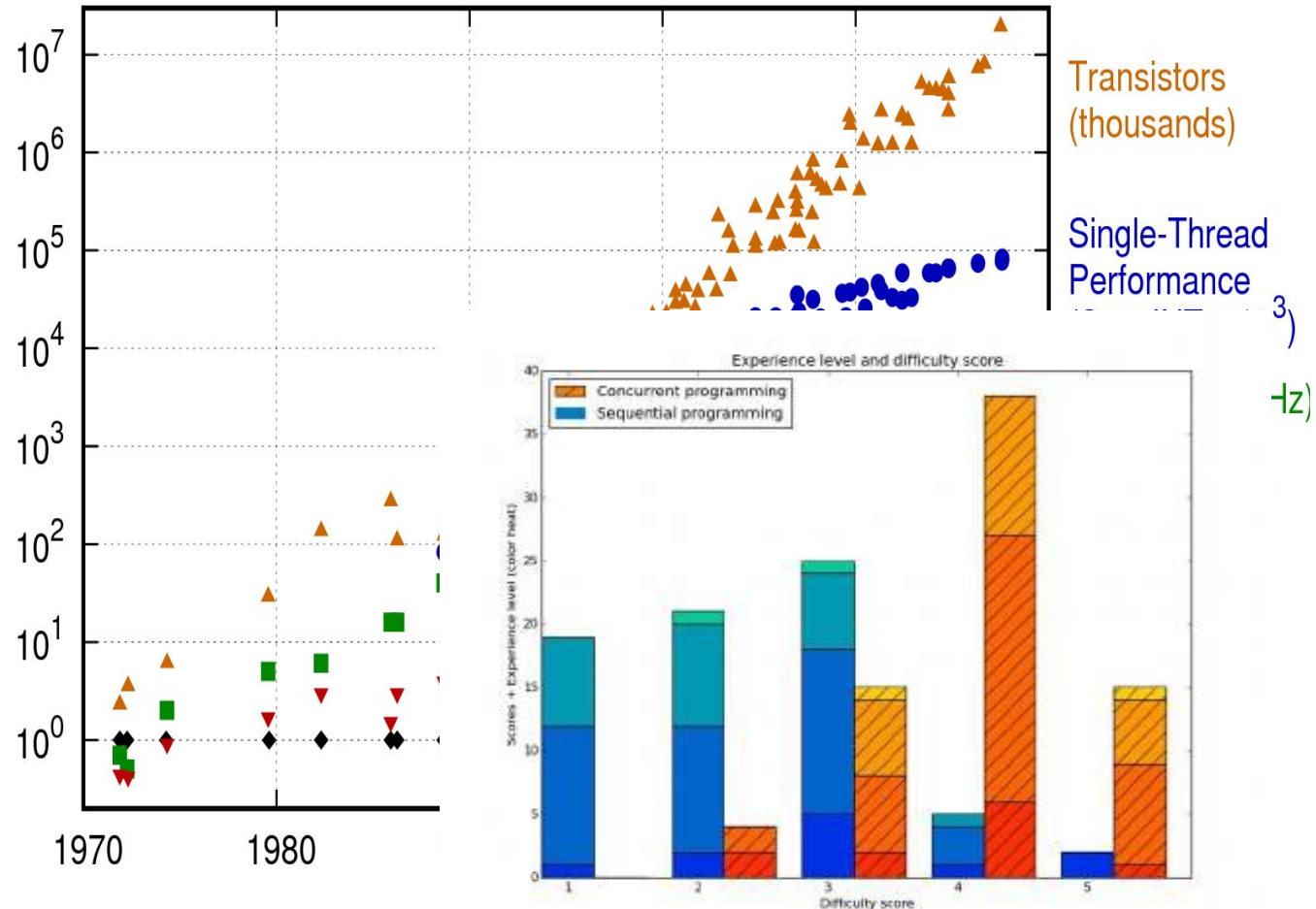
42 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Multi Core?

42 Years of Microprocessor Trend Data

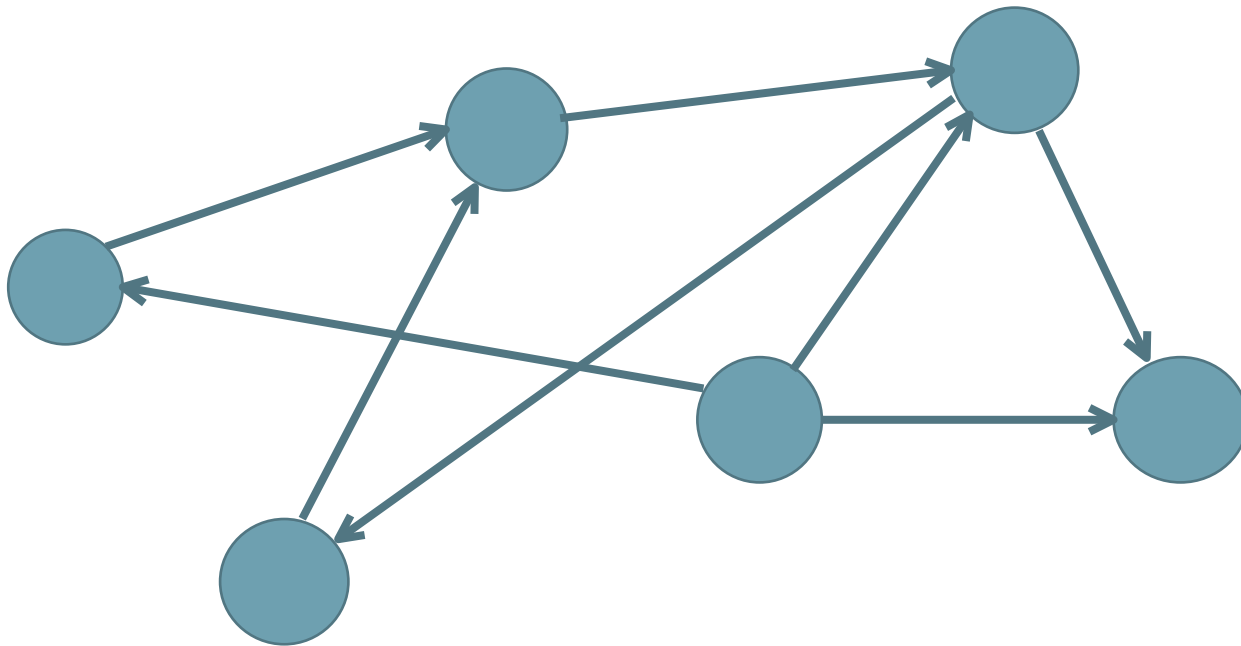


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
 New plot and data collected for 2010-2017 by K. Rupp

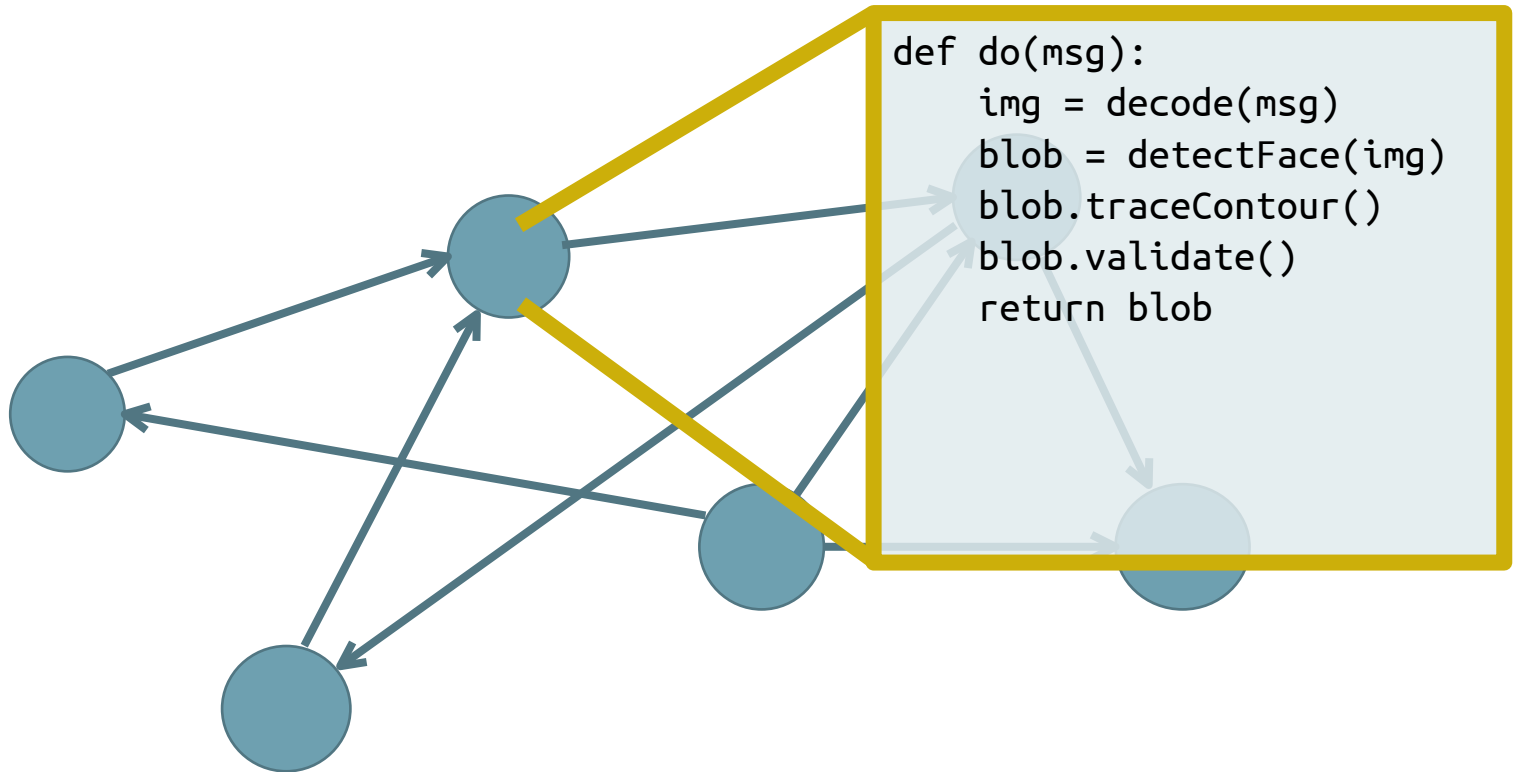
Actor Model

- message passing
- defined 1973 Hewitt
- 80's -> erlang -> whatsapp
- actor == sequential program sending and receiving
- actors are simple

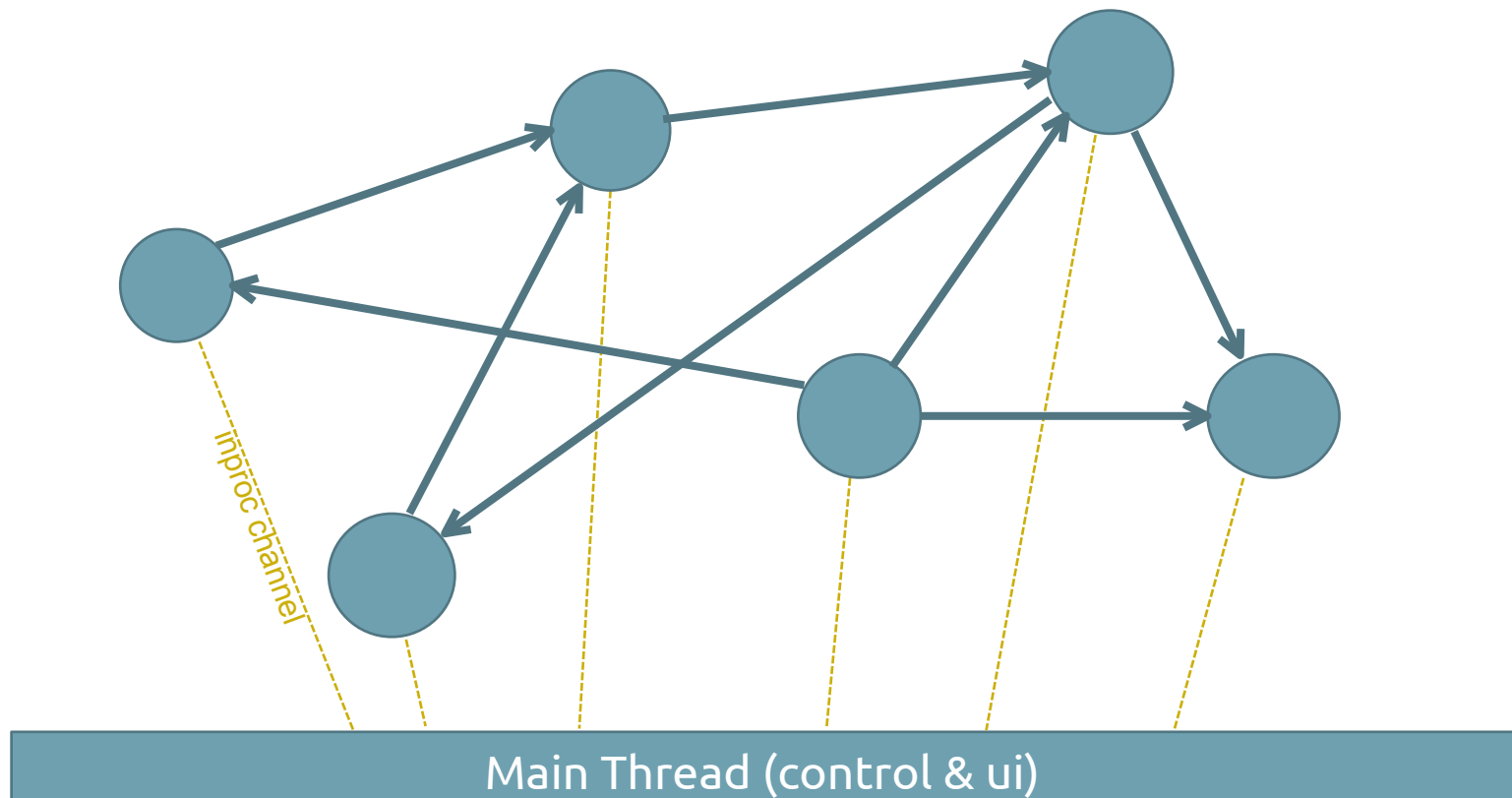
Actor Model - sphactor



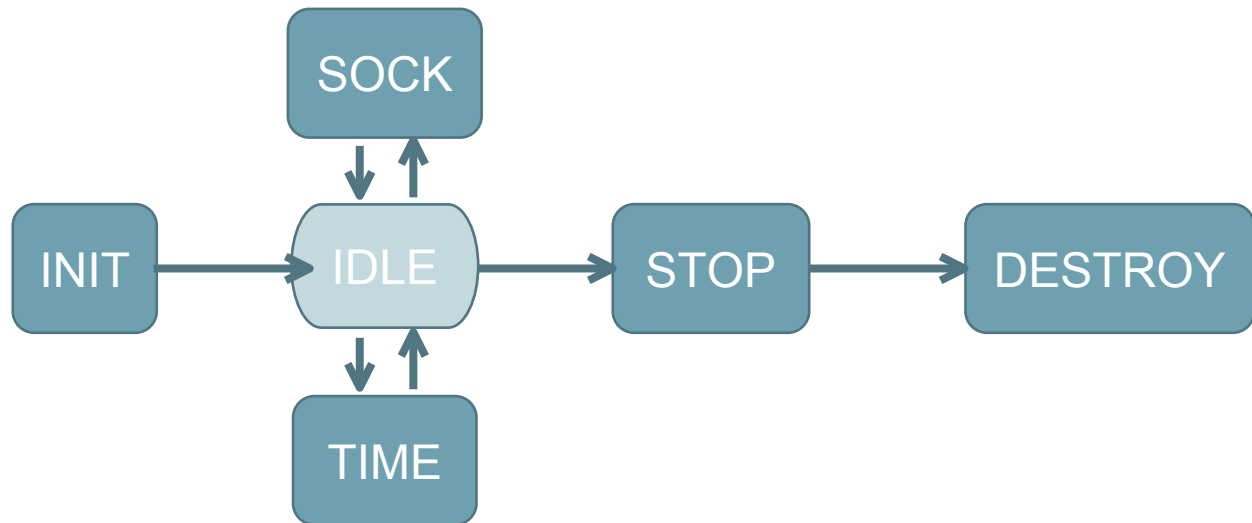
Actor Model - sphactor



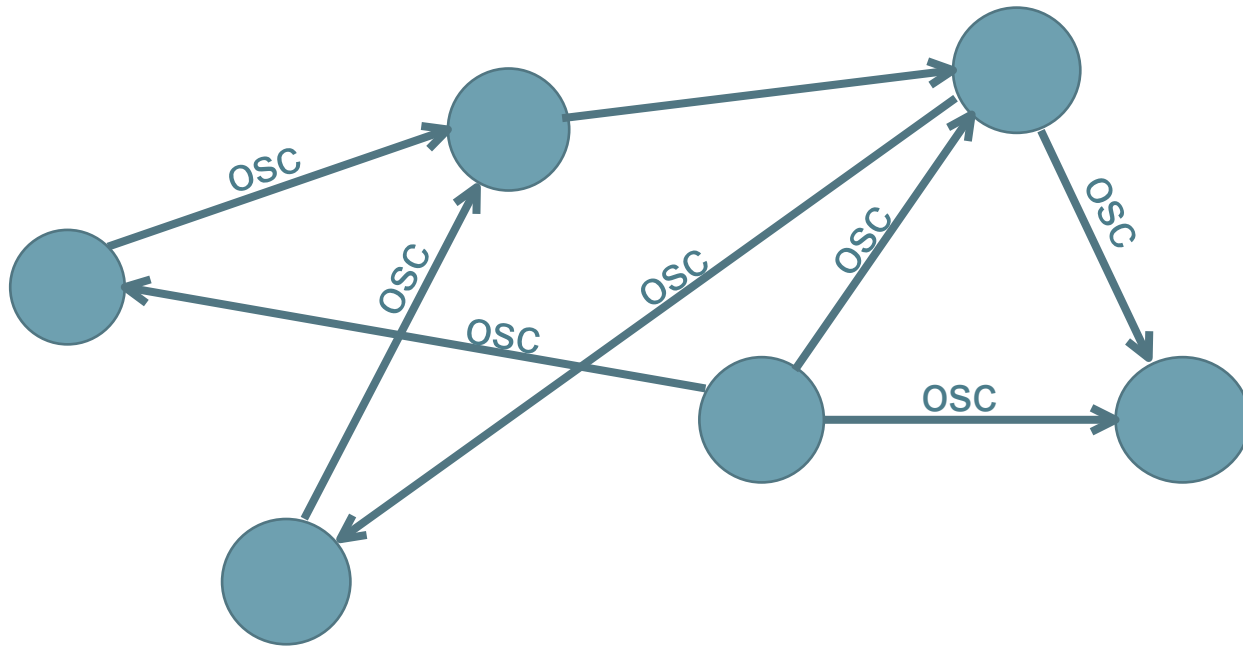
Actor Model - sphactor



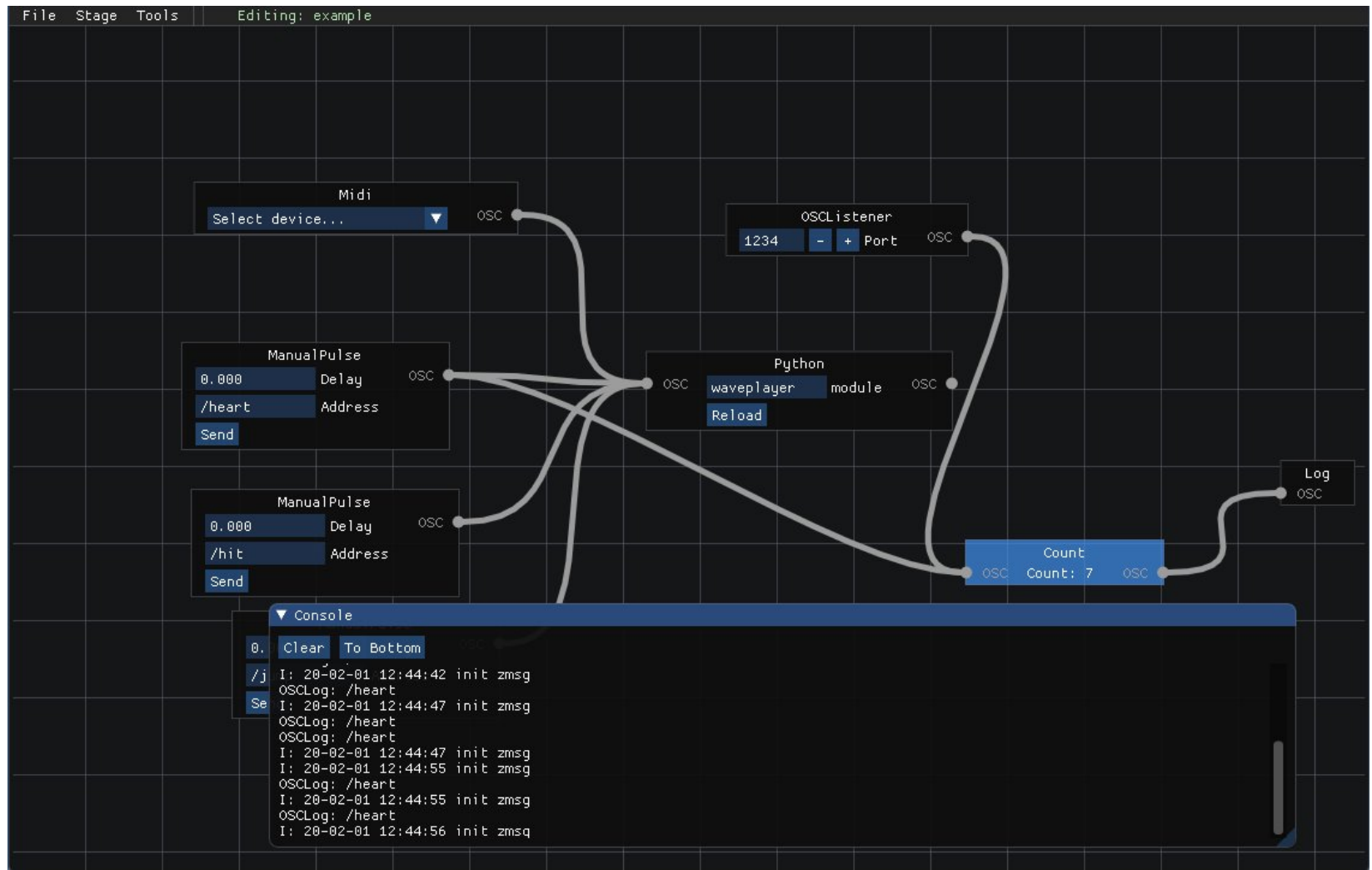
Actor states



gazeboosc



Gazebosc demo



Gazebo Python Actor

```
import sph
# pip install python-osc
from pythonosc import osc_message_builder

class tester(object):

    def handleMsg(self, msg, type, name, uuid, *args, **kwargs):
        # just pop the first string and return the rest
        t = msg.popstr()
        print("Message received: {}".format(t) )
        msg = osc_message_builder.OscMessageBuilder(address="/Hello")
        msg.add_arg("hello from python")
        osc = msg.build()
        return osc.dgram
```

Gazebo C++ Actor

```
#include "libsphactor.h"
```

```
class Test {  
public:  
    zmsg_t *  
    handleMsg( sphactor_event *ev ) {  
        char *cmd = zmsg_popstr(event->msg);  
        zsys_info("Cpp actor %s says: %s", event->name, cmd);  
        // if there are strings left publish them  
        if ( zmsg_size(event->msg) > 0 ) {  
            return event->msg;  
        }  
        else {  
            zmsg_destroy(&event->msg);  
        }  
        return nullptr;  
    }  
};
```

Up & Running

```
#include "libsphactor.h"
```

```
int main() {
```

```
    Test a = Test();
```

```
    sphactor_t *actora = sphactor_new(a, "hello-a", nullptr);
```

```
    // actora is running, request its name
```

```
    const char *name = sphactor_ask_name(actora);
```

```
    assert( streq(name, "hello-a"));
```

```
    ...
```

```
    // connect it another actor
```

```
    sphactor_ask_connect(actora, sphactor_ask_endpoint(actorb));
```

```
    ...
```

```
    // cleanup
```

```
    sphactor_destroy(&actora);
```

```
    return 0;
```

```
}
```


API

sphactor API (main thread)

sphactor_new
 (handler, args, name, uuid);
sphactor_destroy(self);

sphactor_ask_endpoint(self);
sphactor_ask_connect
 (self, endpoint);
sphactor_ask_disconnect
 (self, endpoint);
sphactor_ask_set_timeout
 (self, timeout);

sphactor_actor API (actor thread)

sphactor_actor_poller_add
 (self, fd, handler);

Under the hood

ZeroMQ / czmq



Dear ImGui

SDL

Liblo

Embeds Python

Wrapping up

- Actor Model Framework aimed at simplicity
- Early stage so all the cliché todo's
- Try it, help us out. Especially if:
 - want a tool so you can play with technology
 - familiar with file descriptors/reactor pattern

Sphactor

actor model concurrency for creatives

<https://github.com/hku-ect/libsphactor>

<https://github.com/hku-ect/gazebosc>

Background paper: see FOSDEM event link



expertise centre creative technology