**TRANSKRYPT NAGRAŃ**

(przed każdym fragmentem tekstu jest wklejony slajd/printscreen, którego dotyczy)

Spis treści

[BEGINNING 1](#_Toc495887470)

[PRESENTATION 2](#_Toc495887471)

[USE CASE INTRODUCTION 6](#_Toc495887472)

[REMINDER ABOUT HTML 7](#_Toc495887473)

[D3 KEY CODE: UDPATE- ENTER-EXIT SCHEME 9](#_Toc495887474)

[UPDATE PHASE 9](#_Toc495887475)

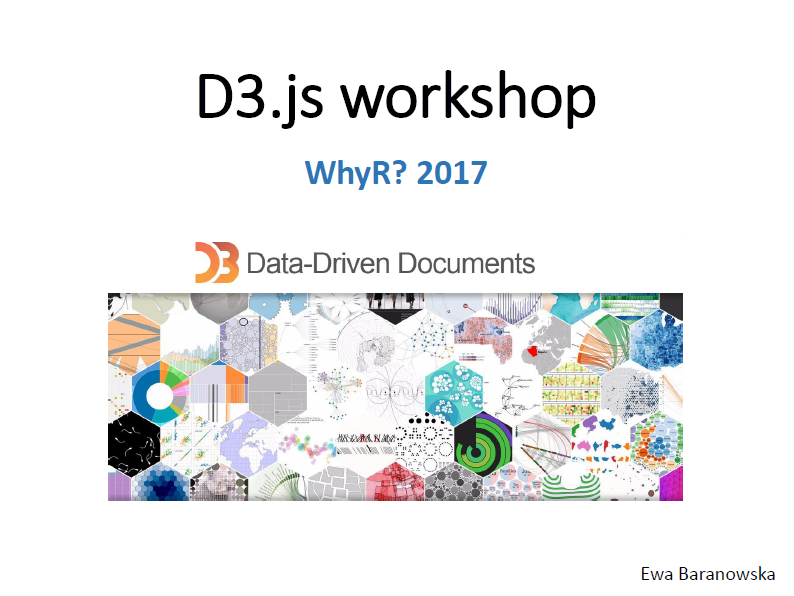
[ENTER PHASE 12](#_Toc495887476)

[EXIT PHASE 14](#_Toc495887477)

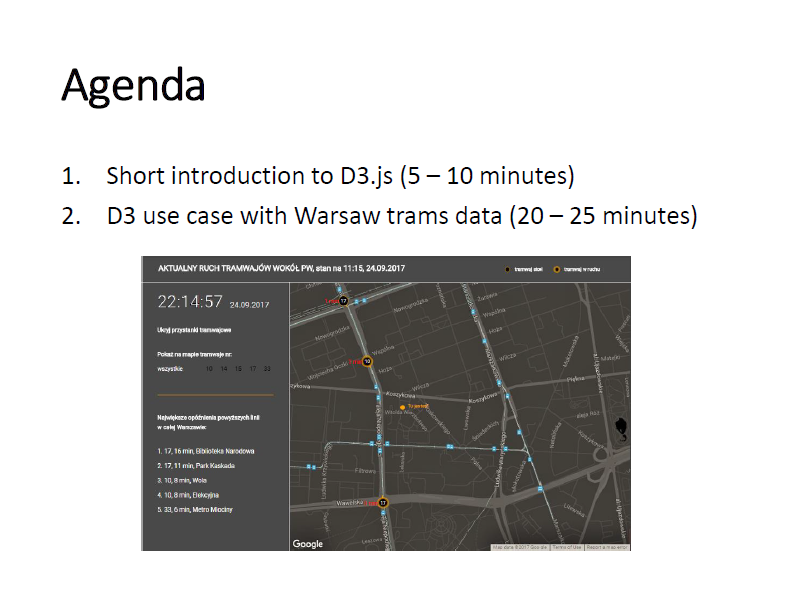
[SUMMING UP – HTML +D3 15](#_Toc495887478)

[~~TRANSITIONS + THE END~~ 15](#_Toc495887479)

# BEGINNING

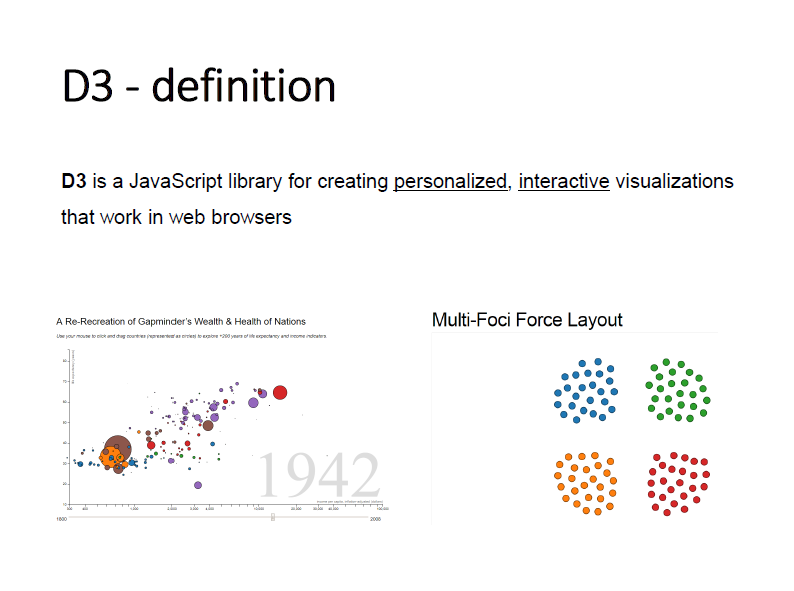


Hi, my name is Ewa Baranowska and I have great pleasure to carry out a workshop about data visualization using D3 library. This workshop will last about thirty minutes which is not enough time to learn how exactly D3 library works, but should be enough to get intuition of D3 main idea and some mechanisms that make this library so popular to use in visualization.



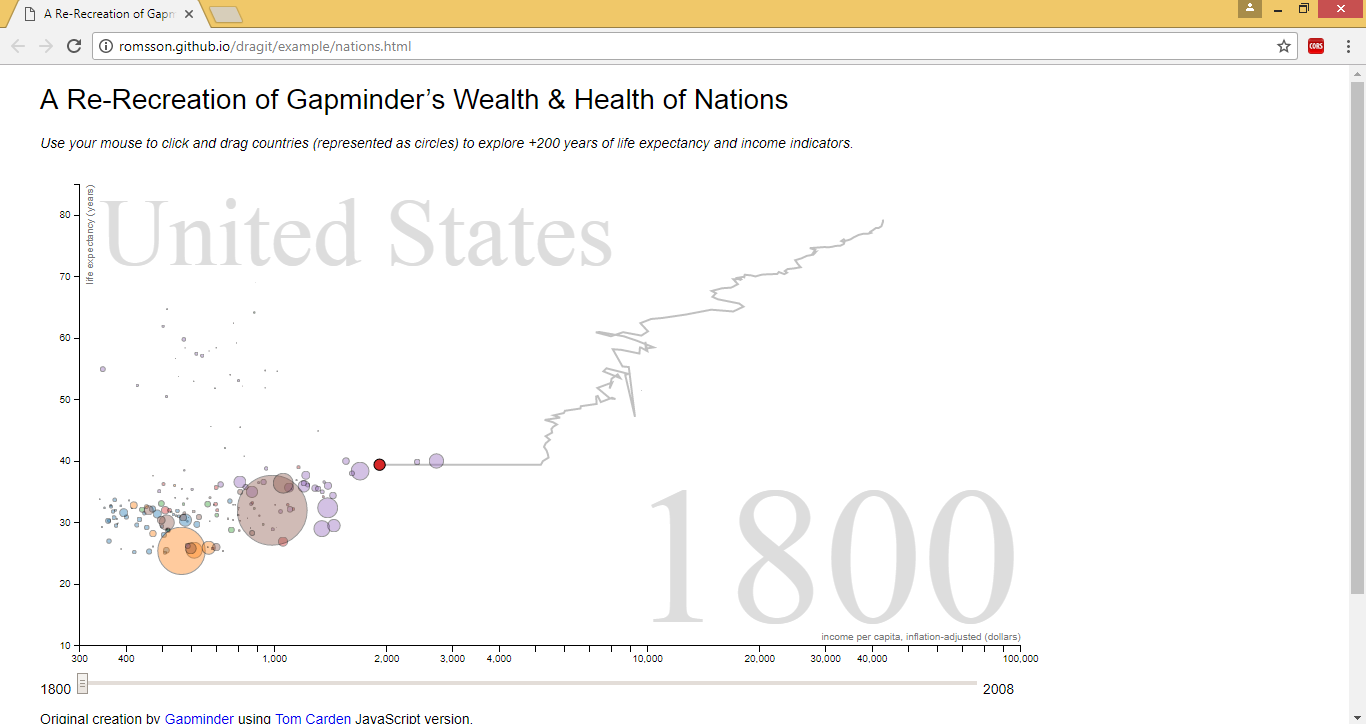
Agenda of this workshop is as follows: firstly I will introduce you quickly to D3, what it is exactly, how it works, what you should know about it before using it, it will take us about 5 up to 10 minutes, and then I will explain you the core mechanism of D3 code on the example of D3 visualization that uses data about Warsaw tram traffic. It will take us about 20 -25 minutes.

# PRESENTATION

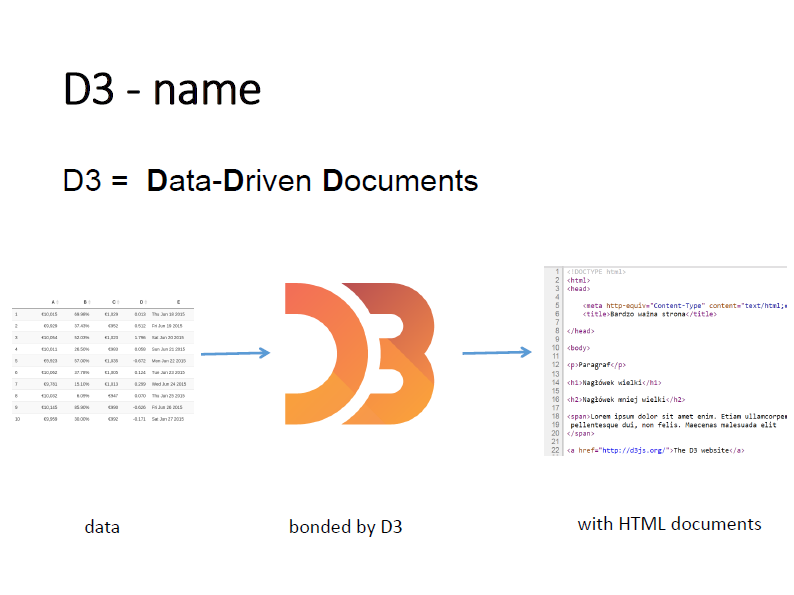


So let’s start with the definition of D3, that in my opinion sums up all the most important characteristics of D3. So D3 is a JavaScript library, therefore the main programming language that is behind D3 is JavaScript, for creating personalized, interactive visualizations that work in web browsers. I will explain later why I underline this word **personalized**, now let’s go to interactiveness of D3. To show you what kind of interactiveness D3 offers look at this example.

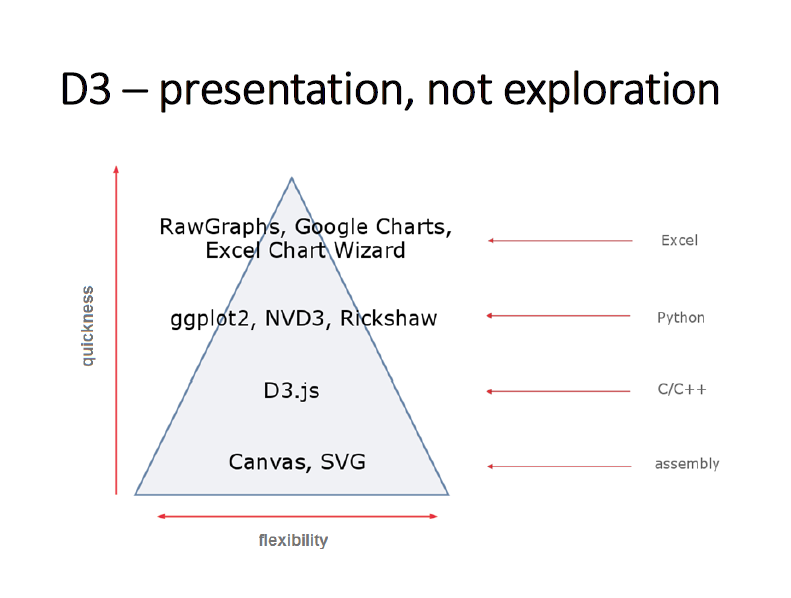
**(kliknąć na lewy wykres, i pokazać animacje na filmie)**



This scatterplot made in D3 uses smooth animations, lines and names of country change automatically, we have also here some mouse events, like tracking the cursor and emerging of the tooltips after looking at specific circle. We can click that circle, getting its stroke bolder, we can even drag it to see change of its position over the years. There are many possibilities of interactiveness in D3, these are the most popular/basic ones, and what more important the whole visualization works in web browser. In reality creating visualization in D3 will be creating html document with some JavaScript code.



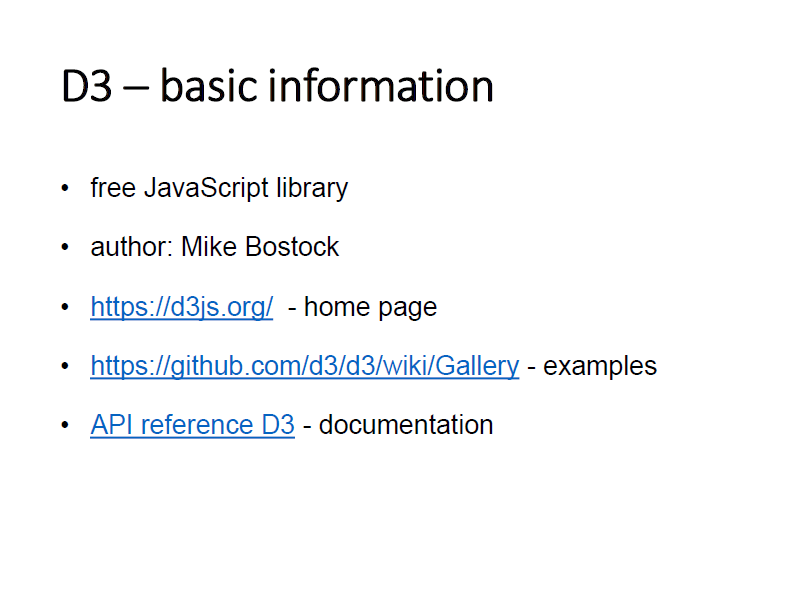
Name of this library, D3 is not random, D3 is the shortcut for Data Driven Documents and it is strictly linked to its main idea. In D3 we will take our data that we want to visualize and bond them with the aid of D3 with our HTML document, precisely each observation will be used to generate specific element of HTML document, like circle, text paragraph or line. How we can achieve this by D3 I will explain later, now it’s important that writing few lines in D3 let us generate automatically elements of HTML, and therefore our visualization, so we don’t have to do it manually.



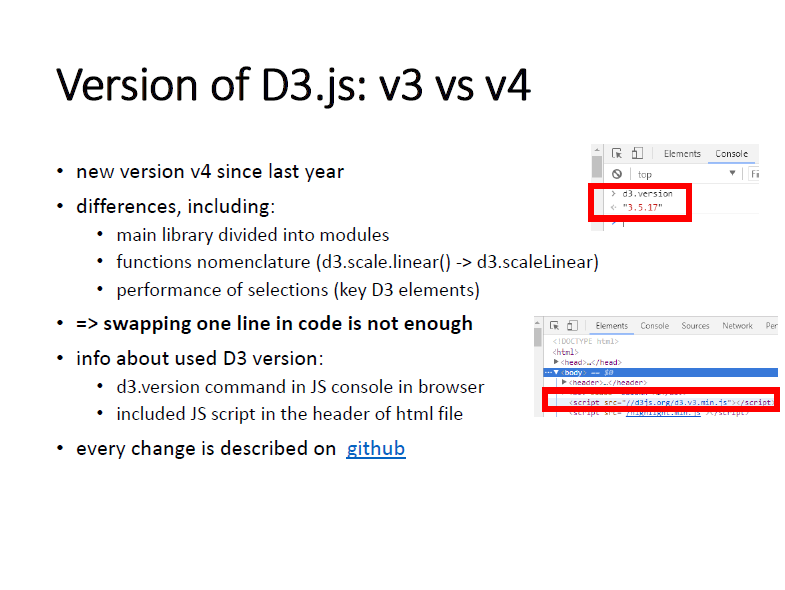
Earlier in the definition of D3 I have underlined word “personalized”. Why is it so? If we want to compare D3 library with another tools for data visualization we can do it by considering their two features: flexibility and quickness. And D3 is situated here in this pyramid (wskazanie na slajdzie kursorem) – it’s very flexible tool, but it comes with a price – you have to put some time to create visualization. So if you want to create a quick chart just to peek at the data – D3 is not the best choice, too much time will be sacrified for such a simple task. But, if your aim is to create more complicated and personalized visualization that will present your data, your point of view on the data – D3 is a great solution that let you do this without limiting you to just a few predefined templates, like Excel does.



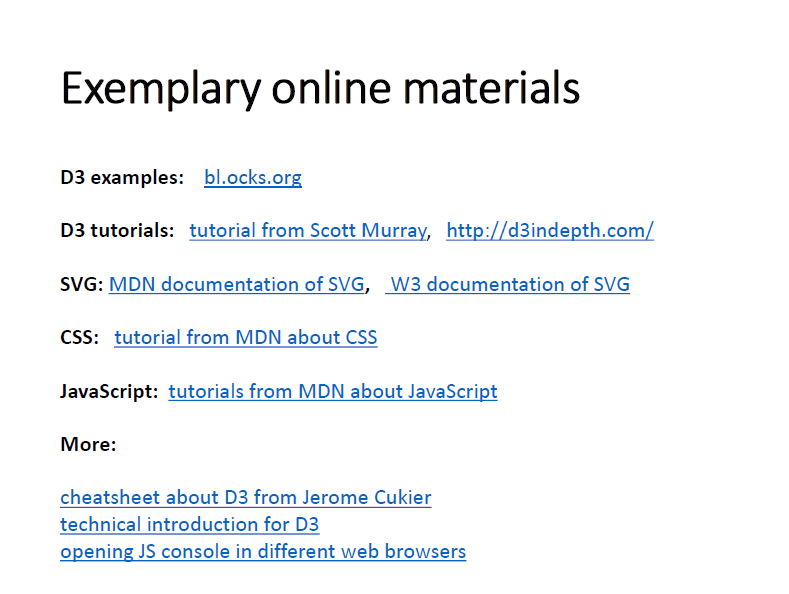
Next thing that should be mentioned about D3 is its technical background. D3 is said to have very steep learning curve, that is, you have to learn a lot in the beginning to freely work with D3. And the main reason is not the exact D3 code/syntax that is reasonably approachable, but the fact that it uses beneath a few other technologies that should be known on at least basic level. What are these technologies? It’s of course HTML, cause as I have said earlier, we will create HTML document elements, it’s SVG, which responds for graphic elements on html documents, it’s CSS, that let us style each HTML and SVG element so it will be even more personalized and of course JavaScript which is the core language for D3. Knowing these web related technologies gives you a head start in working with D3, cause as you remember D3 itself create web-based visualizations.



Couple of some other basic information about D3. It’s a free tool, you don’t have to buy anything, all you need is some text editor and web browser. The author of D3 library is Mike Bostock, you can associate him with New York Times Magazine, where he have created lots of beautiful data visualizations few years ago. Home page is d3js.org. You can find source codes as well as lots of examples and specific documentation for D3 on its github repository, on the slide you can find particular links to it.

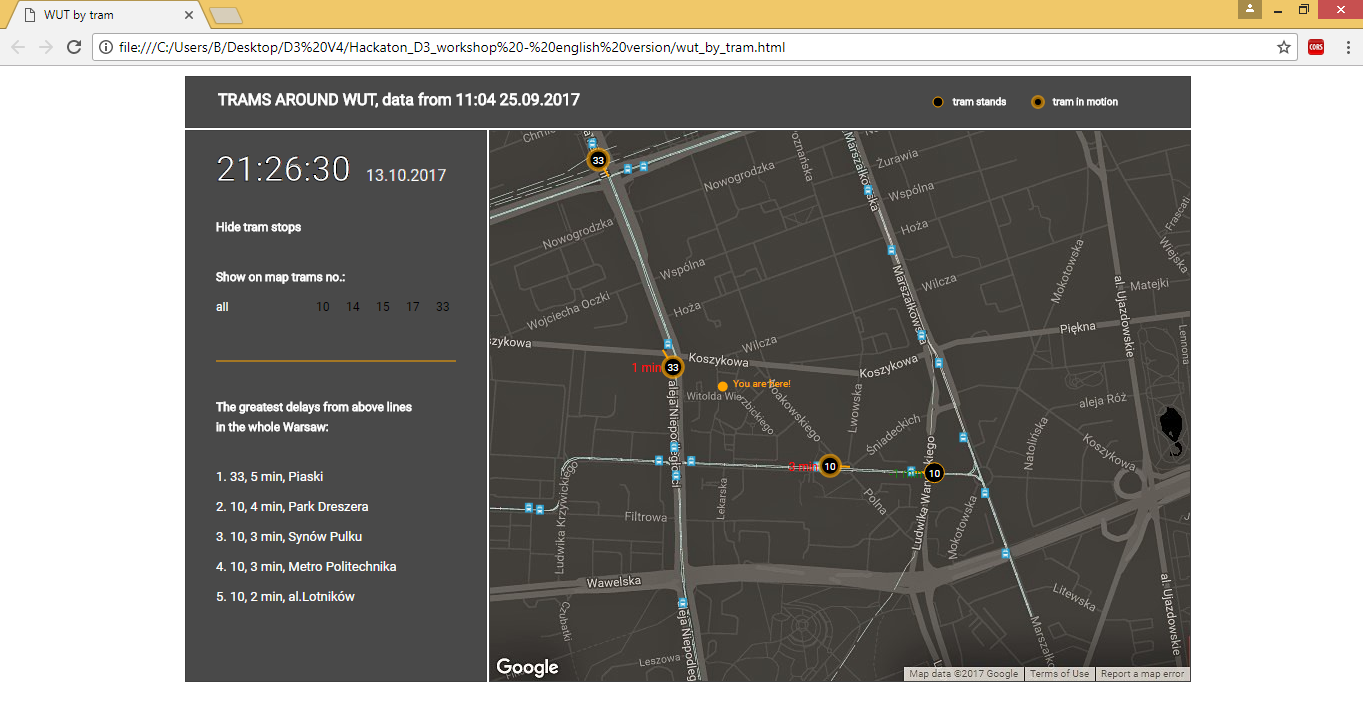


And last technical remark is that since last year we have new version of D3, version 4, and I mention about it, because there’s some differences between old version, 3, and the newest one, that include nomenclature (names of functions) and even performance of some key D3 code elements, so if you will work with some examples from web, you should check which version of D3 it uses, cause simple swapping version of loaded library won’t be enough to make the example work.



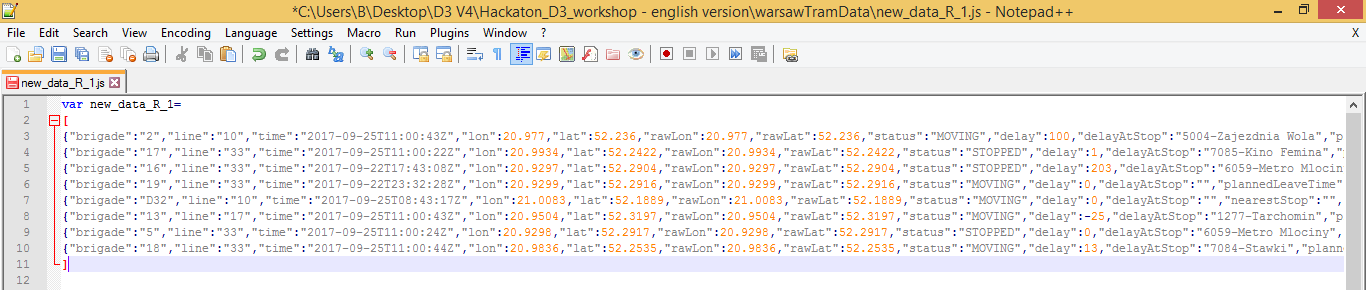
And on the last slide from this theoretical introduction I give you some exemplary links to extra materials if you want to broaden your knowledge about D3 or related technology by yourself. The first link is to the side where you can find a lot of D3 examples, including some that uses code from the newest versions 4 of D3 library.

# USE CASE INTRODUCTION



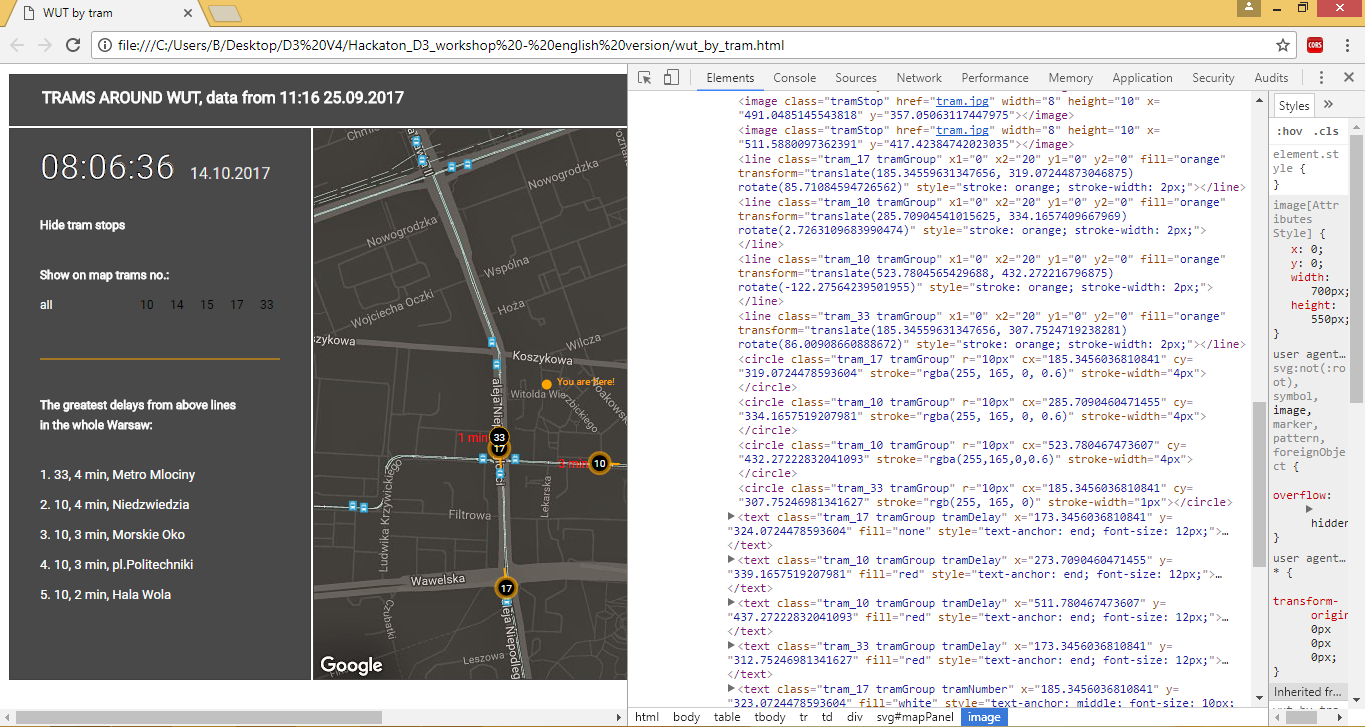
**(pokazać wizualizację ze zmianami, nie tylko printscreen?)**

Ok, visualization that will serve us as an example of D3 code looks like this. It presents how in simplified way tram traffic near Warsaw University of Technology, looked for about 10 minutes on 25th of September. Visualization is divided in two parts, on the right, we have main panel with map and on the left, panel with extra features regarding the map. On the map we visualize position and delay of each tram that was at that time in this neighbourhood. We mark each tram by circle with an arrow and two labels, one for line number of tram and one, in red or green color, for info about its delay. These properties, delay, line number and position of each tram are not manually set, but automatically, automatically visualized, powered by data that looks like this.



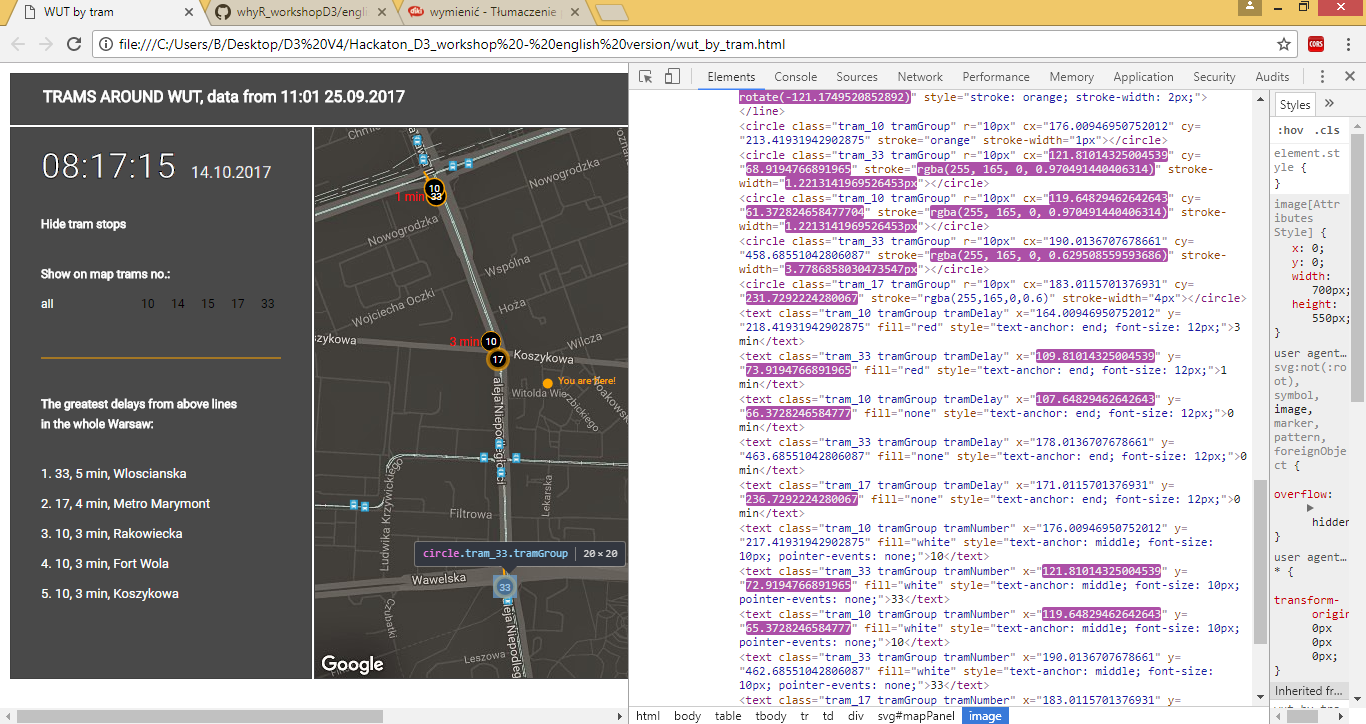
These datasets that feed the visualization are provided by Warsaw city. In these datasets each observation represents one tram and therefore serves as a source for D3 about particular tram info. So, as you have probably already seen, our trams are moving, some enter the map, some exit, some just move a little bit. Each such change of tram traffic situation is linked with a change of underlying dataset, and D3 is responsible for appropriate bonding new data with our visualization, and more precisely with our HTML document that keeps this visualization. So we see here the main idea of Data Driven Documents, D3, we have data and we bond them dynamically, automatically with HTML document elements. How does it work exactly? Now we will see.

# REMINDER ABOUT HTML



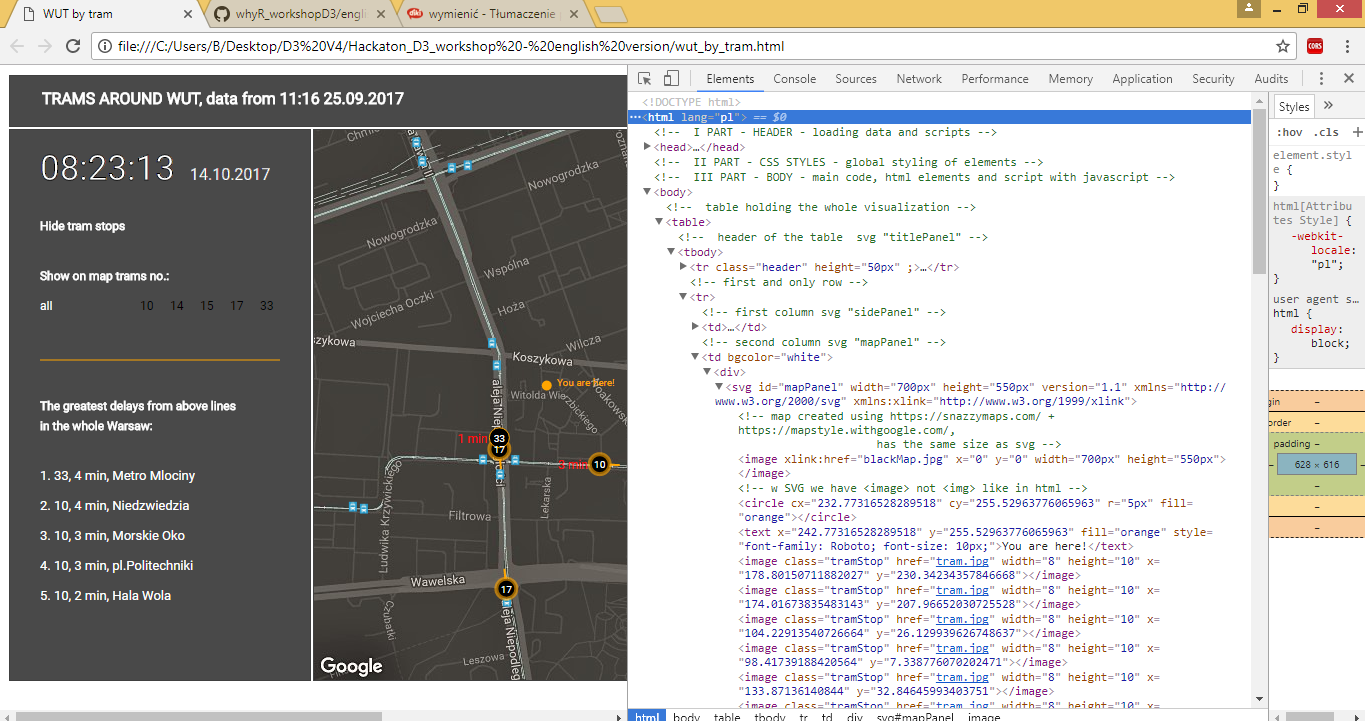
**(pokazać kursorem na tag circle)**

But, before getting our hands on D3 code, I want to remind you quickly about structure of HTML document. So let’s look at source code of our visualization HTML document. Two main things to know. First, element in HTML and SVG is represented by so called tag, here we have for example tag **circle**, that represents circle in the visualization. Each tag has some general properties (attributes), like class or id and some specific properties, all of them are enumerated after tag name, so here we have class of this circle, we have attribute called “r” that is responsible for circle radius, we have attribute “cx” that represents horizontal position of center of the circle etc. etc. These attributes, can be set manually, or thanks to D3, automatically, based on data.



**(odświeżyć stronę i pokazać jak zmieniają się te atrybuty w przeglądarce)**

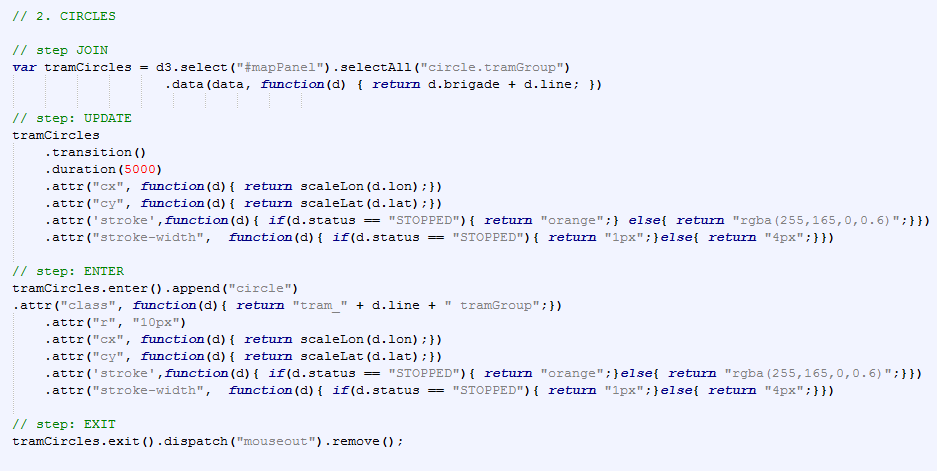
And if we refresh our visualization we can see exactly that. Each change of dataset that feed visualization, and therefore change of tram positions are connected with change of specific HTML and SVG elements’ attributes. So if position of tram is changed in new dataset, the position of circle that is linked with this tram, this row in our dataset, is also changed by the change of its “cx” attribute.



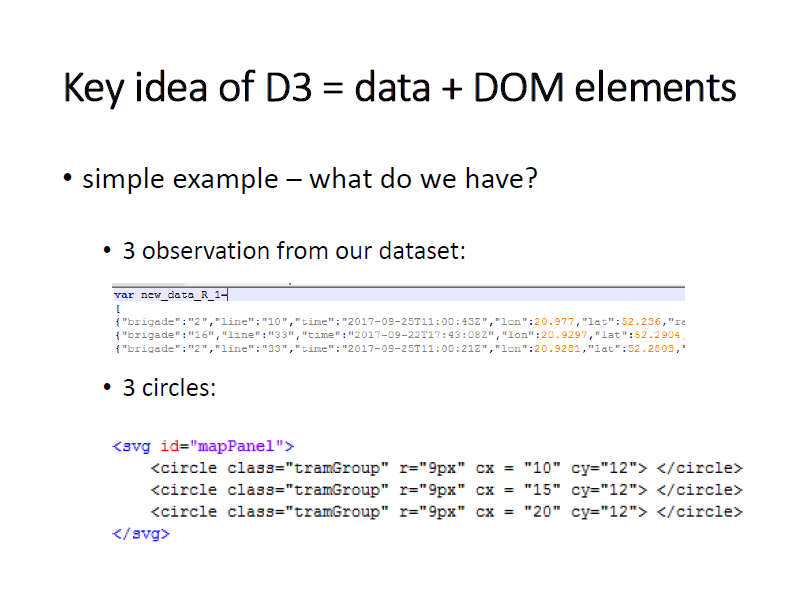
**(pokazać na tag html, potem na tag head, potem na tag image/div/svg/html)**

Second thing about HTML is its hierarchical structure. Each element is nested in some more general tag, so the most general are ‘html’ tags, in them are nested every other tags, so for example tag head is nested in html, or for example tag image is nested in tag svg, more general some tag div and the most general tag html. It will be important when we will talk about D3 code.

# D3 KEY CODE: UDPATE- ENTER-EXIT SCHEME

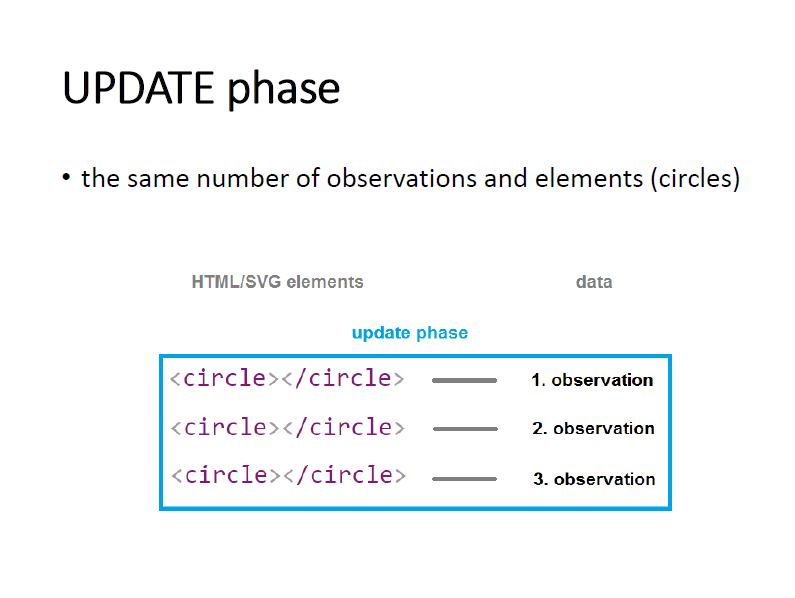


So let’s talk about it. I will discuss now one part of the code that generates our visualization and it represents the most important part of the D3 code in each visualization. It is so called update-enter-exit scheme and is responsible for binding data with HTML document in such way that visualization is capable of updating itself when new data comes.

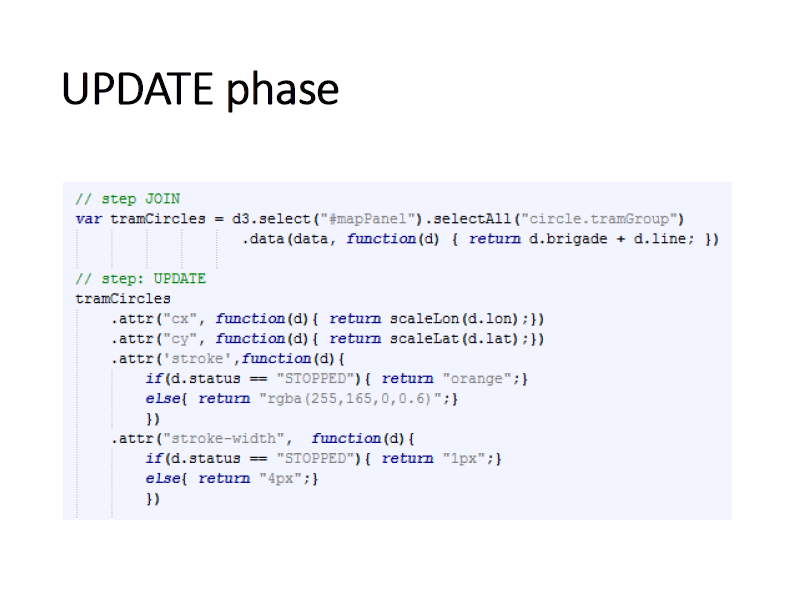


To simplify our example we will focus only on circles and only on a few of them. So what we have in our example here. We have just 3 observations in our dataset and 3 circles in our HTML document. We will now see how we can bind these data with these circles and how we can deal with new data.

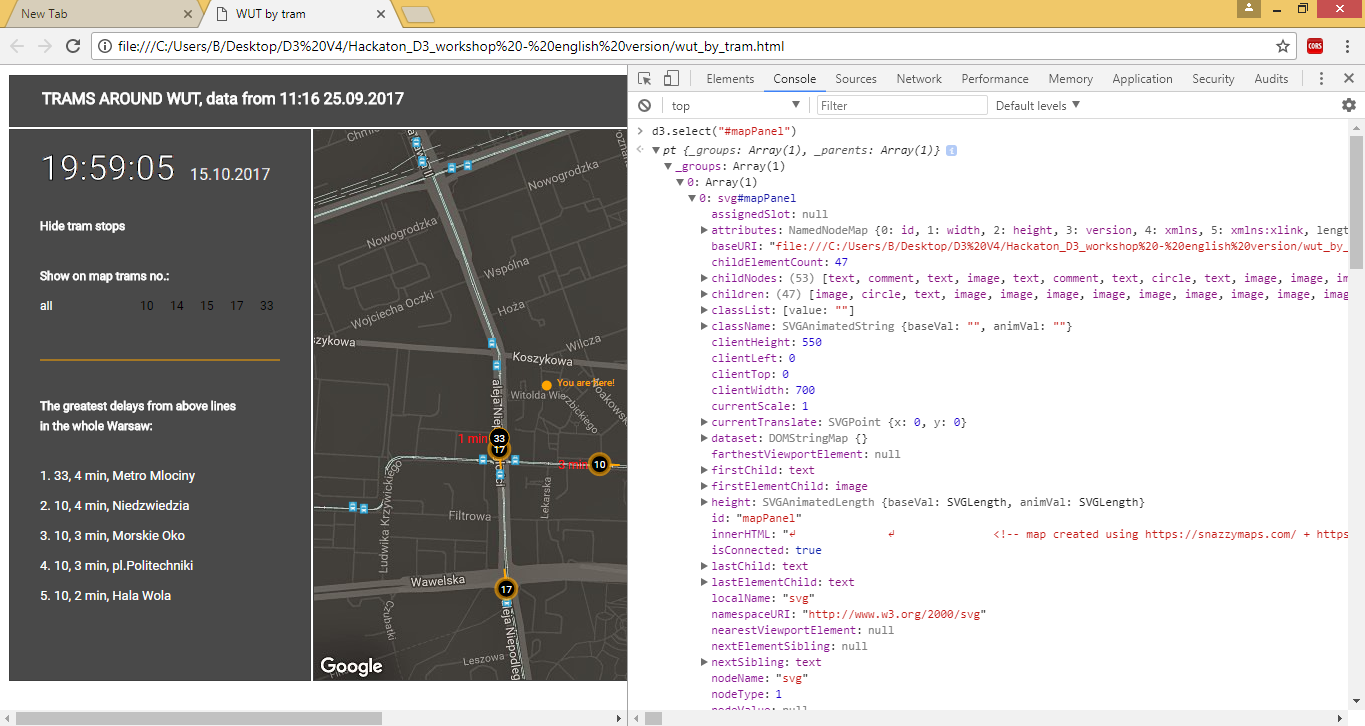
# UPDATE PHASE



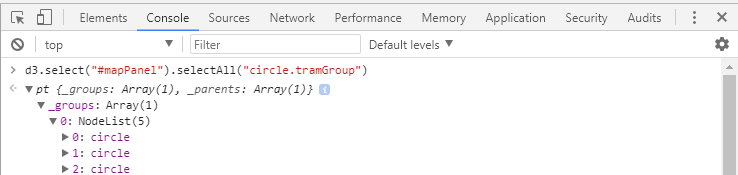
This classic D3 scheme begins with UPDATE phase. In this phase we already have some circles in our HTML document and all we want to do is to bind data with them, that is in our tram example we have 3 observations from our dataset and we have 3 circles in our HTML document and we want to move our 3 circles to the positions that are defined by these 3 observations. What do we have to do?



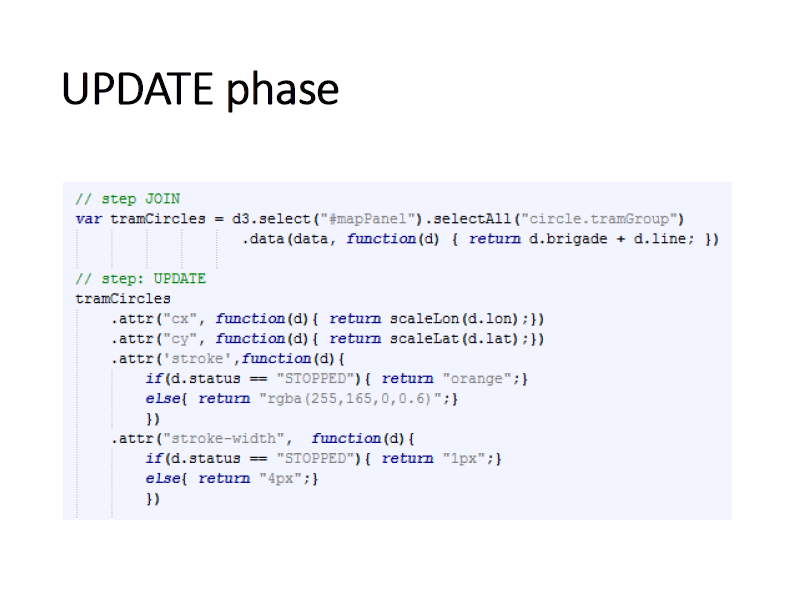
As I said, we bind data with HTML or SVG elements, so, it is logical that first thing to do is to find in our HTML document elements that are interesting for us, that is circles on the map. So firstly we should select the map, and then all circles that are in this map. To select anything from HTML structure we use in D3 functions **select** and **selectAll.** First one select first element that meets given condition and second every element that meets this condition. So as we have here only one map, we can select it using function **select** and within brackets we give condition that identify our map. Earlier I gave our map id called mapPanel, so we use it here.



This function will return us exactly this element in our HTML document that represents our map and it will be returned as an object with lots of attributes that keeps extra info about it, like what are its properties (class, id, color, position, etc.), what is its parent (more general tag), what are its children (tags inside this element) etc. and we can work with it easily. So now we want all circles that are in this element, our map, so we use function **selectAll** on this returned object.



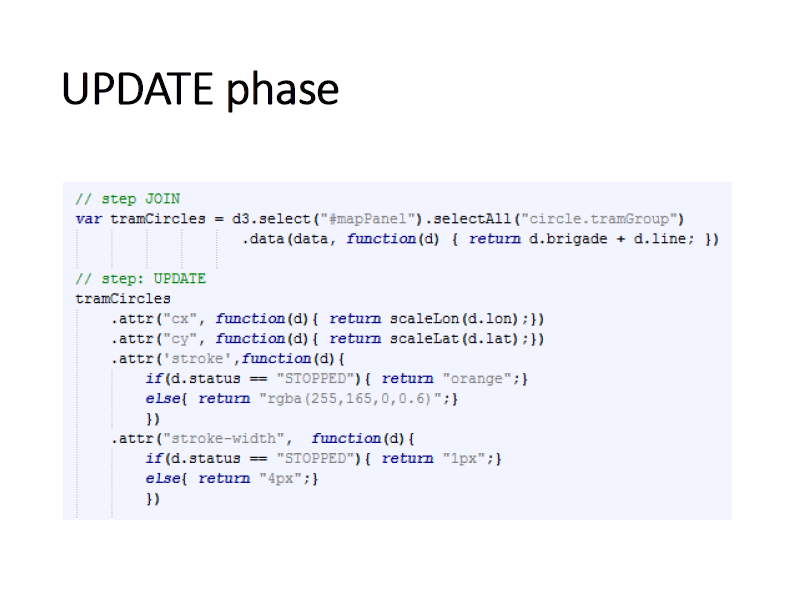
As a result we get all 3 circles that are on our map in this simplified example. As you can see in D3 we can write consecutive commands right after previous ones, we just have to join them by a dot.



Ok, so we have grabbed appropriate elements from our HTML document, now we have to bind them with data. To do that we simply add on the end of these selection function **data(),**  with first argument that defines what will be a source of data for these elements. Here it is an object called simply “data”.



As a result of these operation we have elements bonded with data, more technically we add new property called “\_\_data\_” to each of our circle and in this property are stored appropriate rows from our dataset.



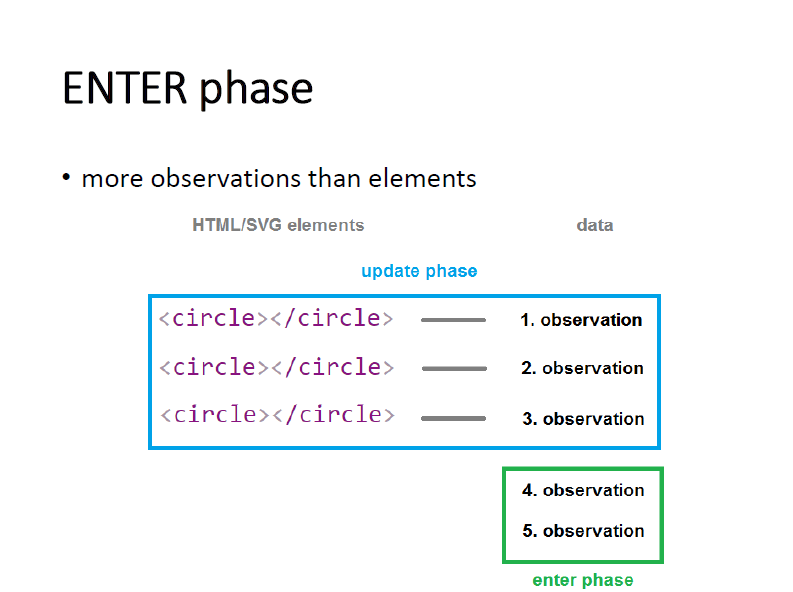
The last step is to simply save these selected elements to a variable **tramCircles.**

So this was just joining data and elements, but now we want to make use of these data, so we want to update properties of our circles according to our data.

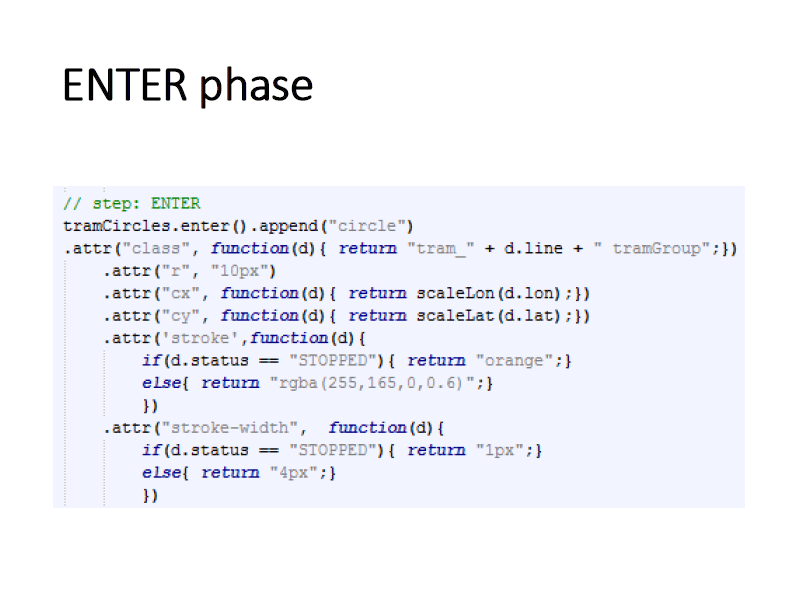
To do this we take our variable “tramCircles” that keeps our circles and with the use of function **attr** we change particular attribute of circle tag to a particular value. First argument defines what attribute we want to change and second one defines to what value. It can be constant value like 5 pixels for attribute “r” (radius of a circle) or it can be set according to data that is bonded with the circle by using function as the second argument, as we did here. Here we change horizontal position of circle that is defined by attribute “cx” to transformed longitude of tram that is defined in data row by “lon” key. Function **attr** works on each circle, this circle is selected separately, we can say that function attr() is called in loop for each circle in selection.

And in this way we can automatically, using our data, change attributes of our circles. If we use now here another 3-elements dataset this code will automatically update attributes of our 3 circles according to this new dataset, that is if for example positions of trams were be updated, the circles are moved to new positions.

# ENTER PHASE

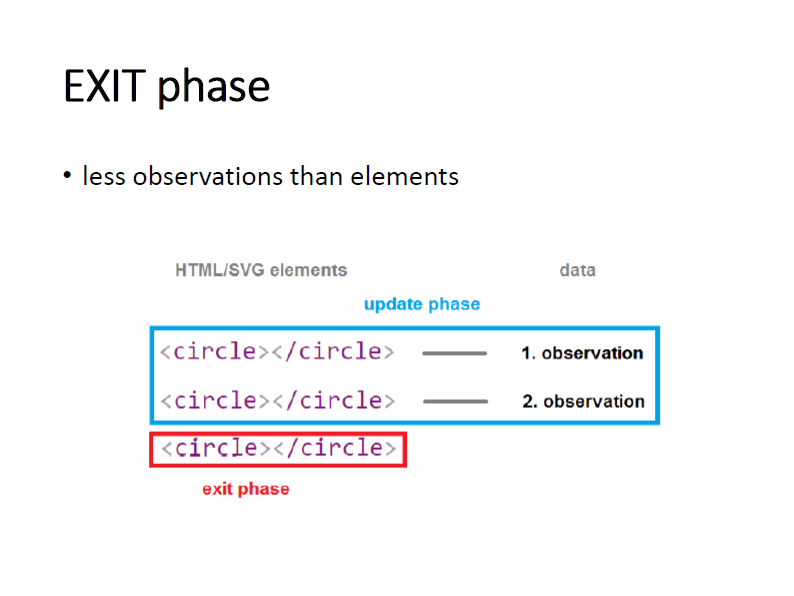


But what if we have situation in which we have more observations in our new dataset than circles? Here we have 5 observations, first three are bonded with these 3 circles that we have, they belong to UPDATE phase that we have just discussed. But what with these remaining two observations? Well, it will be desirable that if we have extra rows, and that means we have new trams in our neighbourhood, we want to add them to our map, so we start for them phase called ENTER. Here we take this new data that are not connected to any circle and create circles for each of them.



To do that we call function **enter()** on our tramCircles variable. Why? We should know that function **data(),**  that was used in the previous UPDATE phase not only binds data with existing HTML elements but also keeps track of unused observations from the dataset. So using **enter()** extracts info about these unused observations that was saved by **data()**  function and makes for each new observation as though empty place in our html document where new circle should be added. So we have 2 new observations here, and we will have two empty places. To really accomplish this addition of circles we use function **append(“circle”)**  that appends circles in these empty places in our HTML document and returns only these new circles binded to these two observations so we can use them later, for example set attributes for them as it was done in phase UPDATE.

# EXIT PHASE



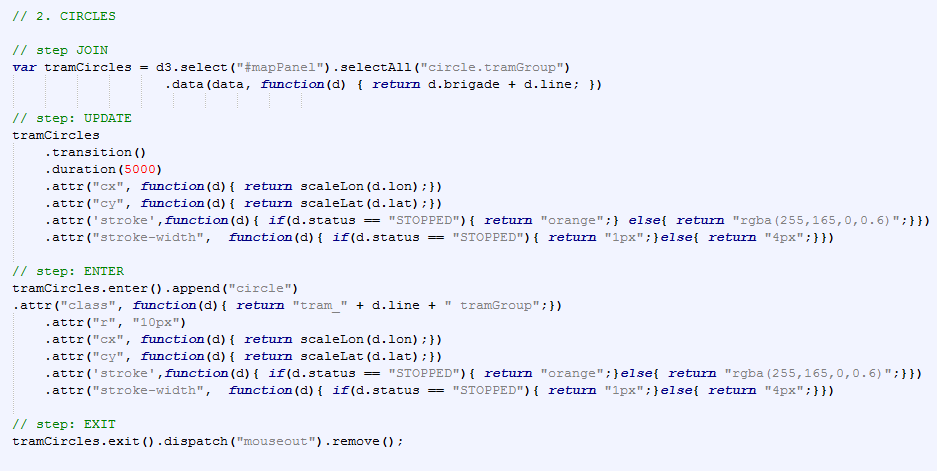
And the last situation that we can encounter when we feed our visualization with new data is situation, when we have more HTML/SVG elements than we have observations in our data. We will deal with this situation in phase **EXIT**.

Earlier in situation where we had the same amount of HTML elements as observations in new dataset, we just swapped the data and updated visualization. When we had more observations than elements, we created new elements for them and now when we have less observations than elements it is natural that we want to remove these extra elements that has no data to bind.



To remove them we use again our tramCircles variable that thanks to data() function keeps also information about elements that have no data, and we call function **exit()** on it that selects these elements and returns them. Now we can remove them using function **remove().**

# SUMMING UP – HTML +D3



And in this way adding, removing and updating of elements of visualization based on the change of datasets is accomplished in D3. Firstly we select appropriate elements by **select** functions, then we feed these elements with data by using **data()** function, and then we can update properties of them using **attr()**, add new elements selecting new data by **enter()** function and adding them by **append()**  or selects redundant elements by **exit()** and remove them by **remove().**

We use this 3-phased scheme UPDATE – ENTER – EXIT that guarantees us that whatever new dataset will come to our visualization, whether it has more, less or the same number of observations as before, it will be properly handled and our visualization will be appropriately updated. And this is the most important code in D3, that accurately explains how D3.js library creates Data Driven Documents.

# ~~TRANSITIONS + THE END~~