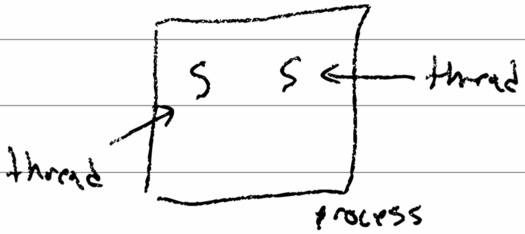
14-1 Concurrency

- Thread



Process = Resources × List < Thread >

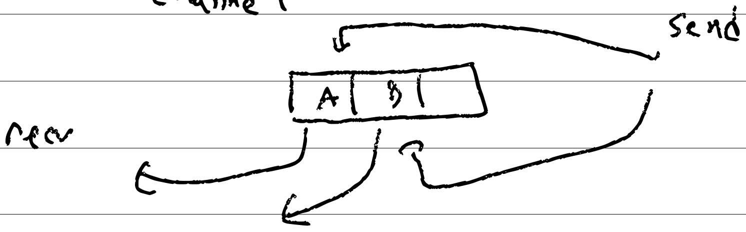
Thread - Continuation

CEK T = < ~~C~~ C, env, K, k . . . >

5

14-2/
```
(define ready-q empty)
(define spawn!
  (λ (f)
    (set! ready-q (cons f ready-q))))
(define exit!
  (λ ()
    (case ready-q of
      [inl _ -> unit]
      [inr p -> (begin (set! ready-q (snd p))
                       ((fst p)))])))
      (let ([mem ...])             (print
19-3/  (spawn!   (λ ()    (+  2  4))   (exit!)))
                          (print
       (spawn!   (λ ()    (+  2  3))   (exit!)))
       (exit!)   )
```

Message - passing concurrency
- channel



If you send, you block until it is received.

Channels are synchronous

14-4)
```
(define make-channel
    (λ () (box empty)))
(define send!
    (λ (chb v)
     (case (first* (unbox chb) (inl false)) of
      [inl _ -> (let/cc K
                  (set-box! chb (cons (inl (pair v K))
                                  (unbox chb) ))
                  (exit!))]
      [inr K -> (begin (spawn! (λ () (k v)))
                  (set-box! chb (rest* (unbox chb)]))))
```

14-5

```
(define first*
  (λ (l def)
    (case l of
      [inl _ → def]
      [inr p → (fst p)])))
```

rest*

snd

14-b) (define recv!
```
        (λ (chb)
          (let chl := unbox chb in
          (case (first* chl (inr false)) of
            [inl p => (let v := fst p in
                        let k := snd p in
                        (Spawn! (λ () (k unit)))
                        (set-box! chb (rest* chl empty))
                        v]
          [inr _ -> (let/cc k (begin (set-box! chb (cons
                                        (inr k) chl))
                                     (exit!)))]))))
```

14-7/

```
(let ch := (make-channel) in
   (spawn! (λ () (send! ch 5))
   (recv! ch))
```

```
(let ch := (make-channel) in
  (spawn! (λ () (let i = 0 in
                   while true
                     send! ch i
                     set i = i+1)))
  (+ (recv! ch) (+ (recv! ch) (recv! ch))))
```

```
14-8/ (define   make-lock
        (λ  ()   (define   lock-ch   (make-channel))
                 (spawn!  (λ  ()
                   (while  true   (define  unlock-ch  (make-channel))
                     (send!  (recv!  lock-ch)
                             (λ  ()   (send!  unlock-ch  unit)))
                     (recv!  unlock-ch)))
         ((lock!  lock-ch)))
     (define  (lock!  lock-ch)
      λ()(define   reply-ch   (make-channel))
       (send!   lock-ch    reply-ch)
       (recv!   reply-ch)))
```

14-9) future / promise

```
(define (make-future f)
  (define reply-ch (make-channel))
  (spawn! (λ () (define v (f))
                (while true
                  (send! reply-ch v))))
  (λ () (recv! reply-ch)))
```

14-10/ Add "special channels" for stdio

$\rightsquigarrow$ nodejs