JEREMY KEPNER: All right, welcome everybody. I'm really pleased to see you all here today for what will be the first of a nine lecture course called signal processing on databases. And this is a really special day for me because this sort of represents a big step in a set of technology that I've been working on for a couple of decades, and really been working intensely on for the last three or four years. And this is the first course on what we hope will be a technology that will become very widely used.

It is a completely unique course on a completely unique technology, so I want to thank you for being the guinea pigs of this course. It's hard enough just doing a new course on an existing topic. Doing a new course on a novel topic is quite a challenge, so we're going to be trying a few different things here. There are going to be certain topics covered from more than one angle, and we'll see which ones sort of make the most sense for you.

The first thing I'd like to do is talk a little bit about the title of the course, which is a little bit different. The title is signal processing on databases, which is not two phrases, signal processing and databases, that are usually connected together very often. And so, signal processing, as we all know, is, here at Lincoln Labs, one of the most core things that we do. In terms of the technology, it's almost difficult to imagine a technology that's more core to what we do. The only thing I could think of was systems engineering, maybe. But signal processing is really at the heart of what we do.

Mathematically, signal processing really, the heart of that is detection theory. How do you find things? Given sets of data, how do you eliminate things you're not looking for and find the things that you are looking for? And then, mathematically, the heart of detection theory is linear algebra, and so, just a show of hands how many people have had a linear algebra course in there? See, that's really awesome. See at Lincoln, that's probably-- this one a few places in the world where it was pretty unanimous. You know, there's not too many places where that's the case.

And so this technology is really designed for you, because it really assumes a foundation in linear algebra, and that's not as common as you might think. And so people who haven't had linear algebra, it's much harder for them to take advantage of the ideas that we're going to talk about. So signal processing, detection theory, linear algebra, that's sort of the root of the first phrase. The second phrase, databases, when we think of databases we think of typically, maybe things that you do, searching on the web, texts, maybe other things like that, which is not necessarily consistent with the mathematics that we think about when we think about signal processing, detection theory, and linear algebra, we naturally go onto real numbers, matrices, those types of things.

So what we're really doing here is we're connecting that mathematics with sort of a whole new branch of data. Words and strings and relationships. And so that's kind of the really novel piece of it, and it's really the core idea that we're bringing here is we're going to bring all that machinery that we know so well, and we're going to try and apply it to this new area which is becoming increasingly important for us. So that's the explanation of the title. That's how we picked the title. There's other titles, and so I just wanted to spend a little time on that.

An enormous number of people have contributed to this work. This is a list of the people that I could remember at the time that I typed the slide, but no doubt I have forgotten somebody important, and so I apologize for that. But certainly these people, and the whole team have been very instrumental in allowing this technology to move forward here.

So this is going to be the outline for this particular lecture. And there's going to be a rhythm to most of these lectures, which is there's going to be a lecture, which is going to be PowerPoint-- I apologize, we all see plenty of PowerPoint-- mainly to introduce ideas. And then there will be sort of a demo, which will go through code that is actually available in your LLGrid accounts now, and some of that code will even be part of homework assignments, if you so choose to do the homework assignments. And so that's going to be the flavor. We'll do the slides, we'll take a short break, and then we'll do the examples and talk about them.

So let me talk about some of the kinds of-- and I stole this slide-- let me talk about some of the kinds of data that the technology that we're going to be talking about today really, really enables. It allows us to deal with data that has to do with relationships, relationships particularly as those represented by graphs. So for example, you might have vehicle tracks. Those represent relationships between locations. A car goes from one place to another. That's a relationship between those locations.

Another is social networks. You know, everyone wants to know who their friends are, how many friends they are, who they're friends of friends are, those types of things. So that's another type of set of relationship. Again, very textural and oriented, very much oriented towards words and not numbers. Another area is cyber networks, a very apropros topic that people are, at least everyone is impacted by, and many people are doing work on, which is relationships between computers, communication between web servers, those types of things. And again, that is a very important area for us.

Just to give you sort of a big picture here, like well, why am I even here? Why do I care about this? What are the things that we can do? I'm going to kind of walk through an example of some of the things that we have used this technology for that have been very exciting for us. And so, trying-- and we'll have examples like that throughout the course, you know, how you can actually use this, how this has had really a real impact. And so the first one we want to talk about is some work we've done in the cyber domain.

So here's an example of a data set, which is a graph that is web traffic. And so, if we see here, this is the source IPs. So, you can imagine these are the computers that are on the inside of some domain, and then these are the destination IPs, all web servers that they hit are the green ones around here. This is sometimes referred to in graph theory as a bipartite graph. It's basically two separate sets, a vertices that talk to each other but have no, at least in this data set, we are not recording any connections within the sets.

And then, hopefully you can see here these faint blue lines here that are the actual connections that show you which of these computers are connecting to which of these computers. There's obviously a lot of data, web traffic data. There's enormous amount of it. It shows 90 minutes of data, and in this particular case there was actually malicious activity that we were looking to detect in this data set.

So one of the biggest challenges that you have in any data set like this is data representation. And sometimes in database parlance this is called the schema. So how you represent your data is usually if you can figure that out, you're almost halfway to solving the problem. And so, given that, in this case, we wanted to use graph techniques, we discover that our data almost always doesn't come to us in a nicely formed graph, and so we have to go through a lot of steps here. And so one thing you'll see is with this technology is that there's almost always possible part of a pipeline from raw data to conditioning the raw data to inserting it into a

database to doing analytics to doing queries and other types of things. You're always really going to be using a pipeline.

And you're going to be wanting to use different technology tools for different stages in the pipeline. We're not in any way trying to suggest that the technologies we're talking about today are the only pieces. It's an important piece, the technologies we're talking about, but they're not the only pieces. And you're always really functioning in the context of a pipeline.

So this just shows the technology stack that we will be leveraging, that we typically leverage in these types of things. Ultimately, our goal is to get up here to these graph analytics. One of the things we'll be talking about most in this course is the high level languages that allow you to essentially do graph analytics very easily and bridge between the databases themselves, and the high performance computing that you will need. So again, not only do you have a pipeline of software and data, you have a pipeline or stack of technology. So when you're addressing problems like this, usually you're starting at one place but you will almost always, when you build a system, work out and deal with these other types of things.

So that's sort of just a quick overview and example and I don't expect any of you-- it's probably just kind of, the purpose of the example's probably to raise more questions. Well, how do you really do this? So now I'm going to sort of take a bit of a right turn here and really talk about what the course is about, OK. And I'm going to actually talk about the course a little bit of a big picture. And I always felt that every course should tell you why that course is the most important course you'll ever take in your career. You know, I think every single course you can come up with an argument for that, but people don't say it. So I'm going to tell you why this course is the most important course you will ever take. Of course, every course is the most important course you'll ever take, but I'm going to tell you why this one is.

So stepping way back, we're all part of MIT here and MIT has a formula for success. MIT is the number one science and engineering university on earth. We can actually, by any reasonable measure that is true. And in fact, by the people who track this, it's true by a lot. And so you know, it's a real honor to be associated with this organization, to be at an organization that's at its zenith. And MIT has had this incredible run of success over the last 60 years because fundamentally they recognize the formula for making discoveries is a combination of theory and experiment. Bring those two together, and as an organization, we've actually organized ourselves, MIT is organized around this formula in that we have theory and experiment, and theory is the academic part of the mission. That's run out of

departments.

Typically, it involves mathematics. This is a place-- I mean, everyone here raised their hands to linear algebra, and I can go to any part of MIT and that will be true. We probably have the highest mathematical literacy of any organization on earth, and that's a very powerful, powerful capability. The mathematics that we use, they become manifest or real, typically in the forms of algorithms, and our algorithms become real in the form of software.

Likewise, on the experimental side, you know, that's the research that we conduct. MIT actually has large laboratories. Laboratories that are fully peered with the departments. That is actually quite a unique thing for an academic institution to have laboratories and laboratory heads that are right up there with the department heads in how they're positioned. Lincoln being the largest of them, CSAIL, the Media Lab, there's a number of them. So we have this thing.

And the focus of research, of course, is measurement, which is data, which often gets reduced to bytes. And so, often implementing this MIT formula comes down to marrying software and bytes together in a computer. That's sort of where we bring these two ideas together. And so we're going to be talking a lot about that. Implementing software bringing bytes together analyzing together that's where these two things come together, and I think that's really, really important.

Now what's the tool we use? What's the machine that we use to bring these together? It's a computer, all right. And nowadays, computers have gotten a little bit more complicated than the way we used to think about them. We used to think about them typically as a von Neumann machine, which means you had data and operations, you know, bytes and software. And the bytes go into a part of the computer, they get operated on, they come out. And that model is still true at a high level, but the computers we tend to deal with now are much, much, much more complicated.

So on the left is a standard parallel computer. Almost every computer used today looks like this. You have processors, you have memory, you have some kind of persistent storage, you have a network connecting them together, and this form a very complicated what we call memory hierarchy. It begins with registers here at the top, caches, local memory, remote memory, and then storage. And these are the units that go between them.

And there's a lot of implications of this hierarchy, in terms of what things are-- you know,

bandwidth goes up as you go this way, latency goes down. Programmability generally goes up the closer you are. You don't have to worry about things. And likewise, the capacity of data goes down as you go traverse this way in a hierarchy. So all modern computers that you use are these von Neumann architectures with multi-level memory hierarchy. And fundamentally the thing you have to know is that this architecture selects the algorithms you use. If algorithms are not going to run well in this architecture, you probably don't implement them. So one has to be very much aware of one's algorithms with respect to this architecture, because if you pick algorithms that aren't going to run well in this architecture, it's going to be very, very difficult.

The goal of this course is to teach you techniques that will allow you to get your work done faster. And so this is a rather complicated drawing here, but this shows, given a problem that you want to solve, if you implement it in software using a variety of techniques, what the performance of those techniques is. So in a certain sense, you could view this as the volume of code that you have to write to solve a problem. And then we have implementing the program and C as a reference point. So as you move over here and you do things like in assembly or Java, MapReduce, you end up writing more code. If you move over here and do things in C++ or Java or MATLAB, you write less code, and this shows you the relative performance.

Again, as you do things in lower level environments like assembly, you can get more performance. And as you do things in higher level environments like MATLAB, you get less performance. And this is a fairly general trade off space here. If we add parallel programming models here, of which there is many, we have direct memory access, we have message passing, we have a manager worker style or map reduce style, we have a whole different styles of programming here.

We're going to want to try and be here. We're going to want to do things that take less effort and deliver you more performance. And so, we are going to be mostly focusing on MATLAB in this class and a technology we call [INAUDIBLE] to describe that more later. But this is a software combination that allows you to get relatively good performance at a significantly reduced effort, and that's really the only thing we're doing in this course. We're just saying, if you're doing these types of problems, there's a right tool for the job or there are right tools for the job, and if you use the right tool for the job, it will take you less time to get the job done. And we're going to try and teach you about these tools and how they actually work.

Now, a little bit of bait and switch here. We call the course signal processing databases, but we're going to get to databases at the very end, unfortunately, to disappoint you all. And the reason is because databases are good for one type of problem, but it's often at the very end of the game where you really need to work with databases. So to understand, we have a variety of different ways we can solve problems with algorithms, OK. We can solve our problems just using a single computer's memory. We can solve our problems using the memory on multiple computers. We can solve our problems using disks or storage, or even with many disks, OK, and as we go this way, we can do more and more and more data.

But one of the things that really matters, and it kind of goes back to that other chart there is well, when I'm working with this data, what is the chunk size? What are the blocks that I need to get the data in? And if I need to get the data in very, very small chunks, well then obviously having the data in memory is going to be the most efficient thing I can do. If I want to just get one record out of many records, then that's the most efficient, having the data in memory.

Likewise-- so that's where we typically start, right? If our problem is small when we almost always try and start with our problems with small, we're going to be working here. We're going to be writing what we call serial programs on one computer. And then, the great thing about serial memory is that even as our requests size gets larger, they're really not going to do anything better than just having the data in memory. So we'll be able to grow that model up to data sets that are fairly small, maybe gigabytes, and we can continue to use that model just fine.

And then, if we decide, well, we're really moving beyond that, we need to do more data, we can go to a parallel program to get more memory, or we can use data in files. We can write our data to files, we can read them in, process them, and read them out, and that model works very, very well. And in fact, particularly if we're going to be accessing the data in large chunks. If I have a lot of data and like I want to do some processing on the whole piece of it.

Likewise, if we go to a very large data sets, we find that we can even do parallel programs writing files in parallel if we want to do extreme large problems that we want to traverse all the data. But finally, we're going to come back to this case where we need a database. And a database is when we have very large amounts of data but we want to access-- database, data that won't fit in our memory, but we want to access in little bits. We want to look up something. And that's really the only time we really need a database, unless of course someone says this is your data. It is available to you on a database. That's the only way you're going to access it.

But if you have control-- so you always want to follow this path. You want to start with your serial program memory, maybe go to a parallel program using files, maybe parallel files, and then the database is almost the two of last resort because it adds a great deal of complexity, and while it can look up things much quicker, if you actually end up doing something where you're going to be traversing a significant fraction of the data, you will find it's actually significantly slower than if you have the data on files.

So we're going to show you techniques, we're going to follow this path, and then hopefully by the end, you'll be like, oh, well yes, now I know why we use a database for these instances, but I realize that I can get most of my work done using these other techniques that work just fine. So hopefully you're going to be learning what we call the fast path. So if you want to get a certain level of performance, OK, if you have no training or minimal training, this is the path that you follow. You began using a technology. Your program gets slower for a while. You beat your head against the wall and the web for a while. Maybe eventually, after days, you get back to where you were, and then slowly you climb up here to get it to some level of performance weeks later.

This is something we observe time and time again, and typically what happens is that right around here people just give up with the technology and they say the technology didn't work for them. The point of this class is to give you the techniques that an expert would have. An expert could use the technology, know exactly how to use it, and then can go on this path where basically within hours you're getting good performance and if they really want to take it to the next level they can spend days doing that. And so that's kind of our whole philosophy here.

So we're going to actually have some new concepts in this course. I think they're talked about in the course outline a little bit, some of the really new ideas that you probably wouldn't have run across in other courses. So we're going to be talking about graphs a lot in this course. The standard graph that-- how many people have taken a graph theory course or a computer science course that talked about graphs? Most. I'm glad to see linear algebra is still the stronger one here.

That's more important. Between the two, having the linear algebra background is the more important. And I can, if you know linear algebra, I can teach you graph theory pretty easily. The traditional definitions of graphs, though, that are taught in most computer science

courses, they tend to focus on what we call undirected, unweighted graphs. So these are a graph is a set of vertices and edges, and so what we're saying when we say it's an undirected and unweighted graph is that all the edges are the same between any two vertices. There's no real difference between them. Either an edge exists or it doesn't. It's kind of like a zero or a one. And then it doesn't really-- when we say two vertices connected, there's no directionality of that.

The majority of graph theory and what's taught in class is based on that. Excuse me. Unfortunately, when we get into the real world-- unfortunately, when we get into the real world, we find that I've never run into that kind of graph in the real world. In the years I've been doing this, I've never actually run into that kind of graph. In fact, another part of that is they'll talk about the distribution. In fact, you'll often hear something called an Erdos-Renyi graph, which is just a random connection. It is any two vertices are randomly connected with maybe some probability. And again, I've never run into that either in the real world.

So the focus of graph theory that we have now that we have all this data and can compare the theory and the data, can bury the theory of the experiment, we see that the theory is wanting. The theory is not a good stepping stone into the data. And so what we see is that the real data is not random. It's often what we call a power law distribution, that a certain vertices are very popular and most vertices don't have a lot of connections to them. Usually the edges are directed, that is, there's meaning. If I friend you, it's not the same as you friending me on Facebook or Twitter, or you twit face, or whatever the technology is of the future.

Usually the edge of self has meaning, has weight. That weight could be of value. It could be words, it could be other types of things. Typically you have multiple edges. I could send you many friend requests. You know, that's not the same thing as me sending you one friend request. Maybe I can't send you friend requests. I have multiple friend requests. I don't know. But you can have multiple edges between vertices. And probably most importantly is edges are often what we call hyper, and this is a very important concept that is almost never touched on in graph theory, which is that an edge can connect multiple vertices at the same time.

A classic example would be an email that you send to multiple people. In fact, most emails these days have more than one recipient. And so, me sending an email to everyone in the class, one email to everyone in the class, is a hyper edge. That edge, the email, connects and has direction, you know, me with all the recipients. That's very different than me individually sending an email to every single one of you, which would be a standard edge. So this is an

important concept not typically discussed a lot in standard graph theory that is very important.

So we're going to move beyond the sort of standard definition of graphs in this class, and that's going to be hard. We're going to deal with a bigger definition of linear algebra. Traditionally, linear algebra is dealing with matrices. They have indices for the rows and the columns and the values are real numbers. Maybe integers, may be complex numbers, we deal with complex numbers here a lot, but we're going to be dealing with things like strings. We're going to be taking linear algebra and applying it to things like words. And so that's going to be different. In fact actually, one of my favorite-- it might be the part of the course that puts you all asleep, but it's actually one of my favorite parts of the course.

And then, bigger definition of processing. One of the dominant things you'll hear about in this space-- there's a lot of technologies out there. A popular technology is called Hadoop, and it's map/reduce parallel programming paradigm, which is a great technology to kind of get people started, but for people with the mathematical sophistication that we have, we can use programming models that are simply better. If you understand mathematics to the level that we do, we can give you tools for programming parallel computers that are just more efficient in terms of you write less code and you get better performance than using techniques that really are designed for people that don't have the mathematical, the linear algebraic background that you have. So this is really going to be the foundation of the course.

So let me continue here with the course outline. So this is going to be, essentially, the nine lecture of the course. I think this is still pretty accurate. So we're in the introductory course, we're going to review the course and goals. The next course is really going to be dealing with this concept of associative arrays, which is the core technology that sort of brings all this together. When I talk about extending linear algebra to words, using funny, using, sorry, by using fuzzy algebra that really gets into mathematical abstract algebraic concepts called group theory. It's not nearly as scary as it may sound. We'll get into that. It's actually, I think, quite elegant and really good to know.

Good to know that when you work with the technology and you work with associative arrays, that we've actually thought about the mathematics of how these things work together. And when we add a feature, it's usually because it fits mathematically, and when we don't add a feature, it's usually because someone has made a request and like, yeah, that's really going to take people into a bad place.

Then we're going to get into really sort of the center part of the class where we're going to show you how we do analysis of entities on unstructured data, and then doing analysis on unstructured data. We're going to--I've talked about power laws. We' going to whole class on power law data, modeling of that deal, but then we have another class in cross-correlation, and then we're really going to get into parallel processing and databases. And the last two classes aren't ever going to really be lecturers. They're going to be really just the demos, and really-- because that's where we're going to really bring a lot of things together, and we're going to walk you through code examples, but we're going to need to take some time to really walk you through them, because you're going to see, wow, there's all these ideas that we've talked about coming together in these code examples.

So there is, we have lectures. If you see in the software that's available in your LLGrid accounts, all the lectures are there already, and I think we have lectures one through seven. So seven is the eighth lecture. Or, I'm sorry, zero through seven, so seven is the eighth lecture, and that will be a kind of a short lecture. And then the ninth lecture will be just going through examples. So we believe in this whole philosophy of taking a linear algebraic point of view with graphs so much that we even wrote a book about it. So you have all been given copies of this book. This really gives you the philosophy and the power of if you think about graphs using the techniques, so I'd highly encourage you to flip through it. There are certain parts of this, certain sections here that are really almost taken directly out of this book, but a lot of it is supplementary material. And what you'll discover is that this course is really the bridge to these techniques.

This course allows you to go from kind of this mess of unstructured words and other types of things, and then put them into this nice linear algebraic format, so that is now you can do these techniques that are described in the book. So this is almost a bridge that and I'd certainly encourage you to look at that and to ask me questions about it at any time.

So now I'm going to come back to our original example, which was the cyber thing and kind of walk us through that in a little bit more detail and maybe highlight a few of the things that I've talked about already. So here's a very standard data processing pipeline. So we want to do is we have raw data, in this case our raw cyber data, which is a series of records. And we need to convert that into a set of vertices and edge lists from which we can then do graphs and graphs analysis.

And, as you'll see, typically what you do is you write parsers to convert this data from some

raw format to some format that is sort of the next step in your processing chain. And then we convert these edge lists into adjacency matrices, which is how we view our graphs.

One word on adjacency matrix, let me go back here. This is a graph, [LAUGHS] for those of you who don't know. The set of vertices and edges, OK? And this is an adjacency matrix of that graph. Basically every single row is a vertex, every single column is a vertex, and if an edge exists we put a dot here.

There's a formal duality between graph theory, the fundamental operation of graph theory is what's called breadth-first search, which is shown here, giving a starting vertex. You know, traverse its edges to its neighbors, and we have the same thing going on here. Given a starting vertex in a vector, we do a matrix multiply to identify the neighbors. And so at the deepest, most fundamental level, graph theory and linear algebra are linked. Because the fundamental operation of graph theory is breadth-first search and the fundamental operation of linear algebra is vector matrix multiply.

So we wish to form these adjacency matrices so we can take advantage of that. Here's a little bit more detail on how we're going to use this technology D4M to do that. So we have our raw data, we convert the raw data into a CSV file-- stands for a comma separated value files. It's kind of the default format of spreadsheets and tables. It's become very popular over the years. I remember when it first came out 20 or 30 years ago, it wasn't that popular. You know, it's like people did-- but now it's become extraordinarily popular file format. It really is ideally suited for this kind of data.

So we convert that into CSV files. We then take those CSV files, read them in using our D4M technology to insert them into a distributed database. We can then query that distributed database to get these associative arrays. And then we can, from the associative arrays, now we have the full power to do our graph algorithms. So D4M really helps you in going, you know, these few steps and setting the table so that you can then you graph algorithms. And you can even then do the graph algorithms without even using D4M at all.

You can just use regular linear algebraic operations in MATLAB just fine. But this is that bridge, that connector. In fact, I highly recommend people do that. I highly recommend you use D4M for only the parts of your problem that it's good for, and use other technologies, you know, like MATLAB, ordinary matrices, and other types of things. That's good for you, you'll get better performance. It's good for me because, if you have problems, you'll bother

somebody else, and not us. So that's good, it's good for everybody.

So an example of kind of what this looks like. Here's a proxy log that we got from the data. It shows a web, essentially a web log, the first thing we do is essentially convert this into a CSV file. So basically, each entry here gets a column, we have an ID for the actual entry. In this case it's a source IP, a sever IP, a time stamp. And then, the actual line of text that's associated with it. And this can go on and on and on. We can have many different types here. And if you are a standard database person, put your SQL hat, which is sort of the standard database, on. You would just take that data, and you would make a table that would have these columns, and you would insert them and away you would go up. OK?

But what we discover is that-- the challenge becomes when you want to look up data quickly. Or you want to insert data. Let's say you have an enormous volume of data that you want to insert and you want to go to look up any, say, IP address, or any time stamp, or something like that. What you discover is that SQL might give you good performance, but there are databases out there and technology that will give you better performance. And these databases are called triple stores, because they store all the data as-- anyone?

Triples, yes. They store all the data as triples, which means we will have a row, and a column, and a value associated with everything. So what we do is we take our dense data, and we do what we call, create an exploded table. So essentially we have the log ID, let me append that, and we take the column name and we append its value to it, and server IP. And this is what you would call an exploded table.

Now, if you're an old SQL person like me, this schema will immediately give you hives, and other types of allergic reactions, and you will want to go to the hospital, because you're creating an enormous number of columns. And in SQL you where you can dynamically create columns, but at your peril. Basically, you can create columns dynamically, but the database reserves the right to recopy the entire table, and reformat the entire table, if it so requires.

Well triple stores don't have this requirement. Triple stores only store the non-blank entries. And so they're perfectly happy with as many columns you want. If you have a billion rows, 10 billion columns, completely fine with that. So that's a powerful thing.

So I think we talked about that. So we use the indices as rows, and we create unique type-value pairs for every single column.

Now, by itself, this schema does you absolutely no good. You basically just made your life harder, because, by itself, most database tables, database systems, including triple stores, are going to have an orientation. They're going to be row oriented, or column oriented. And in this case, the databases we will be talking about are almost always row oriented. Which means they can get a row in constant time, give it a row key like this and you can look it up very quickly. And it will give it back to you very quickly. That's what it's design to do.

And you can insert enormous volumes of data, and they will preserve that contract. You can be inserting millions of entries per second, and you can still look up stuff in milliseconds. So very powerful technology, but all we've done so far has made it so that the format of my data is in this funny format, coming out the other side. I've just made myself harder. Well, there is a payoff, which is we also store the transpose of the table. So, to anybody who is used to sparse linear algebra, this is an old ancient technique.

We almost always store A and A transpose, because in sparse linear algebra we'll either have a row oriented or column based store. And then we just want to keep one around because certain operations are going to be faster with the transpose, and certain operations are going to be faster with the matrix. We do the exact same thing here, and combined with our exploded schema, now every single one of those columns is a row. And now we have indexed every single string in the entire database. Once, with one simple schema of two tables.

And not to say that all your databases will only have two tables, but they might be three tables or four tables-- is typically what we see, which is a huge reduction compared to the standard database schemas that one uses where it's easy to have dozens and dozens of tables, even 100 of tables in a well thought out schema. Here, essentially it's one schema to rule them all. One table and its pair, and you've indexed every single thing. And the cost is, you've doubled your storage. And every single database person that you talk to-- if I said in exchange for doubling your storage I would give you a fast index access to every single string in your database, they'll usually take that deal 100 times out of 100.

So that's the power here of this technology, is you can reference enormous data sets. And we'll come back to this schema over and over again. But that's sort of the madness. We'll be-- as we move forward with associate arrays we always view things in this way. And I want you to know, at the end, there's a really good reason why we do that.

So now we've ingested our data into the database and we can do queries. So if I have a

binding to a table-- and one of the things in the deformed technology that we do that's very nice for you, is if you're using a table and its transpose, we hide that for you completely. If you do an insert into the table, it will automatically insert its transpose. And if you do a look-up on rows, it will know to use one table, and it does a look-up on columns, it will know to use the other table.

So here we're doing essentially a range query. This just says give me all the data from this range. Time stamp May whatever to, it looks like about a 3 hour time window. So that's a query into the database, and it returns an associative array of keys. And here is what that looks like. It looks like a bunch of triples.

I'm going to then take the rows of those keys. So basically I found every single row, within a particular time stamp, and I want to get the whole row. So I can get the row of that and pass that back into the table to get the actual data. Now you see I get the whole row. So I get server IP, and all that stuff, the complete list.

Now I'm going to do a little algebra here. I want to create the graph. It's all source IPs and server IPs. I get to do this with this basically correlation here, which is essentially a matrix-matrix multiply. That's it, I've just constructed the entire source IP and server IP graph from all the data in this time window. That is the power of D4M. It allows you to do correlations on this kind of data, same way we do correlations using traditional techniques in linear algebra. And you can be like, "Wow."

This is motivational at this point. We'll get into exactly how you do this, and exactly how this works in the example. But this is really what people use D4M for, is to get to this point where can do these types of manipulations. And the key to deduction theory is doing correlations. Correlations allow us to determine what the backgrounds of our data are, what the clutter of our data are, and then we are on our way with traditional detection theory, and signal processing techniques, and all the things that we know we love. So this is kind of that connection made manifest.

And so here you can see, we now have an associative array or an adjacency matrix that eventually, essentially sorts IPs and their server IPs. You can actually plot that and that's how we get the adjacency matrix of this graph, G. This is what it is. That's how we constructed-- this is the code we actually used to construct that graph.

So moving on here. So this just shows the kind of the whole thing here. We had a whole

week's worth of proxy data. This just shows you the timing, that it took about two hours to ingest it, and about three hours to do this processing. 100 million proxy logs, 44.5 billion triples. This is big data. You know, really this is big data. This is the kind of thing that you can do. Whatever anybody else is doing, you should be able to do on larger scales than other people using this technology.

So that brings us to the end of the lecture. So just to summarize, big data of this type is found in a wide range of areas, document analysis, computer networks, DNA sequencing. There's a kind of a gap between the tools that people traditionally use to do this problem, and D4M, this technology, fills this gap. So, with that, are there any questions before we get onto the examples slide? Yes?

AUDIENCE: [INAUDIBLE]?

JEREMY KEPNER: Yes.

AUDIENCE: [INAUDIBLE]?

JEREMY KEPNER: So the data--

AUDIENCE: Can you repeat the question?

JEREMY KEPNER: Oh yes, the question was does it generalize to multi-dimensions, I guess? And so, there may be kind of two questions there.

If you had-- if I just correlated all the data, took that table out, and just fully correlated it, I would end up with a giant matrix of blocks, each one representing different correlations. So the source IP, server IP correlation, the source IP, source IP correlation, which would be empty because it's a bipartite graph. And then all the other correlations would form different blocks, and that type of thing. So in that sense we support multi-dimensions.

Whether we support-- but supporting multi-dimensions in terms of tensors, OK? Well, generally in sparse linear algebra, there's very minimal support for higher dimensions. I'll put a little shout out to my friend Tammy Kolda, who has a technology called the Sparse Tensor Toolbox. She's a scientist at Sandia and so I encourage you to do that. But all of our stuff is in two dimensions from a linear algebra perspective. Also linear algebraically, the math there is a

little bit more common than when you get into the tensor. And then you have these kind of weird products that you're starting, which you can define, but are just not commonly taught.

So and you might be like, well you're really losing something there. I'd really like to have multiple dimensions. Well, when we've looked at it, the fact of the matter is the sparsity always increases as you go to higher dimensions. And the sparsity is sparse enough. So typically the kinds of data that we will have-- if you have n vertices, you'll have maybe 10 times n edges. So the sparsity is extremely low. We added a dimension. This sort of internal data structure costs associated with adding that dimension would not reap the benefits. Because now your sparsity is even lower. And so that's generally not something that is commonly used. But, if you're interested exploring that I certainly encourage people to download that toolbox. Very well written software, and people can explore that.

Other questions? That was an outstanding question. And other questions at this time? All right well why don't we stop the recording right now, and then we will switch-- why don't we take like a five minute break. I think there's some water out there and stuff for people. And then we'll get back to the examples, which probably will be a lot more fun. So hopefully wake you up after these new graphs.