Dean Wampler
dean.wampler@typesafe.com
@deanwampler
polyglotprogramming.com/talks

Typesafe

Two Architectures: Reactive & Parthenon

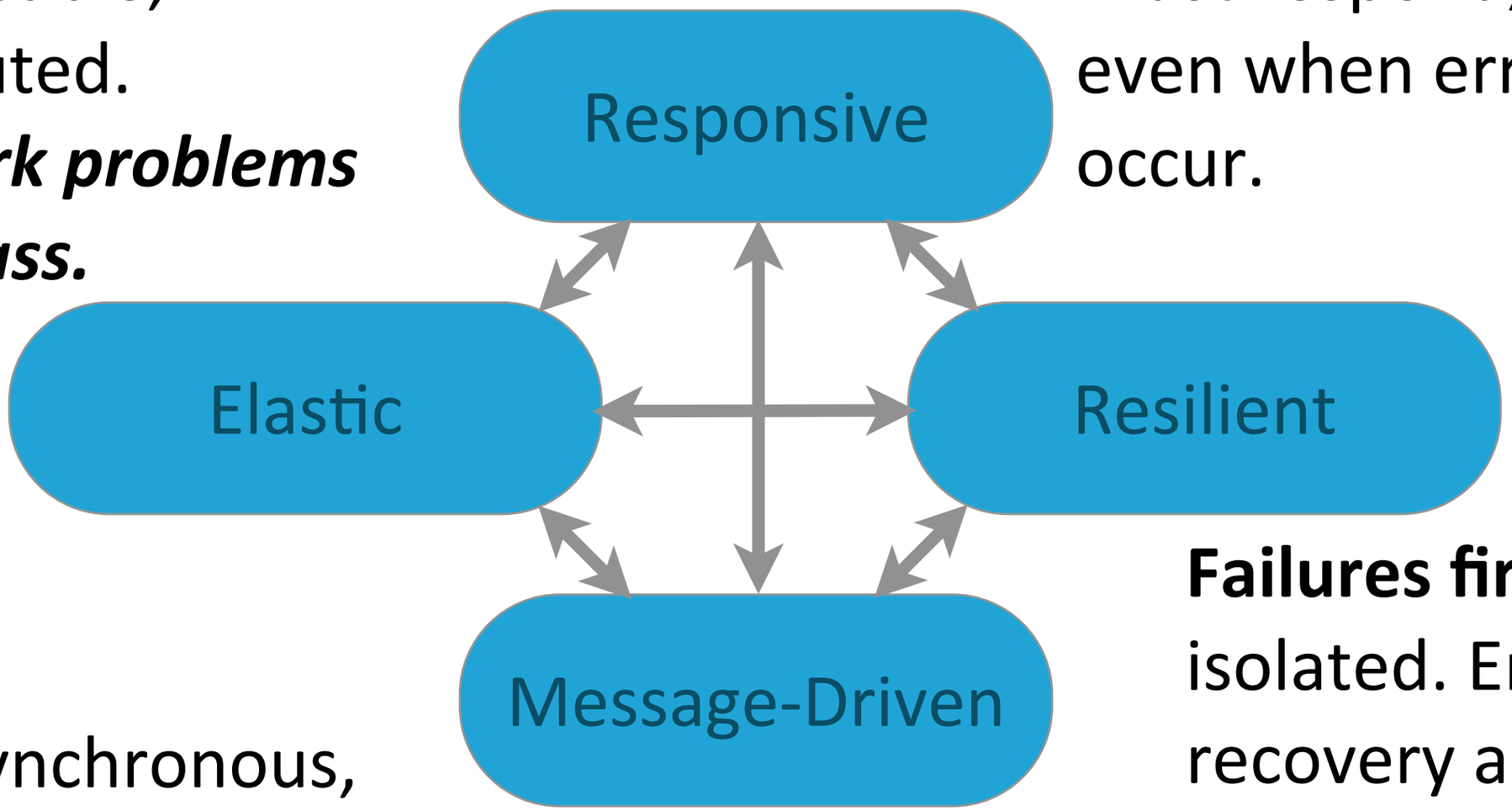
GOTO Copenhagen
September 26, 2014

Reactive Programming

Photo: Tubes in Berlin

Photo: Foggy day in Chicago.

Loosely coupled, composable, distributed. ***Network problems first-class.***



Must respond, even when errors occur.

Asynchronous, non-blocking. Facts as events are pushed.

Failures first-class, isolated. Errors/ recovery are just other events.

reactivemanifesto.org

The bonus slides walk through the traits in more details.

Parthenon Architecture

Friday, September 26, 14

Photo: Parthenon temple in Athens. Photo from Wikipedia.

Dilemma

Dilemma

Ubiquitous language is a compelling idea...

Dilemma

But it leads to fragile, overly-specific OO code.

Dilemma

FP code is better without ubiquitous language.

Dilemma

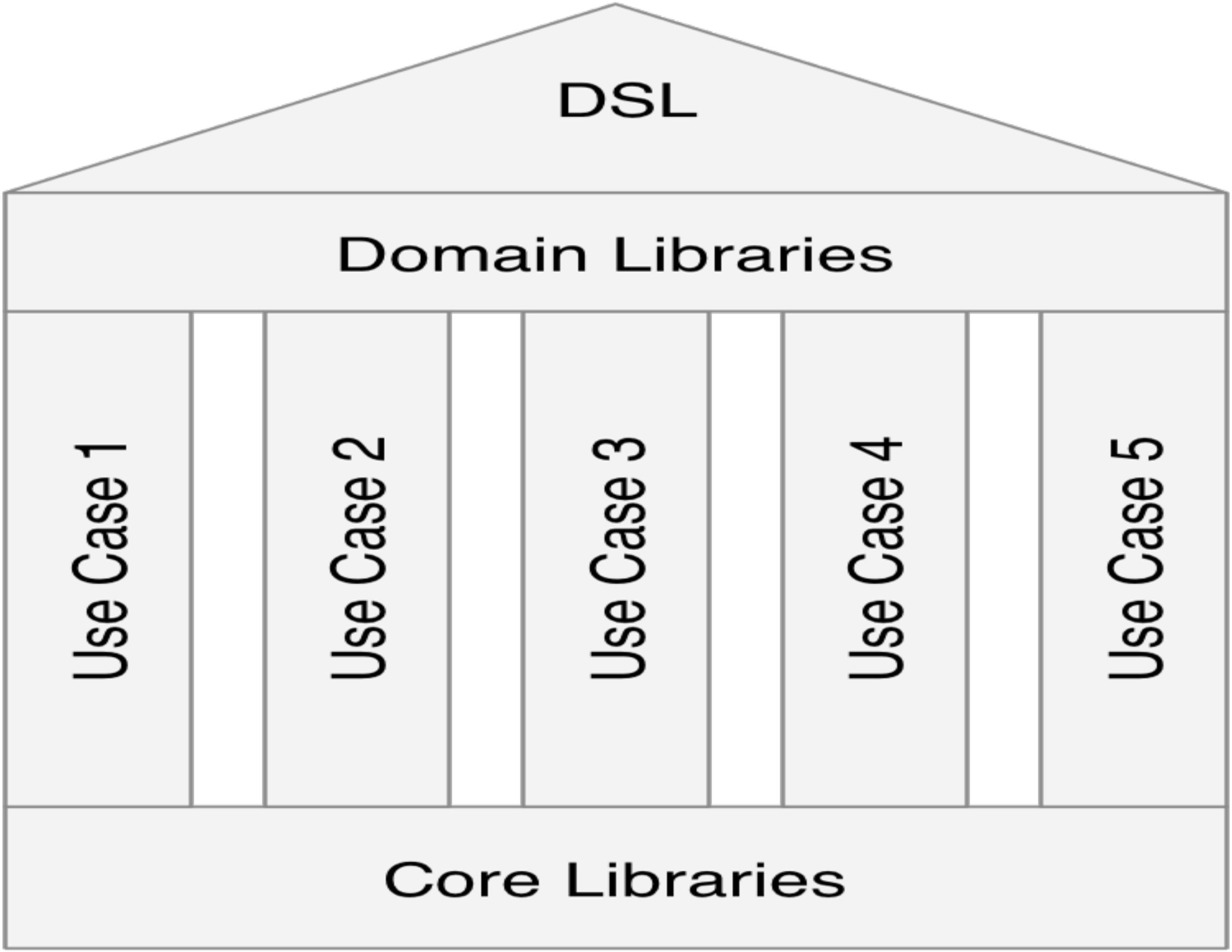
Can we get the benefits of both?

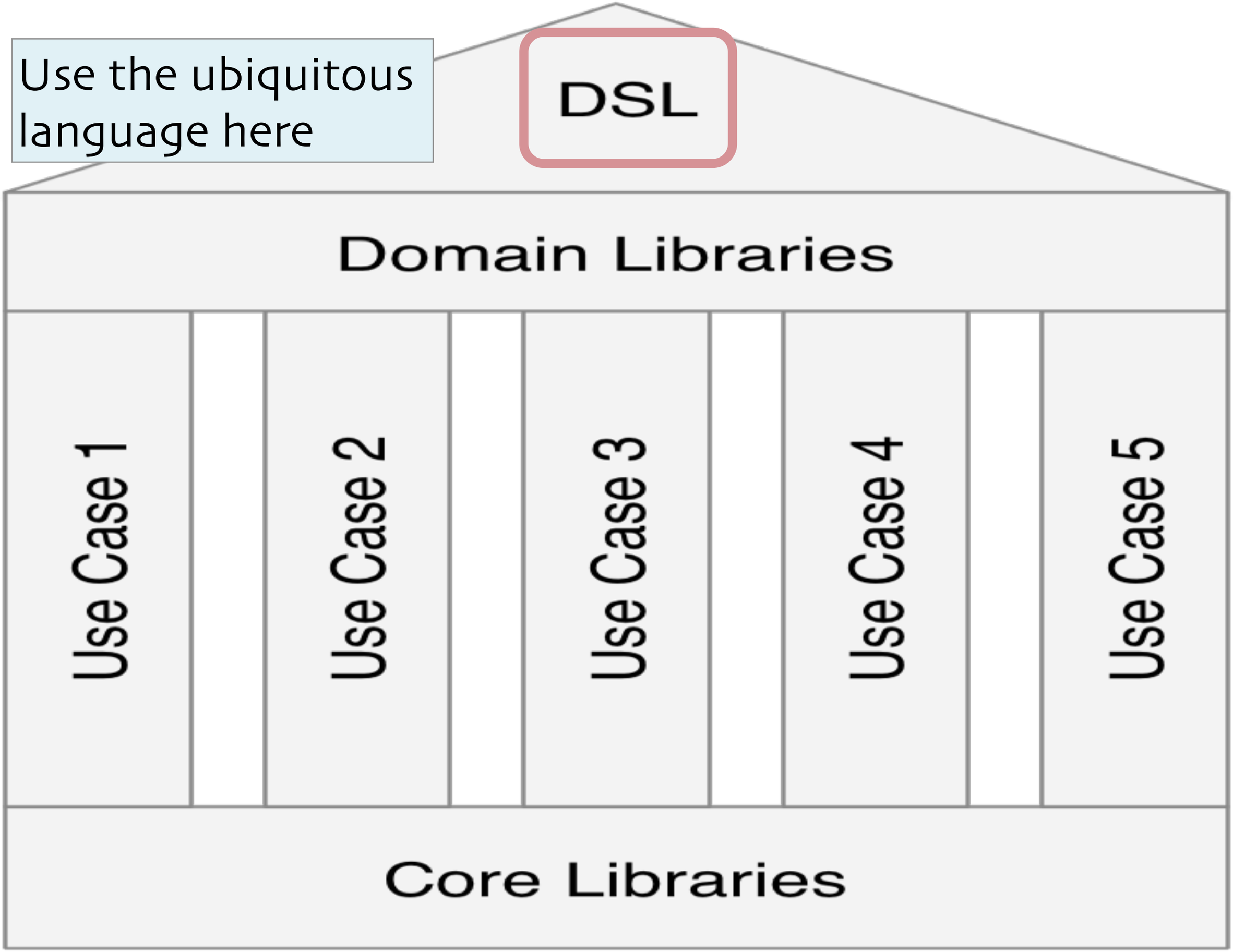
WARNING

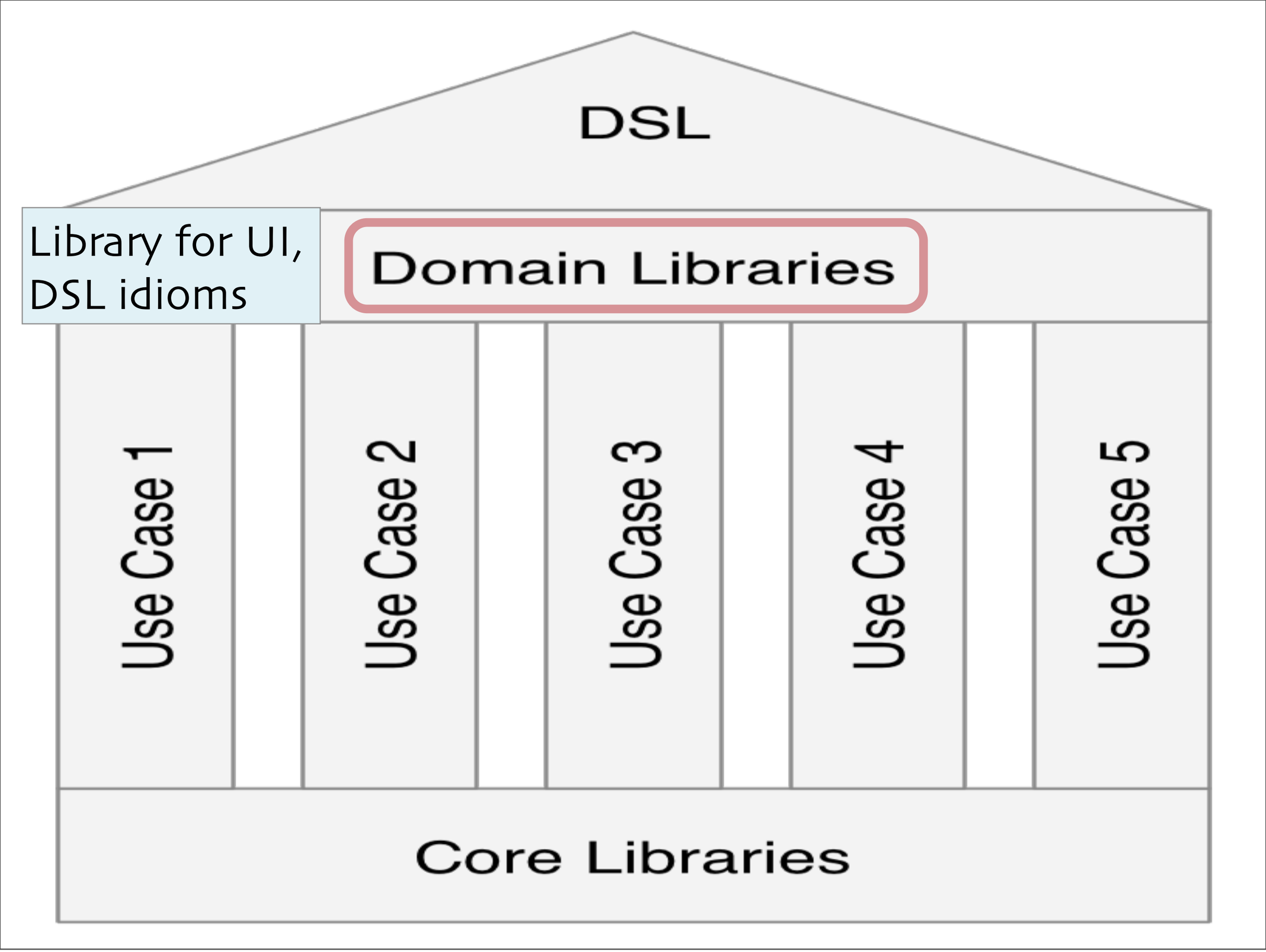
These ideas are half-baked!

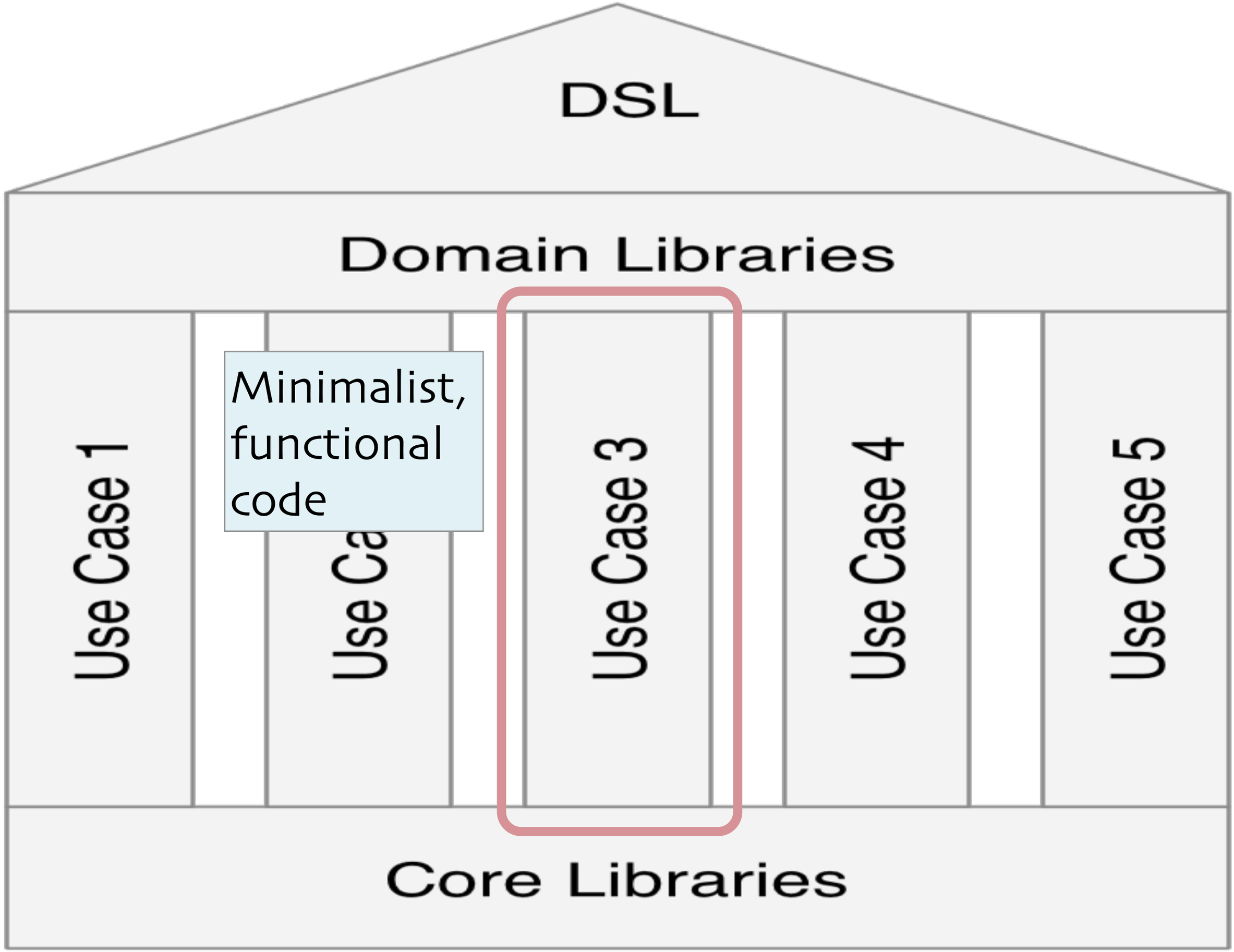
Friday, September 26, 14

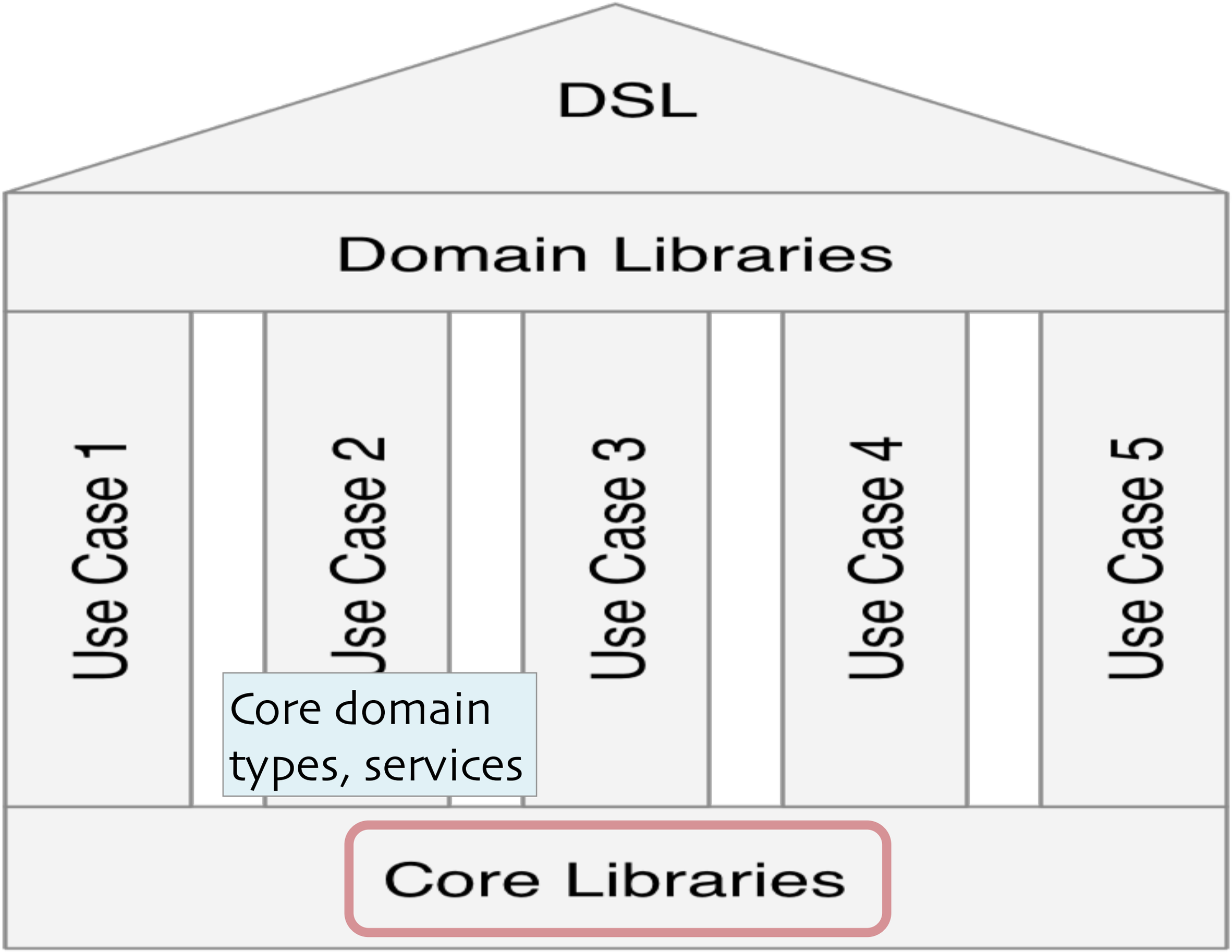
Photo: Parthenon temple in Athens. Photo from Wikipedia.


DSL
Domain Libraries
Use Case 1
Use Case 2
Use Case 3
Use Case 4
Use Case 5
Core Libraries


Use the ubiquitous language here
DSL
Domain Libraries
Use Case 1
Use Case 2
Use Case 3
Use Case 4
Use Case 5
Core Libraries


DSL
Library for UI,
DSL idioms
Domain Libraries
Use Case 1
Use Case 2
Use Case 3
Use Case 4
Use Case 5
Core Libraries


DSL
Domain Libraries
Minimalist, functional code
Use Case 1
Use Case
Use Case 3
Use Case 4
Use Case 5
Core Libraries


DSL
Domain Libraries
Use Case 1
Use Case 2
Use Case 3
Use Case 4
Use Case 5
Core domain types, services
Core Libraries

Example:

Payroll

```scala
// From Programming Scala, 2nd Ed.

case class Money(amount: Double) {
  require(amount >= 0.0,
    s"Negative amount $amount not allowed")

  def +  (m: Money): Money =
    Money(amount + m.amount)
  def -  (m: Money): Money =
    Money(amount - m.amount)
  def >= (m: Money): Boolean =
    amount >= m.amount

  override def toString = "$"+amount
}
```

```
import ...dsl.PayrollParser

object PayrollParthenon {
  val dsl = """biweekly {
      federal tax              %f  percent,
      state tax                %f  percent,
      insurance premiums       %f  dollars,
      retirement savings       %f  percent
    }"""

  type EmployeeData1 = (String, Money, String)

  def readData(data: File):Seq[EmployeeData1] =
    // read each employee record from a
    // file. For each line, convert to
    // a tuple:
    // (employee's name, salary,
    //  rule string for deductions)
```


Example: a payroll calculator that uses a DSL (embedded for the example) for the calculation rules. Each “use case” is implemented as a function. Not all details are shown.

```
val parser = new PayrollParser

def toDeduction(rule: String): Deduction =
  parser.parseAll(rule)

type EmployeeData = (String, Money, Deductions)

def processRules(input: File):
    Seq[EmployeeData] = {
  val data = readData(input)
  for ( (name, salary, rule) <- data )
    yield (name, salary, toDeduction(rule))
}

def biweeklyPayrollPerEmployeeReportUseCase(
    data: Seq[EmployeeData]): Unit = {
  val fmt  = "%-10s %6.2f  %5.2f  %5.2f\n"
  val head = "%-10s %-7s  %-5s    %s\n"
  println("\nBiweekly Payroll:")
```


Example: a payroll calculator that uses a DSL (embedded for the example) for the calculation rules. Each “use case” is implemented as a function. Not all details are shown.

```
  }

  def biweeklyPayrollPerEmployeeReportUseCase(
      data: Seq[EmployeeData]): Unit = {
    val fmt  = "%-10s %6.2f  %5.2f  %5.2f\n"
    val head = "%-10s %-7s  %-5s    %s\n"
    println("\nBiweekly Payroll:")
    printf(head,
      "Name", "Gross", "Net", "Deductions")
    printf(head,
      "----", "-----", "---", "----------")
    for {
      (name, salary, deductions) <- data
      gross = deductions.gross(salary.amount)
      net   = deductions.net(salary.amount)
    } printf(fmt, name, gross, net, gross - net)
  }

  def biweeklyPayrollTotalsReportUseCase(
      data: Seq[EmployeeData]): Unit = {
```


Example: a payroll calculator that uses a DSL (embedded for the example) for the calculation rules. Each “use case” is implemented as a function. Not all details are shown.

```
      gross = deductions.gross(salary.amount)
      net   = deductions.net(salary.amount)
    } printf(fmt, name, gross, net, gross - net)
}

def biweeklyPayrollTotalsReportUseCase(
    data: Seq[EmployeeData]): Unit = {
  val (gross, net) =
    (data foldLeft (0.0, 0.0)) {
     case ((gross, net), (nm, sal, deds)) =>
      val g = deds.gross(sal.amount)
      val n = deds.net(sal.amount)
      (gross + g, net + n)
    }
    printf("\nBiweekly Totals: Gross %7.2f, Net
%6.2f, Deductions: %6.2f\n",
      gross, net, gross - net)
  }

  def main(args: Array[String]) = {
    val input = args(0)
```


Example: a payroll calculator that uses a DSL (embedded for the example) for the calculation rules. Each “use case” is implemented as a function. Not all details are shown.

```
          val n = deds.net(sal.amount)
          (gross + g, net + n)
      }
      printf("\nBiweekly Totals: Gross %7.2f, Net
%6.2f, Deductions: %6.2f\n",
        gross, net, gross - net)
    }

    def main(args: Array[String]) = {
      val input = args(0)
      val data = processRules(new File(input))
      biweeklyPayrollTotalsReportUseCase(data)
      biweeklyPayrollPerEmployeeReportUseCase(data)
    }
}
```


Example: a payroll calculator that uses a DSL (embedded for the example) for the calculation rules. Each “use case” is implemented as a function. Not all details are shown.

Thanks!

Photo: Parthenon temple in Athens. Photo from Wikipedia.