Paradigmi e Linguaggi di Programmazione

Programma & Obiettivo

- Introduzione ai linguaggi formali
  - Espressioni regolari,Analisi sintattica e generatori di parser
- Cenni di Natural Language Processing
- Programmazione ad oggetti
  - E confronto tra vari linguaggi: Java, Python, Go
- Programmazione generica
  - Template C++ e STL
- Programmazione funzionale
  - LISP, Haskell, Caratteristiche funzionali del C++
- Linguaggi dinamici e metaprogrammazione
  - Caratteristiche dinamiche del C++, Linguaggi dinamici e di scripting, metaprogrammazione in Python

➢ ***Obiettivo:*** Sviluppo multi-paradigma, adatto al contesto di sistemi distribuiti, eterogenei e paralleli

Docenti & Riferimenti

- Agostino Poggi

- Michele Tomaiuolo
- Monica Mordonini

- Gianfranco Lombardo

Michele

- Phone: +39 0521 90 5708
- michele.tomaioulo@unipr.it

Monica

- Phone: +39 0521 90 5735
- monica.mordonini@unipr.it

Bibliografia

- *Slide del corso*
  - http://sowide.ce.unipr.it/teaching/linguaggi/
  - https://elly.dia.unipr.it/2019/course/view.php?id=497

- M. Gabbrielli, S. Martini (2011). Linguaggi di programmazione - Principi e paradigmi.
- M. Lipovaca (2012). Learn You a Haskell for Great Good.

http://learnyouahaskell.com/

- S. B. Lippman, J. Lajole, B. E. Moo (2013). C++ Primer.

Modalità d'esame

1. *Progetto,* su aspetti o tecnologie legati al corso*:*

   - Se il lavoro è completato entro la fine del corso, presentazione alla classe
   - Possibilità di lavorare in coppia
   - Possibilità di integrare il progetto con l'insegnamento di Sistemi Distribuiti o Intelligenza Artificiale

2. Verifica sugli argomenti trattati a lezione

   - Con esame finale scritto, oppure...
   - Con la partecipazione assidua a lezioni ed esercitazioni

Organizzazione del corso

- Lezioni frontali
- Esercitazioni
- Presentazioni finali

Linguaggi di programmazione

Introduzione

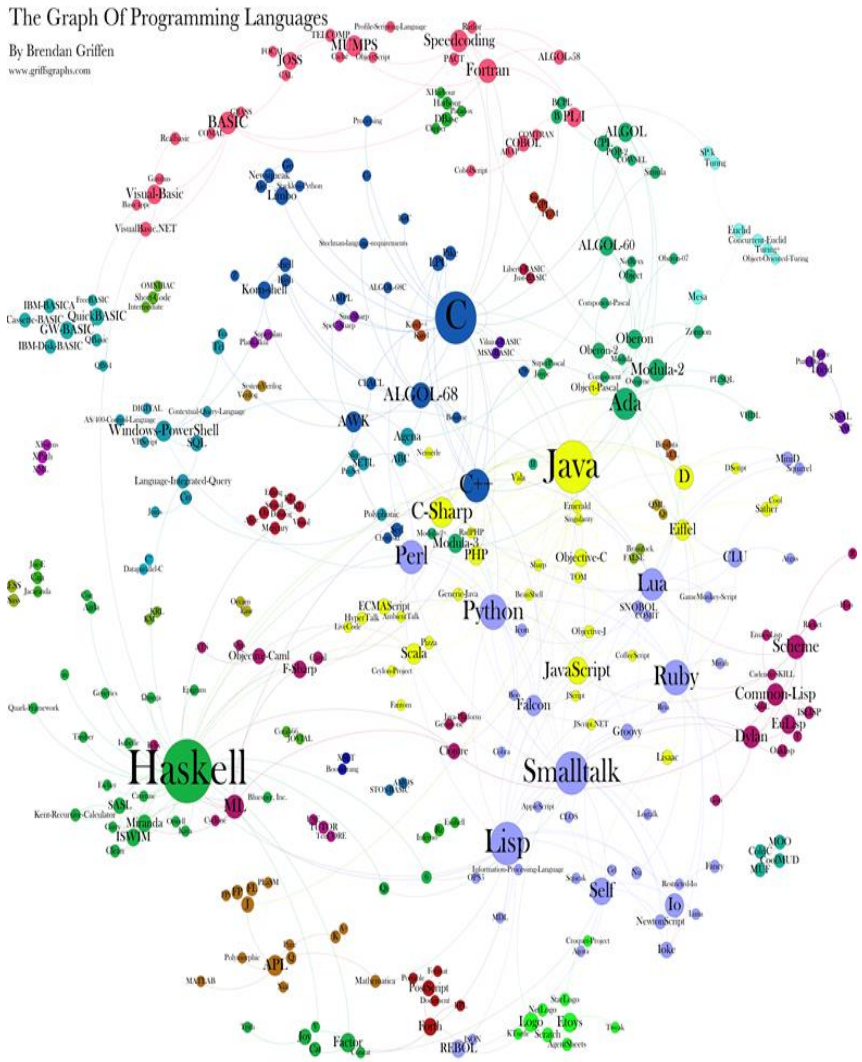
Linguaggi di programmazione

- ***Perché esistono tanti linguaggi, oltre ad un fatto "storico"?***
- ***Qual è il linguaggio di programmazione migliore?***

Linguaggi di programmazione

Teoricamente si è dimostrato che tutti i linguaggi non banali sono, dal punto di vista della *capacità computazionale*, equivalenti.

- *In pratica* un programma scritto in un certo linguaggio può sempre essere codificato in qualsiasi altro linguaggio
- ovviamente la valutazione astratta della capacità computazionale non tiene conto dei molteplici fattori accessori, spesso dipendenti dall'hardware



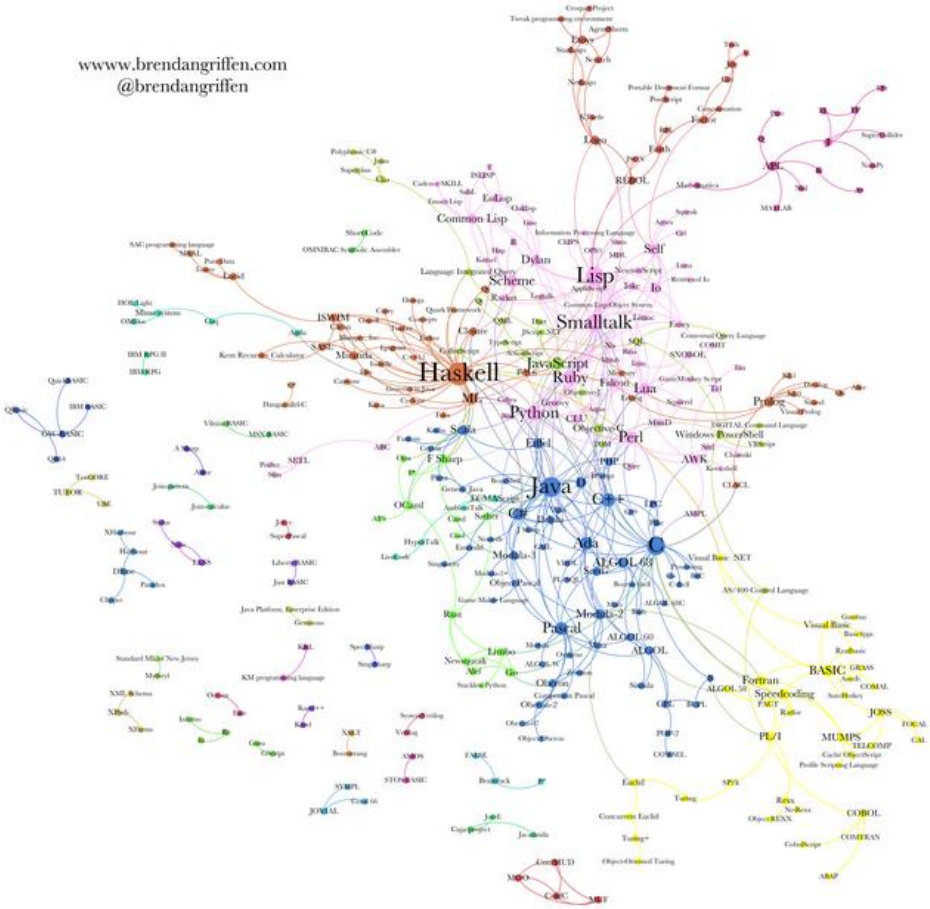
Linguaggi di programmazione

- Per ottenere il grafico si sono recuperati i dati dall'ontologia sui linguaggi di programmazione recuperabile in JSON a:
- http://dbpedia.org/ontology/ProgrammingLanguage
- Con una query del tipo:

```
SELECT *
WHERE { ?p a <http://dbpedia.org/ontology/ProgrammingLanguage
?p <http://dbpedia.org/ontology/influenced> ?influenced . }
```

- Per visualizzarsi si sono utilizzati script in python e le librerie di Gephi.

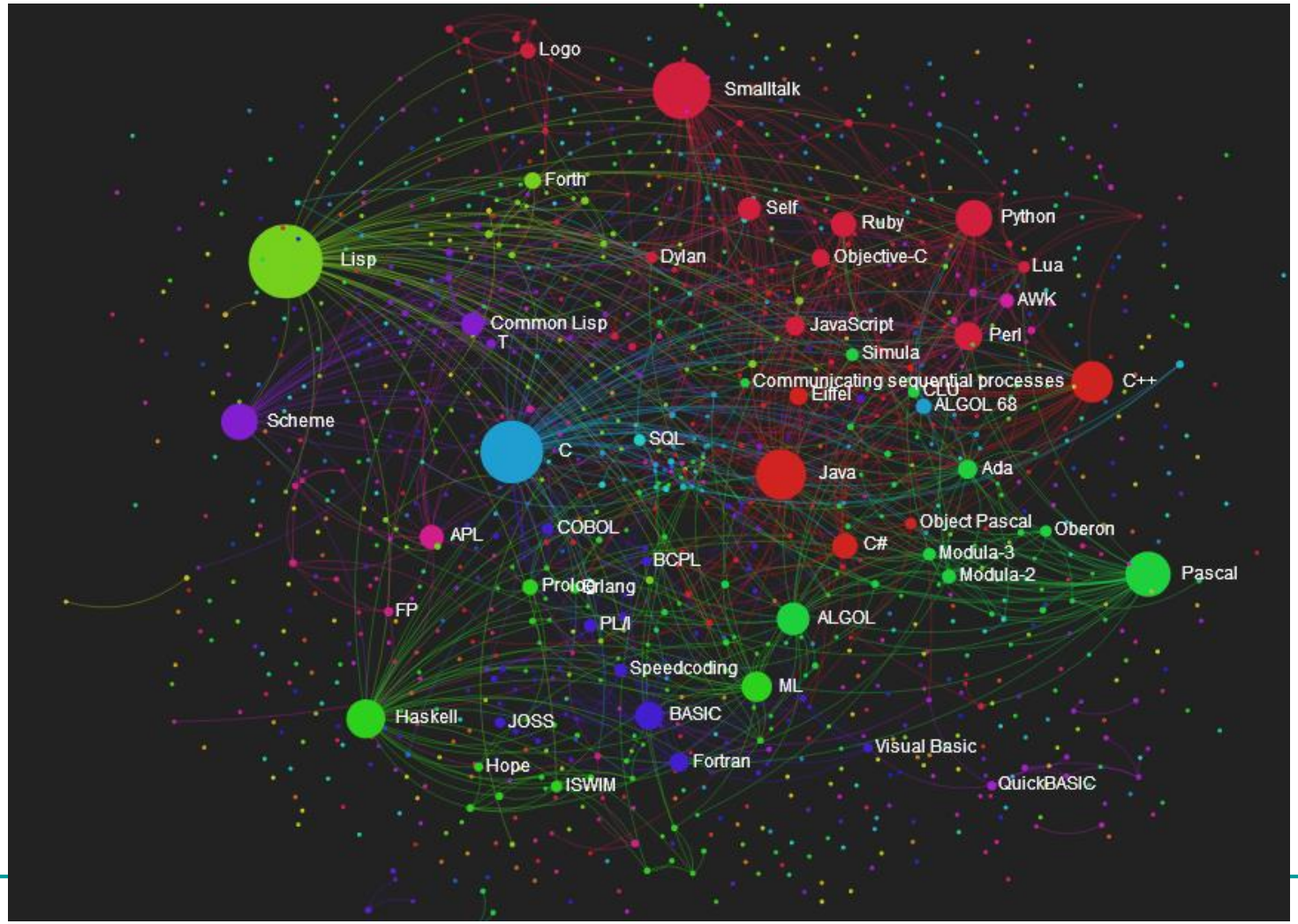
- Gli archi rappresentano se uno dei due linguaggi ha avuto influenza sull'altro
- Nodi grandi significa che quel linguaggio ha influenzato molti dei linguaggi presenti nel grafo



http://brendangriffen.com/gow-programming-languages/

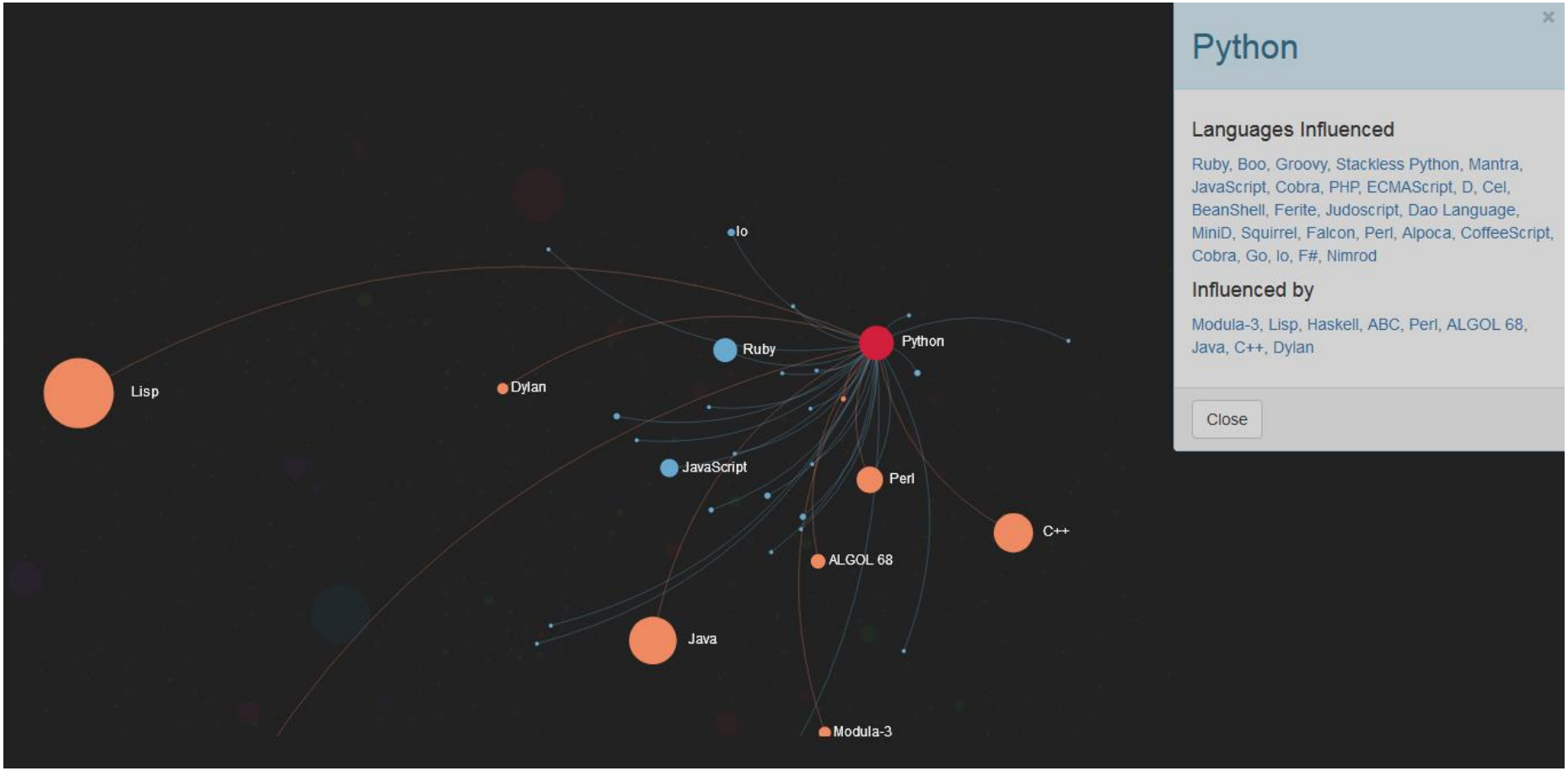
Linguaggi di programmazione

- https://exploring-data.com/vis/programming-languages-influence-network/

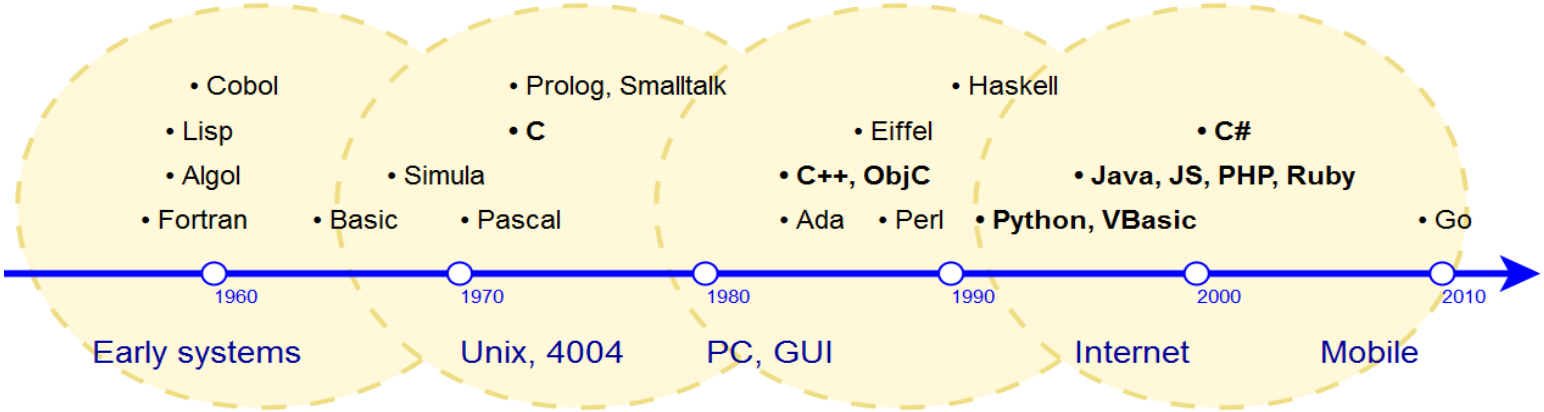


Linguaggi di programmazione

- https://exploring-data.com/vis/programming-languages-influence-network/



Storia dei linguaggi di Programmazione



Anni ’30:

- *il lambda-calcolo*

• Linguaggio Turing-equivalente

Fino anni ’60:

- *linguaggi assembly*

• Linguaggi direttamente eseguibile da H/W dedicato

• Schede perforate, architetture batch

Anni ‘60-’80:

- *linguaggi ad alto livello*

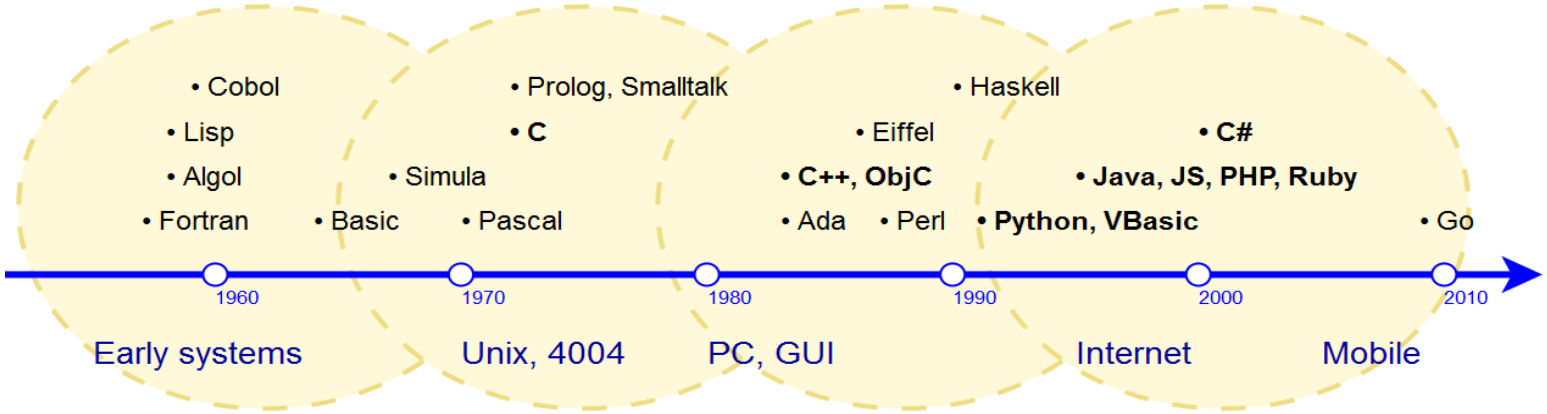
• Rappresentazione di strutture dati e di controllo più generali e più vicine ai modi umani

- ***Funzionali***: Lisp (1965)

- ***Logici***: Prolog(1972)

- ***Imperativi***: Fortran (1957), Basic (1964), Cobol (1961)

Storia dei linguaggi di Programmazione



Anni '60-'80:

- *linguaggi ad alto livello*

• Imperativi strutturati e con typing

- ***Pascal (1973)***

TeX e buona parte delle prime versioni del sistema operativo del Macintosh e di Windows furono scritte in Pascal.

- ***C (ansi C-1983)***

Utilizzato per riscrivere la maggior parte del codice del sistema UNIX

Anni '80-90 in poi :

- *Linguaggi ad oggetti*

• Smalltalk (1983), C++ (1999), Java (1995)

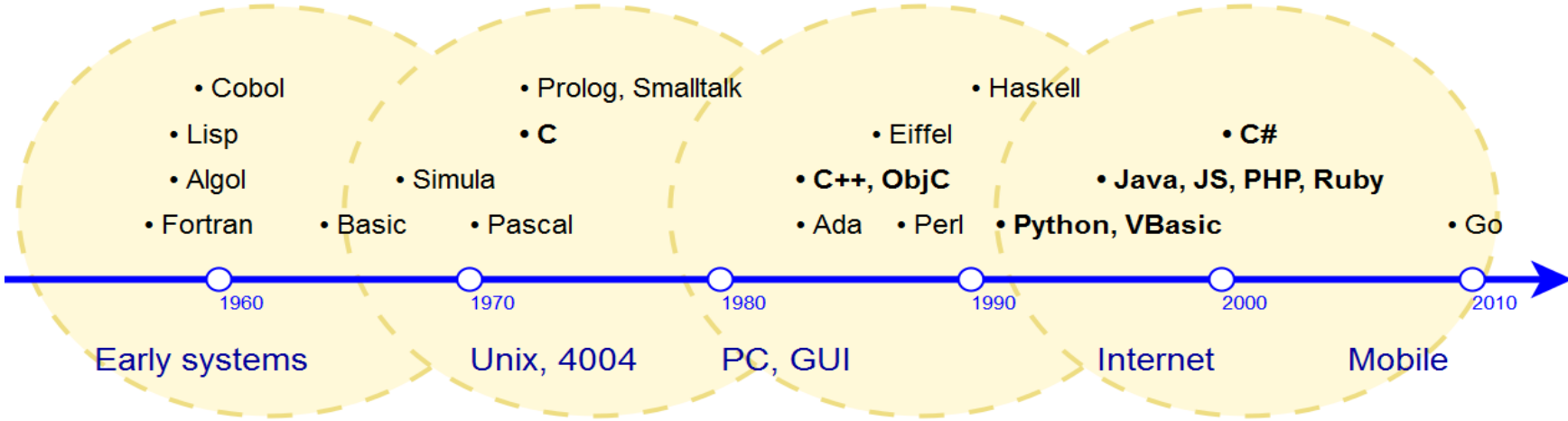
- *Linguaggi di scripting*

• *Javascript 1997), PhP (1994) Perl (1987)*

- *Linguaggi multi-paradigma*

• *Python (1991)*

Storia dei linguaggi di Programmazione

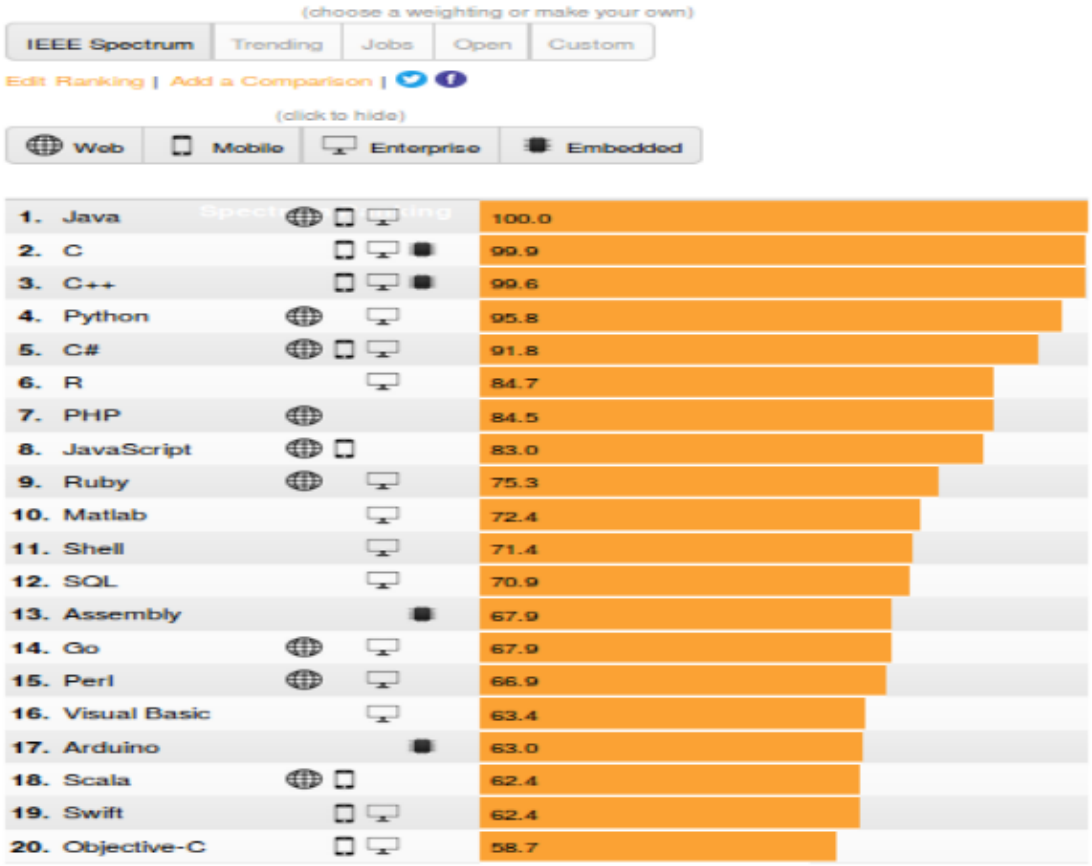


- http://www.oreilly.com/news/graphics/prog_lang_poster.pdf
  http://www.levenez.com/lang/history.html
  http://www.cs.brown.edu/~adf/programming_languages.html

Linguaggi di programmazione

The Top 20 (IEEE Spectrum, 2015)

Oltre al fatto storico



http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015

Linguaggi di programmazione

Oltre al fatto storico

- Il linguaggio deve
  - essere non troppo difficile
  - permettere la creazione di applicazioni in tempi ragionevoli
  - produrre eseguibili efficienti

- Ogni linguaggio di programmazione risulta più o meno adatto rispetto al tipo di applicazione
  - Lisp meno efficiente del C ma adatto a manipolazioni simboliche (IA)

Linguaggi di programmazione

Un linguaggio di programmazione va scelto in base

- all'area applicativa
  - scientifico, gestionale, elaborazione testi, simulazioni, ecc.
- alle caratteristiche del problema da risolvere.

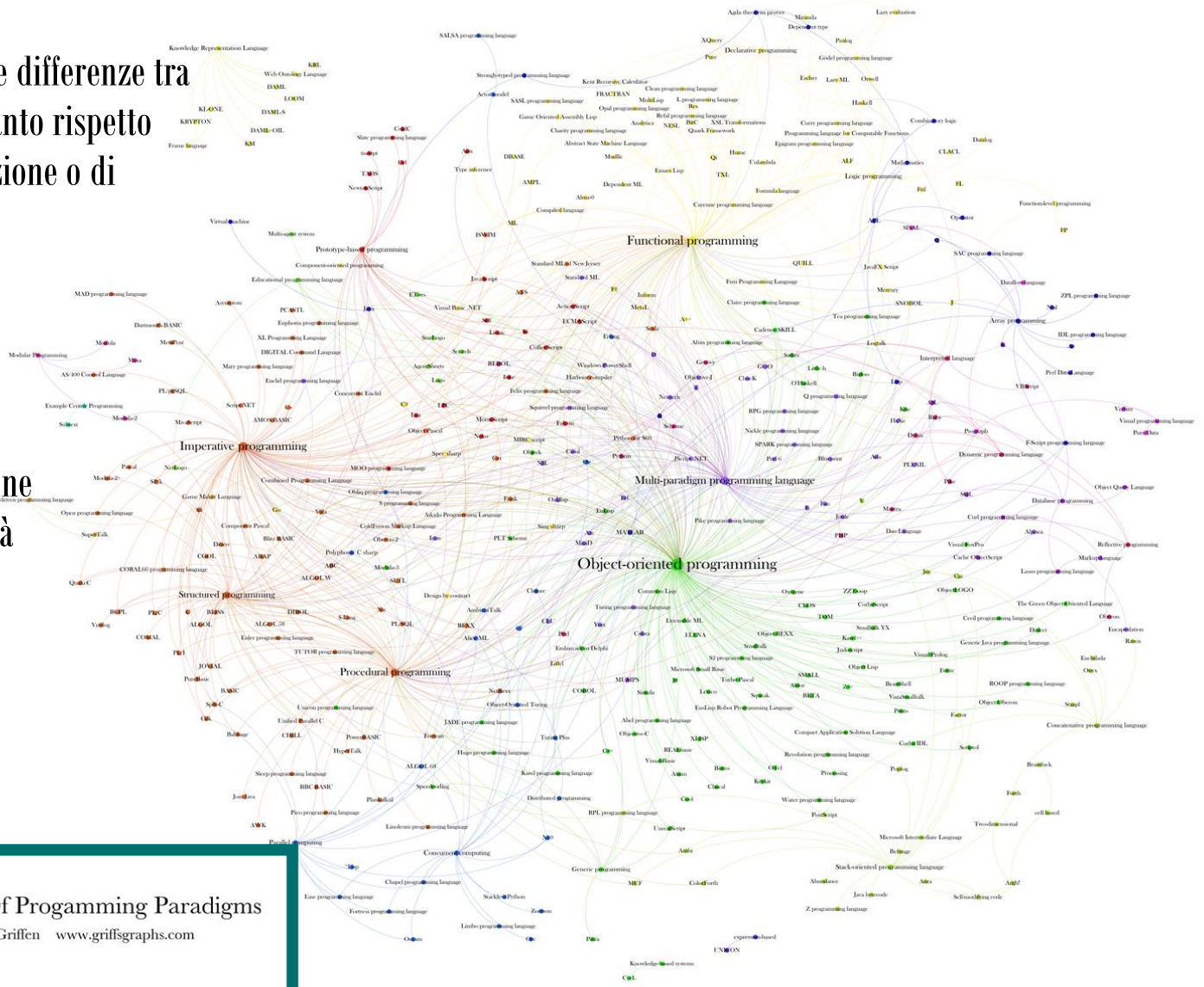
Analogia *linguaggi naturali:*

Carlo V “Parlo spagnolo a Dio, italiano alle donne, francese agli uomini e tedesco al mio cavallo”.

Linguaggi di programmazione

Si deve comprendere le differenze tra i vari linguaggi, non tanto rispetto alle differenze di notazione o di terminologia

quanto rispetto alla filosofia e al modello computazionale che li contraddistingue e che ne determina la personalità

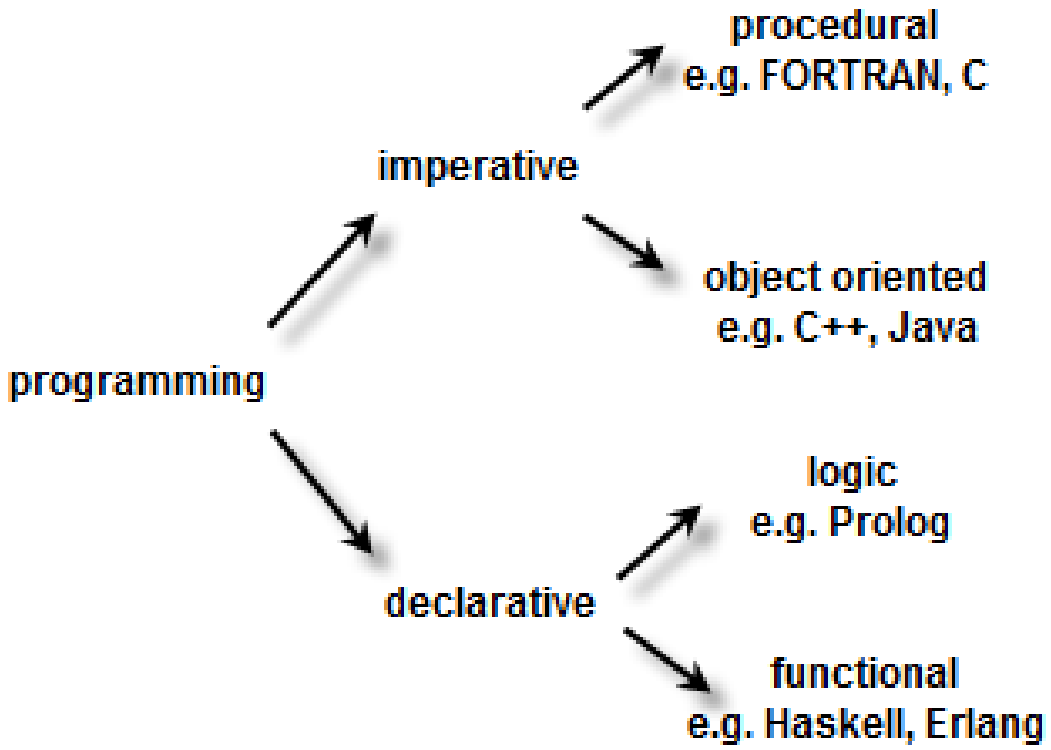


The Graph Of Progamming Paradigms

By Brendan Griffen www.griffsgraphs.com

Linguaggi di programmazione

Prima classificazione



Linguaggi di programmazione

Imperativi

- il programma è costituito da una sequenza di istruzioni il cui effetto è quello di modificare il contenuto della memoria dell'elaboratore o di determinare le modalità di esecuzione di altre istruzioni;
- in questo modello assume un ruolo fondamentale *l'istruzione di assegnazione*.
  - Sono imperativi la maggior parte dei linguaggi più diffusi (Pascal, Basic, Fortran, C, Cobol, ecc.).

Linguaggi di programmazione

Imperativi

- Adottano uno ***stile prescrittivo***, ossia è prescritto l'insieme delle istruzioni e l'ordine in cui queste devono essere eseguite

PROGRAMMA = ALGORITMO + DATI

- Parte dichiarativa in cui si dichiarano tutte le variabili del programma ed il loro tipo di dato (DATI)
- Parte di istruzioni che descrivono l'algoritmo risolutivo (ALGORITMO).

Linguaggi di programmazione

Strutturati

- man mano che l'*ars programmandi* si sviluppava (e si potenziava l'hardware) si sono individuate delle metodologie appropriate.

- La ***programmazione strutturata*** è una tecnica il cui scopo è di semplificare la struttura dei programmi, limitando l'uso delle strutture di controllo a pochi casi semplici, tutti con un solo ingresso e una sola uscita.

- Oltre alla programmazione strutturata secondo il paradigma classico imperativo si è sviluppata la *programmazione ad oggetti*

Linguaggi di programmazione

Orientati ad oggetti

- il programma è considerato l'effetto dell'interazione di un insieme di oggetti (insiemi di dati e algoritmi che manipolano questi dati) che comunicano con l'esterno mediante messaggi.
- Assumono rilevanza concetti quali incapsulamento, ereditarietà e polimorfismo.
  - Oltre a linguaggi specializzati (Smalltalk), sono nate delle estensioni dei linguaggi già esistenti (ad es. C++ per il C, CLOS per il Lisp).

Linguaggi di programmazione

Dichiarativi (o logici)

- il programma è considerato come la dimostrazione della verità di una asserzione; il sorgente è costituito da una sequenza di asserzioni di fatti e regole.
- Non è necessario indicare esplicitamente il flusso di esecuzione, ma dato un obiettivo di partenza (il *goal*) è il sistema che cerca di individuare i fatti e le regole rilevanti.
- In tale ricerca assumono importanza meccanismi quali il *pattern matching* e il *backtracking*.
- I linguaggi logici risultano particolarmente adatti a risolvere problemi che riguardano *entità* e le loro *relazioni*.
  - il Prolog (PROgramming in LOGic)

Linguaggi di programmazione

Dichiarativi (o logici)

- Il programma può essere considerato come la dimostrazione della verità di una asserzione.

- Un programma Dichiarativo è :

PROGRAMMA = CONOSCENZA + CONTROLLO

- CONOSCENZA : un insieme di fatti e regole
- CONTROLLO : un insieme di regole

Linguaggi di programmazione

Funzionali

- il programma è considerato come il calcolo del valore di una funzione
- Una funzione è una regola di corrispondenza che associa ad ogni elemento del suo dominio un unico elemento del suo codominio.
- La valutazione di una funzione consiste nell'applicazione della regola di associazione all'elemento del dominio per produrre un risultato (valore del Codominio).
- Il primo rappresentante di questa categoria è il Lisp (LISt Processing).

Linguaggi di programmazione

Funzionali

- Quindi in un linguaggio funzionale è:

PROGRAMMA = FUNZIONE

- ESECUZIONE DI UN PROGRAMMA = VALUTAZIONE DI UNA FUNZIONE
- In un linguaggio funzionale *puro* le variabili sono variabili matematiche (denotano un valore non modificabile)
  - non esiste infatti l'operazione di assegnazione : si utilizza soltanto il passaggio dei parametri.
- Rivestono particolare importanza la ***ricorsione*** e, come struttura dati, la ***lista*** (sequenza ordinata di elementi).

Linguaggi di programmazione

Altri Linguaggi

- **Paralleli:** vi sono dei meccanismi espliciti per indicare compiti che possono essere effettuati in parallelo.

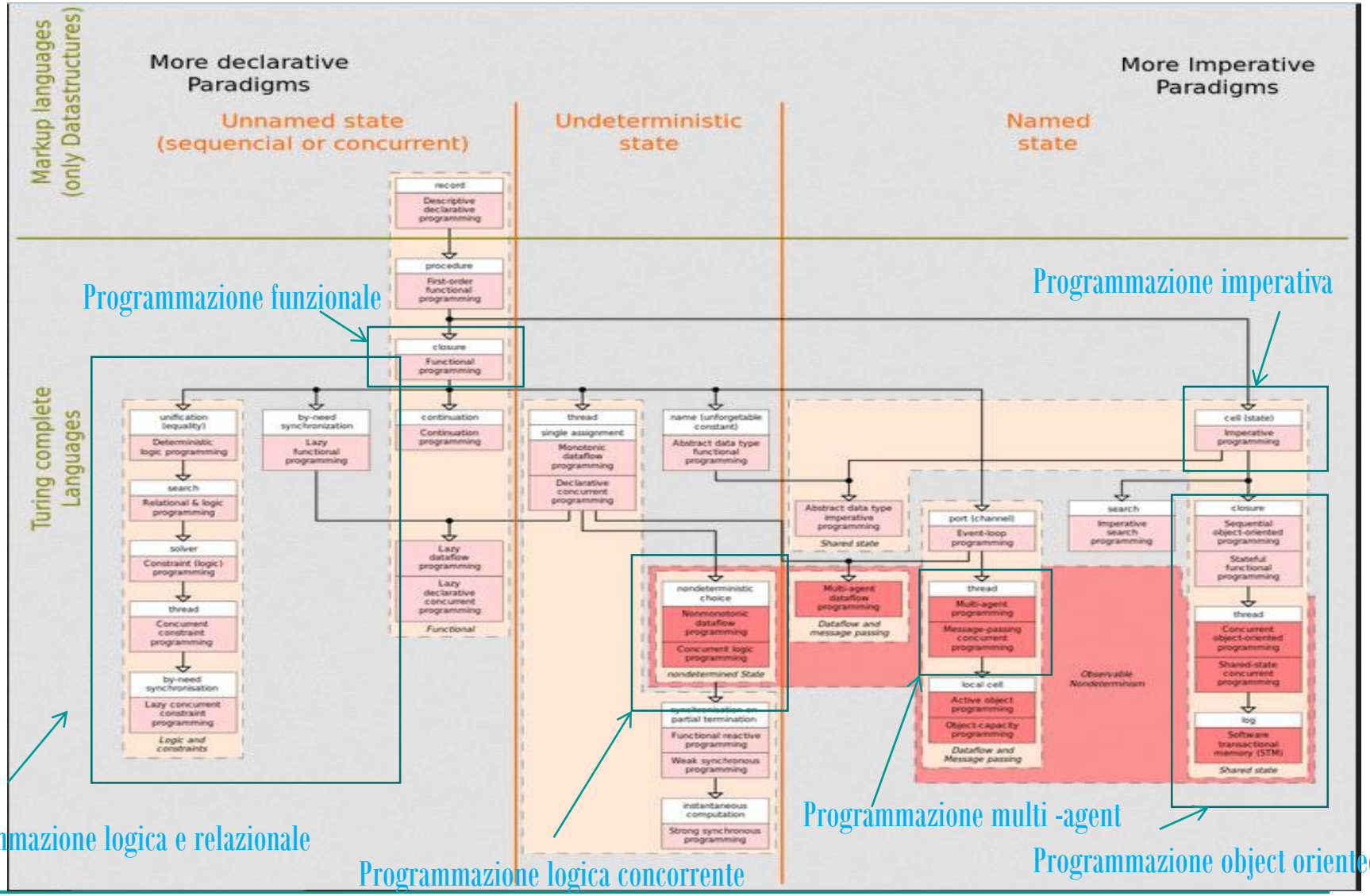
- **Event driven:** in un ambiente event driven non esiste più una sequenza determinata di comandi da eseguire ma una serie di reazioni che il sistema ha rispondendo a determinati stimoli esterni o interni. La programmazione è facilitata da un linguaggio che supporta tale modello.

Linguaggi di programmazione

Altri Linguaggi

- **Atipici:** es: i fogli elettronici possono essere considerati linguaggi di programmazione in cui, in una certa misura, le relazioni temporali sono sostituite da relazioni spaziali (il valore di una cella dipende dal valore di altre)
- **Linguaggi visuali:** utilizzano costrutti e grammatiche di tipo grafico o iconico
  - LabVIEW, Simulink - comunità scientifiche, Scratch - ragazzi
- **Scripting:** tipizzazione dinamica, principio DRY - Don't Repeat Yourself
  - JavaScript, PHP - internet, Python, Ruby - mondo Object Oriented

Overview of programming paradigms according to Peter Van Roy



https://commons.wikimedia.org/wiki/File:Programming_paradigms.svg

Valutazione dei Linguaggi di Programmazione

- Dato che non ha senso dire che in generale, un linguaggio è migliore di un altro
- è possibile però individuare ed analizzare le caratteristiche che entrano in gioco nella scelta del linguaggio da utilizzare in una determinata situazione
  - *Entrano in gioco fattori <u>che non sono solo</u> legati al paradigma*

Valutazione dei Linguaggi di Programmazione

- Suddividiamo in
  - fattori ***intrinseci***:

  legati strettamente al linguaggio
  - fattori ***ambientali****:*

  legati a come il linguaggio è entrato a fare parte del sistema per cui sono legati anche ad altre variabili non direttamente dipendenti dal linguaggio
    - potenza degli elaboratori disponibili, presenza di librerie, diffusione raggiunta, ecc.

  e possono quindi anche variare con il passare del tempo anche se il linguaggio non viene modificato.

Fattori intrinseci

- ***Espressività***

√ facilità con cui è possibile esprimere la soluzione ad un dato problema.

- Determina la velocità con cui possono essere risolti i problemi mediante l'uso di un linguaggio, la quantità di codice necessaria e persino la qualità del risultato finale.
- La valutazione dell'espressività di un linguaggio dipende dal tipo di problema in esame e da quanto il linguaggio sia congeniale al suo utilizzatore tipo.

- **Es**: scrivere in Lisp un programma che calcoli la derivata simbolica di una espressione matematica, richiede solo 20/30 righe di codice.
- Usare il C comporterebbe una quantità di codice maggiore e un maggiore sforzo da parte del programmatore, ottenendo comunque un risultato, probabilmente più efficiente, ma di qualità minore in termini di leggibilità e modificabilità del codice

Fattori intrinseci

- ***Facilità di apprendimento***

- complessità del linguaggio,
- tipo di background culturale necessario al suo apprendimento
- e anche un fattore esterno: la presenza di buoni manuali

- ***Qualità pedagogiche***:

- indica l'attitudine del linguaggio ad essere utilizzato come strumento per l'insegnamento della programmazione.

- Es. Pascal e Prolog perché imparandoli si acquisiscono tecniche di risoluzione di problemi efficacemente utilizzabili, in determinate situazioni, in altri linguaggi di programmazione più diffusi.

Fattori intrinseci

- ***Leggibilità***

- è in relazione con la facilità con cui persone estranee ad un progetto possono leggere e comprendere il relativo programma.

- La leggibilità di un linguaggio non è assoluta ma dipende anche
  - dal tipo di problema affrontato
  - dalla lunghezza del codice utilizzato per la sua soluzione
  - dallo stile di programmazione del progettista.

Fattori intrinseci

- ***Facilità di manutenzione***

- possibilità modificare un programma per nuove esigenze o per eliminare problemi.

  - *Modularità*: la capacità del linguaggio di favorire suddivisione di un programma in parti di codice quanto più indipendenti l'uno d'altro(principio del *divide et impera*).

  - I*nformation hiding:* scatole nere di cui all'esterno si può accedere solo attraverso un ben determinato protocollo.

Fattori intrinseci

- ***Plasmabilità:***
  - la possibilità di poter modificare o estendere la sintassi del linguaggio . I linguaggi ad oggetti presentano in genere un notevole grado di plasmabilità.
    - Ad es. il C++ permette l'overloading degli operatori.
- ***Generalità:***
  - il grado con cui il linguaggio può essere utilizzato in diversi campi di applicazione.
    - Es. contro: Cobol difficilmente utilizzato al di fuori dall'area dei gestionali.

Fattori intrinseci

- ***Ben definizione***:
  - la sintassi e la semantica del linguaggio devono essere privi di ambiguità e avere una coerenza interna.

- ***Efficienza:***
  - capacità del linguaggio di creare applicazioni efficienti sia in termini di memoria che di velocità.
  - I linguaggi più astratti tendono a produrre codice più inefficienti
    - Anche perché, proprio per la loro astrazione, risulta più semplice scrivere programmi inefficienti con essi.

Fattori Ambientali

- ***Diffusione***
  - maggiore disponibilità di documentazione, librerie, strumenti di sviluppo, piattaforme su cui è implementato,
  - maggiore ricerca nello sviluppo compilatori efficienti
    - Generalmente sono correlati

- ***Integrazione***
  - la possibilità di chiamare codice scritto in altri linguaggi e viceversa.

Fattori Ambientali

- ***Portabilità***

  - Strettamente legato alle piattaforme hw/sw su cui è implementato.

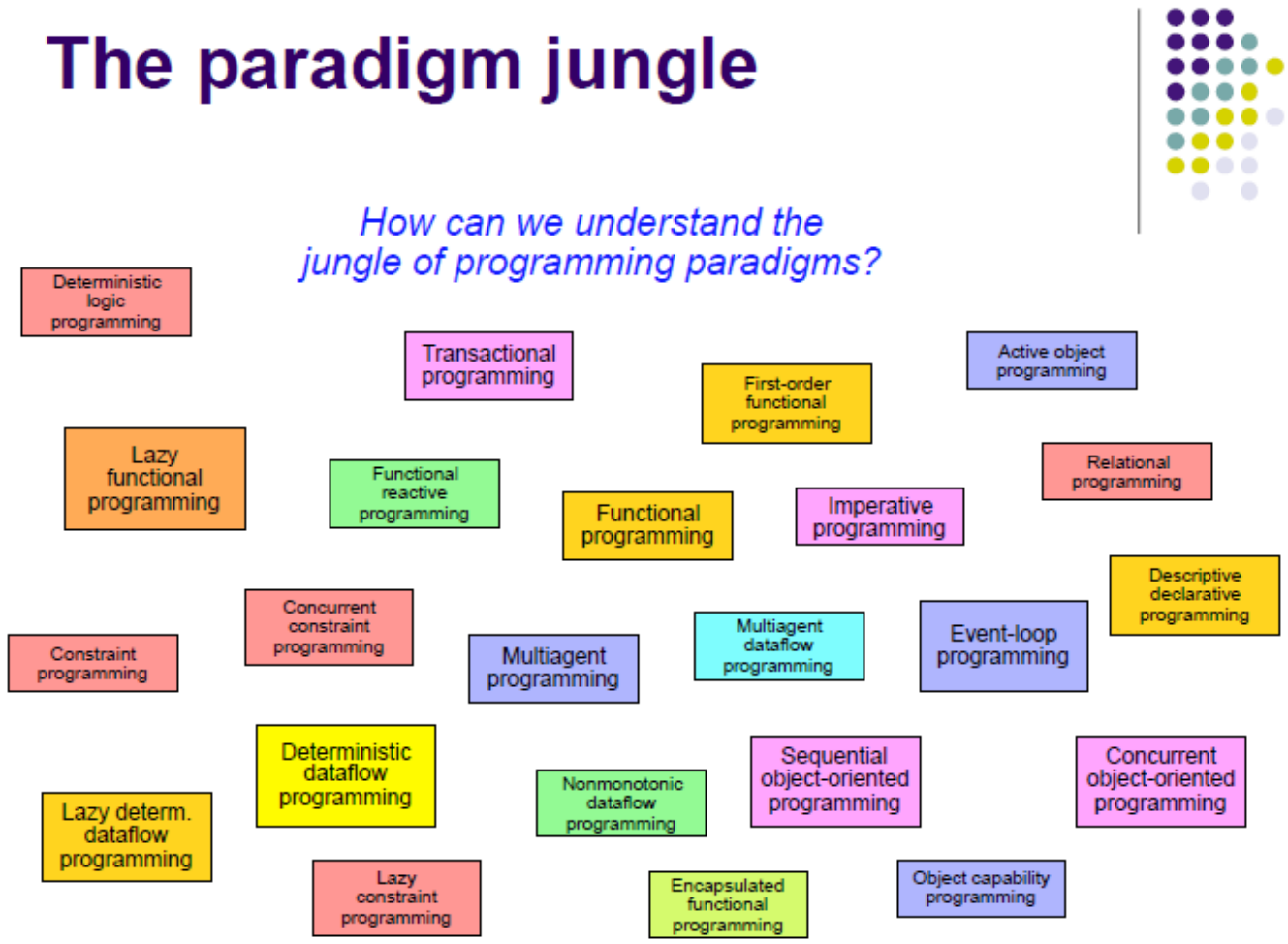
    - Spesso è determinante la presenza di una standardizzazione ufficiale

- ***Grado di standardizzazione***

  - Di uno stesso linguaggio possono esistere più versioni tra loro incompatibili anche per piccoli particolari.

    - per poter utilizzare un programma scritto in un altro dialetto, è necessario apportarvi delle modifiche: a volte lievi, a volte sostanziali

E allora che linguaggio scegliere?



https://studio.edx.org/c4x/LouvainX/Louv1.01x/asset/13-2paradigmsFigure.pdf

E allora che linguaggio scegliere?

- Risolvere un problema di programmazione richiede di identificare i giusti concetti

- Un ***paradigma di computazione*** è un modello che si basa su una teoria matematica o su un insieme coerente di concetti di programmazione
  - Esempi di programming concept: procedura, cella di memoria, thread …

- Un paradigma che ha un certo insieme di concetti può essere quindi quello che supporta al meglio la risoluzione di alcuni tipi di problema e non altri.

Linguaggi, paradigmi e concetti

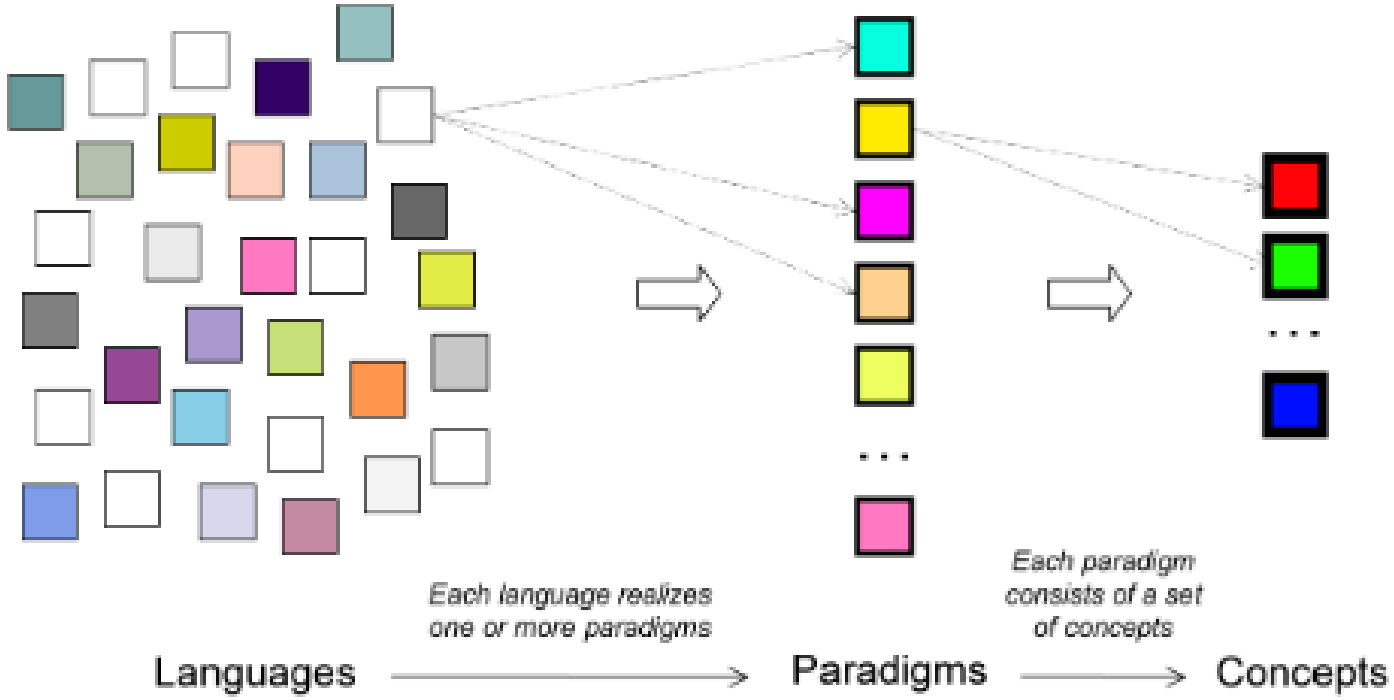
- Poiché esiste un numero minore di paradigmi di computazione che di linguaggi programmazione, è interessante focalizzare l'attenzione su questi.
  - Java, Javascript, C#, Ruby, and Python sono virtualmente identici implementando tutti il paradigma object-oriented

- Due linguaggi che hanno lo stesso paradigma possono però essere «amati» dai programmatori in modo diverso
  - Questo perché operano scelte diverse sulle tecniche pratiche e gli stili della programmazione

Programming Paradigms for Dummies: What Every Programmer Should Know
P Van Roy - New computational paradigms for computer music, 2009

https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

Linguaggi, paradigmi e concetti



- ✓ Ogni linguaggio di programmazione implementa uno o più paradigmi
- ✓ Ogni paradigma è definito da un insieme di concetti di programmazione, che rappresenta il “paradigm's kernel language”

https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

Linguaggi, paradigmi e concetti

- Sono meno dei linguaggi ma anche i paradigmi sono diversi
  - Nell' articolo sono citati 27 paradigmi diversi tutti con buone implemenatzioni e applicazioni pratiche

- Fortunatamente i paradigmi non sono isole ed hanno molto in comune.
  - E' possibile oragnizzarli in una ***tassonomia*** che mostra come questi sono correlati e quindi può orientare nella scelta del linguaggio giusto
  - In questa tassonomia i kernel language sono ordinati secondo il "creative extension principle": un nuovo concetto è aggiunto quando questo non può essere codificato solo con trasformazioni locali

Linguaggi, paradigmi e concetti

Creative Extension Principle

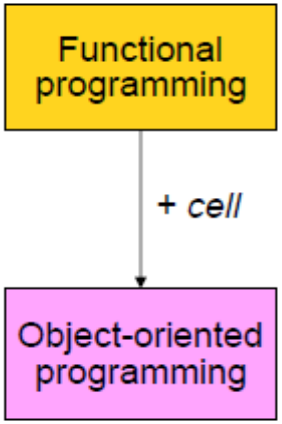
- Quando, in un determinato paradigma, i programmi iniziano ad essere complicati per motivi tecnici che non sono correlati al problema da risolvere,
  - Es: sono necessari trasformazioni di programma non locali
- allora c'è bisogno di un nuovo concetto di programmazione
  - L'aggiunta di questo concetto al paradigma permette ai programmi di essere semplici di nuovo

Esempio: il concetto di *concorrenza*
Se il paradigma non supporta la concorrenza, e questa è necessaria, allora si è costretti a forzare il paradigma ed implementarla «a mano», rendendo i programmi complicati.

Linguaggi, paradigmi e concetti

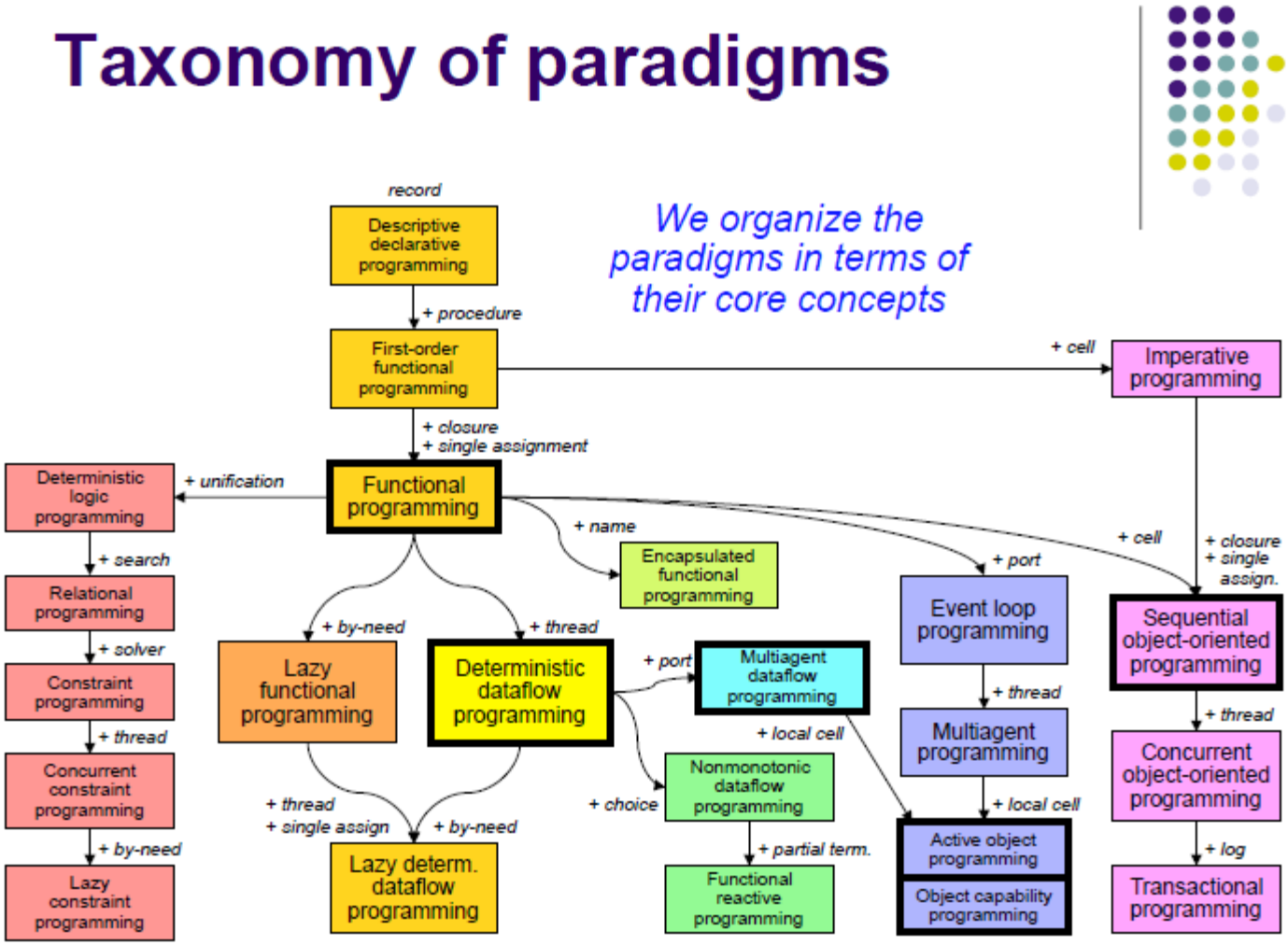
Disassembling a paradigm into concepts

- A paradigm = a set of concepts organized as a kernel language
- Kernel languages of different paradigms are related
  - Often two kernel languages differ only in one concept
- So we can organize the kernel languages as a family tree
  - Two paradigms are linked when they differ in exactly one concept
  - This gives us a taxonomy of programming paradigms



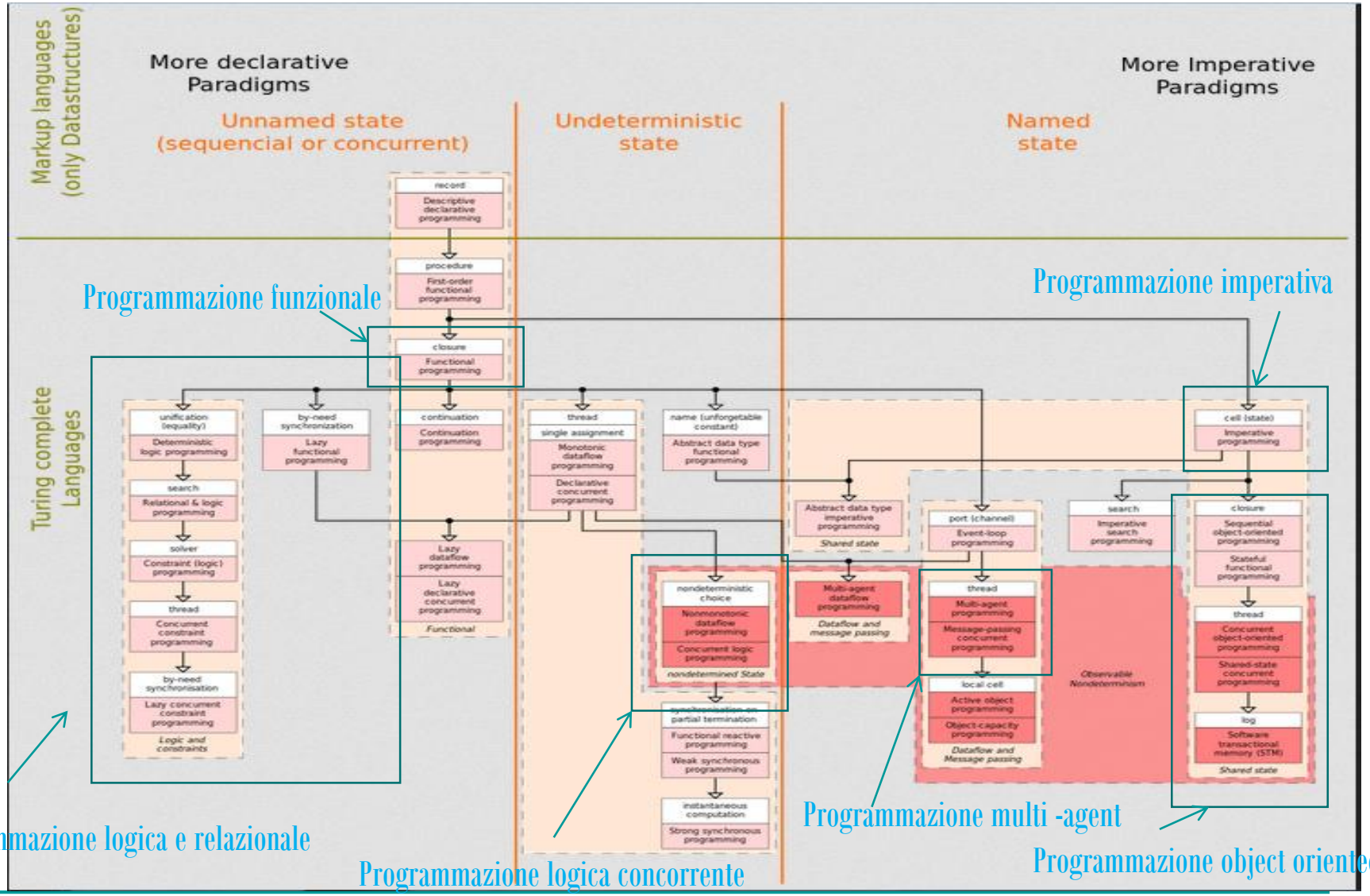
https://studio.edx.org/c4x/LouvainX/Louv1.01x/asset/13-2paradigmsFigure.pdf

Linguaggi, paradigmi e concetti

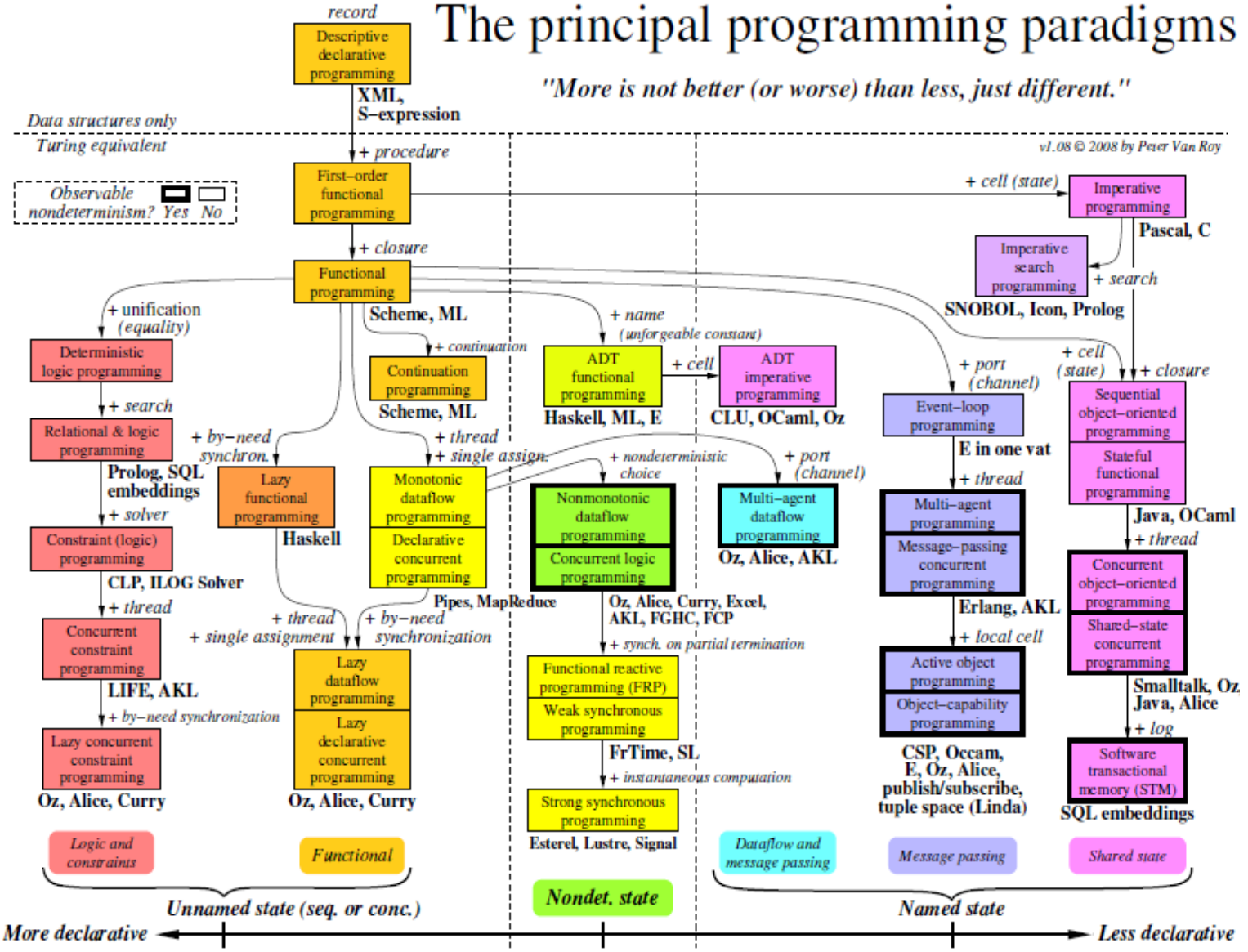


https://studio.edx.org/c4x/LouvainX/Louv1.01x/asset/13-2paradigmsFigure.pdf

Overview of programming paradigms according to Peter Van Roy



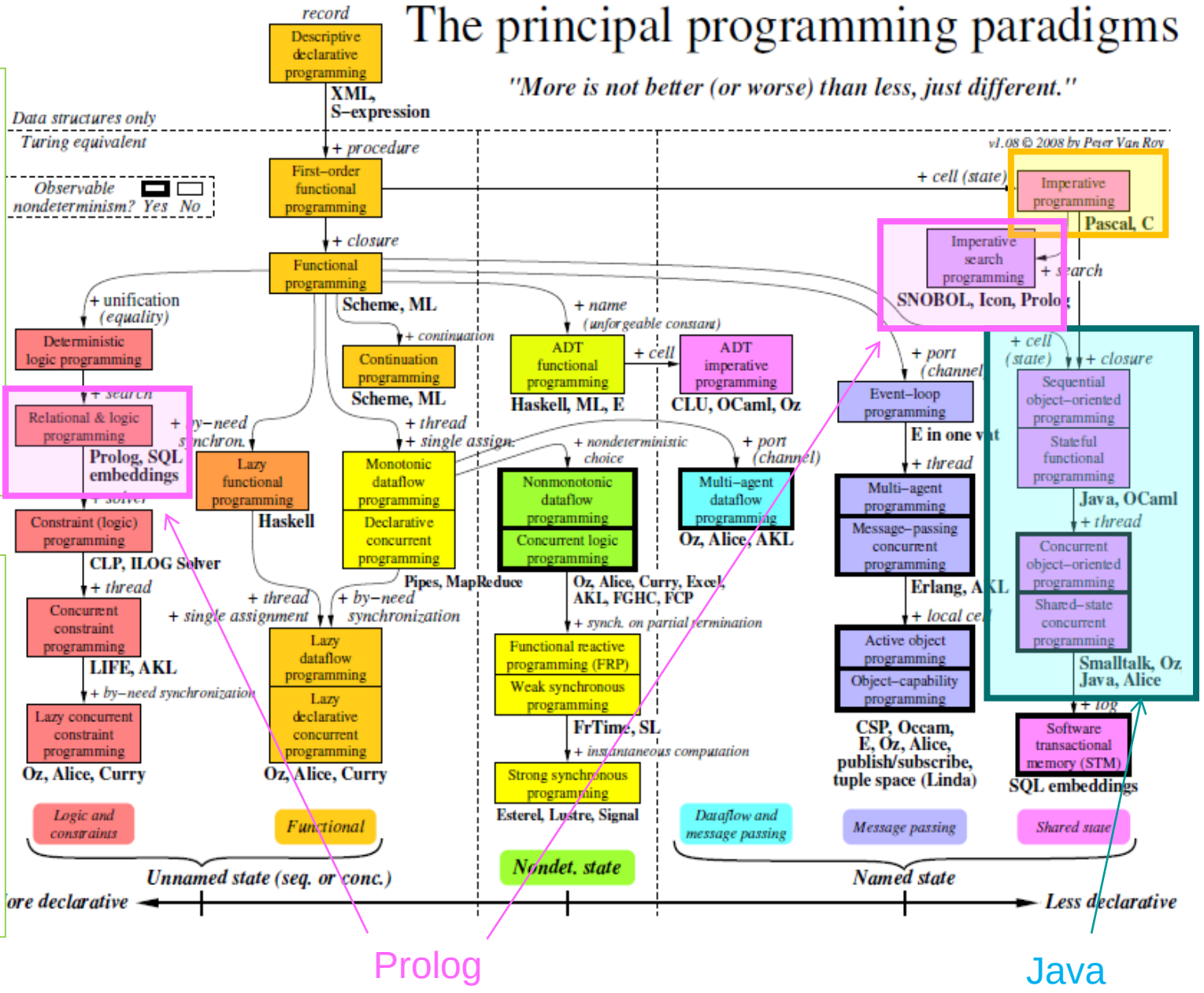
https://commons.wikimedia.org/wiki/File:Programming_paradigms.svg



https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf

√ When a language is mentioned under a paradigm, it means that part of the language is intended (by its designers) to support the paradigm without interference from other paradigms

√ It is not enough that libraries have been written in the language to support the paradigm. The language's kernel language should support the paradigm.



State is the ability to remember information -more precisely, to store sequence of values in time.

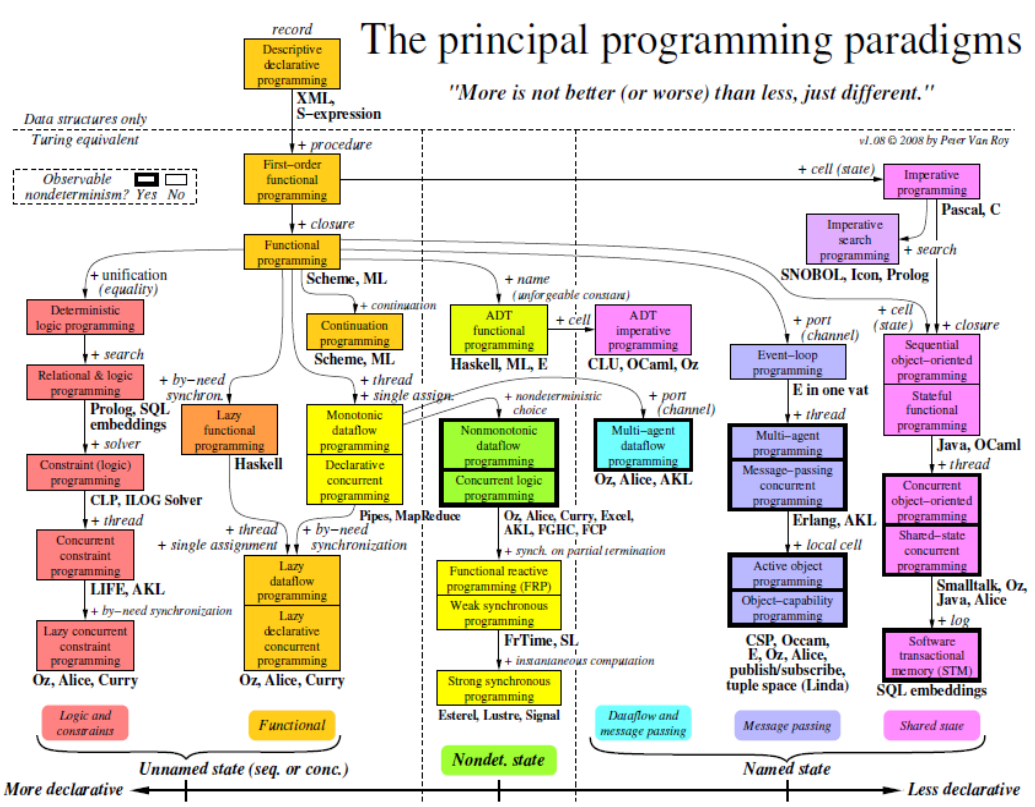
→Its expressive power is strongly influenced by the paradigm that contains it.

○3 levels of expressiveness whether the state is:

- unnamed or named
- deterministic or nondeterministic
- sequential or concurrent.

√ The least expressive is functional programming.

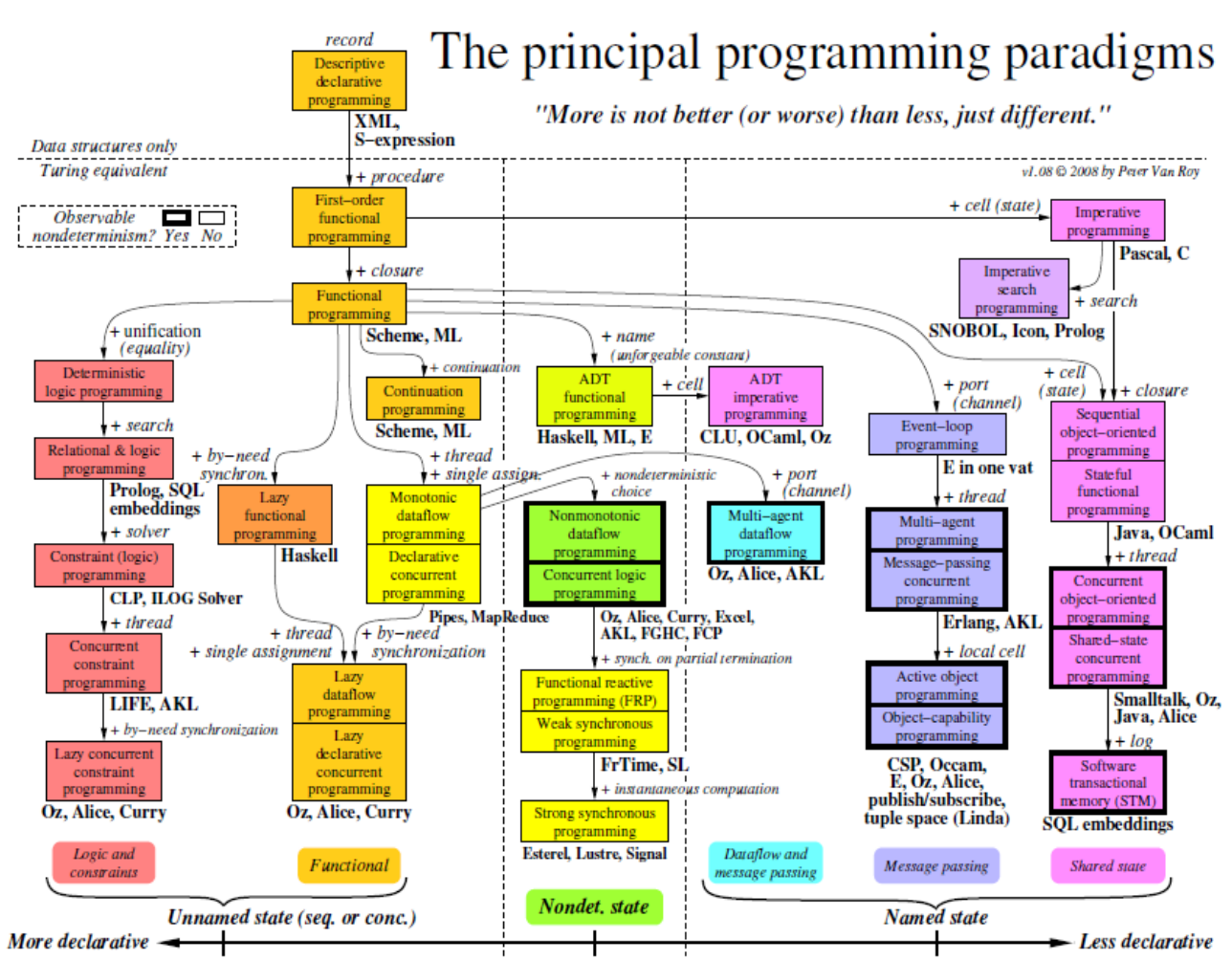
- Adding concurrency gives declarative concurrent programming
- Or adding cells gives shared state, that is named state, which is important for the modularity.



Metaprogramming is another way to increase the expressiveness of a language, but this flexibility is not shown in the chart

https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf

Linguaggi, paradigmi e concetti



Explanations

See "Concepts, Techniques, and Models of Computer Programming".

The chart classifies programming paradigms according to their kernel languages (the small core language in which all the paradigm's abstractions can be defined). Kernel languages are ordered according to the creative extension principle: a new concept is added when it cannot be encoded with only local transformations. Two languages that implement the same paradigm can nevertheless have very different "flavors" for the programmer, because they make different choices about what programming techniques and styles to facilitate.

When a language is mentioned under a paradigm, it means that part of the language is intended (by its designers) to support the paradigm without interference from other paradigms. It does not mean that there is a perfect fit between the language and the paradigm. It is not enough that libraries have been written in the language to support the paradigm. The language's kernel language should support the paradigm. When there is a family of related languages, usually only one member of the family is mentioned to avoid clutter. The absence of a language does not imply any kind of value judgment.

State is the ability to remember information, or more precisely, to store a sequence of values in time. Its expressive power is strongly influenced by the paradigm that contains it. We distinguish four levels of expressiveness, which differ in whether the state is unnamed or named, deterministic or nondeterministic, and sequential or concurrent. The least expressive is functional programming (threaded state, e.g., DCGs and monads: unnamed, deterministic, and sequential). Adding concurrency gives declarative concurrent programming (e.g., synchrocells: unnamed, deterministic, and concurrent). Adding nondeterministic choice gives concurrent logic programming (which uses stream mergers: unnamed, nondeterministic, and concurrent). Adding ports or cells, respectively, gives message passing or shared state (both are named, nondeterministic, and concurrent). Nondeterminism is important for real–world interaction (e.g., client/server). Named state is important for modularity.

Axes orthogonal to this chart are typing, aspects, and domain–specificity. Typing is not completely orthogonal: it has some effect on expressiveness. Aspects should be completely orthogonal, since they are part of a program's specification. A domain–specific language should be definable in any paradigm (except when the domain needs a particular concept).

Metaprogramming is another way to increase the expressiveness of a language. The term covers many different approaches, from higher–order programming, syntactic extensibility (e.g., macros), to higher–order programming combined with syntactic support (e.g., meta–object protocols and generics), to full–fledged tinkering with the kernel language (introspection and reflection). Syntactic extensibility and kernel language tinkering in particular are orthogonal to this chart. Some languages, such as Scheme, are flexible enough to implement many paradigms in almost native fashion. This flexibility is not shown in the chart.

https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf