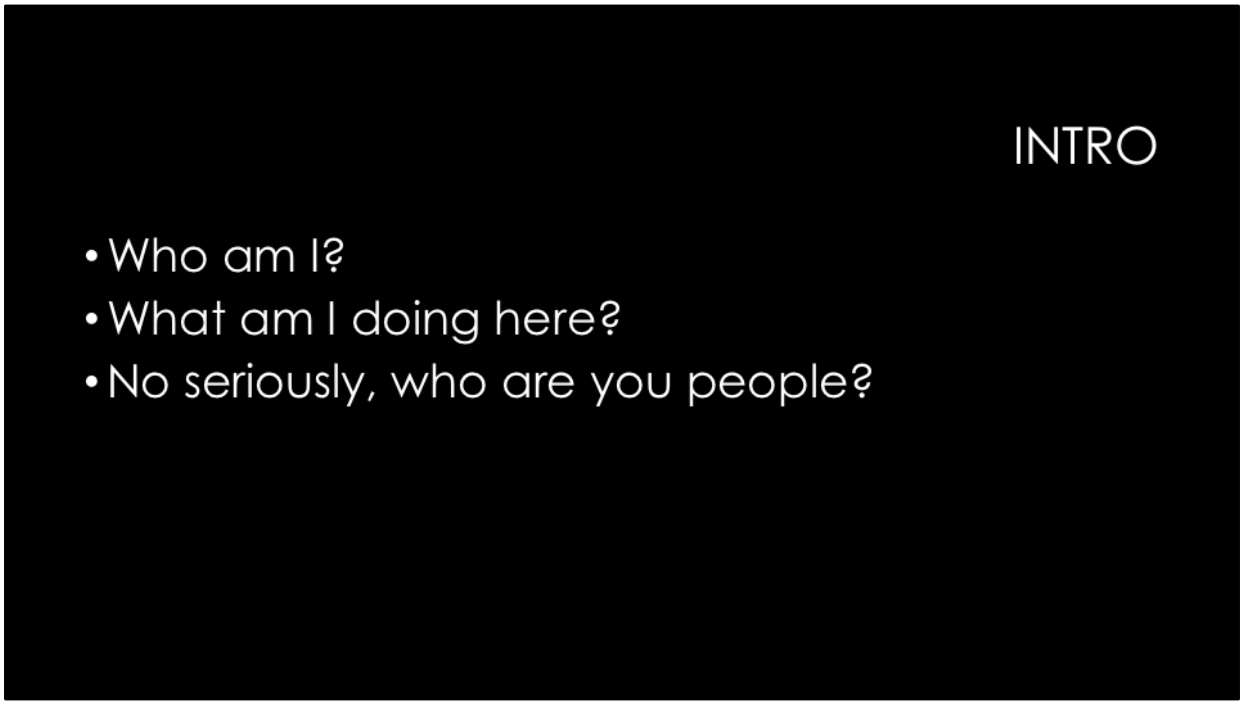

3 WAYS NOT TO USE SELENIUM
Jain Waldrip,
Blackboard



Hi! My name is Jain.

I hope you enjoy this presentation,
but like Mitch Hedberg said,
you can't please all of the people all of
the time, and last time,
all of those people
came to my
presentation.

I work at BLACKBOARD,
on the LEARN project

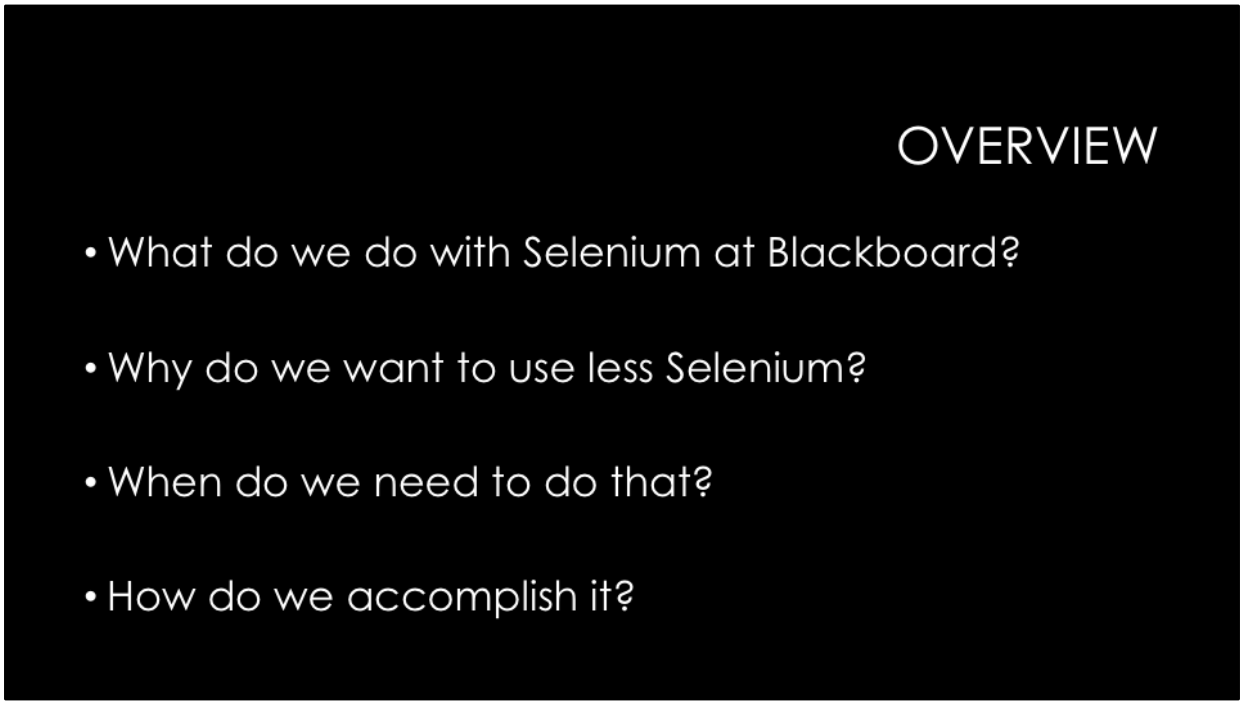
If you don't know what that is, we'll talk about it in a moment

I do AUTOMATION, and I do kind of a lot of it

we use Selenium to automate UI tests at Blackboard

and as you will see,
we do a LOT of it

I would like to talk to you
about 3 ways to use Selenium better
by using LESS of it



In this talk, I intend to
answer the following questions and
introduce a technique that may be new to some of you

What?

We'll talk about how we use Selenium
on the Blackboard Learn project
and look at some exciting numbers

Why?

We'll talk about why we sometimes
need to use less Selenium

When?

We will take a look at when
it might be a good time
to use less Selenium

How?

Finally, we'll look at some methods
that can be used
to decrease a test suite's
dependency on Selenium
There will be a code example
for one of the less commonly used methods,
HTML form injection, which we'll talk about soon

But first, let me tell you a little

CAUTIONARY TALE

Outside of work, I am a fire performer.
That's me, by the way. Spitting fire out of my mouth.

My favorite prop is the FIRE STAFF.
It's like a regular staff, except that it's on fire.

One of my favorite moves is to take a wide stance
and weave the staff under my legs.

Audiences love it because it is SCARY.
You can actually see the fire touch my skin.

By the way, this is a story about a time
that I actually lit my own butt on fire.

ANYWAY.

I was at a burn once,
wearing an extremely gauzy skirt.

Air flow is good for your skin when you burn,
as fire is, unsurprisingly, really hot.

Air flow is also good for fire.
I think you see where this is going.

I did one of my favorite moves,
that weave under the legs
that I mentioned earlier.

It's a very dangerous move.

I was extremely cautious and calculated with my movements.

. . .

It was fine. Nothing went wrong.

AND THEN

I brought the staff in close to my body for a figure eight weave.

This is one of the first moves any staff spinner ever learns.

If I have done this weave once,
I've done it a thousand times.

And then my gauzy skirt caught fire.

In fire performance, it's not a matter of "IF you burn yourself."
It's a matter of WHEN.

When a gauzy cotton skirt catches fire,
"WHEN" is "in about 4 seconds."

We were prepared, so we put the fire out pretty quickly,
but not quicker than 4 seconds.

The moral of this story is,

even if you've done something a thousand, even a million times,

it could ALWAYS go wrong.

In about 4 seconds.

Just like in automation.

You may not know something can go wrong
until the millionth time,
when it does.

So ALWAYS be cautious.


WHAT DO WE DO?
Selenium and Blackboard Learn

Blackboard is a global educational software solution provider
I read that on our website, so I know that it's true

we have over 30 products

If you've ever taken an ONLINE COURSE
you may have used one of our products
or something like it

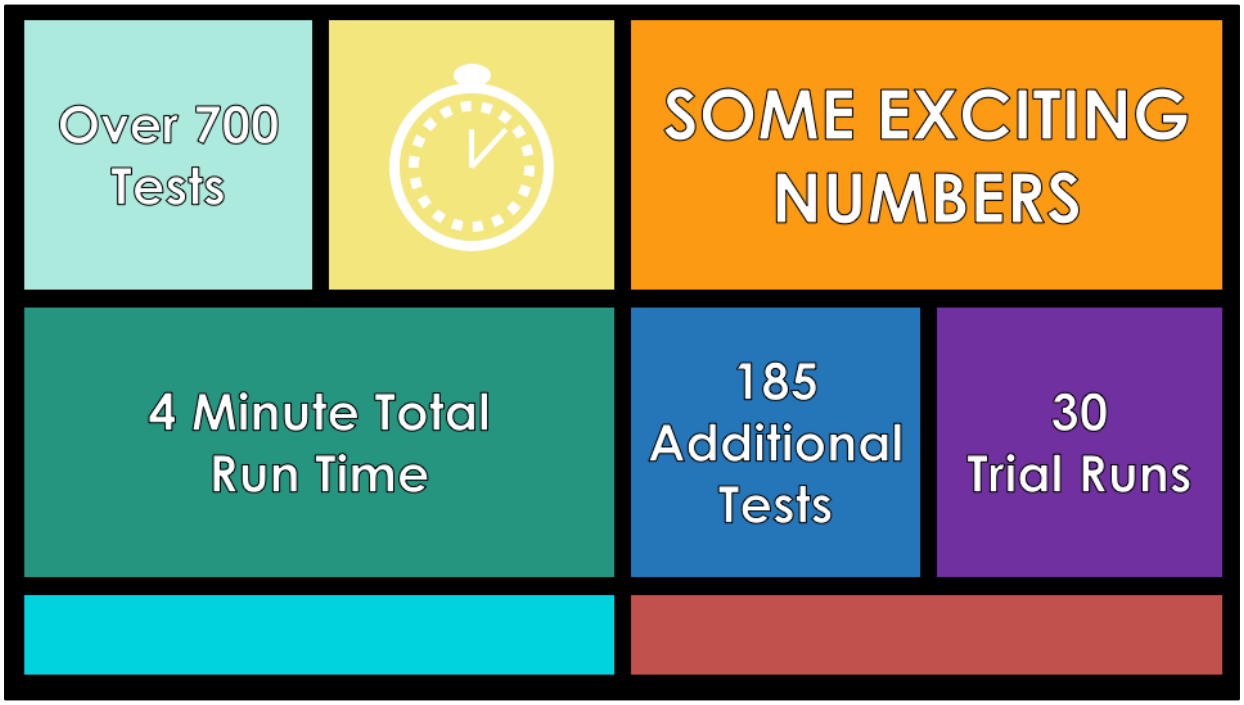
If not, then if you've ever taken an ONLINE QUIZ
it's not really like that at all
but it's close

but we do offer quizzes, tests,
and homework integration in some of our products
Like LEARN
Our flagship LEARNING MANAGEMENT SOLUTION

it needs to be

RELIABLE
and STABLE under HEAVY USE
think MIDTERMS and FINALS

I work on the PRODUCT DEVELOPMENT team for LEARN



Our main UI test suite
consists of over 700 tests

runs against each commit
on a feature branch
and each approved Pull Request
including during the development of new tests

An additional suite of 185 UI tests
in the Critical Usage and Slow Test suites
run against every build of the development branch

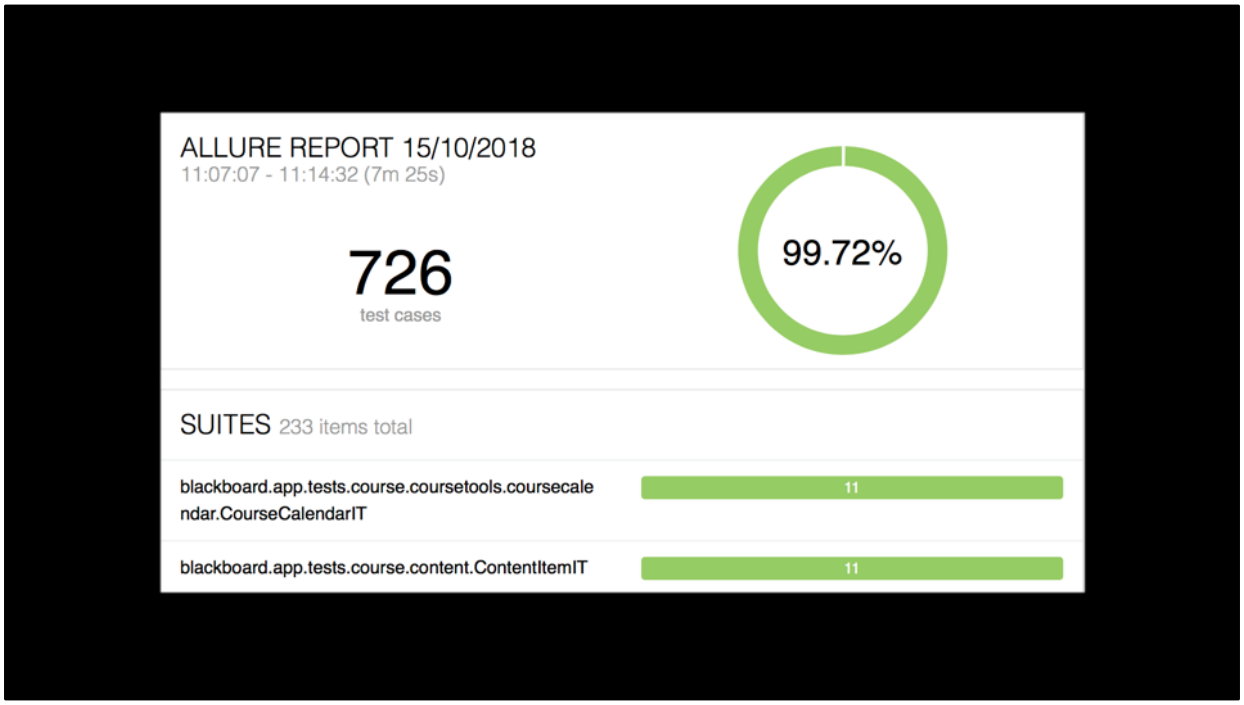
Critical Usage
includes tests
for previous regressions

Slow Tests
are those that
take more than about 2 minutes, start to finish

We have an average of 330 test suite runs per day
entire suites, not individual tests

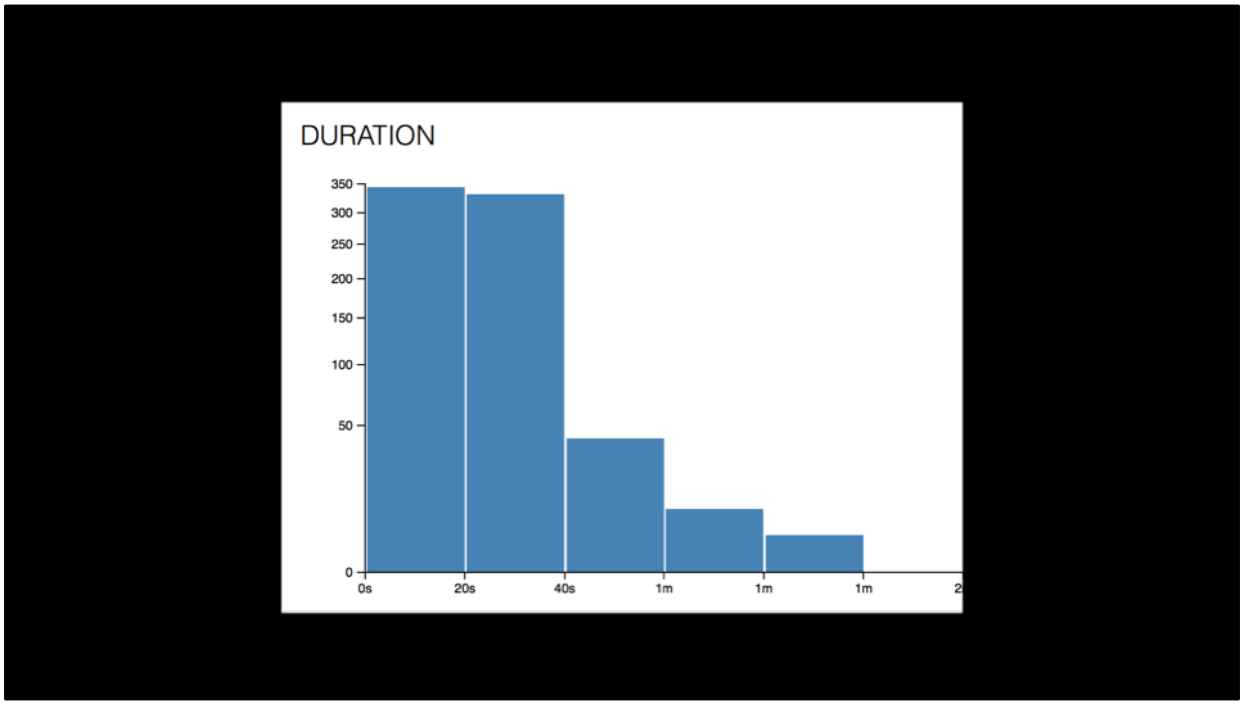
New tests are ”baked” in our test oven
which executes 30 runs
of each new test
to ensure that it’s ready for daily use
with no erratic behavior

Through parallelization,
we have reduced
the actual
run time
for our Selenium tests from several hours
down to about 4 minutes



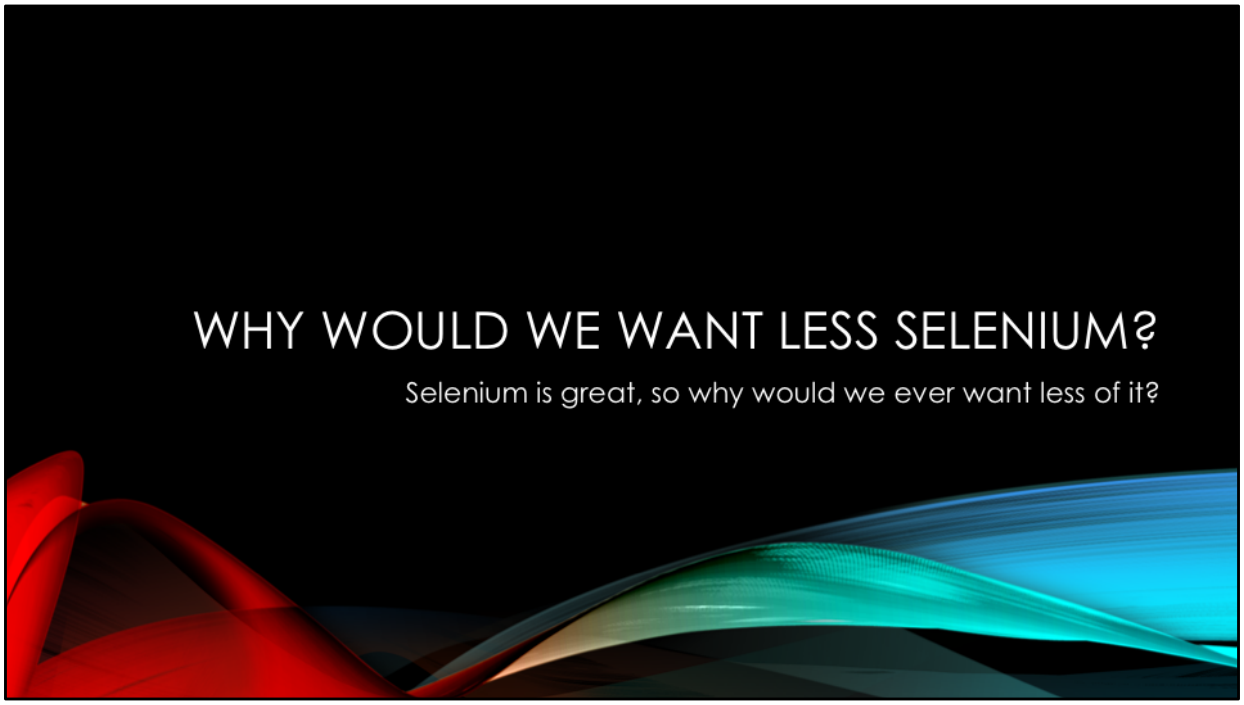
So let's take a closer look
at what these numbers represent

We write only HAPPY PATH automation
Avoiding NEGATIVE cases and EDGE cases



and our tests average between
about 10 and 40 seconds each

We filter test cases
according to a risk assessment matrix
and only automate tests
at the intersection of RELIABILITY
and NECESSITY
in our core workflows



Because we want to use Selenium BETTER! FASTER! STRONGER!

SO, let's talk
about some
principles
for Selenium usage

Efficiency
a test should be as efficient as possible

Independence
no test should depend upon another test

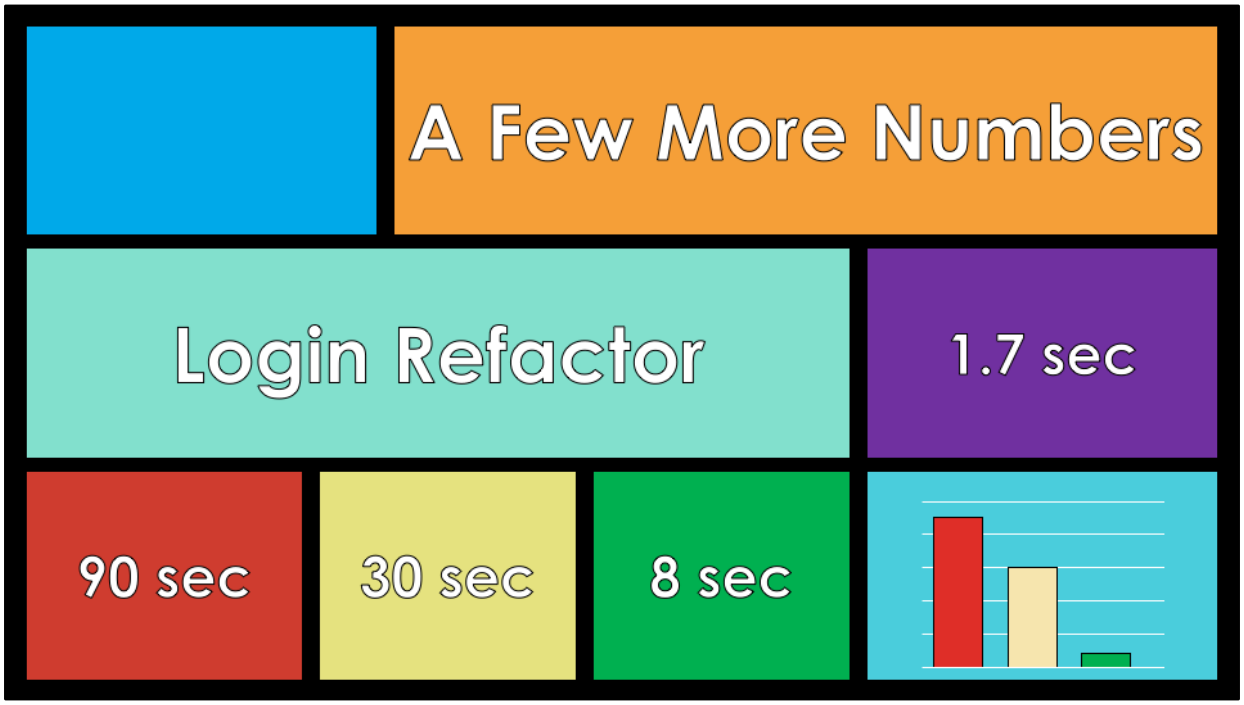
Isolation
a test should isolate the target system

We'll see an example workflow shortly

But FIRST

Let's look at what can happen

With just one refactor
to follow these principles



I recently refactored our login code
to use a backend process

for
INCREASED
PERFORMANCE

which saved at LEAST 1.7 seconds per test

assuming each test
only logged in to the server ONCE

as you'll see
some tests login and out
2 or 3 times
during setup,

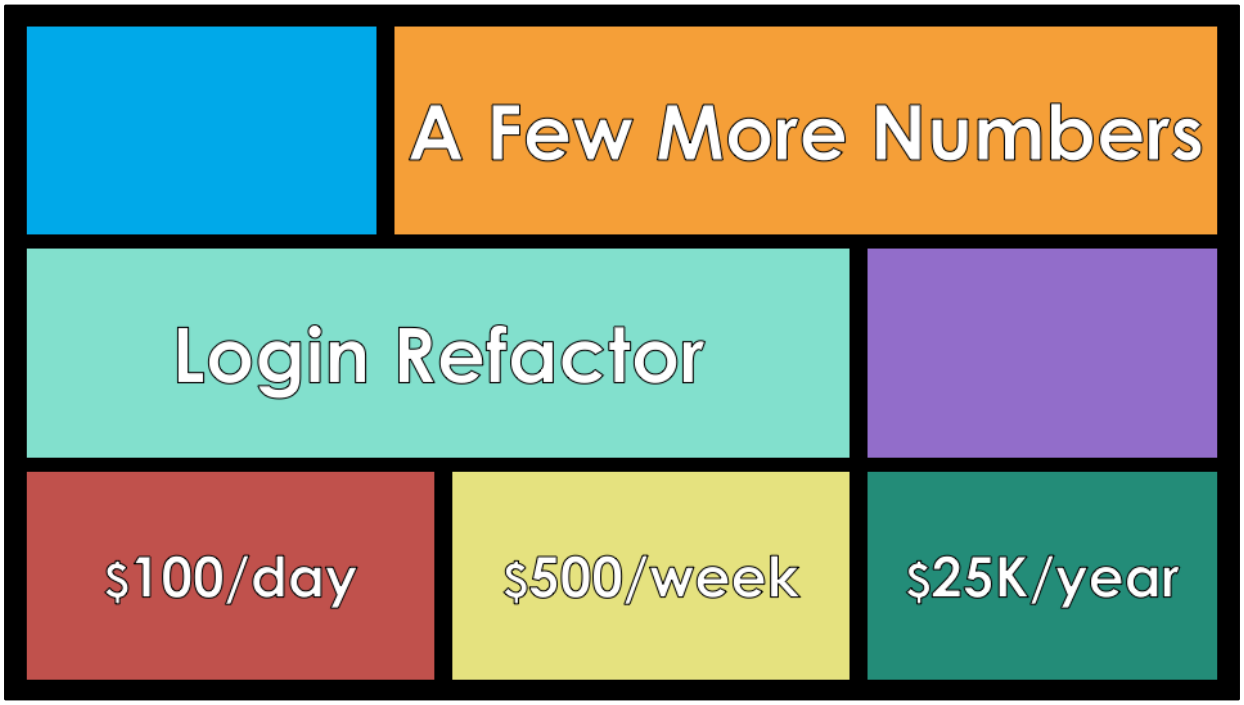
so the actual number
is probably

a little higher

The initial refactor saved us 60 seconds
of total execution time on our API test suite

refactoring again
to use the thread-safe
Apache HTTP Client library

allowed our team to bring total
execution time
down to 8 seconds
through PARALLEL EXECUTION



We CONSERVATIVELY ESTIMATE
that this increase in performance saved us

$100 A DAY

that adds up pretty quickly

$500 per week

If we assume 50 work weeks per year in the US

that's

$25,000 a year

We'll explore the new login code
in the examples near the end of this talk

but FIRST let's talk about

WHEN we know we might need a refactor

- Login
- Create Data
- Test
- Logout

Probably the most important thing about crafting

an Efficient, Independent, Isolated
SELENIUM test

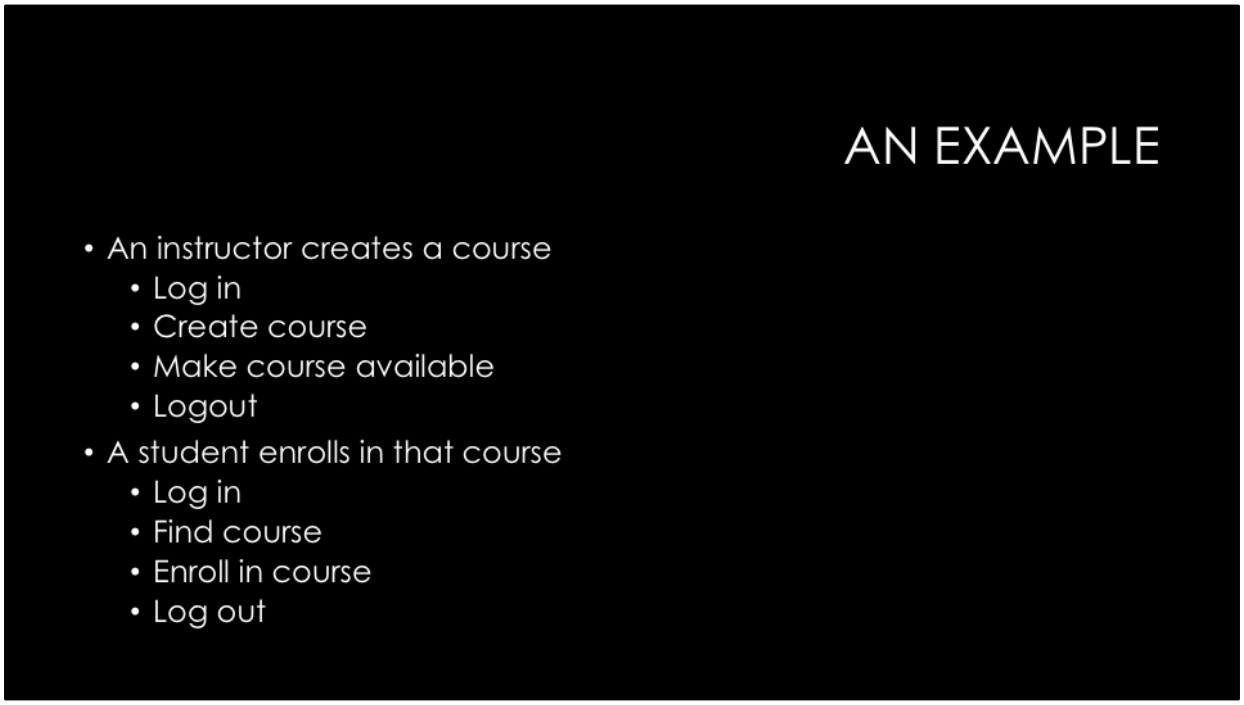
is to STAY ON TARGET

By focusing only on TARGET FUNCTIONALITY

And handling STATE and DATA CREATION

behind the scenes with BACKEND PROCESSES

We manage to keep our use of SELENIUM on target



Here's a typical workflow example in LEARN

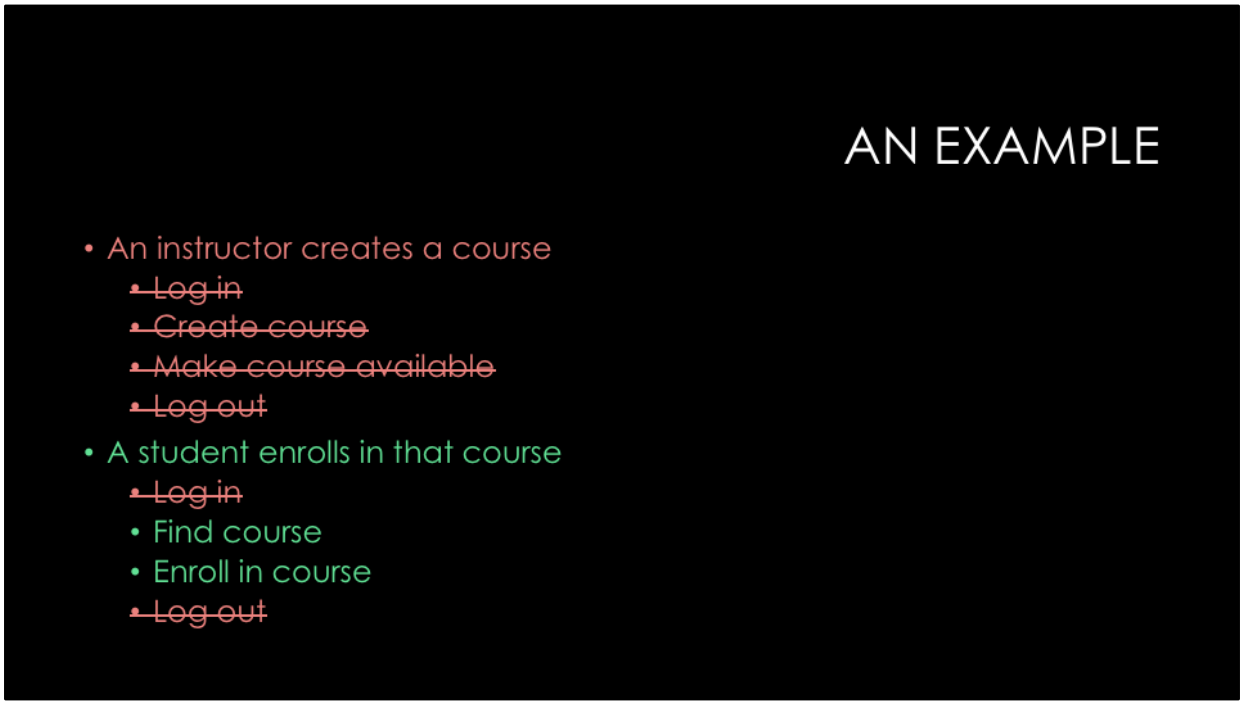
If we want to TEST that a student can ENROLL in a COURSE
These steps are necessary:

Log in
Create the course
Publish the course
Log out
Log in as a student
Find that course in a search
Enroll in the course
And ideally, log out

Where can we reduce Selenium interactions
to increase
efficiency, isolation, and independence?

Only two targeted functions
are absolutely CENTRAL

to the test case

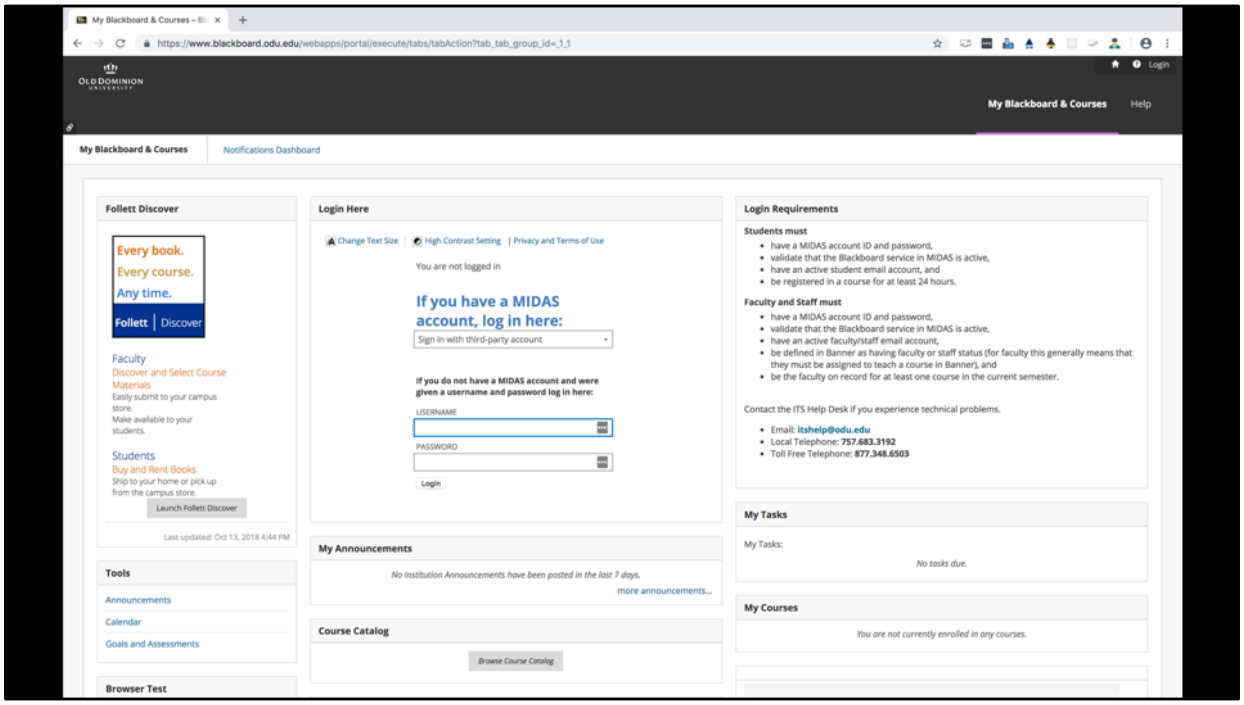


Finding the course
and enrolling in it

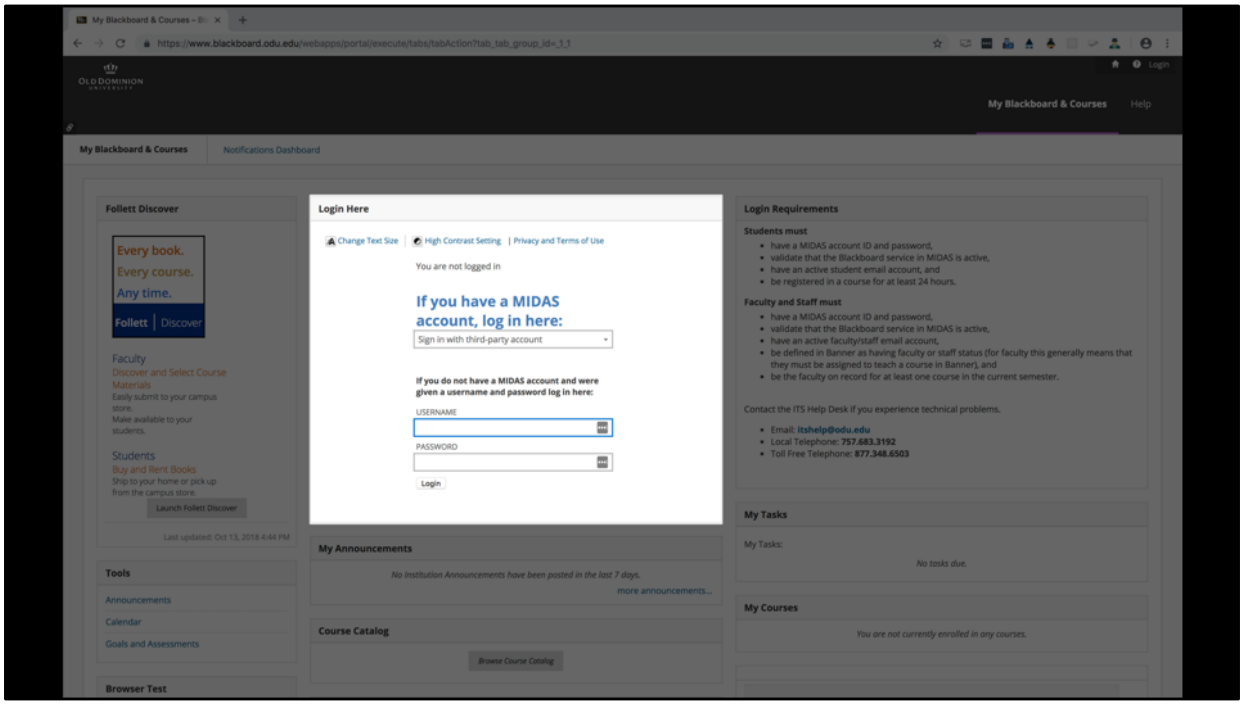
Refactoring to
STAY ON TARGET
means
replacing these workflows
with backend processes:

- Logging In
- Course Creation
- Course Publishing
- Logging out

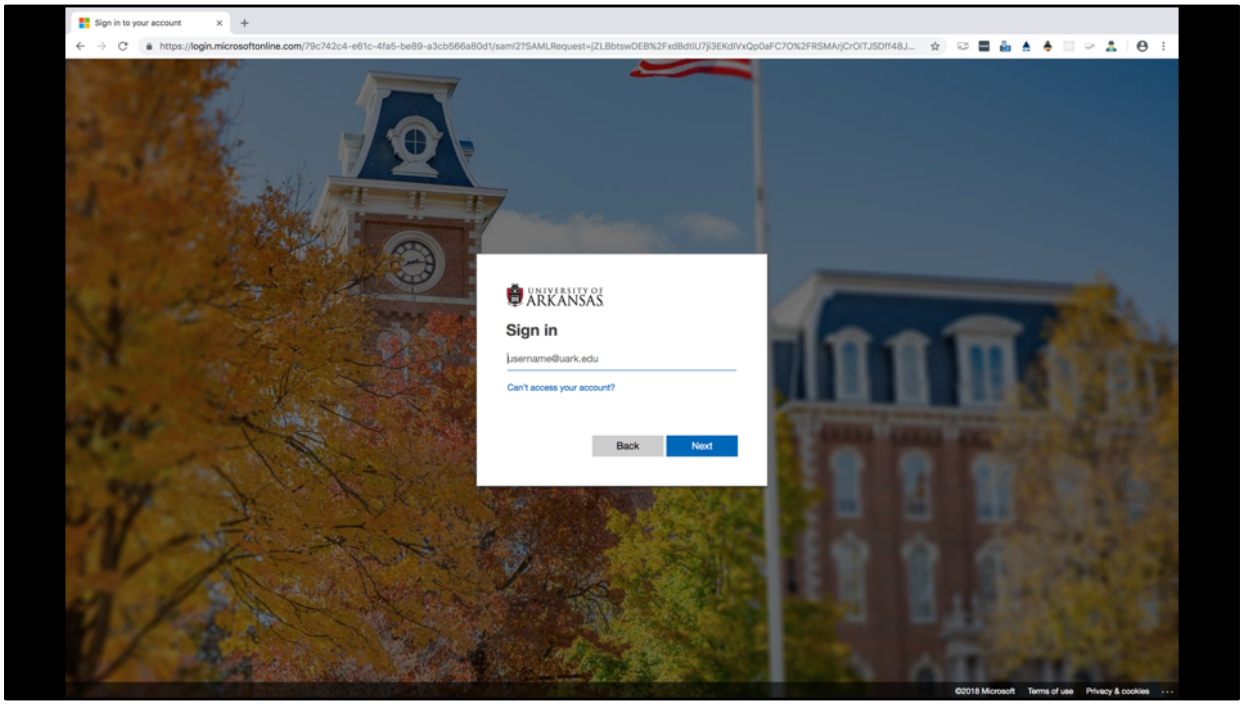
The LOGIN screen in BLACKBOARD LEARN
is extremely customizable



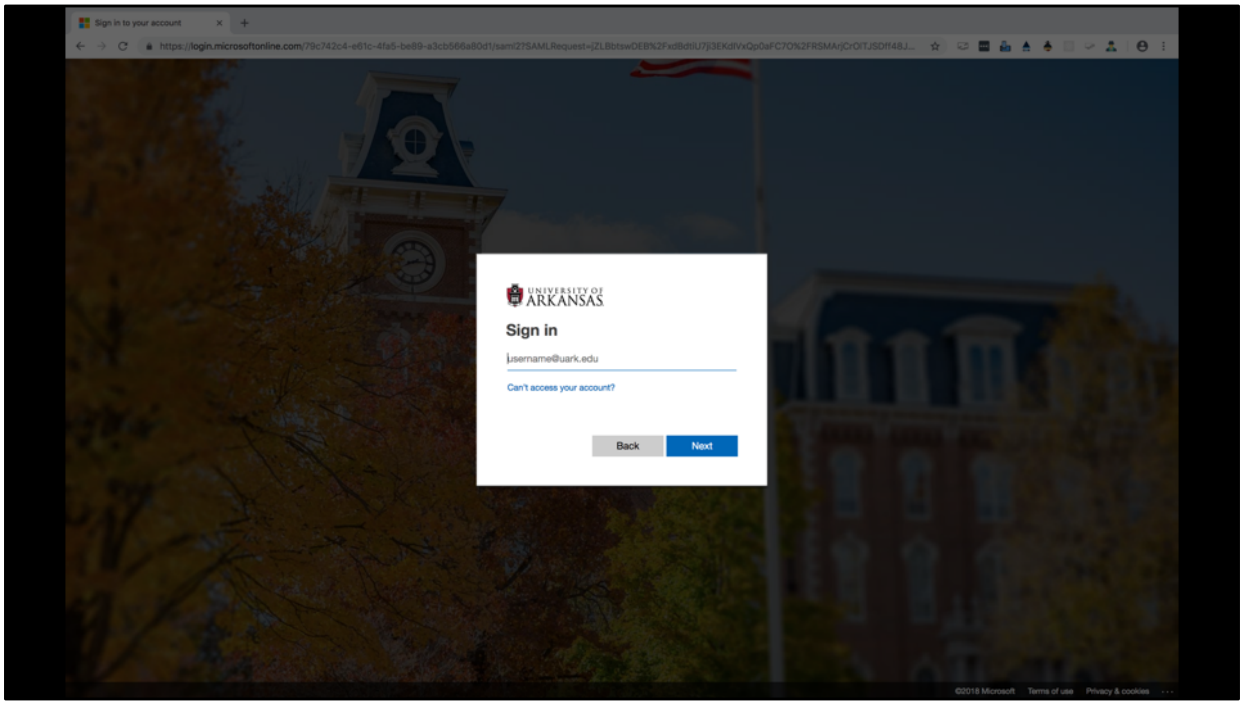
able to function as a TAB



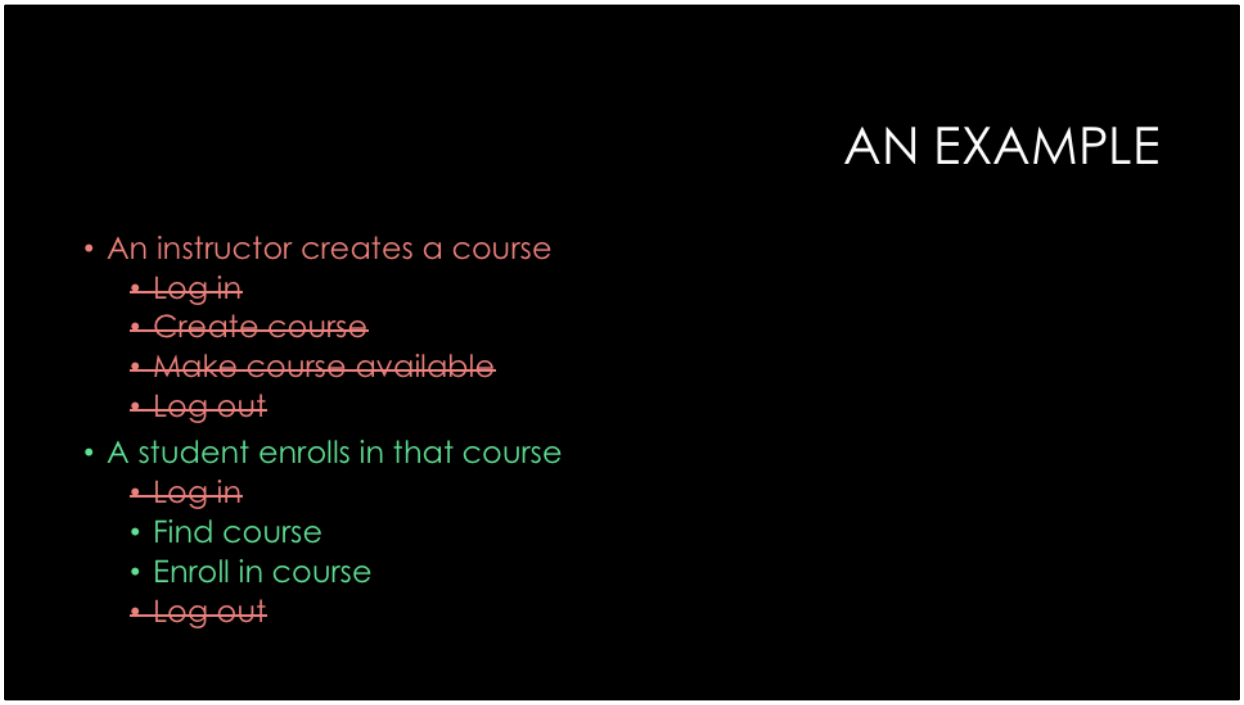
As seen in this example
From Old Dominion University



Or as a separate portal



As seen in this example
From the University of Arkansas
and capable of multiple
style customizations



all of which can be set
in the configuration options

As you can see,
some of these options
change both
look and
Behavior

So replacing that step alone
Creates more reliable tests
By eliminating downstream failures
From login screen UI customization

And as we have seen
Also boosts SPEED
And saves us a LOT of money

So with these refactors

We stick to our principles

Our test is more EFFICIENT
when Selenium is used for what it's best at

The test is INDEPENDENT
by being responsible for its own
STATE and DATA
which reduces FAILURES
from unrelated functions

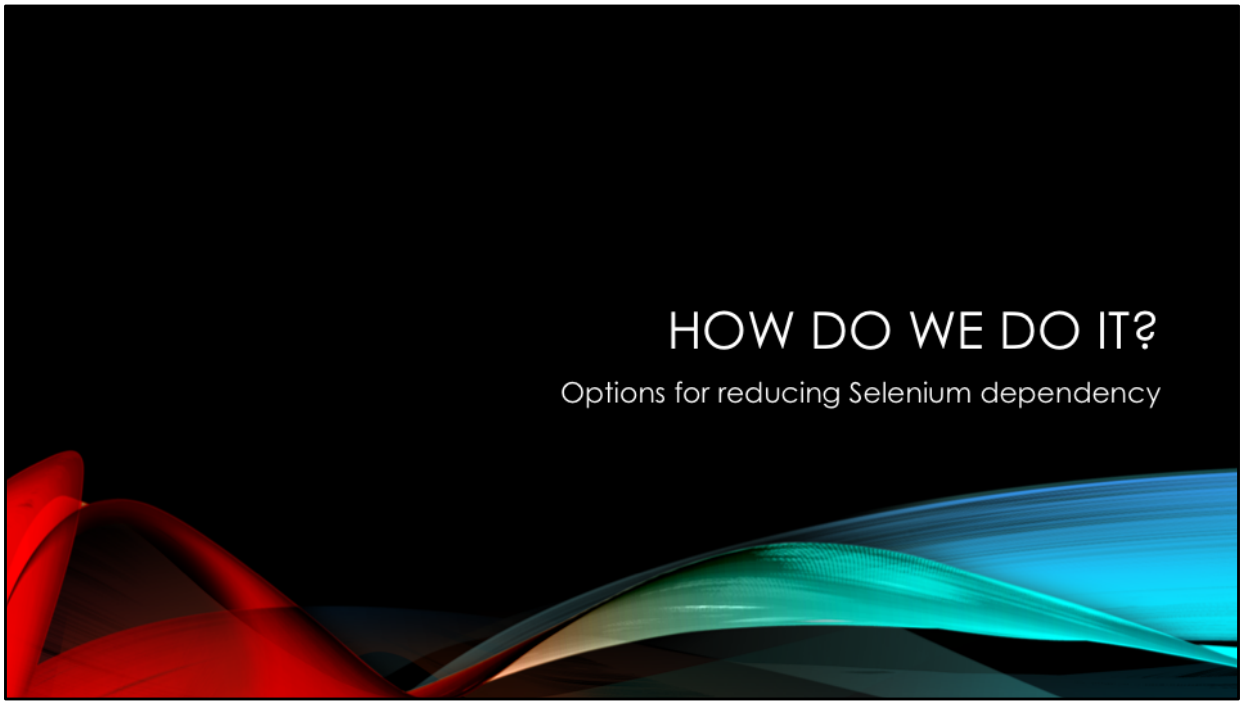
The test is fully ISOLATED
SELENIUM is used only for the
TARGET USER INTERACTION

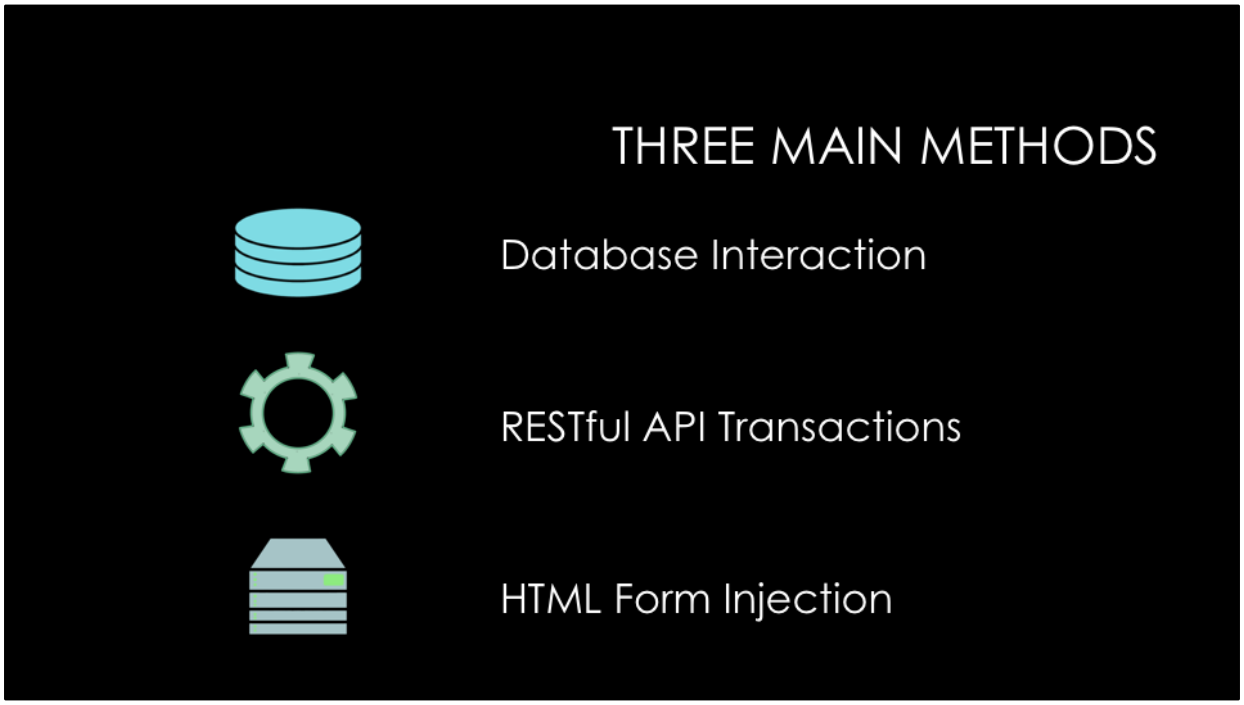
With all that being said,
we'll move on in the next section
to talk about HOW
we refactor setup steps in tests

RIGHT after we pay the CAT TAX

THIS PROCESS

IS THREADABLE


HOW DO WE DO IT?
Options for reducing Selenium dependency



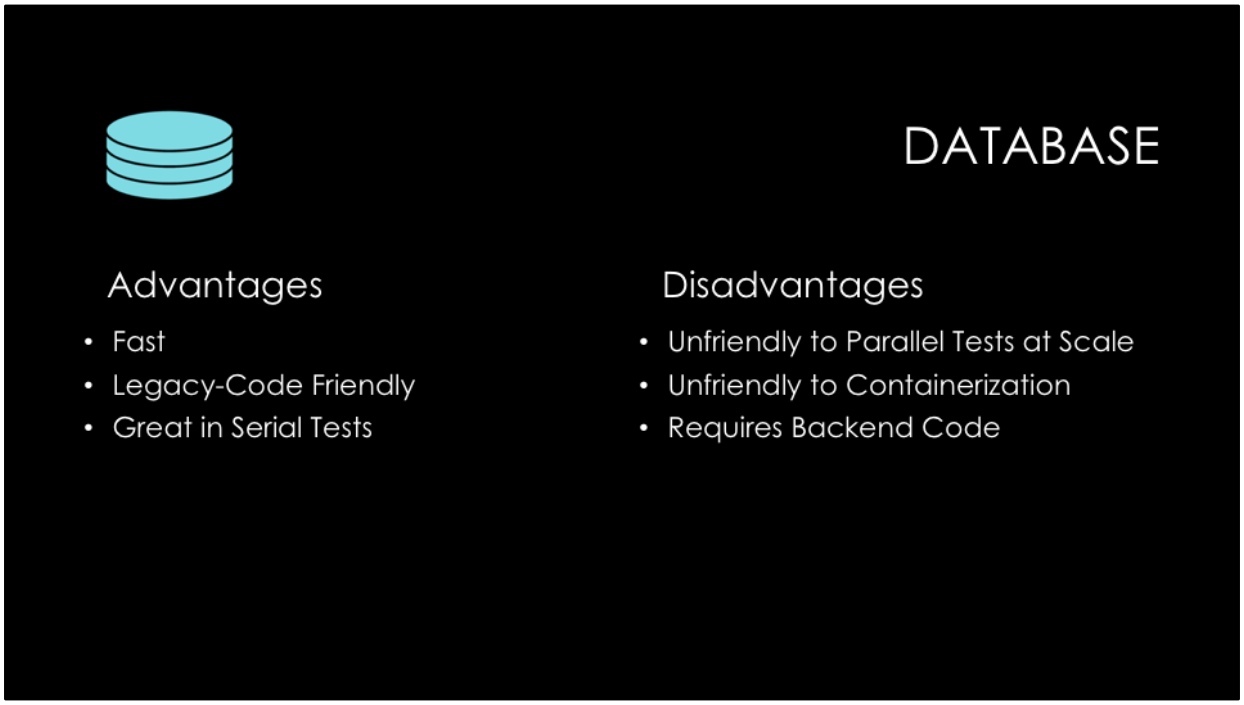
We have 3 main methods
available to us:

Database Interactions are performed
using our developers' own backend code

RESTful API transactions are performed
using JSON data

HTML form injections
are a HYBRID approach
between UI methods
and backend
HTTP transactions
using the same forms a user would

Let's talk about the ADVANTAGES
and DISADVANTAGES of each method



The advantages of using
DATABASE TRANSACTIONS
are that it
works very QUICKLY
and it should, since we rely on it daily in production use
The method is friendly
to LEGACY APPLICATIONS
and it works really well
in SERIAL TESTS

Now let's talk about the
DISADVANTAGES
If you run a lot of tests
in PARALLEL
You can quickly DOS
your database

If you run your tests
and test environment

in CONTAINERS
You must negotiate access
across containers

And it may
require knowledge
of BACKEND CO
or advanced SQL techniques
Which may present a
technical challenge

The advantages of interacting
with the REST API

Are that it is also very FAST

Works for Parallel Tests and Serial Tests

It works well across containers

And it is EXTREMELY EASY

You can capture a JSON
transaction in your browser
and recreate it in your code
with any JSON client library

The only real DISADVANTAGE
is that legacy applications
may not have endpoints

for every function that you need

Using HTML Forms in application testing
is a familiar approach
if you've done any
performance testing

with frameworks like
Gatling.IO or
Locust.IO
or the ever-popular
Apache jMeter

The ADVANTAGES are
that it's friendly to
Parallel Tests
and Containerization

It is moderately easy to learn
because it works about the same
way as submitting JSON

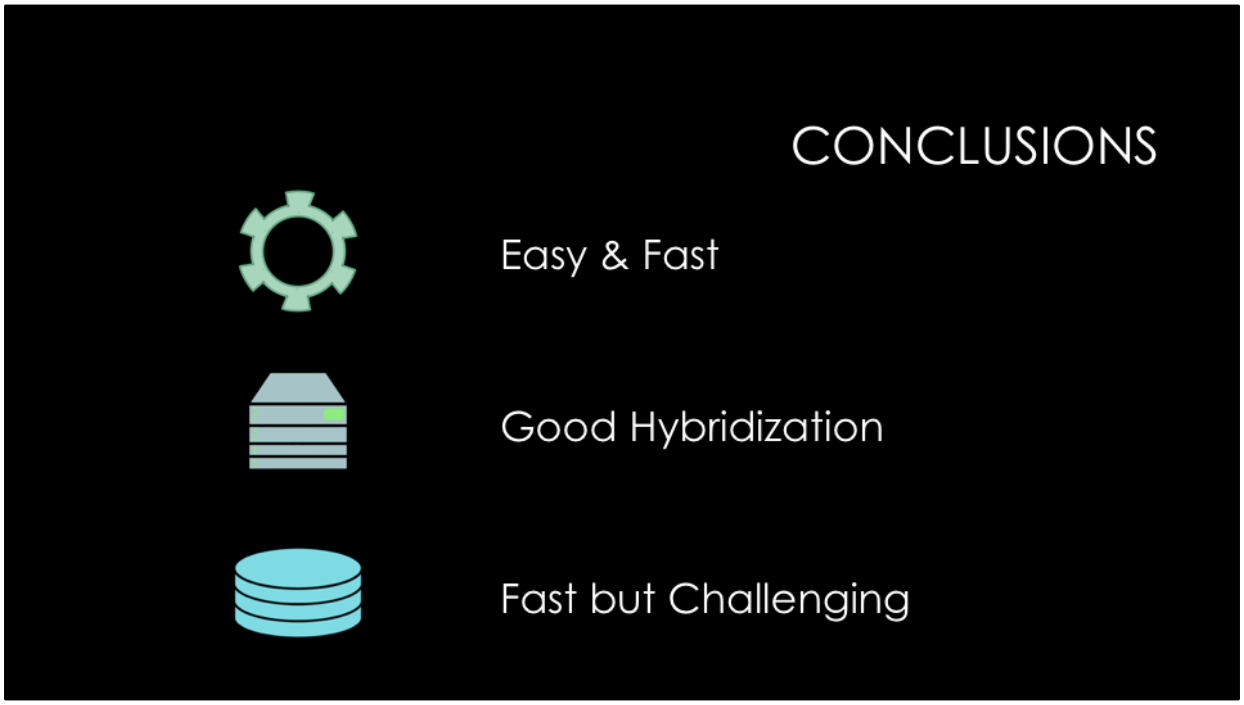
to a REST API

and it is extremely friendly
to LEGACY CODE

The DISADVANTAGES are primarily
that it may not always be
as fast as REST API
transactions depending
on the library you use
or interacting with the DATABASE

And you may encounter
some TECHNICAL LIMITATIONS

For example, if a page with a form
relies heavily on JavaScript execution
to get data into the right state or format
like with dates, or mathematical operations
or if some form controls
depend on a certain state
in other form controls



To recap and draw some conclusions

Using RESTful APIs is both EASY and FAST
they're container-friendly and parallel test friendly
and probably the all-around best approach

Using HTML FORMS is a good hybrid approach
for parallel tests or tests in containers
and for ease of use
that also supports legacy applications

DATABASE INTERACTIONS are fast
and LEGACY-FRIENDLY
but sometimes TOO FAST
and more CHALLENGING to implement
especially with CONTAINERS and PARALLEL TESTS

So I would recommend using REST APIs
wherever possible

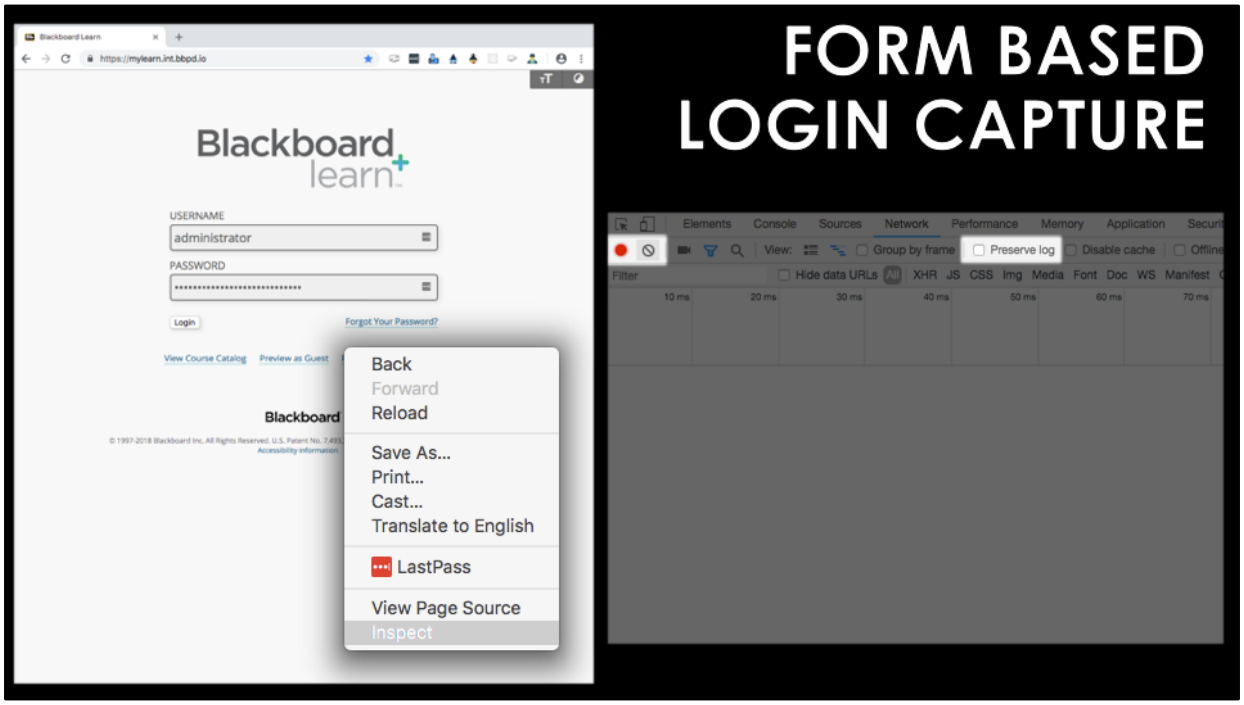
and HTML FORMS to supplement

We'll take a closer look at HTML FORM injection
right after paying the CAT TAX

THIS PROCESS

IS EMBEDDED


HTML FORM INJECTION
Let's look at some code examples!



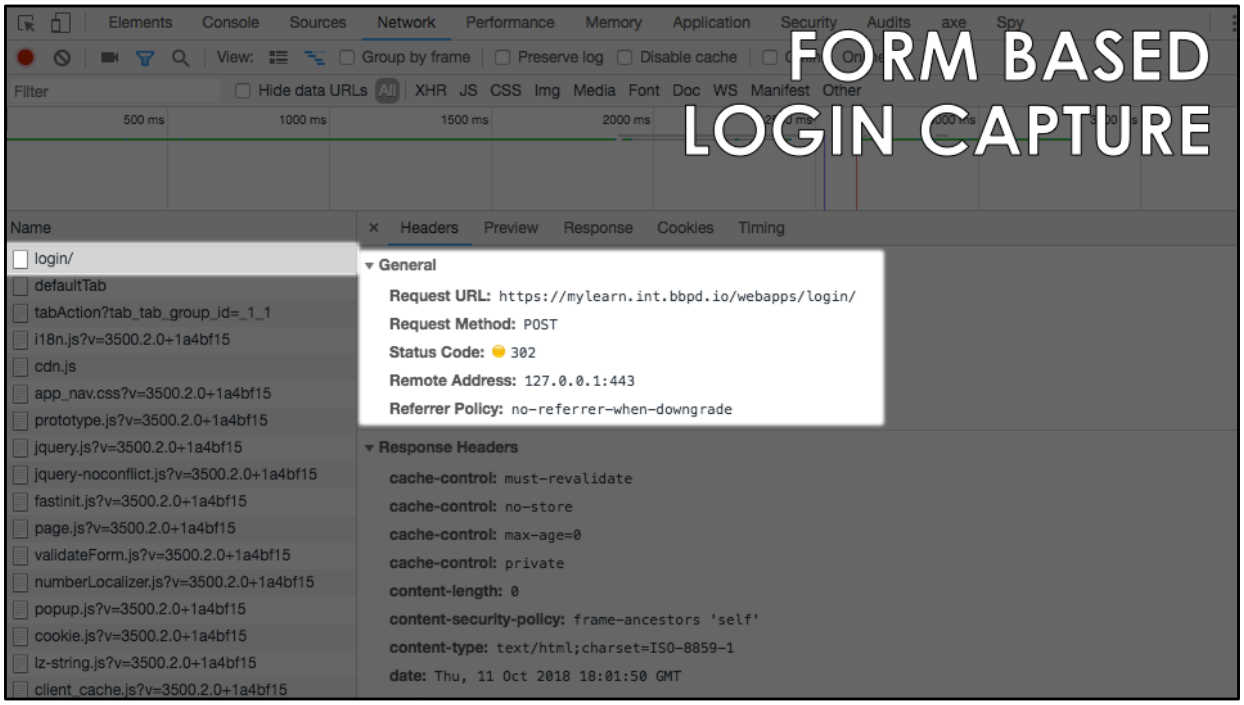
Let's look at
how to CAPTURE an HTML transaction

IN CHROME

Open INSPECT

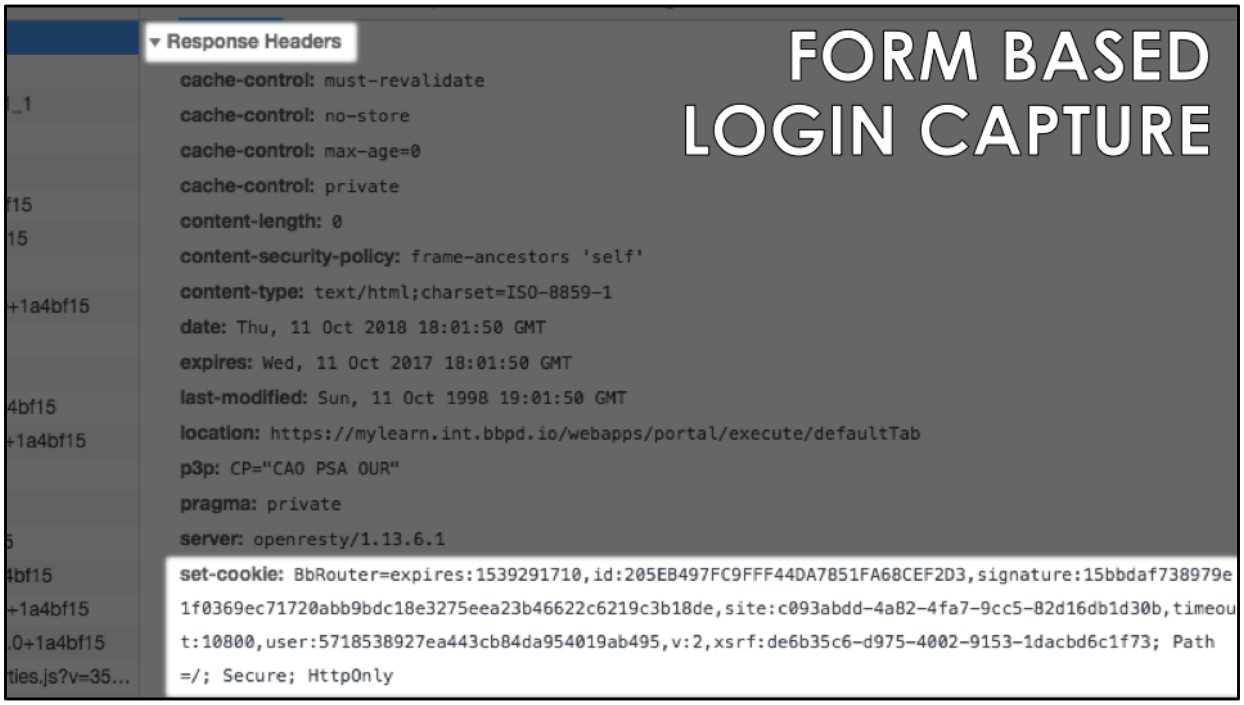
Select the
NETWORK tab

and put
a check in
"PRESERVE LOG"



Look for the POST transaction

Here is the HTML FORM DATA



In the RESPONSE HEADERS
look for
SET-COOKIE

And pass the cookies
to Selenium

FORM-BASED LOGIN EXAMPLE

```
1   public LoginPage login( User user )
2   {
3     CookieRequest cookieRequest = new CookieRequestFactory().getUserCookieRequest( user );
4     Map<String, String> userCookies = CookieClient.loginAndGetCookies( cookieRequest );
5
6     open( Configuration.baseUrl );
7
8     userCookies.forEach(
9                         ( name, value ) -> {
10                          Cookie cookie = new Cookie( name, value );
11                          WebDriverRunner.getWebDriver().manage().addCookie( cookie );
12                        } );
13
14    return this;
15  }
```

This is our LOGIN PAGE class

As you can see, there is a LOGIN method
which calls the LOGIN AND GET COOKIES method
and then iteratively PASSES COOKIES TO THE BROWSER

FORM-BASED LOGIN EXAMPLE

```
1   public static Map<String, String> loginAndGetCookies( CookieRequest cookieRequest )
2   {
3     try
4     {
5       return getCookies( cookieRequest );
6     }
7     catch ( Exception exception )
8     {
9       throw new RuntimeException( exception );
10    }
11  }
```

Not a lot going on, just making a request
to get login cookies from another method

FORM-BASED LOGIN EXAMPLE

```
1   private static Map<String, String> getCookies( CookieRequest cookieRequest ) throws
    Exception
2   {
3     Map<String, String> userCookies = new HashMap<>();
4     String baseUrl = cookieRequest.getBaseURI();
5     CookieStore cookieStore = new BasicCookieStore();
6     HttpClientBuilder httpClientBuilder =
    HttpClientBuilder.create().setDefaultCookieStore( cookieStore )
7           .setRedirectStrategy( new LaxRedirectStrategy() );
8     HttpClient httpClient = httpClientBuilder.build();
9
10    // Get base page & extract nonce ID
11    HttpGet initialRequest = new HttpGet( baseUrl );
12    httpClient.execute( initialRequest );
13
14    HttpPost secondaryRequest = new HttpPost( baseUrl );
15    HttpResponse secondaryResponse = httpClient.execute( secondaryRequest );
16    String secondaryResponseString = new BasicResponseHandler().handleResponse(
    secondaryResponse );
17    String nonceId = parseNonceId( secondaryResponseString );
```

2 main things going on here

Setup to retrieve cookies
to be returned

Hit the main page to get a Nonce ID

FORM-BASED LOGIN EXAMPLE

```
1    // Post username & password
2    HttpPost loginPost = new HttpPost( baseUrl );
3    List<NameValuePair> parameters = new ArrayList<>();
4    parameters.add( new BasicNameValuePair( "user_id",
cookieRequest.getUser().getUserName() ) );
5    parameters.add( new BasicNameValuePair( "password",
cookieRequest.getUser().getPassword() ) );
6    parameters.add( new BasicNameValuePair(
"blackboard.platform.security.NonceUtil.nonce", nonceId ) );
7    loginPost.setEntity( new UrlEncodedFormEntity( parameters ) );
8    HttpResponse loginResponse = httpClient.execute( loginPost );
```

Here we create a form by adding parameters to an Array List,
then submit it to the login endpoint with an HTTP POST.

This emulates a user login,
but without executing code in a browser.

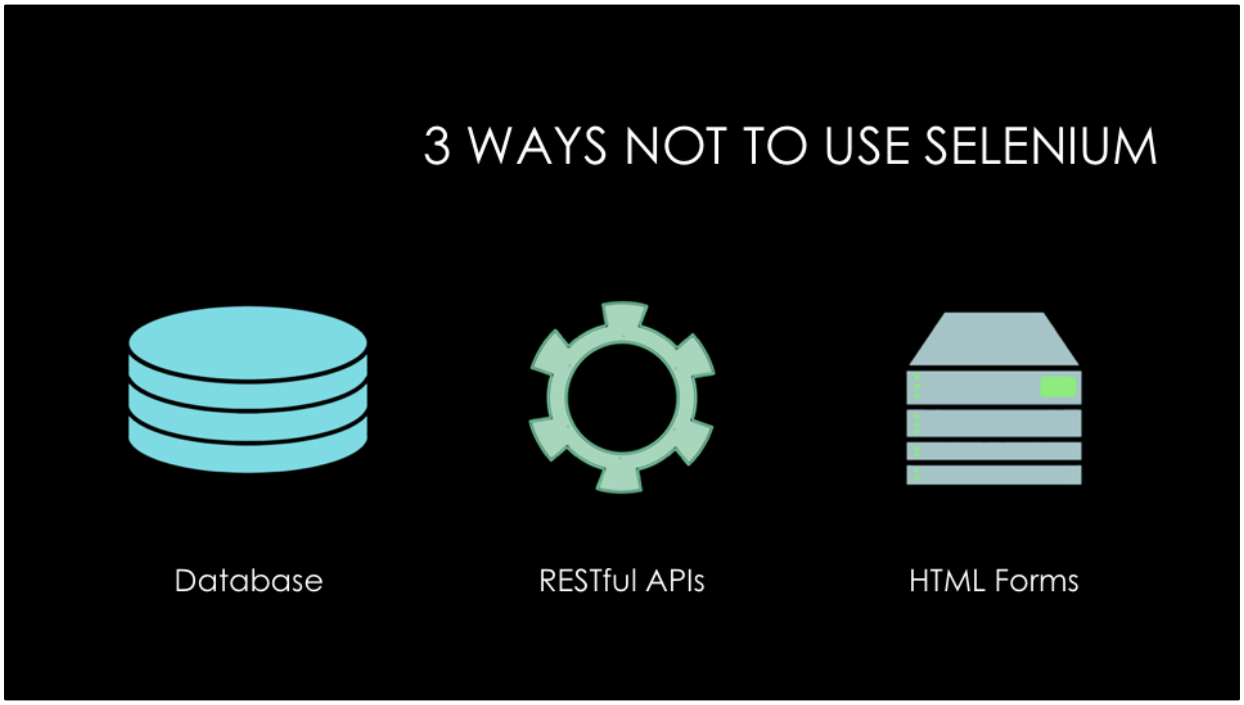
FORM-BASED LOGIN EXAMPLE

```
1     // Throw an exception if we get an abnormal status code
2     int loginStatusCode = loginResponse.getStatusLine().getStatusCode();
3     if ( loginStatusCode != 302 && loginStatusCode != 200 )
4     {
5       String loginResponseHtml = new BasicResponseHandler().handleResponse( loginResponse
);
6       throw new CookieGatheringException( cookieRequest, loginStatusCode,
loginResponseHtml );
7     }
8
9     // Extract cookies from cookie store
10    List<Cookie> cookies = cookieStore.getCookies();
11    for ( Cookie cookie : cookies )
12    {
13      userCookies.put( cookie.getName(), cookie.getValue() );
14    }
15
16    return userCookies;
17  }
```

Here we have some final steps:

throw an error if we receive a failing status code on the response

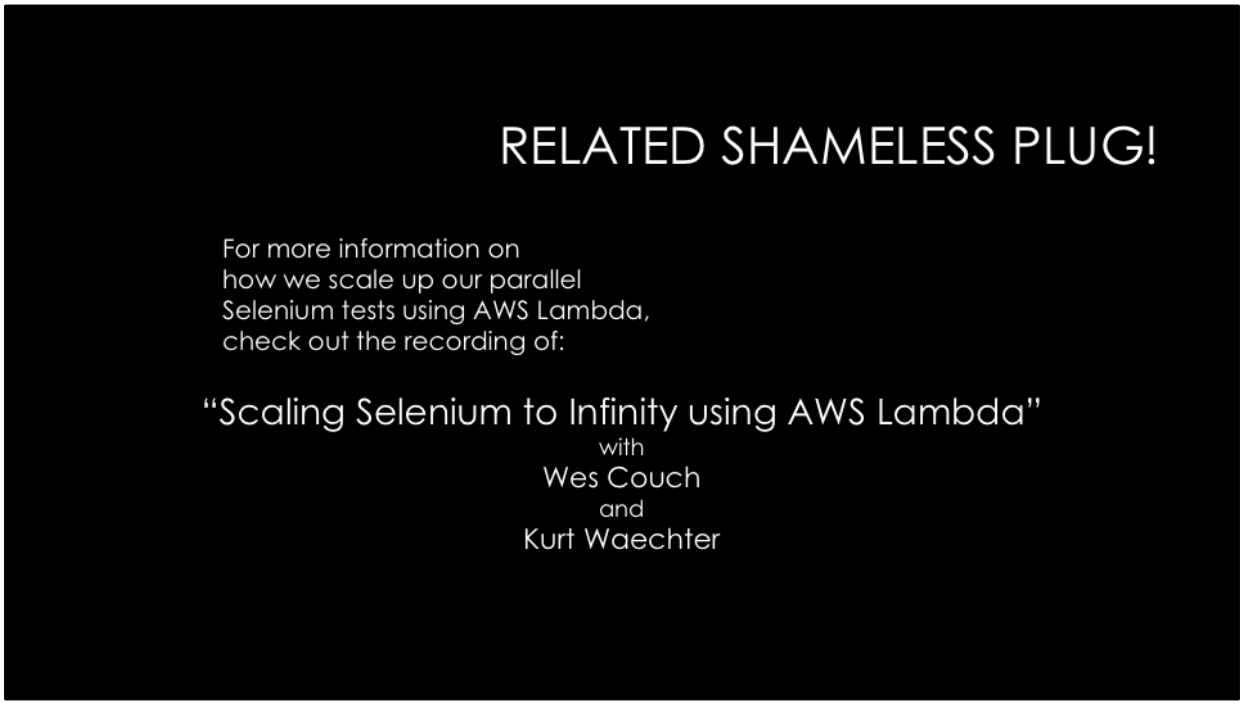
extract cookies from the Cookie Store
and return them as a List



DATABASE Interaction

RESTful API Transactions

and HTML Form Injections



For more information about how we scale up
our test automation software with massive parallelization

CREDITS

- Stay On Target
  - Frasca Designs
  - https://www.teepublic.com/user/frascadesigns
- Gears
  - Brad Wilmot
  - https://www.flickr.com/photos/baldbrad/
- Alone
  - Nikki McLeod
  - https://www.flickr.com/photos/mommypants/
- IOU/USA 05
  - Chris Murphy
  - https://www.flickr.com/photos/chrism70/

A very special thanks to Ashley Hunsberger, Kevin Wilson, Joe Nolan, and Pj Smith for their support!

SOME REFERENCES

- JSON Lint
  - An online JSON formatter
  - https://jsonlint.com
- JSON Editor
  - A visual JSON editor
  - http://jsoneditoronline.org
- Apache HTTP Client
  - Form Post Example
    - https://memorynotfound.com/apache-httpclient-html-form-post-example/
- This Slideshow
  - Github
    - https://goo.gl/hTGQiP