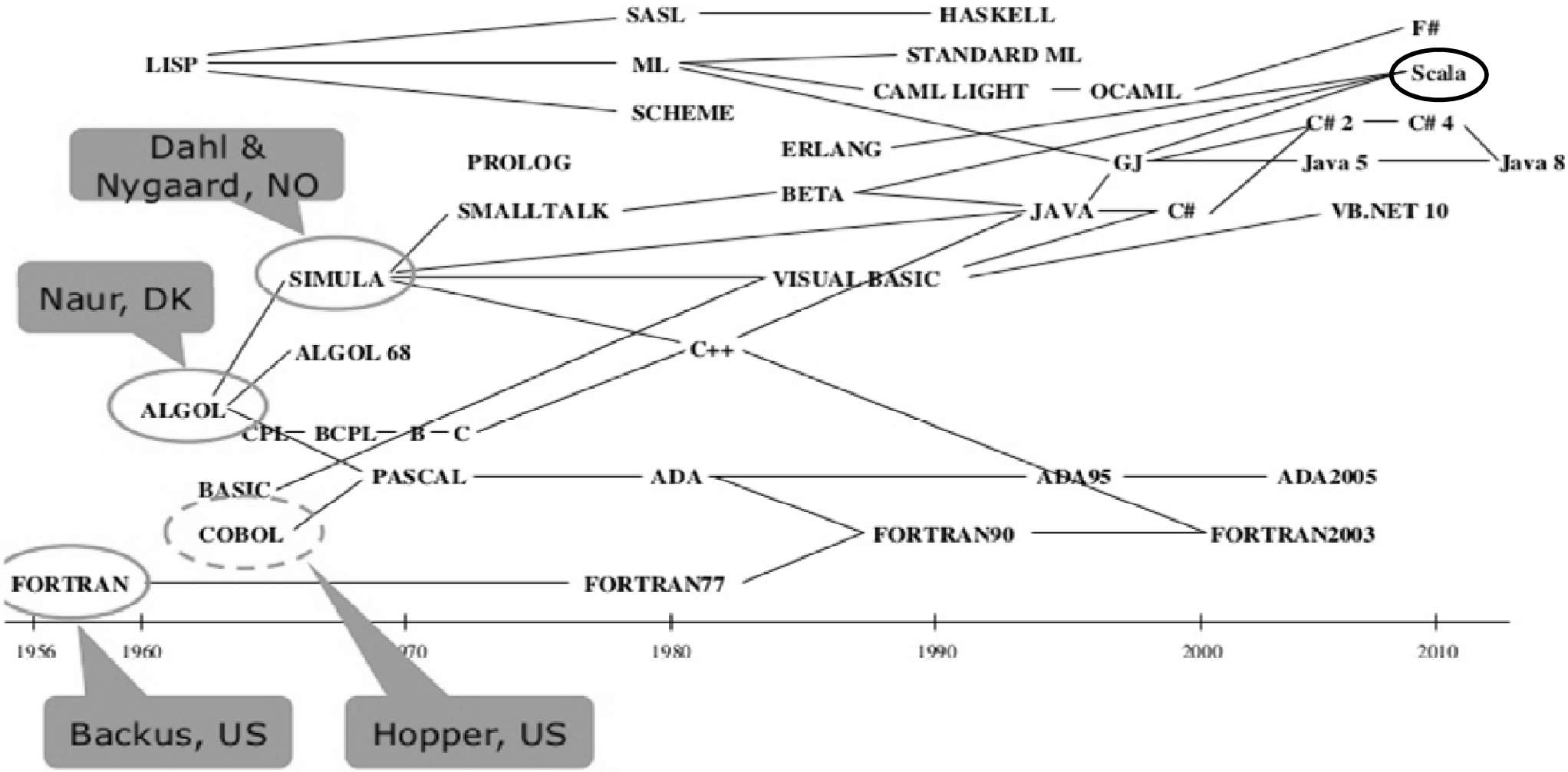
Funkcionalno programiranje

Uvod

Funkcionalno programiranje - uvod

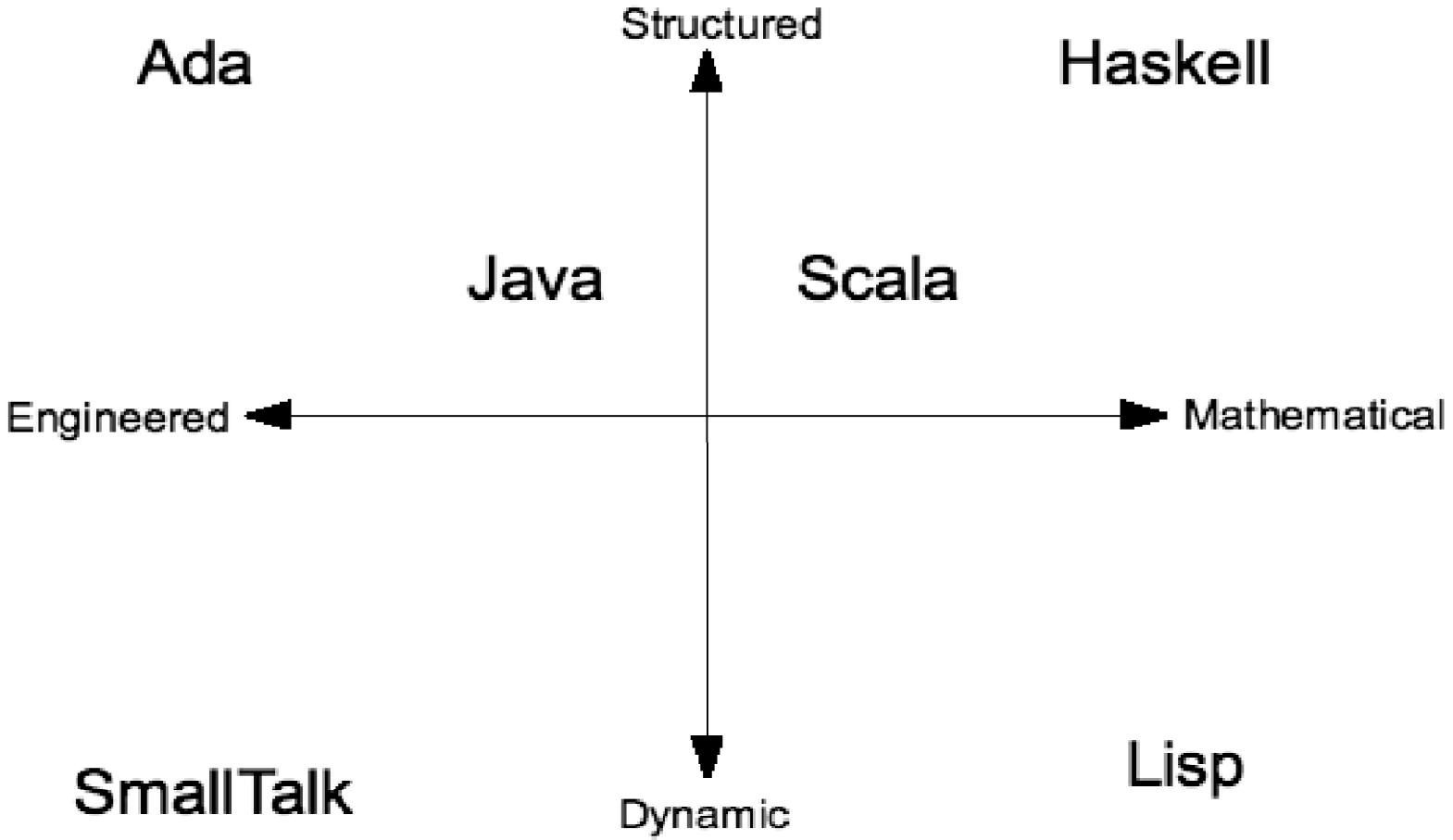
- Razlike u odnosu na druge paradigme u programiranju
  - nepromenljivi (nemutabilni) parametri funkcija
    - funkcije služe za **preslikavanje** ulaznih podataka u izlazne
  - funkcije su "građani prvog reda"
    - postoje funkcijski literali, promenljive
    - funkcije mogu biti prenete kao parametri drugih funkcija
    - funkcije mogu da vrate funkcije kao svoje rezultate
- Scala
  - objedinjuje koncepte OO, imperativ. i funkc. programiranja
  - "skalira" sa potrebama programera
    - može se koristiti za pisanje malih skripti
    - može se koristiti za pisanje složenih softverskih sistema

Genealoško stablo nekih programskih jezika



http://www.slideshare.net/InfinITnetvaerk/a-history-of-nordic-compilers-and-autocodes

Strukturiranost i namena jezika



http://erikengbrecht.blogspot.rs/2008/01/programming-language-continuum.html

Paradigme u programiranju

- Paradigma:
  - ono šta služi kao obrazac ili model
  - skup pretpostavki, koncepata, vrednosti – način razmišljanja
  - ►obrazac koji služi kao pravac razmišljanja u programiranju

- Glavne paradigme
  - imperativno programiranje
  - funkcionalno programiranje
  - logičko programiranje

- Ortogonalno na glavne paradigme
  - objektno-orijentisano programiranje

Imperativno programiranje

- Programira se upotrebom:
  - promenljivih (varijabli)
  - dodela
  - kontrolnih struktura (uslovna grananja, ciklusi, ...)

- Dva načina konceptualizacije programa
  - sekvenca instrukcija (za von Neumann-ovu mašinu)
    - programer navodi svaku instrukciju
  - relacija između predikata
    - uspostavljaju se logički uslovi nad podacima u vidu predikata, mat. formulacija
- Nedostaci:
  - loša skalabilnost programa
  - ograničenost principima koje je uveo von Neumann

Funkcionalno programiranje

- U užem smislu:
  - programiranje bez promenljivih, dodela ili kontrolnih struktura
- U širem smislu:
  - programiranje sa akcentom na upotrebi funkcija
- Funkcije postaju vrednosti koje se
  - proizvode
  - konzumiraju
  - sastavljaju
- Funkcionalan jezik pojednostavljuje ovakav pristup
  - funkcije mogu biti definisane unutar drugih funkcija
  - funkcije mogu biti parametri i rezultati drugih funkcija
  - postoje osnovne operacije koje omogućavaju sastavljanje funkcija

Poređenje Java/Scala (1/4)

Java

```
public class Person {
  public final String name;
  public final int age;
  Person(String name, int age) {
      this.name = name;
      this.age = age;
  }
}
```

Scala

```
class Person(val name: String,
             val age: Int)
```

Poređenje Java/Scala (2/4)

Java

```
Person[] people;
Person[] minors;
Person[] adults;
...
{
  ArrayList<Person> minorsList = new ArrayList<>();
  ArrayList<Person> adultsList = new ArrayList<>();
  for(int i = 0; i < people.length; i++)
    (people[i].age < 18 ? minorsList : adultsList)
                                              .add(people[i]);
  minors = minorsList.toArray(people);
  adults = adultsList.toArray(people);
}
```

Scala

```
val people: Array[Person]
val (minors, adults) = people partition (_.age < 18)
```

Poređenje Java/Scala (3/4)

Java

```
Person[] people;
Person[] minors;
Person[] adults;
...
{
  ArrayList<Person> minorsList = new ArrayList<>();
  ArrayList<Person> adultsList = new ArrayList<>();
  for(int i = 0; i < people.length; i++)
    (people[i].age < 18 ? minorsList : adultsList)
                                              .add(people[i]);
  minors = minorsList.toArray(people);
  adults = adultsList.toArray(people);
}
// Kako paralelizovati?
```

Scala

```
val people: Array[Person]
val (minors, adults) = people.par partition (_.age < 18)
```

Poređenje Java/Scala (4/4)

- Novije verzije Jave su donele koncepte iz funkcionalnog programiranja
  - lambda funkcije
  - tokove (stream) za podršku paralelnoj obradi
- To ukazuje na trendove povećanog interesa za FP
- Međutim
  - sam jezik Java je i dalje veoma eksplicitan
  - nije koncipiran oko ideje funkcionalnog programiranja
  - nije proširljiv tako da proširenja deluju kao deo osnovnog jezika

Scala

- Jezik razvijen u EPFL (Martin Odersky)
  - Poseduje sve konstrukte funkcionalnih jezika
  - Statičko tipiziranje
  - Objektno-orijentisan jezik
  - Funkcioniše na JVM, interoperabilnost sa Javom
  - Adaptiran za simbolički račun
  - Adaptiran za paralelno izvršavanje
- Kao i svaki netrivijalni jezik, poseduje:
  - primitivne izraze – najjednostavnije entitete kojima se manipuliše
  - sredstvo kombinovanja – za sastavljanje složenih elemenata od jednostavnih
  - sredstvo apstrakcije – za imenovanje elemenata i njihovo upravljanje kao da su osnovni

Prednosti FP

- Manje pisanja koda
  - jednostavnije za razumevanje (ali zahteva veliko predznanje)
  - jednostavnije za održavanje, povećava produktivnost
- Nema bočnih efekata
  - podaci "ulaze" u funkcije, ali ih funkcije ne menjaju
  - jednostavnije za razumevanje
  - reupotrebljivost koda
- Rekurzija
  - prirodna kontrolna struktura u funkcionalnim jezicima
  - alternativa za cikluse u imperativnim jezicima

Mane FP

- Ulaz i izlaz
  - prirodna je protočna obrada, ne odgovara stilu FP
- Složeniji razvoj interaktivnih aplikacija
  - zasnivaju se na ciklusima zahtev/odgovor
  - inicira korisnik
- Programi koji se izvršavaju u ciklusu su složeniji za razvoj
- Manja efikasnost na modernom hardveru
  - FP je dominantno bio zastupljen u akademiji, gde performanse nisu bile od velikog značaja
- Nije orijentisano ka podacima
  - dohvatanje, manipulisanja i vraćanje podataka (baze podataka)
  - OO pristup je prirodniji

Radno okruženje

- REPL (Read-Eval-Print-Loop)
- IDE (ScalaIDE, Eclipse, NetBeans, IntelliJ)
- http://www.scala-lang.org