À la découverte de Julia !

Thibaut Cuvelier

À la découverte de Julia !

Pourquoi pas Python ?
Qui l'utilise et pour quoi ?
À quoi ressemble l'écosystème ?

Pratiquons !
Syntaxe de base
Manipulation de données (DataFrames.jl)
Manipulation de graphes (Graphs.jl)

Thibaut Cuvelier — software engineer, Google Paris

Installation des prérequis

- Installer Julia pour votre plateforme:

  https://julialang.org/downloads/

- Installer les dépendances **depuis Julia** :

```
] add IJulia DataFrames RDatasets Chain Graphs
using IJulia
notebook()
```

  - Si vous demandez d'installer Conda, ce sera une version **locale** à Julia

Qui suis-je ?

- Actuellement *software engineer* chez Google Paris
  - Recherche opérationnelle sur des tournées de véhicules
  - Services disponibles depuis peu chez Google Cloud : https://cloud.google.com/optimization/docs/overview
- Auparavant, doctorat Cifre chez Orange
  - Recherche opérationnelle sur du routage dans des réseaux informatiques
  - Algorithmes efficaces pour l'apprentissage par renforcement
- Contributeur au logiciel libre : JuMP en Julia, Qt et LyX en C++…

Pour quoi ai-je utilisé Julia ?

- Travail de fin d'études (2014-2015) : optimisation stochastique et robuste
- Optimisation de gestion de barrages (2015-2016)
  - ReservoirManagement.jl
- Optimisation de processus industriel (2016-2017)
  - IndustrialProcessFlexibilisation.jl
- Optimisation de réseaux informatiques (2017-2022)
  - Seleroute.jl
- Algorithmes de bandits combinatoires (2017-2021)
  - CombinatorialBandits.jl, Kombinator.jl
- Programmation par contraintes (2021-2022)
  - Extensions pour JuMP.jl : ConstraintProgrammingExtensions.jl, CPLEXCP.jl

Analyse de données
Modèles statistiques
Optimisation stochastique et robuste

Algorithmes combinatoires (thèse)

Pourquoi pas Python ?

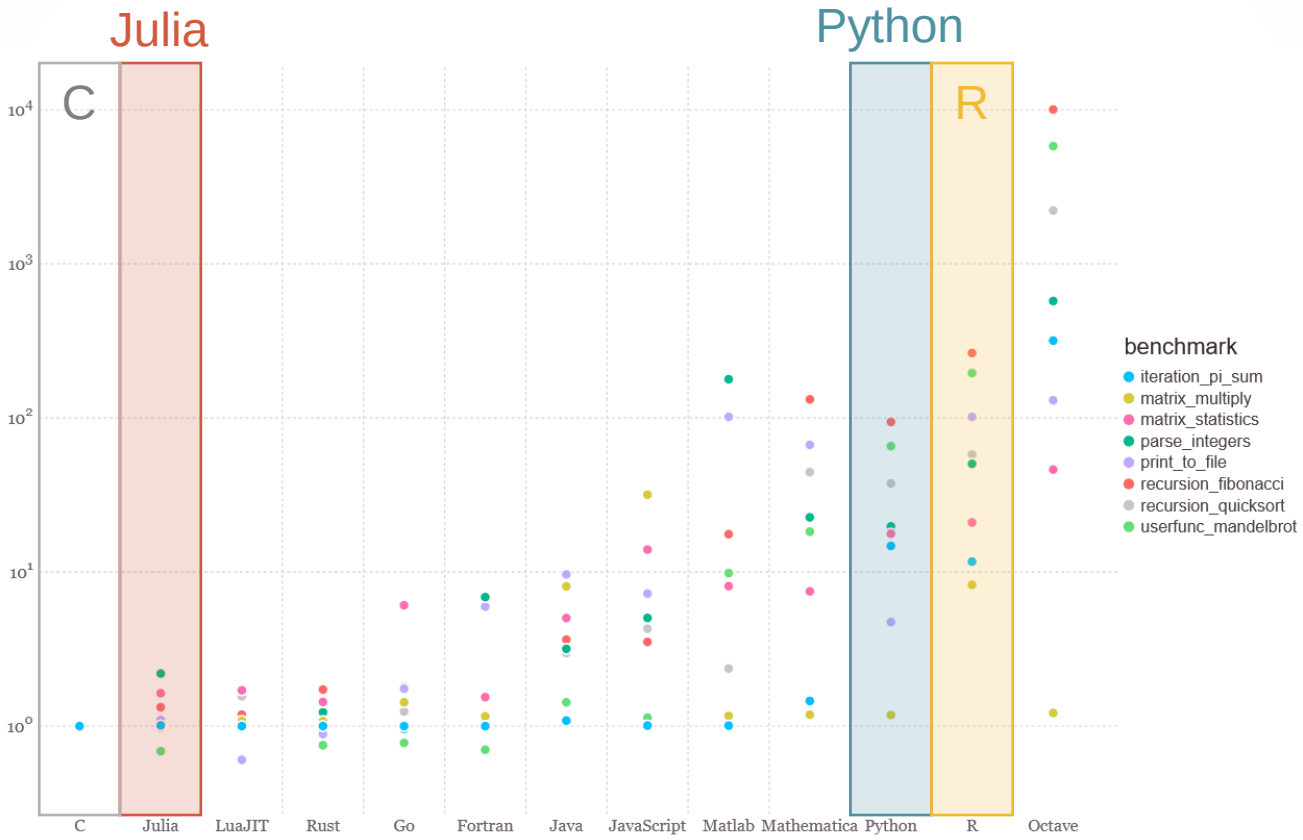
- Raison nº 1 : performance
- Raison nº 2 : problème des deux langages

➢ Pour du code Python vraiment performant, il faut **quitter Python**

Pourquoi Python ?

- Syntaxe agréable, productive
- Facile à apprendre

Pourquoi se contenter de compromis ?



Ont-ils réussi ?

Source : https://julialang.org/benchmarks/

Effectué en 2020

Depuis, Python 3.11 a changé la donne

groupby | join | groupby2014

0.5 GB | 5 GB | 50 GB

basic questions

Input table: 1,000,000,000 rows x 9 columns (50 GB)

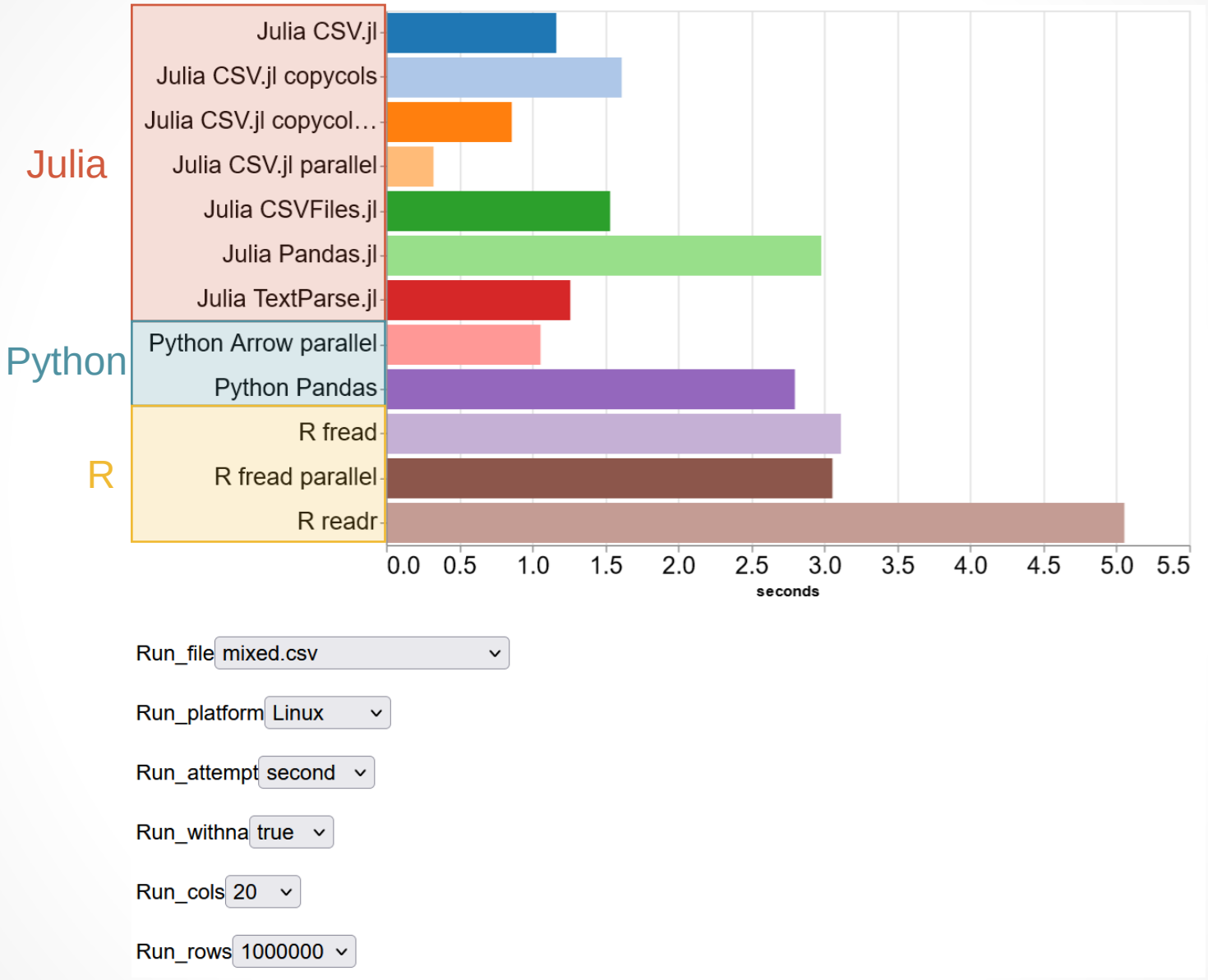
Rust	Polars	0.8.8	2021-06-30	143s
R	data.table	1.14.1	2021-06-30	155s
Julia	DataFrames.jl	1.1.1	2021-05-15	200s
SQL	ClickHouse	21.3.2.5	2021-05-12	256s
	cuDF*	0.19.2	2021-05-31	492s
	spark	3.1.2	2021-05-31	568s
	(py)datatable	1.0.0a0	2021-06-30	730s
	dplyr	1.0.7	2021-06-20	internal error
Python	pandas	1.2.5	2021-06-30	out of memory
	dask	2021.04.1	2021-05-09	out of memory
	Arrow	4.0.1	2021-05-31	internal error
	DuckDB*	0.2.7	2021-06-15	out of memory

Ont-ils réussi ?
> *Data frames*

Source : https://h2oai.github.io/db-benchmark/

Effectué en 2021

Voir aussi : Julia et Python, en 2022

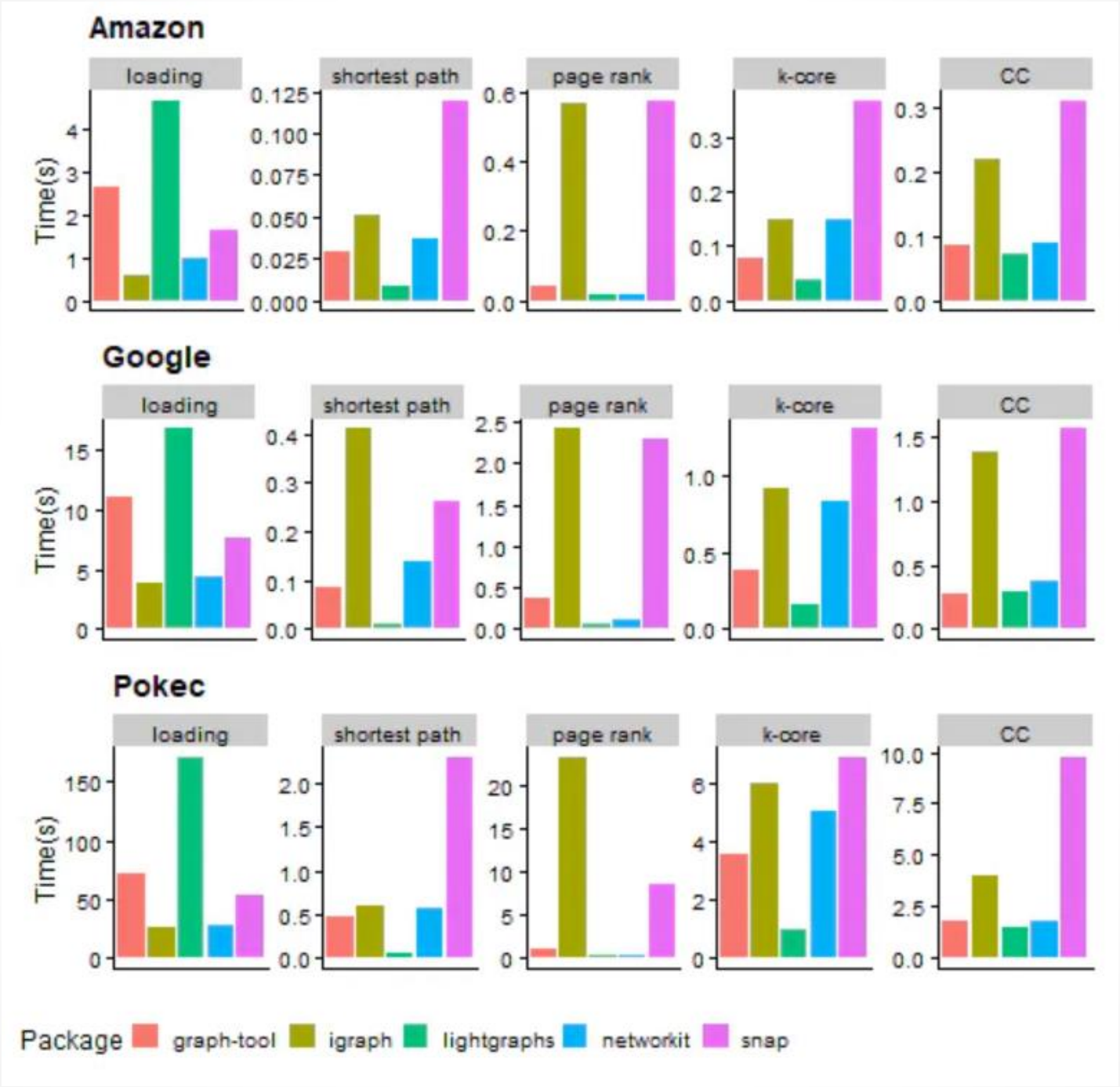


Ont-ils réussi ?
> Lecture de CSV

Source : https://www.queryverse.org/benchmarks/

Effectué en 2021

Temps de calcul après compilation (*"second attempt"*)



Ont-ils réussi ?
> Graphes

Source : https://www.timlrx.com/blog/benchmark-of-popular-graph-network-packages-v2

Effectué en 2020

Qui utilise Julia ?

- Google : prototype de PDLP développé en Julia, le passage en C++ n'améliore pas la performance
- Jupyter (ex-IPython) : Ju pour… Julia !
- New York Federal Reserve Bank : simulation économique, 10 x plus rapide que MATLAB
- Pfizer : pharmacologie, 175 x plus rapide que C++ (accès au GPU)
- INPE (Brésil) : simulation pour des missions spatiales (Amazonia-1)

À quoi ressemble la communauté Julia ?

- Julia et son écosystème sont développés sur GitHub, en public
  - Facile de contribuer !
  - Calendrier public des réunions de l'écosystème (GPU, HPC, ML, etc.)
- Pour obtenir de l'aide :
  - Discourse, StackOverflow
  - Slack, Zulip, Discord
- Une conférence majeure : JuliaCon, active depuis 2014
  - Prochaine édition : 24-29 juillet 2023, Cambridge, MA (USA)
  - Accompagnée d'un journal : Proceedings of the JuliaCon Conferences

Écosystème Julia

- Analyse de données :
  - Tables.jl : interface abstraite pour des tableaux (*data frame*, CSV ou JSON, base de données…)
  - DataFrames.jl : similaire à Pandas ou data.frame
- Visualisation :
  - Plots.jl, Makie.jl : interface de haut niveau (à la MATLAB ou PyPlot)
  - AlgebraOfGraphics.jl : interface de type *grammar of graphics* (ggplot2 en R ou plotnine en Python)
  - RecipesBase.jl : interface entre graphiques et types non standard
  - Moteurs de rendu : natifs (GLMakie.jl, WGLMakie.jl, CairoMakie.jl, UnicodePlots.jl, Gadfly.jl, etc.) ou non (GR, PyPlot, Vega, VegaLite, etc.)
- Apprentissage :
  - MLJ.jl : similaire à scikit-learn
  - Flux.jl : similaire à TensorFlow ou PyTorch
  - JuliaStats : écosystème pour les statistiques (tests d'hypothèses, distributions, etc.)

Écosystème Julia

- Un seul gestionnaire de paquets : Pkg.jl
  - Chacun peut avoir son propre dépôt privé
  - Gestion des dépendances binaires multiplateforme intégrée (Yggdrasil, JuliaBinaryWrappers…)
- Super rapide !

- Précompilation des paquets dès leur installation

Écosystème Julia

- Environnements de développement ?
  - Extension pour Visual Studio Code
  - (Juno — extension pour Atom, plus développé activement)
- Calepins ?
  - Jupyter : noyau Julia (IJulia.jl)
  - Pluto.jl : plus léger, entièrement réactif (comme Excel), spécifique à Julia

Syntaxe de base

Comparaison avec Python et R

Syntaxe de base

Julia

- ```
  if a >= b
  end
  ```
- ```
  'a'^5 * "bc"
  ```
- ```
  slow = []
  fast = Int[]
  ```

Python

- ```
  if a >= b:
      pass
  ```
- ```
  'a'*5 + 'bc'
  ```
- ```
  slow = []
  fast = np.zeros(shape=(3, 2),
                  dtype=np.intc)
  ```

Syntaxe de base

Julia

- `list[1] == first(list)`
- `list[end] == last(list)`
- `list[begin+1:end-1]`

Python

- `list[0]`
- `list[-1]`
- `list[1:-1]`

Syntaxe de base

Julia

- Rien !
- ```
  struct X
    field
  end
  ```
- ```
  mutable struct X
    field::Float64
  end
  ```

Python

- ```
  class X: pass
  ```
- ```
  X = namedtuple('X', ['field'])
  # Immutable
  ```
- ```
  @dataclass
  class X:
    field: float
  ```
  ```

Pourquoi Julia est-il rapide ?

- Compilation à la volée (JIT), comme Python 3.12
  - Lors du chargement des modules
- Multiméthodes : le code à exécuter pour une fonction dépend des types de **tous** les arguments
  - Fonction : un nom (`+`, `read`), souvent partagée entre des paquets (`filter!`)
  - Méthode : une implémentation d'une méthode, comme `+(::Int, ::Int)`
  - Polymorphisme à la C++, Java, Python : seul le type de l'objet est considéré

Place à la pratique !

Et si Python revient dans la course pour la performance ?

- **Syntaxe et DSL** : macros très puissantes en Julia
- **Syntaxe et indexation** : Julia plus cohérent (Pandas et loc, iloc, [], etc.)
- **Parallélisation** : pas de GIL, bibliothèque standard bien fournie
- Dérivation automatique : Zygote.jl travaille directement sur l'assembleur LLVM compilé, pas besoin d'interpréteur limité
  - TensorFlow, PyTorch & co. si : jusque ~ 3500 x plus lents
  - XLA (TensorFlow moderne / JAX) : mêmes principes que Zygote, toute la syntaxe de Python n'est pas disponible
  - Les développeurs de Python le trouvent trop flexible pour la dérivation automatique :
    - Python's flexibility makes it difficult for DSLs embedded in it to use such an approach

➢ **Le compilateur de Julia est très accessible depuis Julia** !