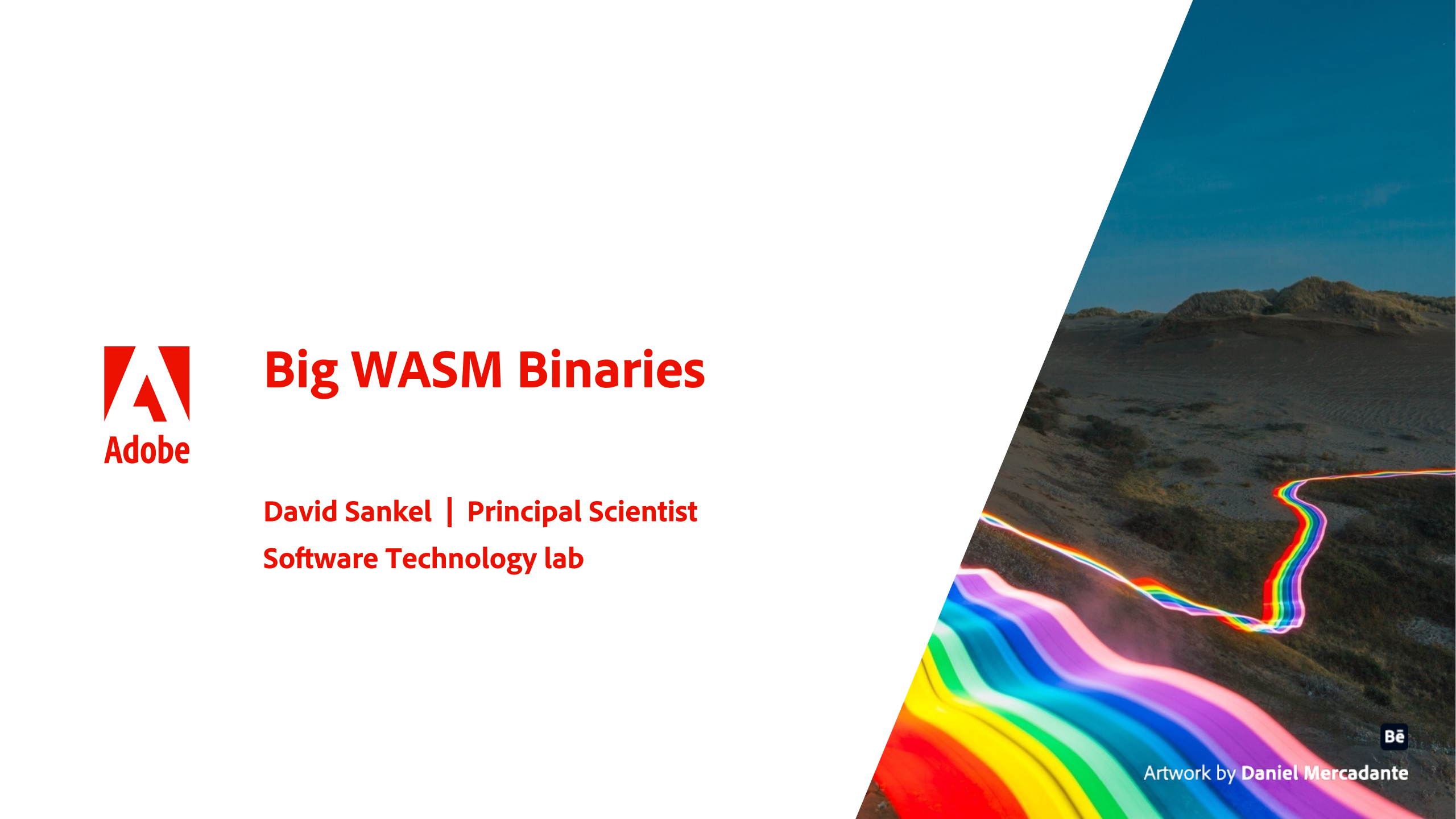
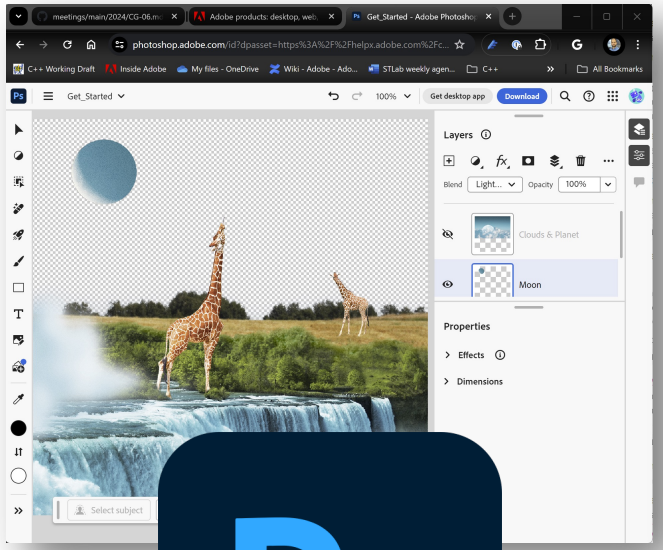
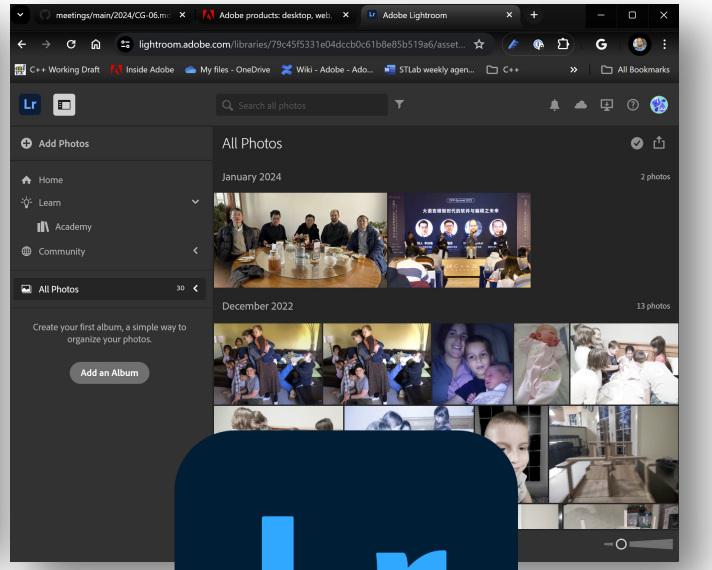

Adobe
Big WASM Binaries
David Sankel | Principal Scientist
Software Technology lab
Bē
Artwork by Daniel Mercadante

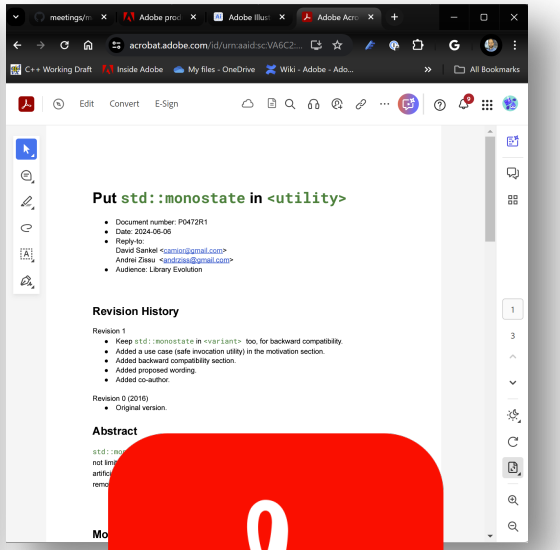
Photoshop



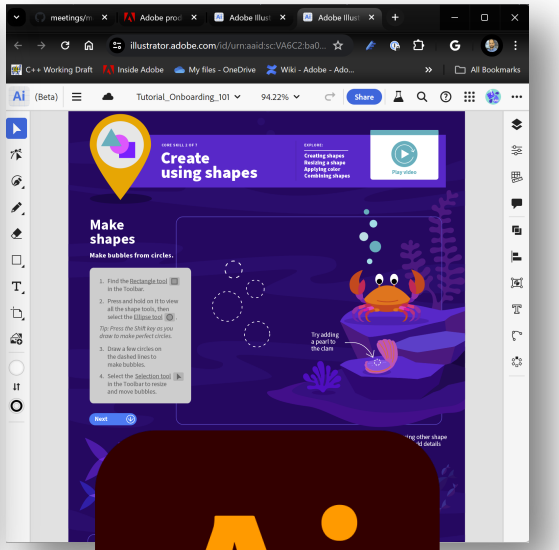
Lightroom

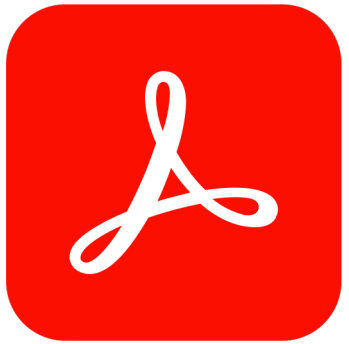


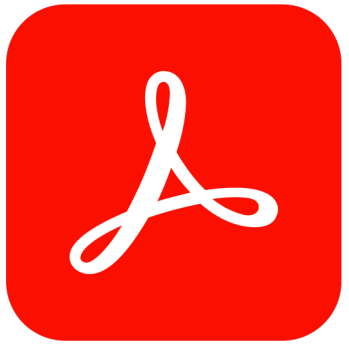
Acrobat



Illustrator





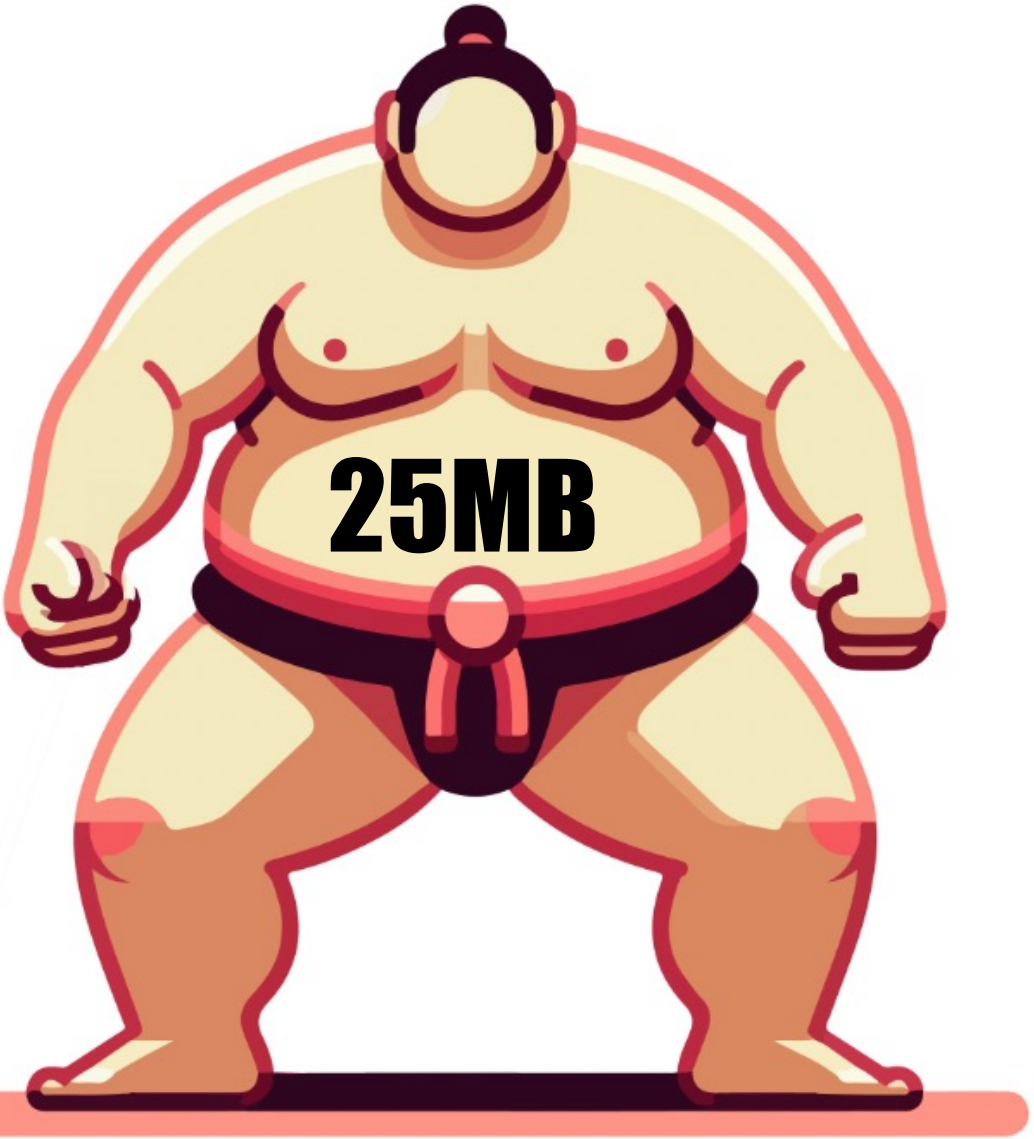




We put the *Web* back into *Web* Assembly

Our formula

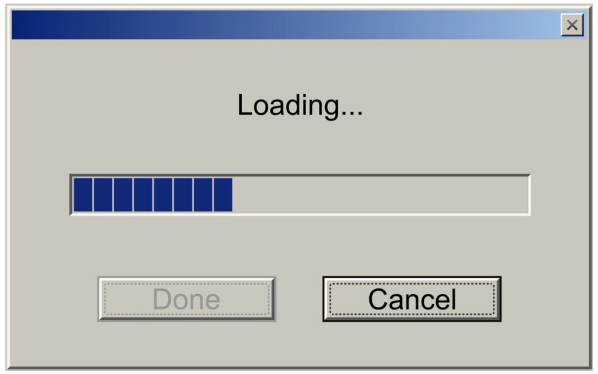
1. Collect the source code for our flagship applications
2. Compile to WASM and deploy to the Web
3. ???
4. Profit!!!

We’ve got a problem



Big WASM binaries

Well...



Long loading time

Implications…

- Users will either give up after waiting or not return due to bad experience
- We're missing out on providing value to our users

How do you get the "instant web" experience for 2GB+ desktop applications?

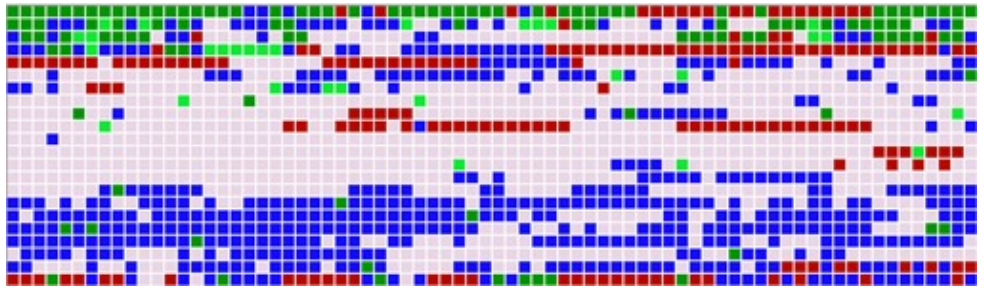
Three observations

- Hypothesis: most WASM code is never used
  - 30+ years of accumulation
  - A small fraction of capabilities is exposed
- We can collect massive usage data
  - Action recording
  - Images used
- Code analysis and transformation tooling exists
  - wasm-split and conventional code coverage profiling
  - Clang and Wasm-based code transformation libraries

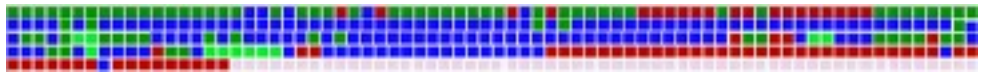
Basic idea: reduce the Wasm binary size

Step 1: Collect massive usage data in a way that can be replayed

Step 2: Replay the actions on an instrumented build that tracks code usage



Step 3: Use profile to automatically remove or sidecar unused code



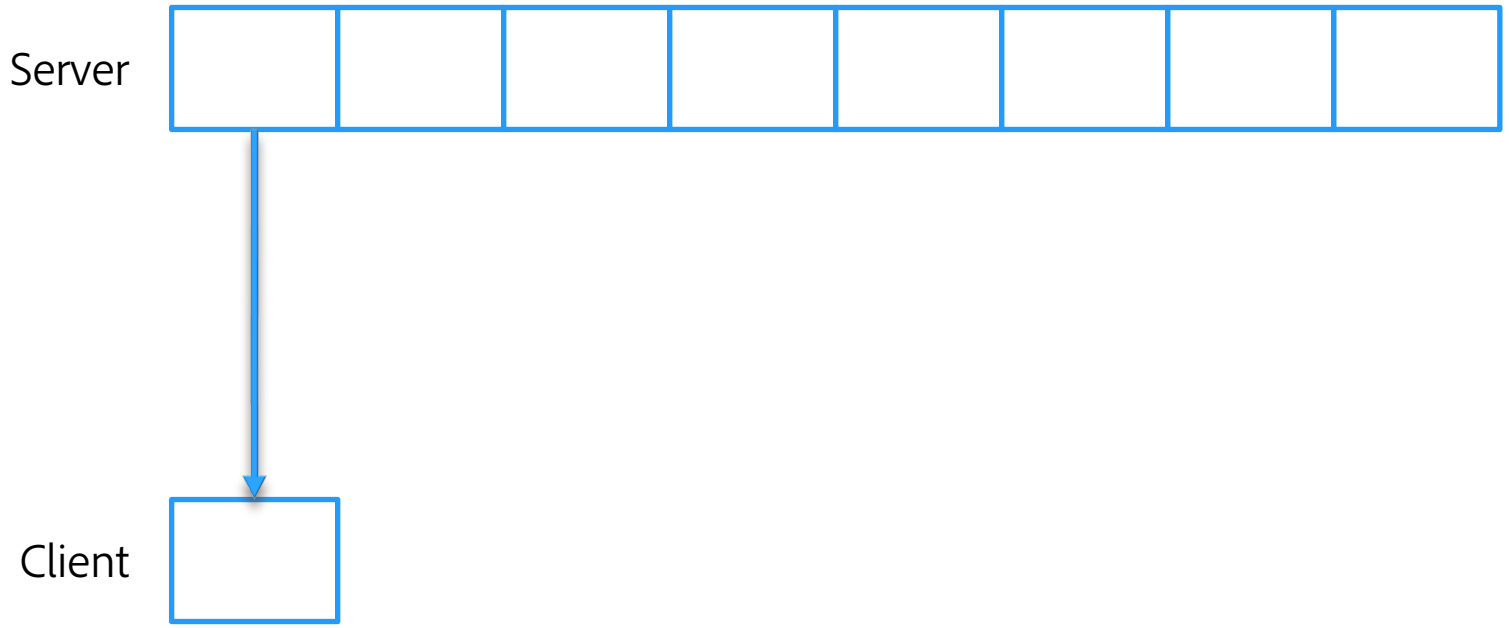
That's great, but…

- Splitting a binary in two is very coarse grained
- Function calls between binaries are heavy
- Threads cause problems
- It's intrusive

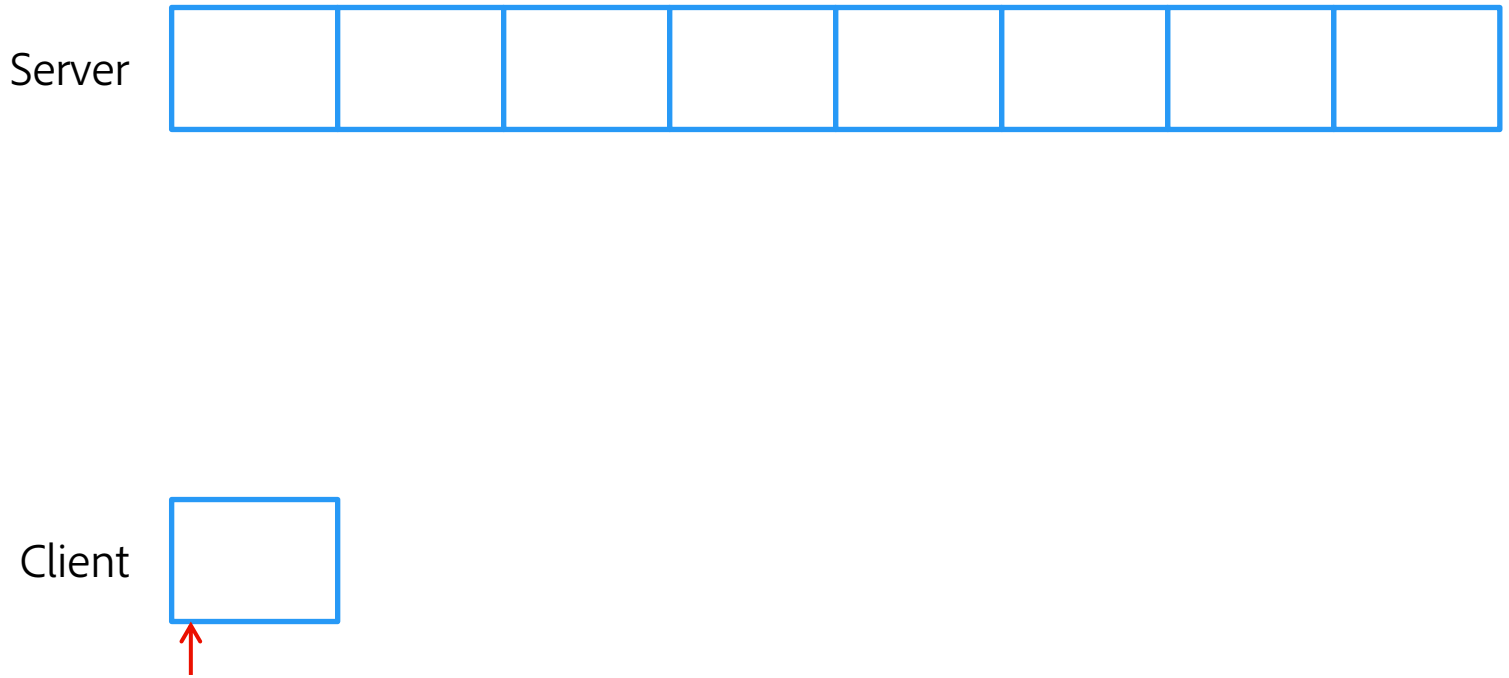
What about virtual memory?

Client

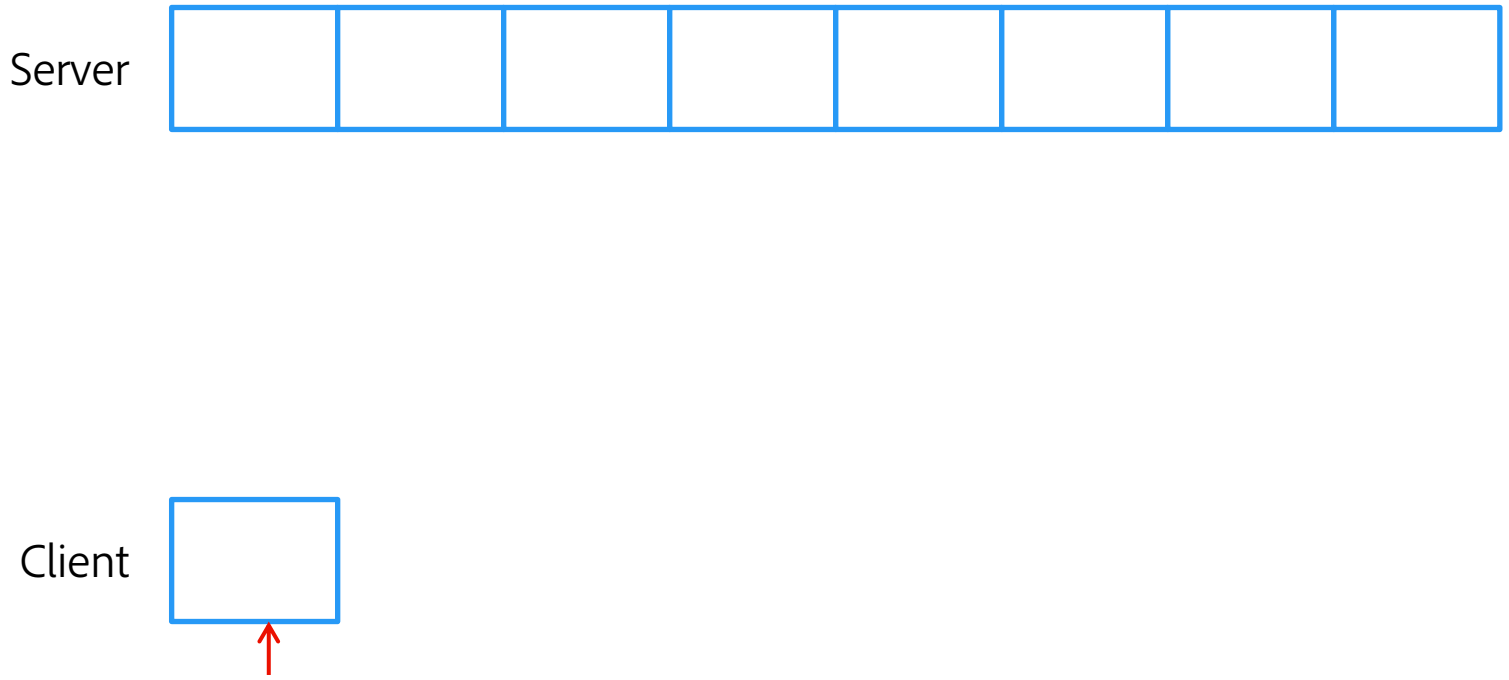
What about virtual memory?



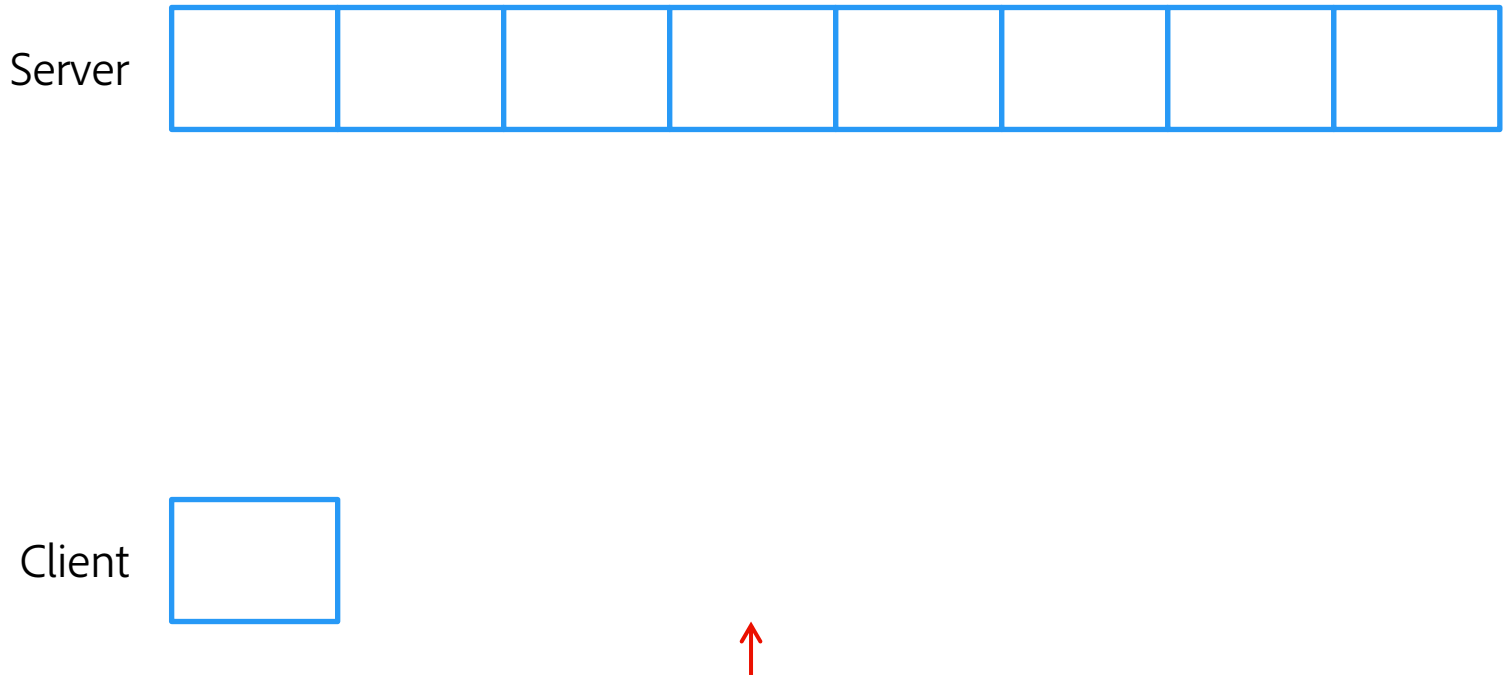
What about virtual memory?



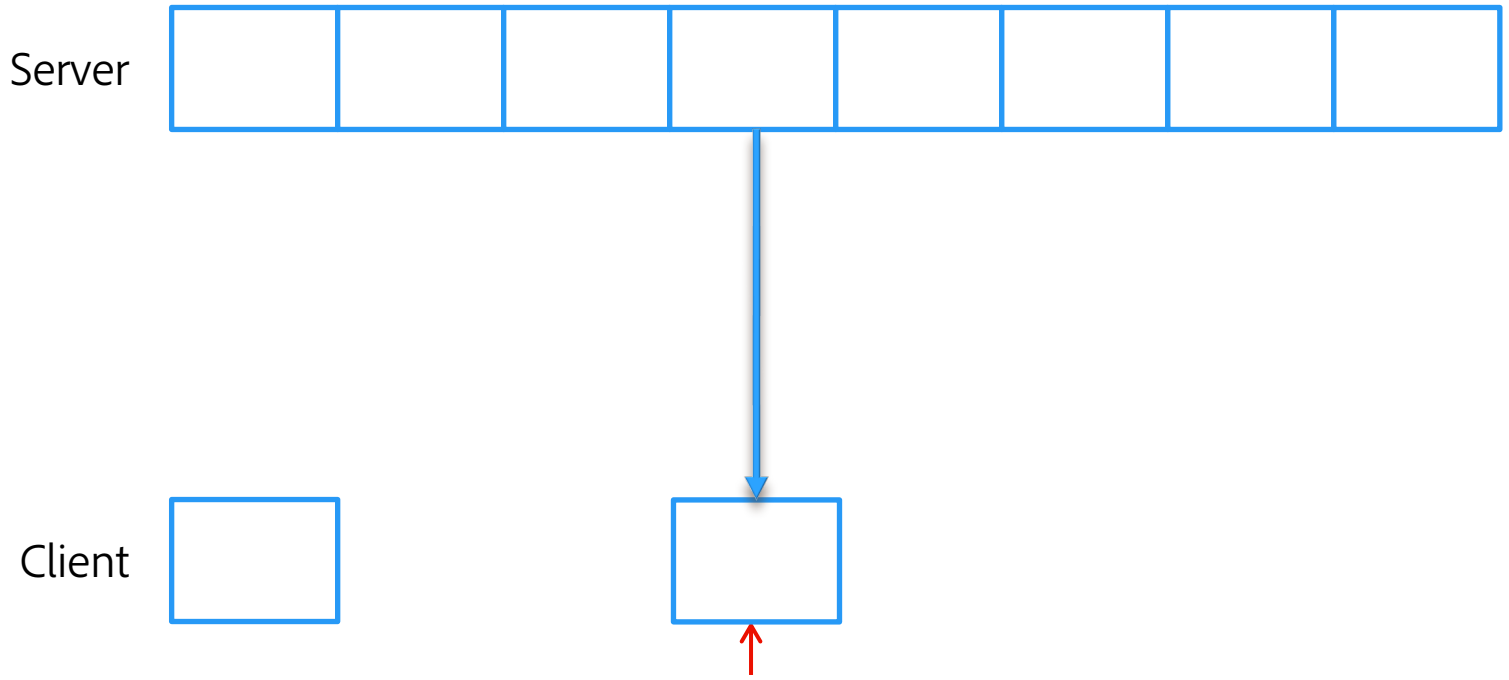
What about virtual memory?



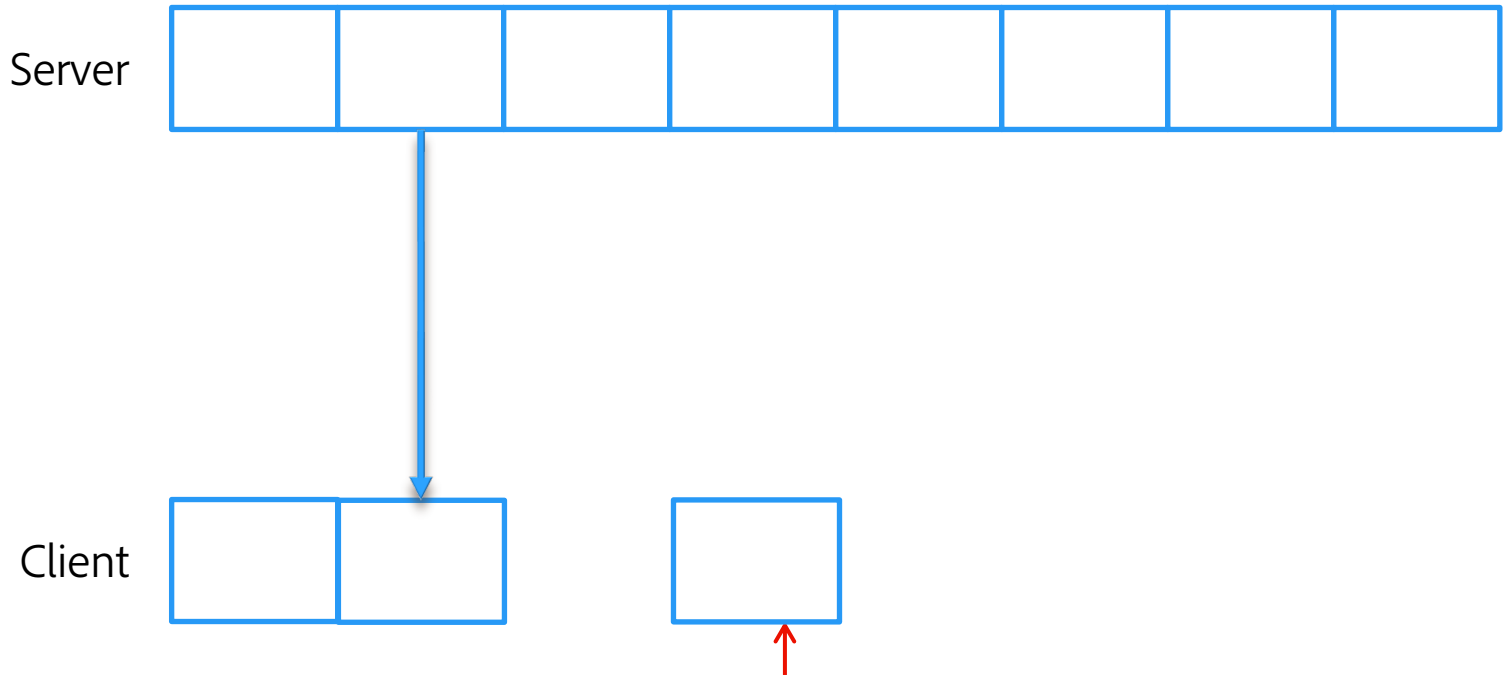
What about virtual memory?



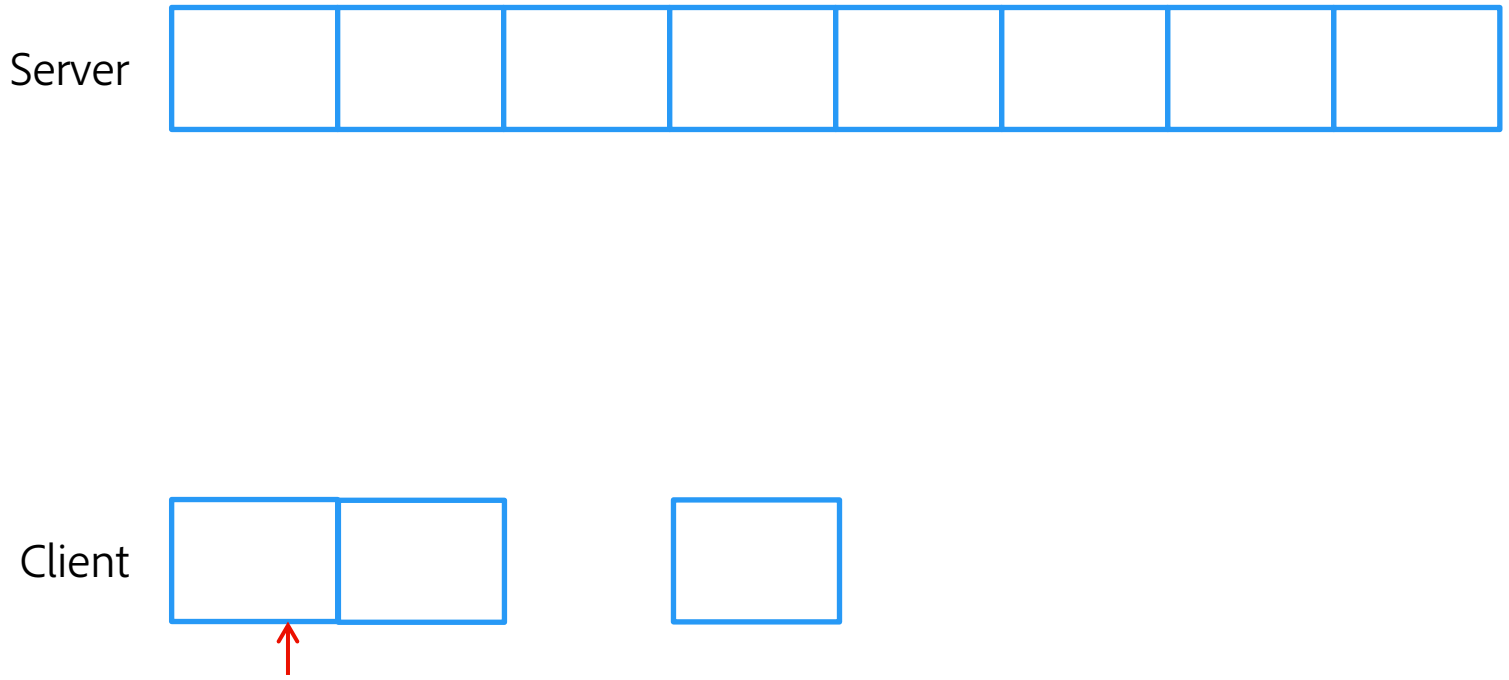
What about virtual memory?



What about virtual memory?



What about virtual memory?



WASM virtual memory for instructions

- Non-intrusive
  - WASM code doesn't change, just how it is executed
  - Transparent to threads
- Fine grained
- Adaptive
  - Fetch based on usage pattern
  - Pre-fetch based on profile-based optimizations

How can we make this a reality?

(I have no idea)

Please help


Adobe
Bē
Artwork by Daniel Mercadante