

Veiliger programmeren  
met functionele talen



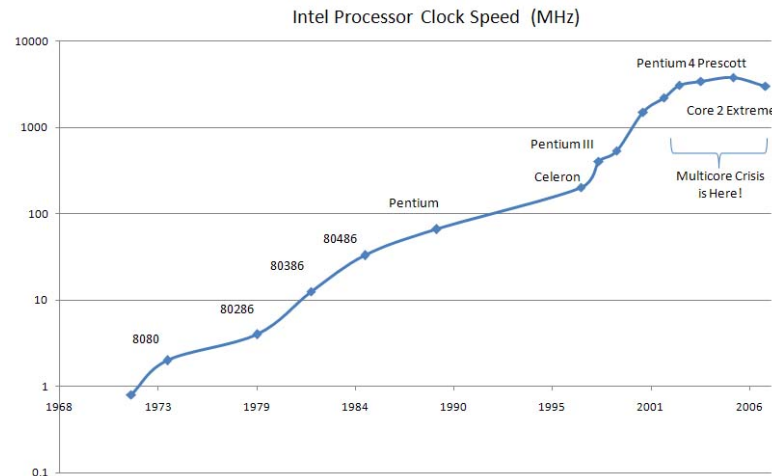
# Waarom functionele programmeertalen?

Door functionele programmeertalen wordt parallel verwerken van gegevens eenvoudiger, minder foutgevoeliger en schaalbarer....

# Schaalbaarheid is noodzakelijk voor complexe IT toepassingen

Binnen ICT wordt het parallel verwerken van gegevens de norm:

- ICT toepassingen vereisen steeds meer processorkracht (big data, web-scale IT).
- Processoren worden niet meer sneller per kern sinds 2005.



Bob Warfield (2007). *A picture of the multicore crisis*. SmoothSpan Blog, 6 september, 2007.  
<https://smoothspan.wordpress.com/2007/09/06/a-picture-of-the-multicore-crisis/>

# Functionele programmeer- talen

1959	Lisp
1975	ML, FP, Scheme
1986	Standard ML
1990	Haskell, Erlang
2000	OCaml
2003	Scala
2005	F#
2007	Clojure

Dominic Graefen (2012). *LISP: How I Learned To Stop Worrying And Love Parantheses*.  
SlideShare, 22 oktober 2012.

<http://www.slideshare.net/devboy/lisp-how-i-learned-to-stop-worrying-and-love-parantheses>

# Functionele programmeertalen zijn nog niet mainstream...

Worldwide, Feb 2015 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	24.7 %	-0.4 %
2		PHP	11.7 %	-1.2 %
3		Python	10.6 %	+0.9 %
4		C#	8.9 %	-0.3 %
5		C++	8.2 %	-0.5 %
6		C	7.8 %	+0.1 %
7		Javascript	7.2 %	-0.3 %
8		Objective-C	6.1 %	-0.2 %
9		Matlab	3.0 %	-0.2 %
10	↑↑	R	2.7 %	+0.6 %
11		Ruby	2.5 %	+0.0 %
12	↑↑↑	Swift	2.5 %	+2.9 %
13	↓↓↓	Visual Basic	2.3 %	-0.7 %
14	↓	Perl	1.3 %	-0.3 %
15	↓	lua	0.5 %	-0.1 %

© Pierre Carboneille, 2015

## Populariteit van programmeertalen

Geen enkele pure functionele taal in de top 15....

# Functionele programmeertalen zijn lang academisch gebleven



Randall Munroe (s.d.). *Haskell*.  
XKCD. [xkcd.com/1312/](http://xkcd.com/1312/)

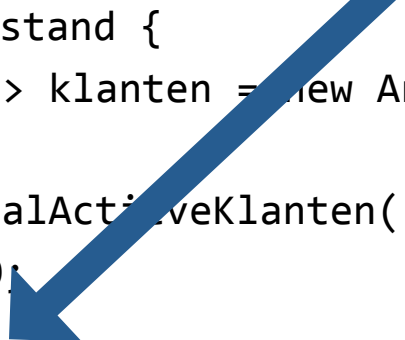
# Wat is functioneel programmeren/ een functionele programmeertaal?

Functioneel programmeren is een programmeerparadigma, dat:

- niet toestaat dat variabelen binnen een functie gewijzigd worden nadat ze een waarde hebben gekregen.

# Waar merk je dat als programmeur het eerste?

Niet mogelijk in een functionele programmeertaal



```
public class KlantenBestand {
    private List<Klant> klanten = new ArrayList<Klant>();

    public int telAantalActieveKlanten() {
        int teller = 0;

        for (int i = 0; i < klanten.size(); i++) {
            if (klanten.get(i).isActief()) {
                teller++;
            }
        }
        return teller;
    }
}
```



# Functioneel programmeren in Scala:

## map

```
object Upper {  
  def main(args: Array[String]) = {  
    args.map(_.toUpperCase()).foreach(printf("%s ",_))  
    println("")  
  }  
}
```

- Map voert een functie uit op alle elementen van een lijst.
- Map is een van dé manieren om met iteratie om te gaan...

Dean Wampler & Alex Payne (2009). *Programming Scala: Scalability = Functional Programming + Objects*. Oreilly, 2009, p.15.

# Functioneel programmeren in Scala:

## filter

```
object Dogs {  
  def main(args: Array[String]) = {  
    val dogBreeds = List("Doberman", "Yorkshire Terrier",  
                        "Dachshund", "Scottish Terrier",  
                        "Great Dane", "Portuguese Water Dog")  
    for ( breed <- dogBreeds  
          if breed.contains("Terrier")) // filter  
      println(breed)  
  }  
}
```

- Filter maakt een selectie van elementen uit een lijst.

Dean Wampler & Alex Payne (2009). *Programming Scala: Scalability = Functional Programming + Objects*. Oreilly, 2009, pp.59-60.

# Functioneel programmeren in Scala: folding

```
object folding {  
  def main(args: Array[String]) = {  
    println(List(1,2,3,4,5,6).reduceLeft(_ + _))  
    println(List(1,2,3,4,5,6).foldLeft(10)(_ * _))  
  }  
}
```

- Folding (reduce en fold) maakt het in functionele programmeertalen mogelijk om een lijst samen te voegen tot minder waarden.

Dean Wampler & Alex Payne (2009). *Programming Scala: Scalability = Functional Programming + Objects*. Oreilly, 2009, pp.59-60.

# Functioneel programmeren in Scala: hogere orde functies

```
var factor = 3
val multiplier = (i:Int) => i * factor

val l1 = List(1, 2, 3, 4, 5) map multiplier

factor = 5
val l2 = List(1, 2, 3, 4, 5) map multiplier

println(l1)    // List(3, 6, 9, 12, 15)
println(l2)    // List(5, 10, 15, 20, 25)
```

Dean Wampler & Alex Payne (2009). *Programming Scala: Scalability = Functional Programming + Objects*. Oreilly, 2009, p. 169.

Waarom is parallel verwerken in functionele programmeertalen eenvoudiger?

Parallel programmeren is complex in object-georiënteerde talen

- Tussen de threads van een programma moeten wijzigingen van gegevens gesynchroniseerd worden.
- Deze synchronisatie mechanismen zijn complex en foutgevoelig...

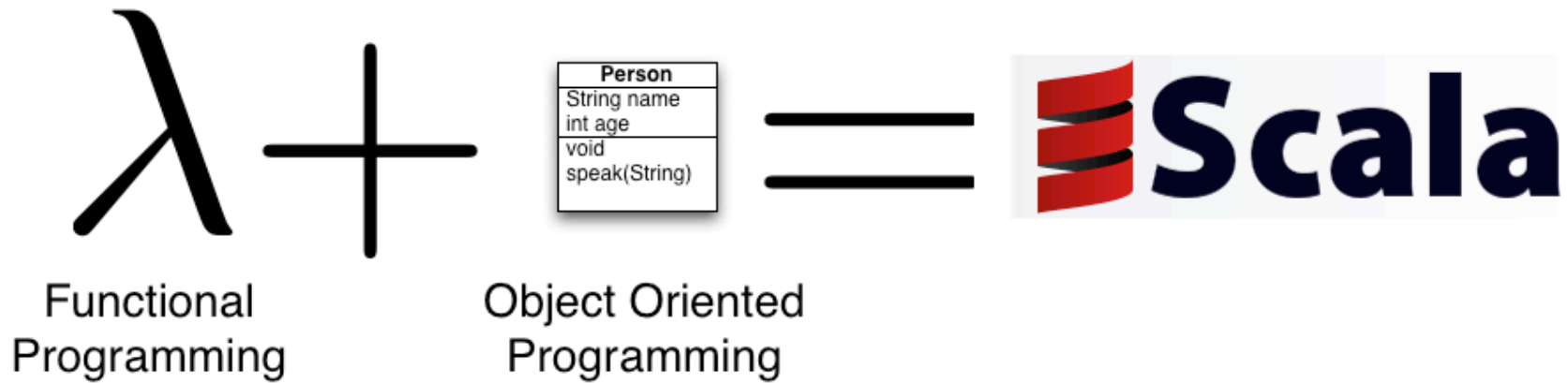
In functionele talen mogen variabelen niet wijzigen, waardoor parallel programmeren veel eenvoudiger gaat...

Dean Wampler (2011). *Functional Programming for Java Developers*. O'Reilly Media, 2011.

# Functionele principes worden ook in OOP-talen gebruikt

- Onveranderbare objecten (immutable objects)
  - Bekend voorbeeld: String's van Java
- Methodes die alleen een waarde teruggeven en geen side-effects hebben
  - Ontwerppatroon: Command-Query separation

# Scala



Richard Lee (2015). *Getting Started With Scala*. Lifuzu, 31 januari 2014. <http://lifuzu.com/blog/2015/01/31/getting-started-with-scala/>

# Waarom Scala

- Schaalbaarheid
- Compacte code
- Hergebruik van Java investeringen

# Waarom niet?

- Weinig programmeurs beschikbaar
- Leer curve
- Sluit niet aan bij bestaande ontwerpen





# Meer weten?

