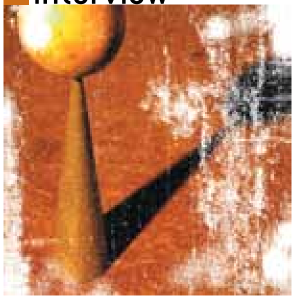


A Conversation with Arthur Whitney

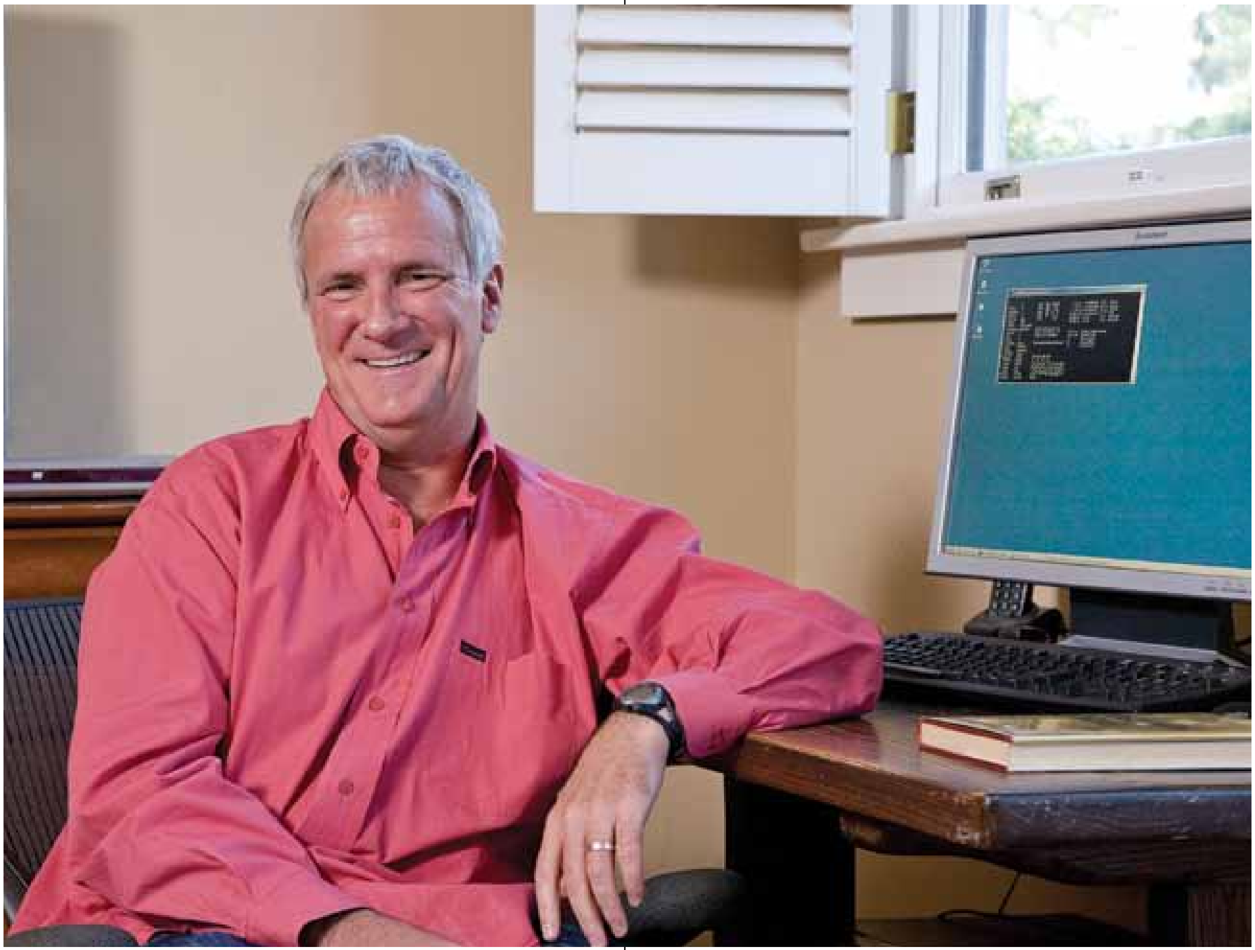
Photography by Tom Upton

A few well-chosen words about programming languages from a long-time designer

When it comes to programming languages, Arthur Whitney is a man of few words. The languages he has designed, such as A, K, and Q, are known for their terse, often cryptic syntax and tendency to use single ASCII characters instead of reserved words. While these languages may mystify those used to wordier languages such as Java, their speed and efficiency has made them popular with engineers on Wall Street.

Whitney began his Wall Street career in the 1980s, building trading systems at Morgan Stanley using his own version of APL (the language on which all of his later languages are based). Eventually he started his own company, Kx Systems, which today provides realtime and historical data-analysis software to many Wall Street investment banks. The company's signature product, KDB+, is a column-oriented database based on the K language.

Eager to learn what's behind Whitney's unique languages (and curious to see if his reputation for concision carries over into real life), we invited him to speak with



Group Term Life Insurance**

10- or 20-Year Group Term Life Insurance*

Group Disability Income Insurance*

Group Accidental Death & Dismemberment Insurance*

Group Catastrophic Major Medical Insurance*

Group Dental Plan*

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan***

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh Affinity Group Services at **1-800-503-9230** or visit **www.personal-plans.com/acm.**

*Underwritten by The United States Life Insurance Company in the City of New York, a member company of American International Group, Inc.

**Underwritten by American General Assurance Company, a member company of American International Group, Inc.

***Coverage is available through Assurant Health and underwritten by Time Insurance Company.

AG5217

MARSH
Affinity Group Services
a service of Seabury & Smith

Queue editorial board member Bryan Cantrill. Cantrill is best known for developing DTrace, a tool for dynamic instrumentation of production systems that helps companies identify and fix performance bottlenecks. Whitney was gracious enough to invite Cantrill to his home in Palo Alto, where they spoke about his career, his languages, and the essence of elegance.

BRYAN CANTRILL You are a bit of a rarity in software engineering in that you have been writing software on a daily basis for decades. Your first introduction to computing was APL with the master, Ken Iverson. What was that like?

ARTHUR WHITNEY In 1969, I was 11, and Ken Iverson was at IBM Research in Yorktown. He had been a friend of my dad's at Harvard in the '40s. We lived in Alberta, but we were driving around the continent and went to visit him. He showed me programming on a terminal in his house in Mount Kisco. This was in the '60s, and already it was interactive, and it was very quick to write programs and get results.

BC You must have been the only 11-year-old on the planet getting that kind of demonstration of programming in 1969.

AW Of course, I had no idea about that, and I didn't really pay much attention. He showed me some stuff, and I thought it was cool. In '74 when I went to a university and took a computer class, they were using punch cards, which made no sense because five years earlier I had already seen interactive programming.

BC Did you start working on APL at Waterloo?

AW No, at Waterloo I just did some APL for a week. I was a math major and I wasn't interested in computers because I just wanted to do pure math. So I really missed a big opportunity.

BC Well, I'm not sure if you missed it or if you just found the opportunity a different way.

AW It took me a long time. For the next 10 years I did a little bit of APL in the summers as a consultant, but it wasn't until about 1980 when I was working with Ken at a Canadian company called I.P. Sharp that I really began using it regularly. Ken had retired from IBM after 20 years and was working at I.P. Sharp in Toronto.

I.P. Sharp was an amazing company. It had its own worldwide network that had nothing to do with DARPA (Defense Advanced Research Projects Agency). We were sending e-mails and instant messages to Australia and Singapore. The whole company was APL.

BC They were selling APL time sharing, right?

AW Yes, and it was easy because the one computer in Toronto was running the entire world.

BC What kinds of problems were people using the APL time-sharing service for?

AW It was mostly general-purpose business computing, such as accounting systems. I did a 2-billion-row database, so we were doing very big databases and data analysis—what today, 20 years later, they call OLAP (online analytical processing).

I left I.P. Sharp sometime around 1980. Then I went to graduate school at the University of Toronto where I did pure mathematics, but mostly I was just goofing around. All through the '80s I was implementing my own languages: object-oriented languages, a lot of different LISPs, Prolog. In 1985 I got a job at Stanford, where I implemented a Prolog inference-engine kind of language. Then I was with an artificial intelligence company called Teknowledge.

BC Were you developing these languages because you needed a certain expressive power in the language to solve a particular problem at hand? What were the motivations for these languages?

AW My motivation was always to create a general-purpose programming language that would solve all problems and be interpreted, but fast.

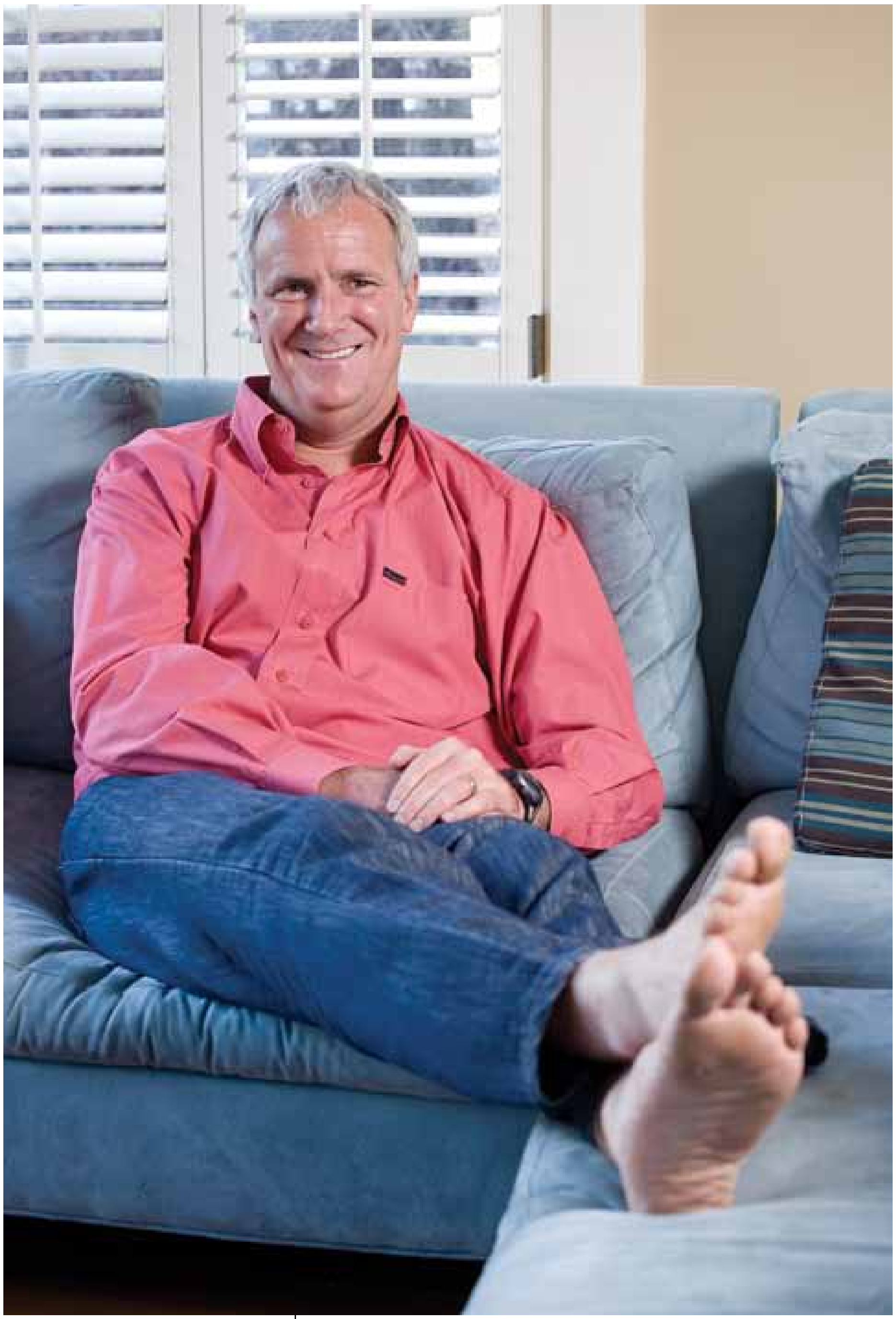
At Stanford the language was determined by the professor, and he wanted to have an inference engine, so the motivation there was artificial intelligence, but I wasn't much interested in that.

My big break was in 1988 when I joined Morgan Stanley. There the motivation was a terabyte of TIC (Treasury International Capital) data, and back then there were a few million transactions a day being processed by realtime trading systems. I think we had one of the biggest trading operations in the world. We had a portfolio that was a billion dollars: half a billion long, half a billion short. We were trading every second electronically. The data set was a terabyte, but we compressed it down. It was pairs trading, and I wrote an APL to do all of that—the big database and the realtime trading—so our entire department was using my language.

BC You had used APL, and then you explored these other languages—Prolog variants and so on—but when you got to Morgan Stanley you came back to APL. What brought you back?

AW I much preferred implementing and coding in LISP, but once I was dealing with big data sets and then having to do fairly simple calculations, APL just seemed to have the better vocabulary.

It had to come up one level. Common LISP even then had about 2,000 primitives. I didn't like that. What I liked was the original LISP, which had `car`, `cdr`, `cons`, and `cond`, but that was too little. Common LISP was way too big, but a stripped-down version of APL was in the middle with about 50 operations. It's about the same size as C. But the thing about the languages that I implement is that there are no libraries: those 50 operations are it. Everybody builds from there, and the resulting programs are extremely short.



BC There the problem did serve as a motivator. You had this massive amount of data, and you needed a language that could deal with that large amount of data in a first-class fashion. Did other people around you see the expressive power, because even at that time I would assume that APL was beginning to wane a bit?

AW APL peaked in the '70s, but in the finance industry APL was very strong, so there was no difficulty in doing my own APL version.

BC How did your own APL differ from the original? Did you change the primitives that were being exported?

AW The primitives were a little different; the grammar was pretty much the same. The syntax was the same. The vocabulary was very similar, but not enough to be anything close to portable.

BC I'm sure that practitioners who know APL only by reputation are going to wonder if it used the same wonky characters as the original APL.

AW Yes, at Morgan Stanley I did use the APL characters, but on my next iteration, K, which was in '92, I gave up on those characters.

BC Why did you give up on them? And how did you feel about giving up on the characters?

AW Well, it felt great because it was easier to send e-mails. They're beautiful characters, but I had to strip the language down. K today has no reserved words; it just uses the ASCII keyboard. It's completely arbitrary, but it makes me keep the language small.

BC You speak about the arbitrariness in using the ASCII keyboard. I heard one feature being described as this: "When Arthur ran out of punctuation, he used a leading underscore to denote system primitives." When I read that I thought to myself, "That's a little ridiculous," but then I thought of all the goofy punctuation characters we have in other languages: C uses nearly all of them; many languages use the balance. And we use them in different contexts and different ways.

AW Certainly it's unfamiliar, and people say, "Oh, it looks like line noise." But even kids can learn this quickly.

BC Obviously, a point of pride for K is the ability to phrase things concisely. Is there any length that is too short, where you've actually squeezed too much information out in terms of its readability?

AW Yes, and I expect I cross that boundary a lot. But if every line has up to seven operations, then I think that's manageable. In fact, we can remember seven things.

BC Right. People are able to retain a seven-digit phone number, but it drops off quickly at eight, nine, ten digits.

AW If you're Cantonese, then it's ten. I have a very good friend, Roger Hui, who implements J. He was born in Hong Kong but grew up in Edmonton as I did. One day I asked him, "Roger, do you do math in English or Cantonese?" He smiled at me and said, "I do it in Cantonese because it's faster and it's completely regular."

BC This raises an interesting question. When I heard about your early exposure to APL, a part of me wondered if this was like growing up with tonal languages. I think for most people who do not grow up with a tonal language, the brain simply cannot hear or express some of the tone differences because we use tone differently in nontonal languages. Do you think that your exposure to this kind of programming at such a young age actually influenced your thinking at a more nascent level?

AW I think so, and I think that if kids got it even younger, they would have a bigger advantage. I've noticed over the years that I miss things because I didn't start young enough.

BC To ask a slightly broader question, what is the connection between computer language and thought? To what degree does our choice of how we express software change the way we think about the problem?

AW I think it does a lot. That was the point of Ken Iverson's Turing Award paper, "Notation as a Tool of Thought." I did pure mathematics in school, but later I was a teaching assistant for a graduate course in computer algorithms. I could see that the professor was getting killed by the notation. He was trying to express the idea of different kinds of matrix inner products, saying if you have a directed graph and you're looking at connections, then you write this triple nested loop in Fortran or Algol. It took him an hour to express it. What he really wanted to show was that for a connected graph it was an or-dot-and. If it's a graph of pipe capacities, then maybe it's a plus-dot-min. If he'd had APL or K as a notation, he could have covered that in a few seconds or maybe a minute, but because of the notation he couldn't do it.

Another thing I saw that really killed me was in a class on provability, again, a graduate course where I was grading the students' work. In the '70s there was a lot of work on trying to prove programs correct. In this course the students had to do binary search and prove with these provability techniques that they were actually doing binary search. They handed in these long papers that were just so well argued, but the programs didn't work. I don't think a single one handled the edge conditions

"If you can find a shorter, more elegant program that isn't much slower than my code, I want to hear about it."

—Arthur Whitney

correctly. I could read the code and see the mistake, but I couldn't read the proofs.

Ken believed that notation should be as high level as possible because, for example, if matrix product is plus-dot-times, there's no question about that being correct.

BC By raising the level of abstraction, you make it easier for things to be correct by inspection.

AW Yes. I have about 1,000 customers around the world in different banks and hedge funds on the equity side (where everything's going fine). I think the ratio of comment to code for them is actually much greater than one. I never comment anything because I'm always trying to make it so the code itself is the comment.

BC Do you ever look at your own code and think, "What the hell was I doing here?"

AW No, I guess I don't.

BC Wow! I confess that I tend to write comments for my future self. I know that when I come back to code I've written, I often don't recall instantly what the problem at hand was or how I solved it. Now you've got me thinking that maybe I'm just in the wrong language. When you're at this higher level of abstraction, maybe it's easier to see your intent.

In terms of debugging your code, obviously the power of a terse language such as K or Q is that, presumably, it's easier to find bugs by inspection. How do you debug them?

AW In C I never learned to use the debugger so I used to never make mistakes, but now I make mistakes and I just put in a print statement. K is interpreted, so it's a lot easier. If I'm surprised at the value of some local at some point, I can put in a print, and that's really all I do.

BC That works well when you have deterministic inputs. What if the nature of the problem is just less reproducible—for example, if you were in an event-driven system where you had a confluence of events that led to a problem?

AW It has been 20 years now that I've had Wall Street customers—they're doing 2 billion transactions a day and they have trillion-row databases—and in those 20 years, there was one time where we couldn't reproduce the bug. That was nasty. I knew the kinds of operations that they were doing and I finally found it by just reading my code.

BC Was this a bug in K or Q, or was it in the C base implementation?

AW It was a bug in C, in my implementation.

BC Is the nature of the problems that K and Q solve such that you just don't have nonreproducible problems?

AW It seems ridiculous, but it's only recently that we've been doing multithreading, so I guess we might start to see things that are much harder to reproduce. Of course it has been event-driven since 1988. I don't know why it is, but it has always been the case that people can quickly find a tiny script that will show the problem.

BC I think it's fair to say that you've written a lot of flawless code.

AW Yes. I went millions and millions of hours with no problems—probably tens of millions of hours with no problems.

BC That's a relief to hear because it seems that societally we have come to accept bugs as being endemic in software. When you're talking about the program being its own proof, I think it gets to the fact that really these programs are much more like proofs. A proof is either correct, or it's flawed: there's no middle ground for a proof.

AW I want to see if I can get better. Kx is doing fantastic, and it takes just a few hours a month for me, so now I have a clean slate. Every few years I have to do a new language, but the customers don't really like that.

BC Q was the last iteration of that process. What are some of the differences between Q and K?

AW K was all symbolic. It was 20 symbols with a prefix and an infix meaning. With Q, the idea was to have all the monadic cases be words. So now infix are the symbols and prefix are the words.

BC This gives it what you call the *wordiness*—I think what others might call *readability*. For those who are not in that world, will a Q program look more readable than a K program?

AW Absolutely, because a lot of these symbols are familiar to people from other languages—plus, minus, times, greater than, less than. If they're looking at a K program that's using all 20 of them, they will know a half or a third of them, whereas if they're looking at a Q program they will know about two-thirds of them.

BC How important is the readability to the uninitiated?

AW From a sales point of view, I think it has helped a lot. For someone who programs a few hours a week, I don't think it would make any difference once they learned K or Q.

BC There are other changes, as well. For example, Q seems to be much more closely tied to the data.

AW Right. It's a little confusing because every three or four years I do an entirely new implementation of K. There was a 1993 K and then there was a year 2000 K. It's the 2000 K that's underneath Q, so that implementation of K and Q are exactly the same, except that Q has a library of 50 additional operations, which are table-related, written in K.

BC If you were to write a program, would you be using the primitives that Q offers or would you write it in K?

AW Most programming I do would be in K, but if it was a lot of relational-table stuff, I would use Q because a lot of those words are already defined.

BC When you're actually in the practice of writing code, do you try many drafts?

AW I've found the best thing is just to get something running, and then I'll redo it probably 10 or 20 times until I can't get it any smaller.

BC Do you redo it for aesthetics?

AW Yes. What I tell my community is if you can find a shorter, more elegant program that isn't much slower than my code, I want to hear about it. And if it's shorter and *faster*, I absolutely want to hear about it.

BC Although I don't know that I've got the same discipline, I share your sense of aesthetics about beautiful code. I don't see that sense of aesthetics being very widespread in software. Shouldn't it be, though?

AW I think so. The thing about beautiful code is, first of all, it's beautiful. Second, it's a lot easier to maintain.
BC I think *elegant* is something that we all know when we see it, but how would you describe elegant code?
AW It's just really clear. I don't know what it is. In our community we have a listbox where people post questions and answers about coding, and the elegant code is always the shortest code.
BC Is it elegant because it's the shortest, or is being short a side effect of being elegant?
AW I guess it's both. All things being equal, less code is always better.
BC I was just thinking of the analog to a proof. The shorter proof is almost always the more elegant proof.
AW It's the same thing. It's usually easier to understand.
BC Software has often been compared with civil engineering, but I'm really sick of people describing software as being like a bridge. What do you think the analog for software is?
AW Poetry.
BC Poetry captures the aesthetics, but not the precision.
AW I don't know, maybe it does.

BC Let's talk about the data sets a little, because you're dealing with enormous amounts of data, and it's column-oriented.
AW The typical data is trades, quotes, and orders. These days, there are about a billion quotes a day just in the United States equities. The order events are probably 2 or 3 billion a day, and there are about 50 million trades. The customers tend to keep track of all that and execute trades during the day as well, but they also keep all the history so they can try different strategies.

I've done column-oriented databases since 1974. In the '50s they were doing column-oriented databases on file systems. It's the same data type, so of course you would store it by column.
BC Obviously that's the right choice when you're dealing with that kind of a data hose. If you were to build a transactional system on K, would you still want it to be column-oriented?
AW Yes, column-oriented databases seem fine. I think the reason they're fine is because we always set it up so that the hot stuff is in memory. We did that in the '70s when our memory was 32 K and we did high transaction rates. Now the guys have 128 gig, which is enough for a billion because these records are only 20 or 30 bytes.
BC So they load the whole thing into memory and then operate on it?
AW All day long all the hot stuff is in memory, and then during the day it takes about two minutes to write the whole thing down to disk and then flip to a new day and start from scratch.
BC In that case, is the data coming from a feed or from disk?
AW Multiple feeds, so the realtime systems and the historical systems are all running 24/7. It's just that there's always a quiet time.
BC But the transactions in that system are really appending temporal data to the end of a very large table.
AW Yes, but with all the analytics, they could be doing all kinds of updates to smaller tables. That's very typical. In fact, we encourage them to do that because all your realtime analytics need to be look-ups. You can't do any aggregations in realtime, so you have a lot of raw data. You have these billion rows of raw data spread among three tables, maybe. You might have 10 or 20 smaller tables that represent a certain state, such as book. There are also certain calculations that you want to maintain so that you can do either constant-time look-up or binary-search look-up.
BC You were saying that keeping data in DRAM is incredibly important for your performance. Looking down the track, what do you see in terms of the technologies that are coming? In particular, I've got to ask you about Flash and whether you think Flash memory is interesting in terms of its ability to get not DRAM speeds, but much-better-than-disk speeds. Does that pose any sort of change?
AW I think the customers are starting to investigate. It sounds great. It should provide more opportunities for other kinds of mid-range stuff.

Obviously, right now there's no random access to disk, except for the research people. The average customer's database is 30 terabytes, a trillion rows. So when they want to say, "Give me all the IBM activity for a certain day," we teach them, by all means, since it's column-oriented take as few columns as you need for whatever it is you need to do. You might need four columns: time, price, size, and something else. You've got to do four seeks, because we've got all these indexes set up so that's all in memory. If you want all the IBM activity for a certain day, that's going to be four seeks and then—boom!—you'll read a few megabytes out of each of those columns. Of course, if you go back to IBM on that day, it will probably be sitting in your file cache.
BC That's assuming, too, that when I'm accessing a file sequentially, it corresponds to sequential accesses on disk, which is not necessarily the case for copy-on-write

file systems. For file systems such as ZFS and WAFL (write anywhere file layout), if that data were not written in a temporally sequential manner, it would not necessarily be sequential on disk. Do you find that you run into those kinds of problems, or does the data tend to be written temporally sequentially as well?

AW It's always written temporally sequentially.

BC So that doesn't become an issue?

AW I don't think so. I probably would have heard about it.

BC Yes, that's probably a safe bet because the performance would be terrible.

AW But it's funny—I think all databases are like this. We're basically keeping every transaction, so that's all sequential. What happens at the end of the day, because of the way people query it, is that we actually sort the entire day by instrument and then write it out sequentially to disk. That operation happens in memory, and then it goes to disk, so it's actually sorted by security and then time. During the day, however, it's sorted by time.

BC That's a large sort. How long does it take?

AW You could be sorting a billion rows. That takes a couple of minutes.

BC The single CPU pipes are approaching their limits. In terms of that sort taking a couple of minutes, that's 100 percent compute time. Do you use single or multiple cores when you do it?

AW Single core. The data volumes are getting much bigger, and, of course, the core speed is not improving, so our customers have to split the symbol groups.

BC Then you've got to segment your data flow somehow to reflect the fact that single-core performance is not improving.

AW Yes, and we're right at that limit now, because with a single core we can do about a million updates a second.

BC What about making K or Q implicitly parallel, where you're parallelizing under the hood? Is that a possibility?

AW Maybe. I've done parallel programming since '75, and K is a parallel language. How ironic—this must be the most parallel language there is. The most prominent operator is each, which is parallel. There are no control structures. The primitives themselves are parallel.

BC Is that something you're thinking about doing? Will that parallel each actually consume multiple cores?

AW Yes, but that doesn't solve the sorting problem, and it really doesn't solve the realtime problem, because in realtime if I get an IBM quote, it's one record. I might want to check it against everything else. Certainly, if I've got one-eighth of the symbols operating entirely on their own, then that's very easy to parallelize; but if your strategy involves all of the symbols all the time, that would be very difficult to run in parallel.

BC What's the solution?

AW I think we just won't be able to do those kinds of algorithms.

■

BC You have this four-year itch to write a new programming language, so you're coming due. Are the constraints on the problem any different? What's the new language going to look like?

AW It will probably be 95 percent the same. It's the same semantics: noun, verb, adverb—same data types, same functions. But I like to try different things under the covers. For example, I like to try different memory allocation schemes. It's all call by value but reference count, which is kind of amazing when you think about it, so there's no garbage collect. Everything is reference counted; when it's free, you know immediately so you get good reuse. Under the covers, I play with different things. For example, if you're doing a vector operation and the reference count is one, well, then reuse the vector. I also always try to make the code smaller.

BC Are you actually redoing the implementation, or are there going to be semantic differences as well?

AW The implementation is 100 percent new. I write everything from scratch, so the C code is entirely different but the semantics are about 95 percent the same.

BC You start over in terms of your C code? You take all that and throw it out?

AW Yes, completely.

BC What does it feel like to part with all that code that's so lovingly created?

AW I love starting from scratch—and it's stupid because doing the parser, tokenizer, and printer takes me months.

BC Do you find that you can come up with a better solution?

AW I think they're getting a little bit better, but I think I'm converging.

BC Is that advice you would give to practitioners: to throw out more?

AW Yes, but in business it's hard to do that.

BC Especially when it's working!

AW But I love throwing it all out. **Q**

LOVE IT, HATE IT? LET US KNOW
feedback@queue.acm.org