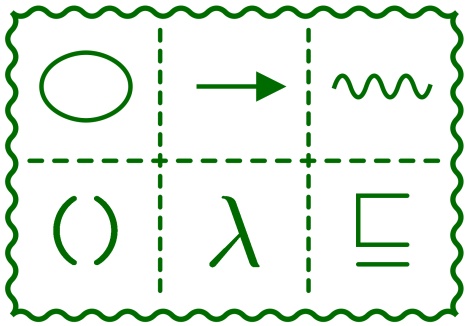
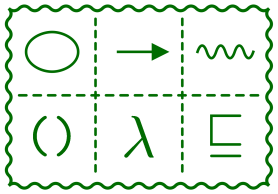


並行システムの検証と実装

第4章 プロセスの逐次合成

PRINCIPIA Limited
初谷 久史



プロセスの逐次合成

- 逐次合成 seq
- 内部イベント tau
- プロセスの再利用
- プロセスの合流
- メモプロセス

逐次合成 seq

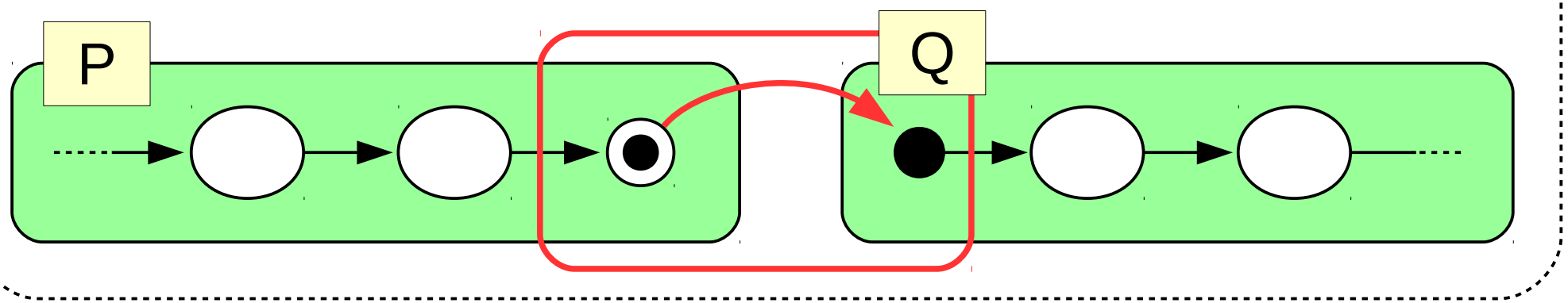
書式

(seq プロセス式$_0$ プロセス式$_1$ ...)

プロセスを指定された順に実行する

(seq P Q)

プロセス P が正常に終了したら
プロセス Q に移行する

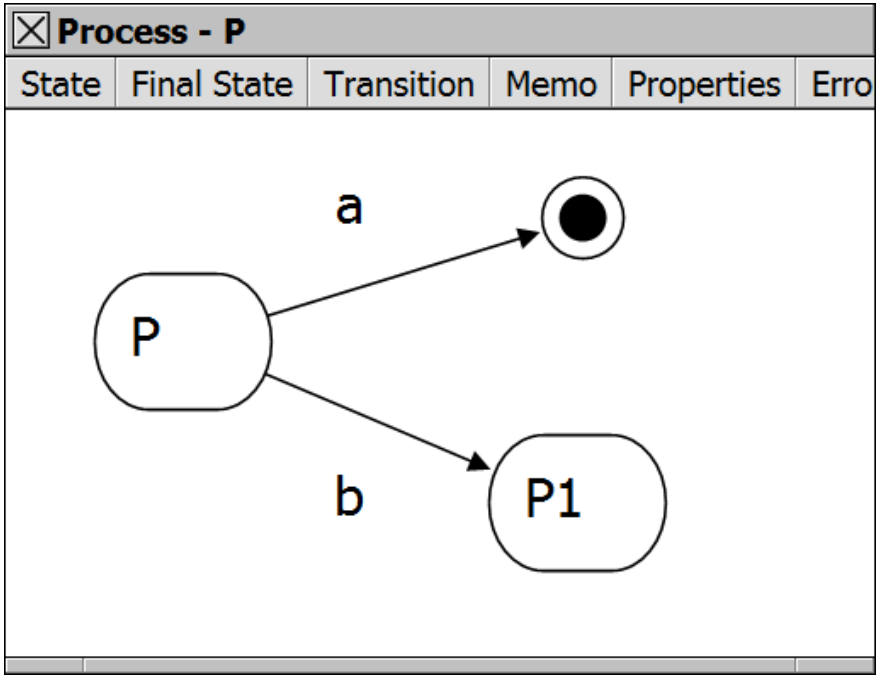


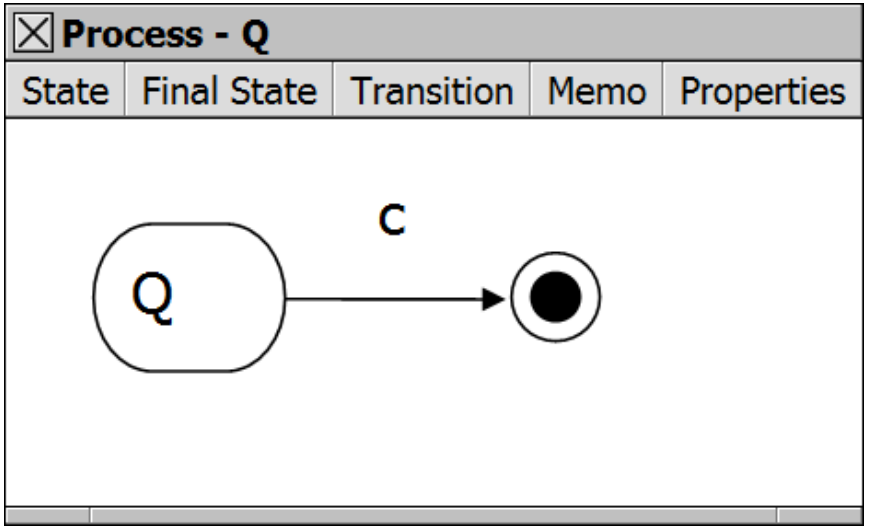
P の終了や Q への移行は
外からは見えない

example-4-0-seq.ss

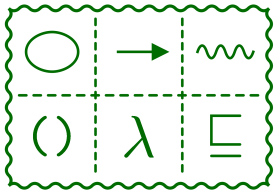
逐次合成の例

```
(define-process SYSTEM
  (seq P Q))
```

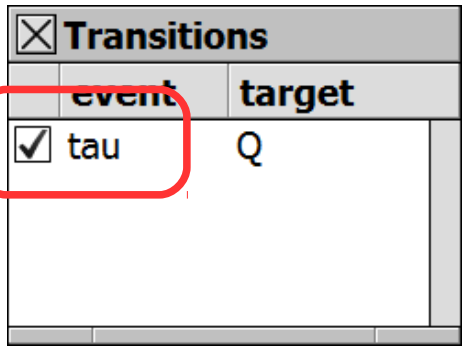




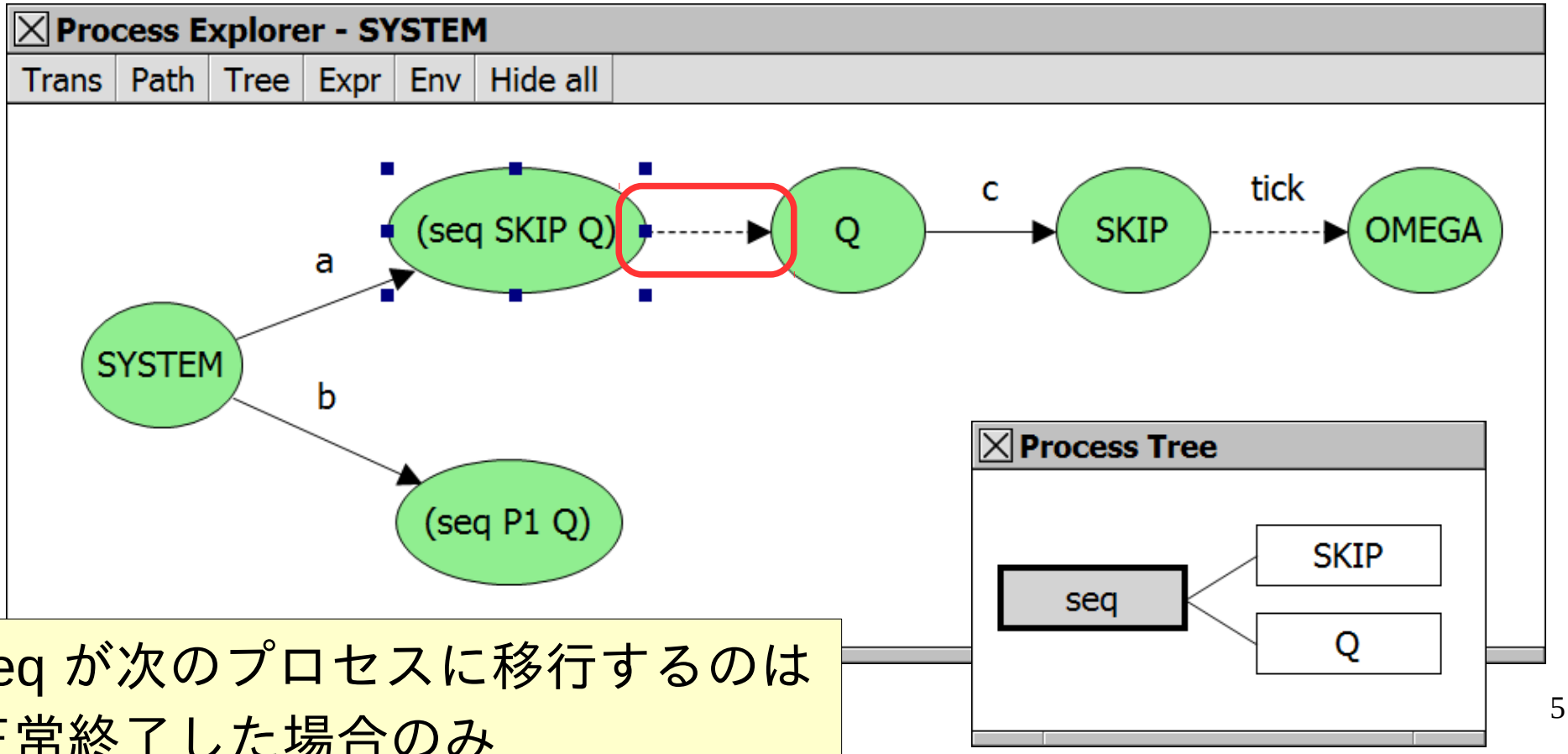
example-4-0-seq.ss †



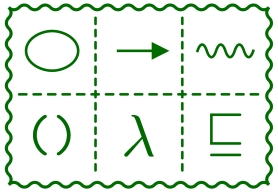
内部イベント tau



P の終了は外部から見えない tau という特別なイベントになる
tau は同期なしに自発的に発生する

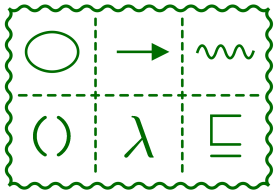


seq が次のプロセスに移行するのは
正常終了した場合のみ

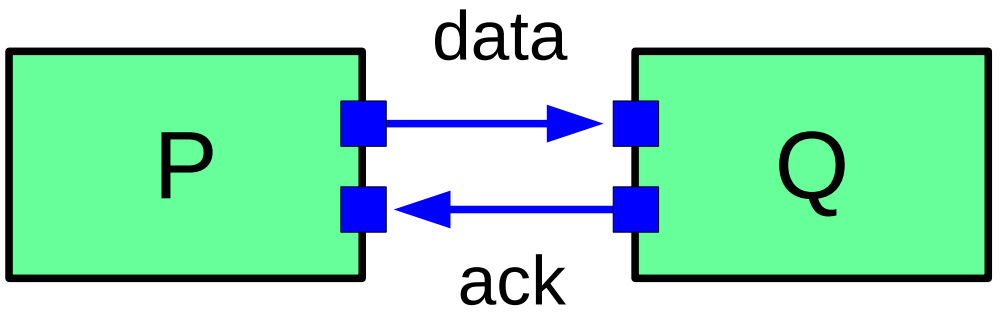


逐次合成 seq の応用

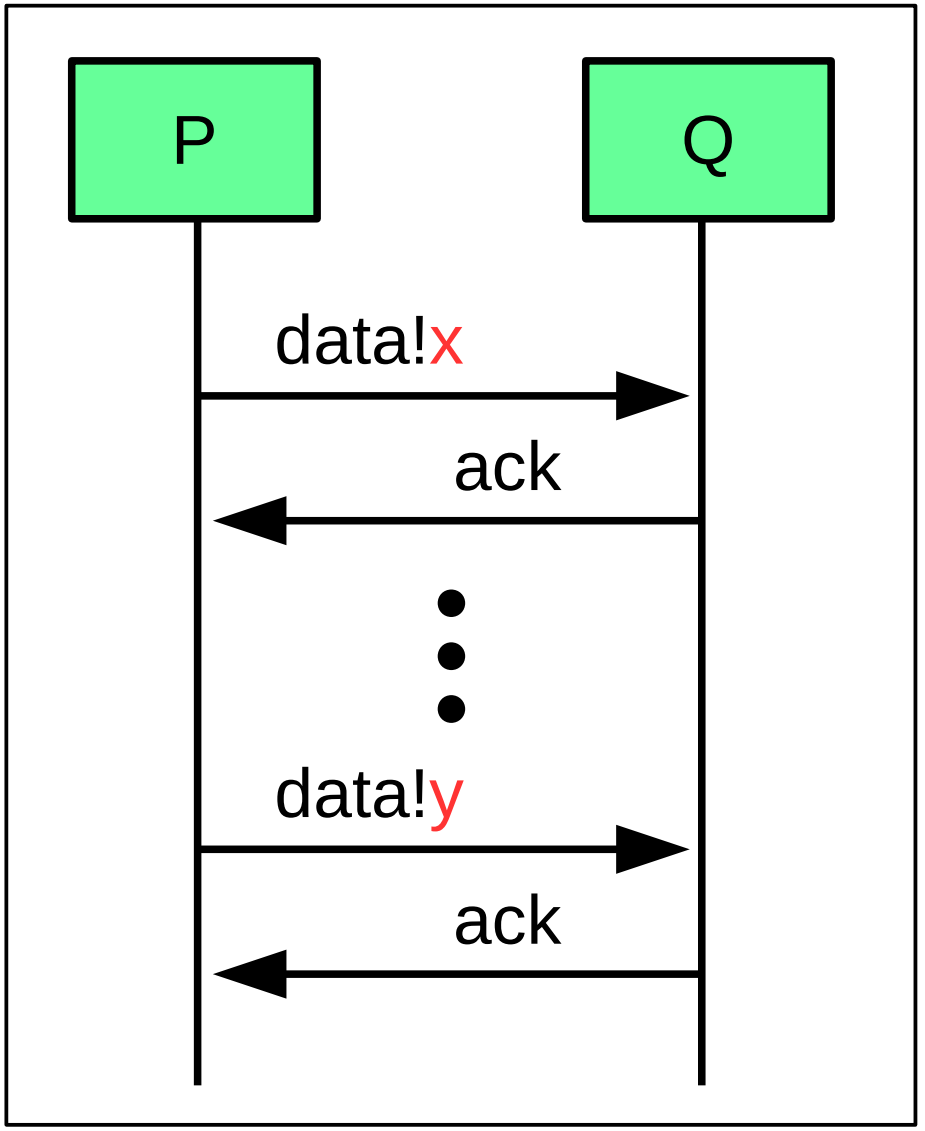
- プロセスの再利用
- プロセスの合流
- メモプロセス

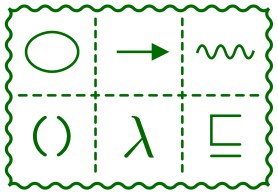


プロセスの再利用




```
(define-process P
  ...
  (! data (x)
   (! ack
    ...
    (! data (y)
     (! ack
       ...
```





プロセスの再利用

共通部分を実行するプロセス Comm を定義すると
逐次実行 seq を使って関数呼び出しのようなことができる

```
(define-process P
  ...
  (seq
   (Comm x)
   ...
   (seq
    (Comm y)
    ...
```

```
(define-process (Comm z)
  (! data (z)
    (! ack SKIP)))
```

途中の処理も SKIP で区切り
1つの seq にまとめてもよい

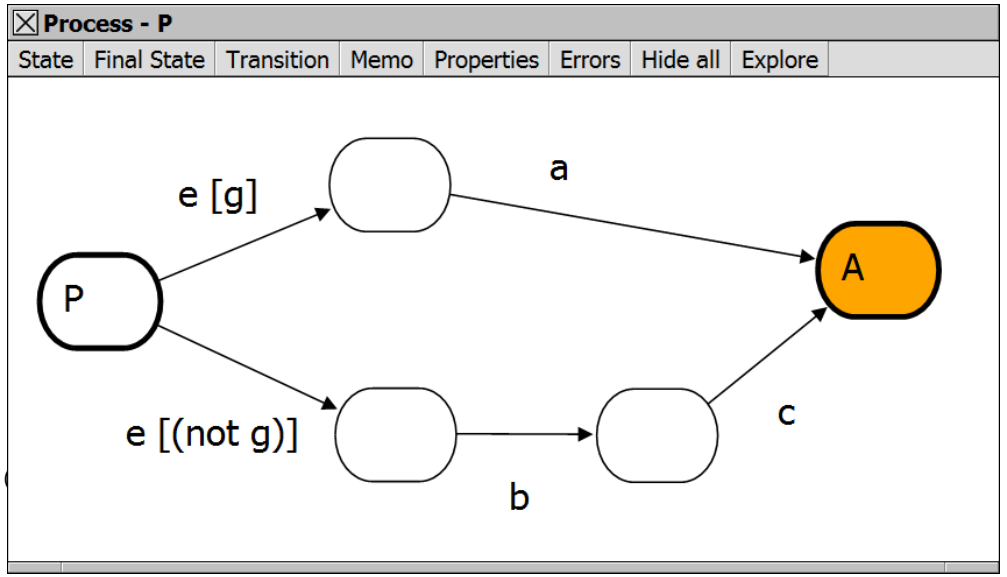
プロセスの合流

```
(define-process P
 (! e
  (if condition
    (! a A )
    (! b (! c A ))))
```

分岐先に共通部分がある場合

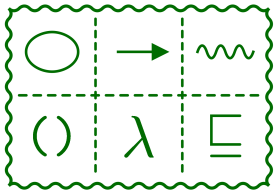
別プロセスを定義する方法

状態遷移図では合流すればよい




```
(define-process P
 (! e
  (if condition
    (! a A)
    (! b (! c A)))))
```

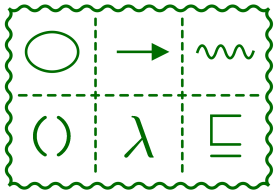
```
(define-process A
  A )
```



プロセスの合流

seq と SKIP を使って合流する

```
(define-process P
 (! e
  (seq
   (if condition
     (! a SKIP)
     (! b (! c SKIP)))
    A )))
```

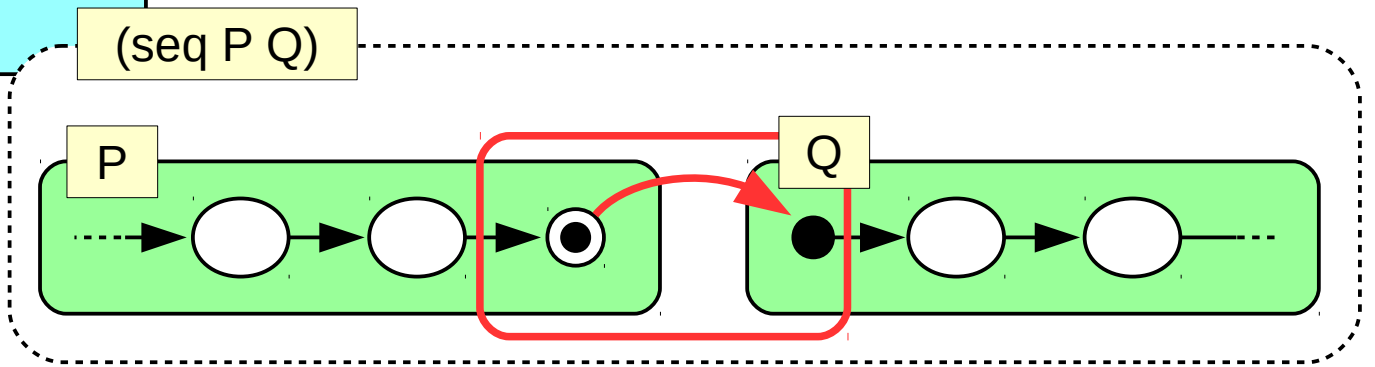


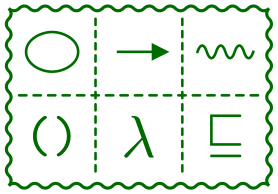
メモプロセス

```
(define-process P
  ...
  (seq
   (Comm x)
   ...
   (seq
    (Comm y)
    ...
```

seq を使ったプロセスの再利用ではプロセスパラメータを通じて値を渡すことはできるが，値を受け取ることはできない

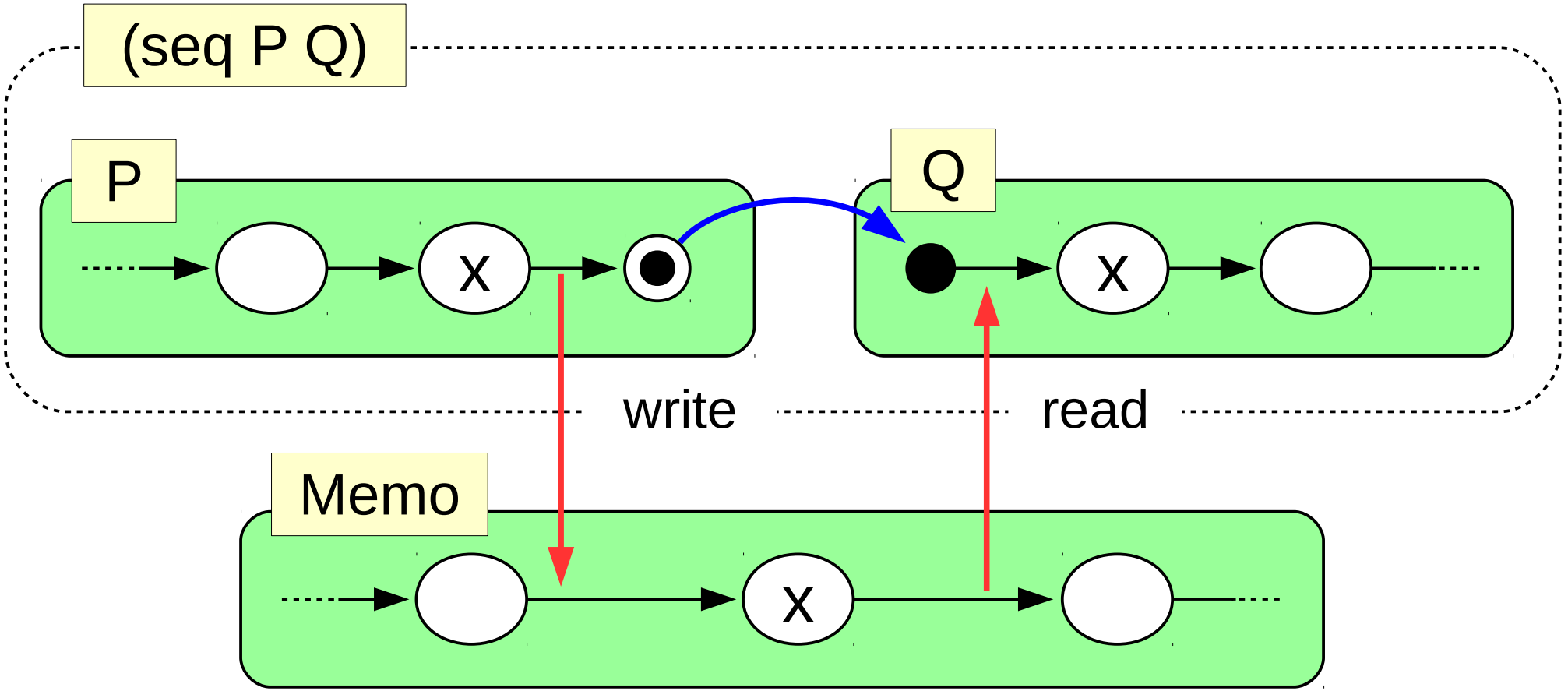
seq におけるプロセスの移行でデータを渡すことができないため





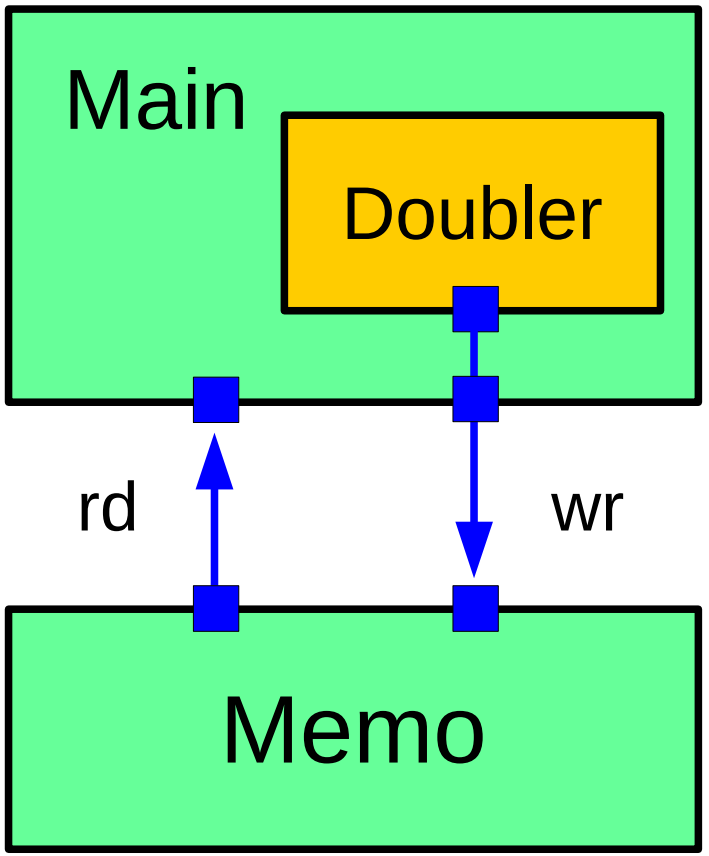
メモプロセス

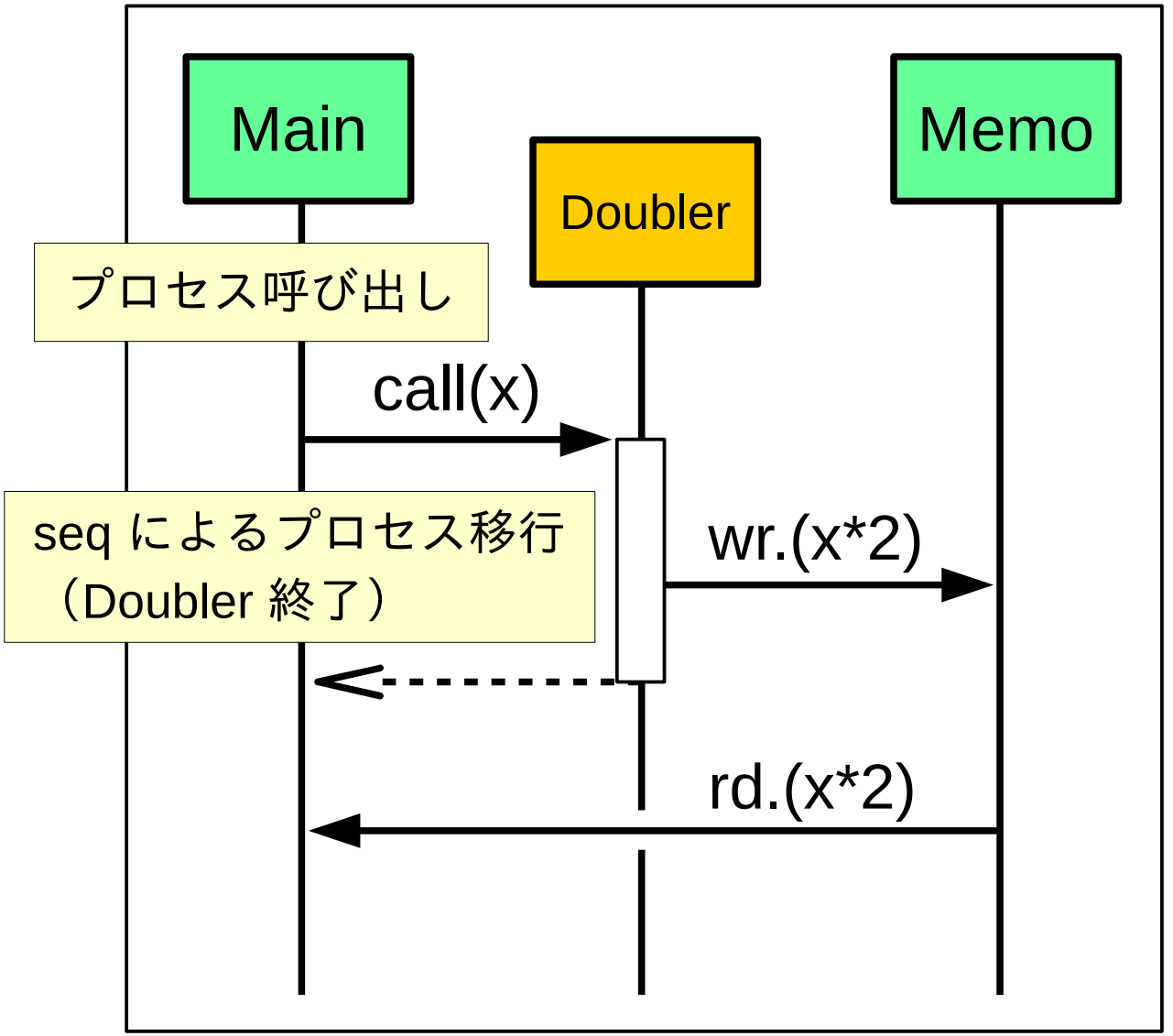
値の受け渡しを仲介するプロセスを並行に走らせる

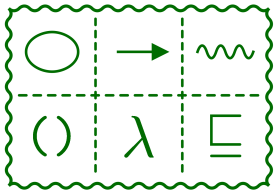


このようなプロセスを従属プロセスという

例: 計算をする従属プロセス



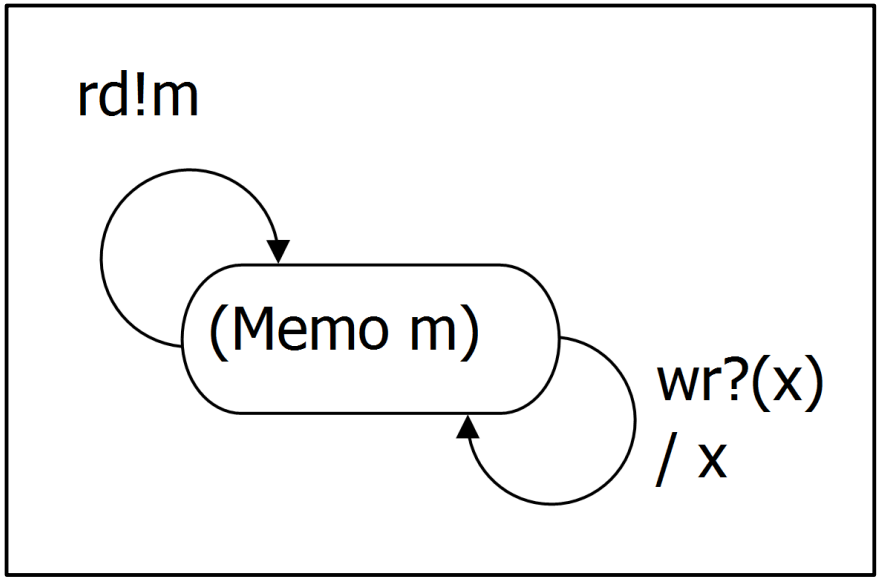




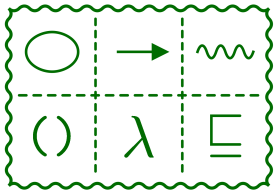
例: 計算をする従属プロセス

```
(define-process Main
 (seq
  (Doubler 1)
  (? rd (x)
   (seq
    (Doubler x)
    (? rd (y)
     (! out (y) SKIP))))))
```

```
(define-process (Doubler x)
  (! wr ((* x 2)) SKIP))
```

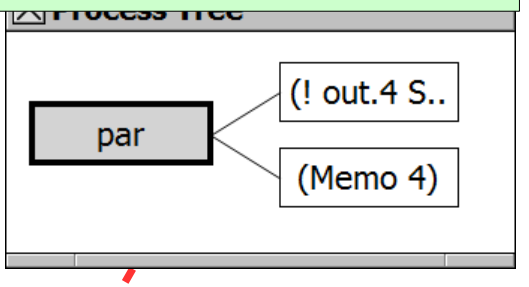


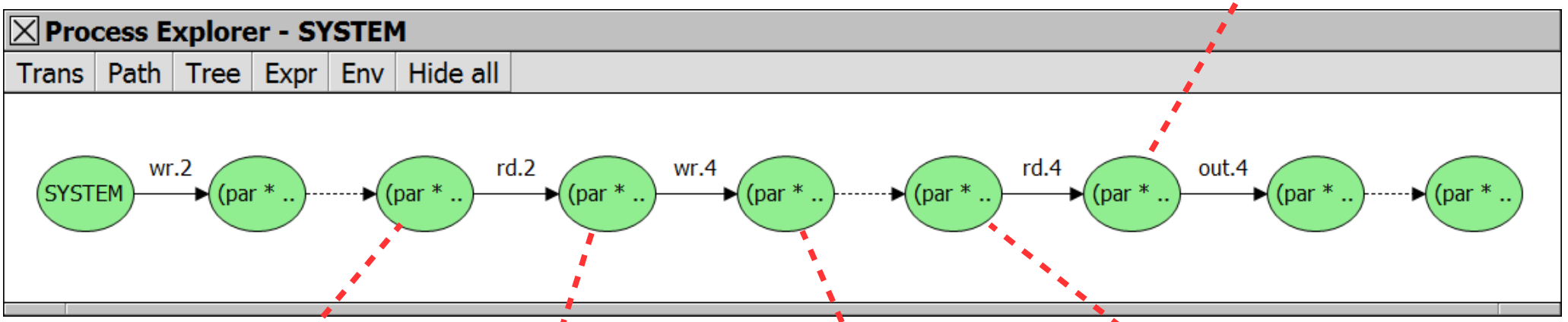

```
(define-process SYSTEM
  (par (list rd wr)
    Main (Memo 0)))
```

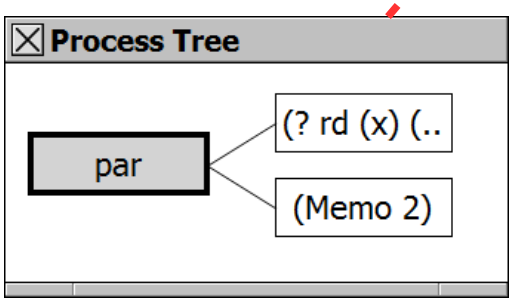


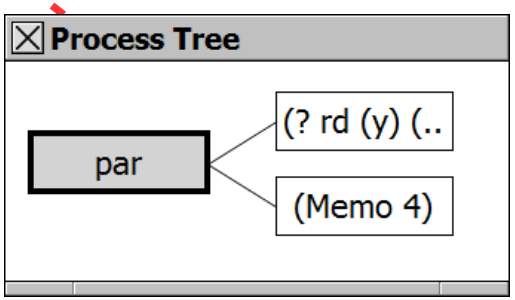
計算木とプロセス木

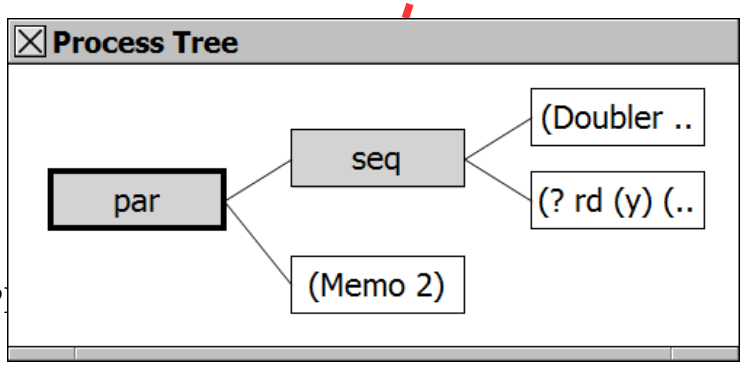
example-4-1-memo.ss †

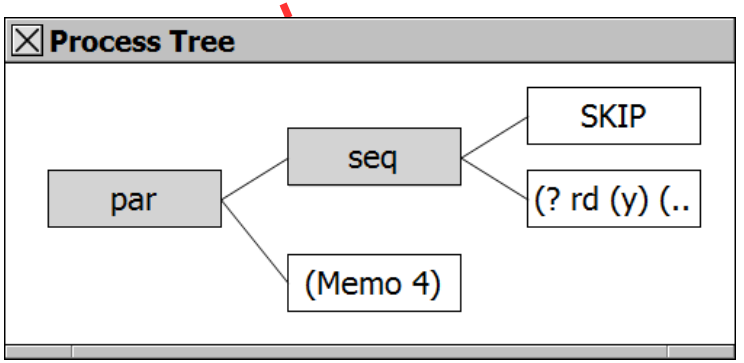


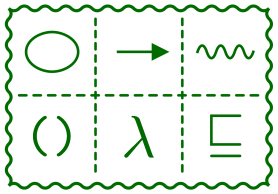






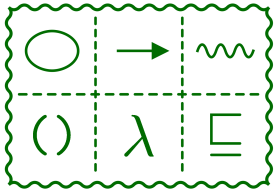






逐次合成まとめ

- 演算子 seq によってプロセスを逐次合成することができる．引数プロセスは指定された順に実行される
- seq 内部でのプロセスの終了および移行は外部から見えない．終了イベント tick は内部イベント tau に変換される
- イベント tau は内部動作を表す特別なイベントである
  - tau は自発的に発生する
  - tau は同期しない．他のプロセスから観測できない



逐次合成まとめ: テクニック

- プロセスの再利用
  - seq を使うとプロセスを再利用できる
  - 関数呼び出しのようなことができる
- プロセスの合流
  - seq を使うと分岐先にある共通な処理の記述を1つにまとめることができる
- メモプロセス
  - メモプロセスはプロセス間でのデータ受け渡しを仲介する
  - メモプロセスを使うと seq におけるプロセス移行をまたいでデータを渡すことができる