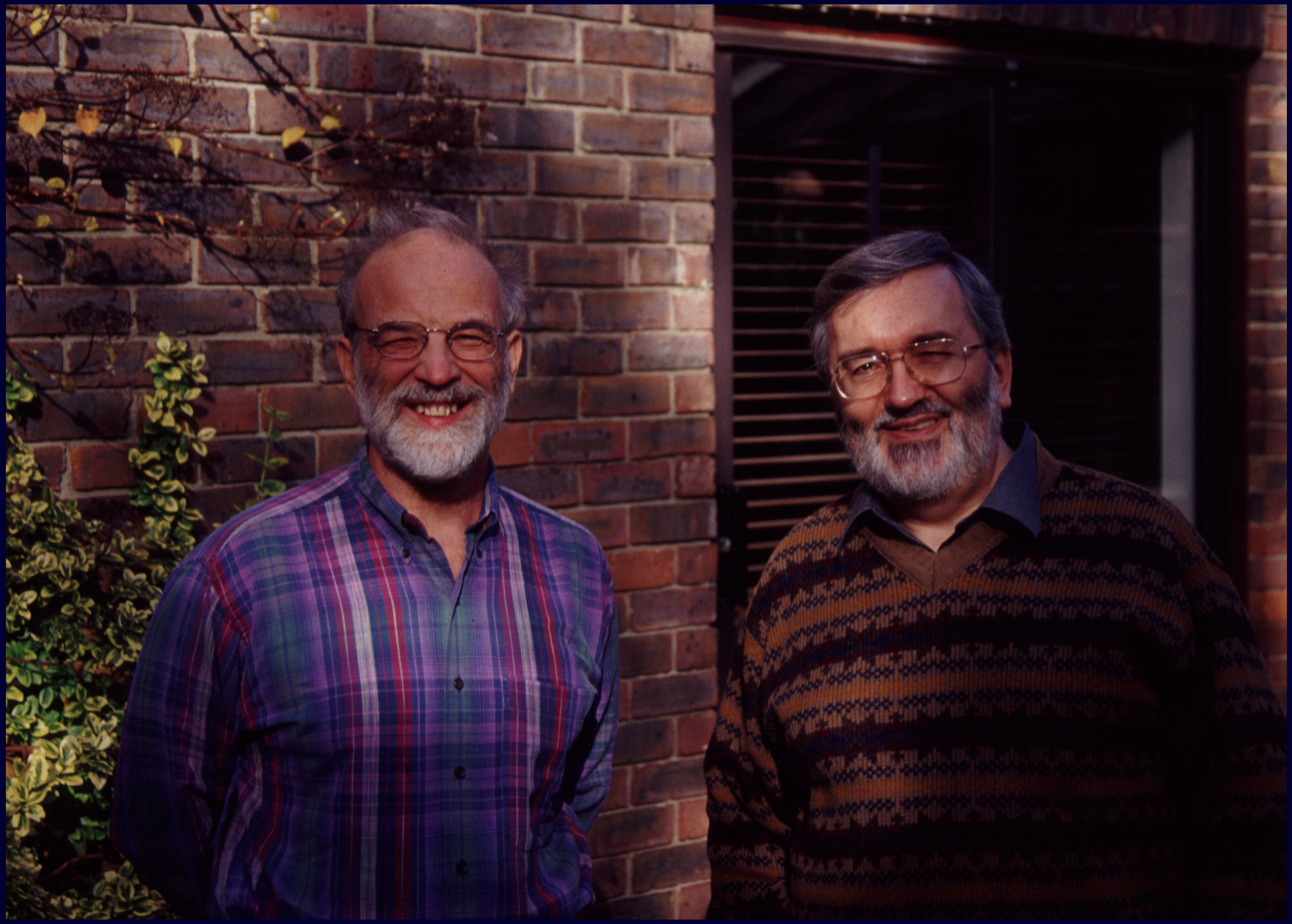
Modular Concurrency

Peter Grogono

Computer Science and Software Engineering

Concordia University



public scope

protected scope

private scope

package scope

instance methods

static methods

interfaces

local inner classes

anonymous inner classes

member inner classes

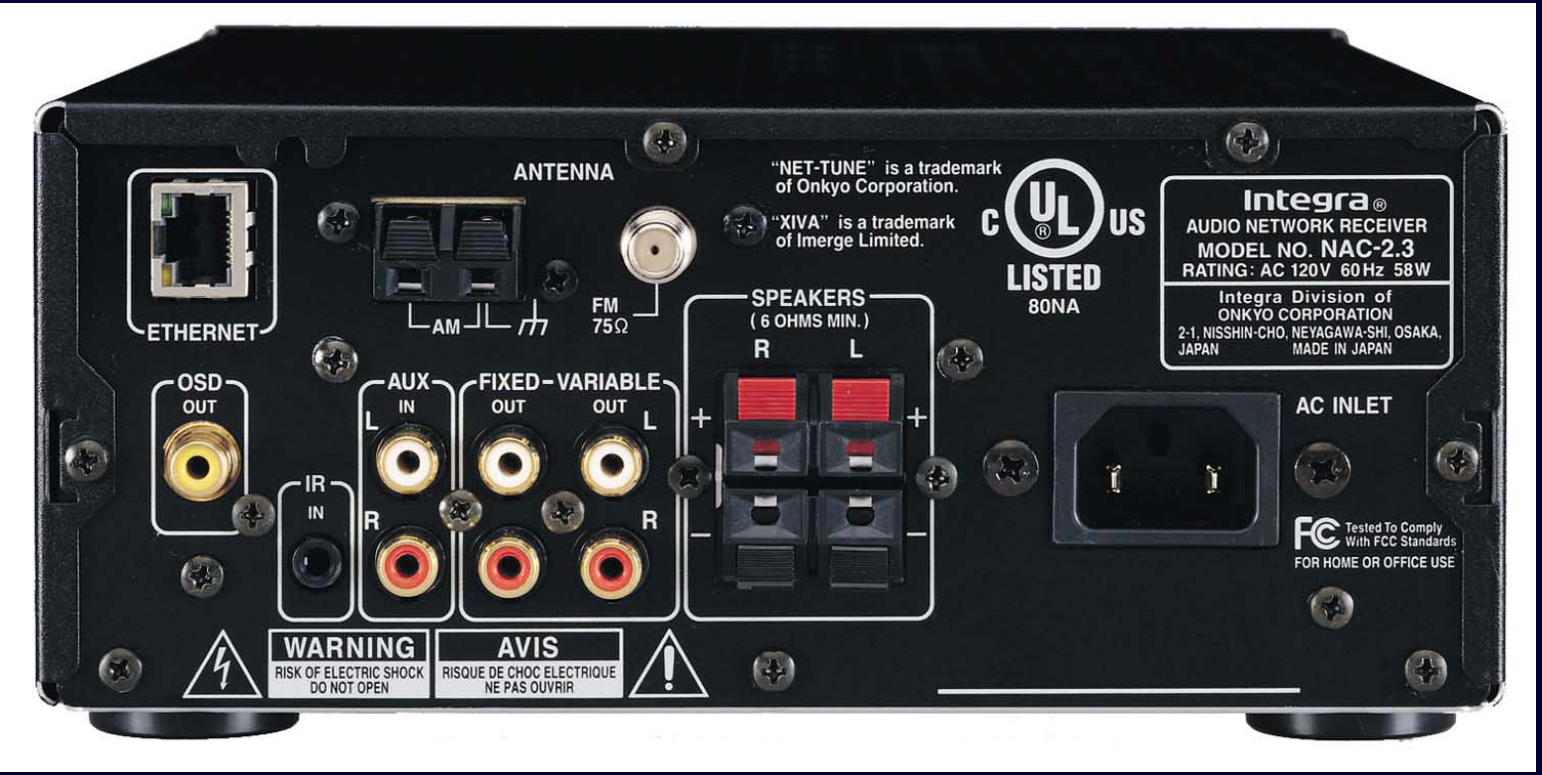
static member inner classes

delegates

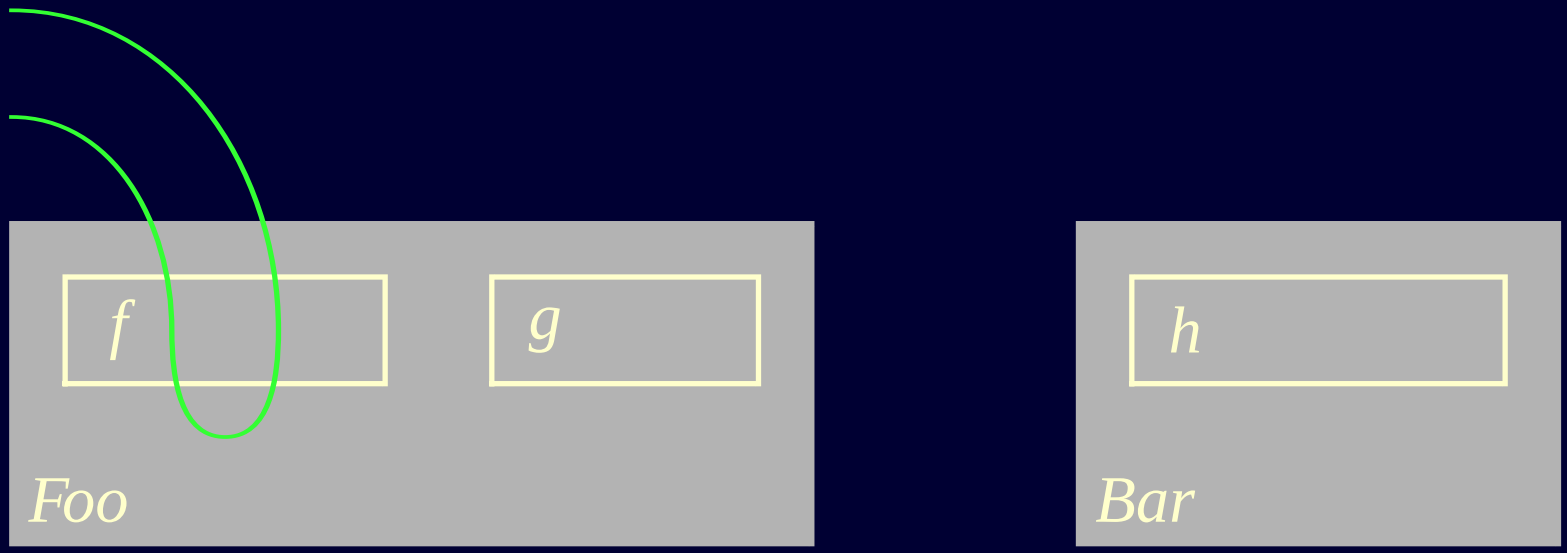
classes

packages


Integra
Standby/On
Standby
Setup
Phones
Server Playing
^Track 1: 1:37>
v Ups and Downs
Display:[4]8 0 ♪Trks
POWERED BY
XiVA
VECTOR LINEAR SHAPING CIRCUITRY
NETTUNE
Audio Input
Display
Select
Volume
Stop
Play/Pause
NAC-2.3

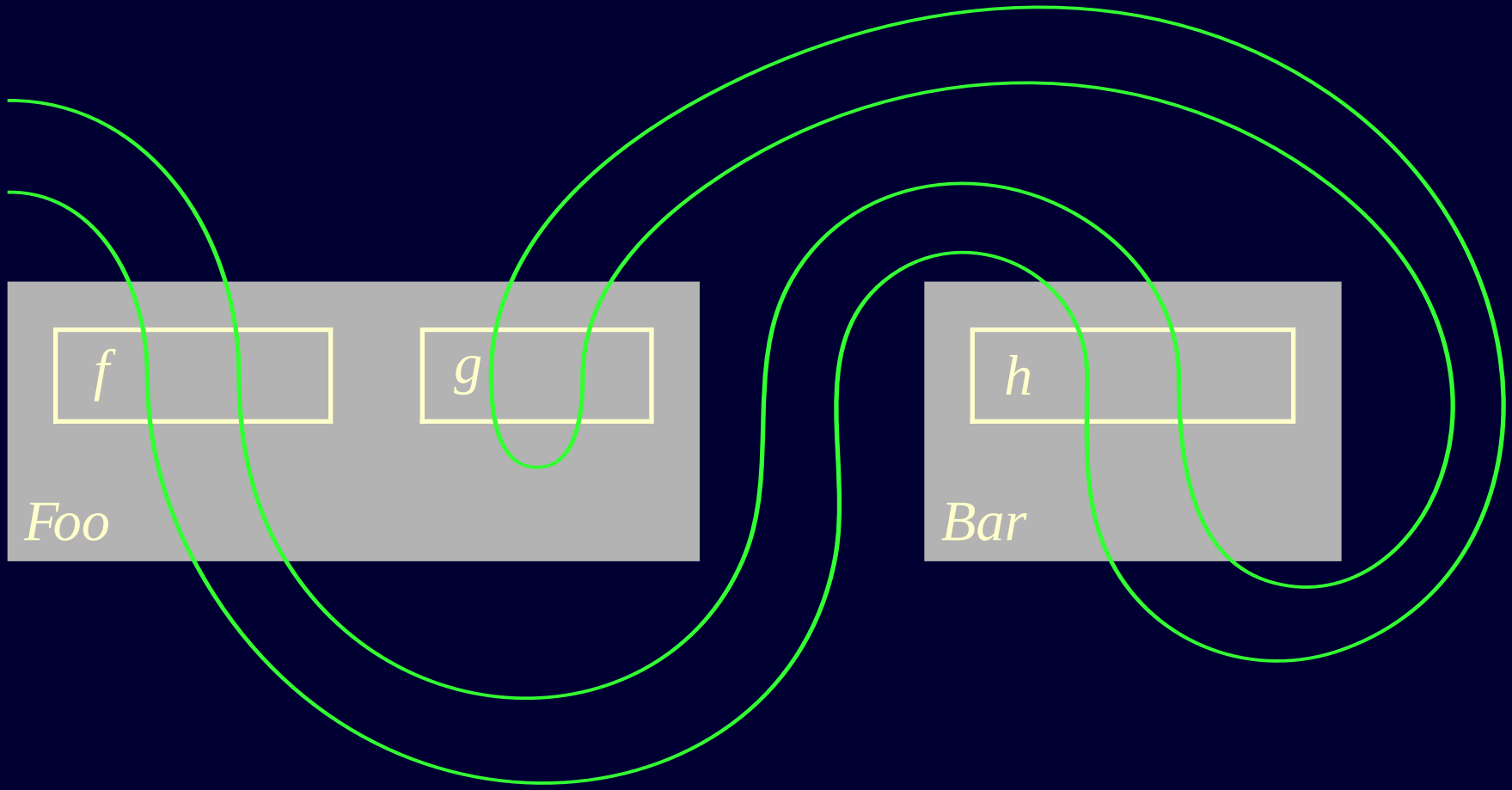

ETHERNET
ANTENNA
AM
FM
75Ω
"NET-TUNE" is a trademark of Onkyo Corporation.
"XIVA" is a trademark of Imerge Limited.
C UL US
LISTED
80NA
Integra®
AUDIO NETWORK RECEIVER
MODEL NO. NAC-2.3
RATING: AC 120V 60Hz 58W
Integra Division of
ONKYO CORPORATION
2-1, NISSHIN-CHO, NEYAGAWA-SHI, OSAKA, JAPAN
MADE IN JAPAN
SPEAKERS
(6 OHMS MIN.)
R L
OSD
OUT
IR
IN
AUX
IN
L
R
FIXED - VARIABLE
OUT OUT
L
R
AC INLET
Tested To Comply With FCC Standards
FOR HOME OR OFFICE USE
WARNING
RISK OF ELECTRIC SHOCK
DO NOT OPEN
AVIS
RISQUE DE CHOC ELECTRIQUE
NE PAS OUVRIR

```
class Foo
{
    private int x = 0
    private int y = 0
    invariant y = 2 × x
    public void f()
    {
        x += 1
        y += 2
    }
    public void g()
    {
        x *= 3
        y *= 3
    }
}
```


f
g
Foo
h
Bar

```
class Foo
{
    private int x = 0
    private int y = 0
    private Bar b
    invariant y = 2 × x
    public void f()
    {
        x += 1
        b.h()
        y += 2
    }
    public void g()
    {
        x *= 3
        y *= 3
    }
}
```

```
class Bar
{
    private Foo f
    public void h()
    {
        f.g()
    }
}
```


f
g
h
Foo
Bar

$x = 0$	0
$y = 0$	0
$x += 1$	1
$x *= 3$	3
$y *= 3$	0
$y += 2$	2

serves various **clients**

offers various **services**

logs activities

usage **statistics**

state **queries**

metastate **queries** (reflection)

state control: start/stop/suspend/resume/···

responds to **tests**

If I have seen farther than others, it is because I was standing on the shoulder of *giants*.

Isaac Newton

We should be standing on the shoulders, not the feet of those who worked on these problems.

Richard Fateman

James Gosling

Edsger Wybe Dijkstra

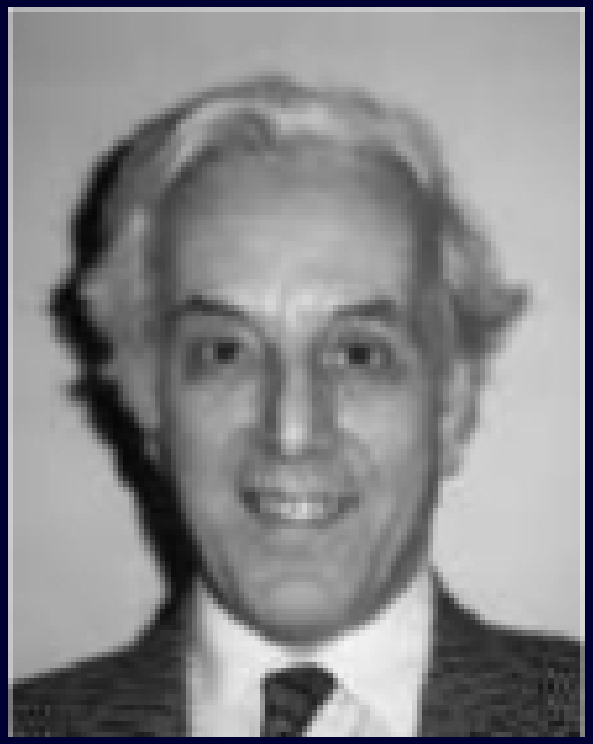
Charles Anthony Richard Hoare

Per Brinch Hansen

We have stipulated that processes should be connected loosely; by this we mean that apart from the (rare) moments of explicit intercommunication, the individual processes themselves are to be regarded as completely independent of each other.

Edsger Dijkstra

Michael Jackson #2



Michael Jackson #1

Born in the ice-blue waters of the festooned Norwegain coast; amplified (by an aberration of world currents, for which marine geographers have yet to find a suitable explanation) along the much grayer range of the Californian Pacific; viewed by some as a typhoon, by some as a tsunami, and by some as a storm in a teacup — a tidal wave is hitting the shores of the computing world.

Bertrand Meyer (1988)

Kristen Nygaard Ole-Johan Dahl

Alan Kay

Adele Goldberg

Barbara Liskov

The Best Part of Being an Engineer

I find a career in engineering to be very satisfying. I like making things work. I also like finding solutions to problems that are both practical and elegant. And, I like working with a team of people; engineering involves lots of team work.

Barbara Liskov

It is astounding to me that Java's insecure parallelism is taken seriously by the programming community, a quarter of a century after the invention of monitors and Concurrent Pascal. It has no merit.

Although the development of parallel languages began around 1972, it did not stop there. Today we have three major communication paradigms: monitors, remote procedures, and message passing. Any one of them would have been a vast improvement over Java's insecure variant of shared classes. As it is, Java ignores the last twenty-five years of research in parallel languages.

Per Brinch Hansen (1999)

You can blame some of this model inaccuracy on the extremely detailed and sensitive nature of current programming language technologies. Minor lapses and barely detectable coding errors, such as misaligned pointers or uninitialized variables, can have enormous consequences. . . . If such seemingly minute detail can have such dire consequences, how can we trust models to be accurate, since models, by definition, are supposed to hide or remove detail?

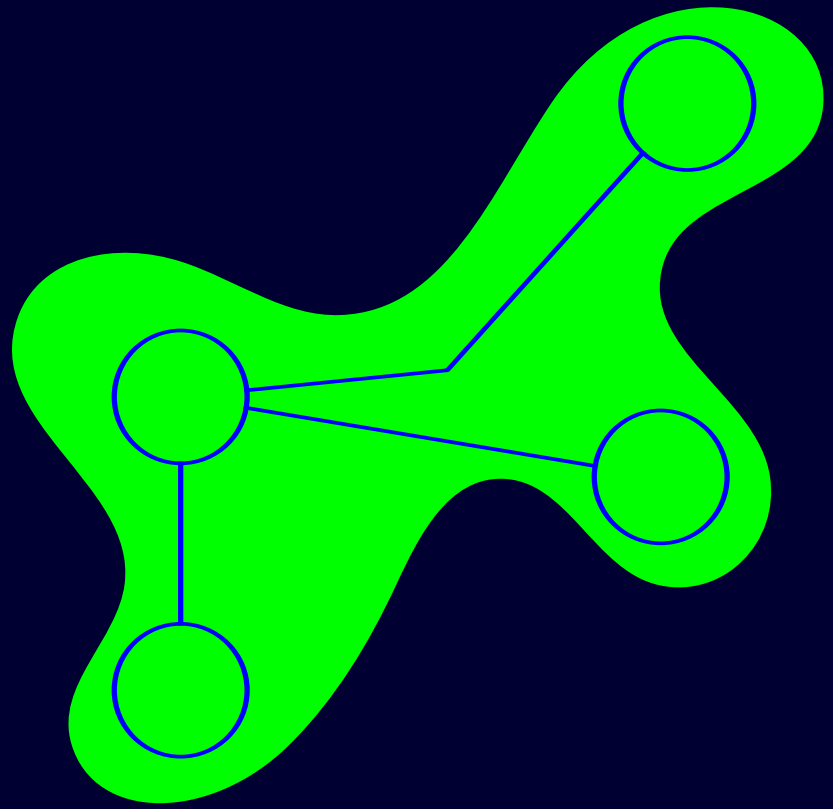
Bran Selic

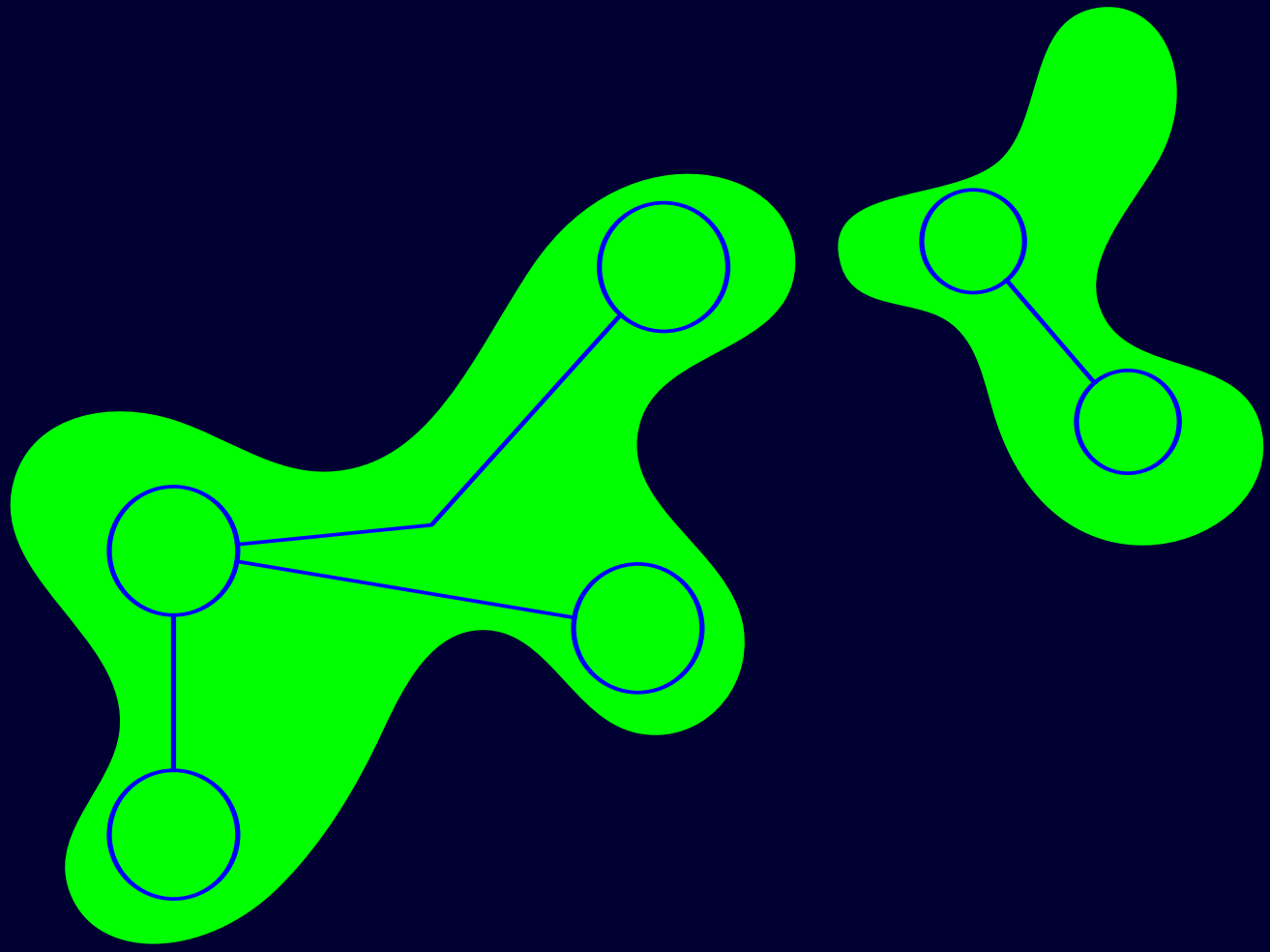
Recent work in concurrent programming

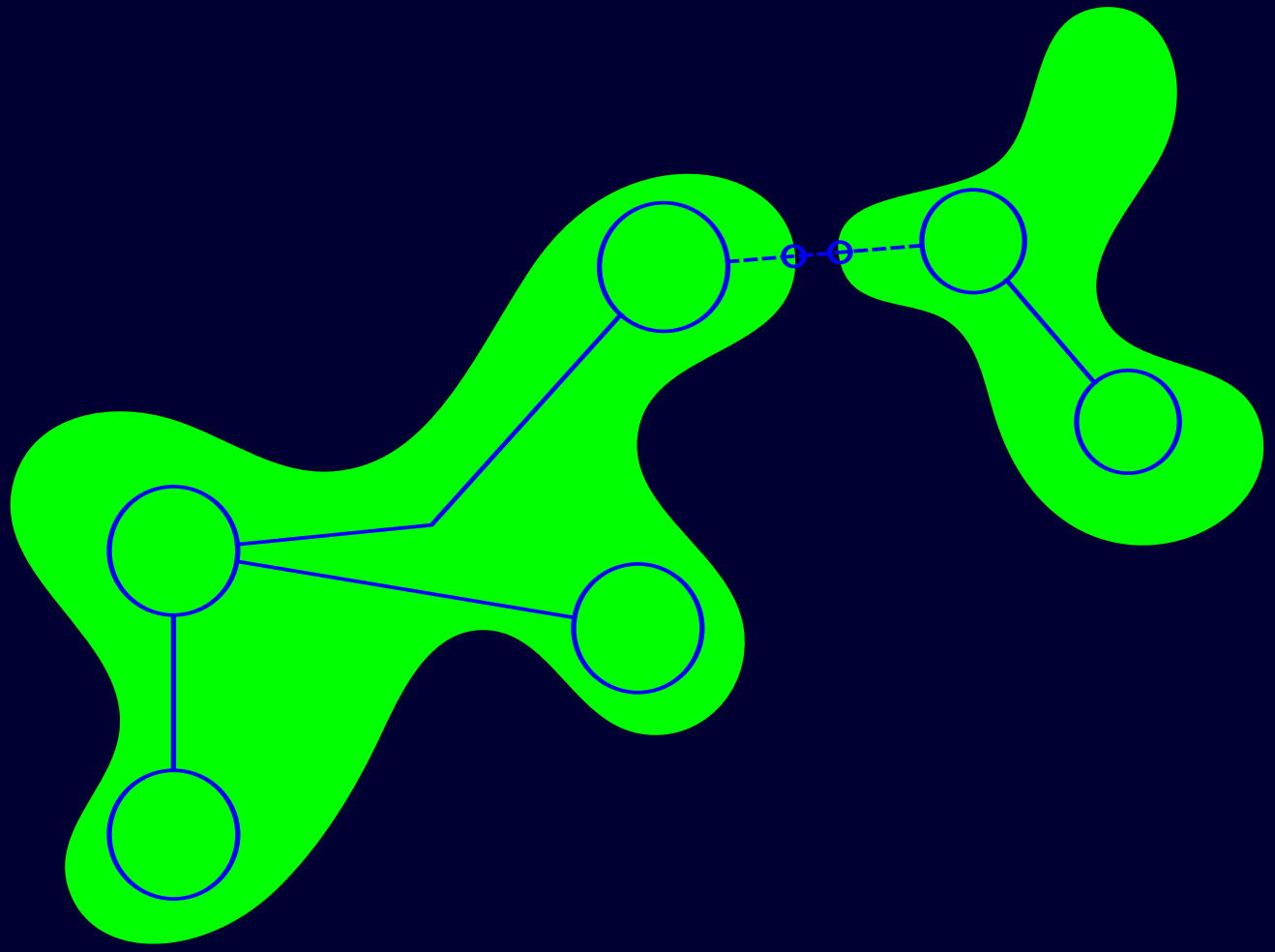
⋆ Joyce	Per Brinch Hansen
⋆ Hermes	IBM
⋆ Oz	Seif Haridi and Nils Franzén
⋆ occam-π	Fred Barnes *et al.*
⋆ Separation Logic	John C. Reynolds *et al.*

Assorted ideas:

1. A component is a *cell* running a multithreaded process
2. Cells exchange data
3. A cell gets exactly the capabilities that it needs
4. Semantics is decoupled from deployment
5. Programs are scale-free
6. Tests are part of the code



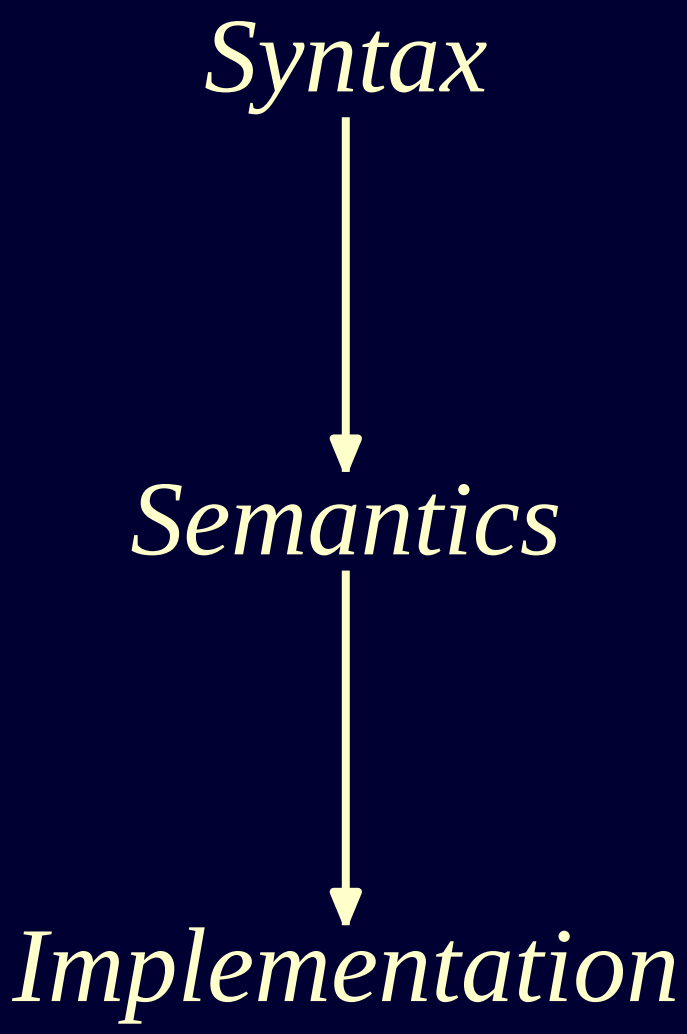


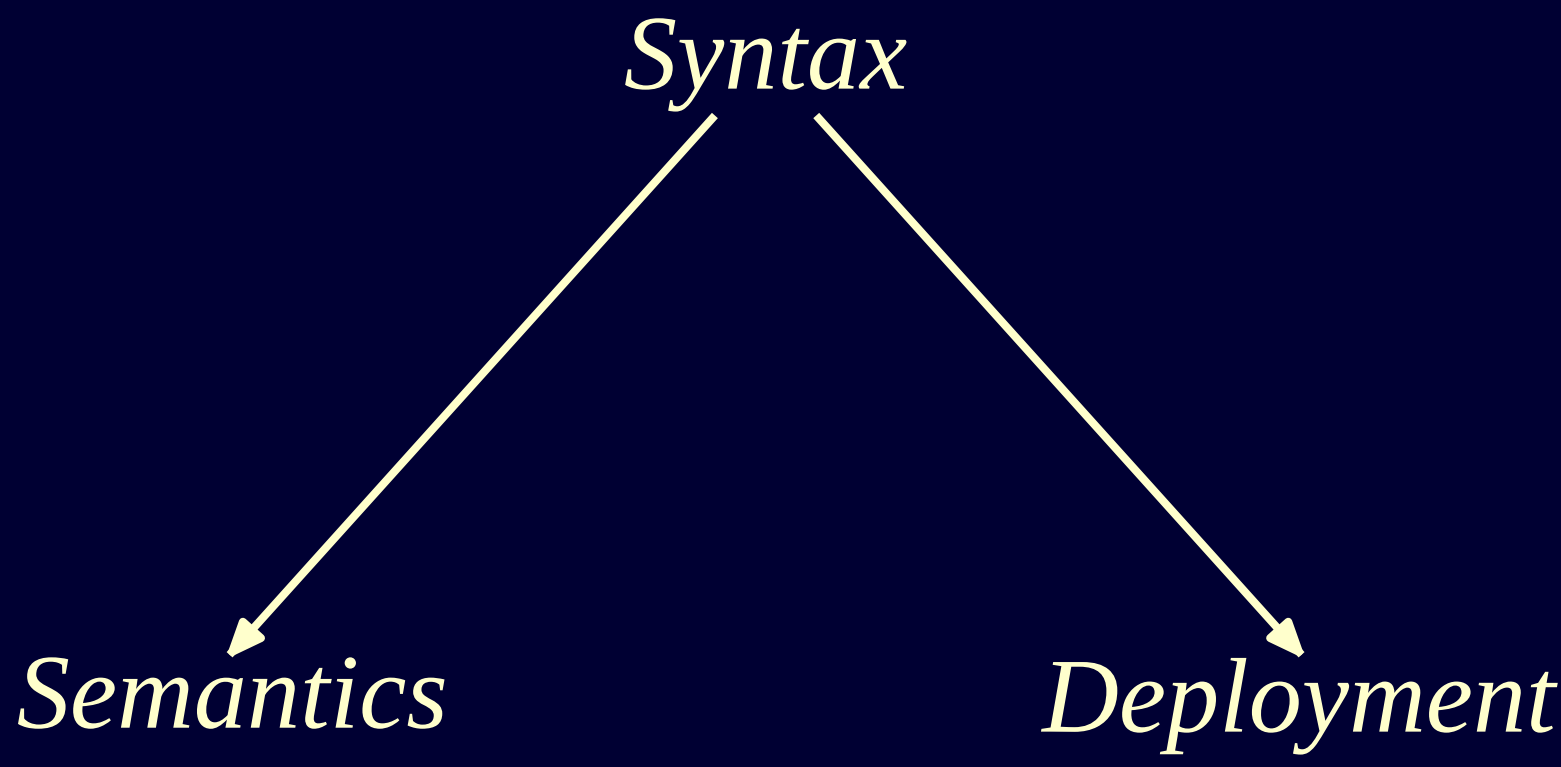


```
public static void main(String[] args)
{
    ....
    Random generator = new Random();
    ....
```

```
import java.util.Random;


public static void main(String[] args)
{
   ....
   Random generator = new Random();
   ....
```


Syntax
Semantics
Implementation


Syntax
Semantics
Deployment

```
income := salary() - fed.tax() - prov.tax()
```

```
abstract type MessageProtocol = [];
type SumProtocol = MessageProtocol[p, q: int;  result: *int];

type Sum =
[
  main: process(init: *SumProtocol) =
    var params: SumProtocol;
    init ? params;
    params.result ! move params.p + params.q
  end
]
```

```
var initProtocol: **SumProtocol;
new Sum({}) ! alias initProtocol;
var sumFunction: *SumProtocol;
initProtocol ? sumFunction;
var return: *int;
sumFunction ! copy (2, 3, alias return);
var x: int;
return ? x
```

```
type Sum =
[
  main: process(p, q: int;  result: *int) =
    result ! move p + q
  end
]
```

```
var return: *int;
Sum({}) ! copy (2, 3, alias return);
var x: int;
return ? x
```

Conclusions 1

1. Object Oriented programming has had a good run, but even the best shows don't last forever

2. Concurrent programming, after lurking in the wings for decades, is moving to centre stage

3. One play, many shows

Conclusions 2

1. Learn the tools of your trade
2. Trust your intuition
3. Question the foundations