EPISODE 1495

[INTRODUCTION]

[00:00:00] ANNOUNCER: Apache Arrow defies a language-independent column or memory format for flat and hierarchical data organized for efficient analytic operations on modern hardware like CPUs and GPUs. The Arrow memory format also supports zero copy reads for lightning fast data access without serialization overhead.

Wes McKinney is the CEO of Ursa Computing, a new startup working on accelerated computing technologies, powered by Apache Arrow for data science languages like Python and R. He joins the show to discuss the next generation computational infrastructure for data science.

[INTERVIEW]

[00:00:39] JM: Wes, welcome to the show.

[00:00:40] WM: Thanks. Glad to be here.

[00:00:42] JM: I want to start by talking about Apache Arrow, which in my mind has for a pretty long time now been one of the most important successful Apache projects that does not have a company built around it. That's largely because it's essentially a Data Interchange Format and it's – Whenever I talk to people about Apache Arrow, I always thought to myself, “Man, it would be really cool if they could build a company around this.” But it's just hard to conceive of an entire company built around a Data Interchange Format. Has that been a discussion or a thought for as long as the Apache Arrow project has been in creation?

[00:01:29] WM: I mean, it's a good question, and it's a question that many people have, is that I think given the way that the Arrow project started, many people do think of it as being primarily interchanged format. But the reality is that the scope of the project is significantly larger than that or at least has grown significantly larger than that over time.

If you go back to like the initial conversations that we had about starting the project, certainly developing a universal language in an attended columnar format was a central part of that and coming up with like the basic ideas around developing this protocol for productivity that could span across different computing engines and programming languages. But like we were all very much on board with this idea of what we call the deconstructed database. So building these modular components for solving all of the problems, kind of the first and second order problems, that go into building a high performance data application that deals with tabular data.

It's not enough to simply have the data format and serializing it and handing it over the – Throwing it over the wall to another application. We needed all of these other pieces to make it possible to build systems that are Arrow native. In recent times, the work has shifted to now that we've established like the core, the Arrow columnar data format, the wire protocol, we created an API for connecting applications at the C level, which is in memory, interoperability between different library components much easier.

But the ecosystem has shifted towards building very engines and memory computing tools for doing high performance processing and Arrow data. So we're starting to see like next generation databases and query engines being developed that are designed as getting error data from the ground up. So seven years ago, when we were getting the initial group of open source developers together to start the project, these were things that we discussed, and these were part of our ambitions. But it's taken many years to realize those ambitions. In many ways, even six, seven years into the project, it feels like we're just getting started.

But certainly, back to your original point, many people do think that Arrow is being just another file format. So we've had to invest some energy in dispelling that kind of conception or understanding of the projects to show like how diverse and kind of wide reaching the contract has become over time.

[00:04:08] JM: I think probably some people listening are unfamiliar with Arrow, so it's worth backing into an explanation of it. In a world where there's already so many large data components, data warehouses like Snowflake or Redshift, there's systems like Presto that function as large scale query engines, there's high volume analytic systems like Pinot and Druid, there's Spark, obviously, it seems like there's already enough large data infrastructure

upgrade projects going on in the enterprises that you would be selling to. How do you convince a large enterprise, “Hey, here's another big system that we recommend you use.”?

[00:05:03] WM: I guess we can step back and talk about like Arrow specifically as a technology and how it's different from many of the projects that you mentioned. So if you go back to the pre-error days, so the world before 2016, traditionally, analytic database systems, big data systems, distributed computing systems are these very vertically integrated systems where you have SQL query comes in. That query is parsed and analyzed. A query plan is constructed. Then that plan is optimized and handed over to an execution engine.

Inside the execution engine, you have all of the algorithms and the data crunching to take input data from storage and compute the query results. So the developer of the database system would develop hash tables and sorting algorithms and all of the like function implementations, column expressions, array expressions. All those things that you would need to execute to evaluate a business user SQL query.

Then on the storage side, historically, even the storage system, the storage layer for the database was vertically integrated. So you would ingest data into the database system, and then that data would get converted into the databases like proprietary data formats, and then indexed so that the queries would run faster against it. So what began to happen in the late 2000s was that there was this move toward what's called disaggregated storage, and that meant that there was a decoupling between the query engines, the analytic query engines, and the storage.

That was what initially what to do, and to move towards like data lakes and cloud storage was all about for now you have systems like Presto, for example, or WhatsApp, and Microsoft coined this term a poly engine. So this idea of like a query engine that can query many different data formats, typically open source data formats that have been developed, as well as like data formats that have been around for forever like CSV files or JSON files.

Having the ability to have a query engine that can just query any of your data wherever it is, particularly, even on curated data with Presto or Spark SQL, gives enterprises a lot of flexibility with how they can do large scale data management. But if you think about this vertical

integration that you have with traditional database systems and query engines, it's pretty inefficient because developers are rebuilding the same things over and over again.

So we aren't selling other open source projects and enterprise customers on yet another vertically integrated system. It's more that we are focused on achieving horizontal integration in the stack, so creating reusable computational building blocks for solving many of the problems that were traditionally solved vertically integrated systems. That includes in memory columnar query processing, building distributed systems that involve moving large quantities of data over the wire, dealing with open source file formats. Things like Parquet and ORC doing high-performance connectivity into databases and storage engines.

I think what's motivating some this for us is because the reality for modern enterprises is that data work is polyglot. So it's not just a SQL work in a SQL world anymore, SQL world anymore. There's an increasing amount of data engineering and data work that's happening. End user data science languages like Python and R and Julia, as well as systems languages, Java continues to be very popular, but you're seeing an increasing amount of Rust and Go out in the wild. So it's simply not efficient to have every programming language like solving all of the same problems again and again.

Instead, we're creating these modular optimized components for solving all these different problems that are commonly occurring in building data platforms, making them consumable across programming languages so that we can consolidate efforts toward creating really high quality and reusable software components that are built for kind of this polyglot, less vertically integrated future, if that makes sense.

[00:09:36] JM: It does make sense. I want to dive a little bit deeper into the question of in-memory systems versus on-disk systems. To my knowledge, if I recall correctly from my previous interviews on Arrow, a typical use case would be you have maybe a Spark program that's using the JVM, and you build a really big data structure in Spark in the JVM. Then maybe you want to hand it off to a Python program for evaluation, and you would like to have the data structure be serialized in the same way so that it can be operated on in the same way. Maybe you can correct me if I'm wrong, and you can maybe generalize. Why is it a data interchange system important for this kind of analytics?

[00:10:32] WM: Right. So I guess let's just focus on this particular example that you gave. So one of the first projects that we worked on when we were starting the Arrow projects, indeed, was integrating Arrow with Spark for this specific purpose of accelerating, making Python run faster on top of Spark SQL, so making it more efficient to plug custom Python code into Spark. Essentially, you can use pandas or you can use NumPy to express custom algorithms that run inside Spark SQL.

But something to keep in mind is that Spark SQL does not use Arrow internally. It has its own Spark-specific custom row-oriented data format. They've developed some like columnar, column-oriented caching for Spark SQL. I don't – I'm not familiar with the exact technical details. But Spark SQL principally has this custom row-oriented format that its algorithms execute against. So whenever you need to plug in some custom Python code, that data format would need to get sent over across the language boundary from the JVM into Python and then converted or serialized into NumPy arrays or its pandas DataFrames so that you can execute your function, which uses pandas DataFrames, for example.

What we did was we developed a very fast – Well, we developed the Arrow specification interchange format, and then we built very fast adapters between Spark SQL's data format on the JVM side. Then on the Python side, between Arrow and pandas and NumPy, in some cases achieving zero copy, like don't need for serialization. So we enabled Spark SQL and Python to handshake using Arrow purely as an intermediary but still having to convert between Spark SQL's internal data format and Arrow and then between Arrow and Candice.

Even with all of that converting from Arrow happening to move to using Arrow as the interchange format, yielded several times to even, in some cases, 100 times performance speed up in some of those workloads. There's still a lot of serialization going on. So one of our motivations with kind of how we're thinking about the Arrow project and building out the ecosystem is that what if we didn't have any of that conversion at all and if future systems, so not Spark, necessarily, but systems that are functionally like Spark in the sense of being a distributed data processing engine were built as Arrow native from the ground up?

So your primitive unit of data is chunk of a table that's in this efficient columnar format that we've designed and where data between among across a distributed system or across programming languages that are participating in that distributed system. For example, a user's Python code or a user's Rust code or Java code, without needing to convert between data formats. So that's really what we're interested in long term is enabling that kind of polyglot, scalable, distributed computing without the need to serialize because what's happened is that as networking and storage has become more efficient and more applications, particularly distributed data applications, are becoming increasingly CPU-bound and less IO-bound, so delivering data to the system is not as much of a bottleneck. Now, the bottleneck is processing and pre-processing the data and getting it ready, feeding it into the algorithm.

Serialization has taken on a much larger fraction of the total runtime of these distributed big data programs. And so by reducing serialization and data conversion to a minimum is essential for enabling kind of next generation performance in distributed big data processing.

[00:14:34] JM: Okay, great. Could you go through a prototypical use case of how – Just to really make this concrete and maybe an example of a kind of an application that would want to use Arrow. Then maybe you can talk through why the open source project or what the differentiation between the open source Arrow project versus what you're building in Voltron, how those two differentiate?

[00:15:08] WM: Sure. Let's talk through one popular use case for Arrow right now and some of the ways that people are using the project. So Arrow has been really successful as a bridge between the ecosystem and data science tools in people's data lakes. For example, suppose that you have a data lake which lives in Amazon S3 or Azure Blob Storage or Google Cloud Storage. It is constituted principle. We have Parquet files, which live directory structure at the data lake. So we provide high performance access layer for those datasets that are stored in the data lake.

Because of the way that we designed the Arrow project, each of the components is modular and compostable. So you can, for example, use DuckDB, which is a kind of in-memory analytics SQL engine. So you can take a dataset which is being accessed through kind of **[inaudible**

00:16:07] of the data lake. It's being accessed through the Arrow libraries. We're handling credit pushdown filtering column selections, as well as the metadata of all the files in the data lake.

Then we can see the results of reading that data set as they're streaming in into DuckDB, which has implemented the Arrow in-memory data interface, so it can natively execute against a real time stream of Arrow chunks, like error batches of data. But there's an increasing number of components that are developing for composability with the Arrow interface.

For example, within the Rust Arrow project, there are some project called DataFusion, which is also an in-memory embeddable SQL analytic query engine, and that can also be composed and used as an executor for running, for executing data transformations, analytics SQL workloads, like that sort of thing.

But results can also be written back. From within Python or R using the Arrow libraries, you can read a data set, do some transformations on SQL or with data frame APIs, and then write the results back into the data lake. There's a lot of users which are actively using that to do data engineering work. We expect to see more of that over time.

As far as we're open source ends and Voltron data begins, as a company, and you can see this if you look at our contributions to the Arrow project and peripheral ecosystem projects centered around Arrow like Substrate and **[inaudible 00:17:41]** and things like that, we're really focused – Our primary focus as a company right now is growing the Arrow ecosystem and working to partner with other open source projects and companies and software projects that are in the peripheral orbit around the Arrow ecosystem.

So this includes database vendors, storage vendors, working to help guide the ecosystem through its adoption of Arrow in their APIs, in their interfaces so that we can enable more and more systems to support Arrow more natively because we want to increase the number of systems that can send and receive. Arrow is a preferred input output interface because we believe that the more system support Arrow and send and receive Arrow, that brings a great deal more value to systems that provide accelerated processing capabilities for the Arrow data format.

Some systems I mentioned like DuckDB and DataFusion, which are already Arrow native, as well as RAPIDS, which is GPU acceleration for GPU-accelerated analytics for Arrow. So we want to enable the processing components, which are already working across different types of hardware, CPUs, and GPUs, to be used in a very natural way with data, no matter where it's coming from, whether it's coming from a database system or a data lake or some other kind of storage system so that we can reduce the amount of impedance that's associated with crossing between different storage systems, processing components, programming languages, that sort of thing.

Our focus as a company is really accelerating that adoption cycle, given that I think a lot of the success that we've seen at the Arrow ecosystem has been through very organic means. So we've been – I for many years was focused on building the open source project and working as best as I could with a small team to help unblock people who are looking at initial kind of early integrations of Arrow into their systems.

But we reached the point kind of in looking at the growth ecosystem where the absence of an enterprise software company providing direct partnership, enterprise support, essentially being a development partner through significant engineering projects that involved Arrow to the absence of the company providing that kind of peace of mind hold back growth in the ecosystem, if that makes sense.

[00:20:09] JM: It does make sense. So if I kind of repeat my previous question, maybe can we talk about like a specific domain, where maybe you don't want to talk about a particular customer? But if you could talk about a use case, like a high level use case, I just wanted for the listeners to have something in their head that they can comprehend. A use case where data interchange and the latency imposed by that data interchange without a common data format would be particularly prohibitive.

[00:20:46] WM: Right. So let's focus on, let's say, a machine learning application, which is written in Python that uses PyTorch or TensorFlow, and the training and scoring of that model is dependent on our pipeline data access and pre-processing feature engineering tasks that involve data retrieval from data lakes, in-memory processing, both things that can be expressed

in SQL, as well as things that require custom logic, which is written in Python and uses APIs from NumPy or pandas or from other Python libraries.

What the Arrow project is providing these types of applications is a lot of it is accelerating like data access and like that heavy lifting of like how applications are efficiently obtaining the data that they need to carry out their work. So that's providing like a new path that's been engineered to be a lot more efficient, take better advantage of modern hardware, getting more mileage out of what modern CPUs can do, better vectorization, and just faster data loading and processing.

When it comes to I think some of the interoperability benefits, they occur between programming languages or different steps in a processing pipeline, so just going back to what I was saying before where giving the DuckDB example. So DuckDB is an analytic SQL engine that runs in process. So if you wanted to use DuckDB to express one stage of your data transformation in your pipeline or getting data ready for a machine learning application, we've done works directly with the DuckDB projects to develop a very efficient streaming interface between the DuckDB query engine and the Arrow data format.

Any kind of earlier steps in your processing pipeline, which may involve simply reading Parquet files or doing some in-memory transformations of incoming stream of Arrow data, that can be fed directly into the SQL engine with effectively no impedance. That's hugely beneficial in terms of just analytical efficiency and being able to jump between these different processing components that address different needs in the overall application, the overall data pipeline, where traditionally you would have like a bunch of data conversion that's pretty expensive.

It’s not uncommon to see applications spending 80 or 90 percent of their time on data serialization and converting between different data formats because whenever you switch between a different processing component or a programming language or a library, like often kind of in the spirit of vertical integration, like I was describing before, that systems have their own internal data structures and algorithms, which are the data has to be converted into particular format in order to use those algorithms. Whereas when applications are built on Arrow, they don't have that impedance mismatch.

[00:23:56] JM: Great example. If we talk about the actual ongoing development of Arrow, I think about the work that needs to go into a project like Arrow, I think about systems to serialize and deserialize stuff into the Arrow format and to maybe modules to make programs in different languages consume, all be able to work with the Arrow project. I think a lot of that stuff has been in place for a while since the Arrow project has been in place for a while. What's the ongoing R&D for Arrow look like?

[00:24:45] WM: Yeah. There's a few different areas. So about halfway, so about midway into Arrow's lifecycle since like 2018, 2019, we have not yet developed like a standard toolkit for building distributed systems that send and receive error data over the wire. So the problem with that is that you end up having fragmentation because every distributed system that wants to use Arrow and use Arrow to communicate natively between different nodes of the distributed system, like they would implement their own RPC system or embed Arrow into their custom kind of RPC layer, which maybe is based on Protocol Buffers or Thrift or something like that.

What we did was we developed something called Flight, which is a standardized RPC. RPC is just to turn the remote procedure call, and it just describes like a protocol for developing clients and servers that can connect to each other or send commands and move data. So we developed Flight to provide like an out-of-the-box wire protocol and client server framework for building distributed systems that are Arrow native. We built implementations of Flight in many programming languages, Java, C++, Rust, Go, for example. We've developed a set of middlewares for Flight to enable light to be used as an interface to SQL systems. That's called Flight SQL.

The idea or the way you can think about it is that many database systems have their own custom wire protocols. Like there's the PostgreSQL wire protocol. MySQL has its own wire protocol. Many databases have their own custom wire protocols. Those wire protocols are often not very efficient. They're usually not columnar. They're row-oriented, and the result is that it's very inefficient to export large amounts of data from a database system. So what we're working to do there is making it simpler for database systems to produce Arrow on the server side and deliver Arrow directly from the database to the client.

That not only makes their wire protocol significantly more efficient because they're moving data in these column-oriented batches. But then clients also don't have to go through the whole rigmarole of converting between the databases in efficient row-oriented wire protocol. They're fast, tabular column-oriented data structures. They're pandas type libraries, data frame libraries, and that sort of thing.

We've done a bunch of design and development for those goals. We have a closely related project, which is ongoing right now called Arrow Database Connectivity, which sounds devilishly similar to Flight SQL but is more about API standardization for how database drivers can incorporate Arrow into their driver interface. So they don't necessarily need to adopt our wire protocol and what we've developed with Flight in Flight SQL. But instead, like they can use their custom wire protocol and deliver Arrow data to clients in whatever fashion they deem appropriate for their system.

But at least we can achieve a measure of API standardization on the client side so that you don't have to write different code to access each different database system that can produce Arrow for you. This is an important problem because we've already seen a number of database systems and data warehouses incorporate Arrow into their client server interfaces. So for example, Snowflake has integrated Arrow into their Python connector, Java connector, BigQuery. Google BigQuery has incorporated Arrow into their client interface, so you can request results of SQL query that BigQuery executes come back to you by an Arrow format.

But this is all happening through like very bespoke needs, and either you have to write different code to access Arrow data coming from Snowflake versus Arrow data from BigQuery. So we'd like to achieve interchangeability of these database drivers to make things simpler for the downstream consumer applications. Those are some major efforts that are going on the connectivity side of Arrow.

I would say that more and more, larger and larger fraction of Arrow development R&D within the Arrow project itself is happening in the actual query processing and execution. So that stands not only the development of in-process and distributed query engines. There's multiple query engine projects that execute both in like in-process embedded use, as well as in distributed mode. But also concerns like computing primitives, so like the algorithms that you need to do

either in memory or out of court, sorting hash table operations that are associated with like aggregations or joins, the nuts and bolts that you would need to build effectively database engines.

There's work happening along those lines in Rust, very kind of so many years of work put into doing query engine development for Rust projects. They're called DataFusion, which is an embeddable query engine and Ballista, which is a distributed scheduler, distributed executor that is designed for use with DataFusion, but also has been developed along with the Arrow modularity mantra of using flight as the kind of distributed systems like connectivity layer and providing for modular execution engines like the single node task level.

Really, like we're building these systems to be the foundation for people to build next generation databases and data warehouses. One of the really active parties in the Rust side of Arrow is influx data, and so some of the listeners will be familiar with InfluxDB, which is a time series database, which has been around for many years. Originally, it was written in Go. So some years ago, maybe two or three years ago, Paul Dix, the CEO of InfluxData, wrote a blog post about how InfluxData was betting on Apache Arrow and decided to migrate from Go to Rust as their systems language. So they had built a team and have been working actively to build their next generation InfluxDB called InfluxDB IOx based on DataFusion and Rust Arrow, which has been really exciting.

Really, like this is exactly, I think, like the story of InfluxDB and what they're doing with DataFusion and their next generation version of InfluxDB IOxes, exactly what we've been hoping to achieve with the Arrow project, which is to build these reusable processing components that are achieving rather than a vertical integration, you can think of it as more like horizontal integration, where we can consolidate effort in building these systems that saw particular layers of the stack of what you would ordinarily see in a database system and enabling downstream software projects to build on that, rather than forking or building custom components that are only ever used to solve the problems within their walls, if that makes sense.

But it's been a pretty massive effort. We're approaching like 1,000 unique contributors to the Arrow project. Obviously, like we've had to invest a great deal in developer productivity, CICD

packaging, just supporting. To have an open source project that can support 1,000 unique developers is a massive undertaking in and of itself, and we also have to invest significantly in education about what Arrow is. It's more than a data format. There's all these different components that solve different problems. So that not only we can continue to grow the developer community and make the project accessible to new contributors, but then also like the downstream users can understand the different dimensions of the projects, and how they're supposed to use it, and how they can kind of participate in the ecosystem and be successful.

[00:33:03] JM: So when I think about InfluxDB, they own their own stack, and they can write their entire database in Rust. When I think about what the operations of the database look like, it's like having large amounts of time series data in memory and writing large amounts of time series data to disk. Maybe this is a really naive question, but you can help me understand why a project like Apache Arrow makes such a difference to a database that owns their entire stack.

[00:33:42] WM: Right. I think it comes down to picking your battles. So like let's just use the InfluxDB example. If you look at InfluxDB like prior to Arrow, the original GO language-based database, which is a vertically integrated system that includes file formats, like a file format for storing time series data, indexing data structures, code to build, and manage indexes, a full query engine, including relational operators, query optimization, parsing, planning, primitive algorithms for processing the data, as well as the front end user interface, client interface, a wire protocol, distributed system for distributed execution, there's like many layers to building a successful system.

What a project like InfluxDB is achieving by working with Arrow, well, first of all, they can collaborate and work with a much larger group of developers on some of like the core primitives, select the query processing, query execution primitives, so like the query optimizer and the relational operators, so like the algorithms that are used to compute joins and sorts and like the essential algorithms of the database engine. There's a much larger community that simply InfluxData that is invested in improving the performance and efficiency of that layer of the stack.

Rather than owning the whole stack, they're focusing on, well, firstly, they developed new active development and are very active in working on the execution engine. But they're working with a much larger group of developers who are using that same execution engine to build other types

of query processing systems. So that enables InfluxDB to focus more of their energy on the thing, the aspects that provide them with competitive differentiation. That's their storage and indexing layer and the distributed coordination, some of the particular things that are specific to how InfluxDB is used to capture store query time series data.

Kind of the goal, and this is true across the many other projects that are currently building or in various stages of building on Arrow, that the goal is kind of to reduce the dimensionality of the space to simplify the problem for software developers, while also improving outcomes because they have all these off-the-shelf components that are really high quality. So I guess the thesis is that in the longer term, the components that exist within the Arrow stack are going to be as good or better than the thing that you can build all by yourself, starting from scratch.

I believe that the result of all this is that we will see the acceleration of innovation in the field of database systems and making it easier for people to build different kinds of special purpose data warehouses or different database systems without having to build a whole new query engine from scratch or a new parser plan or optimizer from scratch. One project that we have that's adjacent to Arrow, but it's a lot of the same people working on it, is called Substrate. It’s maybe a little bit esoteric for some people in the audience, but it's an intermediate representation for analytical queries, and the idea is that different systems have different SQL dialects.

That can make like portability across database or query processing systems difficult because you have to figure out how to like port the SQL like with one dialect and another. That makes it difficult to make the query engines themselves more modular and interchangeable. So we’ve developing this intermediate representation called Substrate, which enables us to better modularize the execution engines so that we can do better decouple the frontend like user interface layer of the analytical system from the back end, like storage and query processing, if that makes sense. So it's like a tool in our mission for modularization and better horizontal integration, as opposed to vertical integration.

[00:38:05] JM: Take me into the future of databases. Like you've spoken a couple of times about the future of databases being potentially powered by Arrow. What would necessitate? Or when you look at the database in the data warehouses of today, what are the shortcomings

there, and what can you envision in the future that would be lower latency or new kinds of queries? What do you think will be enabled by something like Arrow?

[00:38:36] WM: Well, there's a few things. So one thing that we're working on as a company **[inaudible 00:38:41]** is we want to enable – Let's call it a race to the bottom in terms of computing efficiency on the part of the hardware vendors. So like we want X companies that have ARM or X86 chips to be invested in making the core data processing algorithms as efficient as possible on their silicon.

Just to give an analogous example, you may be familiar with the Intel Math Kernel Library, which is Intel's – A suite of optimized linear algebra that's been – For many, many years, it's been optimized to get the most out of Intel architecture using SIMD instructions and taking advantage of memory bandwidth and caches and all those sorts of things. So by having the ecosystem in increasingly standardized Arrow is in memory processing format, in addition to being a fast interchange format, then that gives the hardware vendors more of a stable target to optimize.

So investments of engineering time doing SIMD, whether it's ABX optimizations for X86 or neon optimizations for ARM or GPU optimizations that are You know CUDA or ROCm for NVIDIA or AMD GPUs, respectively. That those investments that are made on the part of the hardware vendors have long-term staying power, such that they can be justified, rather than having every database vendor or every system developer having to take responsibility for getting the most mileage, the best efficiency out of the hardware that they're using. That it can be more of a partnership between parts of the hardware and software sides of the industry.

I think this will free up like a lot of developer cycles in the database ecosystem that focus more on usability and integration. Like better integration at the language levels make individuals more and more productive. So rather than having the database space like kind of get a shooting match with each other about who's SQL engine is faster than the other, you might have recently seen the competitive blog posts between Snowflake and Databricks about who can run TPC-DS in the cloud faster.

I think that having comparatively more investment in developer and user experience, language integration, better language support, more natural access to high performance query engines within languages like Python, I'm excited to see more of that work being more focused, rather than having this or that vendor, just competing over who could run SQL queries the fastest.

[00:41:24] JM: Well, this has been a really good conversation about Arrow and applications thereof and some nice visions into the future. I guess I want to wrap up by getting a little bit more of a picture of the company, what you're working on right now and I guess the roadmap. Like what do you expect to be working on like, assuming this is a massive, super successful company in 10 years? Imagine it gets as big as Databricks or something like that. What will you have accomplished in 10 years? What will you have built?

[00:42:01] WM: Our hope is to make the stack, the ecosystem of large scale data processing tools substantially faster and more efficient, as well as simpler and easier to use. We're really bullish on a future that is not only polyglot, but also in terms of programming languages, but also polysilicon, to being able to leverage GPU acceleration in a more natural way, as well as we can start to think about using FPGAs and custom silicon ASICs in data analytics. So one of our goals in modularizing these different layers of the stack is to facilitate that migration towards more seamless hardware acceleration and data analytic applications.

I think as a company, we're really long-term-minded in terms of these technology, events, and evolutions. As I was saying earlier, we're really putting a lot of energy into our Arrow open source partnership program. We call it kind of Voltron data enterprise subscription for Arrow, building relationships with companies that are betting on and contributing towards a future that's more Arrow native so that we can accelerate the adoption cycle and also continue to work and identify like what are the barriers to enabling that evolution to take place.

So we're building software that we want to be ubiquitous and be a picture with an analytical processing systems everywhere into – Yeah. So essentially, to be an important part of not only making systems faster and more efficient, use less carbon, maybe decrease the carbon footprint of running our SQL queries, but also improve the usability of the stack and to kind of along the lines of futures polyglot. So we have to give every programming language its due

consideration and being usable in terms of being able to leverage high performance analytics in a natural way within the developer's workflow.

For me, I mean, I've been working in data technology for 15 years now, and you've really been really passionate about all these things. So for me, it feels like a continuation of work that I started years ago in Python with Candice and making data science simpler and more accessible. Now, I think the frontier is how we can leverage innovations in 2D hardware to bring about kind of a new wave of simplification and improve performance and efficiency. So I believe that we've created an organization that can bring about that big transformation through the years.

[00:44:48] JM: Cool. Well, Wes, thank you so much for coming on the show.

[00:44:51] WM: Thank you. Thanks for having me.

[END]