pseuCo.com

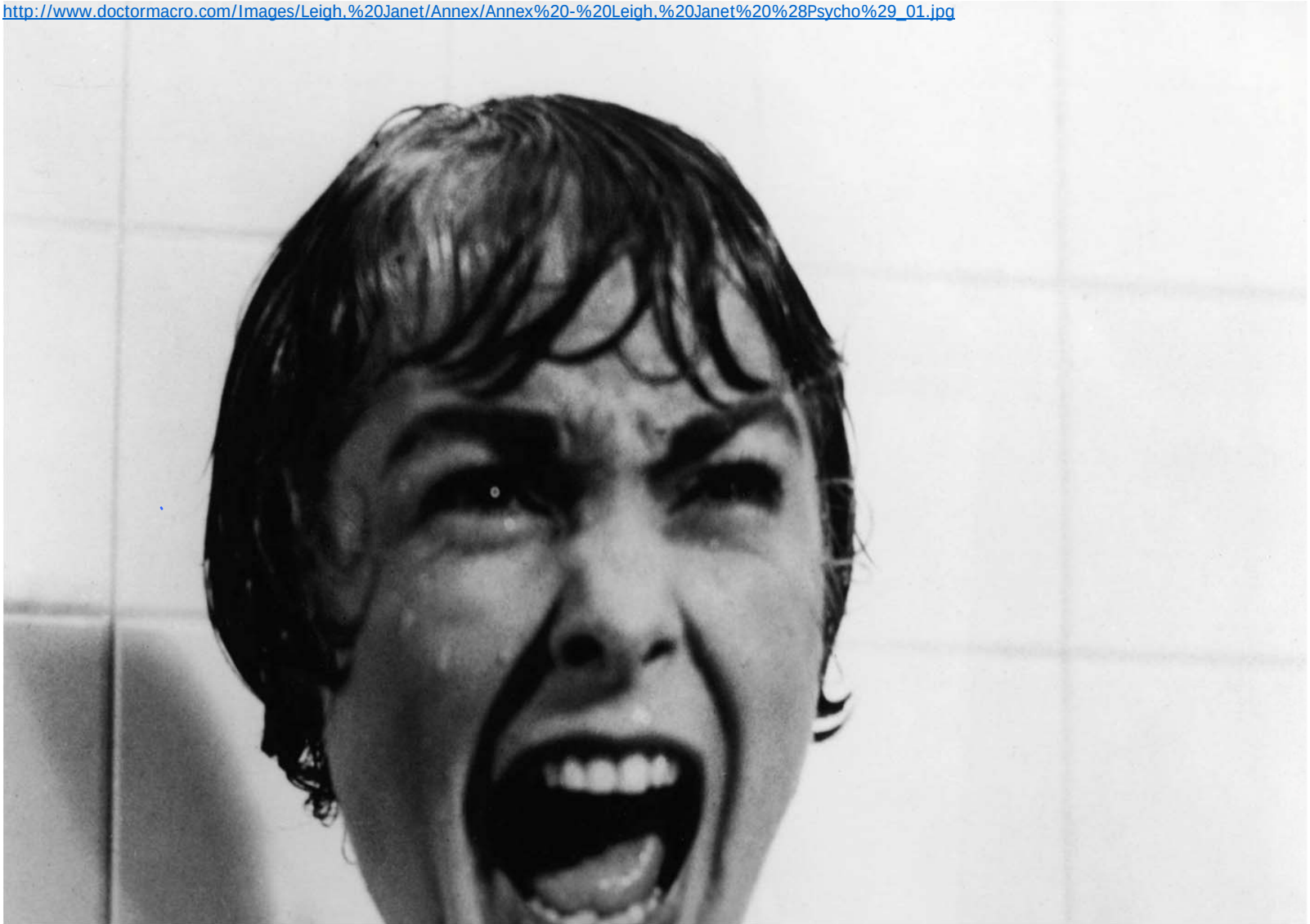
Holger Hermanns

Saarland University

joint work with Christian Eisentraut

and with Markus Bauer, Sebastian Biewer, Lisa Detzler, Felix Freiberger, Pascal Held, Marc Jose, and Janis Sprenger

http://www.doctormacro.com/Images/Leigh,%20Janet/Annex/Annex%20-%20Leigh,%20Janet%20%28Psycho%29_01.jpg



pseuCo.com

This is a web application allowing to create pseuCo and CCS files, compile pseuCo to CCS, and interactively explore the semantics of both. Support for CCS is stable, support for pseuCo is under hectic development.

Get help Get started

About CCS

Editing unsaved file CCS

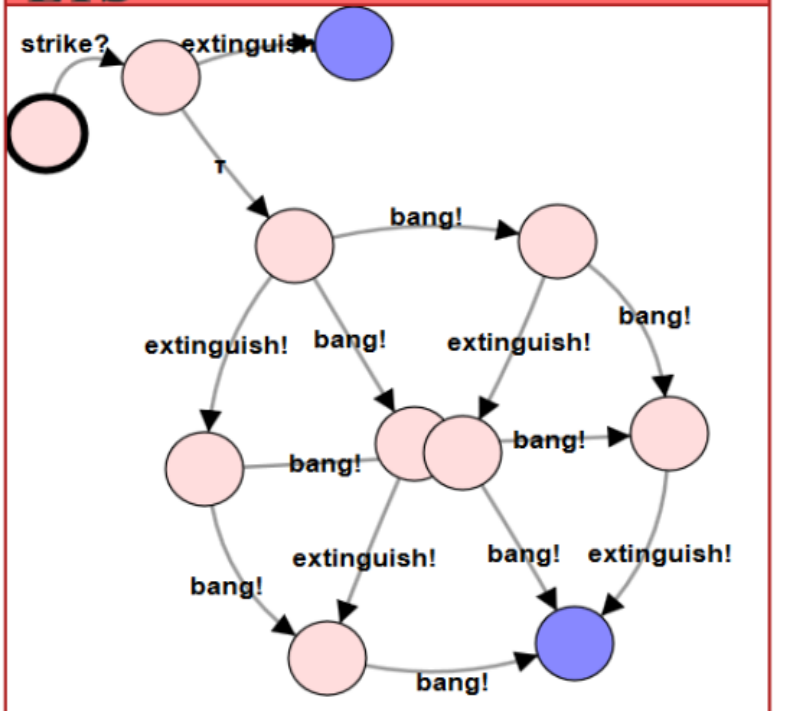
CCS

```
1  (* a simple example to demonstrate CCS syntax *)
2
3  (* first some defining equations*)
4
5  Match := strike?. MatchOnFire
6  MatchOnFire:= light!. MatchOnFire + extinguish!.0
7  TwoFireCracker := light?. (bang!. 0 | bang!. 0)
8
9  (Match | TwoFireCracker) \ {light} // this is the initial process
```

We use a pragmatic extension of CCS supporting value passing (of Booleans, integers, and strings), sequencing and termination.

No issues.

LTS



Collapse all

Random path Export LTS Share this file

pseuCo.com Files Backup Help About

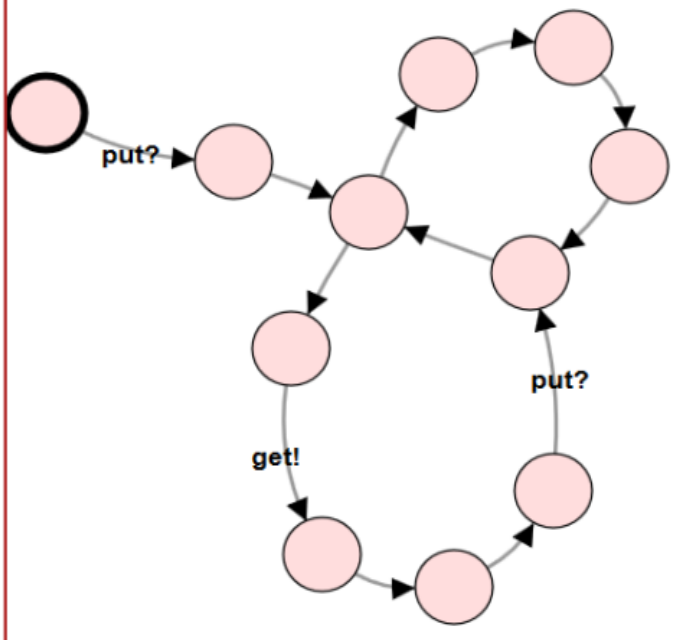
Editing Protocol CCS

CCS

```
1  DupMedium := AckMedium | Medium
2
3  Medium := send?. (receive!. Medium + i
   .garbled!.Medium)
4  AckMedium := sendAck?. receiveAck!.
   AckMedium
5          + sendNack?. receiveNack!.
   AckMedium
6
7  Sender := put?. send!. Sending
8  Sending := receiveAck?. Sender
9          + receiveNack?. send!.
   Sending
10
11 Receiver := receive?. get!. sendAck!.
   Receiver
12         + garbled?. sendNack!.
   Receiver
13
14 Protocol := (Sender | Receiver |
   DupMedium)
15         \{send,receive,sendAck
```

multiple issues, first one: warning in

LTS



Collapse all

Random path Export LTS Share this file

pseuCo.com File

Editing Value Passing

CCS

```
1  // defining a range c
2  range Dig := 0..9
3
4  DupMedium := AckMediu
5
6  Medium := send?x:Dig.
   + i. garbled!.Medium)
7  AckMedium := sendAck
   AckMedium
8          + sendNack
   AckMedium
9
10 Sender := put?x. send
11 Sending[x] := receive
12         + receive
   Sending[x]
13
14 Receiver := receive?x
   sendAck!. Receiver
15         + garbled?.
16
17 Protocol := (Sender |
   DupMedium)
```

multiple issues, first

Random path

The following shows a random path throughout the LTS, until a terminal state is reached.

- [x] only show the trace
- [x] hide τ-steps

Restart

↓ get!(2)
↓ get!(4)
↓ get!(2)
↓ get!(8)
No more transitions.
55 τ-steps are hidden.

Close

Open Problems in Concurrency ~~Theory~~ Practice

Bertinoro (Italy), 18-21 June 201

- Understanding Concurrent Programs
- Writing Concurrent Programs

Concurrent Programming

A mandatory module for Bachelor students in CS

- 2nd year
- 6 CP ECTS
- 18 Lectures
- 8 Tutorials
- 1 Programming project

Anstrengender als ein 9CP Modul

Didaktisch ist die Veranstaltung beliebigen anderen um ein Vielfaches überlegen!

exzellent!

Urkunde

Der Fakultätentag Informatik der Universitäten der Bundesrepublik Deutschland verleiht den Herren

Christian Eisentraut, MSc.
Prof. Dr. Holger Hermanns
von der Universität des Saarlandes

den mit 2.500 € dotierten Preis des Jahres 2013 für eine theoretische Grundlagenvorlesung im Bachelor-Studiengang Informatik. Die von den Preisträgern konzipierte und seit Jahren durchgeführte Lehrveranstaltung

Nebenläufige Programmierung

wurde wegen ihres innovativen Ansatzes ausgewählt, der eine Einführung in die Theorie der Nebenläufigkeit anhand klassischer Prozesskalküle mit praxisnahen Programmieraufgaben und innovativen Übungskonzepten kombiniert.

Berlin, den 22. November 2013

Prof. Dr. H.-U. Heiß
1. Vorsitzender

Fakultätentag Informatik

Concurrent Programming

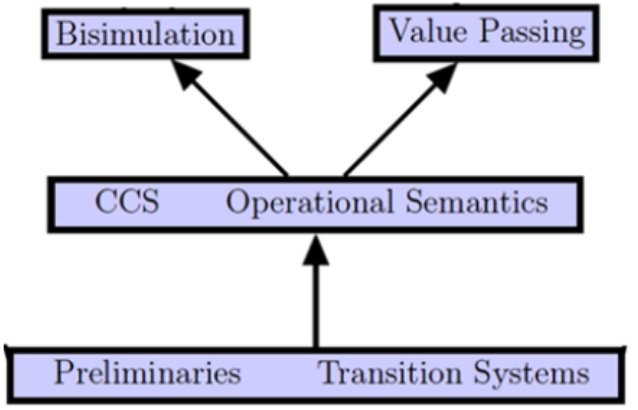
.... is a mess anyway.

Needs Understanding of Concurrency Theory.

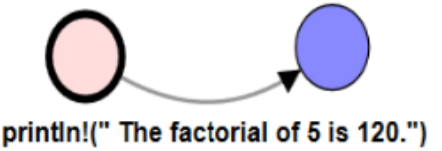
But, where is the bridge from theory to mess?

What we do: We first introduce

Transition Systems - CCS - operational semantics - partial order semantics - observational congruence - value passing



pseuCo, a simple pseudo code



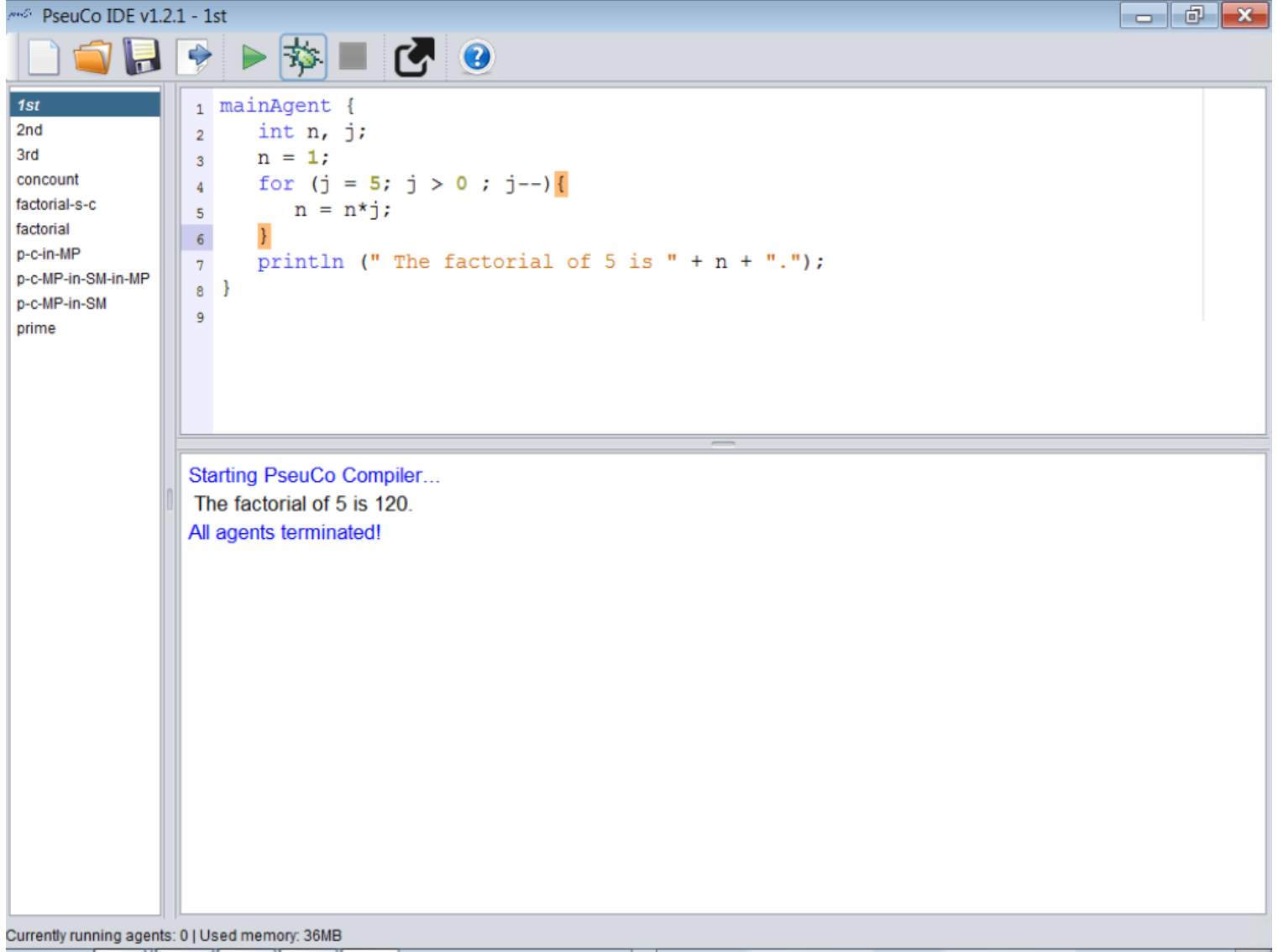
... with a CCS semantics

```
mainAgent {
   int n, j;
   n = 1;
   for (j = 5; j > 0 ; j--){
      n = n*j;
   }
   println (" The factorial of 5 is " + n + ".");
}
```

`mainAgent` is always there.

Syntax is stripped down Java.

Otherwise there is nothing special.


PseuCo IDE v1.2.1 - 1st
1st
2nd
3rd
concount
factorial-s-c
factorial
p-c-in-MP
p-c-MP-in-SM-in-MP
p-c-MP-in-SM
prime
1 mainAgent {
2 int n, j;
3 n = 1;
4 for (j = 5; j > 0 ; j--){
5 n = n*j;
6 }
7 println (" The factorial of 5 is " + n + ".");
8 }
9
Starting PseuCo Compiler...
The factorial of 5 is 120.
All agents terminated!
Currently running agents: 0 | Used memory: 36MB

pseuCo.com Files Backup Help About

Editing unsaved file CCS

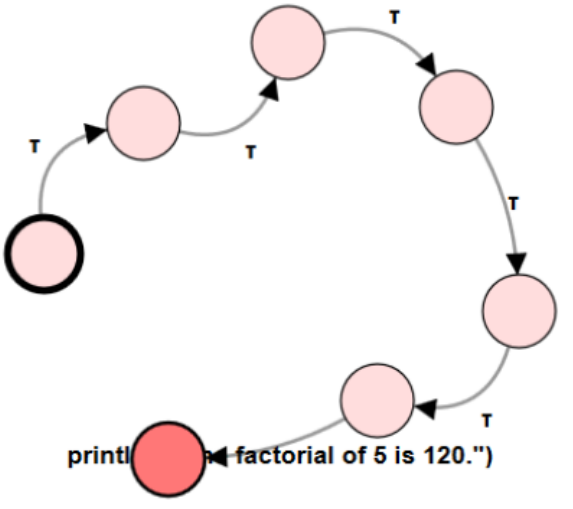
This file has no name and is only saved temporarily. If you want to keep it, enter a name.

CCS

```
).AgentManager[next_1+1]
10  AgentJoiner[a, c] := join_register(a
    )?.AgentJoiner[a, c+1] + agent_terminate(a
    )?.JoinDistributor[a, c]
11  JoinDistributor[a, c] := when (c<=0)
    join_register(a)?.join(a)?.JoinDistributor
    [a, 0] + when (c>0) join(a)?.JoinDistribut
    or[a, c-1]
12  MainAgent := MainAgent_1[1, 5]
13  MainAgent_1[$n, $j] := when (!($j>0))
    MainAgent_2[$n, $j] + when ($j>0) i.
    MainAgent_1[$n*$j, $j-1]
14  MainAgent_2[$n, $j] := println!("The
    factorial of 5 is "^$n^".").0
15
16  MainAgent | Mutex_cons[1] | WaitRoom_cons[
    1] | ArrayManager[1] | ChannelManager[1] |
    AgentManager[1] | Return \{*, println,
    exception}
```

⚠ multiple issues, first one: warning in line 4:

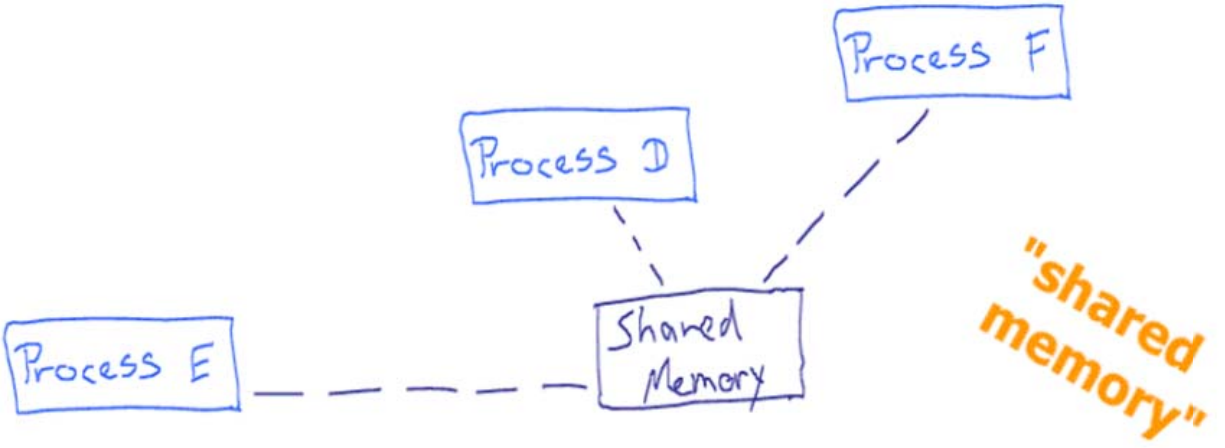
LTS



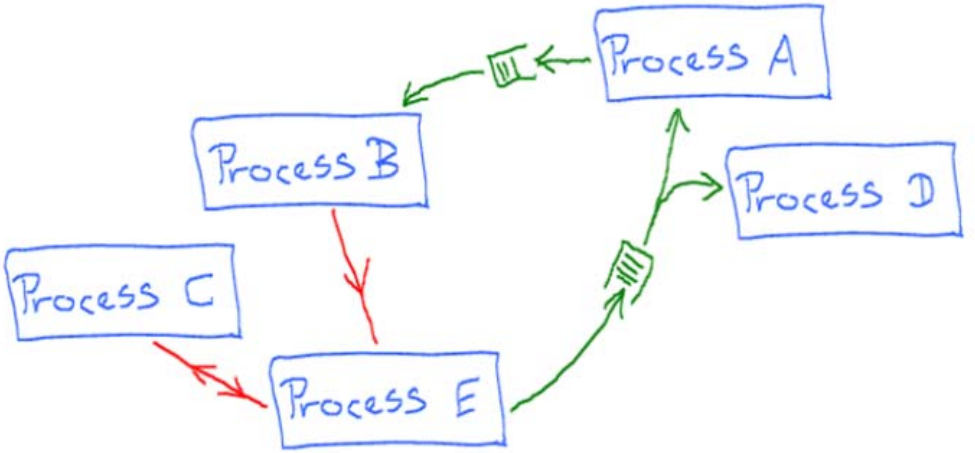
Expand all

Random path Export LTS Share this file

Concurrent Programming



Concurrent Programming



„message passing"

pseuCo Message Passing

- Typical imperative programming
  (no dynamic data structures)

- Agents - including **`mainAgent`**

- Channels typed, fixed capacity
  - synchronous **`intchan`**
  - asynchronous **`intchan3`**

- Receive **`<?`**

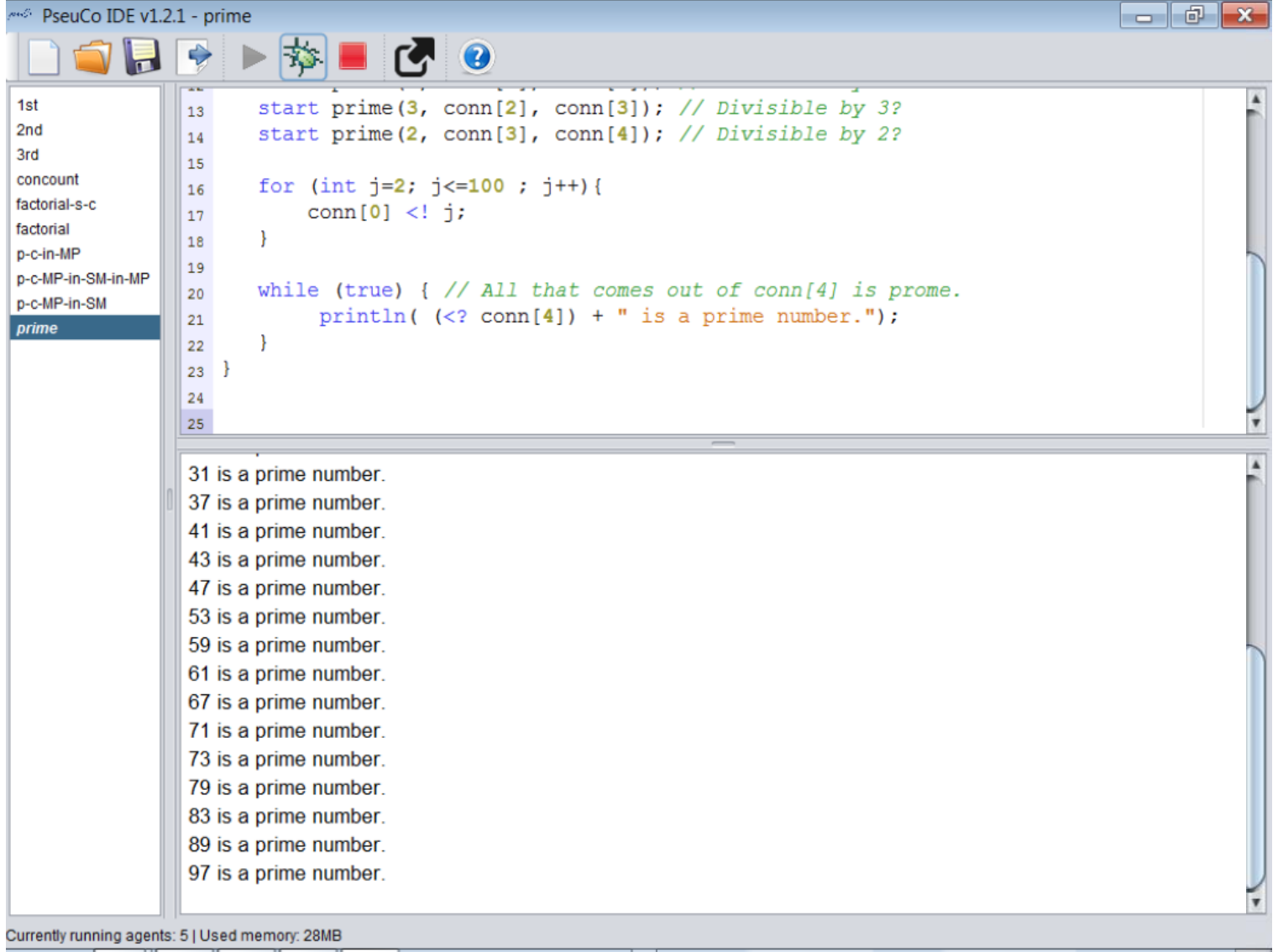
- Send **`<!`**

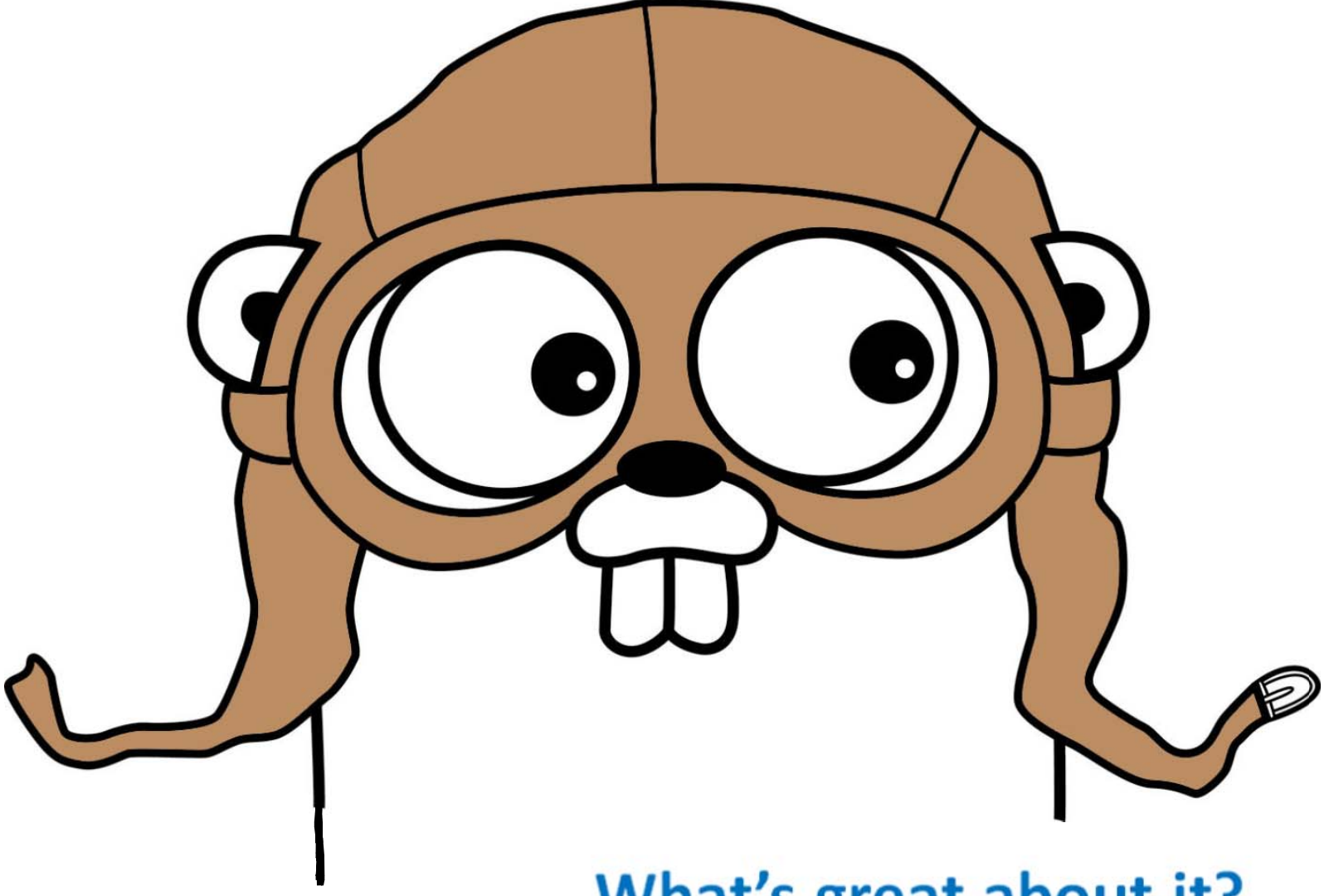
- **`select-case`**

```
Chn[fill,v0,v1,v2] := when (fill == 0) put?x. Chn[1,x,v1,v2]
                    + when (fill == 1) put?x. Chn[2,v1,x,v2]
                    + when (fill == 2) put?x. Chn[3,v1,v2,x]
                    + when (fill > 0) get!v0. Chn[fill-1,v1,v2,0]
```

- Nothing else

No access to shared memory!


PseuCo IDE v1.2.1 - prime
1st
2nd
3rd
concount
factorial-s-c
factorial
p-c-in-MP
p-c-MP-in-SM-in-MP
p-c-MP-in-SM
prime
start prime(3, conn[2], conn[3]); // Divisible by 3?
start prime(2, conn[3], conn[4]); // Divisible by 2?
for (int j=2; j<=100 ; j++){
conn[0] <! j;
}
while (true) { // All that comes out of conn[4] is prome.
println((<? conn[4]) + " is a prime number.");
}
}
31 is a prime number.
37 is a prime number.
41 is a prime number.
43 is a prime number.
47 is a prime number.
53 is a prime number.
59 is a prime number.
61 is a prime number.
67 is a prime number.
71 is a prime number.
73 is a prime number.
79 is a prime number.
83 is a prime number.
89 is a prime number.
97 is a prime number.
Currently running agents: 5 | Used memory: 28MB



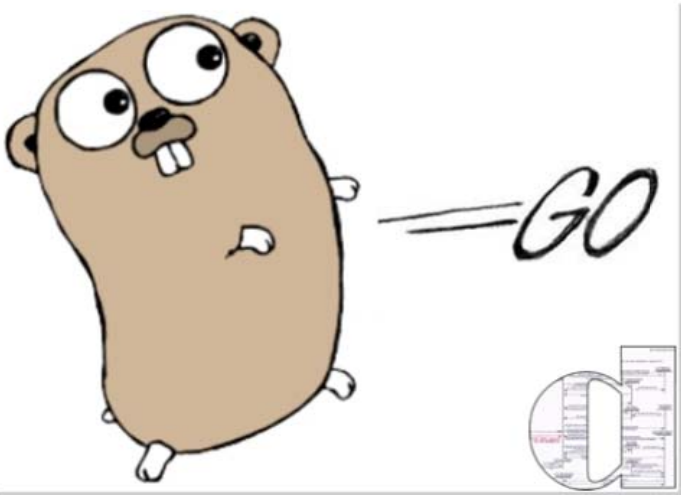
What's great about it?

```go
package main
import "fmt"
func prime(v int, in chan int, out chan int) {
	var p int
	for {
		p = <-in
		if p%v != 0 || p == v {
			out <- p
		}
	}
}

func main() {
	var conn [5]chan int
	for j := range conn {
		conn[j] = make(chan int, 100)
	}

	go prime(7, conn[0], conn[1])
	go prime(5, conn[1], conn[2])
	go prime(3, conn[2], conn[3])
	go prime(2, conn[3], conn[4])

	for j := 2; j <= 100; j++ {
		conn[0] <- j
	}

	for {
		fmt.Println((<-conn[4]),
			"is prime.")
	}
}
```


```
void prime(int v, intchan in, intchan out){
   int p;
   while (true) {
       p = <? in;
       if (p % v != 0 || p == v) {
           out <! p;
       }
   }
}

mainAgent{
   intchan100[5] conn;
   start prime(7, conn[0], conn[1]);
   start prime(5, conn[1], conn[2]);
   start prime(3, conn[2], conn[3]);
   start prime(2, conn[3], conn[4]];

   for (int j=2; j<=100 ; j++){
       conn[0] <! j;
   }

   while (true) {
        println( (<? conn[4]) +
                              " is prime.");
   }
}
```

What's great about it?

Google what?



Google what?

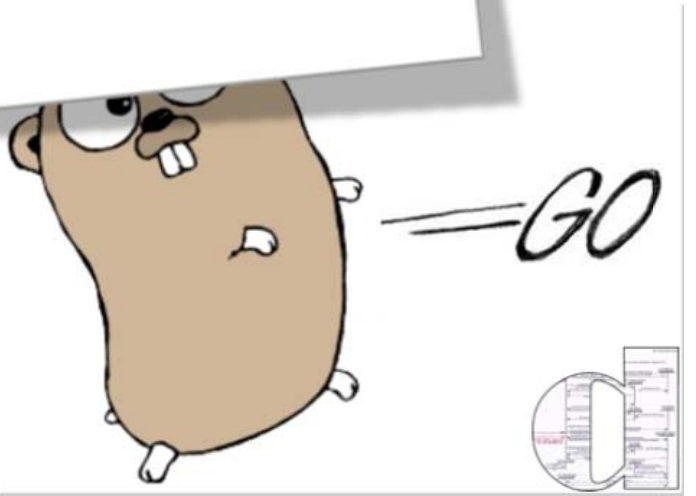
History

To many, the concurrency features of Go seemed new.

But they are rooted in a long history, reaching back to Hoare's CSP in 1978 and even Dijkstra's guarded commands (1975).

Languages with similar features:

- Occam (May, 1983)
- Erlang (Armstrong, 1986)
- Newsqueak (Pike, 1988)
- Concurrent ML (Reppy, 1993)
- Alef (Winterbottom, 1995)
- Limbo (Dorward, Pike, Winterbottom, 1996).



Google what?

History

To many, the concurrency features of Go seemed new.

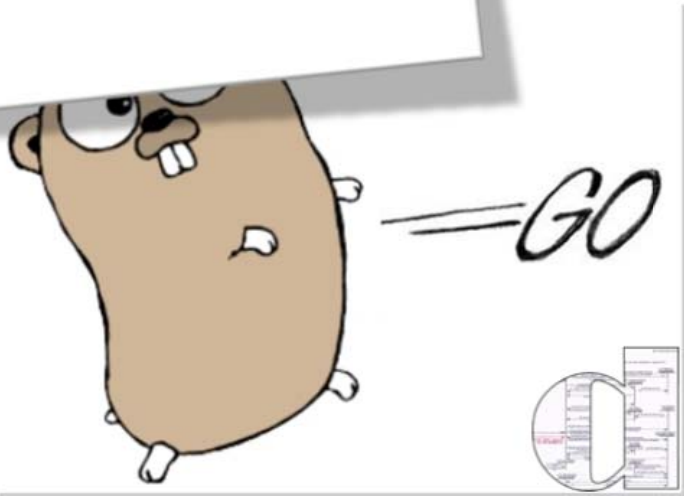
But they are rooted in a long history, reaching back to Hoare's CSP in 197
commands (1975).

Languages with similar features:

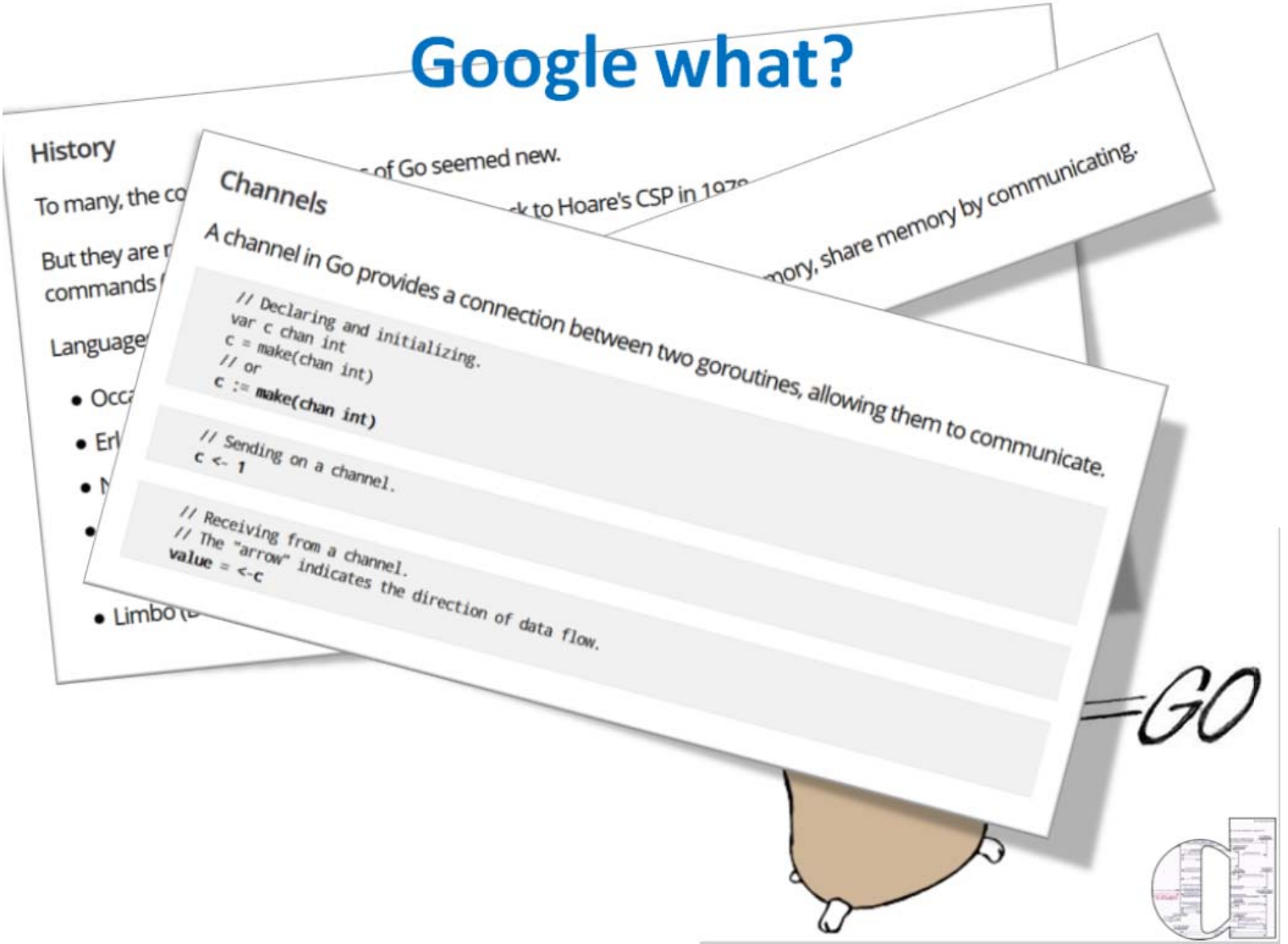
- Occam (May, 1983)
- Erlang (Armstrong, 1986)
- Newsqueak (Pike, 1988)
- Concurrent ML (Reppy, 1993)
- Alef (Winterbottom, 1995)
- Limbo (Dorward, Pike, Winterbottom, 1996).

The Go approach

Don't communicate by sharing memory, share memory by communicating.



Google what?


History
Channels
A channel in Go provides a connection between two goroutines, allowing them to communicate.
// Declaring and initializing.
var c chan int
c = make(chan int)
// or
c := make(chan int)
// Sending on a channel.
c <- 1
// Receiving from a channel.
// The "arrow" indicates the direction of data flow.
value = <-c
GO

Google what?

History

To many, the co

But they are r
commands

Language

- Occa
- Erl
- N

of Go seemed new.

k to Hoare's CSP in 197

mory, share memory by communicating.

Channels

A channel in Go provides a connection between two goroutines, allowing them to communicate.

```
// Declaring and initializing.
var c chan int
c = make(chan int)
// or
c := make(chan int)

// Sending on a channel.
c <- 1

// Receiving from a channel.
// The "arrow" indicates
value = <-c
```

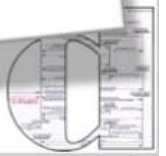
Synchronization

When the main function executes <–c, it will wait for a value to be sent.

Similarly, when the boring function executes c <– value, it waits for a receiver to be ready.

A sender and receiver must both be ready to play their part in the communication. Otherwise we wait until they are.

Thus channels both communicate and synchronize.



Google what?

History

To many, the co

But they are r
commands

Language

- Occa
- Erl
- N

Channels

A channel in Go provide

```
// Declaring and initia
var c chan int
c = make(chan int)
// or
c := make(chan int)

// Sending on a channel.
c <- 1

// Receiving from a channe
// The "arrow" indicates
value = <-c
```

Select

The select statement provides another way to handle multiple channels. It's like a switch, but each case is a communication:
- All channels are evaluated.
- Selection blocks until one communication can proceed, which then does.
- If multiple can proceed, select chooses pseudo-randomly.
- A default clause, if present, executes immediately if no channel is ready.

```
select {
case v1 := <-c1:
    fmt.Printf("received %v from c1\n", v1)
case v2 := <-c2:
    fmt.Printf("received %v from c2\n", v1)
case c3 <- 23:
    fmt.Printf("sent %v to c3\n", 23)
default:
    fmt.Printf("no one was ready to communicate\n")
}
```

Synchronization

When the main function ex

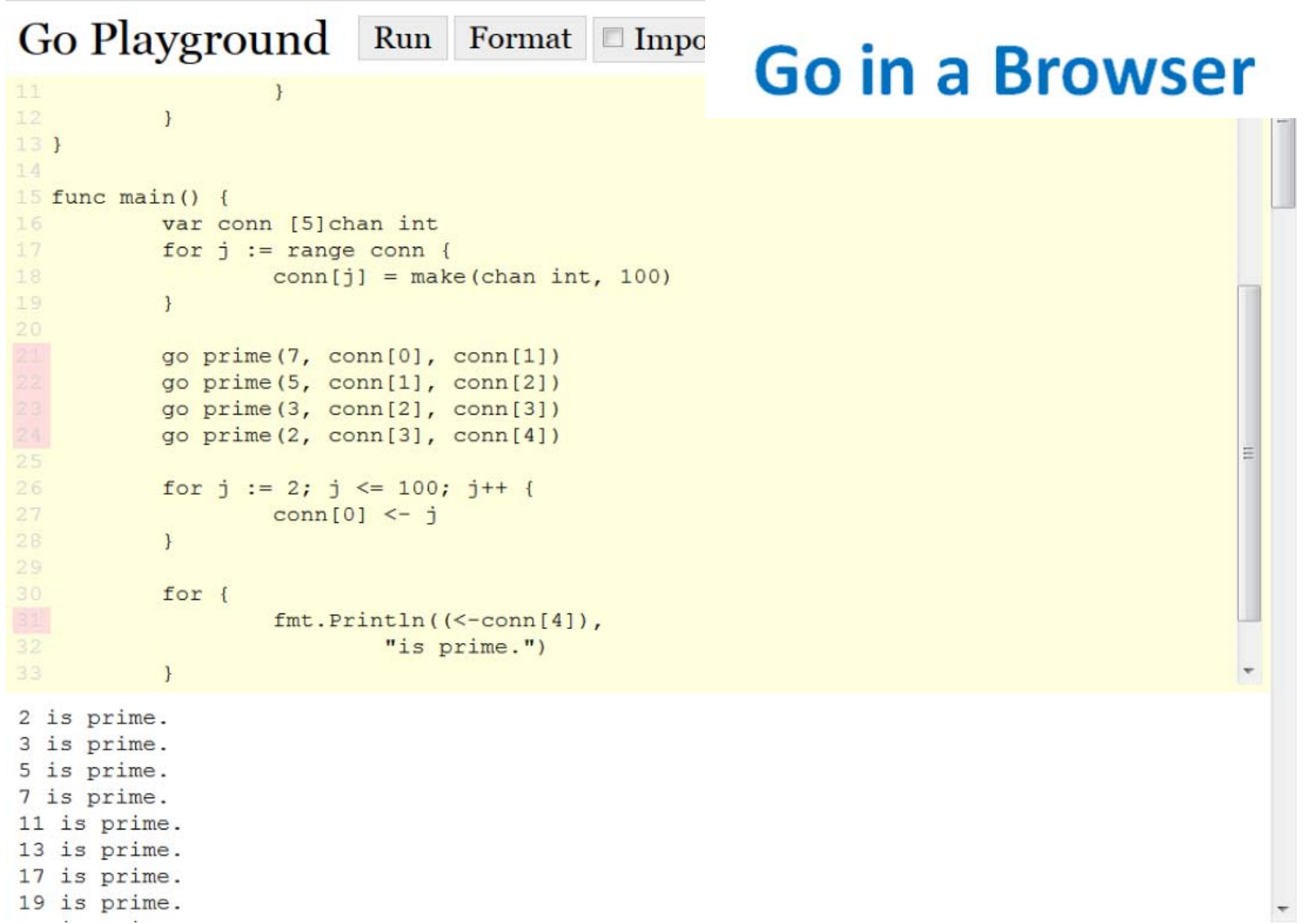
Similarly, when the boring function ex

A sender and receiver must both be ready to play their part in t
are.

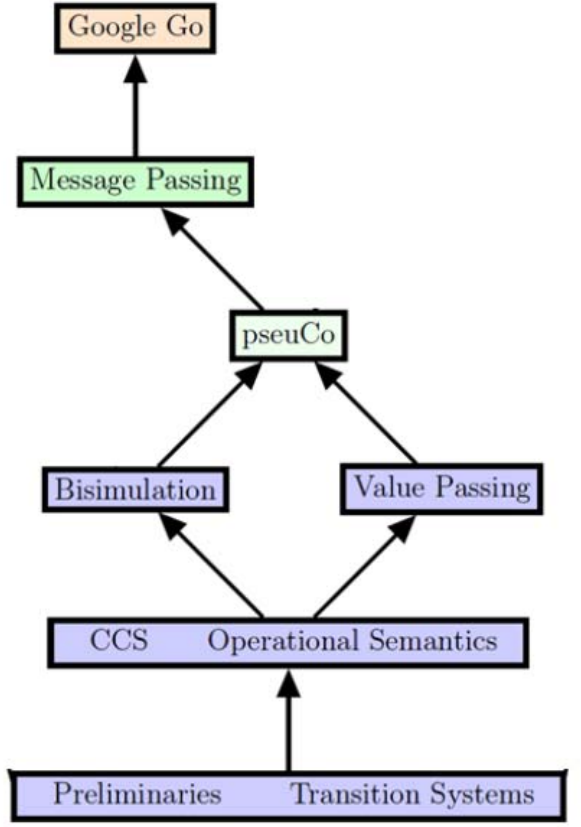
Thus channels both communicate and synchronize.

by communicating.

they


Go Playground
Run
Format
Impo
Go in a Browser
}
}
}
func main() {
var conn [5]chan int
for j := range conn {
conn[j] = make(chan int, 100)
}
go prime(7, conn[0], conn[1])
go prime(5, conn[1], conn[2])
go prime(3, conn[2], conn[3])
go prime(2, conn[3], conn[4])
for j := 2; j <= 100; j++ {
conn[0] <- j
}
for {
fmt.Println((<-conn[4]),
"is prime.")
}
2 is prime.
3 is prime.
5 is prime.
7 is prime.
11 is prime.
13 is prime.
17 is prime.
19 is prime.

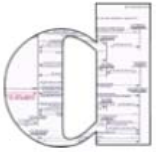
Where we stand



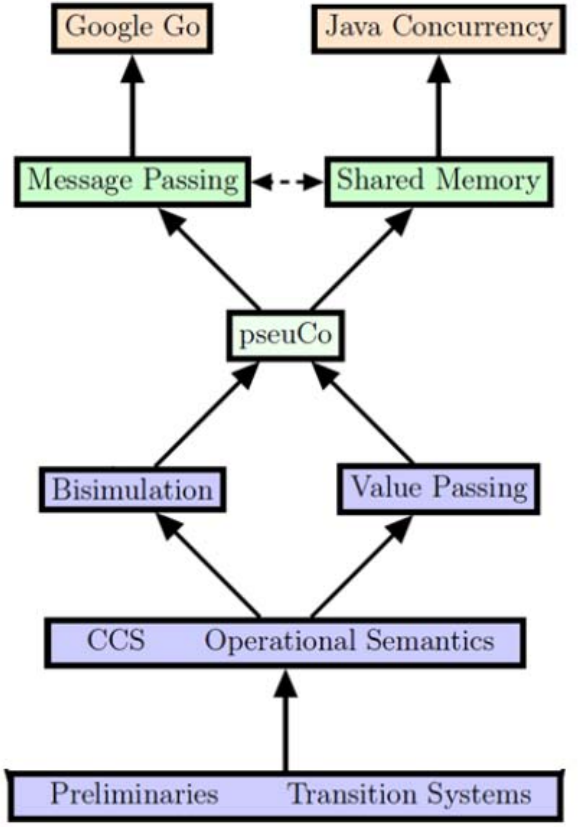
Practice

Concepts

Theory



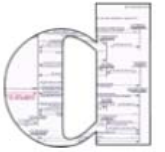
Where we go



Practice

Concepts

Theory



pseuCo Message Passing

- Typical imperative programming
  (no dynamic data structures)

- Agents - including `mainAgent`

- Channels
  - synchronous `intchan`
  - asynchronous `intchan3`

- Receive `<?`

- Send `<!`

- `select-case`

- Nothing else

No access to shared memory!

pseuCo Shared Memory

- Typical imperative programming
  (no dynamic data structures)

- Agents - including `mainAgent`

- Shared Variables

- Mutexes `lock` `unlock`

- Monitors `monitor`

- Conditions `condition ... with ...`
  `waitForCondition`

- Signals `signal` `signalAll`

- Nothing else

No channels, no send, no receive!

Semantics of pseuCo-SM?

given in terms of CCS

```
Lock:= lock?.unlock?.Lock;
```

```
Env_global_n[n] :=
   env_global_get_n!n.Env_global_n[n]
 + env_global_set_n?n.Env_global_n[n]
```

pseuCo Shared Memory

- Typical imperative programming
  (no dynamic data structures)
- Agents - including **mainAgent**
- Shared Variables
- Mutexes **lock** **unlock**
- Monitors **monitor**
- Conditions **condition ... with ...**
  waitForCondition
- Signals **signal** **signalAll**
- Nothing else

No channels, no send, no receive!

Semantics of pseuCo-SM?

given in terms of CCS

explained in terms of MP

```
void cell ( intchan rd , intchan wt , int init, int id) {
   int v = init ;
   while ( true ) {
      select {
         case rd <! v:
            println ("... Current Value:" + v + " in Cell " + id + ".");
         case v = <? wt:
            println ("... New Value:" + v + " in Cell " + id + ".");
      }
   }
}
```

```
void mpLock(boolchan1 g){
   g <! true;
}

void mpUnlock(boolchan1 g){
   <? g;
}
```

```
signalData <! true;
```

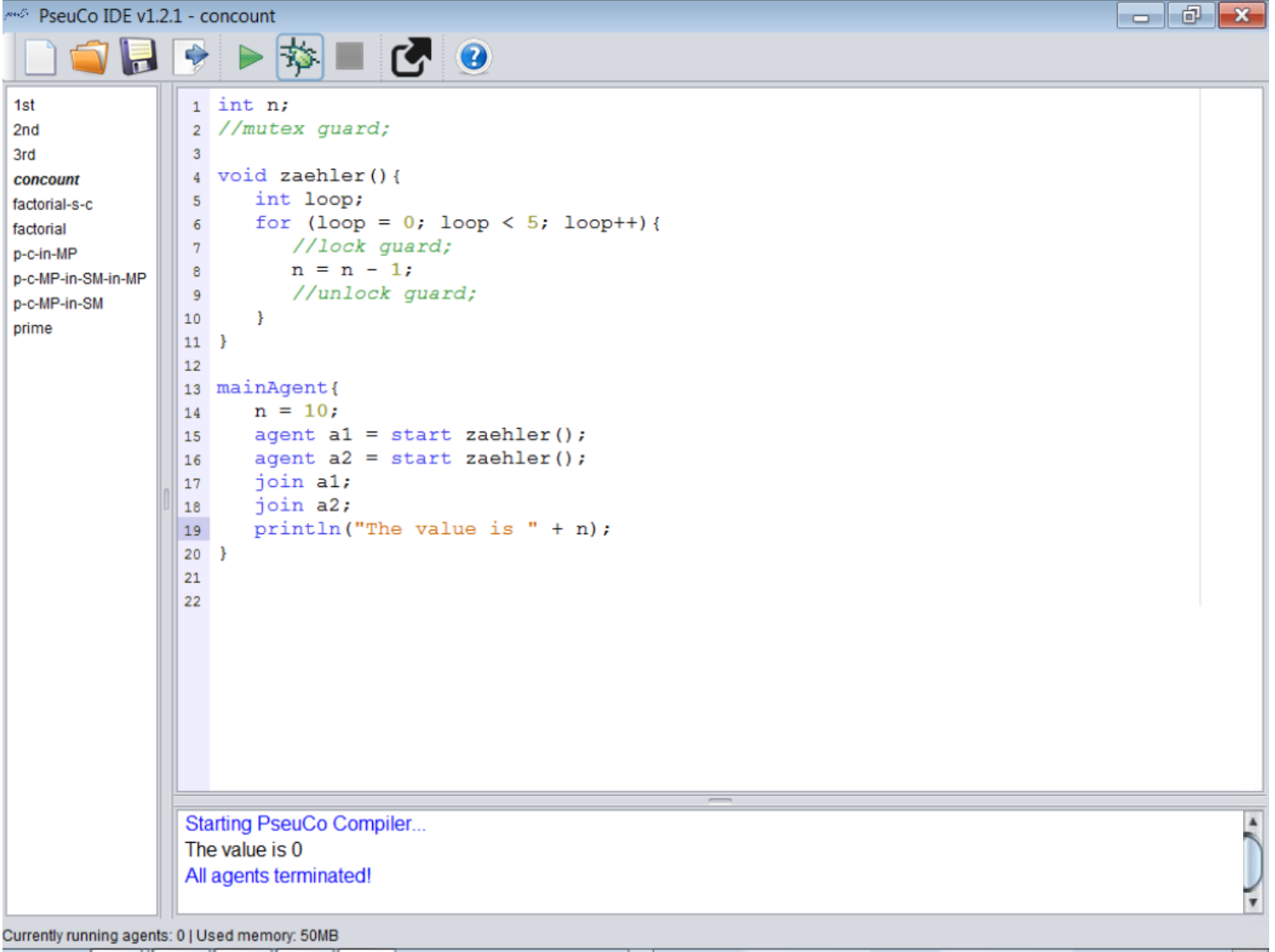
```
while (!(used < 3)){
   sleepPlatzFrei <! true;
   mpUnlock(guard);
   <? wakeupPlatzFrei;
   mpLock(guard);
}
```

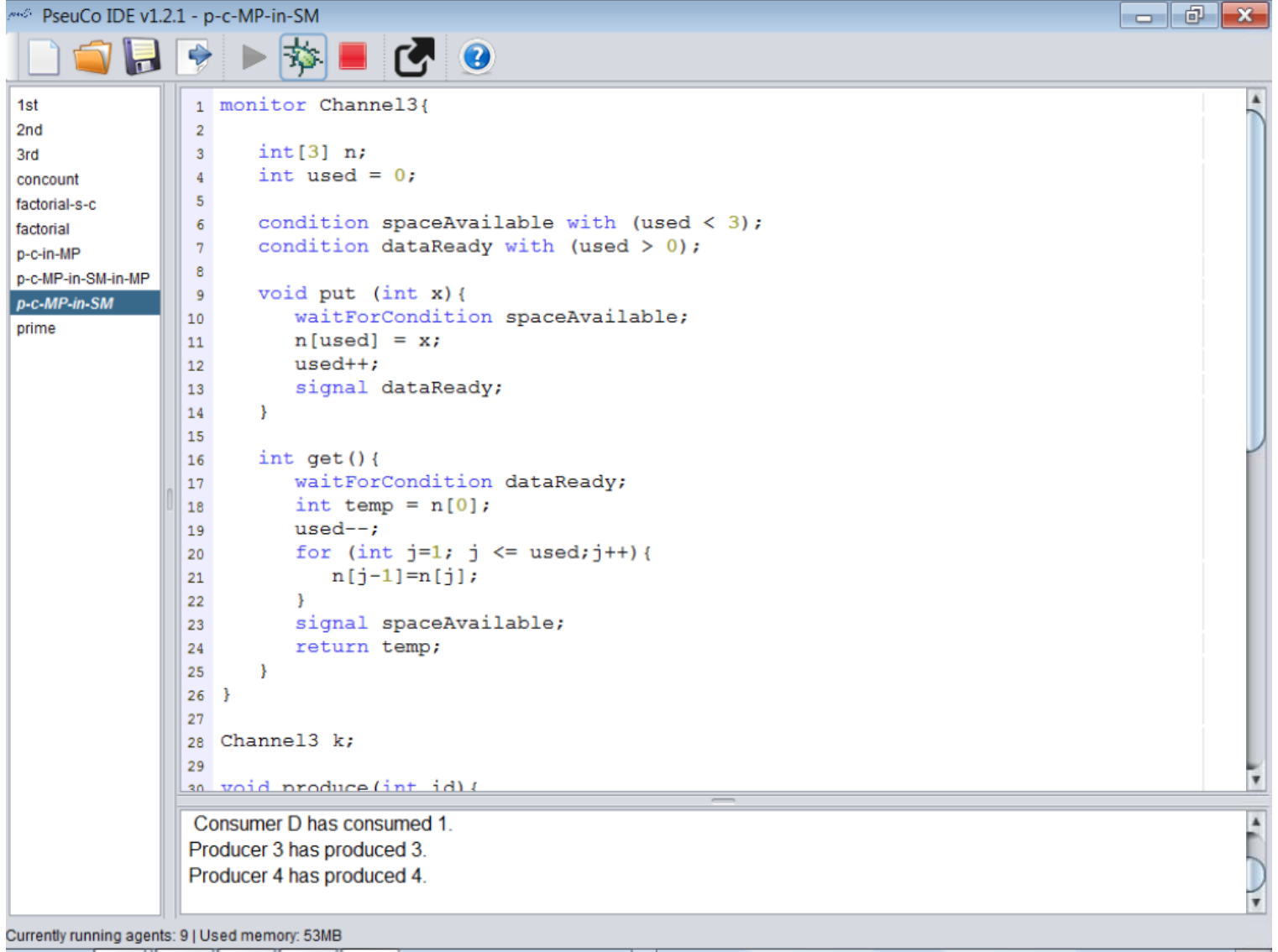
```
void conditionDaemon(boolchan sleep, boolchan wakeup, boolchan sig, string name){
   int count = 0;
   while (true) {
      select{
         case <? sleep:{
            count++;
            println("One more waiting for " + name + ".");
         }
         case <? sig: {
            if (count != 0){
               count--;
               wakeup <! true;
               println("One less waiting for " + name +".");
            }
         }
      }
   }
}
```

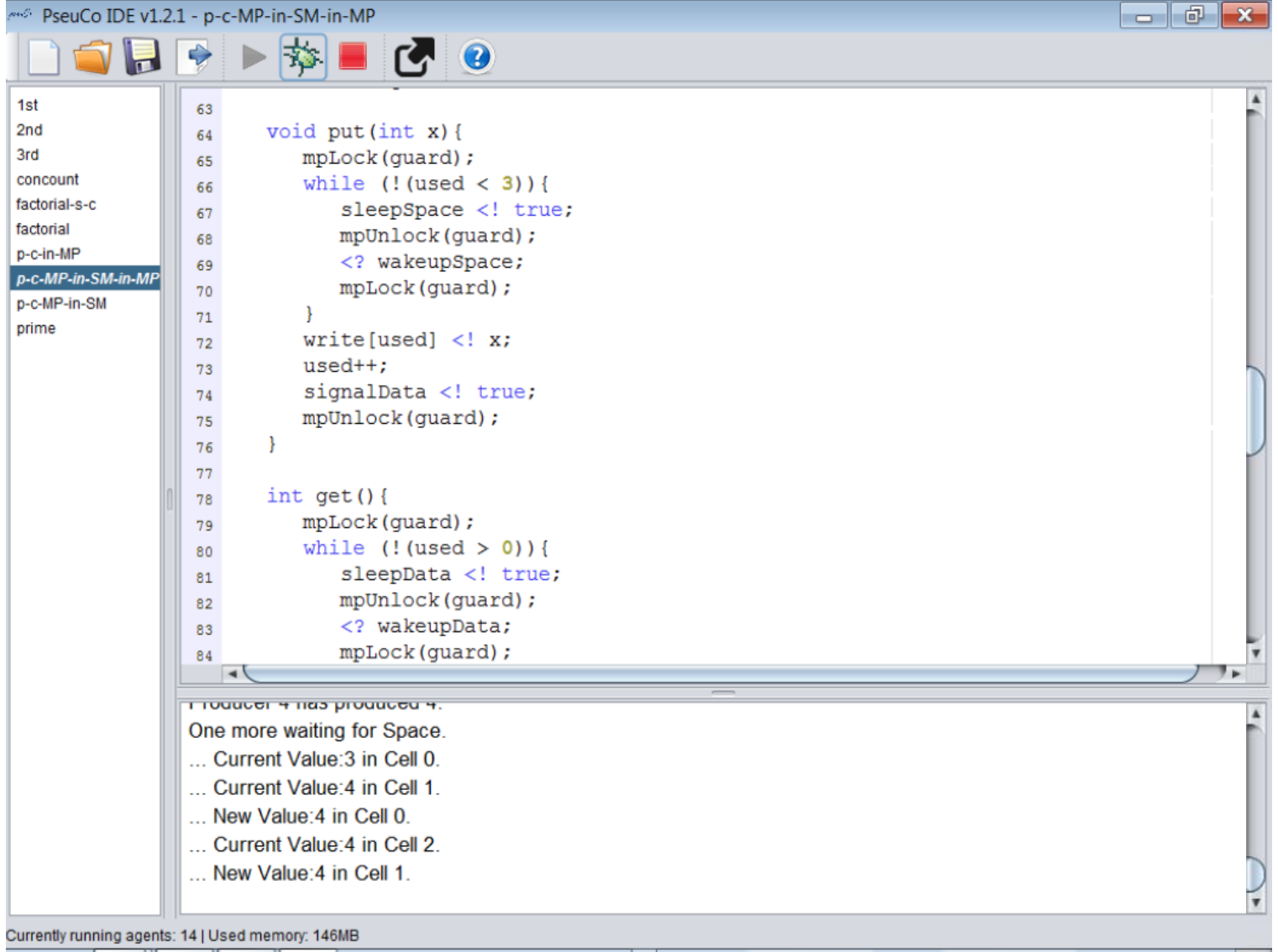
pseuCo Shared Memory

- Typical imperative programming
  (no dynamic data structures)
- Agents — including `mainAgent`
- Shared Variables
- Mutexes — `lock` `unlock`
- Monitors — `monitor`
- Conditions — `condition ... with ...` `waitForCondition`
- Signals — `signal` `signalAll`
- Nothing else

No channels, no send, no receive!


PseuCo IDE v1.2.1 - concount
1st
2nd
3rd
concount
factorial-s-c
factorial
p-c-in-MP
p-c-MP-in-SM-in-MP
p-c-MP-in-SM
prime
1 int n;
2 //mutex guard;
3
4 void zaehler(){
5 int loop;
6 for (loop = 0; loop < 5; loop++){
7 //lock guard;
8 n = n - 1;
9 //unlock guard;
10 }
11 }
12
13 mainAgent{
14 n = 10;
15 agent a1 = start zaehler();
16 agent a2 = start zaehler();
17 join a1;
18 join a2;
19 println("The value is " + n);
20 }
21
22
Starting PseuCo Compiler...
The value is 0
All agents terminated!
Currently running agents: 0 | Used memory: 50MB


PseuCo IDE v1.2.1 - p-c-MP-in-SM
1st
2nd
3rd
concount
factorial-s-c
factorial
p-c-in-MP
p-c-MP-in-SM-in-MP
p-c-MP-in-SM
prime
1 monitor Channel3{
2
3 int[3] n;
4 int used = 0;
5
6 condition spaceAvailable with (used < 3);
7 condition dataReady with (used > 0);
8
9 void put (int x){
10 waitForCondition spaceAvailable;
11 n[used] = x;
12 used++;
13 signal dataReady;
14 }
15
16 int get(){
17 waitForCondition dataReady;
18 int temp = n[0];
19 used--;
20 for (int j=1; j <= used;j++){
21 n[j-1]=n[j];
22 }
23 signal spaceAvailable;
24 return temp;
25 }
26 }
27
28 Channel3 k;
29
Consumer D has consumed 1.
Producer 3 has produced 3.
Producer 4 has produced 4.
Currently running agents: 9 | Used memory: 53MB

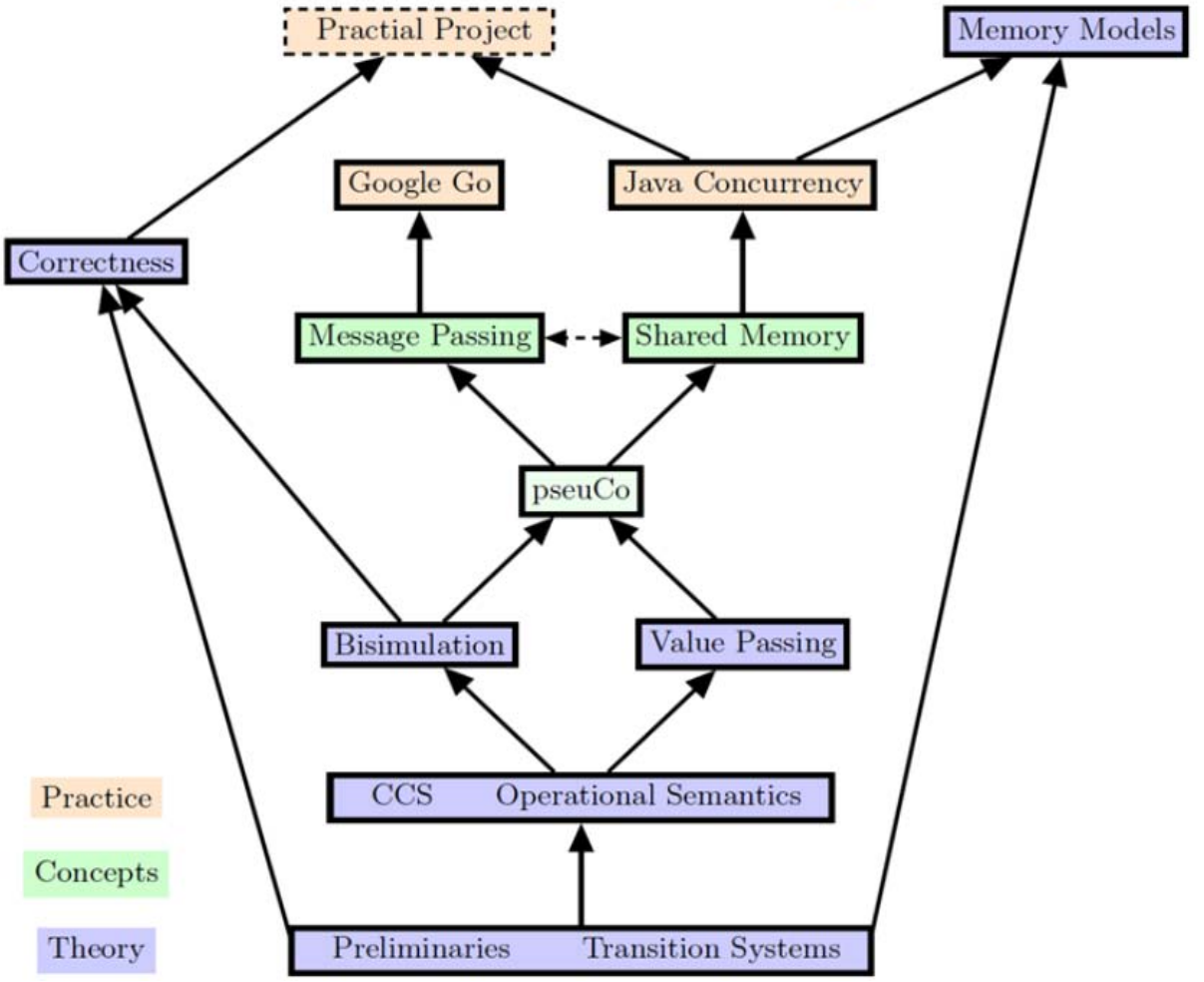

PseuCo IDE v1.2.1 - p-c-MP-in-SM-in-MP
1st
2nd
3rd
concount
factorial-s-c
factorial
p-c-in-MP
p-c-MP-in-SM-in-MP
p-c-MP-in-SM
prime
63
64 void put(int x){
65 mpLock(guard);
66 while (!(used < 3)){
67 sleepSpace <! true;
68 mpUnlock(guard);
69 <? wakeupSpace;
70 mpLock(guard);
71 }
72 write[used] <! x;
73 used++;
74 signalData <! true;
75 mpUnlock(guard);
76 }
77
78 int get(){
79 mpLock(guard);
80 while (!(used > 0)){
81 sleepData <! true;
82 mpUnlock(guard);
83 <? wakeupData;
84 mpLock(guard);
One more waiting for Space.
... Current Value:3 in Cell 0.
... Current Value:4 in Cell 1.
... New Value:4 in Cell 0.
... Current Value:4 in Cell 2.
... New Value:4 in Cell 1.
Currently running agents: 14 | Used memory: 146MB

We then turn to Java

- Focussing on the object-oriented aspects
  and their particularities inside Java.
- Is rather straightforward now.
- We discuss a spectrum of advanced concepts
  and data structures, including

`Reentrance` `ConcurrentModification`
`Client-Side Locking` `Deque` `Barriers`
`Latches` `Work Stealing` `Semaphores`
`CompareAndSwap` `CopyOnWriteArray`

The full monty



18 Lectures

1. Motivation, LTS
2. Nondeterminism, Concurrency
3. Sequential CCS
4. Full CCS
5. Equal Behaviour
6. Bisimulation
7. Observational Congruence
8. Value Passing CCS

"Theory"

9. pseuCo
10. Message Passing and Go
11. Shared Memory, Data Races
12. Locks and Monitors
13. Conditions, MP vs. SM
14. Java Threads
15. Java Helpers etc
16. Safety, Liveness, Fairness
17. Memory Model
18. Epilogue, Project Intro

"Practice"

Summary

pseuCo

- A simple pseudocode
  - enriched with message-passing primitives
  - enriched with shared-memory primitives
- Trivial translation

  from pseuCo-MP to Go
- Executable semantics
  - to Java
  - to JVM-Bytecode with direct execution

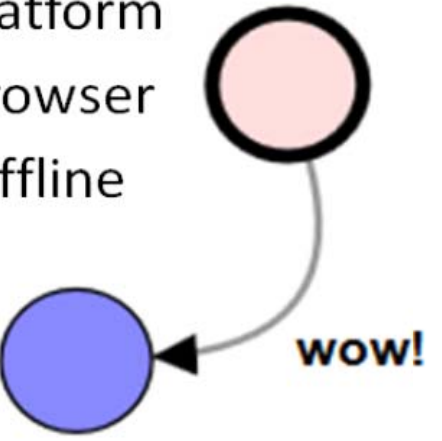
pseuCo.com

- formal semantics
  - ☞ CCS
  - ☞ LTS

explorative

- Zero installation
- Cross-platform
- Cross-browser
- Works offline

everything under GPL



compiler from pseuCo to CCS, and can visualize the Labeled Transitions System described by a CCS term.

To get more information about this application, its developers and license (and to switch to the beta version), open the About tab.

Get involved

Help us to make this application better! Get the code, see the development status, browse the list of known problems and feature requests, and send us new bugs and suggestions – you can do all of that on our development site.

Go there

http://www.doctormacro.com/Images/Leigh,%20Janet/Annex/Annex%20-%20Leigh,%20Janet%20%28Psycho%29_01.jpg