“XBlock 2” Proposal

This proposal outlines a way to make incremental changes to XBlocks (and the LMS/CMS) to improve some of the shortcomings of the current learning component architecture.

What do we need to fix?

XBlocks have been very successful and useful in their role as the building blocks of the Open edX platform’s learning experience. However, experience has shown that the design of XBlocks has a number of issues that holds the platform back today:

1. **XBlocks are slow and complex to render**
   To display an XBlock to a learner (or even to an instructor in Studio), a lot of “work” and resources are required. First, the runtime must be initialized, then the field data must be loaded from the database(s), then the XBlock “view” python code must be run, then the resulting HTML and JS bundle must be sent to the browser, and combined with various other JavaScript that the XBlock “expects” to be present. If you are [pre]viewing a page with 100 XBlocks, some parts of this (like running the “view” python code) must happen 100 times.

2. **No firewall between the LMS and XBlocks**
   XBlocks are written in python code and run as part of the LMS (or Studio) process. XBlock code can therefore read, access, and modify any part of the platform or do almost anything on the system (e.g. access full student records in the database, erase the database).

   Consequences:
   - **a.** you cannot install an XBlock unless you trust its author and have verified that the code doesn’t do anything malicious, so in practice:
   - **b.** Only administrators can install new XBlocks; course instructors cannot.
   - **c.** XBlocks can do stuff that slows down the LMS (degrades performance)

3. **No sandbox around user content**
   Whatever HTML code authors put into an XBlock or its various fields typically will appear on the LMS page unescaped, which means that a malicious course author (or one who was misled by phishing etc.) could put JavaScript code into an XBlock that exfiltrates user data like session tokens.

   Consequences:

   a. Open edX administrators must be careful about who has permission to author new courses, and access to course authoring must be restricted.

4. **Sometimes vague API**
   There is an official XBlock python API that defines how XBlocks can interact with the platform, and the LMS provides additional API extensions (e.g. self.location) and services, but in practice XBlocks can (and do) call any python code they would like (e.g. use "submissions", declare django models, etc.). It's also unclear which parts of the emergent API "should" be used and which should not.

   The frontend (JavaScript) runtime also has a vague API - for example, the studio_view passes a certain parameter to the initialization function as a jQuery element, while the author_view passes it as a DOM element, which is completely different. (This inconsistency is the sort of problem that TypeScript was created to solve.) Many XBlocks assume that specific JavaScript libraries (like jQuery) are available, but don't explicitly declare that dependency.

   Consequences:
   a. It's hard to know what modifications to the platform will cause issues with existing XBlocks, because they aren't limited to a well-defined interface. This may slow overall platform development.
   b. The LMS must continue to load and include a whole bunch of huge JavaScript files on every page, even if the LMS is no longer using them, because some XBlock might need them. This is especially problematic when trying to render many separate XBlocks on a page, each in its own iframe.

5. **Specific to the Open edX LMS, but also LMS-agnostic**
   The XBlock API was originally made with the intention that XBlocks would be used both in Open edX and in some other learning platform(s) - specifically Google Course Builder; however, that never really happened. In practice, the fact that XBlocks can only possibly work in python-based LMSs fundamentally limits their applicability, and the fact that XBlocks often depend on edx-specific APIs and conventions mean that they are not portable at all.

   Consequences:
   a. The core XBlock API and documentation is unnecessarily abstract (LMS agnostic), and it can be hard for XBlock authors to figure out how to do common things like "issue a score."

6. **No offline support**
   XBlocks fundamentally cannot run offline. Some workarounds are in place to allow downloading some content (like videos) in the edX mobile app, but XBlocks in general cannot work offline. Interactive XBlocks (that save user state or earn a score) in

particular do not work at all, because there is no mechanism to sync with the server.

Consequences:

a. People who live without reliable access to the internet may be unable to use Open edX.

b. People cannot use Open edX to learn when commuting on underground railways or airplanes that lack a phone/Wi-Fi signal.

7. **Arbitrary Composition**
   XBlocks can be "composed" by grouping several into an arbitrarily complex tree structure: (a Sequential XBlock contains a Vertical XBlock which contains a Problem Builder XBlock which contains a Short answer XBlock…)

   Consequences:

   a. XBlocks play both a structural role and a content role.
   b. Determining the overall structure and metadata of a course (defined by XBlocks) is so inefficient that several layers of workarounds and caching (CourseOverview, block transformers, learning sequences API) have been developed to get the course outline efficiently.
   c. The XBlock runtime has a lot of complexity (due to the need to support arbitrary child XBlocks, though this functionality is not often used for content purposes).

Goals

Though I was previously thinking about a successor to XBlocks ("NeXBlocks"), I now have a different set of goals in mind:

Priority goals:

- **Make an incremental improvement to the XBlock architecture that will allow us to "upgrade" existing XBlocks to address some of these problems *without* a major rewrite**
- **Focus for now on improving rendering performance, to facilitate the full launch of MFEs, and the removal of LMS static assets / legacy UI.**
  - This addresses https://openedx.atlassian.net/wiki/spaces/OEPM/pages/4205019182/Making+XBlock+Preview+Fast in particular.
- **Maintain OLX compatibility backwards and forwards**
- Non-composable: every XBlock is a leaf node in the content tree; it cannot have children and does not have a "parent" XBlock. Features like randomization, cohorting, and adaptive learning should be implemented as part of the "course outline" itself, not by composing XBlocks.
- Compatible with Learning Core

Proposal Details

Fundamentally, I am now proposing we define a new “v2” XBlock standard that will address some of the issues listed above, while keeping OLX compatibility and with a clear migration path. XBlocks can “opt in” to the new v2 API on a per-block basis (e.g. the “video” block can be upgraded to v2 while “problem” can remain on v1), but only the upgraded blocks will benefit from the new APIs and performance.

At the same time, a “v2” standard that is an incremental change to the existing APIs won’t be able to solve all or even most of the problems above. That’s why I later want to propose a **ScriptBlock** that implements the v2 API but allows authors to define new block types on a per-course basis, in sandboxed JavaScript.

XBlock API v2

To start with, v2 XBlocks will be much the same as existing XBlocks: written in python, defined by a class that inherits from XBlock, defined by entry points, etc. XBlocks that are upgraded to v2 will declare that with in some statically analyzable way - either by subclassing “XBlock2” (if they don’t need backwards compatibility) or by mixing in the XBlock2Mixin (if they are 3rd party XBlocks that still want to be installable on pre-Sumac instances).

When opting in to the “v2” API:

1. **XBlock “view” functions like “student_view”, “author_view”, “studio_view”, “problem_view” etc. that return HTML are not used for v2 XBlocks. Instead, XBlocks define a learner_view_data() function which returns JSON** at the moment the learner views the block. The default implementation is a no-op and many XBlocks would be able to just use that.
2. XBlocks cannot access “parent” nor “children”
3. In development mode, something like an import linter or modulefinder is used to check if the XBlock is importing any edx-platform APIs that aren’t part of the formal XBlock API. If so, the XBlock class won’t be loaded, and a warning displayed.
4. **XBlock client-side code must be self-contained ES modules (ESM)** that don’t assume the presence of RequireJs, React, jQuery, MathJax, or the presence of any other “global” JS/CSS. However, some standard shared libraries will be pre-defined and available for import as needed - preact, mathjax, possibly others.
5. **The XBlock learner UI for v2 XBlocks is a web component**, which is *not* rendered in an iframe but which *may* choose to be isolated from the surrounding CSS and JS context.

    - Iframes add a lot of complexity that we don't need; we assume the XBlock authors are trusted. If we later design a "ScriptBlock" that allows authors to define new blocks using JavaScript, it would probably need to run in an iframe.
6. The client (JavaScript) runtime cannot call handlerUrl() anymore, but only the new callHandler() method.
7. The .save() method is prohibited. Field values can only be modified during an @XBlock.json_handler method or during the learner_view_data() function; otherwise field data is read-only.
    - This serves two purposes: (1) simplify and optimize the runtime by reducing the number of places where XBlock field data gets written down to two, and (2) facilitates automagic updating of field data on the frontend.
8. Field values can be optionally marked as "private", in which case they will not be sent to the frontend/browser.

Here's how this speeds up rendering:

1. Instead, the backend does a couple database queries to load all the field data for all of the XBlocks in the unit at once, then uses the static XBlock definitions to remove any fields marked as "private" and validate/coerce the datatypes. For blocks that implement learner_view_data(), the runtime is fully loaded and that function is also called to do additional data updates/calculations/retrievals. Both the field data and any learner_view_data is passed to the frontend JavaScript code. No HTML is generated on the backend.
2. The frontend gets the (non-private) field data as a JSON object, along with any learner_view_data.
3. For each XBlock type on the page, the frontend loads its JavaScript implementation and renders the XBlock as a Web Component (custom element). This should not require any additional network requests, if that block type has been seen by the user before.
4. **No JavaScript/CSS is loaded other than what the XBlock explicitly depends on.**

What would that look like?

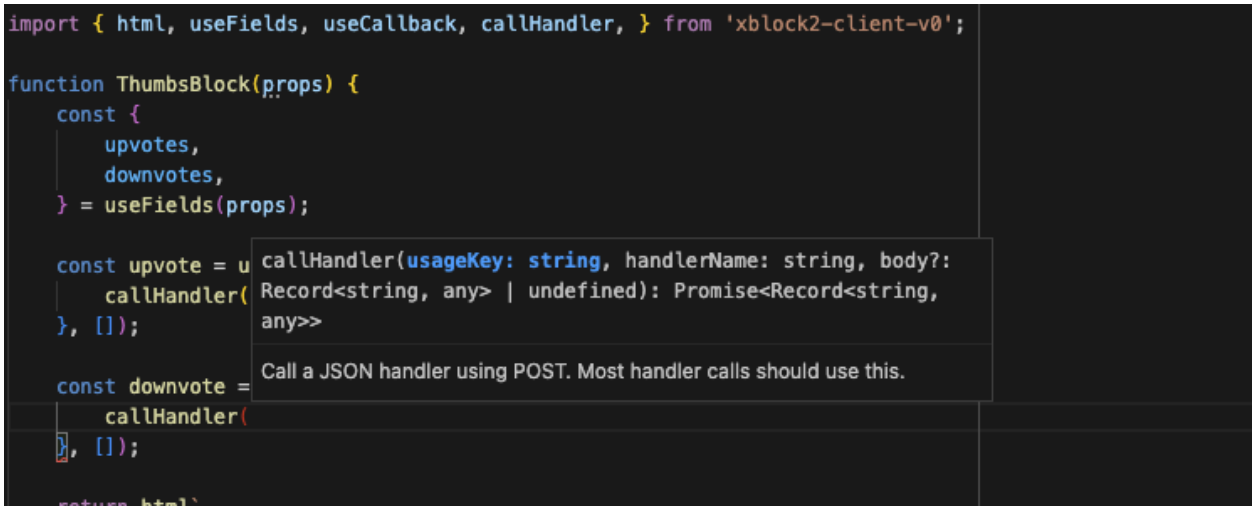
Here's a video showing how v2 Blocks render much faster:
2023 XBV2 RenderingSpeed.mov

Here's what it looks like to convert a *simple* XBlock from v1 to v2:
https://github.com/open-craft/xblock-thumbs/pull/1/files
Note that converting a more complex block like video or problem is much, much, much more complex due to the messy UI code they have.

Here's the proof of concept code I put together:

edx-platform#34887 - platform changes to create a new fast “render unit” API
XBlock#755 - changes to the XBlock repo itself to define the XBlock2 class/mixin.
frontend-app-learning#1402 - changes to the Learning MFE

XBlock UI code can be type-checked, and the JavaScript API is strictly defined. When writing frontend code in an IDE, it will be aware of the API (TypeScript):



XBlock UI can choose to be isolated from the platform CSS or not
As implemented in the demos above, the “thumbs” XBlock renders in a shadow DOM (isolated from the CSS environment and DOM tree of the parent page), whereas the “html” XBlock opts out of isolation, so that the platform styles will apply to its content. Neither use an iframe. Note this is not a security feature and doesn’t provide a true sandbox.

XBlock UI code can be written in a modern, declarative style. When XBlock fields are changed during a handler, the component is automatically re-rendered.

XBlock UI code runs in isolated modules (ESM); nothing gets added to the global scope unless you really try to do so by accessing ‘window’.

Most XBlocks will only depend on their own JS file (a few KB except for complex UIs like video) and the shared “xblock-client-v0” JavaScript runtime (19KB minified).

What are some challenges with this approach?

1) Some blocks, like HTML and “Thumbs” are easy to port. Even capa should be relatively easy if we don’t try to actually speed up its rendering and just use learner_view_data() to ship some HTML to the frontend. However, other **blocks like video, drag-and-drop-v2, ora2, etc. would**

essentially require a complete rewrite of their frontend code to benefit from this new API. Personally, I think the video block needs that anyways.

2) Supporting both “v1” and “v2” blocks on the backend increases complexity. The system as a whole won’t benefit from simplification until v1 support is no longer required on the backend. However, it might be possible to solve this in the future by moving all v1 support code (especially frontend) to a separate repo/shim.

3) My attempt to remove the hierarchy completely may make it harder to deal with randomized content block, especially on the Studio side. I think it’s possible to work around this in the LMS by using block transformers to give learners the view of blocks they would see without actually having to load the randomized content block runtime. On the backend it may not be possible though.

4) Paragon wouldn’t be available in the slim preact-based client library. We could offer an alternative version that includes React + Paragon, but it would be a huge dependency and Paragon is still changing it’s API frequently.

Other problems I noticed while researching this

1. The Learning MFE has a *ton* of console warnings and errors.
2. To get the list of units in a sequence, the Learning MFE calls the **/api/courseware/sequence/:sequenceUsageKey** API, which renders the student_view of all the units in the sequence (get_metadata -> _get_render_metadata -> _render_student_view_for_blocks) and each unit then tries to pick an icon for itself by loading all the XBlocks in the unit, which includes bind_for_student() calling save() on every block in every unit in the sequence; though the save() should be a no-op, this seems very inefficient.

IFrame Pros and Cons

- *If* the XBlock frontend code is all written using React or Preact, there is generally no security concerns around user-generated content (including authored content) except in the places where XBlock authors explicitly use the **dangerouslySetInnerHTML** to render user-provided HTML directly. In such cases, they can and should put it in an <iframe> to isolate it, if that can be done without breaking compatibility.
- XBlocks which aren’t using React/Preact should be rendered inside an iframe for backwards compatibility and security.

Cons of iframes generally:

- Resizing responsively isn't perfect; can create jumps during rendering and "scroll traps" on the page.
- Rendering is generally slower as the content in the iframe (and all its dependencies) doesn't start loading until the iframe itself has been rendered.
- Iframes break copy-paste. If you select text outside of the iframe and inside of the iframe then paste into a text editor, only the text from outside the frame will be pasted. (try it)
- Makes the implementation more complicated (postmessage and friends are required, plus resizing code, etc.)

Pros of iframes generally:

- *If* the iframe is on a different domain and the sandbox features are turned on, it provides the strongest level of protection against authors injecting malicious JavaScript code into the page, either accidentally or on purpose. It doesn't depend on XBlock authors doing the right thing.