

PCPP: PRACTICAL CONCURRENT & PARALLEL PROGRAMMING

MESSAGE PASSING CONCURRENCY I / II



Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

 IT University of Copenhagen

Introduction

Problem:

- **Sharing && Mutability!**



Solutions:

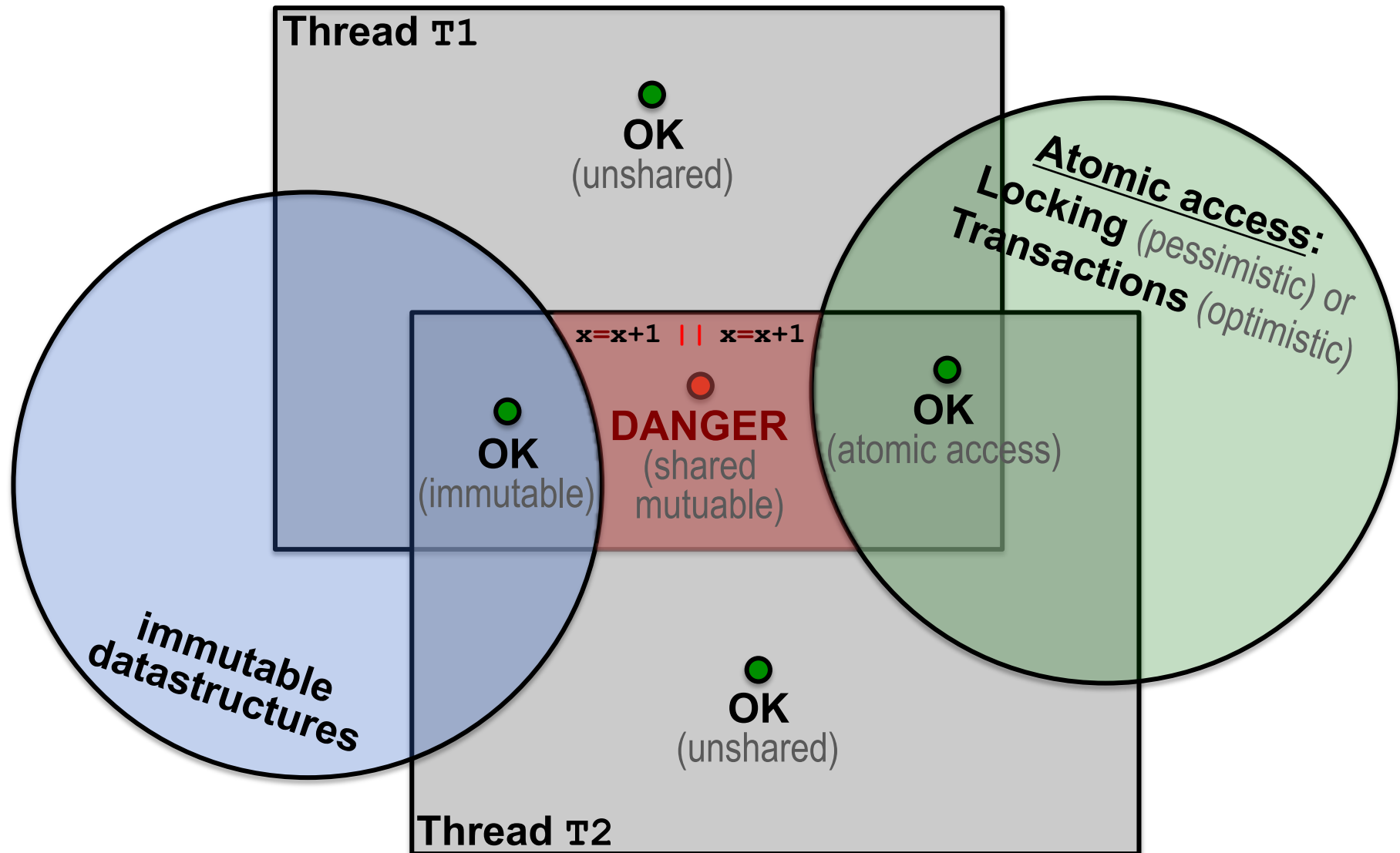
- **1) Atomic access (shared res.):** **"synchronized"**
 - **a) Locking** (pessimistic concurrency)
 - **b) Transactions** (optimistic concurrency)
- **2) Eliminate mutability:** **"final"**
 - E.g., functional programming
- **3) Eliminate sharing...:** **message passing concurrency**

PROBLEM:

Sharing & Mutability!

SOLUTIONS:

- 1) atomic access!
locking or transactions
NB: avoid deadlock!
- 2) avoid mutability!
- 3) avoid sharing...

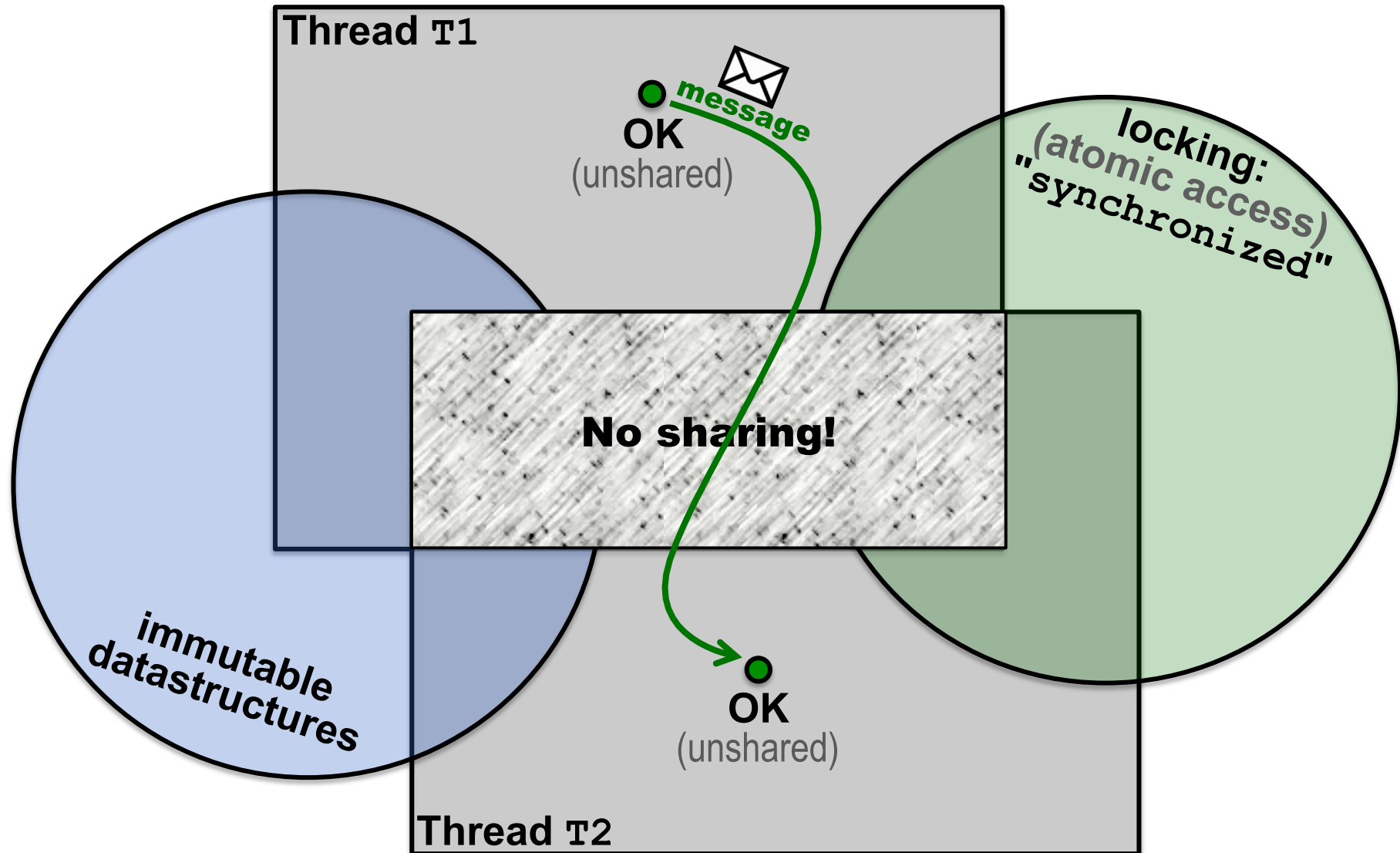


PROBLEM:

Sharing & Mutability!

SOLUTIONS:

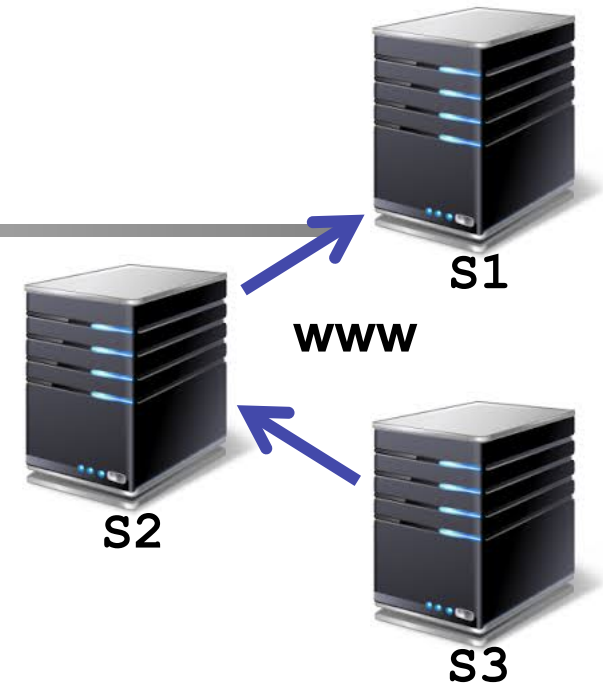
- 1) atomic access!
locking or transactions
NB: avoid deadlock!
- 2) avoid mutability!
- 3) avoid sharing...



World Wide Web...

In a distributed setting,
there's no shared memory:

- Communication is achieved
via "*message passing*"
 - (between concurrently executing servers)



Message Passing Concurrency:

- Same idea (*message passing*)
usable in non-distributed setting:
 - (between processes, inside a server)



Forms of Message Passing

■ Operations:

- send
- receive

■ Symmetry:

- symmetric (send and receive)
- asymmetric (send xor receive)

■ Synchronicity:

- synchronous (e.g., phone)
- asynchronous (e.g., email)
- rendez-vous (e.g., barrier)

■ Buffering:

- unbuffered (e.g., blocking)
- buffered (e.g., non-blocking)

■ Multiplicity:

- one-to-one
- one-to-many (or many-to-one)
- many-to-many

■ Addressing:

- direct (naming processes)
- indirect (naming addresses)

■ Reception:

- unconditional (all messages)
- selective (only certain msgs)

■ Anonymity:

- anonymous
- non-anonymous

Synchronous Message Passing !



Send:

```
send v to q ;
```

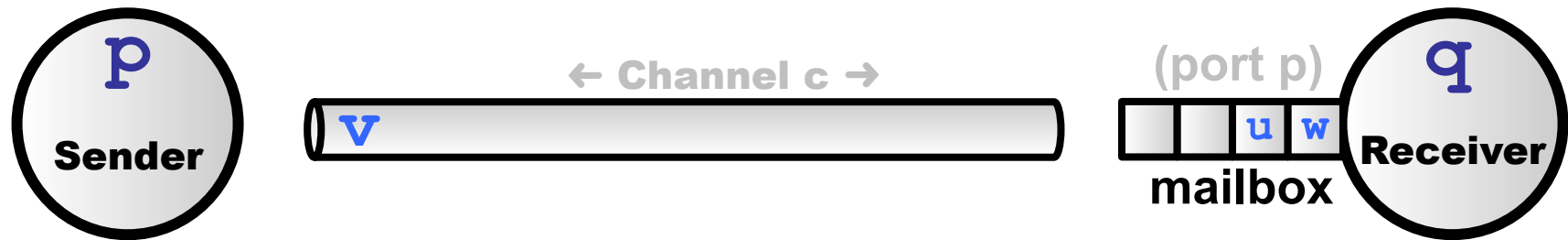
- Sender process **p** *sends* value **v** to receiver process **q**
- Sending process **p** *blocked* until process **q** receives **v**

Receive:

```
v = receive() ;
```

- Receiver process **q** attempts to *receive* a value **v**
 - Receiver process **q** is *blocked* until value received
- **Synchronous** (i.e., no message buffering)!

Asynchronous Message Passing !



Send:

```
send v to q;
```

- Sender process **p** **sends** value **v** to process **q**'s mailbox
- Sending process **p** **continues after sending**

Receive:

```
v = receive();
```

- Receiver process **q** attempts to **receive** **v** from its inbox
- Receiver process **q** is **blocked** until inbox is non-empty
- **Asynchronous** (i.e., messages are buffered)!

Philosophy & Expectations !

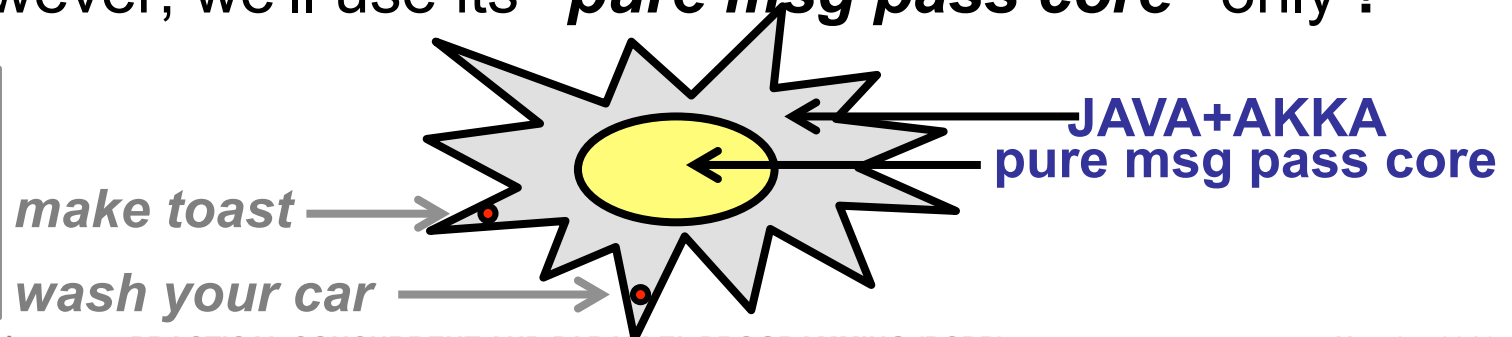
■ ERLANG:

- We'll use as message passing *specification language*
- You have to-be-able-to **read** simple **ERLANG** programs
 - (i.e., not *write*, nor *modify*)

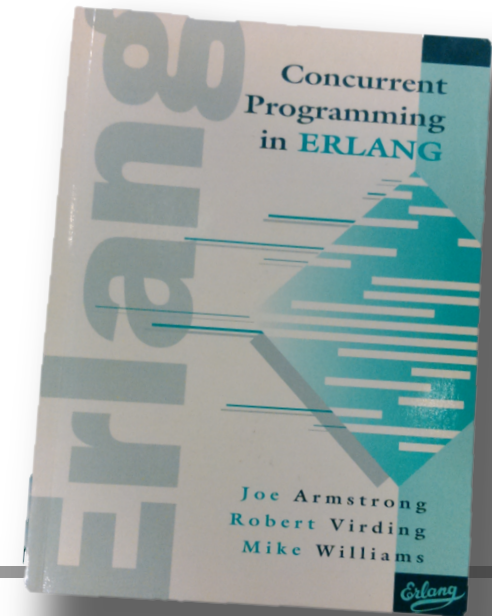
■ JAVA+AKKA:

- We'll use as msg passing *implementation language*
- You have 2-b-a-2 *read/write/modify* **JAVA+AKKA** p's
- However, we'll use its "*pure msg pass core*" only !

NB: we're not going to use all of its fantazillions of functions!



An ERLANG Tutorial



"Concurrent Programming in ERLANG"

(Joe Armstrong, Robert Virding, Claes Wikström, Mike Williams)

[Ericsson 1994]

ERLANG

- Named after Danish mathematician **Agner Krarup ERLANG:**

...credited for inventing:

- *traffic engineering*
- *queueing theory*
- *telephone network analysis*



[1878–1929]

[http://en.wikipedia.org/wiki/Agner_Krarup_Erlang]

- **The ERLANG language:**

[http://en.wikipedia.org/wiki/Erlang_%28programming_language%29]

- by Ericsson in 1986 (Ericsson Language? :-)

The ERLANG Language (1986)

- Functional language with....:

- **message passing concurrency !!!**

- **garbage collection**

- **eager evaluation**

- **single assignment**

- **dynamic typing**

"Though all concurrency is explicit in ERLANG, processes communicate using **message passing** instead of shared variables, **which removes the need for explicit locks.**"

-- Wikipedia

- Designed by **Ericsson** to support....:

distributed, fault-tolerant, soft-real-time, non-stop applications

- It supports **"hot swapping"**:

- i.e., code can be changed without stopping a system!

Hello World

■ Hello World (in ERLANG)

```
% hello world program:  
-module(helloworld)  
-export([start/0]).
```

An **actor** named "**start**" taking zero arguments (i.e., arity zero)

```
start() ->  
    io:fwrite("Hello world!\n").
```

■ Output:

```
Hello world!
```

■ Try it out:

[www.tutorialspoint.com/compile_erlang_online.php]

Online ERLANG Compiler

■ Online ERLANG Compiler:

[www.tutorialspoint.com/compile_erlang_online.php]

■ Documentation:

[<http://www.erlang.org/doc/man/io.html>]

■ Simple usage:

- One module called: `helloworld`
- Export one function called: `start/0`
- Call *your code* from `start()` and `io:write` output

```
-module(helloworld) .  
-export([start/0]) .  
  
yourcode(...) -> ...  
  
start() -> Val = yourcode(...), % single assign: unchangable!  
           io:write(Val).       % NB: use fwrite for strings!
```

Factorial

■ Factorial (in ERLANG)

```
% factorial program:
-module(mymath) .
-export([start/0, factorial/1]) .

factorial(0) -> 1;
factorial(N) -> N * factorial(N-1) .

start() -> io:write(factorial(6)) .
```

■ Usage:

```
> mymath:factorial(6) .
720
```

■ Try it out:

```
> mymath:factorial(25) .
15511210043330985984000000
```

[www.tutorialspoint.com/compile_erlang_online.php]

Modularization: Import / Export

■ Factorial (in ERLANG)

```
-module(mymath) .  
-export([double/1]) .  
  
double(X) -> times(X, 2). % public  
  
times(X, N) -> X * N.      % private
```

■ Usage:

```
> mymath:double(10) .  
20
```

```
> mymath:times(5,2) .  
** undef'd fun': mymath:double/2 **
```

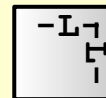
■ Try it out:

[www.tutorialspoint.com/compile_erlang_online.php]

Pattern Matching

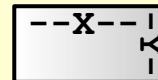
```
-module(mymath) .  
-export([area/1]).
```

```
area( {square, L} ) ->  
    L * L;
```

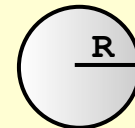


%% patterns in purple!

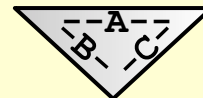
```
area( {rectangle, X, Y} ) ->  
    X * Y;
```



```
area( {circle, R} ) ->  
    3.14159 * R * R;
```



```
area( {triangle, A, B, C} ) ->  
    S = (A + B + C)/2,  
    math:sqrt(S*(S-A)*(S-B)*(S-C)).
```



%% immutable assignment

```
> Thing = {triangle, 6, 7, 8}.  
{triangle,6,7,8}  
> mymath:area(Thing).  
20.3332
```

Values (with lists and tuples)

- Numbers: 42, -99, 3.1415, 6.626e-34, ...
- Atoms: abc, 'with space', hello_world, ...
- Tuples: {}, { 1, 2, 3 }, { { x, 1 }, { 2, y, 3 } }
- Lists: [], [1, 2, 3], [[x, 1], [2, y, 3]]

```
PCPP =  
  {course, "Practical Concurrent and Parallel Programming",  
   {master, 7.5, { fall, 2016 } }  
   { teachers, [ 'Peter Sestoft', 'Claus Brabrand' ] },  
   { students, [ aaa, bbb, ccc, ... ] }  
  }
```

String (really just
list of characters)

- Recall: *dynamically typed*

Lists: `member/2`

- `[H|T]` is (standard) **"head-tail constructor"**:
 - `H` is the **head**; i.e., *the first element* (one element)
 - `T` is the **tail**; i.e., *the rest of the list* (zero-or-more)

```
-module(mylists) .  
-export([member/2]) .
```

*...for list
construction
de-construction*

```
member( X, [] ) -> false;  
member( X, [X|_] ) -> true;  
member( X, [_|T] ) -> member(X, T) .
```

```
> mylists:member(3, [1,3,2]) .  
true
```

```
> mylists:member(4, [1,3,2]) .  
false
```

Lists: `append/2`

- `[H|T]` is (standard) **"head-tail constructor"**:
 - `H` is the **head**; i.e., *the first element* (one element)
 - `T` is the **tail**; i.e., *the rest of the list* (zero-or-more)

```
-module(mylists) .  
-export([append/2]) .
```

```
append( [], L ) -> L;
```

```
append( [H|L1], L2 ) -> [H|append(L1, L2)] .
```

*...for list
construction
de-construction*

and re-construction

```
> mylists:append([], [a,b])  
[a,b]
```

```
> mylists:append([1,2], [3,4])  
[1,2,3,4]
```

Message Passing in ERLANG!

Actor: Send / Receive / Spawn

■ Send:

- `Pid ! M` // Message M is sent to process Pid
- `Pid ! {some, {complex, structured, [m,s,g]}, 42}`

■ Receive:

- ```
receive
 pattern1 -> ...
;
 pattern2 -> ...
end
```

- ```
receive
  {init,N} when N>0 -> ...
;
  {init,N} -> ...
end
```

■ Spawn:

- `MyActorId = spawn(mymodule,myactor,[a,r,g,s])`

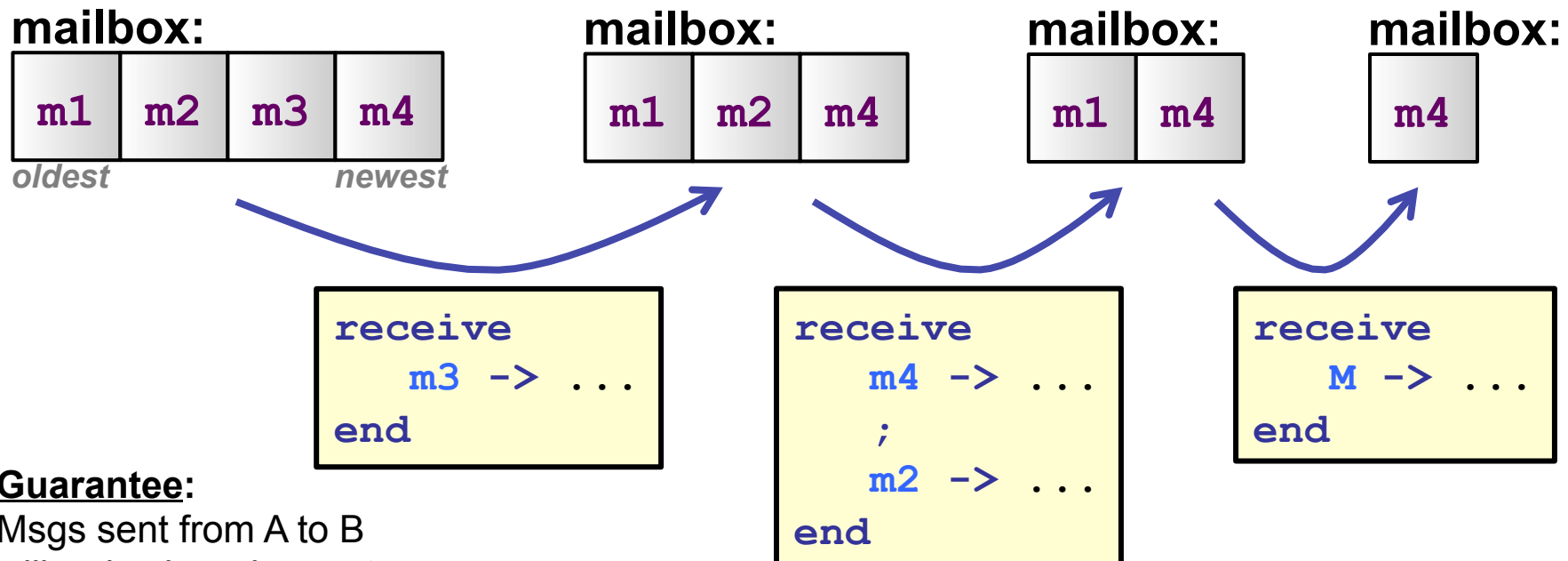
Order of Receiving Messages

■ Semantics:

```
for (M: message) {  
  for (P: pattern) {  
    M~P (i.e., M matches P)?  
  }  
}
```

This is what happens inside each actor.

■ Example:



Guarantee:

Msgs sent from A to B
will arrive in order sent

5 Examples (ERLANG & JAVA+AKKA)

1) HelloWorld:

The "Hello World" of message passing; one message is sent to **one actor**.

2) Ecco:

A **person actor** sends a msg to an **ecco actor** that responds with three suffix messages (used for ye olde "hvad drikker møller" kids joke).

3) Broadcast:

Three **person actors** unsubscribe/subscribe to a **broadcast actor** that forwards subsequent incoming msgs to subscribed persons.

4) Primer:

An **actor primer** is created that when initialized with **N=7** creates a **list[]** of that many **slave actors** to factor primes for it. Main bombards the prime actor with msgs ($p \in [2..100]$) that are evenly distributed among the slaves according to **list[p%n]**.

5) ABC:

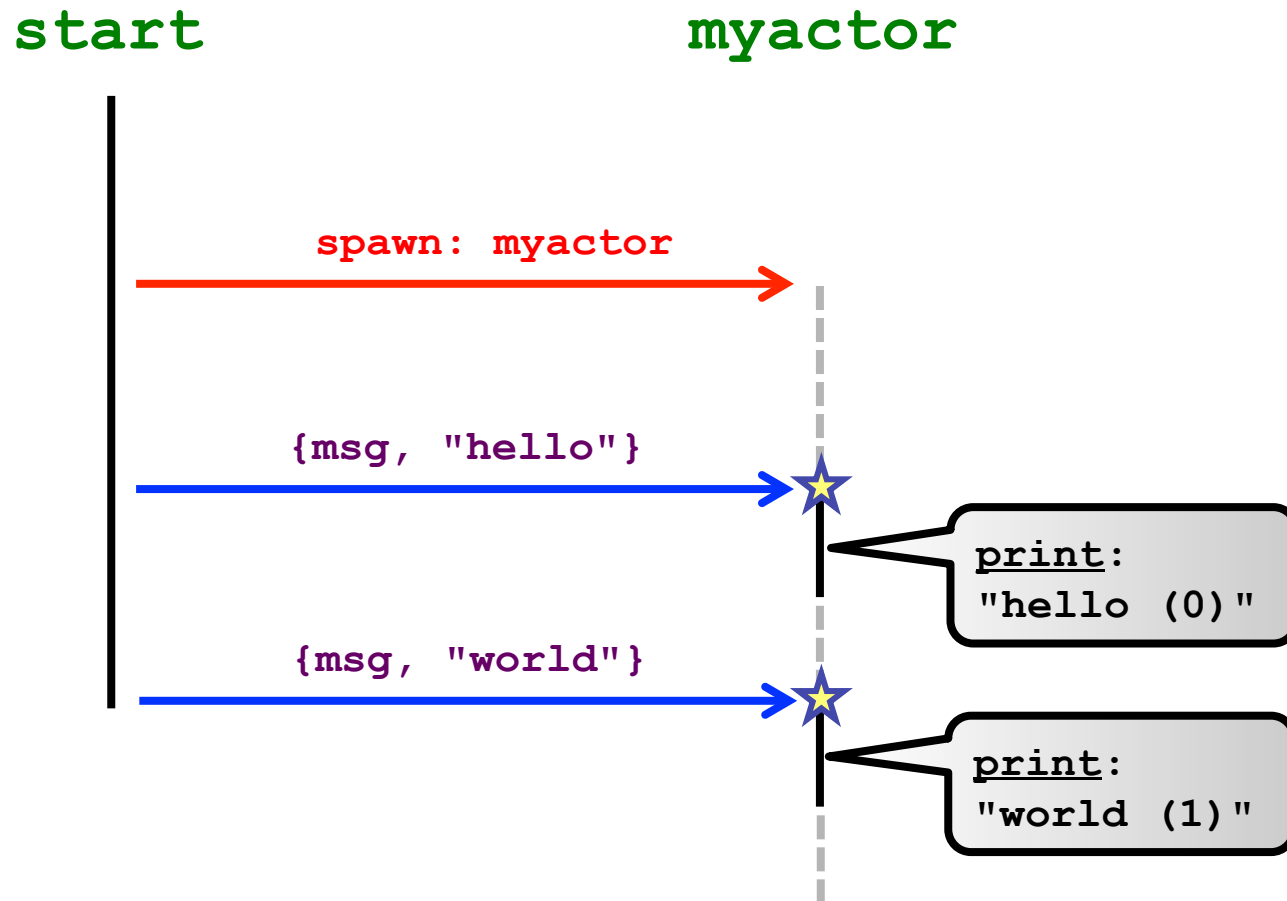
[lecture-#06]

Two **clerk actors** each bombard a **bank actor** with 100 transfer-random-amount-x-from-an-account-to-other-account msgs. The banks transfer the money by sending deposit(+x) to one **account actor** and deposit(-x) to the other **account actor**. (The system is called ABC as in Account/Bank/Clerk.)

1) HelloWorld

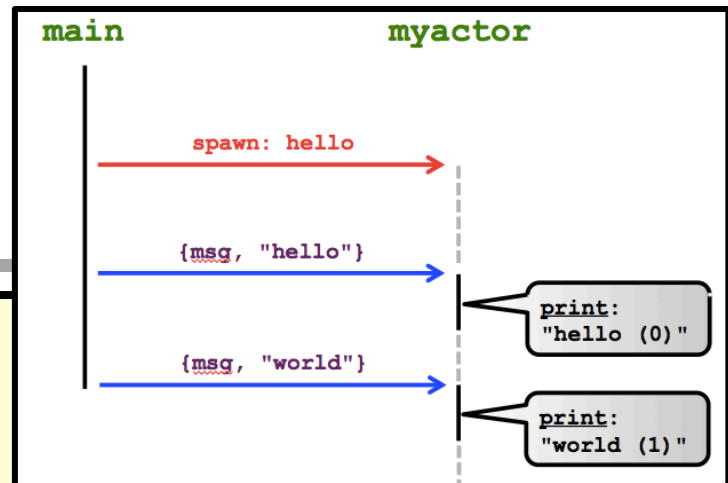
LEGEND:

send, receive, msgs
actors, spawn, rest.



1) HelloWorld.erl

```
-module(helloworld) .  
-export([start/0,myactor/1]) .  
  
myactor(Count) -> %% can have (local) state  
    receive  
        {msg, Msg} ->  
            io:fwrite(Msg ++ " ("),  
            io:write(Count),  
            io:fwrite(")\n"),  
            myactor(Count + 1)  
    end.  
  
start() ->  
    MyActor = spawn(helloworld, myactor, [0]),  
    MyActor ! {msg, "hello"},  
    MyActor ! {msg, "world"}.
```



```
hello (0)  
world (1)
```

1) HelloWorld.java

```
import java.io.*;
import akka.actor.*;
```

```
// -- MESSAGE
```

In JAVA+AKKA,
we want to pass
immutable msgs

Otherwise,
we're back to
shared mutable!

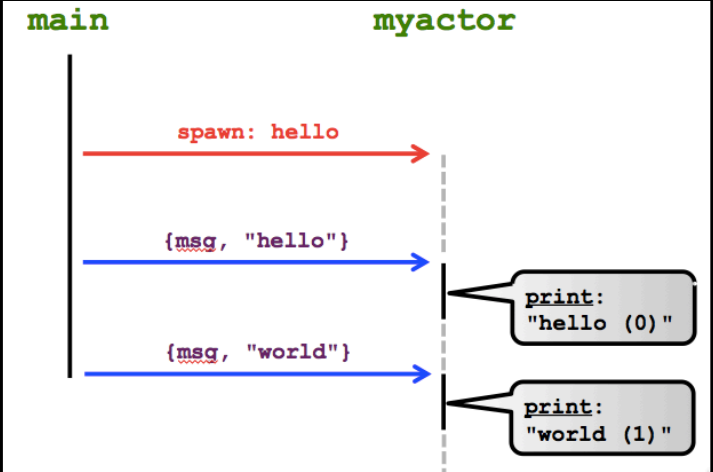
```
class MyMessage implements Serializable { // must be Serializable:
    public final String s;
    public MyMessage(String s) { this.s = s; }
}
```

```
// -- ACTOR -----
```

```
class MyActor extends UntypedActor {
    private int count = 0; // can have (local) state

    public void onReceive(Object o) throws Exception { // reacting to message:
        if (o instanceof MyMessage) {
            MyMessage message = (MyMessage) o;
            System.out.println(message.s + " (" + count + ")");
            count++;
        }
    }
}
```

```
hello (0)
world (1)
```



1) HelloWorld.java

In JAVA+AKKA,
the `main()` thread
is NOT an actor !

In JAVA+AKKA,
the `main()` thread
is NOT an actor !

// -- MAIN

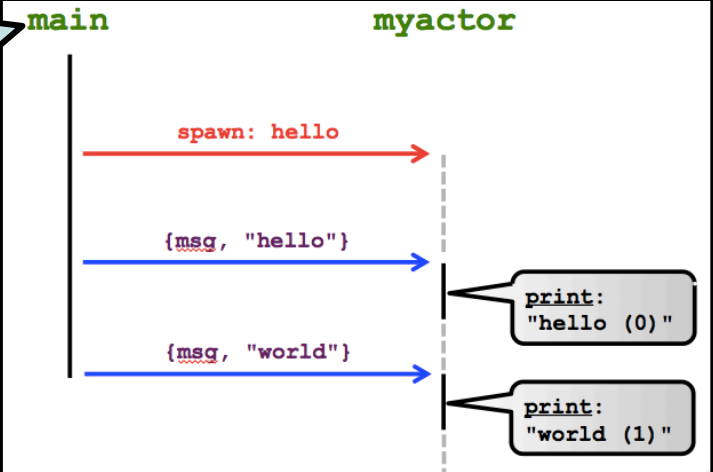
```
public class HelloWorld {
    public static void main(String[] args) {
        final ActorSystem system = ActorSystem.create("HelloWorldSystem");

        final ActorRef myactor =
            system.actorOf(Props.create(MyActor.class), "myactor");

        myactor.tell(new MyMessage("hello"), ActorRef.noSender());

        myactor.tell(new MyMessage("world"), ActorRef.noSender());

        try {
            System.out.println("Press return to terminate...");
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            system.shutdown();
        }
    }
}
```



```
hello (0)
world (1)
```

1) HelloWorld.java

■ Compile:

```
javac -cp scala.jar:akka-actor.jar HelloWorld.java
```

■ Run:

```
java -cp scala.jar:akka-actor.jar:akka-config.jar:. HelloWorld
```

■ Output:

```
hello (0)  
world (1)
```

2) Ecco

- From Old Danish Kids Joke:

- [<http://www.tordenskjoldssoldater.dk/ekko.html>]

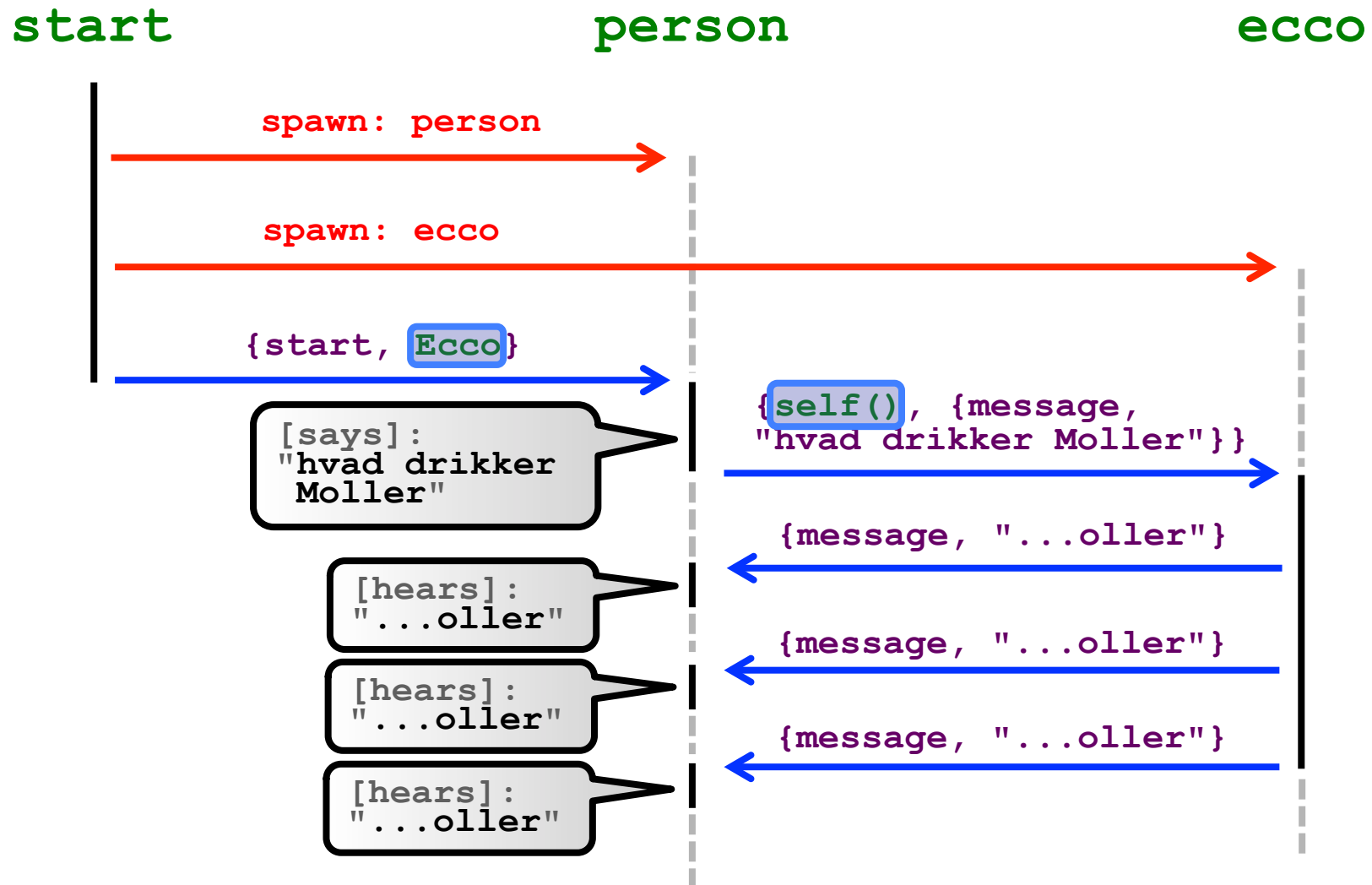


- Huge graffiti in Nordhavnen, Copenhagen:



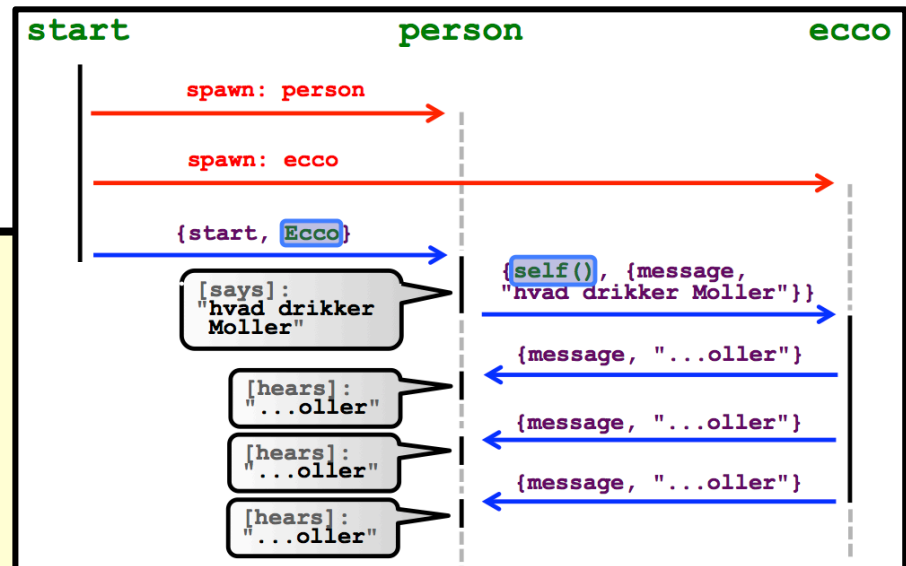
[<https://www.flickr.com/photos/unacivetta/5745925102/>]

2) Ecco



2) Ecco.erl

```
-module(helloworld).  
-export([start/0, person/0, ecco/0]).  
  
person() ->  
    receive  
        {start, Pid} ->  
            S = "hvad drikker Moller",  
            io:fwrite("[says]:  " ++ S ++ "\n"),  
            Pid ! {self(), {message, S}} ;  
        {message, S} ->  
            io:fwrite("[hears]: " ++ S ++ "\n")  
    end,  
    person().  
  
ecco() ->  
    receive  
        {Sender, {message, S}} ->  
            Sub = substr(S),  
            Sender ! {message, Sub},  
            Sender ! {message, Sub},  
            Sender ! {message, Sub},  
            ecco()  
    end.  
  
start() ->  
    Person = spawn(helloworld, person, []),  
    Ecco = spawn(helloworld, ecco, []),  
    Person ! {start, Ecco}.
```



```
substr(S) when length(S) < 6 -> "..." ++ S;  
substr([_|T]) -> substr(T).
```

```
[says]:  hvad drikker Moller  
[hears]: ...oller  
[hears]: ...oller  
[hears]: ...oller
```


2) Ecco.java

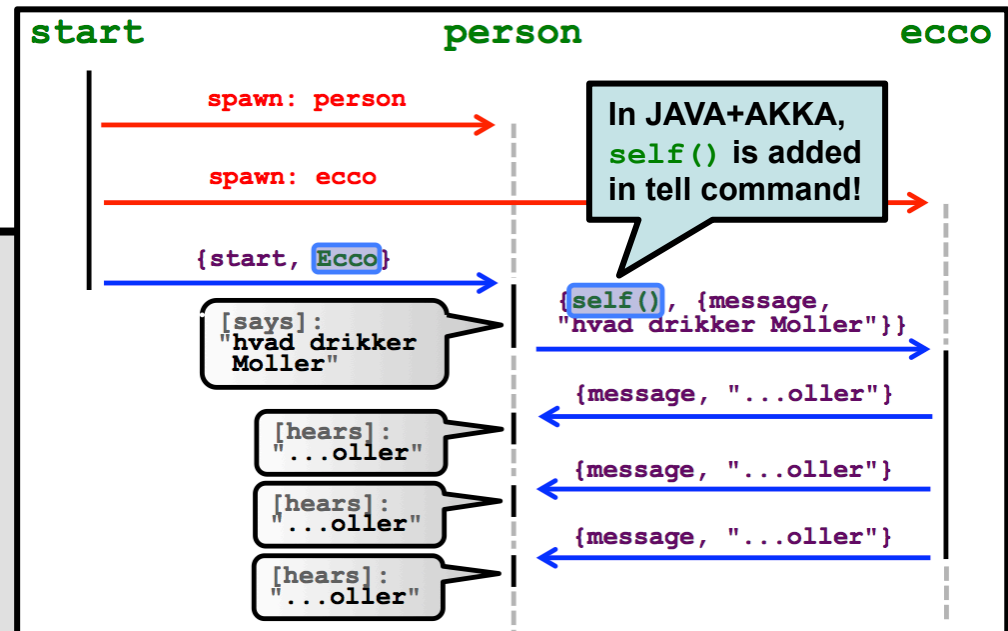
```
import java.io.*;
import akka.actor.*;
```

```
// -- MESSAGES -----

class StartMessage implements Serializable {
    public final ActorRef ecco;
    public StartMessage(ActorRef ecco) {
        this.ecco = ecco;
    }
}

class Message implements Serializable {
    public final String s;
    public Message(String s) {
        this.s = s;
    }
}
```

Used for...:
 person ← ecco
 ...and also for:
 person → ecco



```
[says]:  hvad drikker Moller
[hears]: ...oller
[hears]: ...oller
[hears]: ...oller
```

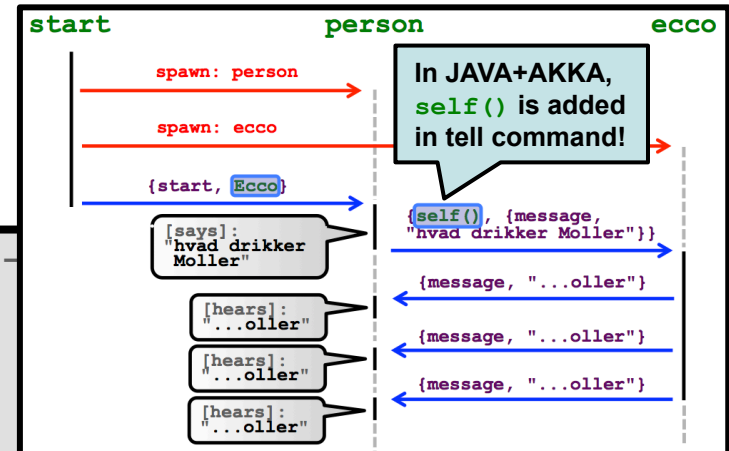
2) Ecco.java

```
// -- ACTORS -----
```

```
class PersonActor extends UntypedActor {
    public void onReceive(Object o) throws Exception {
        if (o instanceof StartMessage) {
            StartMessage start = (StartMessage) o;
            ActorRef ecco = start.ecco;
            String s = "hvad drikker moller";
            System.out.println("[says]: " + s);
            ecco.tell(new Message(s), getSelf());
        } else if (o instanceof Message) {
            Message m = (Message) o;
            System.out.println("[hears]: " + m.s);
        }
    }
}

class EccoActor extends UntypedActor {
    public void onReceive(Object o) throws Exception {
        if (o instanceof Message) {
            Message m = (Message) o;
            String s = m.s;
            Message reply;
            if (s.length() > 5) reply = new Message("..." + s.substring(s.length() - 5));
            else reply = new Message("...");
            getSender().tell(reply, getSelf());
            getSender().tell(reply, getSelf());
            getSender().tell(reply, getSelf());
        }
    }
}
```

Here, could also have been:
`ActorRef.noSender()`



```
[says]:  hvad drikker Moller
[hears]: ...oller
[hears]: ...oller
[hears]: ...oller
```

2) Ecco.java

```
// -- MAIN -----
```

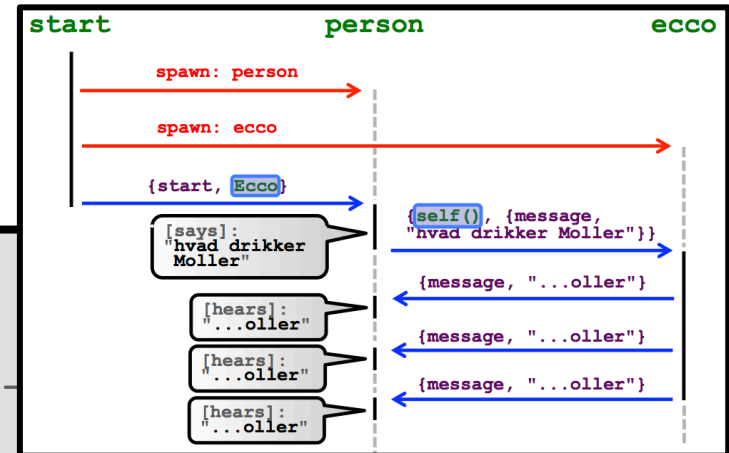
```
public class Ecco {
    public static void main(String[] args) {
        final ActorSystem system = ActorSystem.create("EccoSystem");

        final ActorRef person =
            system.actorOf(Props.create(PersonActor.class), "person");

        final ActorRef ecco =
            system.actorOf(Props.create(EccoActor.class), "ecco");

        person.tell(new StartMessage(ecco), ActorRef.noSender());

        try {
            System.out.println("Press return to terminate...");
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            system.shutdown();
        }
    }
}
```



```
[says]:  hvad drikker Moller
[hears]: ...oller
[hears]: ...oller
[hears]: ...oller
```

2) Ecco.java

■ Compile:

```
javac -cp scala.jar:akka-actor.jar Ecco.java
```

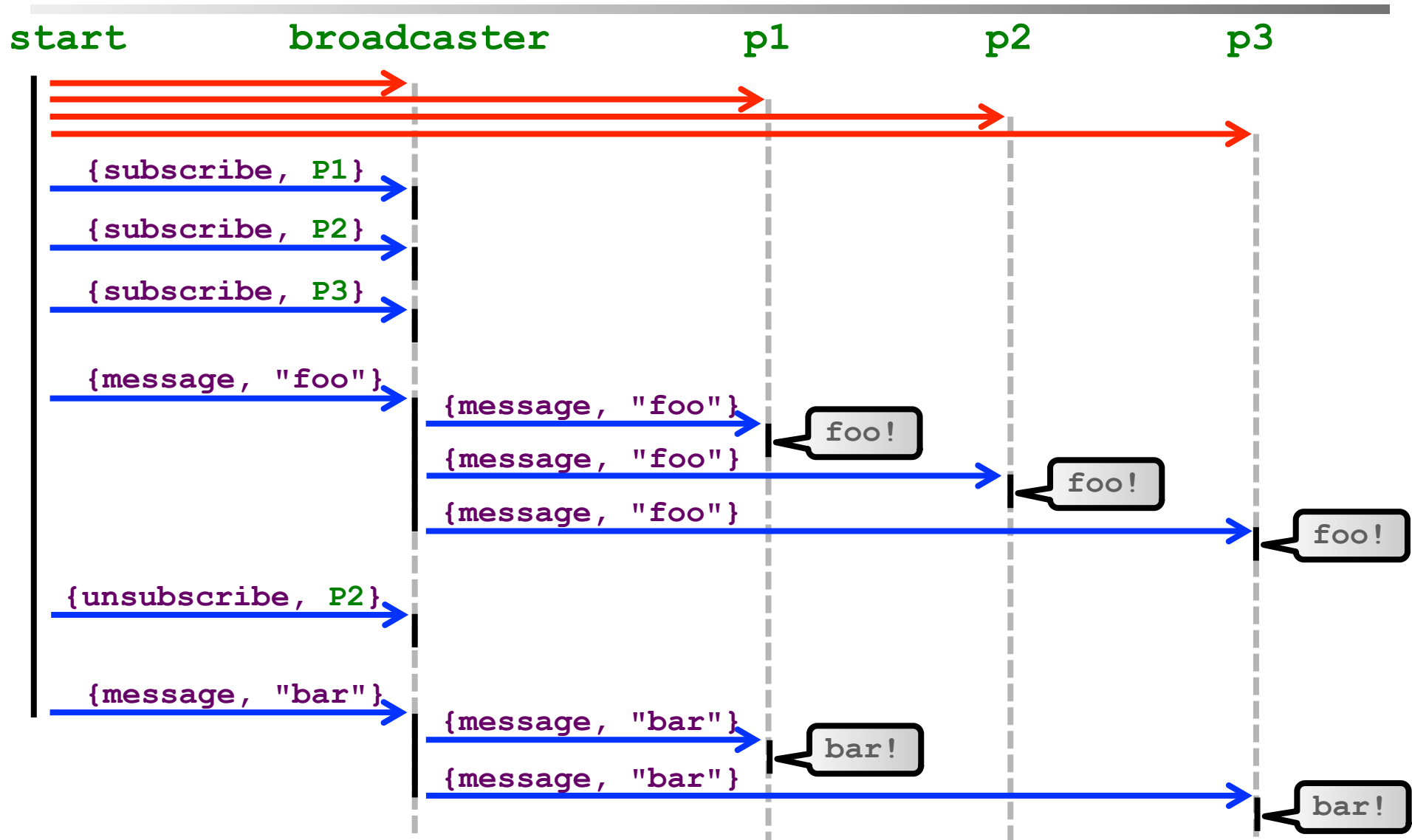
■ Run:

```
java -cp scala.jar:akka-actor.jar:akka-config.jar:. Ecco
```

■ Output:

```
Press return to terminate...  
[says]:  hvad drikker moller  
[hears]: ...oller  
[hears]: ...oller  
[hears]: ...oller
```

3) Broadcast



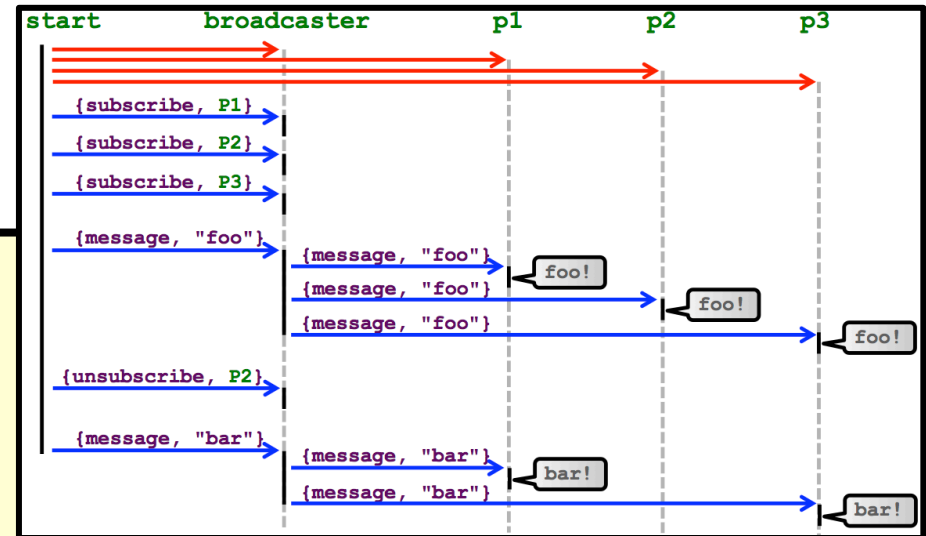
3) Broadcast.erl

```
-module(helloworld) .
-export([start/0, person/0, broadcaster/1]) .

person() ->
    receive
        {message, M} ->
            io:fwrite(M ++ "\n"),
            person()
    end.

broadcast([], _) -> true;
broadcast([Pid|L], M) ->
    Pid ! {message, M},
    broadcast(L, M) .

broadcaster(L) ->
    receive
        {subscribe, Pid} ->
            broadcaster([Pid|L]) ;
        {unsubscribe, Pid} ->
            broadcaster(lists:delete(Pid, L)) ;
        {message, M} ->
            broadcast(L, M),
            broadcaster(L)
    end.
```



```
start() ->
    Broadcaster = spawn(helloworld, broadcaster, [[]]),
    P1 = spawn(helloworld, person, []),
    P2 = spawn(helloworld, person, []),
    P3 = spawn(helloworld, person, []),
    Broadcaster ! {subscribe, P1},
    Broadcaster ! {subscribe, P2},
    Broadcaster ! {subscribe, P3},
    Broadcaster ! {message, "Purses half price!"},
    Broadcaster ! {unsubscribe, P2},
    Broadcaster ! {message, "Shoes half price!!"}.
```

```
purses half price!
purses half price!
purses half price!
shoes half price!!
shoes half price!!
```

3) Broadcast.java

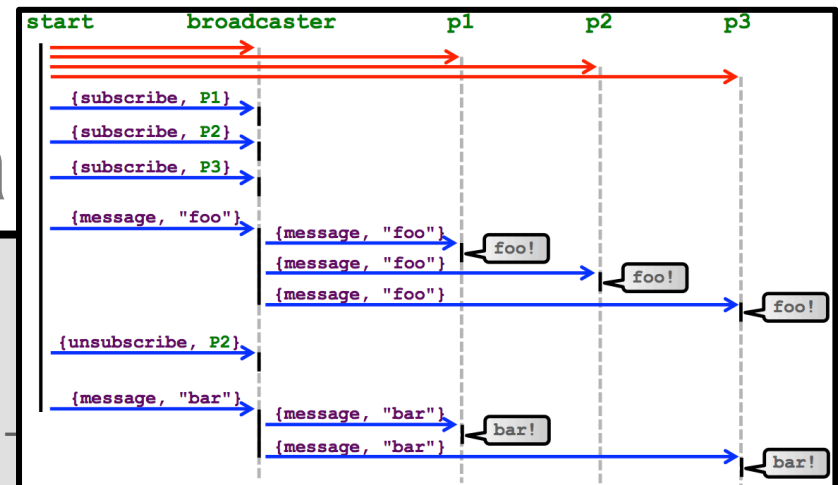
```
import java.util.*;
import java.io.*;
import akka.actor.*;
```

```
// -- MESSAGES -----
```

```
class SubscribeMessage implements Serializable {
    public final ActorRef subscriber;
    public SubscribeMessage(ActorRef subscriber) {
        this.subscriber = subscriber;
    }
}
```

```
class UnsubscribeMessage implements Serializable {
    public final ActorRef unsubsubscriber;
    public UnsubscribeMessage(ActorRef unsubsubscriber) {
        this.unsubsubscriber = unsubsubscriber;
    }
}
```

```
class Message implements Serializable {
    public final String s;
    public Message(String s) {
        this.s = s;
    }
}
```



```
purses half price!
purses half price!
purses half price!
shoes half price!!
shoes half price!!
```

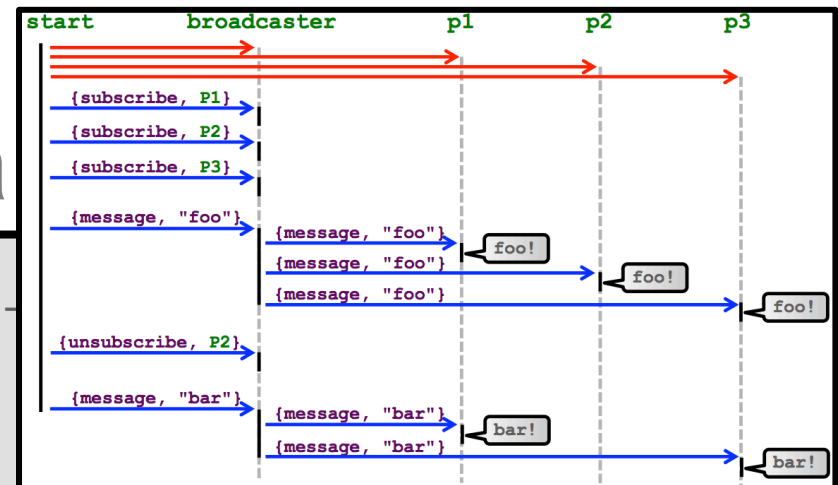
3) Broadcast.java

```
// -- ACTORS -----
```

```
class BroadcastActor extends UntypedActor {
    private List<ActorRef> list =
        new ArrayList<ActorRef>();

    public void onReceive(Object o) throws Exception {
        if (o instanceof SubscribeMessage) {
            list.add(((SubscribeMessage) o).subscriber);
        } else if (o instanceof UnsubscribeMessage) {
            list.remove(((UnsubscribeMessage) o).unsubscribe);
        } else if (o instanceof Message) {
            for (ActorRef person : list) {
                person.tell(o, getSelf());
            }
        }
    }
}

class PersonActor extends UntypedActor {
    public void onReceive(Object o) throws Exception {
        if (o instanceof Message) {
            System.out.println(((Message) o).s);
        }
    }
}
```

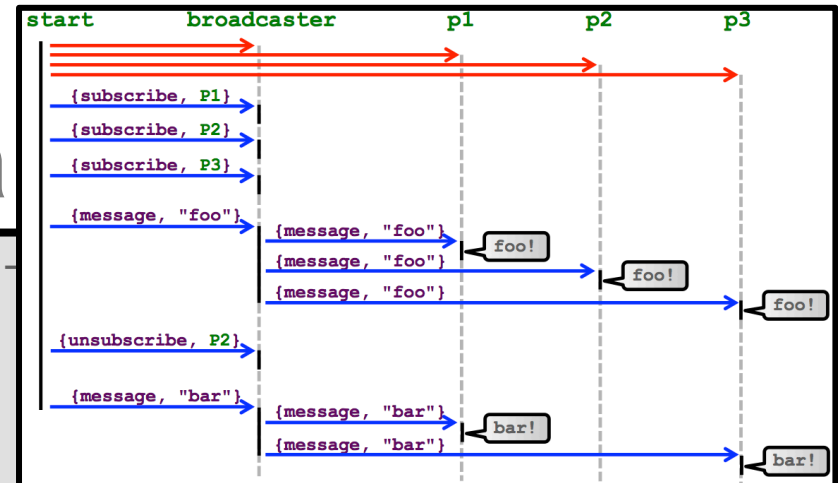


```
purses half price!
purses half price!
purses half price!
shoes half price!!
shoes half price!!
```


3) Broadcast.java

```
// -- MAIN -----

public class Broadcast {
    public static void main(String[] args) {
        final ActorSystem system =
            ActorSystem.create("EccoSystem");
        final ActorRef broadcaster =
            system.actorOf(Props.create(BroadcastActor.class), "broadcaster");
        final ActorRef p1 = system.actorOf(Props.create(PersonActor.class), "p1");
        final ActorRef p2 = system.actorOf(Props.create(PersonActor.class), "p2");
        final ActorRef p3 = system.actorOf(Props.create(PersonActor.class), "p3");
        broadcaster.tell(new SubscribeMessage(p1), ActorRef.noSender());
        broadcaster.tell(new SubscribeMessage(p2), ActorRef.noSender());
        broadcaster.tell(new SubscribeMessage(p3), ActorRef.noSender());
        broadcaster.tell(new Message("purses half price!"), ActorRef.noSender());
        broadcaster.tell(new UnsubscribeMessage(p2), ActorRef.noSender());
        broadcaster.tell(new Message("shoes half price!!"), ActorRef.noSender());
        try {
            System.out.println("Press return to terminate...");
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            system.shutdown();
        }
    }
}
```



```
purses half price!
purses half price!
purses half price!
shoes half price!!
shoes half price!!
```

3) Broadcast.java

■ Compile:

```
javac -cp scala.jar:akka-actor.jar Broadcast.java
```

■ Run:

```
java -cp scala.jar:akka-actor.jar:akka-config.jar:. Broadcast
```

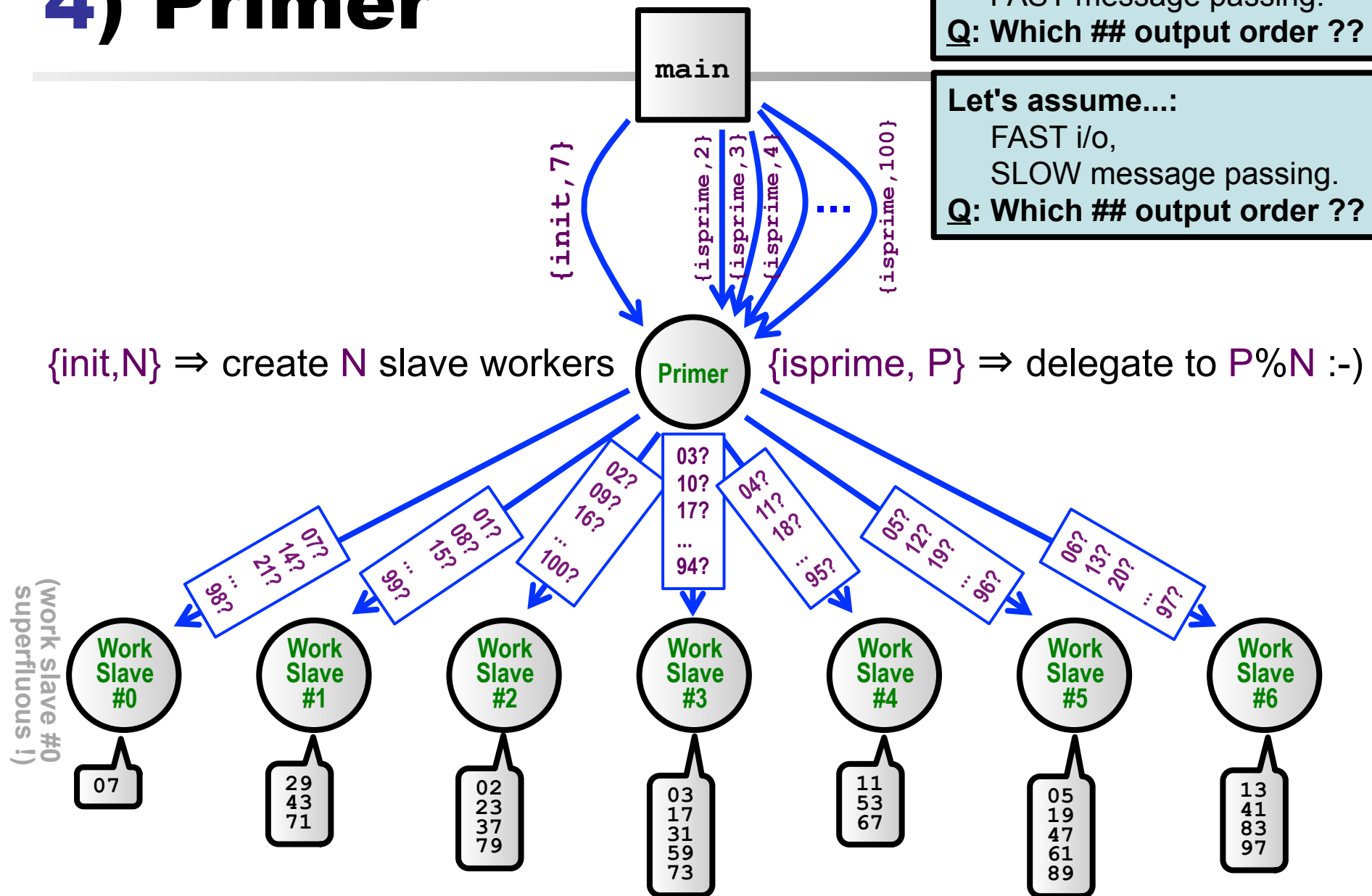
■ Output:

```
purses half price!  
purses half price!  
purses half price!  
shoes half price!!  
shoes half price!!
```

4) Primer

Let's assume...:
SLOW i/o,
FAST message passing.
Q: Which ## output order ??

Let's assume...:
FAST i/o,
SLOW message passing.
Q: Which ## output order ??



4) Primer.erl

Slave

```
-module(helloworld).
-export([start/0,slave/1,primer/1]).

is_prime_loop(N,K) ->
    K2 = K * K, R = N rem K,
    case (K2 <= N) and (R /= 0) of
        true -> is_prime_loop(N, K+1);
        false -> K
    end.

is_prime(N) ->
    K = is_prime_loop(N,2),
    (N >= 2) and (K*K > N).

n2s(N) ->
    lists:flatten(io_lib:format("~p", [N])).

slave(Id) ->
    receive
        {isprime, N} ->
            case is_prime(N) of
                true -> io:fwrite("(" ++
n2s(Id) ++ ") " ++ n2s(N) ++ "\n");
                false -> []
            end,
        slave(Id)
    end.

end.
```

Primer

```
create_slaves(Max,Max) -> [];
create_slaves(Id,Max) ->
    Slave = spawn(helloworld,slave,[Id]),
    [Slave|create_slaves(Id+1,Max)].

primer(Slaves) ->
    receive
        {init, N} when N<=0 ->
            throw({nonpositive,N}) ;
        {init, N} ->
            primer(create_slaves(0,N)) ;
        {isprime, _} when Slaves == [] ->
            throw({uninitialized}) ;
        {isprime, N} when N<=0 ->
            throw({nonpositive,N}) ;
        {isprime, N} ->
            SlaveId = N rem length(Slaves),
            lists:nth(SlaveId+1, Slaves)
                ! {isprime,N},
            primer(Slaves)
    end.

spam(_, N, Max) when N>=Max -> true;
spam(Primer, N, Max) ->
    Primer ! {isprime, N},
    spam(Primer, N+1, Max).

start() ->
    Primer =
        spawn(helloworld, primer, [[]]),
    Primer ! {init,7},
    spam(Primer, 2, 100).
```


4) Primer.java

```
import java.util.*;
import java.io.*;
import akka.actor.*;

// -- MESSAGES -----

class InitializeMessage implements Serializable {
    public final int number_of_slaves;
    public InitializeMessage(int number_of_slaves) {
        this.number_of_slaves = number_of_slaves;
    }
}

class IsPrimeMessage implements Serializable {
    public final int number;
    public IsPrimeMessage(int number) {
        this.number = number;
    }
}
```



4) Primer.java

```
// -- SLAVE ACTOR -----

class SlaveActor extends UntypedActor {
    private boolean isPrime(int n) {
        int k = 2;
        while (k * k <= n && n % k != 0) k++;
        return n >= 2 && k * k > n;
    }

    private int delay(int n) {
        int res = 0;
        for (int i=0; i<10000000*n; i++) {
            res = res + i;
        }
        return res;
    }

    public void onReceive(Object o) throws Exception {
        if (delay(p)==p) System.out.println("will not happen: "); // artificial delay
        if (o instanceof IsPrimeMessage) {
            int p = ((IsPrimeMessage) o).number;
            if (isPrime(p)) System.out.println("(" + p%7 + ") " + p); %% HACK: 7 !
        }
    }
}
```

4) Primer.java

```
// -- PRIME ACTOR -----
class PrimeActor extends UntypedActor {
    List<ActorRef> slaves;

    private List<ActorRef> createSlaves(int n) {
        List<ActorRef> slaves = new ArrayList<ActorRef>();
        for (int i=0; i<n; i++) {
            ActorRef slave =
                getContext().actorOf(Props.create(SlaveActor.class), "p" + i);
            slaves.add(slave);
        }
        return slaves;
    }

    public void onReceive(Object o) throws Exception {
        if (o instanceof InitializeMessage) {
            InitializeMessage init = (InitializeMessage) o;
            int n = init.number_of_slaves;
            if (n<=0) throw new RuntimeException("*** non-positive number!");
            slaves = createSlaves(n);
            System.out.println("initialized (" + n + " slaves ready to work)!");
        } else if (o instanceof IsPrimeMessage) {
            if (slaves==null) throw new RuntimeException("*** uninitialized!");
            int n = ((IsPrimeMessage) o).number;
            if (n<=0) throw new RuntimeException("*** non-positive number!");
            int slave_id = n % slaves.size();
            slaves.get(slave_id).tell(o, getSelf());
        }
    }
}
```

4) Primer.java



```
// -- MAIN -----

public class Primer {
    private static void spam(ActorRef primer, int min, int max) {
        for (int i=min; i<max; i++) {
            primer.tell(new IsPrimeMessage(i), ActorRef.noSender());
        }
    }

    public static void main(String[] args) {
        final ActorSystem system = ActorSystem.create("PrimerSystem");
        final ActorRef primer =
            system.actorOf(Props.create(PrimeActor.class), "primer");
        primer.tell(new InitializeMessage(7), ActorRef.noSender());
        try {
            System.out.println("Press return to initiate...");
            System.in.read();
            spam(primer, 2, 100);
            System.out.println("Press return to terminate...");
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            system.shutdown();
        }
    }
}
```


4) Primer.java

■ Compile:

```
javac -cp scala.jar:akka-actor.jar Primer.java
```

■ Run:

```
java -cp scala.jar:akka-actor.jar:akka-config.jar:. Primer
```

■ Output:

```
press return to initiate...  
initialized (7 slaves ready to work)!
```

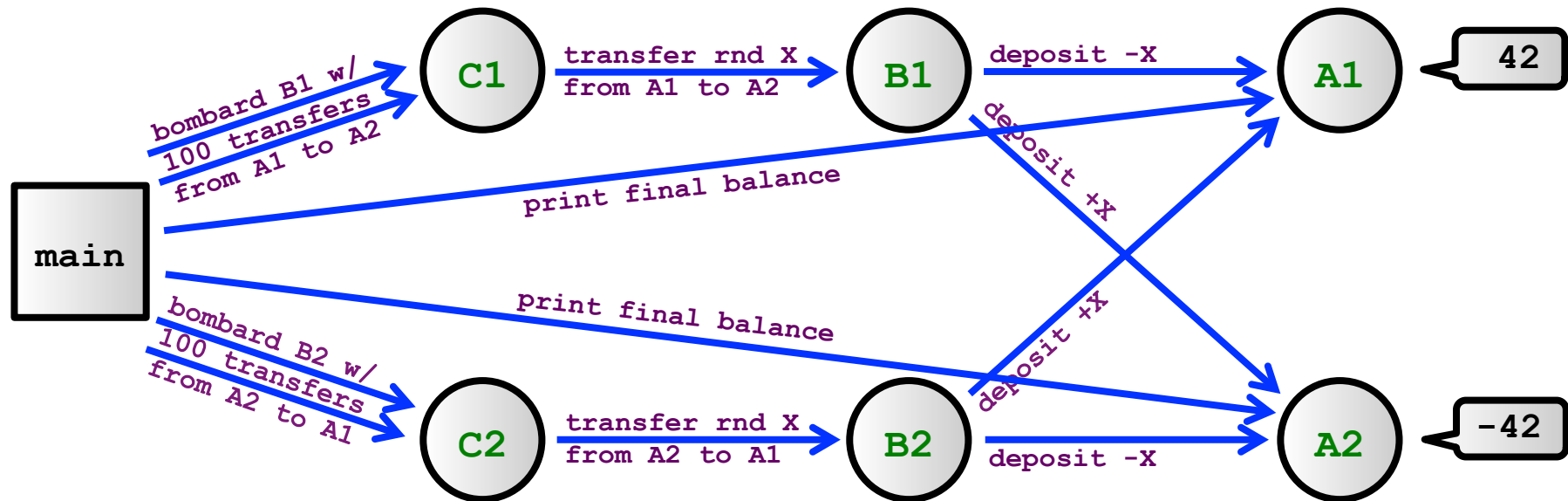
```
(2) 2  
(3) 3  
Press return to terminate...  
(0) 7  
(5) 5  
(4) 11  
(6) 13  
(3) 17  
(5) 19  
(2) 23  
(1) 29  
(3) 31
```

```
(2) 37  
(6) 41  
(1) 43  
(5) 47  
(4) 53  
(3) 59  
(5) 61  
(4) 67  
(1) 71  
(3) 73  
(2) 79  
(6) 83  
(5) 89  
(6) 97
```

Mandatory Hand-in Exercise

For Message Passing

5) ABC (Clerk / Bank / Account)



5) ABC.erl

```
-module(helloworld) .
-export([start/0,
        account/1,bank/0,clerk/0]) .

%% -- BASIC PROCESSING -----
n2s(N) -> lists:flatten( %% int2string
    io_lib:format("~p", [N])). %% HACK!

random(N) -> random:uniform(N) div 10.

%% -- ACTORS -----

account(Balance) ->
    receive
        {deposit,Amount} ->
            account(Balance+Amount) ;
        {printbalance} ->
            io:fwrite(n2s(Balance) ++ "\n")
    end.

bank() ->
    receive
        {transfer,Amount,From,To} ->
            From ! {deposit,-Amount},
            To ! {deposit,+Amount},
            bank()
    end.
```

```
ntransfers(0,_,_,_) -> true;
ntransfers(N,Bank,From,To) ->
    R = random(100),
    Bank ! {transfer,R,From,To},
    ntransfers(N-1,Bank,From,To) .

clerk() ->
    receive
        {start,Bank,From,To} ->
            random:seed(now()),
            ntransfers(100,Bank,From,To),
            clerk()
    end.

start() ->
    A1 = spawn(helloworld,account,[0]),
    A2 = spawn(helloworld,account,[0]),
    B1 = spawn(helloworld,bank,[]),
    B2 = spawn(helloworld,bank,[]),
    C1 = spawn(helloworld,clerk,[]),
    C2 = spawn(helloworld,clerk,[]),
    C1 ! {start,B1,A1,A2},
    C2 ! {start,B2,A2,A1},
    timer:sleep(1000),
    A1 ! {printbalance},
    A2 ! {printbalance}.
```

5) ABC.java

(Skeleton)

```
import java.util.Random;   import java.io.*;   import akka.actor.*;

// -- MESSAGES -----
class StartTransferMessage implements Serializable { /* TODO */ }
class TransferMessage implements Serializable { /* TODO */ }
class DepositMessage implements Serializable { /* TODO */ }
class PrintBalanceMessage implements Serializable { /* TODO */ }

// -- ACTORS -----
class AccountActor extends UntypedActor { /* TODO */ }
class BankActor extends UntypedActor { /* TODO */ }
class ClerkActor extends UntypedActor { /* TODO */ }

// -- MAIN -----
public class ABC { // Demo showing how things work:
    public static void main(String[] args) {
        final ActorSystem system = ActorSystem.create("ABCSystem");

        /* TODO (CREATE ACTORS AND SEND START MESSAGES) */

        try {
            System.out.println("Press return to inspect...");
            System.in.read();

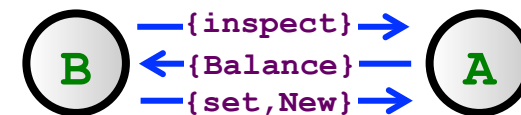
            /* TODO (INSPECT FINAL BALANCES) */

            System.out.println("Press return to terminate...");
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            system.shutdown();
        }
    }
}
```

MANDATORY HAND-IN!

a) Implement ABC.java
(as close to ABC.erl as possible,
but without using "tail-recursion")

b) Answer question:
What happens if we replace
{deposit, ±Amount} w/ the msgs?:



*** OUTPUT ***

```
Press return to inspect...
Press return to terminate...
Balance = 42
Balance = -42
```

Thx!

Questions?

Four Conditions \Leftrightarrow Deadlock !

1) **Mutual exclusion condition** (aka. "Serially reusable resources"):

- Processes involved share resources which they use under mutual exclusion.

2) **Hold-and-wait condition** (aka. "Incremental acquisition"):

- Processes hold on to resources already allocated to them while waiting to acquire additional resources.

3) **No pre-emption condition**:

- Resources cannot be "pre-empted" (i.e., forcibly withdrawn) once acquired by a process, but are only released voluntarily.

4) **Circular-wait condition** (aka. "Wait-for cycle"):

- a circular chain (cycle) of processes exists such that each process holds a resource which its successor in the cycle is waiting to acquire.

[Magee & Kramer]