



---

# Concurrent Object-Oriented Programming

Bertrand Meyer, Piotr Nienaltowski



# Lecture 11:

## More about synchronization in SCOOP



# Outline

---

3

- Refresher on type system
- Handling false traitors
- Lock passing and how to use it



# Refresher: type system

- Current processor  $x: X$

Some processor (top)  
 $x: \text{separate } X$

- Processor tag  $\alpha \in \{\bullet, \top, \perp, \langle p \rangle, \langle a.\text{handler} \rangle\}$

- Attached/detachable  $\gamma \in \{!,$

No processor (bottom)  
**Void**

$$\Gamma \vdash x :: (\gamma, \alpha, C)$$



# Examples

5

$x: X$                        $-- x :: (!, \bullet, X)$

$y: \mathbf{separate} Y$                        $-- y :: (!, \top, Y)$

$z: ? \mathbf{separate} Z$                        $-- z :: (?, \top, Z)$

- Expanded types are attached and non-separate

$i: \mathbf{INTEGER}$                        $-- i :: (!, \bullet, \mathbf{INTEGER})$

- **Void** is detachable                       $-- \mathbf{Void} :: (?, \perp, \mathbf{NONE})$

- **Current** is attached and non-separate

$-- \mathbf{Current} :: (!, \bullet, C_{\mathbf{Current}})$

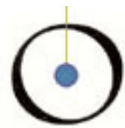


# Subtyping rules

- Since you don't like Greek letters, I'll keep it informal
- $TT_2 \leq TT_1$  means "TT<sub>2</sub> is a subtype of TT<sub>1</sub>"
- Conformance on class types like in Eiffel, essentially based on inheritance

$$D \leq_{\text{Eiffel}} C \iff (\gamma, \alpha, D) \leq (\gamma, \alpha, C)$$

- Attached  $\leq$  detachable  $(!, \alpha, C) \leq (?, \alpha, C)$
- Any processor tag  $\leq T$   $(\gamma, \alpha, C) \leq (\gamma, T, C)$
- In particular, non-separate  $\leq T$   $(\gamma, \bullet, C) \leq (\gamma, T, C)$
- $\perp \leq$  any processor tag  $(\gamma, \perp, C) \leq (\gamma, \alpha, C)$



# Using the type system

- We can rely on standard type rules
- Enriched types give us additional guarantees
- Assignment rule: source conforms to target

$$\text{[Assign]} \frac{x :: TT_x, \quad e :: TT_e, \quad TT_e \leq TT_x}{x := e}$$

- No need for special validity rules for **separate**



# Type combinators

$x: T_x$

$f (fa: T_{fa}): T_{res}$

- Actual argument

$a :: T_x \otimes T_{fa}$

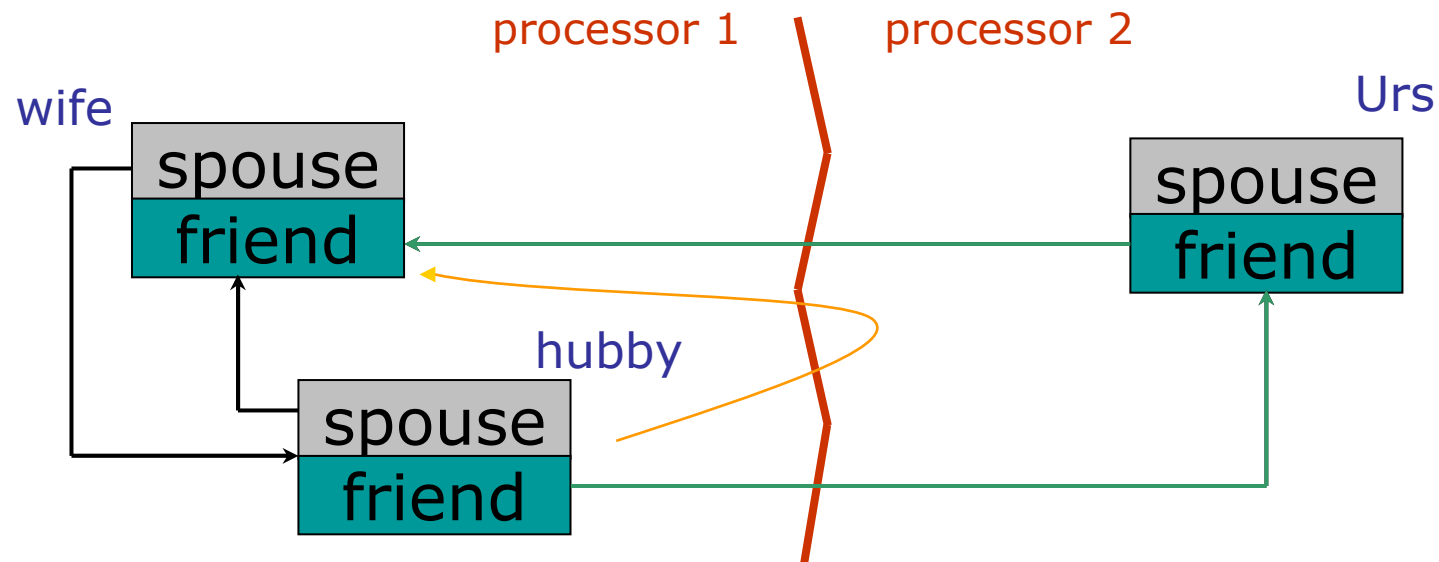
- Result type

$x.f (a) :: T_x * T_{res}$



# False traitors

9



*meet\_friend* (*person*: **separate** *PERSON*) **is**

**local**

*a\_friend*: *PERSON*

**do**

*a\_friend* := *person.friend* -- Invalid assignment.

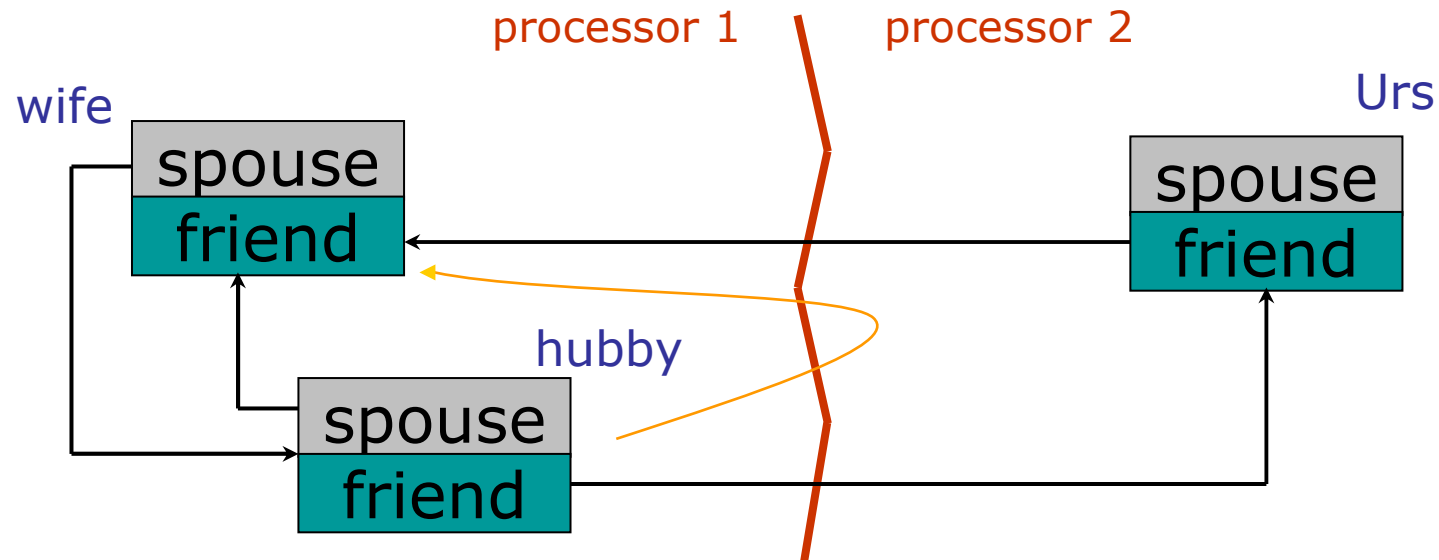
*visit\_locally* (*a\_friend*)

**end**



# Handling false traitors

10



*meet\_friend* (*person*: **separate** *PERSON*) **is**

**local**

*a\_friend*: *PERSON*

**do**

*a\_friend* **?=** *person.friend* -- Valid assignment attempt.

**if** *a\_friend* **/= void** **then** *visit\_locally* (*a\_friend*) **end**

**end**



# Assignment attempt

11

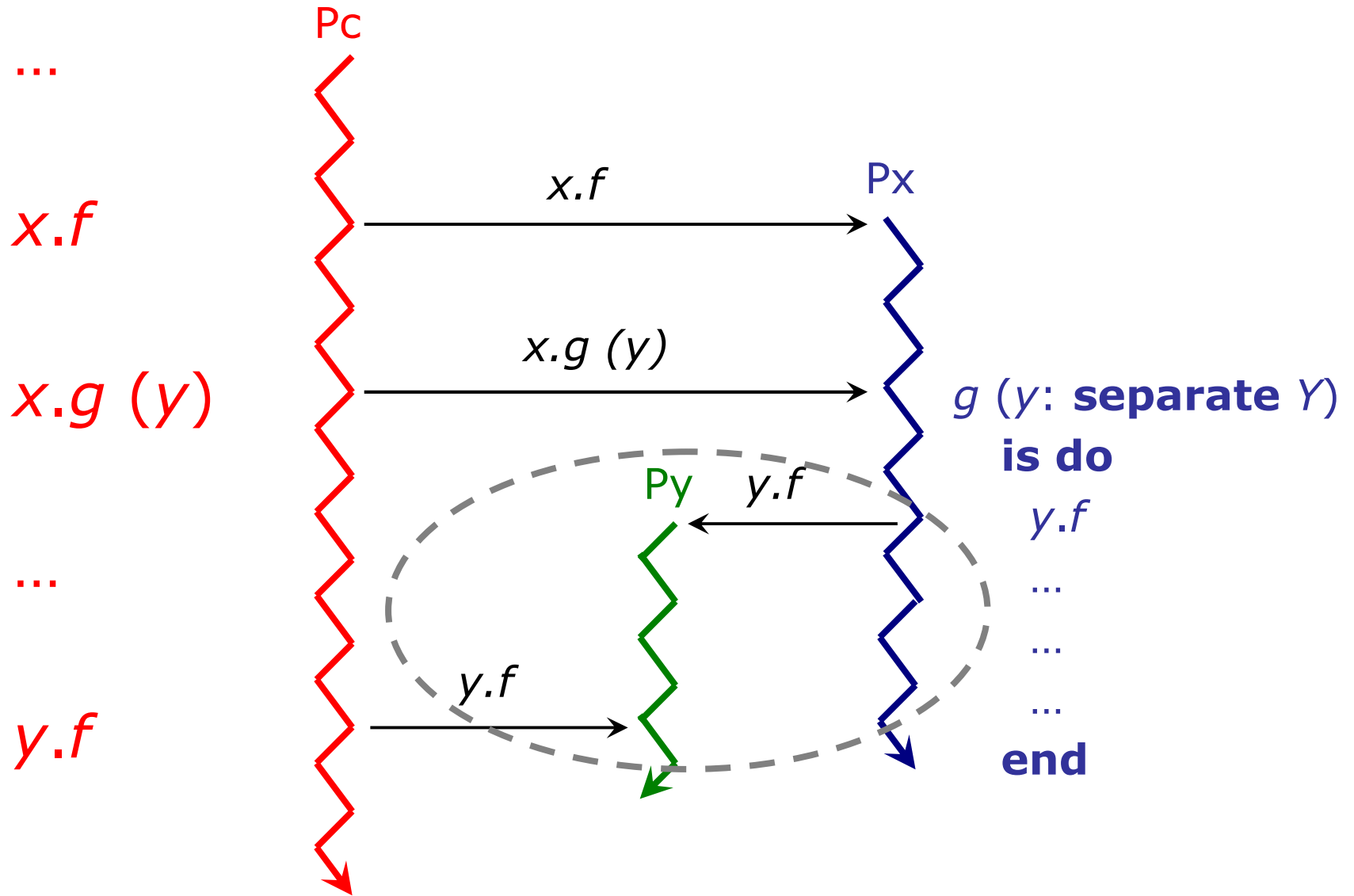
- Like in Eiffel but also “downcasts” processor tags
  - “deep downcast” over expanded attributes
- Eiffel standard has replaced `?=` with **object test**

```
if {a_friend: PERSON} person.friend then  
    visit_locally (a_friend)  
end
```

- Use assignment attempt with `scoop2scoopli`
  - **Attention:** conversion does not work with assignment attempts



# A synchronization problem in SCOOP





# Solution (original SCOOP)

13

- Make  $x$  wait until  $y$  becomes available
- May be **dangerous**. Why?

$r$  ( $x$ : **separate**  $X$ ;  $y$ : **separate**  $Y$ ) **is**

**do**

$x.f$

$x.g$  ( $y$ )

$y.f$

...

$value := x.some\_query$

**end**

- “Business Card principle” for dealing with tricky cases
- Not flexible



# Solution (current SCOOP)

- Lock passing occurs when client passes locked actual argument to separate call that wants to lock that argument.

*r* (*x*: **separate** *X*; *y*: **separate** *Y*) **is**  
**do**

*x.f*

*x.g* (*y*) -- *x* gets access to *y*.

-- Current waits until call has terminated.

*y.f*

...

*value* := *x.some\_query* -- No deadlock here.

**end**

Both calls are synchronous!

- *x* gets exclusive access on *y* immediately
- Client has to wait until separate call terminated



# Solution (current SCOOP)

15

- In fact, client has to pass **all its locks**
- Necessary conditions for lock passing
  - **Current** has lock on supplier  $x$  and argument  $y$ .
  - $x$  is handled by a different processor than  $y$  and **Current**;  $y$  and **Current** may be on the same processor.
  - **Current** calls feature  $f$  on  $x$  with argument  $y$ ; formal argument of  $f$  corresponding to  $y$  is separate and attached.
- If feature  $f$  takes detachable formal argument, no lock passing occurs.
- Original SCOOP semantics can be simulated using object test.



# Why do we need lock passing?

16

- Avoiding deadlocks

*s* (*x*: **separate** *X*) **is**

**do**

*value* := *x.f* (**Current**) -- Deadlocks if no lock passing.

**end**

- Expressiveness and precision: do what the programmer wants.
- Novel approach to reasoning about asynchronous calls
  - Hoare-style rules
  - separate objects locked by client are treated as non-separate



## In practice

---

17

- Lock passing fully implemented
- Detachable and attached types are not yet supported by Eiffel compilers
- Scoop2scoopli does not accept '?' annotation
- Lock passing applied by default



# Demo

---

18