

---

**It's Everywhere**

**DATA? DATA!**

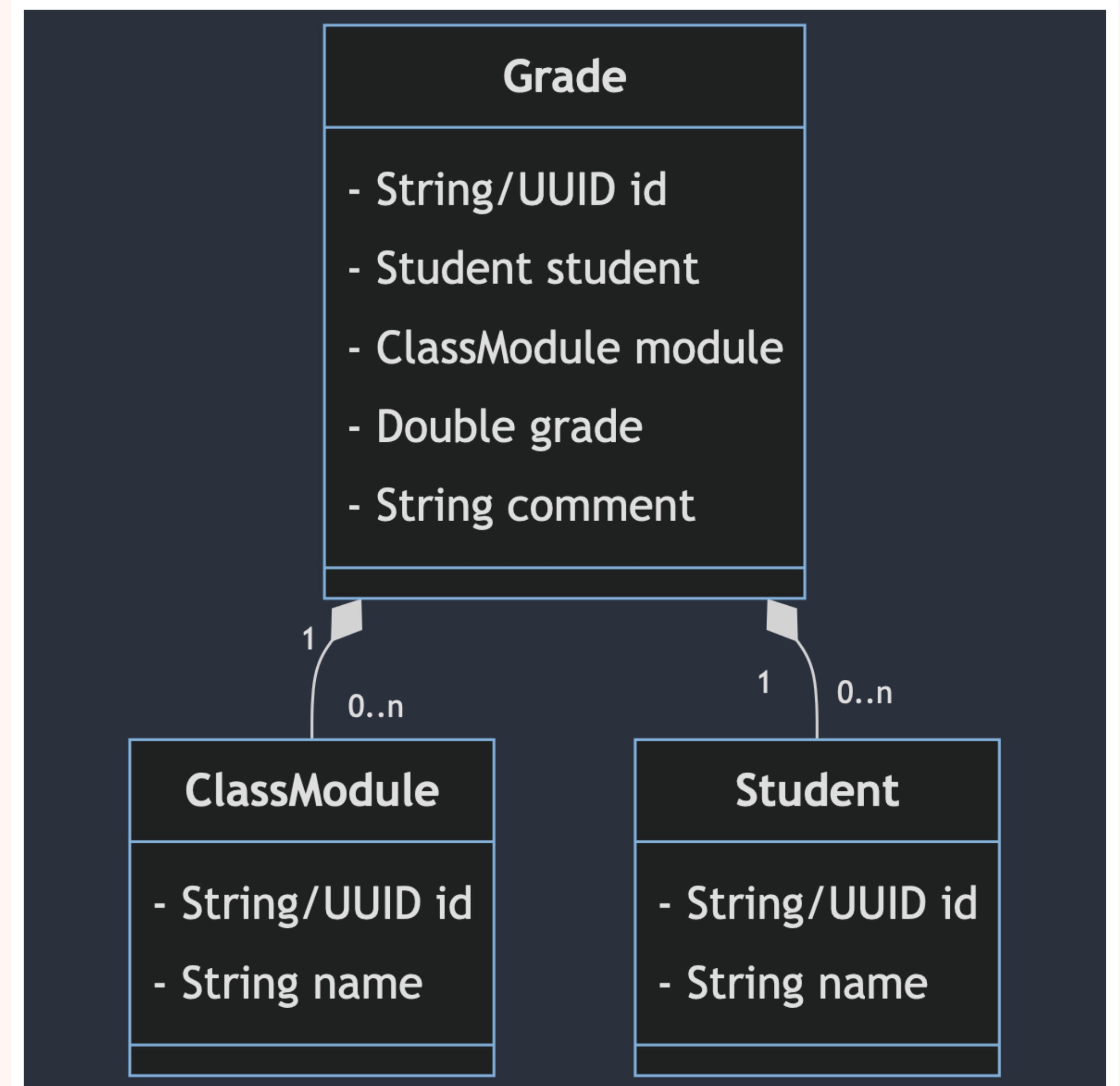
---

# ER... WHY?

- **I was curious**
- **I like data**
- **It looks like an important tradeoff down the line**

# METHODOLOGY

- **1 GB of hierarchical data**
- **~ 7k students, ~30 modules, ~5 grades per both**
- **Generation is random every time to avoid caching bias**



---

# BASELINE

- **Last Mac Mini in Intel version, plenty of RAM, SSD**
- **Re-generate the dataset before each test**
- **Measured with Benchmark, the new-ish Swift framework**



---

# YE OLDE SCHOOL

- **JSON serialization**
- **Manual SQLite commands**
- **No framework!**



So JSON!



Much SQLite!



Wow!

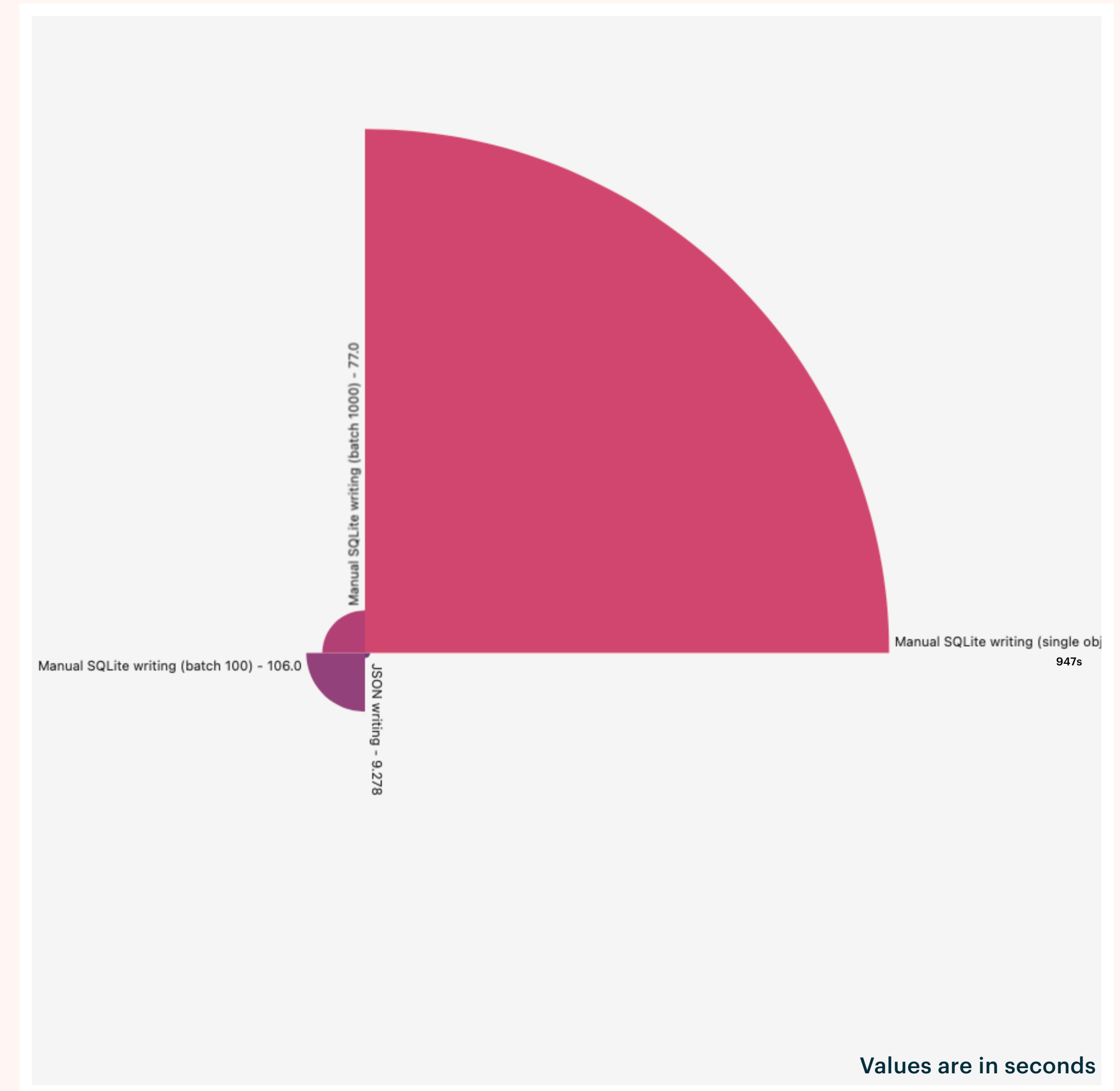
---

# IN-MEMORY VS JSON

- **Roughly the same amount of RAM used (duh)**
- **Roughly the same time calculating the average (duh)**
- **36 ms to calculate the average (once the data is loaded)**
- **7-9s to read or write the whole JSON**

# JSON VS SQLITE (MANUAL)

- **Reading: 10ms for SQLite vs 8000ms for JSON**
- **Writing: Batching is important!**
  - **Minor differences between batching by 100 or 1000, but not batching at all is 10x slower**
  - **Because of ~computer science~, SQLite is slower on the insertion side by 10x, for a 800x speedup on reading in exchange**



---

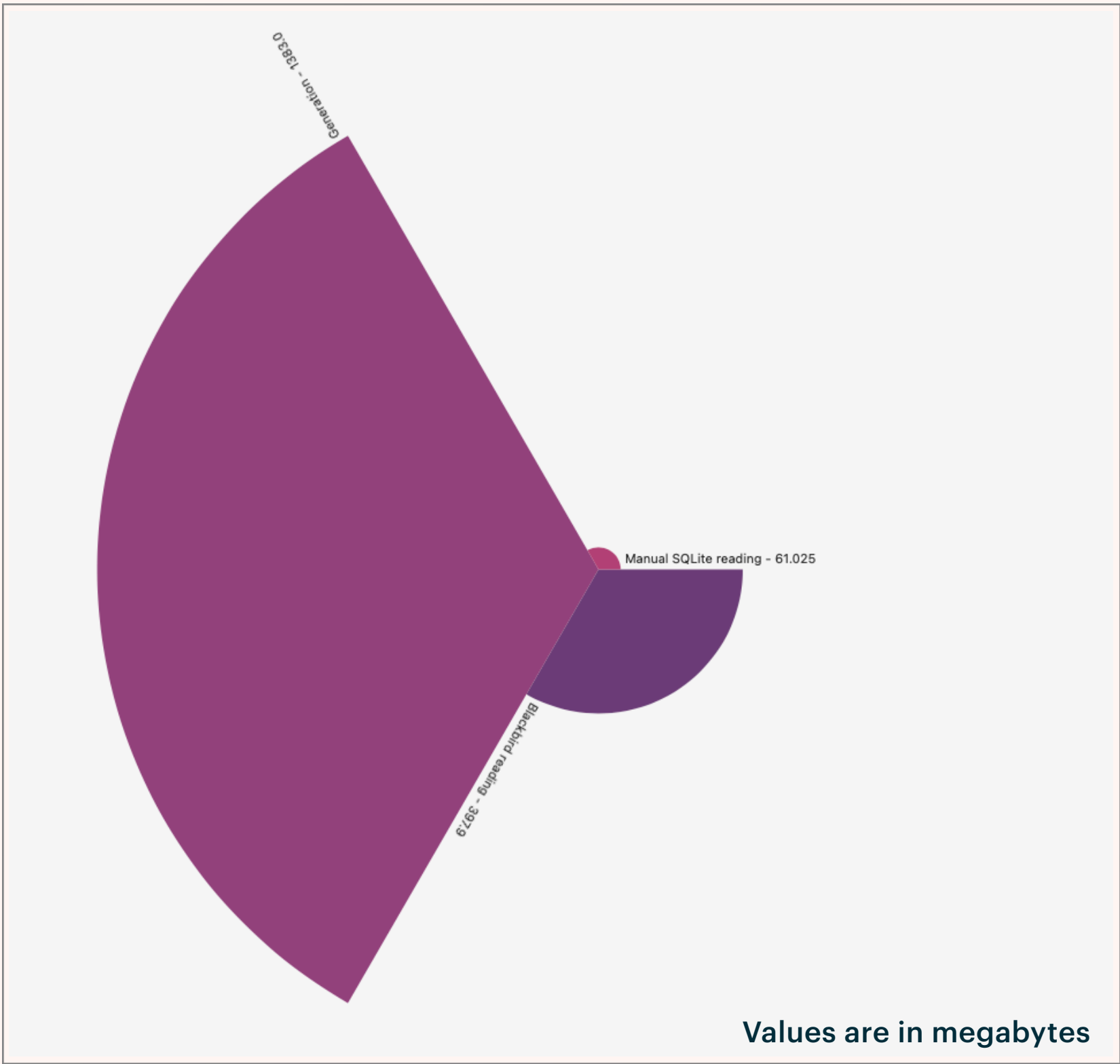
# A MODERN SQLITE FRAMEWORK

- **Blackbird (by Marco Arment) used in production**
- **Not optimized for speed! Optimized for code length**
- **3 times fewer code than manual SQLite and  $\infty$  times less errors and debugging minutes**

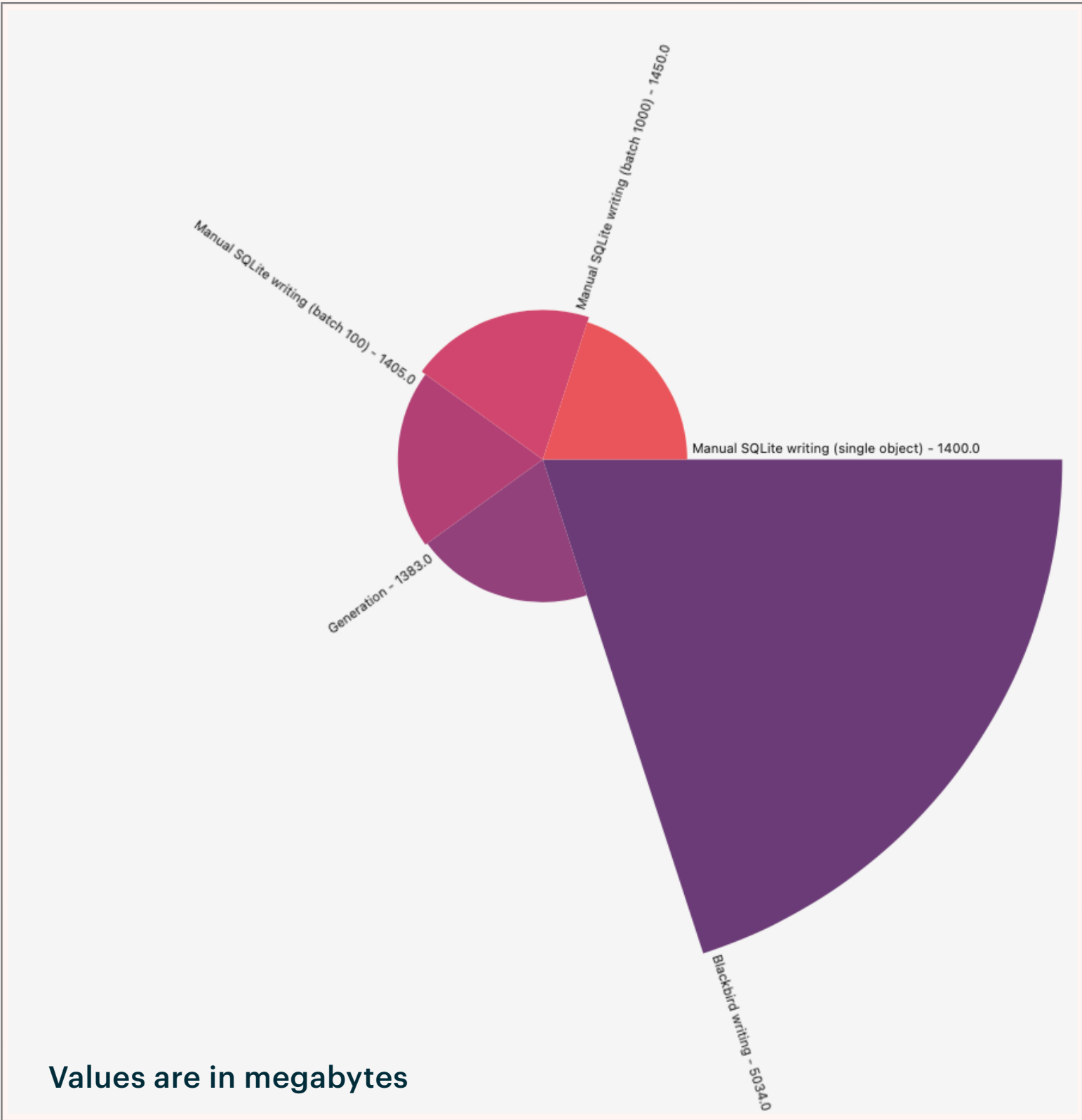


**So modern!**

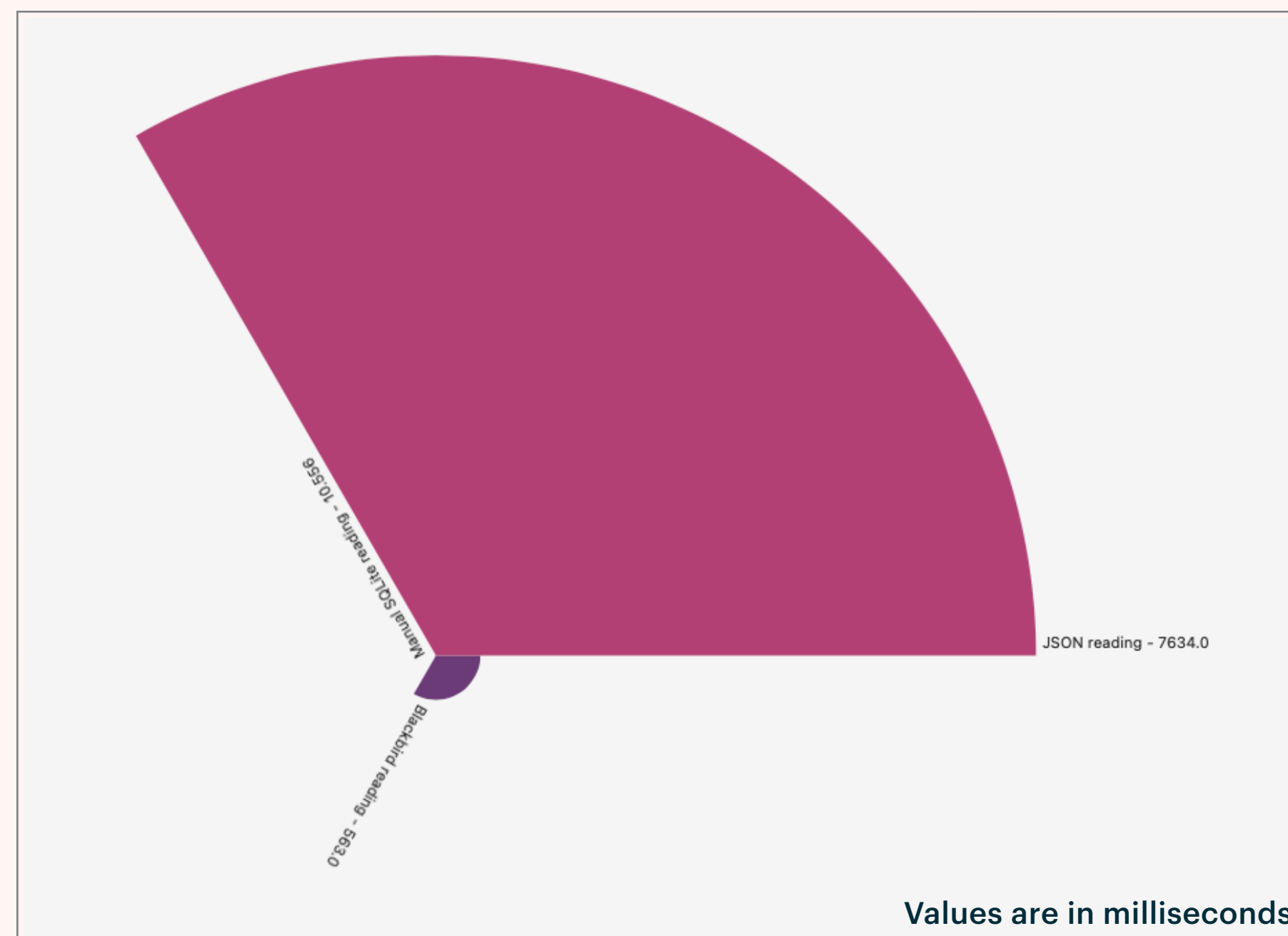
# SQLITE (MANUAL) VS BLACKBIRD (MEMORY)



**3x the memory because of the scaffolding  
Makes sense. Worth it.**

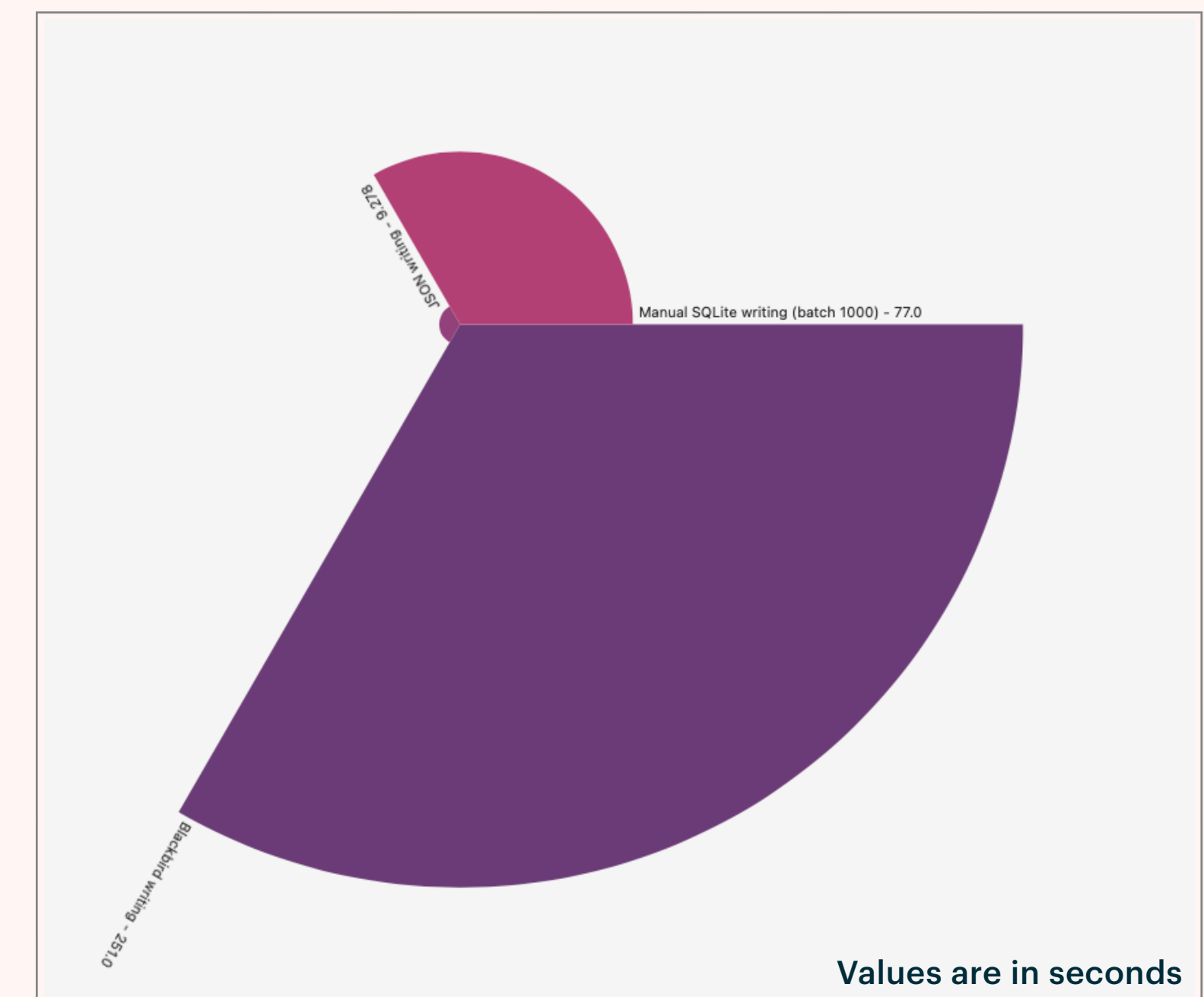


# SQLITE (MANUAL) VS BLACKBIRD (SPEED)



**Not a huge tradeoff for type safety  
and model migrations**

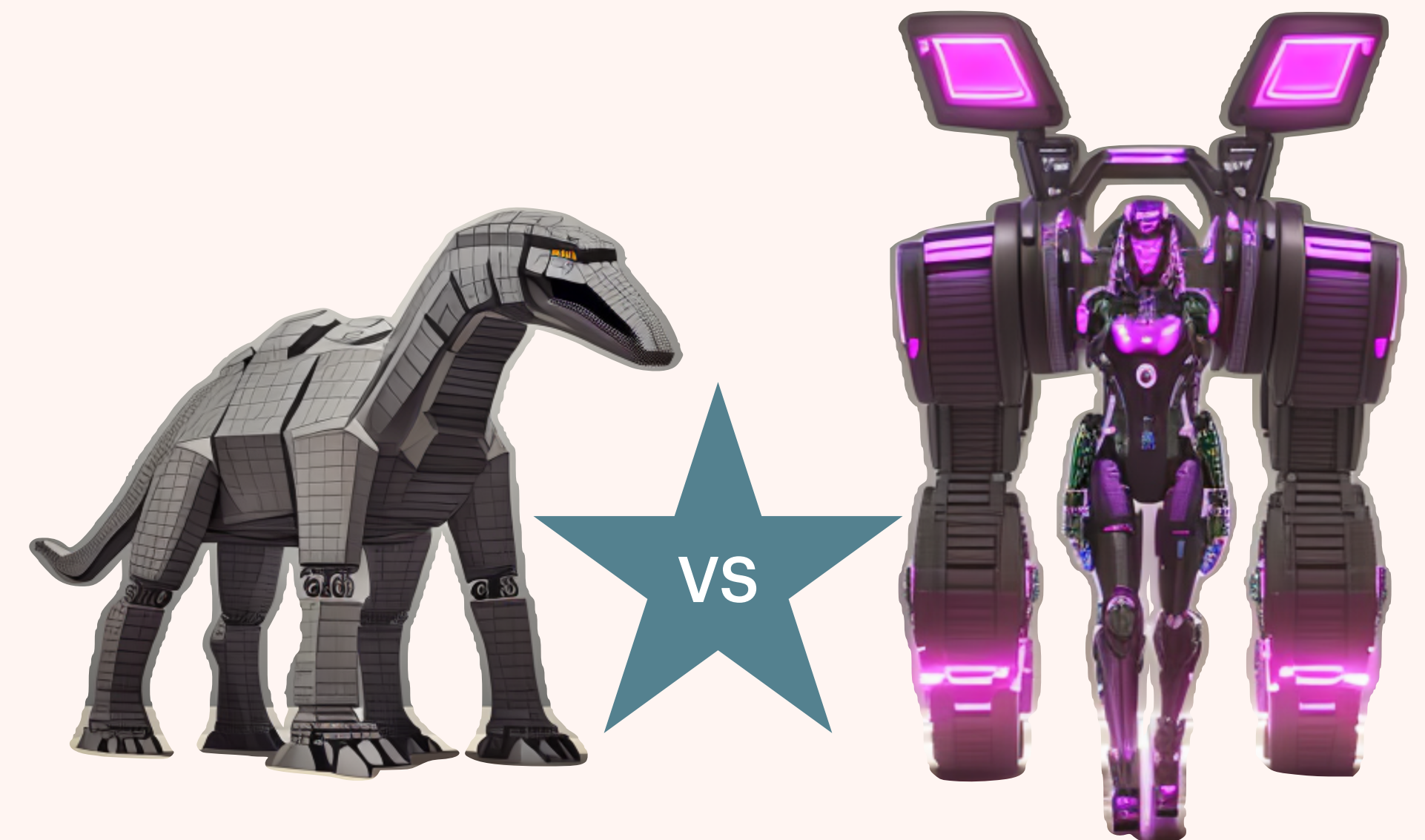
**Plus, full async vs my crappy code**



---

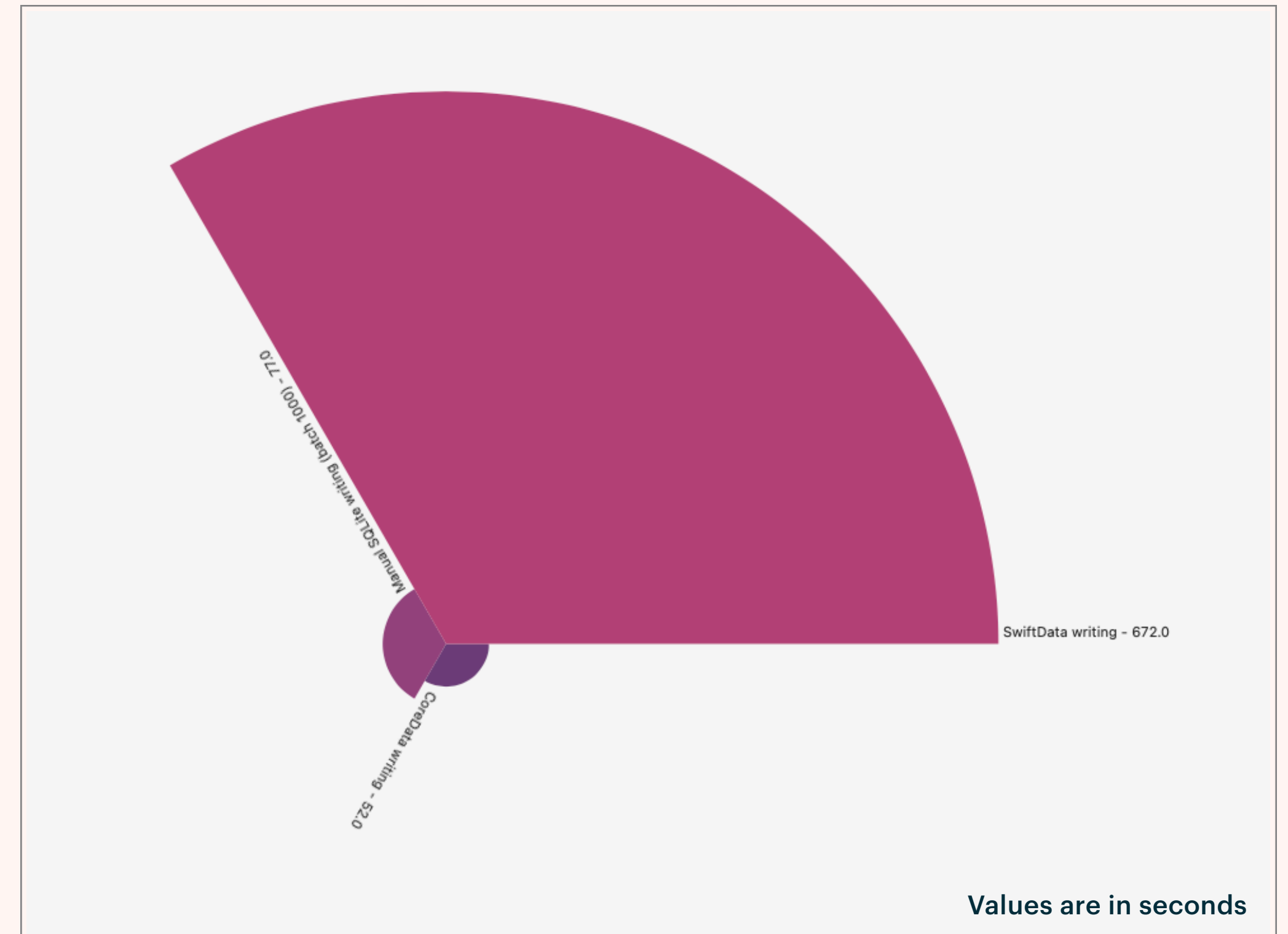
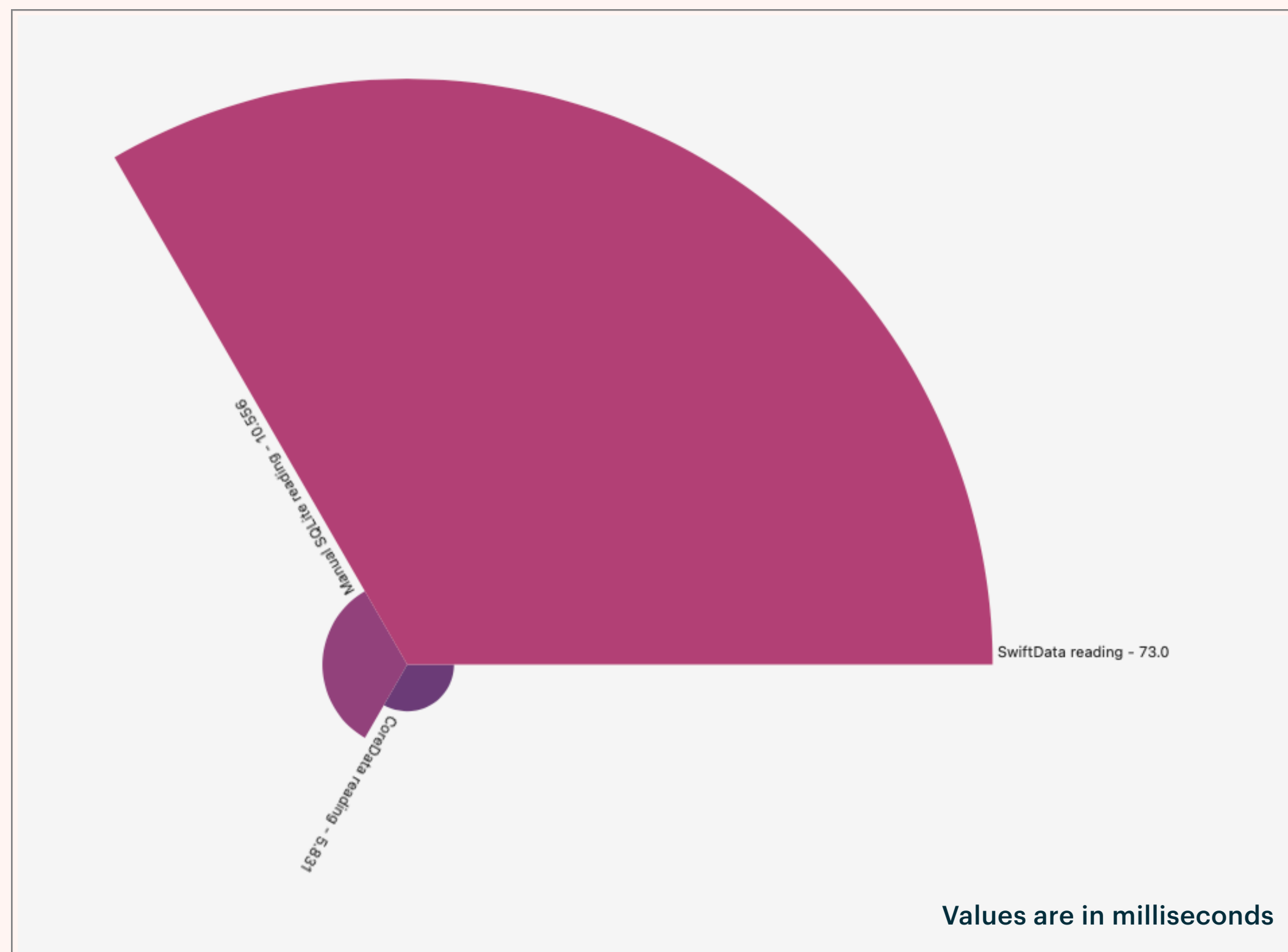
# COREDATA VS SWIFTDATA

- **SwiftData is CoreData under the hood, for the most part**
- **Both use SQLite as a backend**
- **CoreData was born out of EOF on NeXT, and publicly released in 2005**





# SQLITE (MANUAL) VS COREDATA VS SWIFTDATA





---

# SQLITE (MANUAL) VS COREDATA VS SWIFTDATA

- **Hold your horses!**
- **SwiftData is very SwiftUI centric, and changes a lot between Sonoma and Sequoia**
- **It is very opinionated, and so am I. Maybe incompatibly so.**
- **It is absolutely not meant for that kind of usage and I had to fight it every step of the way**



---

# BOOOOOO!

- **SQLite and CoreData are mature, and extremely optimized over the years, but they have idiosyncrasies that you may hate, and the debugging is hard**  
(Time spent on writing the code: ~2 days, because of **CLI vs CoreServices** issues)
- **Blackbird (and other frameworks like it) have a very different optimization strategy: minimizing development time**  
(Time spent on writing the code: ~2 hours)
- **SwiftData is constantly evolving and optimized for a view-centric world**  
(Time spent on writing the code: 5 days, because of the differences between platform versions, especially regarding **async/await** tolerances)

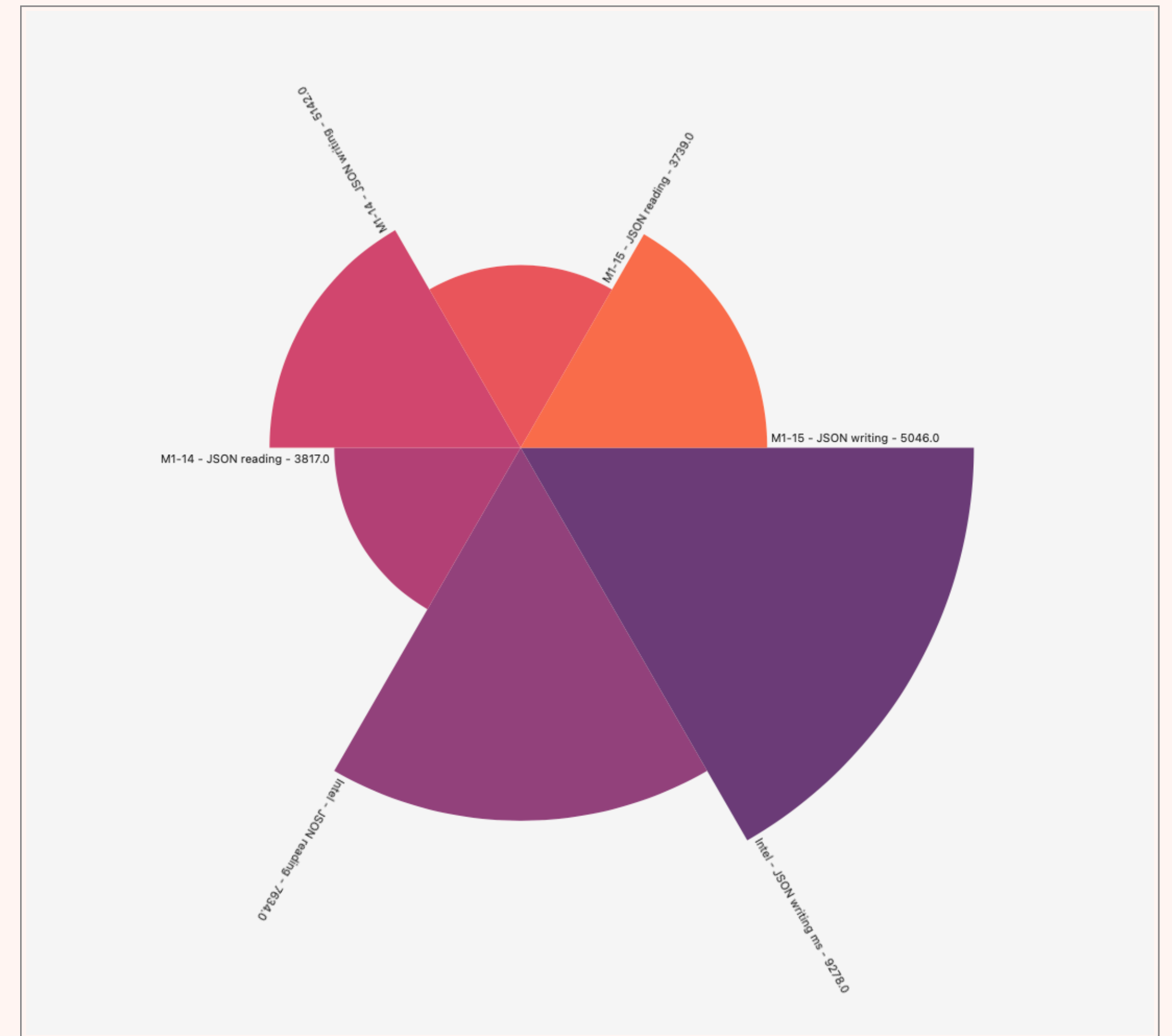
---

# BENCHMARK

- **VERY simple to use for simple CLI tools, but super hard to use in graphical/asynchronous environments**
- **A lot of undocumented behaviors (leaks, warmup cycles, etc)**
- **Promising for datacrazies!**

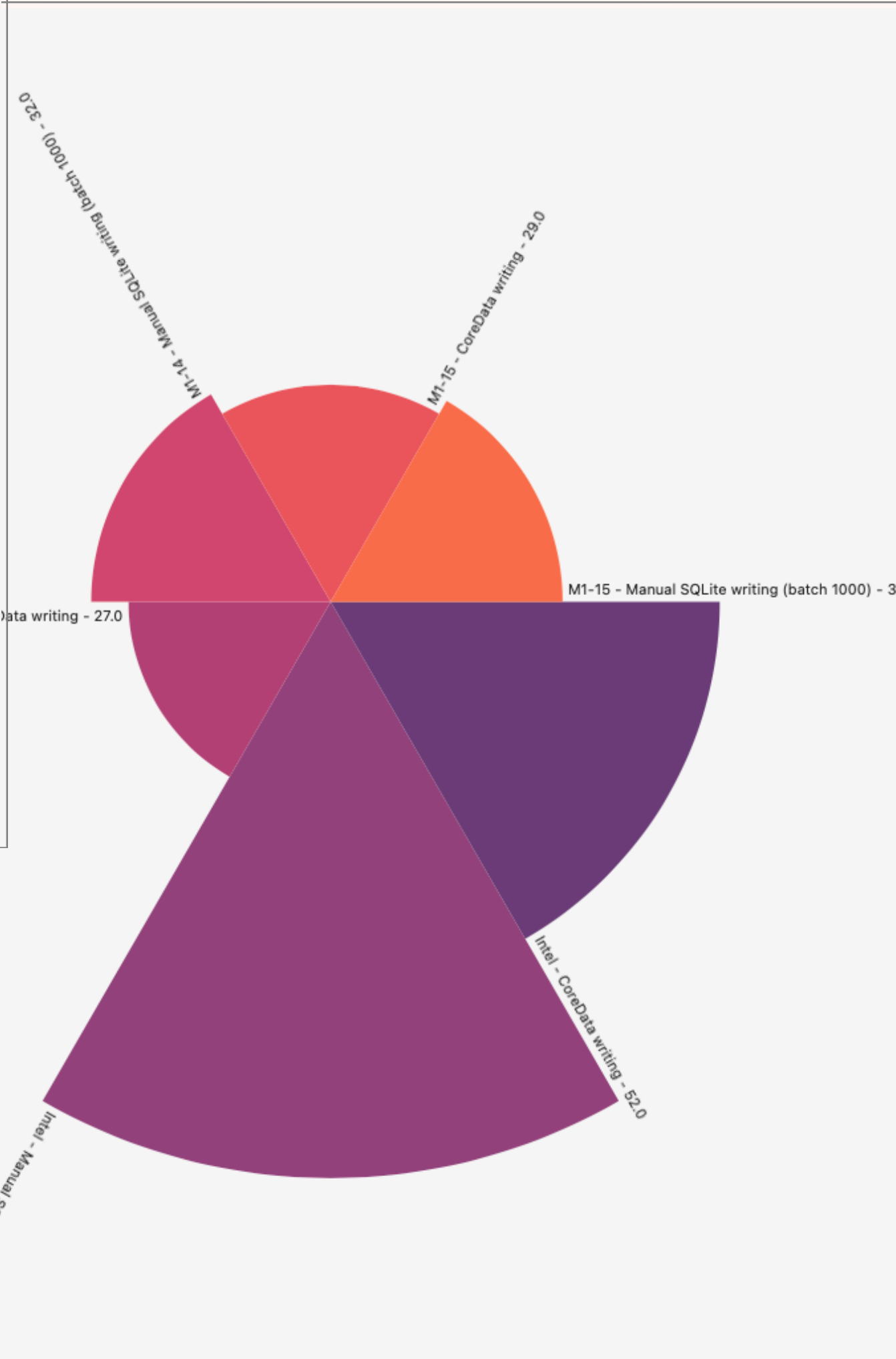
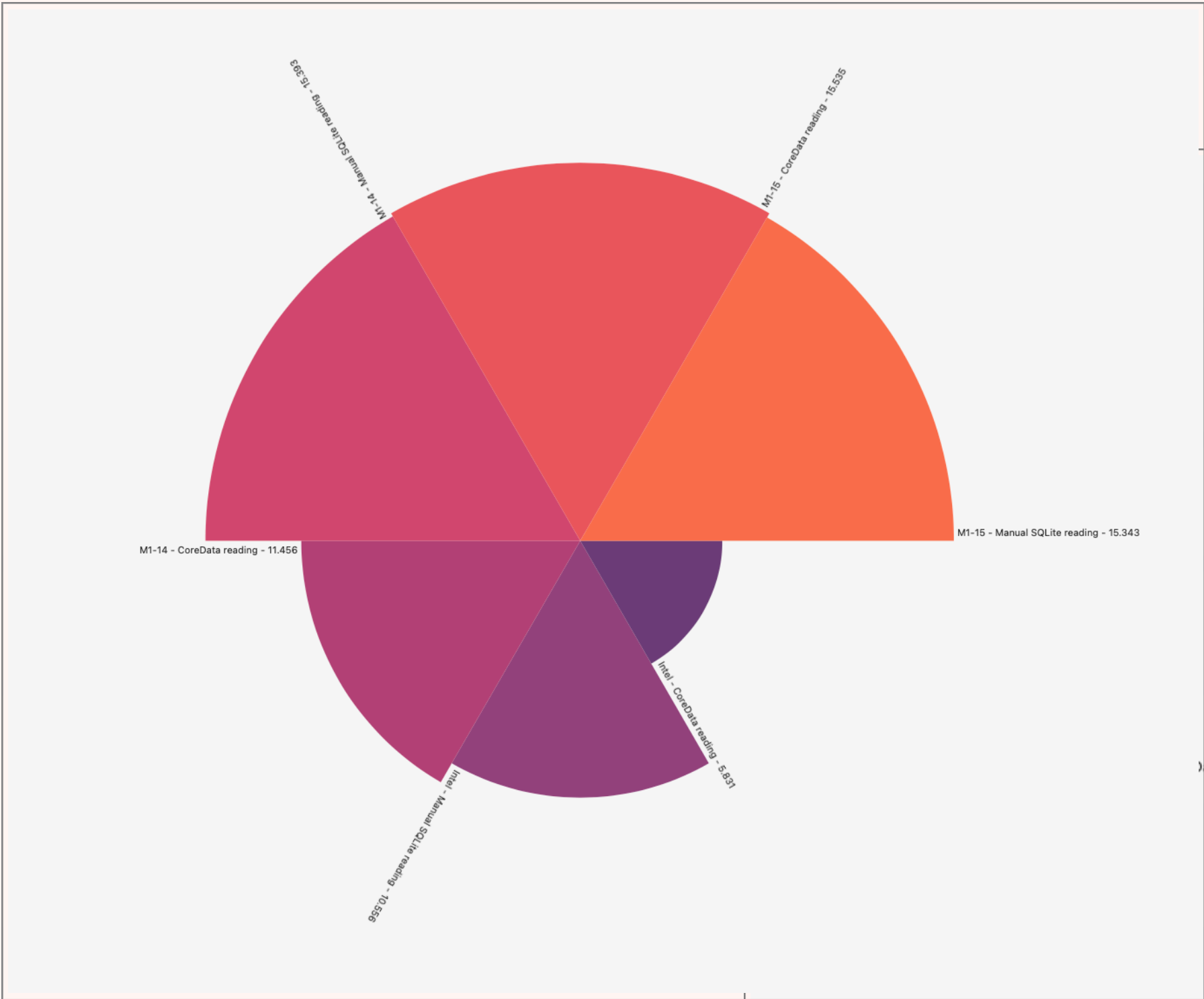
# BONUS ROUND

- **JSON**
- **Intel MacOS 14  
vs M1 MacOS 14  
vs M1 MacOS 15**
- **M1 is roughly 2x as fast, no regression  
between MacOS 14 and 15**
- **Blackbird follows the same pattern, with a  
tiny speedup under MacOS 15, thanks to  
Swift compiler advances**



# BONUS ROUND

- SQLite and CoreData
- Intel MacOS 14 vs M1 MacOS 14 vs M1 MacOS 15
- Intel is faster while reading, slower when writing. No regression



# BONUS ROUND

- **SwiftData**
- **Intel MacOS 14  
vs M1 MacOS 14  
vs M1 MacOS 15**
- **I... don't want to talk about  
it**



Every run is wildly  
different

Memory TRIPLES  
after disk write is  
finished

Writing is  
increasingly better  
Reading is not

---

**It's Everywhere**

**DATA? DATA!**