PROFESSOR: All right, guys, let's get started. So today, we're going to talk about a very different and principled approach to building secure web applications. And it's going to be about a system called Ur/Web. And right now, our guest lecturer is the author of the system, Adam Chlipala, who's a professor at MIT, is going to tell you more about the system he built.

ADAM CHLIPALA: All right, so I want to get to a demo as soon as possible. But before that, I just want to spend some slides setting up part of the context about this system. And you've probably gotten some of that context already from the draft paper that was the reading for this class. So what is Ur/Web? It's always good to start out by explaining what the name of something means.

So Ur/Web, first it's a programming language for building web applications. That's what the Web part of the name means. And it's sort of a full stack system. It does everything you need to do to build web applications. And Ur is a new general purpose functional programming language that is used to implement these web specific features.

And the whole point of Ur/Web is that instead of having a general purpose programming language and then having a library or a traditional framework for building web applications, it's all integrated into a customized programming language in Ur/Web. And it's a language that involves compilation, not interpretation at run time. And the compiler in some sense understands what a web application is supposed to do. And it will point out mistakes that you're making that a conventional compiler, for say Java, would not be able to realize where mistakes.

So there are really three main principles that I was trying to follow in designing this language. The middle one is most relevant in this context. But they are programmer productivity, security, and performance. And the last part, especially on the server side, because that seemed more important for scaling reasons. In many cases, the users of your application won't notice small performance issues on the client side. But a small issue on the server side could force you to buy many more servers than you would have otherwise.

And at this point, there are some users of Ur/Web-- not nearly as much as pretty much any other language you probably think of. But there's at least this one commercial web application, which is an RSS feed reader that supports such exotic features as displaying comments. And there's the URL chosen by a non-native English speaker who regrets it now. It's called BazQux Reader, as a combination of common medicine tactic variables from the hacker community.

And there are a few thousand paying users. And it looks like that-- much nicer than anything I know how to make with CSS. But here's a proof that it can be done using Ur/Web. Feel free to jump in with questions at any point, though I probably haven't gotten to the point yet that provokes many questions.

So the basic sales pitch for Ur/Web is that it has a very high level programming model, which is very different from, say, Django, which I know you spent some time reading about or talking about in class. And it has a good security story. Some features you want for security are really integrated into the system so that you would really have to work hard to avoid inheriting these security benefits. And I'll say more about the detail shortly. And also, the server side performance is unusually good, even among the popular tools for building web applications that you're more likely to have heard of before.

And the caveat is that we probably need to have internalized the big ideas of functional programming languages like Haskell before a programmer's ready to start using Ur/Web. And looking at the questions and answers for this class, maybe a fifth of you were complaining about the functional programming parts of the paper being hard to follow.

I apologize. There are just so many good ideas in the world of functional programming that it's hard not to start from that point and add more cool stuff on top of that. And I will try to avoid any requirement to know that material to follow what I'll be doing in class today.

So the programming model is really closely connected to static typing. And that's not just static typing like in, say, Java, which has a relatively inexpressive clunky type system, but static typing like in Haskell or OCaml. And these types are one of the ways that the compiler understands what you're doing and catches mistakes in your program.

And it turns out that the core Ur language that Ur/Web is built on top of has a very expressive static type system. So many of the things that Ur/Web does are actually just exposed as libraries with no special compiler support. For instance, we'll teach the compiler how to type check SQL queries without actually building the typing rules of SQL into the compiler. They can be encoded as a library and use a standard type checker to make sure your SQL queries are following the rules of SQL.

Most relevant in this context, the security story at a high level-- most of the most common security vulnerabilities are impossible by construction in Ur/Web. You will have to explicitly enable scary looking flag names to be allowed to do most of the most awful things you can do in a web application, like no cross site scripting vulnerabilities unless you really invoke some black magic, say, by using the foreign function interface. And there are a few other security-specific features that I'll highlight later.

And the performance is also very good. The compiler is, first of all, a domain specific compiler for a web application. So it understands what the web application is doing and is able to optimize some things that a more general compiler wouldn't catch. And usually the code that comes out of this compiler that runs on the server is native code, which is very, very competitive with what you might bother to write by hand in C.

And the performance costs that there are compared to other approaches tend to have to do with the concurrency model, which makes the programmer's life easier at some cost in performance. And I'll say a little bit more about that in a moment.

Here's a quick plug for this web framework benchmarking initiative that is run by a third party. This is a screenshot of the results of the most recent round where a number of different web programming tasks were completed in many different frameworks, and they were compared pretty much exclusively on performance so far. And here you can see Ur/Web sitting at fourth out of about 60 frameworks on this benchmark. And there's been some improvements to the Ur/Web compiler since this screenshot was taken. And I expect in the next round it'll move up a little bit higher.

But basically, already this is a simple example using SQL to generate HTML pages. You get about 100,000 requests per second from the Ur/Web server, which is going to be just plenty for most applications. So sort of maybe the important takeaway message from this slide in this class is that you can adopt a high level model that makes security easier to achieve without just giving up all the performance that you would expect to get from more mainstream techniques.

All right, so let me start out by giving my cartoon impression of the way web programmers think about writing web applications in mainstream frameworks today. And then I'll show the different perspective that Ur/Web provides, where some of the things that can go wrong at this level given the abstractions that are exposed can no longer go wrong by construction.

So the basic cartoon picture is there's a web server out there. And it's sort of in charge of the whole process of your application. And there's a whole fleet of browsers out there that are going to interact with that server. It'll have some state that winds up effectively shared across all these browsers through their contact with the server.

So the usual picture is that the browser starts interacting with the web server by sending it an HTTP request that includes some URLs embedded in it. And then the web server throws back, again, the HTTP and HTML page. And there are some URLs embedded in that, which can be used to decide which request to make the web server in the future.

This web server might also be talking to a database that provides a persistent store that is shared across all the users of the application. One popular protocol to speak between the web server and the database is SQL. That's what I'll be focusing on talking about Ur/Web.

And also, with modern web applications, it's not just the one page at a time model where whenever anything has to change on the page you make a new request to the server and then replace the whole page of the unit. There's this Ajax style where the browser within a single page view will sometimes make extra HTTP requests to the web server and receive responses that are processed programmatically in a customized way. And this often uses representations like XML and JSON and other simple wire formats for exchanging data between the client and the server.

And then when the browser gets back that response, there's some JavaScript code running there, which implements arbitrary logic for controlling the UI that we're displaying to the user. And the way this works is that this JavaScript code can read the responses that the server has given to those different Ajax calls.

And then it can modify the page that's displayed basically by mutating a global variable that stands for the page. And any part of the program can have arbitrary effects on this global variable that is the page. And often, parts of the page are looked up by string IDs that are annotated on nodes of the tree that's describing the document.

And finally, one more complication-- sometimes we want to allow what feels like the web server contacting the browser without prompting. So say there's a new email message. The web server wants to tell the browser, new message. So there are a variety of ways of doing this involving acronyms like Comet and WebSockets that really look a lot like the browser contacting the server. It's the same sort of thing conceptually in the other direction.

All right, so I want to bring back on the screen all these protocols and languages, highlight some parts in yellow here. Having read the paper, does anyone have a guess about what is the commonality between all these highlighted parts here from a security perspective? Yes.

STUDENT: They're all strings. So you can put whatever you want in them.

ADAM CHLIPALA: Right, in the mainstream approaches to web application programming, all of these things are strings. And the programming language doesn't understand the way you're using them and can help you avoid making mistakes. So for instance, by representing these things are strings, you get code injection attacks. So as far as I'm concerned, code injection attacks are basically about the consequence of including as a primitive in your programming language or your framework some function that runs programs as text in some sufficiently expressive language.

In Ur/Web, there is no built-in interpreter at runtime for strings as programs. And that makes a lot of the most common mistakes in web applications impossible by construction. So all these things that are highlighted will either be invisible, or they'll be represented with special types that make clear what kind of code you're dealing with and don't have any sort of automatic coercion from string into any of those special types.

All right, so here's the alternative model that Ur/Web exposes, which gets compiled to the traditional model. So it works in all the widely deployed browsers. But the programmer can think at this higher level and avoid the potential for mistakes that were possible in the previous picture. So we still have the web server, which is in charge. And we still have this fleet of browsers that are trying to use the web server.

But now, the first important change is that when the browser wants to initiate use of a web application, it doesn't just send a string of HTTP requests with a URL in it. Effectively, the abstraction is the browser names a function that should be called where the call runs on the server instead of the client.

And then the server responds with not a string of HTTP protocol text but a strongly typed documentary. So instead of a string of HTML, it's a tree, a first class object in the language. And that is how the program manipulates it, not as a string.

And each of these trees contains within it links, which are themselves basically just references to other functions that you might choose to call on the server. So then the browser, when the user clicks on those links, picks out the function and conceptually calls it on the server, just like the original function that we called to get to this point.

And we have a database interface, which is accessed by the web server throwing queries at the database. And these are not just text in the Ur/Web model. They're strongly typed SQL syntax trees. And then the database will respond back with not text, but a list of records of native values in the programming language that we're working with.

So we don't have to worry about incorrectly converting between strings and native representations, or native representations in any other format that the database might traditionally be presenting to us. And here's a key element of how the semantics of Ur/Web makes it easier for programmers to think about fewer scenarios that can actually happen when the application is running.

There's the standard idea of transactions in the world of relational databases where you can run a series of operations that seem to run with no interruption by other concurrent threads. And Ur/Web adopts that model and builds it into the semantics of the language. So when a single function is running on the server on behalf of a client, then all of its database accesses appear to happen as an atomic unit without any interruption by any other concurrent requests to the same server.

And you can't even avoid this behavior if you want to. Transactions are built into the language. And they really make concurrency a lot easier to think about, and potentially help you avoid security issues that only arise when some rare interleaving happens with a particular combination of requests.

And actually, I want to get to one of the questions that someone submitted for this class that I found intriguing. Ur/Web will detect when a transaction fails because of a concurrency problem, like a deadlock, and automatically restart the transaction. Someone's response to a question said, this might make it easier to launch security attacks that depend on causing transactions to fail because of concurrency issues.

I just wanted to ask the class, what's an example of an attack like that, if anyone happens to have one in mind? If you have a system that automatically restarts transactions that run into deadlocks, how does that cause a security problem, if it does? This is a question I don't have an answer in mind for, which is why I'm asking it. It might also have only a non-obvious answer that no one would come up with on the spot like this, which is fine, too. Yeah.

STUDENT: Can you maybe do some sort of denial of service? If it's going to restart a transaction that you're sending, and you know it will fail, can you just keep restarting that and try again?

ADAM CHLIPALA: OK, so--

STUDENT: So if you could cause the system to do some transaction you know is about to fail and repeatedly fail, it keeps trying over and over again, it would never [INAUDIBLE].

ADAM CHLIPALA: Right, so you'd need at least two threads running at once to do that. But potentially that could work. So you could launch a denial of service attack taking advantage of the fact that contention leads to request handlers restarting over and over again and purposely cause contention and use this as a way to amplify the strength of your denial of service attack beyond what you can get with a traditional model. All right, I can believe that. Yeah.

STUDENT: Is [INAUDIBLE] the only way to cause the transaction to fail?

ADAM CHLIPALA: It is. Well, it's the only way to cause it to fail and automatically restart. Yeah.

STUDENT: Perhaps it could have a third party, which would conditionally fail. And then you could use that to monitor some other user's behavior.

ADAM CHLIPALA: You'd also need a way to observe the fact that it had failed, which you should only be able to do through timing. But that could still be an issue. OK, right, so you can use this as a side channel to see what other threads are doing, because their actions might or might not create a conflict in your thread.

OK, that sounds possible in principle, and very twisty. I'm not sure. It's hard to think of a concrete attack that would work predictably. But it could be a fun exercise. Yeah.

STUDENT: So do the transactions you run-- for each request that comes in, you run a transaction for the code you run at the web server. But when you send that code to the database, does that translate into a database transaction as well?

ADAM CHLIPALA: It is, yeah. The whole execution on the server side is wrapped in one database transaction if the application uses the database. Yeah.

STUDENT: So if you have a transaction that's not going to end up obtaining, do you think [INAUDIBLE]?

ADAM CHLIPALA: Yeah.

STUDENT: Are you telling the database that nothing's going to be updated later? Because presumably, the database doesn't know that.

ADAM CHLIPALA: Yes, so the compiler does static analysis and finds out transactions that need to be read-only. And it creates the transaction in read-only mode, which in some database systems enables extra optimizations.

STUDENT: What about if you read some stuff, and some of the stuff you read doesn't affect what you're going to write, but some of the other stuff you read does?

ADAM CHLIPALA: I see, so you're asking, could we use our knowledge of the semantics of the application to give hints to the database system saying some of what looked like concurrency violations are actually benign, and we don't need to restart at that point? I think the short answer is no. The current implementation doesn't do that. But that would be interesting to look into. I think it would require changes to the database engine, not just the interface in the language.

STUDENT: Usually you could split it into two separate transactions, maybe, or something under certain circumstances.

ADAM CHLIPALA: Yeah, that sounds hard to do right, but potentially worthwhile for-- I don't know how to estimate what fraction of applications could take advantage of that, but it's a neat idea. All right, so transactions are great.

We also have-- so I was just telling you about the model, the old school model of the browser requesting a single page from the web server. We can also have this Ajax style stuff that basically looks like code on the client. It's calling a function that's just marked to run on the server. When it finishes, the result comes back in the client code.

And the result is just a native value in the programming language. You don't have to worry about making it into a string somehow and translating it back. And then we have to take the result and use it to change the page that the user sees. Otherwise, it wasn't a very useful request to make.

So the model in Ur/Web is very different from the standard document object model that browsers expose directly. The basic idea is something called functional reactive programming, which I won't try to explain in too much detail. Because I know it requires a nontrivial grokking of functional programming first, even if we cut off that reactive part. But the basic idea is the document is described in terms of a set of mutable cells, which are sort of the data the page depends on.

And the page itself is something different, described as a function that takes as inputs the values of those cells, and then computes a page. And then the runtime system of the language watches changes to those mutable cells. And when they do change, it automatically computes the consequences for the displayed page and efficiently updates just the parts of the page that have changed based on those cells.

All right, and on each client, there can be many different threads running at once.

These threads are spawned in Ur/Web code and themselves run Ur/Web code. But the compiler needs to translate them into JavaScript to get the browser to run them. So that's one of the services the compiler provides. That's one important point about the threads.

Another key point is that the client side threading follows what's call the cooperative multi-threading model. A thread doesn't have to worry about being preempted by another thread at an arbitrary point. There are well defined operations that signal, OK, it's all right to switch to another thread here. One of them is making a remote function call to the server, for instance, or asking to sleep for a certain number of milliseconds.

But just regular code can't be interrupted arbitrarily. So that means the programmer doesn't need to think about as many interleavings, and it's easier to convince yourself that, say, a particular piece of code avoids some security issue or other bug. Because you can more easily enumerate all the possible ways for the two threads to interact with each other.

And this is sort of a natural model to use given the way JavaScript is usually implemented. There isn't preemption in JavaScript and browsers already. So this is just presenting a threading abstraction on top of the callbacks-based model that JavaScript shows the programmer directly.

And the last piece that one of the built-in abstractions that Ur/Web applications use is channels for passing messages between different machines. So each channel has a type, which expresses what kind of data can flow over if. You don't have to convert things to and from string or JSON or anything else to make this work.

And channels can live in the database. So imagine this picture is showing us there's a channel that was created. It has a write side and a read side, which can go to separate places. The write end is sitting in the database. And the read end somehow made its way to the client and is sitting in the variable environment of a thread.

So imagine that thread earlier made a remote call to the server, which created the channel, returned it to the client, and put it in the database in one transaction. So later, the server decides, OK, I'll query that channel out of the database. And I'll dump a value into it. And it just sort of pops out the other end on the client. And everything is strongly tied throughout this process.

All right, I think this is the last step of my animation here. Any questions about this model before I switch to a code demo?

STUDENT: So how is this different than [INAUDIBLE]? Why do you need a message passage if you already have that [INAUDIBLE]?

ADAM CHLIPALA: OK, so RPC interface is going from browser initiates the call, the server handles it. The message is that the channels are intended for cases where the server initiates the communication. For instance, new email message-- that would be a canonical example. And the client is waiting to hear that there's a new email message. But it can't determine on its own when the next message is available. Yeah.

STUDENT: Are all the messages multiplexed through one connection, or is it [INAUDIBLE]?

ADAM CHLIPALA: They are multiplexed through one HTTP connection. I know there are these newfangled things today called web sockets and maybe some other protocols like that, which didn't exist when this was implemented. This all works over old-school HTTP with one connection for all the messages on different channels.

All right, let's see what's next. Yeah, let me switch to a demo here. So here's a Hello World program in Ur/Web. Probably it deserves more of the screen space than this compilation output. So it looks pretty un-scary at this point, I hope.

The unusual thing here maybe is that this is really the whole program. There's no extra routing logic that explains how to map a URL into some code to run to serve requests to that URL. We just have regular functions of a standard kind of programming language. And the compiler exposes all the functions in your main module as callable via URLs.

And the URL is just formed from the function name. And if there's some nested structure modules, the module's structure is also replicated in the URL. And then we have a function that returns a piece of XHTML syntax. The compiler is actually using a special parsing extension for processing this XHTML syntax. And it's also doing some basic type checking to make sure that different XML elements appear inside others that they're actually authorized to appear inside of.

And I think I compiled this before we started. And it does a not very surprising thing in the browser. And here's the HTML page that comes out. So among other properties, it automatically adds the right XHTML header. And it declares the character encoding for this document. I was mildly horrified to look at some of your assigned reading for this class and see how much time this book spends talking about character encodings and what happens if you're not using UTF-8. I hope I understood that correctly.

This forces you to use UTF-8 so that those horrible things aren't going to happen, I hope. But if anyone sees a way to replicate any of the attacks from that book *Tangled Web* in Ur/Web, or has a hypothesis about something we should try to see if it works, I'd be interested to hear that. And by the way, at any point during this demo, please suggest experiments that come to mind about things we should try, mistakes you might make that you wonder whether this system is able to catch. I think that's the most fun kind of demo. Yeah.

STUDENT: So things like CRSF [INAUDIBLE], you said that [INAUDIBLE].

ADAM CHLIPALA: So cross site request forgery I wanted to explain a little later explicitly. I think the paper sort of explains why cross site scripting can't work. And the reason is whenever you build a piece of syntax, it's an object, a tree of different sub parts of that syntax. It's not just a string.

And you're not going to accidentally turn a string from the user into a tree with structure. You would know if you did that. Because it's hard to write an interpreter. And in Ur/Web, you have to write an interpreter. It doesn't automatically happen for you. But I'll have an example shortly that might also address that concern.

So I want to show you what this syntactic sugar actually turns into in the compiler. So this might look like we could just add some double quotes around the HTML, and then we're back in the normal world. We might wonder, why is it such a big deal the omit the double quotes and put XML instead?

So we can actually take my word for it that this is equivalent code for what this does. So tag is a built in function that builds a tree node of an HTML document. And I'm passing a bunch of arguments that are expressing the CSS styling on that node.

This one doesn't really have anything going on, so it's a variety of different ways of saying nothing. And it doesn't take any attributes. And the tag is a body tag. So that's another thing in the standard library. All of the standard tags are functions with first class status in the standard library.

And then we need to put a "Hello World" text inside it. So we call a cdata function where cdata is the XML word for character data or just a string constant. And we can put exactly the text from below. We'll comment that out.

This should give us the same result as before. Let me see if that worked. OK, and now I'll go back to the actual page. Same thing as before, so this is what that function was really doing at the begin. It's not just building a string. It's calling a series of operations that are designed so that they only allow you to build valid HTML, and they never implicitly interpret a string as code instead of just content that sits there. Yeah?

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: Right, you are anticipating the next few steps. Let me do something less complicated first, which is also potentially worrisome. Let's decide that we're really happy to see the world, so we better put the word "hello" in bold and compile that again. It just shows up as interpreting that literally as text instead of markup.

So this presentation of HTML syntax as a function that builds syntax doesn't have any of the usual syntactic encoding conventions built into it. It interprets things in the way you would want it to. And so the implementation of cdata does what's usually called escaping.

But the programmer doesn't need to know there is any such thing as escaping. You can just think of it as, here's a set of convenient functions for building a tree object that describes a page. Did I see a question over there?

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: You want to see the HTML that it generates. OK, it's going to be not the most exciting thing. I don't know if that's [INAUDIBLE]. I can make it bigger, but then it doesn't fit on one line. So let me know if I should make it bigger. It just put in the usual escapes for the less than character with an ampersand.

STUDENT: So given that you're using XHTML, couldn't you just use the cdata [INAUDIBLE] instead of doing manual [INAUDIBLE]?

ADAM CHLIPALA: Probably. That would require me knowing more about XML than I do. All right, so there was another question about JavaScript URLs, which is a good one. If we allow JavaScript URLs, then we have a back door for automatic interpretation of strings as programs at runtime. And that causes all sorts of issues.

So let's try to avoid that. I'll switch back, first of all, to the shorter version of this. And then inside the body, I'll make this multiple lines. And let's put a link that tries to do something appropriate. We'll leave some room for error messages here. This is working correctly.

Invalid URL, JavaScript something, passed bless. So bless is a built in function that is the gatekeeper of which URLs are allowed. And by default, no URLs are allowed. So certainly this one is not allowed. And in general, it is a bad idea to write your URL policy so that you can create values that represent JavaScript URLs. Because then all sorts of guarantees that you might like are invalid.

To make it a little clearer how it works, let me factor this code into a separate function called a linker that takes in a URL. So URL is a type. It's not just string. It's a type that stands for a URL that is explicitly authorized by your application's policy.

And so we can [INAUDIBLE] XML. And instead of a constant, I'll just put u here. And so I'm using the curly braces like in some popular HTML template frameworks to indicate inserting some code from the host language inside the HTML that we're building.

And this is all done in a way where it's type checked statically. So the system will check, yeah, this is a spot where a URL belongs. And this says it is a URL. So that's fine.

And then I can explicitly expose the call to bless by saying, let's just call the linker function here on the result of blessing that URL. We should get basically the same error message as before. There's some program analysis going on here to figure out-- I guess it doesn't need that. Because this string is passed directly to bless.

And we can see-- I couldn't wait to run this for you at runtime and discover the failure. But I can tell it's definitely going to fail. So I'll just make it a compiler error. This URL is not going to be accepted by the URL policy.

STUDENT: So if you didn't have the [INAUDIBLE]?

ADAM CHLIPALA: If I left out this call to bless, it would be a much more basic compile time error. You have a string and need a URL. They're different types. All right, but let's make this a little more interesting. And I'm going to open up the configuration file for this demo.

It's pretty short, as these things go, at least if you look at any Java web application framework. They have these gigantic XML files for configuration. This is a little nicer than that, or so I claim.

We can add a rule that says, anything on Wikipedia is allowed. And then we can put the Wikipedia URL in here. Now we're in good shape. What's missing? Oh, I guess I don't remember the URL scheme for that. But we got to the website. That's good enough.

All right, so the big idea here is to have an abstract type of URL, just like you could have an abstract type of hash tables that encodes invariants about how the hash table looks and prevents code from reaching inside the array of the hash table. We can do the same thing for URLs. And the system enforces via this bless function that every value of this type has passed the appropriate check at some point.

And for instance, with this policy, we know there will never be a JavaScript URL. And it's safe to take a URL value and use it as a link. It won't break the basic abstractions of the language. Yeah.

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: OK, so we have to try something like that. And this should go through. And then the browser knows it's a quote. And we can look at the source. That is because it was escaped in the right way.

STUDENT: But can you still use-- so JavaScript allows you to say, [INAUDIBLE], and then specify inline JavaScript there. Is that something that [INAUDIBLE]?

ADAM CHLIPALA: Yes and no. So we can put body onload. And instead of JavaScript, you put some Ur/Web code that does something. So it would be a disaster to interpret JavaScript code in string form as a program there.

But we can put code of the same programming language you're working with already escaped in with these curly braces. And then it automatically gets compiled to JavaScript to run on the client. All right, any more questions? Yeah.

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: I think it's everything? Is it embarrassing that I said everything? Is there something that shouldn't be allowed?

STUDENT: [INAUDIBLE].

ADAM CHLIPALA: I see, so symbols that would independently have funny things happening with software execution would confuse the human user?

STUDENT: [INAUDIBLE].

ADAM CHLIPALA: OK, I remember reading some of that stuff. And maybe it said the new browser versions avoid those problems. But some old ones will get confused. It's possible this will create problems in the old ones that are too permissive. I'm not sure.

But at least all these things are going to be interpreted as UTF-8 if they go into the document. So if there's some problem with a different encoding, it shouldn't be applicable here. Yeah.

STUDENT: The string of the [INAUDIBLE], right now it's checking a compile time that that string is on allowed URL. But if you compute a string at runtime, does bless perform a check at runtime whether or not the string is allowed, or are you not allowed to--

ADAM CHLIPALA: So let's a write a form to test that claim. So we can put a form in here. And form wants us to enter URL in a text box called URL. Then we can have a Submit button.

When you click on it, it should call the linker function with a record of one value for every field in the form. In this case, there's just one field called URL. And so linker will get passed a record that contains the URL as a string type. And then we'll explicitly try to bless it up there and see if it works.

This is an example of an exciting type error message, which is admittedly sub-optimal in some ways. Here's one of those things that won't make any sense if you're not familiar with Haskell. I forgot a return. But at least now it looks more like a Java program. Have a string-- let me scroll to the end, do one of these, sort of copying the full type of all the attributes that this tag can take.

And I also forgot to say, this is now a full page. So we can't use an a tag until we're inside a body tag. And this is the abstruse type error message for that property. OK, so now let's see what happens. URL is-- yay. There we go. So that was a somewhat long and not necessarily super exciting answer to your question. Yeah.

STUDENT: The URL [INAUDIBLE], are those just for [INAUDIBLE], or is it more restrictive than that?

ADAM CHLIPALA: It's more restrictive. It's currently just constants and prefixes. But you can also have disallow rules. And they run in the order that you write.

STUDENT: Oh, so if you stick to disallow JavaScript [INAUDIBLE] that if you put a line break in the middle of the word "JavaScript," it will still interpret it as--

ADAM CHLIPALA: That would be too bad. That's why it's good to stick to the white list approach instead of the black list approach. So you probably want all the rules to start with a particular protocol, like HTTP, and only allow things that fall in your approved set of protocols. That's what I recommend it is. Yeah.

STUDENT: For many sites, you might let users share links, in which case, you need to allow links to anywhere.

ADAM CHLIPALA: You can allow links. Well, do you want your users to share JavaScript links or, I don't know, Flash links, or whatever's allowed? You see, you can white list all the HTTP, HTTPS, URLs and be in good shape for most websites. That would do that. And the guarantees are a little weaker compared to allowing only particular URLs. But you can at least ensure that there's no automatic execution of the string as a program.

So let me pull up one of the examples from the paper, which is this one, an example of a simple system with a set of chat rooms represented in the database. And the user can click on a link to go to a room and then send a message. This was the first of several variants on that scheme.

First, I'll point out I'm going to recompile this. And then magically, all the database tables that it declares are going to be added to the database. And we can now just start using the application. But first, we have to add some rooms.

So let's open our [INAUDIBLE] interface to the demo database and insert into the room table some values like one and two. Hopefully these are here now. OK, and we go in there, and we can entertain ourselves all day long sending strings of text.

Maybe a little more interesting, you can try to send HTML, and it just gets handled right away. That's the basic functionality there.

And just to quickly go over some of how this works again, so we have these two SQL table that are just declared in this first class way inside the programming language. And we give the schema of each table. And then later, when we try to access those tables, the compiler will check that we're accessing them in a way that's consistent with the schema from a typing perspective.

So we have a table of rooms where each room is a record of an ID, which is the integer, and a title, which is the string. This is the type we were just generating records in. And I created a few rooms at the SQL console. And we also have messages that each message belongs to a room. And it has a time when it was posted. And it has some text, which is the content of the message.

And let me fast forward to the main function. We run an SQL query. So here's an example of SQL syntax embedded inside of Ur/Web. I don't want to go through the expansion of this one into calling functions from the standard library. Because it's pretty verbose if I do that. But take my word for it, this is de-sugared into calls of functions in the standard library that represent the valid ways of constructing an SQL query.

And those functions have types that cause them to type check the query for you, not just guarantee that the syntax is reasonable. So this gets de-sugared into an indication of an SQL query. And then the code here is basically just looping over all the rows that come out of that query and generating a piece of HTML for each one.

In particular, we're going to take the title field of a query result and convert that into HTML with this notation that involves curly braces. And the square brackets are additionally saying, this isn't literally a piece of HTML yet. But please convert it for me in the standard way. So we can do that with strings and integers and all sorts of other types. Yeah.

STUDENT: So if that contained malicious HTML or something, would that be filtered out?

ADAM CHLIPALA: It would be. So in the usual way of talking about these things, escaping happens in the way you'd want it to. In Ur/Web, you can just think of this as building a tree. This is a node that stands for some text. Obviously text can't do anything.

STUDENT: So if that title was User Control, and someone made a chat room with the title Alert something, that would not be JavaScript?

ADAM CHLIPALA: It wouldn't automatically be interpreted as JavaScript or HTML or anything else. It would just be text only. All right, so we have this title. And let's wrap an a tag around it. And instead of href, the usual way to do a link in HTML, we use the link attribute, which is sort of a pseudo attribute in Ur/Web, which takes as an argument not a URL, but basically an Ur/Web expression. And the meaning is when you click on this link, please run this expression to generate the new page that should be displayed.

In this case, we're calling a function called chat, which is defined up here. And here's what it is. I won't go too much into the details. But we have a few more SQL queries using a variety of standard library functions for different ways of using queried results.

We generate this HTML page. And we say, you're in the chat room. Here's the title. We get the same kind of escaping there. And there's a form where the user can enter some text. That's the form that I used to demonstrate this a few moments ago.

And the Submit button of the form has this ask attribute that is containing say, which is the name of a function in Ur/Web. And here it is. So when we submit the form, we call this function. Run some more SQLs. Insert a new row into a table.

We automatically jump in the ID of the chat room and the text field that came from the form. And these are automatically escaped as necessary. But again, you don't have to think about escaping in that way in Ur/Web. Because this is just syntax for building a tree. It doesn't stand for a string. So there's no way to have strange things happen with parsing that you don't expect from the way that the syntax is written. Yeah.

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: Yes, so from the fact that there's one widget, one GUI widget in this form, and its name is text, and that one is a text box, the compiler infers that the record that stands for the form result should have one element called text that is of type string. And this encoding the forms, the typing rules for it are not built into the language.

You can actually with the type system in Ur express as a library, what are the operations for building forms, and how do you check that they're used correctly, including what consequences they have of the types of the functions that actually handle those forms? [INAUDIBLE]. Any other questions about this code before I switch to the next step of the sequence in versions from the paper, which is only a small change?

All right, then here's what I'm going to do. It's basically taking advantage of a way to get enforced encapsulation of different parts of an application that Ur/Web supports, which is at least only rarely supported elsewhere. I'm going to take this room. I'm going to take some of these definitions here and put them inside a module that encapsulates some of them as private.

In particular, the database tables are going to be private. So no one can access them directly. They can only access them through a set of methods that we provide. So one method runs inside a transaction. That's what this type says.

And it produces a list of records with ID and title fields that stand for which rooms are available. And we'll also just expose this chat operation. And one thing I've done here is I've introduced a name for the concept of an ID. I won't just say that an ID is an integer. I'll say it's a new type. And the only way the outside world will ever get one is to list all the rooms. And the only way the outside world can ever use one is to call the chat function on it.

So just like, say, the abstract type of a hash table inside a hash table class where the details of what is an ID and how do they get produced internally are private to this module. And the client code that calls this module isn't going to need to use them. So I'll use this syntax to put everything down here inside the module so it's not exposed to the rest of the code by default.

And we also are going to want to implement this rooms method. We already happen to have chat around. But we can implement rooms in a simple way as using another standard library function for interpreting a query in a useful way.

Let's just select everything from the room table ordering by title. And as usual, this query is type checked for us. And the system determines, OK, this expression is going to generate a list of records that happens to match the type that we declared in the signature of this module. So now outside this module, no other code is allowed to mention the room table or the message table.

So we can, at least from the perspective of this application, enforce whenever invariance we want on them. We can even hide secrets inside of them that would be a security problem if some other part of the code was able to get a hold of them. Yeah.

STUDENT: But couldn't some other part of the code just declare table room as well?

ADAM CHLIPALA: That would be a different table. We could do that, actually. It's got to be in here. I think this should have no effect on the behavior. I think in this case we're going to get something funny happening. Let's put this in a different module just to avoid something goofy.

Great, so we can do that. And we can do whatever we want with this table. And I'll compile this in maybe about 30 seconds and we'll see what happens. But it's actually a different table, just like if you have the same private field name across two classes in Java they're different field names. Yeah.

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: So you're suggesting we have, inside this module, an abstract type called room, which contains both the ID and the title. Is that right?

STUDENT: [INAUDIBLE]?

So I think what would work to do instead is instead of type ID have type room, have room determine a list of rooms, and chat take a room as an input. Is that what you have in mind? So what would happen then is when we call the chat function, it'll actually be called via a URL given the way we use this eventually.

That would be passing the ID and the title within the URL in the URL representation for a function call. And we only need the ID to implement that function. So it would be a little wasteful of space and might look gross to the user to have to have the title passed along as an extra argument in the invocation of chat via a URL. Does that make sense?

Or maybe another way of saying it, if I have this one [INAUDIBLE], is look up at the URL bar. The ID of the channel we're going into is serialized automatically in the URL at the end here. And if we were passing a record that contained an ID and a title, the title would be serialized, too, which is at least a little counterintuative.

OK, the last thing we need to do-- actually, it might be instructive to make just a shallow change to this code, reference the room module there, and then try to access the room table like before. This shouldn't be allowed.

This would be like being able to read and write the private fields of a class in Java. And indeed, we get a pretty straightforward message basically saying, this right here is an unbound variable. There's no table called room in scope.

And we could mention this extra one that we created just for fun. But then it would be a different table. It wouldn't be a problem that we could access that. So instead, what we should do is I'll break this into two parts.

We'll start out by just calling the rooms method, and then do a slightly different thing to read its elements, map over the list of results that gives-- what did I call [INAUDIBLE]? Map all the list of results instead of the other way it was working, which was roughly equivalent except for using different data types. Let's see how this goes.

All right, so I'll go back here. And we can do all the tremendously exciting things we could do before. But we have this encapsulation. And you can sort of think of this room structure as now it's a library, and you can call this from all sorts of different places that want to have this functionality.

You don't have to worry. There's different places that are going to break the internal invariance of the system. Maybe you want to know that once a message is added, it will never be deleted. It's always there in the logs. This structure enforces that independently of which other code the room module might be composed with, for instance. Yeah.

STUDENT: Say you change the definition of room, [INAUDIBLE]. What's going to happen to the database table?

ADAM CHLIPALA: It'll be a little sad. We'll have to manually run an alter table command if you want to save the old data. But when the application starts up, it queries the system database catalog and checks that the schema still matches what it expects. So you'll get a static error then. And that will hopefully give you a hint about what you should change in the database.

STUDENT: But it wouldn't automatically drop your database or something?

ADAM CHLIPALA: I hope not. I don't think it should do that. And you can imagine tweaking the compiler to understand the evolution of a database. I think you have to write alter table commands to run. It doesn't do that right now.

OK, so now let's talk about cross site request forgery and preventing it. Actually, before we do that, let's look at the code on this page. We have a traditional looking HTML form that gets generated here. And there's certainly no cross site request forgery protection in here, which I think is good. Because as I understand cross site request forgery, the problem is there's some implicit context that your application sends on every request.

So there's some attacker out there who doesn't know your implicit context. Let's say your password is stored in a cookie, for a really simple example. And when the attacker tricks you into following a link to the application, your browser sends the implicit context automatically and causes the application to do something the attacker could not have done directly.

In this case, there's no implicit context. So there's no risk of a cross site request forgery. Does anyone want to dispute that characterization before I go on? It could be educational for me. All right, so now let's add some implicit context. And the system is automatically going to deploy the right countermeasures based on program analysis that realizes now there's implicit context.

In particular, we just throw in a cookie here. As another example of module capsulation, actually, I'll put in a whole sort of user authentication system where we have the user accounts and abstract types of IDs and passwords. So you can't just build the value of either of these types directly. You'll have to go through some kind of approved method of building values of these types.

And I'm actually going to expose the table directly in the signature. And I'll put a constraint on it, too, saying the ID form is a key for it. [INAUDIBLE]. But the thing is, on this user table, ID and password are abstract types. So the code can't actually look at the password.

And it can't generate all IDs in sequence and try them against this table. Because the type is abstract. There's no way to make an ID. There's no way to make a password. They just come out of this table, and they're opaque tokens.

But we might want to allow them to be input from strings. You might want to allow one direction of conversion between strings and these types. So that's what I'll do here. Basically, the details I don't want to try to explain. But this is like a declaration, OK, you're allowed to convert strings into IDs. For those who speak Haskell, this is a type class instant. For those who don't, it's permission to turn strings into IDs.

We're going to leave out the other permission. We don't want to be able to turn an

ID back into anything. And the password-- let's do the same thing. We want to be able to read a password from the user but not take a password and turn it into a string where we can actually tell what the user entered.

So other parts of the code will be able to accept password input from the user, convert it into this type, and ship it off to the user module and have it be checked. But what they can't do is query the user table and get all the passwords in a form where they can actually extract the text from them.

Then we can have a login method that takes these two components and just runs for its side effects, which is effectively what that code says. We'll also need a way to tell which user is logged in. That is a code that runs a transaction that produces an ID.

All right, so step one, we can just copy this definition. And I'll fill in what these actually are. It turns out-- surprise, surprise-- user IDs and passwords are both strings. But outside the module, that won't be exposed.

And now we're going to create a cookie. So cookies are another thing that's built into the language. Effectively, they act like mutable global variables that have one copy per client that uses your application. So we're going to create a cookie that on each client will store basically just a copy of the same two fields that we have here.

So this cookie is private to this module. Other parts of the code won't be able to read the cookie, because they just don't have this private field and scope. So no one else will be able to see directly the ID and password that are stored for this user. But they will be persisted across different page views, just like you would expect for cookies usually.

I'm going to give it a login function that's going to run some incantation to check against the database whether this is really a correct pair of username and password. It'll just check, can we find a row in the database that has this user ID and has this password?

If we find one, then yes, good, that's the correct value. Let's just save it into the cookie. We use a method that modifies the cookie value. And we have to put some things in here, like just for simplicity, I'll say this cookie never expires. And I don't want to run SSL here, so I'll say it doesn't need to be secure.

But if you really care about security, obviously you would write secure equals true. And if the check failed, then we can-- I don't know. It doesn't matter. If it signals an error, execution stops with this error description.

Finally, we can create this function that tells who the user is logged in as by getting the current cookie value. And then it might be none if the user hasn't logged in yet, in which case, we can have a different error message. Or it might be some record of exactly the type we used up there. So I'll just copy some of this here. Let's run the same check there. If it worked, then we'll just return the ID part of the record that we just verified against the database. Otherwise, [INAUDIBLE].

So let me just type check this to see if this is on track, that part-- Oops, capital Id. All right, so the important is there's all those implementation details. But from outside this module, we think of it in terms of the interface up there. There are some unknown types of IDs and passwords.

This table of users expressed in terms of them were allowed to turn strings into IDs and passwords, but not the other way around. And we have these two methods to log in in the first place and to check which user is logged in at this point. Any question about this? Yeah.

STUDENT: Do you need to expose the user table?

ADAM CHLIPALA: Because I want to use it as a foreign key later. That was the reason I did it. It's not that great of a reason. All right, so we're almost at the point where I can show you CSRF protection in action.

We have to actually start logging in. So that's easy enough to do. OK, so what can we do here? Let's just add another part of this page that says, here's where you log in. This is the form.

This is where you would put the username and the password. And then click on the button. It's trying to go to call a function called login, which we'll define in a moment. Let's define login as a function that does these things. It's actually just a wrapper around calling the login function from that module where we take each of the components and convert it from string to the abstract types.

That's what read error is doing. Error means if it doesn't work, just abort execution instead of signaling the failure with a special return value. Here's both of those, login and then jump to main. So now we should be able to log in. Let's check if that's true.

OK, so that was [INAUDIBLE]. We'll probably want to create an account to allow us to log in. So let me [INAUDIBLE]. So now I should be able to log in as a. OK, and take my word for it. There's now a cookie set to record that information.

And then let's go back in the chat room and send a message. We didn't actually add any access control here yet. So there's not much going on here. But we can check to see.

There's a cookie. But the system has determined that we're not using the cookie. When we submit this form, the cookie is not read. So there's actually no need to add any CSRF protection here yet. So now we have to add the way to use the cookie. And then we should see the protection appear. Yeah.

STUDENT: What are the contents of the cookie?

ADAM CHLIPALA: What are the contents of the cookie? The contents are exactly what you'd expect from the code. In other words, the cookie is declared as having type this record, an ID, and a password. So that's exactly what's in there in a particular serialized form.

So now let's actually use the cookie. And we should hopefully see despite the fact we're going to use the cookie indirectly, because we're going to use it in the room module, which doesn't even have the cookie in scope. But we'll call methods of the user module, which indirectly are using the cookie. And then the system will realize that means we have dependency on it.

So let's make this really simple and just say call the whoami method. And I'm actually just going to ignore this. Or we can do this. Let's decide this a user we created is really special. And only this user is allowed to post anything. And we'll fail if we're not a. All right, let's see if this works. Did I forget a slash somewhere? Oh, yeah. [INAUDIBLE].

Oh, I expect him to be a string. But it's actually an ID. So let's just read a into an ID just like we did below to process login. And we haven't exposed that the ID type supports equality testing. So I'll just add that to the user module. And then that should work.

ID supports equality testing. And we should be OK. So now we've brought in the interface. Now we can do more things with the ID, which could trigger some security issues. But it lets us add this access control check, so let's see how that works, go back to the main page to [INAUDIBLE].

All right, now the form automatically has a hidden input name sig, which is a cryptographic signature of the values of all of the cookies. And it's signed using a key that's a secret for the server. And when the form is submitted, the application knows, because the compiler told it, that it should be checking signatures for the following set of operations. In this case, the only one is this say operation. Yeah.

STUDENT: Does the signature have any sort of time stamp as well?

ADAM CHLIPALA: It does not have a time stamp.

STUDENT: Otherwise, if the attacker ever saw this live, they could pretend to be the user. It never expires.

ADAM CHLIPALA: It never expires, right. So that's something that could be changed just by modifying the language implementation without modifying the applications, and then deployed instantly. But it's not there now. And I can see why that could be a useful thing to add. Question, yeah.

STUDENT: You could also fix that by just putting an [INAUDIBLE] as well.

ADAM CHLIPALA: That's true. You're right, you can change the application to purposely modify the cookie data frequently enough that the signature would go out of date. That's also true. So we've got 10 minutes left. Any requests for things that someone particularly wants to see before class is over? I can start showing some Ajax stuff by default if no one has another request. Yeah.

STUDENT: Can you remap the URLs?

ADAM CHLIPALA: You can, yes. So what remapping would you like to see?

STUDENT: Any. I just want to see how it's done.

ADAM CHLIPALA: OK, so the compiler is assigning-- as we can see back over here, we called the say function. And basically, that function call is serialized as a particular URL form. Maybe we don't like that form. We decide we're going to rewrite URL so say is inside the room module, inside demo.

Better put this up top so it runs before these other rewrites-- rewrite url Demo/Room/say into Demo/Room/speak. And hopefully that's what I want it to. Let's see what happens. Yep, and you can have wild cards in those rules also to map one prefix to another one.

And the compiler will enforce that every function has a distinct URL schema. So if you add a rule that causes a clash, you'll get [INAUDIBLE]. By default, the automatically generated URL schemes are [INAUDIBLE]. You can break that by using this feature. Any other requests? Yeah.

STUDENT: So you mentioned that the HTML [INAUDIBLE] is not compiler specific. It's like one is a library. Are there other libraries for other formats as well?

ADAM CHLIPALA: There are other libraries that don't do type checking at the same level of richness. But for instance, there's a library for serializing and de-serializing JSON. And most of the automated way that's driven by type structure. So you can do things like that that aren't as integrated with the compiler. Yeah.

STUDENT: Presumably you'd still want to write JavaScript. Is there any--

ADAM CHLIPALA: I don't. You do.

STUDENT: Right, no, but for, say, I don't know, you want to animate things on the page. There are still things where--

ADAM CHLIPALA: Let me load the Ajax version of this. And that might answer your question. [INAUDIBLE].

All right, so this version has client side code. Let's just [INAUDIBLE].

Believe it or not, this time the add worked by an Ajax call. And you get things like, here's a button tag. And it has an onclick attribute that when a user clicks the button, all this code here runs on the client side. But it's Ur/Web code. It's not JavaScript code. The compiler translates it into JavaScript for you and guarantees that it maintains the properties that we want for the abstractions in our list, as long as the user isn't mucking around with it manually [INAUDIBLE].

STUDENT: I'm more thinking that there are a lot of [INAUDIBLE] libraries out there today that do useful things, and in many cases complex things if you want to recode everything yourself. Is there any way interfacing JavaScript from Ur/Web?

ADAM CHLIPALA: Yes, there's a foreign function interface, which lets you give Ur/Web function names to JavaScript function names and call. But then whenever you use the foreign function interface, you don't get all of these nice properties like construction anymore. You have to be very careful.

And to some extend, you have to understand the implementations of some of these abstractions to avoid disturbing them. While I have this code up here, let me just show you. We still have the same say function as before, roughly. But now, instead of calling it via a link, we just take the function call, which is populated with arguments that come about from the context of this onclick handler.

And we just wrap that function called inside the RPC syntax. And that means this is a function call on the client, but run the call itself on the server with access to the database and other server resources, and then shift the result back over here. And it's written in this direct style here instead of the callbacks that you need to use in JavaScript usually to accomplish a remote server call [INAUDIBLE]. Yeah.

STUDENT: [INAUDIBLE]?

ADAM CHLIPALA: The client is allowed to call anything in scope. So you just have to use scope the way we usually use it to hide private fields and so forth inside of an abstraction. I mean, because there's a call here, the functions we're allowed to call are the ones whose names are in scope. This name happens to not be in scope here. So we couldn't call it directly here. But because it's in scope up there, we're allowed to call it. Did I see another hand?

Let's see, is there anything else interesting about this version that I wanted to mention? It involves an implementation of a GUI widget using this functional reactive style, which is cool from a programming modularity perspective but maybe less interesting from a security perspective. But here's an example of calling a method of this abstraction of a portion of the page that displays a list of lines of text that you can add to but never delete from.

And you can actually enforce that. Because we don't have the dom here. It's not that any part of the code can reach into the document tree and mutate it and change the log and delete lines that were previously added.

The more functional style here means you can actually have a GUI widget that owns a part of the page and controls exactly what's shown there, and bugs and other code can't interfere with computing what shows up there. This is probably good point to stop, unless there are any last questions.

STUDENT: Channels?

ADAM CHLIPALA: Channels. I don't think there's enough time to properly demonstrate channels. But there's code in the paper. And there are all sorts of demos and tutorials on the website for this project. Yeah.

STUDENT: It's really hard writing correct [INAUDIBLE] and compilers. How do you mitigate problems that might be present from the abstraction layers themselves?

ADAM CHLIPALA: Get people to use it and report bugs. That's the best I have for you. I guess the idea is compilers like this should be written much less frequently than new applications. So to condense all the bug finding in this one place is still an improvement, even if it's not done in a particularly principled way. Yeah.

STUDENT: Just out of curiosity, how are [INAUDIBLE] files handled?

ADAM CHLIPALA: You can use that configuration file I showed to map them into parts of the URL space. Or you can manually produce values in the program that stand for files and ask to return those as the result of the page. There are a few different approaches. Yeah.

STUDENT: Why Ur?

ADAM CHLIPALA: You're asking how I chose the name?

STUDENT: Yeah, like why--

ADAM CHLIPALA: Oh, you're asking why you want to use this.

STUDENT: No, no, the name of the language, just out of curiosity.

ADAM CHLIPALA: So Ur language is a concept from linguistics to describe the language that is the ancestor of the modern languages. And the idea is in this language, you can embed all sorts of other languages inside it. So it's sort of the ancestor of all those.