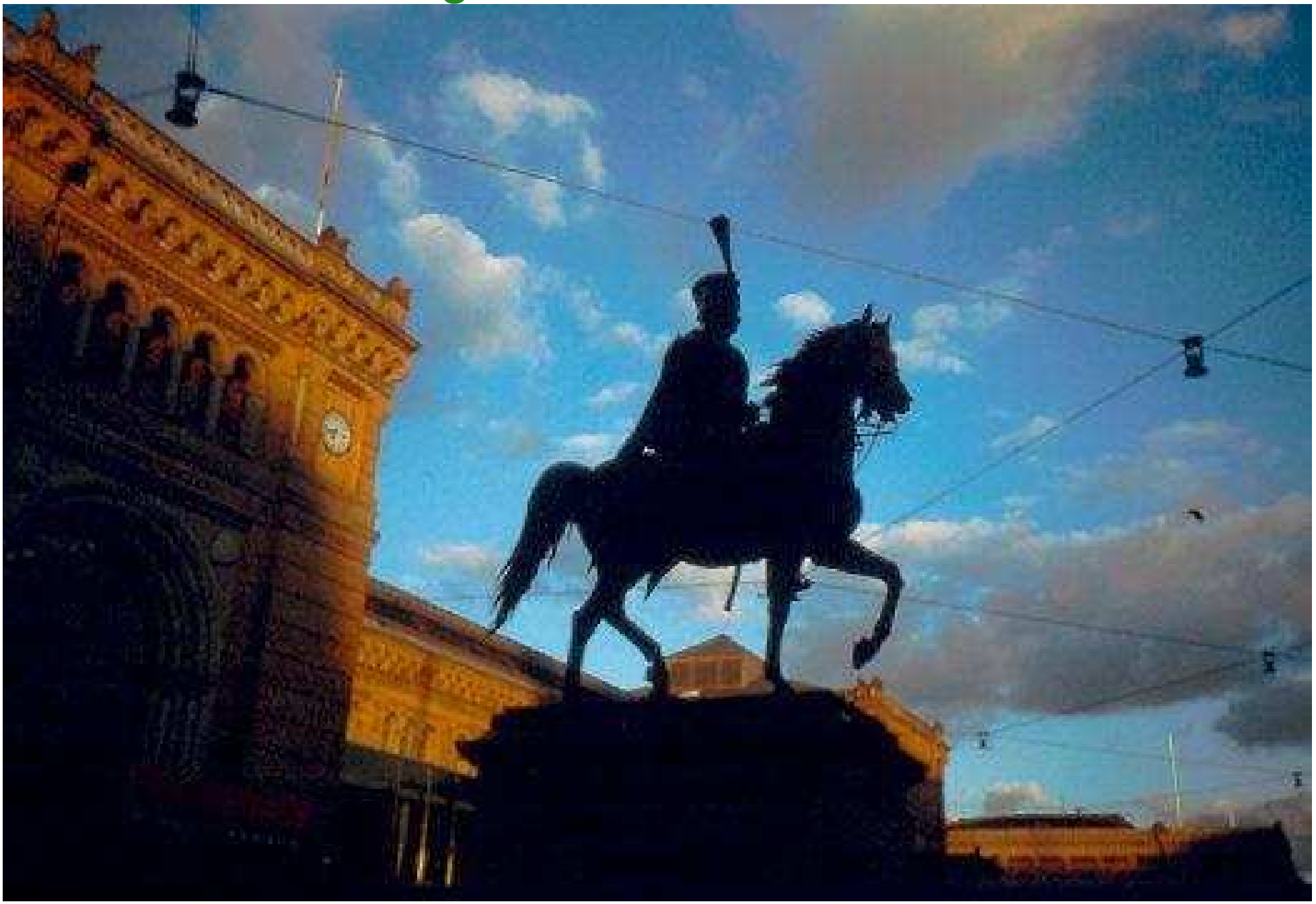
Why (car '()) Is Not An Exception

Or: *This* Exception System Is Not *That* Exception System

Mike Sperber

DeinProgramm

Getting Back to A Known Place



What's an Exception?

- file not found
- division by 0
- `(car '())`
- timer interrupt
- mixed-mode arithmetic

What's an Exception?

- file not found (exceptional situation)
- division by 0
- `(car '())`
- timer interrupt
- mixed-mode arithmetic

What's an Exception?

- file not found (exceptional situation)
- division by 0 (bug)
- `(car '())` (bug)
- timer interrupt
- mixed-mode arithmetic

What's an Exception?

- file not found (exceptional situation)
- division by 0 (bug)
- `(car '())` (bug)
- timer interrupt (asynchronous event)
- mixed-mode arithmetic

What's an Exception?

- file not found (exceptional situation)
- division by 0 (bug)
- `(car '())` (bug)
- timer interrupt (asynchronous event)
- mixed-mode arithmetic (request for implementation extension)

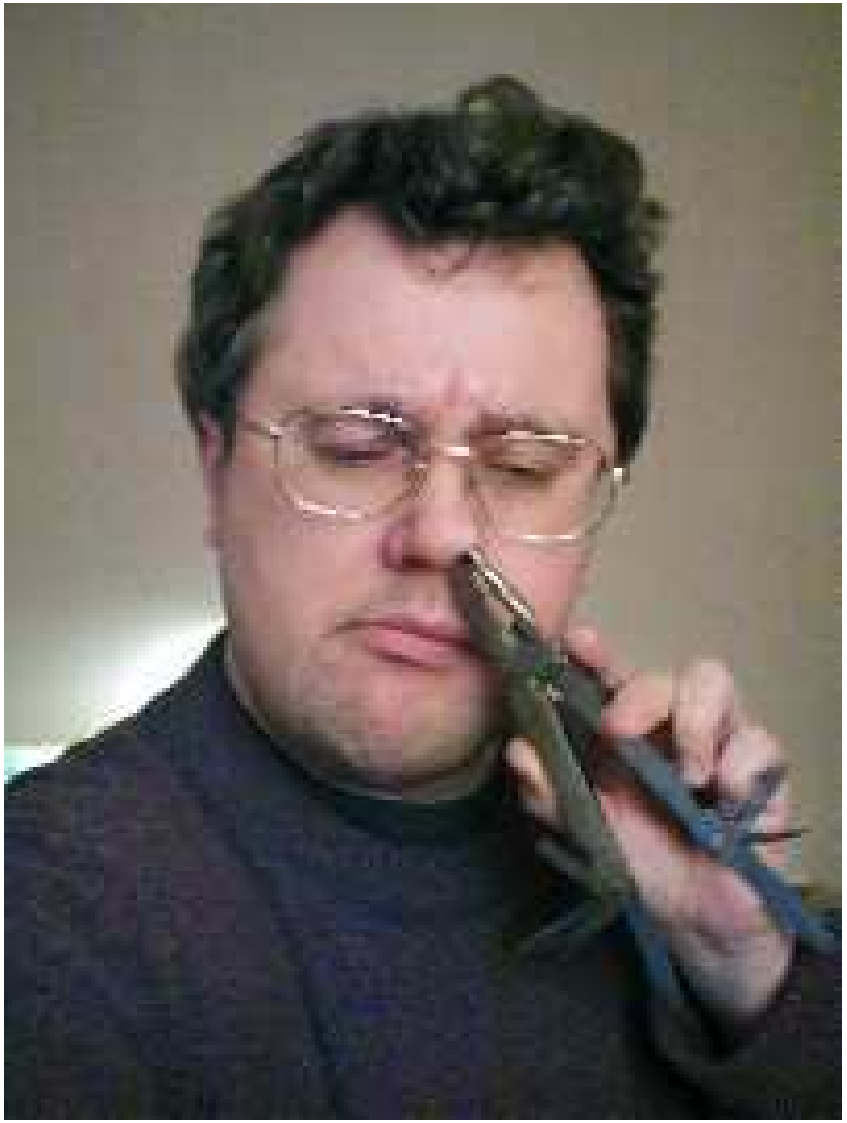
Possible Reactions

- ostrich approach
- abort program
- start debugger
- return to some earlier continuation
- fix, then continue
- ...

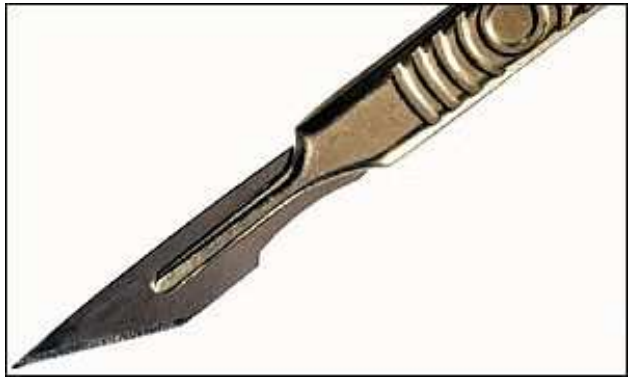
Possible Requirements

- resumption possible
- resumption fast
- handling fast
- problem description rich
- ...

The Right Tool for the Job



or



?

Exceptions for Programs

- purpose: communication between programs
- handlers are installed *often*
- handlers are *bound*, not set
- exceptions are *rare*
- resumption is *unusual*
- descriptions are *rich*

Bugs

- purpose: test suites, graceful program abortion
- handlers are installed *rarely*
- handlers are *set*, not bound
- exceptions are *rare*
- resumption is *very rare*
- descriptions are *rich*

Asynchronous Events

- purpose: notice and synchronize to external events
- handlers are installed *rarely*
- handlers are *set*, not bound
- exceptions occur *very often*
- resumption is *frequent*
- descriptions are *few*

Requests for System Extension

- purpose: modular system extension
- handlers are installed *rarely*
- handlers are *set*, not bound
- exceptions occur *frequently*
- resumption is *frequent*
- descriptions are *from a finite set*

SRFI 35: Conditions

(with Richard Kelsey)

```
(define-condition-type &i/o-filename-error
                       &i/o-error
  i/o-filename-error?
  (filename i/o-error-filename))

(define c1
  (condition
    (&i/o-filename-error
      (filename "/bermuda/triangle/r6rs.txt"))))

(i/o-filename-error? c1) ⇒ #t

(i/o-error-filename c1) ⇒
"/bermuda/triangle/r6rs.txt"
```

Subtyping between Condition Types

```
(define-condition-type &i/o-error &error
  i/o-error?)

(define-condition-type &error &serious
  error?)

(define-condition-type &serious &condition
  serious-condition?)
```

Multiple Conditions at Once

Networking error while accesing NFS file:

```
(define c2
  (condition (&i/o-read-error (port p))
             (&network-read-error (socket s))))
```

```
(i/o-read-error? c2) ⇒ #t
```

```
(network-read-error? c2) ⇒ #t
```

```
(i/o-error-port c2) ⇒ p
```

```
(network-error-socket c2) ⇒ p
```

More Condition Types

```
(define-condition-type &message &condition
  message-condition?
  (message condition-message))

(define-condition-type &i/o-no-such-file-error
                       &i/o-filename-error
  i/o-no-such-file-error?)
```

SRFI 34: Exception Handling for Programs

(mit Richard Kelsey)

```
(call-with-current-continuation
 (lambda (k)
   (with-exception-handler (lambda (x)
                             (display "condition: ")
                             (write x)
                             (newline)
                             (k 'hot))
     (lambda ()
       (+ 1 (raise 'hell))))))
```

```
condition: hell
```

⇒ **hot**

Raise + Continuations

```
(with-exception-handler
      (lambda (x)
        (display "Houston, we have a problem")
        (newline)
        'dont-care)
  (lambda ()
    (+ 1 (raise 'problem))))
```

Houston, we have a problem

⇒ *unspecified*

Common Case: No Resumption

```
(guard (condition
          ((eq? 'heaven condition)
           'sunny)
          ((eq? 'hell condition)
           'hot))
 (raise 'hell))
```

⇒ *hot*

Context Issues

```
(call-with-current-continuation
 (lambda (k)
   (with-exception-handler (lambda (x)
                             (display "reraised ")
                             (write x) (newline)
                             (k 'neutral))
     (lambda ()
       (guard (condition ((eq? 'heaven condition)
                          'sunny)
                         ((eq? 'hell condition)
                          'hot))
         (raise 'purgatory))))))
```

reraised purgatory

⇒ *neutral*

Everyday Use

```
(guard (condition
        ((i/o-filename-error? condition)
         (display "I/O error opening file ")
         (display (i/o-error-filename condition))
         ...)
        ((i/o-error? condition)
         (display "I/O error")
         ...))
  (read-file "/bermuda/triangle/r6rs.txt"))
```

Ingredients

- current exception handler
- dynamic context
- no control flow for the primitives
- **`guard`** for the common case: dispatch + unwinding

Conclusions

- no single exception system is good for everyone
- ***design for a specific problem***
- ***abstract when you're finished***
- ***... or not.***