# TOWARDS MATHJAX V3.0

PETER KRAUTZBERGER, DAVIDE CERVONE, AND VOLKER SORGE

## Executive Summary

Since the release of v1.0 in 2010, MathJax has become the de-facto standard for rendering mathematics on the web. While MathJax's various input and output components have evolved over the years, MathJax's core component remained fixed. Today, both MathJax and browser technology have reached a point where an extensive redesign of MathJax's core component will enable significant improvements and ensure the long-term viability of MathJax.

The core work of the redesign will revolve around modularity. MathJax's original design needed to provide its own, specialized framework to tackle the challenges specific to mathematical rendering across all browsers (in 2010). Since then, the modularization of web technology has made much progress while MathJax's newer components require less complexity from its core component. The timing is right for improving MathJax's module structure and APIs. This change should ensure MathJax will better serve today's highly complex web development ecosystem. A positive side effect should also be performance improvements and the ability for developers to re-use individual components efficiently.

The redesign must be governed by MathJax's mission to provide the best tools for the mathematical and scientific community on the web. A key consideration is one of MathJax's original goals: to spur native MathML implementations in browsers. Thanks to MathJax, millions of users benefit from the advantages of MathML on the web every day, with the MathJax CDN alone serving over 4.5 million daily visitors. Unfortunately, browser vendors continue to lack interest in implementing MathML while volunteer efforts have proven unreliable and of limited scope.

Depending on whether or not we keep this original goal, we considered two directions to guide the redesign. If native browser support for MathML remains a core goal, we believe MathJax should focus on becoming a modern polyfill, i.e., enable MathJax to leverage partial MathML implementations in browsers and become as invisible as possible to developers. If browser support for MathML ceases to be a core goal, we believe MathJax should focus on perfecting its transformation of MathML into HTML/CSS (and into SVG), i.e., MathJax should focus on enabling a rendering that is fully equivalent to MathML and can be generated on both server and client.

After careful consideration and extensive feedback from the MathJax sponsors, the MathJax Steering Committee, as well as various experts in the field, the MathJax Consortium follows the recommendation of its development team to pursue the second direction as the guiding principle for the planned revision.

To ensure that we achieve our goals, MathJax is forming a Technical Committee of developers. This committee is being recruited from the MathJax sponsors as well as experts from the community. In addition, the redesign will require significant resources, including an additional core developer.

We are grateful for the unanimous support from our MathJax sponsors to support this effort and ensure MathJax will continue to provide the best tools for math and science on the web.

## Introduction

This August marked the fifth anniversary of the release of MathJax v1.0. In the past five years, MathJax has become the standard solution for publishing mathematics on the web, growing from a one-developer project to a mature project with a dedicated team, multiple contributors, and a rich ecosystem built around it. Today, the MathJax CDN serves 4.5 million unique visitors each day and is used on thousands of websites, including over 400 of the top 1 million websites according to libscore.com.

We find ourselves in a very different World Wide Web today. With its success, the expectations towards MathJax have grown. When MathJax started, it was considered a temporary solution, to bridge the time until browsers implemented MathML alongside HTML5. Today, that moment seems further away than it was 5 years ago with the two leading browsers (Internet Explorer and Chrome or 55-75% of the market) having no plans to support MathML, even actively removing support for MathML or for plugins that could compensate.

Moreover, web development has changed drastically over the past five years, both in terms of tools (libraries, frameworks, platforms) and standards (HTML5, Web APIs, ECMAScript 2015 etc), with the former often influencing the latter. This has changed the requirements for MathJax within modern web developer workflows and also the expectations towards MathJax. In short, MathJax is showing its age because development had to focus on maintenance and conservatively extending functionality.

This paper discusses the risks and benefits of overhauling a significant portion of MathJax to enable another 5 years of successful development and delivery.

## Background

Making extensive changes to any piece of software carries risks that need to be outweighed by the opportunities provided. This is especially true when considering changes that are not fully backwards compatible. As we are considering significant (breaking) changes to MathJax's core component, we also face a question of timing. MathJax development began at a point where web technology improvements were ideal for re-designing its predecessor jsMath. Given the responsibility towards MathJax's community and MathJax's donors, we need to ensure we have reached a similarly opportune moment in terms of existing and nascent technologies as well as in terms of our approach towards leveraging them for long term sustainability of the project.

**MathJax.** 5 years on the web is an eternity. For an unfair comparison, MathJax is roughly the same age as jQuery, NodeJS, or AngularJS, all of which have passed through several iterations since. Naturally, these projects have a much larger contributor base and financial backing. But the extensive changes in their design indicate how drastically the web development landscape has changed.

Still, MathJax has seen major improvements over the past 5 years. Thanks to its modular system, many components have seen a partial or complete redesign. Additionally, the advantage of modularity is that instead of redesigning a monolithic piece, we could develop new components (inputs, outputs, and their extensions) which provided the same effect as re-writing an existing component while simultaneously keeping older solutions available (and often improved). However, there are some limitations within MathJax that cannot be resolved this way.

At the core of the overhaul we are proposing lies MathJax's core component which has not been revised since v1.0.

Until very recently, a major overhaul of MathJax's internals would have had extremely limited scope. That is, much of the core component was simply necessary for MathJax to provide high-quality layout. However, the progress we made in 2014/15 with the new CommonHTML output alongside improvements in the browser landscape provide us with an opportunity to redesign several aspects of MathJax's core component and its features. Simply put, the internals have begun to hold MathJax development back and we believe their disadvantages are now outweighing their advantages.

Such an overhaul will include many changes that will break current behavior such as changing APIs or removing outdated components. A good example is MathJax's original rendering component, the HTML-CSS output. This original output component was designed to work reliably on IE6+ and all other browsers and platforms that were current in 2009. This forces MathJax to workaround significant limitations, including erratic layout behavior, unreliable webfont integration, and low level of JavaScript features. Not surprisingly, much of MathJax's infrastructure was required for the HTML-CSS output given these restraints on old (now ancient) browsers; changing the core component will result in the removal of the HTML-CSS output while the new CommonHTML output remains as new default. Similarly, changes to the internals will change MathJax's APIs extensively. Of course, it is important that any breaking changes (in particular, feature deprecation) are compensated by improvements.

Equally important to us is that MathJax is not just another software development project. MathJax is not developed for a particular company or organization but instead we are driven by a mission shared by our managing partners as well as our sponsors: to provide the best tools for the mathematical and scientific community on the web. At the core of MathJax has always been MathML, the web standard for math and science. One goal of MathJax was to break the vicious cycle of "no browser support $\Rightarrow$ no MathML on the web $\Rightarrow$ no need for browser support $\Rightarrow$ ...". Despite MathJax's success, we do not seem to be any closer to native MathML support today than we were 5 years ago. In fact, we seem further away than ever.

**Browsers in 2015.** The situation of native implementations can only be described as confusing.

- IE/Edge lists MathML as "not currently planned" on its roadmap while removing support for plugins such as MathPlayer that could compensate.
- Firefox/Gecko's MathML implementation has been the work of volunteers over many years and Mozilla engineers are supportive of these volunteer efforts. In 2014, a crowd-funded volunteer effort made some progress but unfortunately lasted only 6 months; no substantial progress has been made since.

  Gecko MathML support is incomplete but in terms of plain feature coverage not too far from MathJax (and ahead in terms of bidirectional layout). However, layout quality is sometimes sub-par or unreliable while common Web APIs are often not supported. It can be used in production but often requires knowledge about its specific implementation quirks.
- Safari/WebKit has seen three attempts by three successive volunteers to move its implementation forward. These volunteers worked alone with little support from WebKit companies; all of them eventually ran out of time and/or money. For the past year, there has been no active work on WebKit's MathML support. Apple advertises MathML support in Safari while in practice it is not usable in professional production. Notably, WebKit's MathML pages have not been updated in several years and at time of writing there were 136 open bugs filed under MathML; MathML is also not included in the recently added "WebKit Web Platform Status" dashboard. Apple has implemented partial support in its proprietary VoiceOver technology.
- Chrome/Blink lists MathML as "no longer pursuing". Blink removed the code base for MathML that it inherited when forking from WebKit. The Chrome team has been consistent that support in Blink is not planned. While negative, it has been the most transparent and consistent position. In extension, this position applies to Opera, Vivaldi and other Blink/Chromium-based browsers.

It is worthwhile to note that Chrome and IE/Edge make up 55-75% of today's browser market, depending on the metric).

Ultimately, we believe actions speak louder than words: **no browser vendor has worked or is planning to work on implementing MathML**.

In addition, unfunded volunteer-driven efforts have repeatedly failed to reach the 80/20 point of the implementations. In 2013, MathJax extensively investigated ways to reliably fund long-term, third-party MathML browser development, however the lack of interest from browser vendors made it difficult to pitch this idea to potential funders.

On the other hand, other components of HTML5 have been either implemented, revised, or marked as abandoned. In addition, browser layout engines have matured a great deal and layout has become very reliable across browsers. Where MathJax originally could not even rely on basic layout such as text widths being handled correctly, today's browsers are reliable even to the point of CSS 3 implementations.

This enables new approaches to math layout. With nascent technologies, a different path towards native math layout seems feasible.

**Web standards.** In the past year, MathJax has begun to become more pro-active regarding web standards development to better fulfill its mission of moving math and science notation forward on the web. By supporting Peter Krautzberger as an invited experted to the W3C's Digital Publishing Interest Group (DPUB IG) and the W3C's Math Working Group (Math WG), MathJax is developing new expertise on use cases, technologies, as well as evolving standards to align with, while in turn providing feedback to web standards development. In the DPUB IG, Peter Krautzberger leads the STEM task force. Overall, there is a clear frustration among STEM web experts regarding the lack of progress for MathML. At the same time there is interest in exploring pragmatic ways to improve the situation of math and science on the web, especially in the context of recent developments such as the Houdini Task Force and modularization efforts in ARIA (such as the DPUB-ARIA module).

In summary, we believe the improved browser technology landscape as well as our own progress over the past two years provide an opportunity to re-think MathJax's core component on a fundamental level. We believe these decisions have to be connected to MathJax's mission. We need to re-evaluate the current goal of native MathML support in browsers and consider shifting the focus towards other pragmatic goals that help math and science become first class citizens on the web.

## Goals

For a redesign of MathJax's core component it is crucial to re-evaluate our long-term direction. This is where we find ourselves at a cross-roads.

A critical problem today is that MathJax generates output that currently can only replace MathML in the page, not augment it. This means we cannot leverage partial browser implementations (e.g., implementation of roots or tables) and we cannot provide a positive feedback loop for browser implementations. In other words, MathJax's current design cannot function like a polyfill/prolyfill should (in the sense of the Extensible Web Manifesto): providing an implementation in JavaScript that can gradually be replaced with native browser implementations.

**The core work: revisiting MathJax's core component.** We need to redesign MathJax's modular structure and the core APIs derived from it to improve performance, re-use and long-term maintenance of MathJax.

The need to revise MathJax's modular structure relates to current and future best practices in web development. While MathJax's modular extension system has allowed MathJax to gradually upgrade older components as well as develop new, modern components, the modularity has been "internal" only; MathJax components cannot be used outside of MathJax and developers have to understand MathJax's custom module system and programming model to modify or build components.

We need to revise this to better address the use cases of modern web developers. To get a rough idea of the complex workflow and tool chain into which MathJax to fit, consider Table 1.

| NEED | MOTIVATION | TOOLS |
|---|---|---|
| Scaffold | Several tools. Several ways. Several Practices. Need to organize, and give some good foundation - best practices, good design. | yeoman, Seed Projects, Html5Boilerplate, bootstraps (e.g. Twitter Bootstrap) |
| Build / Automation | Lots of tasks to execute. Compile. Test. Minify. Concat. Etc. | grunt gulp, broccoli, component, ake's (e.g. Make, Rake, etc.) |
| Automation Utilities | Tasks that can be put in build the pipeline. | minify, uglify, lint, jshint, watch |
| Dependency Management | Applications are getting complex. They rely on several other libraries and frameworks. | bower, component, NPM |
| Dynamic Loading | Big projects are split among several pieces of js for the sake of modularization. No all of them should be loaded at the same time. | require, curl, amd.js, async.js |
| Javascript Preprocessor | The way you organize code in development time is dierent the way you publish your code. Need to do some processing in your javascript les before using them. | browserift, webpack |
| Application | Applications on web are getting complex, need for frameworks that support app development. | angular, backbone, ember, knockout |
| Application Utilities | Several application features that can be necessary (e.g. routing) | page, director, crossroads2 |
| Test Runner | Execute and visualize test results | karma, saucelabs |
| Test Framework | Write tests | jasmine, mocha, qunit |
| Test End to End | Write tests for the whole application ow | protractor, casperjs, nightwatch.js, watir webdriver |
| Test Support | Support tests and helpers | phantomjs, zombie.js, sinon, chai |
| Dom Utilities | DOM selection and maniputation, some auxiliary functions, need for utilities that make work simple (and cross-browser) | jquery, zepto, polymer, prototype |
| JS Utilities | Clean code, functional programming style, reactive programming features, helpers and utilities | lodash, underscore, promise, fn.js, q.js, bacons.js, sugar.js, chance.js, moment.js, micro.js |
| CI | Continuous integration, continuous delivery, continuous deployment | Any! (e.g. travis ci, jenkins, concrete, semaphore, go, snap) |
| Language | Have a syntactic sugar element, or even completelly dierent syntax (that in the end turn into javascript to run in the browser) | coeescript, clojurescript, typescript 3 |
| CSS Preprocessors | | sass, less |
| Preprocessors Libs | | compass, bourbon |
| CSS Helpers | | susy, zenGrids, neat, normalize, modernizr, exbox |
| CSS Frameworks | | boostrap, foundation, skeleton |

TABLE 1. From Slide 1-3, "The JavaScript Toolkit 2.0".

We need to ensure that MathJax fits better into this diverse ecosystem. At the heart of this problem lie modules.

Originally and for lack of available technology, MathJax had to create its own module system, including dynamic module loading mechanisms and webfont detection. These elaborate internals made MathJax more akin to a full-fledged web framework rather than a "regular" JavaScript library. In other words, developers had to adapt to MathJax's framework rather than having a utility library that they can integrate into their framework of choice. This made the integration work of developers more complex than one would expect today.

In the past few years, several module systems (e.g., CommonJS, AMD, UMD) have given rise to native modules in JavaScript itself (starting with ECMAscript 2015). The progress we have made on MathJax in 2014/2015 allows us to redesign our modular structure so as to simplify it and turn more components into independent, reusable components using modern best practices. Such a redesign must focus on web developers who need to fine tune the components they ship and integrate them into modern tool chains.

We can easily continue to supply compiled packages allowing the average users to use MathJax as they do today – inserting one line of JavaScript into their header and "forgetting" about it.

But by making our components easier to reuse outside of MathJax, we fulfill our mission better, enable developers to use MathJax more flexibly, and lower the threshold for outside contributors.

Alongside our module structure, we want to simplify our APIs so as to make it easier for developers to integrate MathJax into their applications. Currently, MathJax acts more like a framework, with a complex callback, signaling, and queuing system and several internal optimization methods. The problems MathJax solves this way are now much more common place in web development and most developers will have their own tools or frameworks to handle these problems (e.g., reducing paints, reflows and DOM manipulation). This allows us to simplify those APIs by moving the burden from MathJax towards developers since they already carry that burden anyway. At the same time, we avoid MathJax's own methods from interfering with the developers' choices.

To some degree, this kind of change will make MathJax less flexible in certain use cases (e.g., dynamic adaption to client configuration, output switching, font switching etc.) and more importantly this will be heavily influenced by the overall direction discussed below.

To compensate, we can build a developer kit of sample integrations that implement this old functionality. Ideally, a wider community of developers will share sample integrations into the most common tools and frameworks as we've seen with similar situations in the past (such as mobile apps or CMS integration).

The key difference for MathJax will be a shift towards making additional flexibility an opt-in for developers and users, removing negative effects on our core requirements. These technical priorities must be governed by our overall design direction. We evaluated several approaches and identified two main candidates.

**Path 1 "Seamless polyfilling".** As mentioned, we need to decide if our mission should remain focused on enabling browser development. If this remains the focus, then we should re-design MathJax to provide a positive feedback loop for MathML implementations. The strategy for this approach would be to become "invisible" to both users and developers.

This approach would be marked by using newly established webstandards. For example, we would need to

- use custom elements and shadowDOM to make MathJax output appear as its underlying MathML,
- design our components to detect partial MathML features and use them when available (e.g., roots, menclose, mstyle),
- use mutation observers on the MathML lightDOM as new core API,
- so that developers do not interact with MathJax, they just manipulate MathML in the DOM,

- make MathJax fast enough so that developers are comfortable to "forget" about MathJax rendering, i.e., they inject MathML and rendering is seamless and non-interfering.

The advantage of this approach lies in the fact that web components has gained enthusiastic support and the problems MathJax would be facing are to some degree shared with more developers. Additionally, once even partial MathML implementations are available, MathJax can improve automatically.

The risks of this approach lie in the complexity of the task. This kind of functionality is far more complex than what MathJax does today and it eliminates many tricks we use in MathJax to ensure high quality and fast rendering speed. This approach would also have to rely on computationally heavy browser technology (such as mutation observers, getComputedStyle etc). It is unclear at this point whether web components can help with accessibility issues related to polyfilling MathML since custom elements cannot redefine HTML5 elements and assistive technology would still be exposed to the shadow tree (cf. the latest Shadow DOM working draft).

The main question is: can we speed up MathJax sufficiently? And would this modus operandi actually be aligned with web development practices? Finally, this approach would also eliminate any solutions for environments without JavaScript as custom elements can only be created using JavaScript.

**Path 2 "HTML5 rendering".** If leveraging MathML browser implementations is not the goal, then we should design MathJax to provide a fully-equivalent "interpretation" of MathML in HTML.

For example we would

- design the core component to focus on working independently of the client context,
- in particular, make server-side processing a first-class use case alongside client-side processing,
- enrich all output to be at least as powerful as native MathML rendering,
- leverage web standards that enable HTML and SVG output to embed sufficient semantic information and work on improving such standards,
- focus on DOM-independent processing so as to leverage other web technologies (webworkers, serviceworkers) and development techniques (e.g., virtualDOM),
- provide lightweight client-side tools to enhance the pre-generated output.

The advantage of this approach is that MathJax can focus on making its rendering "just another piece" within an HTML page or SVG document, leveraging regular HTML5 structures, and integrating well with other content. It would also allow MathJax to enable more functionality outside the client-side browser, allowing for pre-processing, in particular improve the situation of MathML in non-JS environments such as ebooks. Pre-processing would also resolve most performance problems since no JavaScript processing would be necessary on the client-side browser (but of course would remain equally possible, e.g., for interactive content).

The greatest risk of this approach would be the potential damage for the prospects of MathML on the web, turning the focus to MathML as a data format for HTML and SVG. It would also somewhat complicate the interaction with accessibility tools which, despite lack of (visual) rendering in browsers, have begun to focus on handling mathematics only as MathML in the DOM; as noted earlier, this problem exists in either approach. We have reached out to the assistive technology community as well as the Protocols and Formats Working Group to discuss the potential of improving related web standards and there seems to be interest from both sides.

However, if one assumes that browser implementations are unattainable, then the approach of interpreting MathML in HTML5 can benefit standards development significantly as it would help identify which features of MathML might be obsolete in an HTML5 setting and which forms of mathematics are not yet possible in MathML but possible in a HTML5 setting, e.g., diagramatic content. By providing an easily extensible model, it could allow for a more dynamic development of mathematics on the web.

Finally, this approach could identify a minimal set of additions to HTML and CSS that might simplify such an interpretation of MathML. In particular, it could inform initiatives such as the recently formed Houdini Project on which elementary rendering APIs are necessary for mathematical and scientific notation. Ultimately, this approach could work towards "merging" MathML into HTML rather than work towards MathML as a separate set of standards.

**Moving Forward.** After careful consideration of all alternatives and many fruitful discussion with the MathJax Steering Committee, the MathJax Sponsors, and many experts in the community, we decided to follow the second path.

It seems difficult to argue that browser vendors will actively implement MathML natively in the next 5 years, no matter how much MathJax might do to encourage such implementations. The longer implementations are delayed the less likely it becomes that native MathML will be realized at all, as HTML, CSS and other standards of the Open Web Platform continue to evolve without MathML taking part in such change.

The complexities of the first path pose a serious challenge as they will most likely affect MathJax performance negatively. This could only be compensated with partial native implementations to eventually take over the difficult layout tasks. However, without such implementations, MathJax stands to worsen as a product.

The second path moves our focus towards implemented HTML5 standards. With the new CommonHTML output, developing an "interpretation" of MathML via HTML/CSS and SVG that is as powerful as native MathML is an attainable goal. Working towards HTML/CSS improvements that simplify this "interpretation" seems much more likely to succeed.

This path also makes server-side processing a first class use case for MathJax. The ability to pre-generate rendering resolves performance issues as JavaScript is no longer necessary on the client – yet MathJax will still work equally well on the client.

Finally, the second path would still allow for client-side enhancements that realize the first approach (i.e., using web components to turn the output into faux MathML) while the reverse does not seem possible. Overall, native MathML support, while ideal, seems unrealistic, whereas an "interpretation" of MathML is a pragmatic approach with equal potential.

Another important consideration is future expandability. It is clear that other scientific, technical or artistic content will need a more natural representation on the web. Indeed, there already exist a number of specialist languages for such content (e.g., CML, MusicXML, PhyloXML) that might push for inclusion in web content in the future. Similarly, math on the web pushes beyond MathML on various ends, from geometric representation to computer algebra systems to new forms of visual expression of mathematical thought.

The MathML experience teaches us that not even inclusion into W3C standards leads to implementation of specialist languages in browsers. Given that HTML5 is the basis for ebook standards such as ePub3 and mobi/kindle, it also means that every eBook reader needs to implement the full spec. But if browser vendors already do not implement the full standard, it is even less likely that developers of eBook readers will. Consequently, there is always a necessity for polyfill solutions like MathJax and there is an argument to be made that this will even increase in the future, with other markup languages reaching maturity. And equally consequently, any progress towards realizing MathML and mathematical knowledge in general in HTML and SVG could provide a path for other knowledge domains.

We see a future market in MathJax to provide rendering solutions for these scientific languages and consequently need a flexible and future proof underpinning. That means firstly we want to restrict the rendering output to those parts of the web standard that are widely implemented in all display solutions, i.e. HTML, CSS and SVG. And secondly, we need a flexible and efficient core system that is based on modern JavaScript standards and that can be more easily adapted and updated to future iterations of web applications technology.

## Methods

We want to update our core component and modular infrastructure using current and future best practices of JavaScript development. The redesign should therefore also bring about a change in MathJax's development process.

In this context, it is important to realize that we are considering changes that will hold back development for a signficant amount of time, most likely a full year. However, by modularizing MathJax more aggressively and by following a more dynamic development process, we can release early and often, focusing on smaller and independent releases of our components that developers can integrate rapidly. In particular, we will not strive to deliver an equivalent product from the start but quickly provide incremental releases of individual components.

This change also includes building more on other polyfills and JavaScript compilers/transpilers. This will allow us to develop MathJax for the browser market of 2016 and 2017 rather than the lowest common denominator of 2015.

Because the web development ecosystem has become very diverse, we are painfully aware that we cannot hope to keep track of all current and emerging technologies. That is why we will form a Technical Committee consisting of developers from our sponsors, our contributors, and other specialists. Building a strong connection with this dedicated group of developers will guide our design decisions towards the broadest positive impact.

Finally, we will continue to maintain MathJax 2.x in terms of bug fixes and third party contributions.

## Resources

To make this change, we need to expand our team to permanently have two equal core contributors. For this, we need to ensure that our development schedule can be aligned, e.g., in the form of dedicated development sprints. This includes aligning our work with the members of the technical committee, who we will rely on for providing feedback during development sprints.

## Measuring success

As we pursue a path towards an HTML5 "interpretation", the primary measure of success will be how well MathJax can provide a solution for mathematics on the web that is at least equivalent to native MathML browser implementations, for both developers and end users and in particular in terms of accessibility. Our internal approach of using MathML will not change but its role in our rendering is open to change as we work towards realizing MathML in HTML. We are dedicated to the Open Web Platform and its standards and our work will continue to be focused on providing feedback to standards and browser development; therefore progress on web standards will also be a measure of success.

Rendering speed will naturally be a critical measure, as will be overall functionality (i.e., MathML coverage and more advanced math and science use case such as diagrams and responsive rendering).

Finally, a timeline developed by the new team (with input from the technical committee) will provide further means of tracking progress in the first quarter of 2016.

## Conclusion

We believe we have reached a point in time where we could and should undertake significant changes to MathJax's core component to adapt to a changed landscape. The nature of these changes depend on a critical decision for the overall direction of MathJax's mission – to no longer consider native MathML support in browsers an achievable goal of MathJax. We believe we have identified the relevant technologies to successfully make this change and ensure that MathJax greatly improves and serves the community for the next 5 years.

We have sought out our sponsors and other members of the MathJax community to ensure we have their input and support for this change. As part of this change, a group of expert developers will support us as a technical advisory group that will

advise us in this effort. Thanks to the unanimous support of our MathJax sponsors we look forward to tackling this change towards greatly improving MathJax as a long lasting, high quality tool for the entire community.