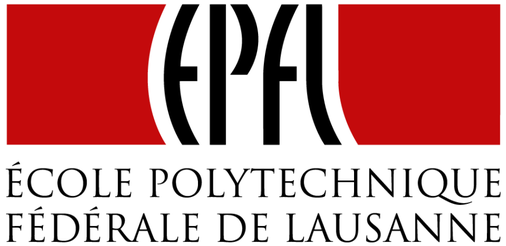


Latency as an Effect (1/2)

Principles of Reactive Programming

Erik Meijer

The Four Essential Effects In Programming

	One	Many
Synchronous	`T/Try[T]`	`Iterable[T]`
Asynchronous	`Future[T]`	`Observable[T]`

The Four Essential Effects In Programming

	One	Many
Synchronous	`T/Try[T]`	`Iterable[T]`
Asynchronous	`Future[T]`	`Observable[T]`

Recall our simple adventure game ….

```
trait Adventure {
  def collectCoins(): List[Coin]
  def buyTreasure(coins: List[Coin]):Treasure
}
```

```
val adventure = Adventure()
val coins = adventure.collectCoins()
val treasure = adventure.buyTreasure(coins)
```

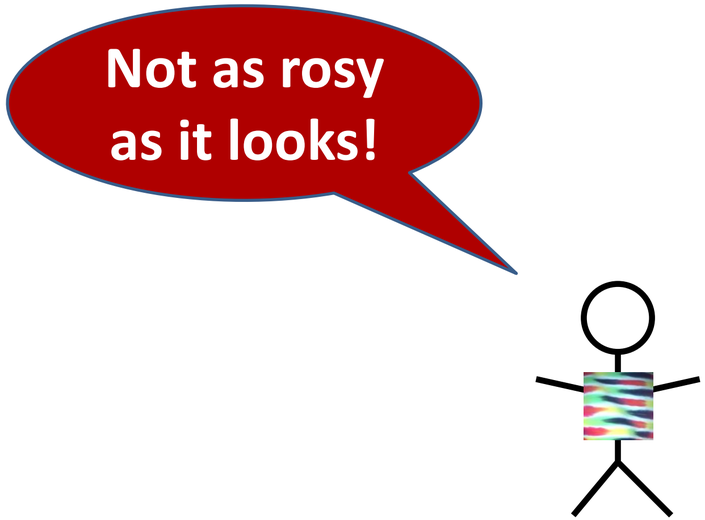
Recall our simple adventure game ….

```
trait Adventure {
  def readFromMemory(): Array[Byte]
  def sendToEurope(packet: Array[Byte]):
}    Array[Byte]


val socket = Socket()
val packet = socket.readFromMemory()
val confirmation =
  socket.sendToEurope(packet)
```

It is actually very similar to a simple network stack

```
trait Socket {
  def readFromMemory(): Array[Byte]
  def sendToEurope(packet: Array[Byte]):
Array[Byte]
}
```




```
val socket = Socket()
val packet = socket.readFromMemory()
val confirmation = socket.sendToEurope(packet)
```

Timings for various operations on a typical PC

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

http://norvig.com/21-days.html#answers

Sequential composition of actions that take time

```
val socket = Socket()
val packet = socket.readFromMemory()
// block for 50,000 ns
// only continue if there is no exception
val confirmation = socket.sendToEurope(packet)
// block for 150,000,000 ns
// only continue if there is no exception
```

Sequential composition of actions

Lets translate this into human terms.

1 nanosecond

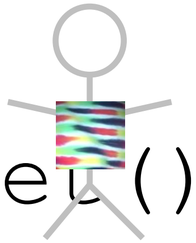
$\rightarrow$

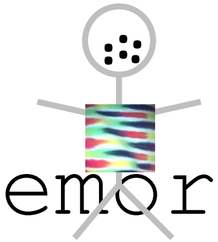
1 second (then hours/days/months/years)

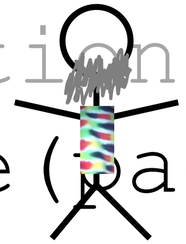
Timings for various operations on a typical PC on human scale

execute typical instruction	1 second
fetch from L1 cache memory	0.5 seconds
branch misprediction	5 seconds
fetch from L2 cache memory	7 seconds
Mutex lock/unlock	½ minute
fetch from main memory	1½ minutes
send 2K bytes over 1Gbps network	5½ hours
read 1MB sequentially from memory	3 days
fetch from new disk location (seek)	13 weeks
read 1MB sequentially from disk	6½ months
send packet US to Europe and back	5 years

Sequential composition of actions







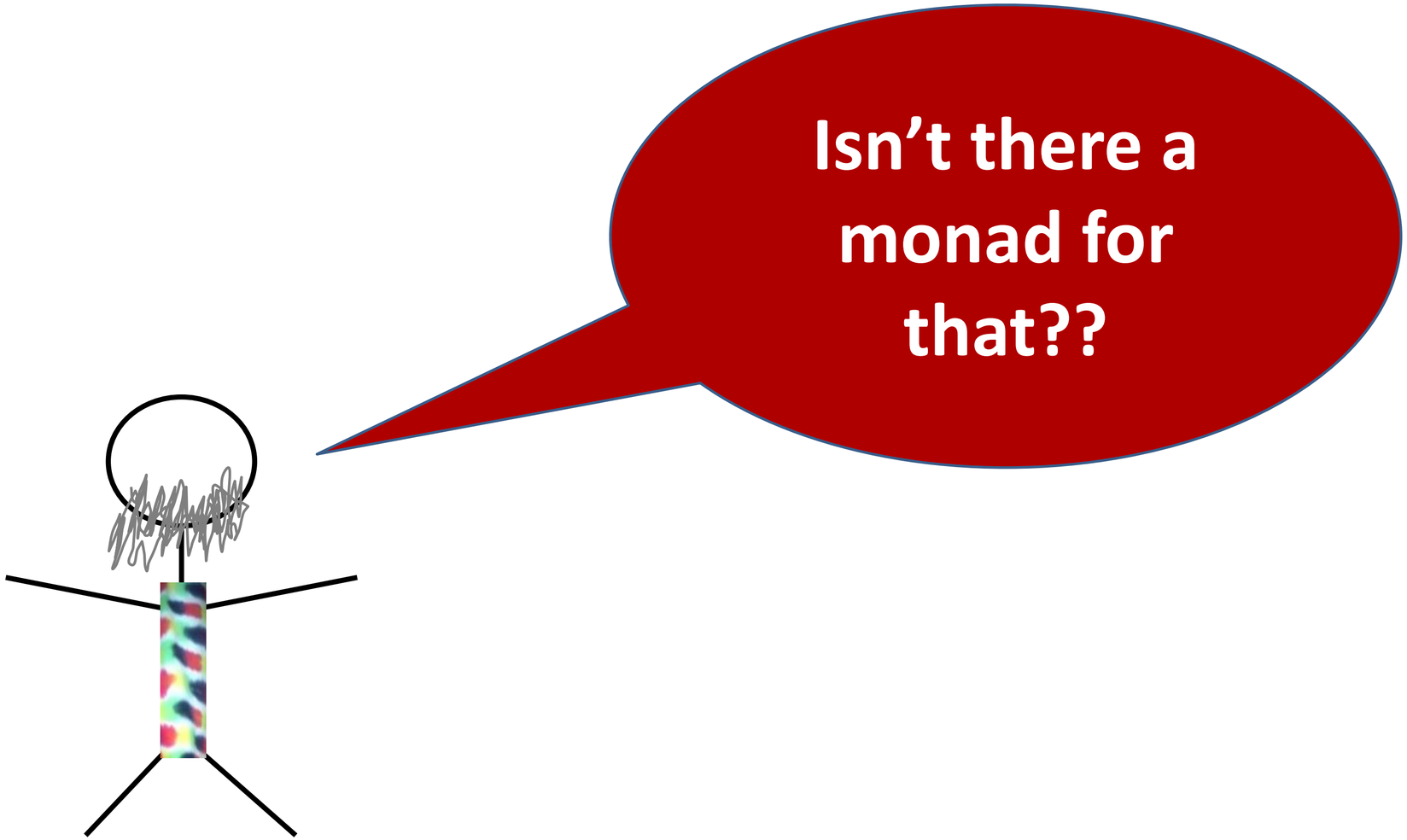
```
val socket = Socket()
val packet = socket.readFromMemory()
// block for 3 days
// only continue if there is no exception
val confirmation = socket.sendToEurope(packet)
// block for 5 years
// only continue if there is no exception
```

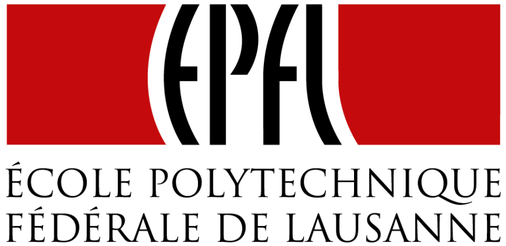
Sequential composition of actions

12 months to walk coast-to-coast
3 months to swim across the Atlantic
3 months to swim back
12 months to walk back

Humans are twice as fast as computers!

Sequential composition of actions that take time and fail





End of Latency as an Effect (1/2)

Principles of Reactive Programming

Erik Meijer