

Behavior Protocols Capturing Errors and Updates

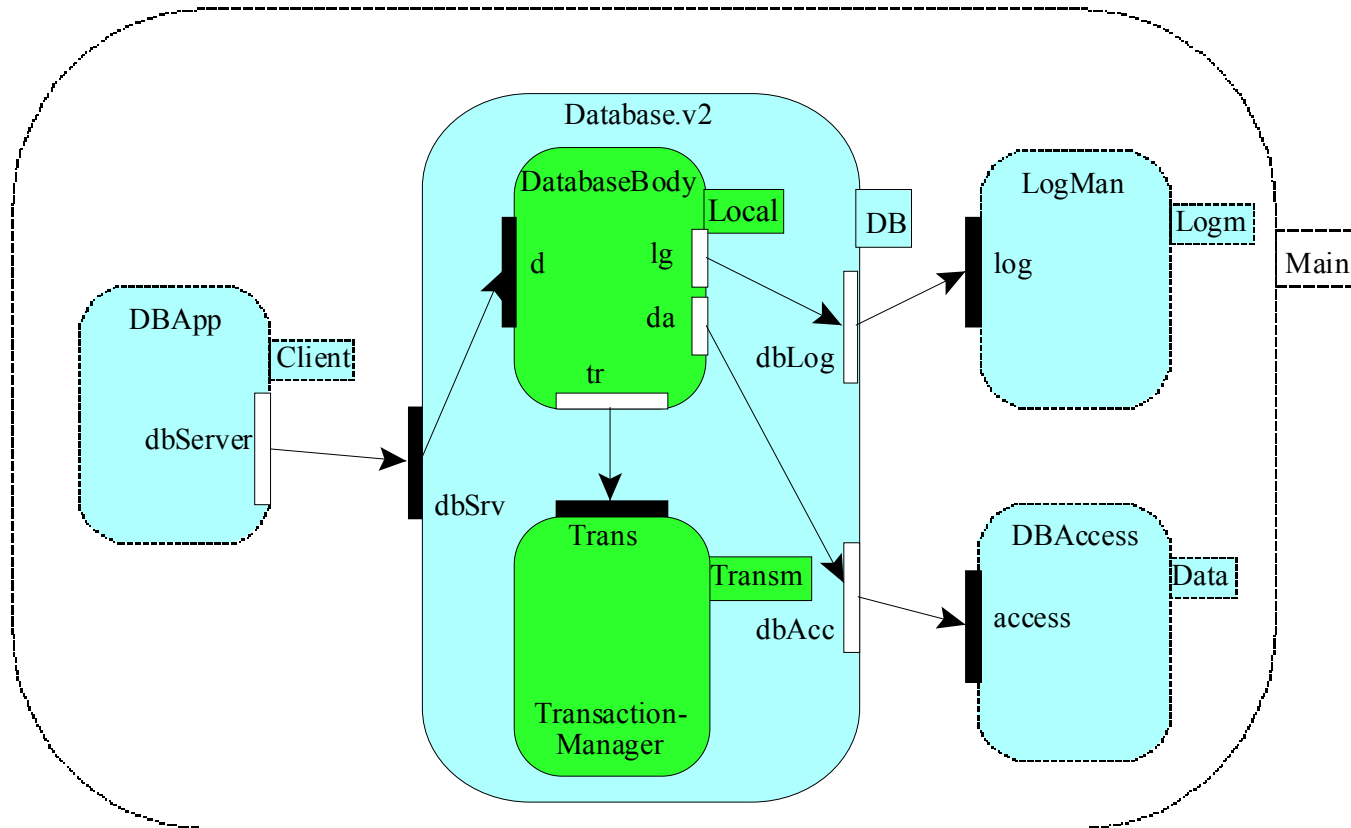
Jiří Adámek, František Plášil

Distributed systems research group

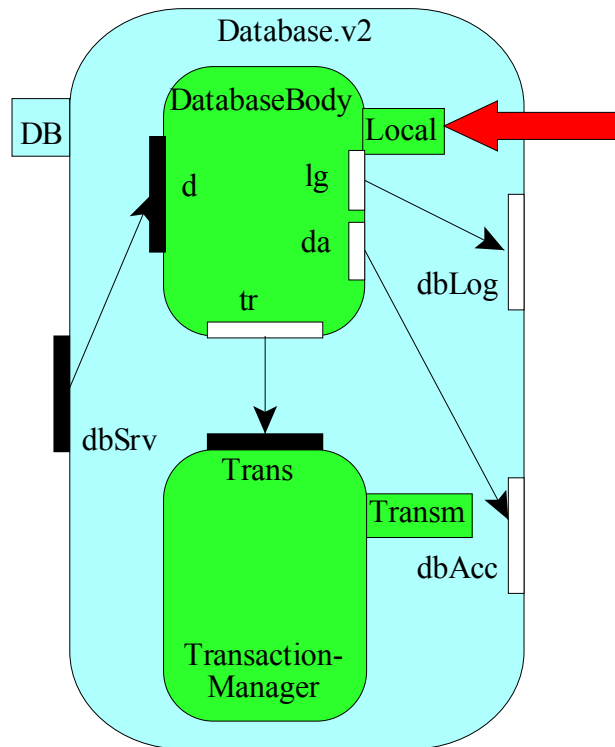
Charles University Prague

<http://nenya.ms.mff.cuni.cz>

SOFA project



SOFA behavior protocols

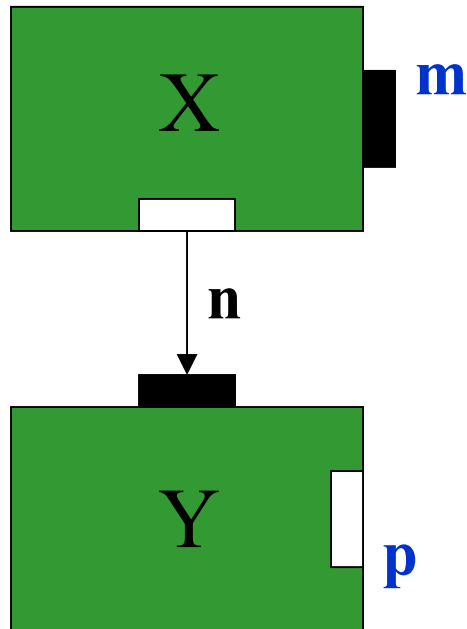


frame protocol:

```
!da.Open ;
( ?d.Insert { !tr.Begin ; !da.Insert ;
  !lg.LogEvent; (!tr.Commit +
  !tr.Abort) }
+
?d.Delete { !tr.Begin ; !da.Delete ;
  !lg.LogEvent;
  (!tr.Commit + !tr.Abort) }
+
?d.Query { !da.Query }
)* ;
!da.Close
```

Plasil,F. Visnovsky: Behavior protocols for SW components.
IEEE Trans. on SE, Nov. 2002

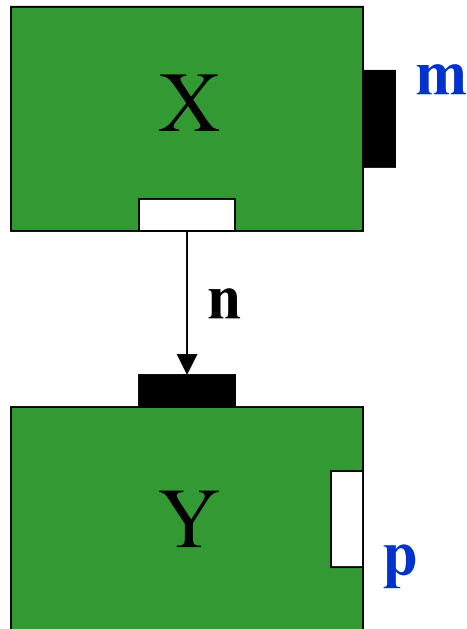
Behavior protocols



$$\text{Prot}_X = (?m\uparrow ; ((!n\uparrow ; ?n\downarrow) + \text{NULL}) ; !m\downarrow)^*$$

$$\text{Prot}_Y = (?n\uparrow ; !p\uparrow ; ?p\downarrow ; !n\downarrow)^*$$

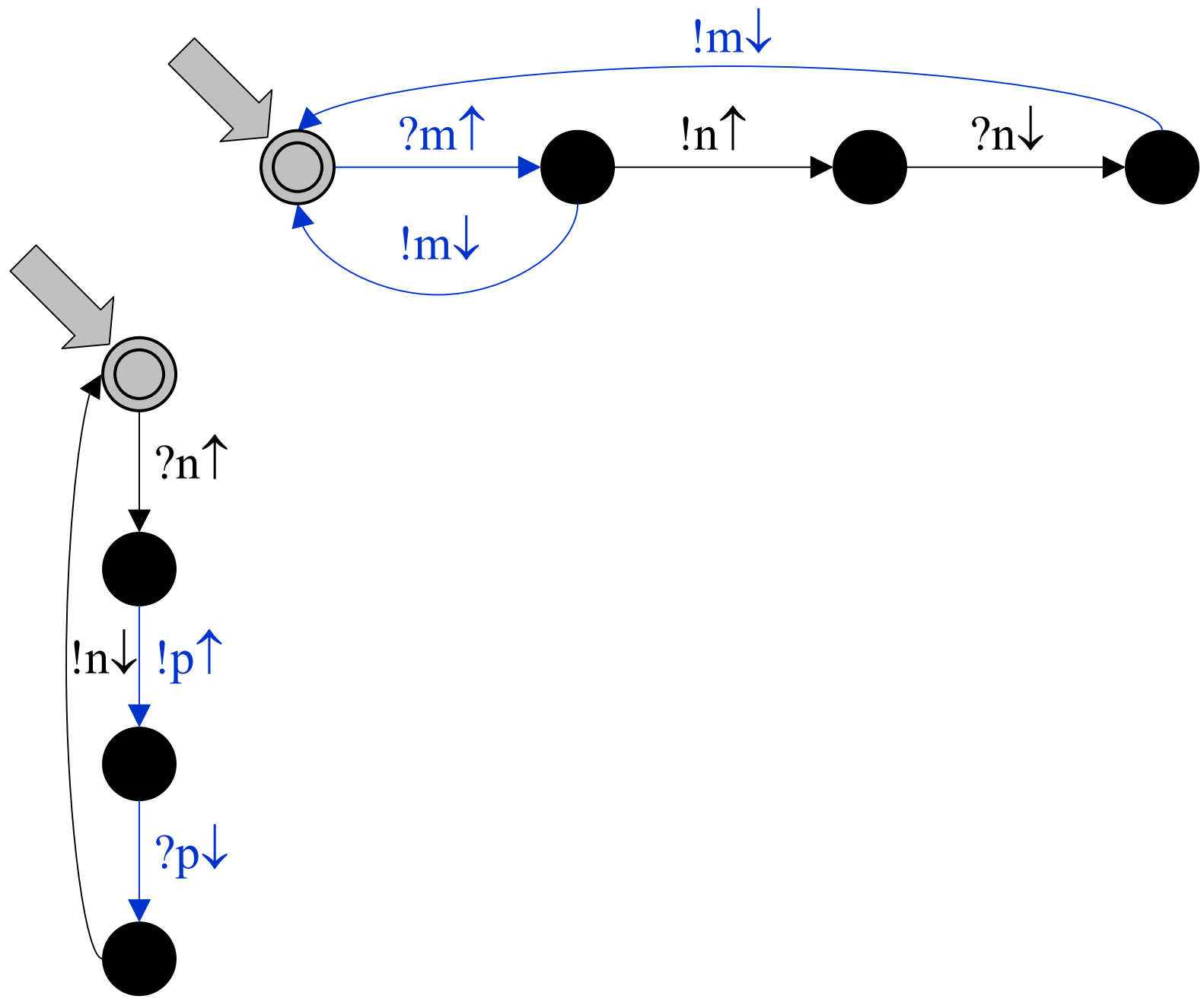
Behavior protocols

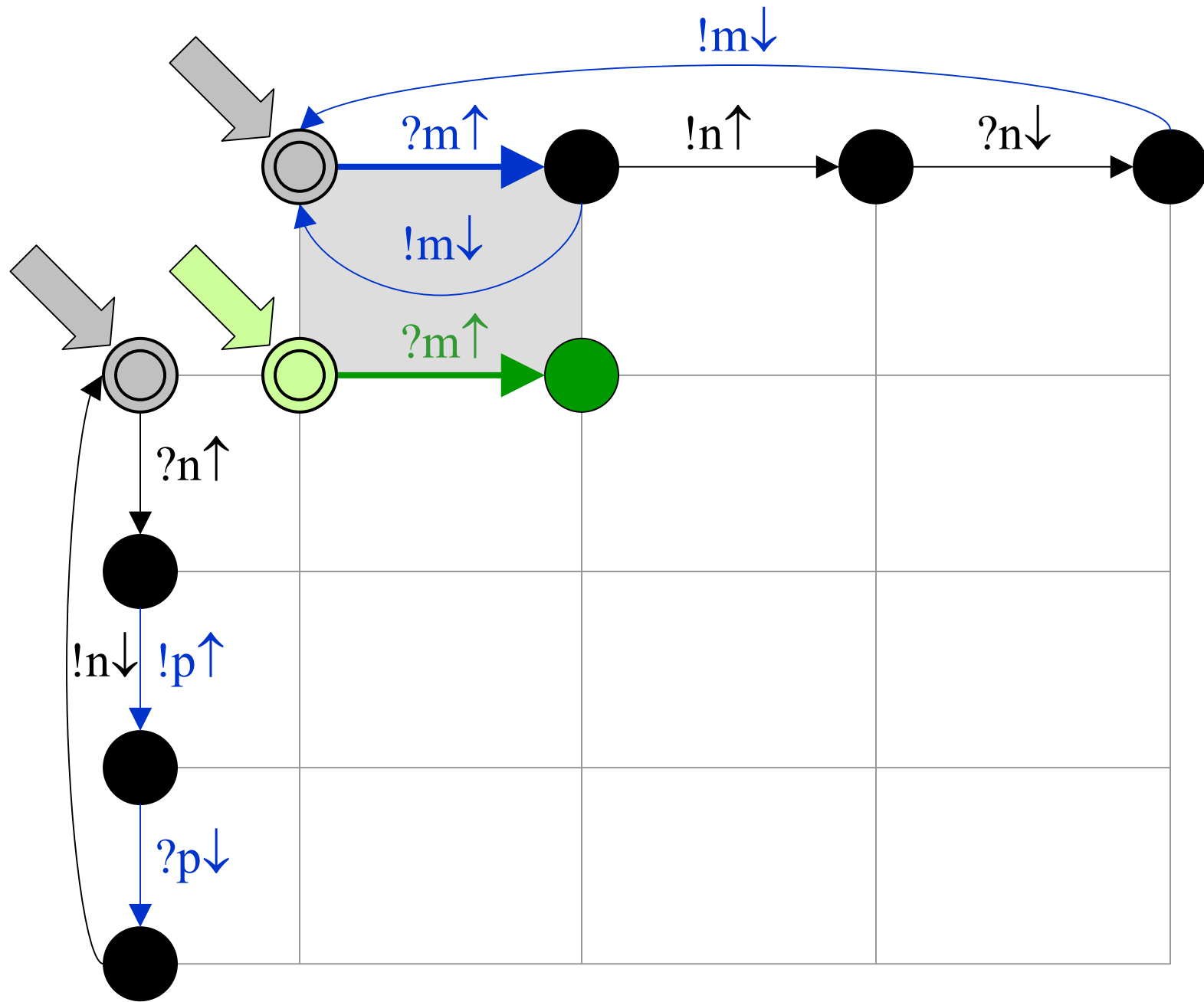


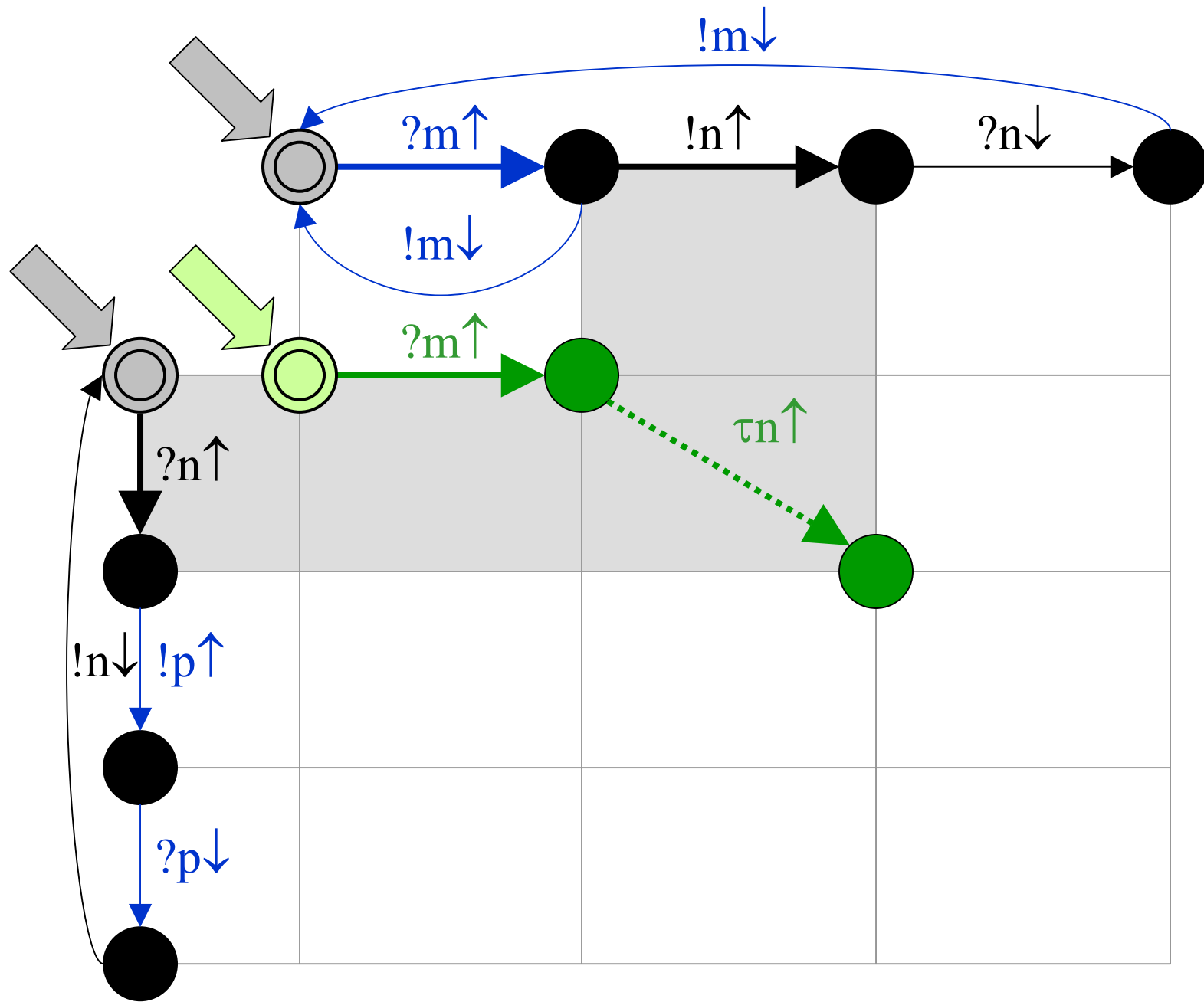
$$\text{Prot}_X = (?m\uparrow ; ((!n\uparrow ; ?n\downarrow) + \text{NULL}) ; !m\downarrow)^*$$

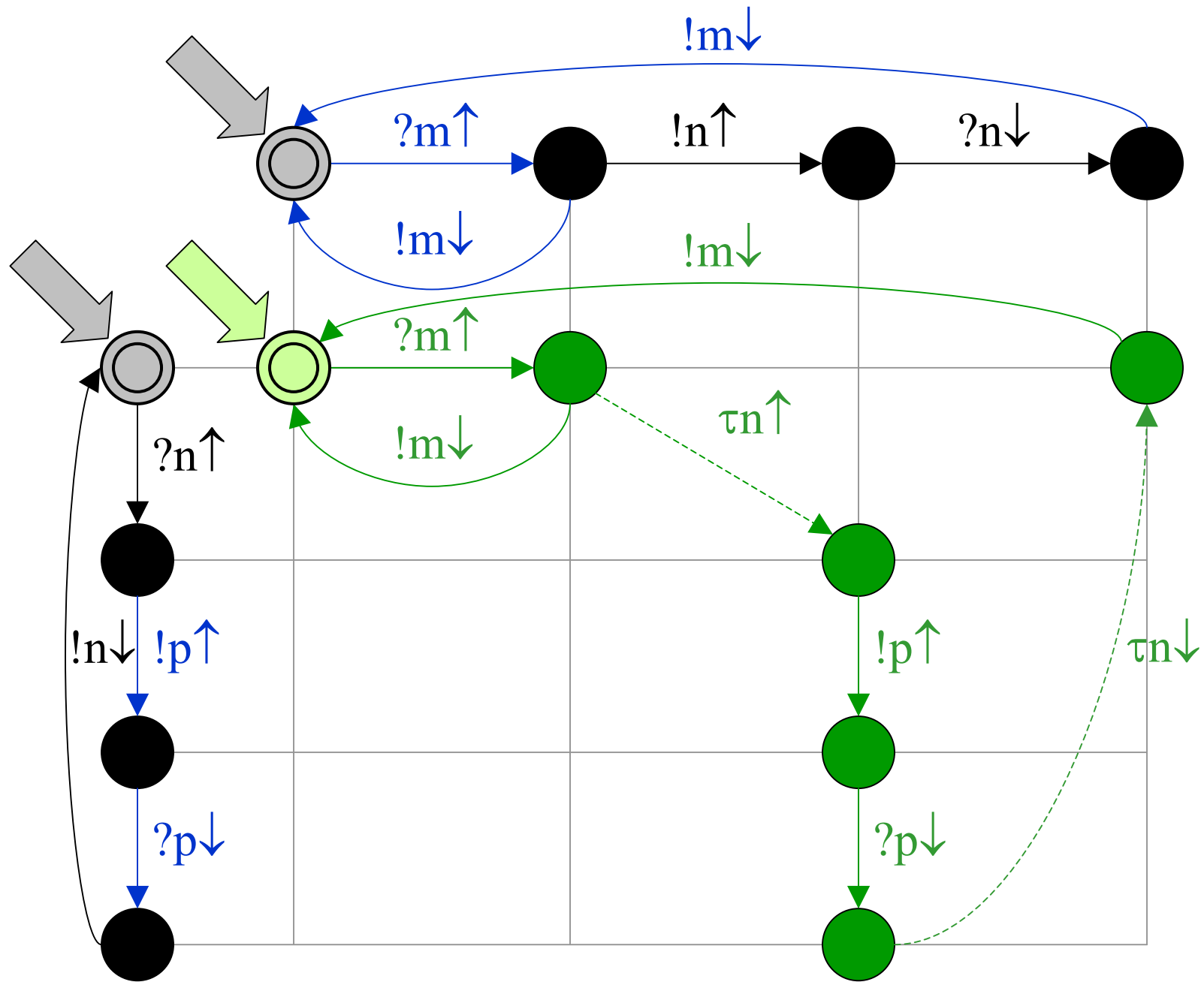
$$\text{Prot}_Y = (?n\uparrow ; !p\uparrow ; ?p\downarrow ; !n\downarrow)^*$$

$$\text{Prot}_X \nabla_{\{n\uparrow, n\downarrow\}} \text{Prot}_Y = \{ \langle ?m\uparrow ; \tau n\uparrow ; !p\uparrow ; ?p\downarrow ; \tau n\downarrow ; !m\downarrow \rangle, \langle ?m\uparrow ; !m\downarrow \rangle, \dots \}$$





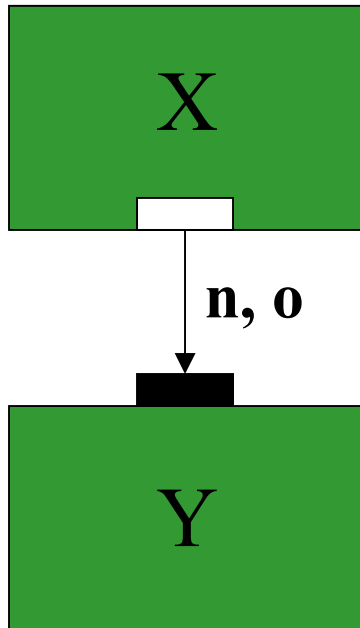




Composition errors

- Bad activity - $\epsilon n \uparrow$, $\epsilon n \downarrow$
- No activity (deadlock) - $\epsilon \emptyset$
- Infinite activity (divergence) - $\epsilon \infty$

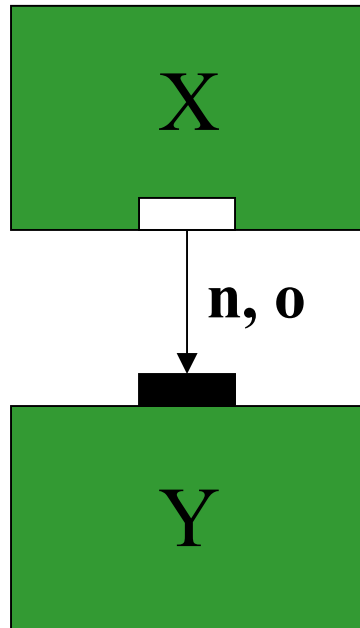
Bad activity



$$\text{Prot}_X = !n\uparrow ; ?n\downarrow ; ((!o\uparrow ; ?o\downarrow) + (!n\uparrow ; ?n\downarrow))$$

$$\text{Prot}_Y = ?n\uparrow ; !n\downarrow ; ?o\uparrow ; !o\downarrow$$

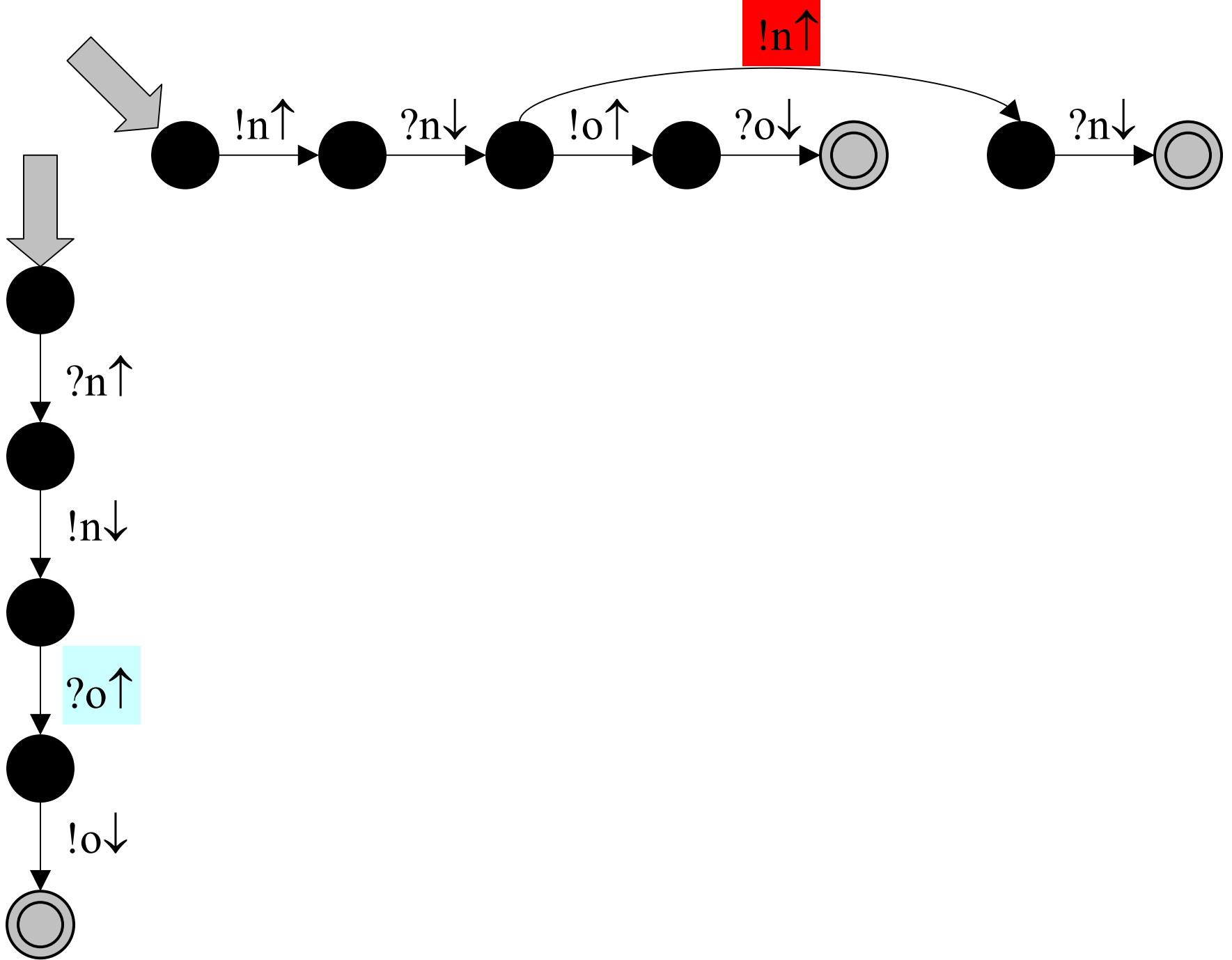
Bad activity

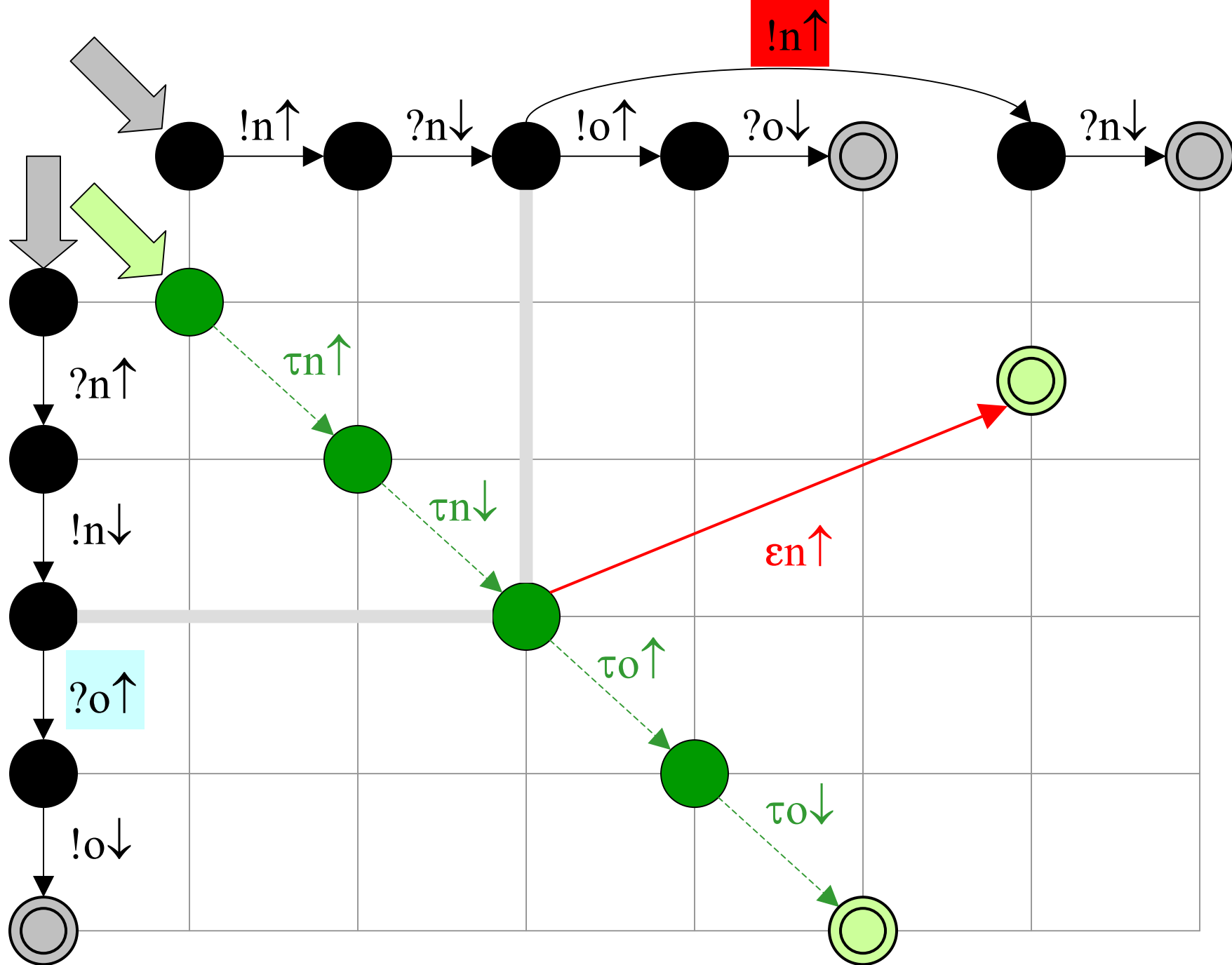


$$\text{Prot}_X = !n\uparrow ; ?n\downarrow ; ((!o\uparrow ; ?o\downarrow) + (!n\uparrow ; ?n\downarrow))$$

$$\text{Prot}_Y = ?n\uparrow ; !n\downarrow ; ?o\uparrow ; !o\downarrow$$

$$\text{Prot}_X \nabla_{\{n\uparrow, n\downarrow, !o\uparrow, o\downarrow\}} \text{Prot}_Y = \{ \langle \tau n\uparrow ; \tau n\downarrow ; \tau o\uparrow ; \tau o\downarrow \rangle, \langle \tau n\uparrow ; \tau n\downarrow ; \epsilon n\uparrow \rangle \}$$

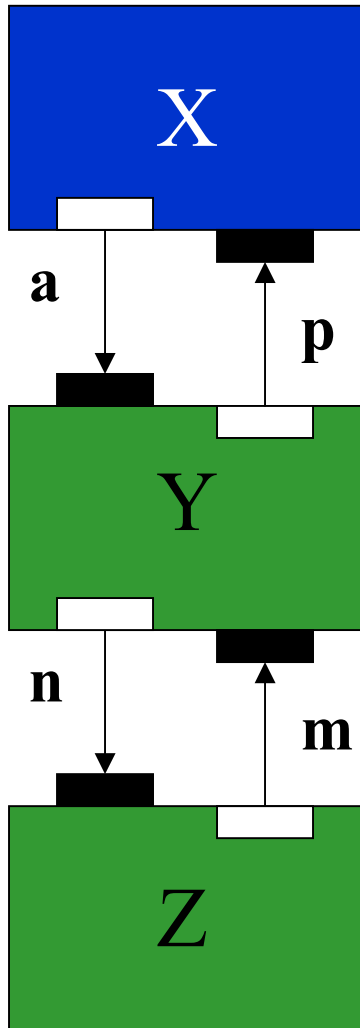




Consent operator ∇

- Constructs behavior protocols
 - of composed components
- Captures
 - composition errors
- Checks
 - atomicity of dynamic updates

Dynamic update (1)

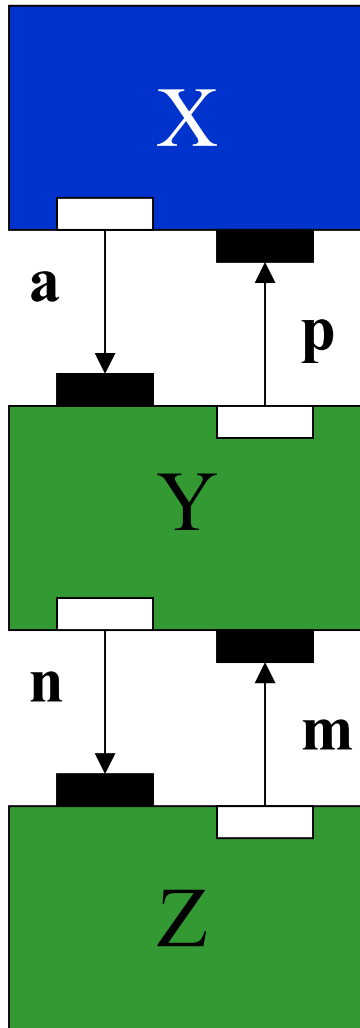


$$\text{Prot}_X = ((!a ; ?p)^* ; ?\pi_1 \uparrow ; !\pi_1 \downarrow)^*$$

$$\text{Prot}_Y = (?a ; !p)^* \mid (?m ; !n)^*$$

$$\text{Prot}_Z = (!m ; ?n)^*$$

Dynamic update (1)



$$\mathbf{Prot}_X = ((!a ; ?p)^* ; ?\pi_1 \uparrow ; !\pi_1 \downarrow)^*$$

$$\mathbf{Prot}_Y = (?a ; !p)^* \mid (?m ; !n)^*$$

$$\mathbf{Prot}_Z = (!m ; ?n)^*$$

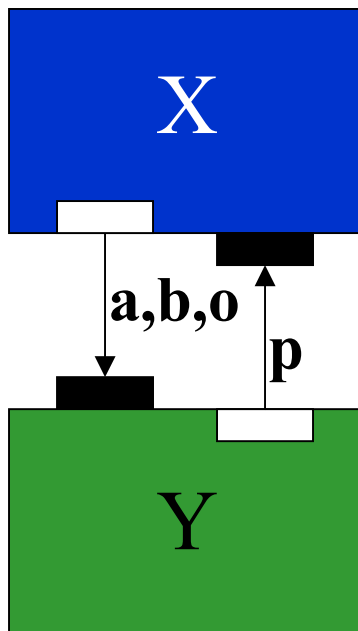
$$\Downarrow (\mathbf{Prot}_X \nabla_{\{a \uparrow, a \downarrow, p \uparrow, p \downarrow\}} \mathbf{Prot}_Y) \nabla_{\{n \uparrow, n \downarrow, m \uparrow, m \downarrow\}} \mathbf{Prot}_Z$$

$$((\tau a ; \tau p)^* ; \tau \pi_1 \uparrow ; \tau \pi_1 \downarrow)^* \mid (\tau m ; \tau n)^*$$

Dynamic update (2)

$$\text{Prot}_X = ((!a ; ?\pi_1\uparrow ; !\pi_1\downarrow) + (!b ; ?\pi_2\uparrow ; !\pi_2\downarrow)) ; (!o | ?p)$$

$$\text{Prot}_Y = (?a ; ?o ; !p) + (?b ; (?o | !p))$$

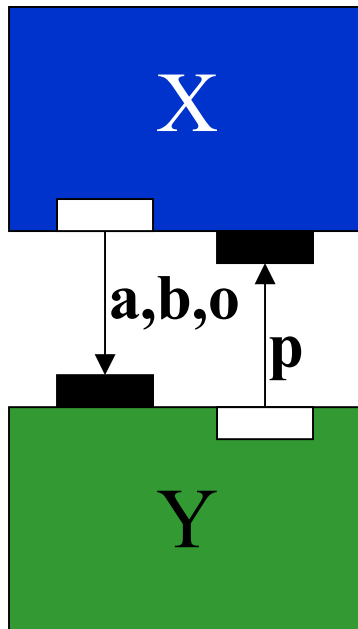


Dynamic update (2)

$$\text{Prot}_X = ((!a ; ?\pi_1\uparrow ; !\pi_1\downarrow) + (!b ; ?\pi_2\uparrow ; !\pi_2\downarrow)) ; (!o | ?p)$$

$$\text{Prot}_Y = (?a ; ?o ; !p) + (?b ; (?o | !p))$$

↓ $\text{Prot}_X \nabla_{\{\dots\}} \text{Prot}_Y$



$$\begin{aligned} & (\tau a ; \tau \pi_1\uparrow ; \tau \pi_1\downarrow ; \tau o ; \tau p) + \\ & (\tau b ; (\\ & \quad (\tau \pi_2\uparrow ; \tau \pi_2\downarrow ; (\tau o | \tau p)) + (\tau \pi_2\uparrow ; \epsilon p\uparrow) \\ & \quad) \\ &) \end{aligned}$$

Conclusion

- Atomicity of component updates
 - tested statically
 - violation indicated by ∇
- No locking
 - ~performance benefit
 - deadlock avoidance
- Key benefit: Info on updates
 - in protocol
 - outside of code

Discussion

- Simplifies code
 - Coordination of locking on multiple interfaces skipped
- Dynamic checker used anyway
 - Let it check whether update comes at the right time
- Nested components
 - Compliance def. modification (technical, not principle)
 - Strictly synchronized with children's willingness to be updated
 - Wait until children finish internal actions (~liveness property)
- Internal threads
 - Assumption: Implementation of X respects updating plan in Prot_X