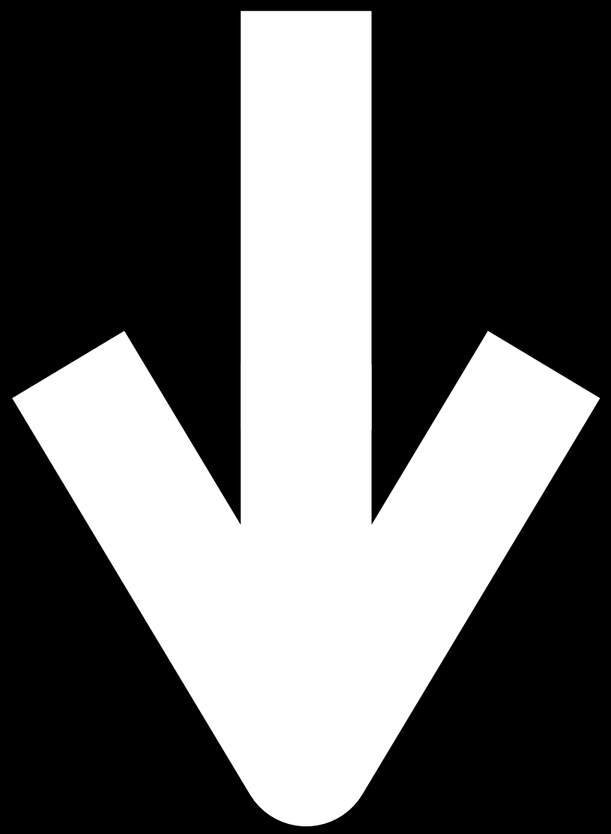
Server Sent Events



By Kornel Lesiński (@pornelski) for London JS, July 2011, http://lanyrd.com/2011/ldnjs03/

XHR

Everyone uses XML HTTP Request. AJAX is in every developers' toolbox. And recently, WHATWG has re-invented TCP/IP with WebSockets.

XHR

Web-
Sockets

And a question has arisen: is there a room…

XHR

?

Web-Sockets

…for a third category of API in the middle – something between XHR and WebSockets? And browser vendors pondered this for years as well. The bar is really high. In order to create a new category of API, it has to be far better at some key things: it has to be very easy to use. It has to have great security. It must be compatible with browsers, servers and proxies.

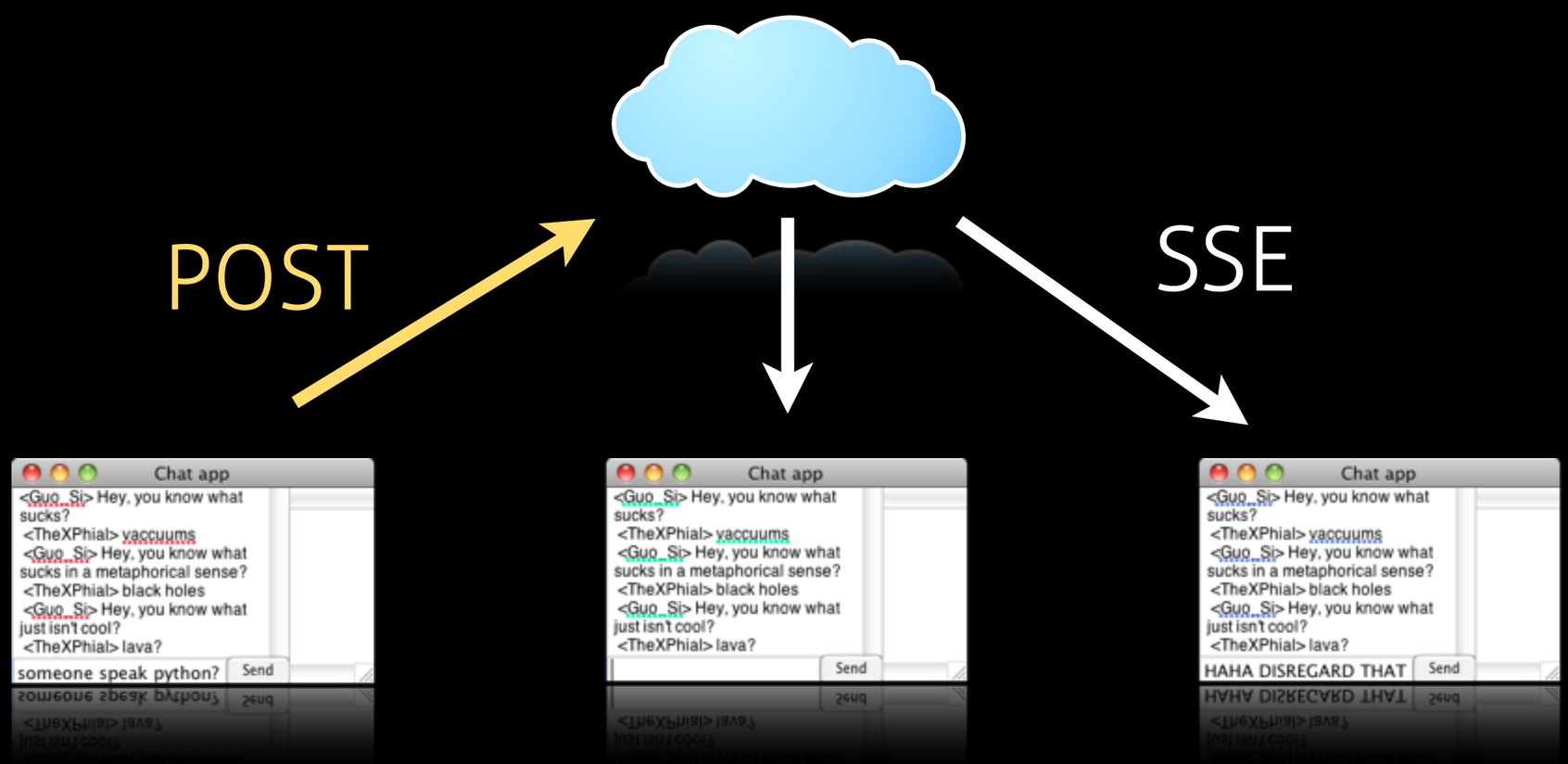
XHR

SSE

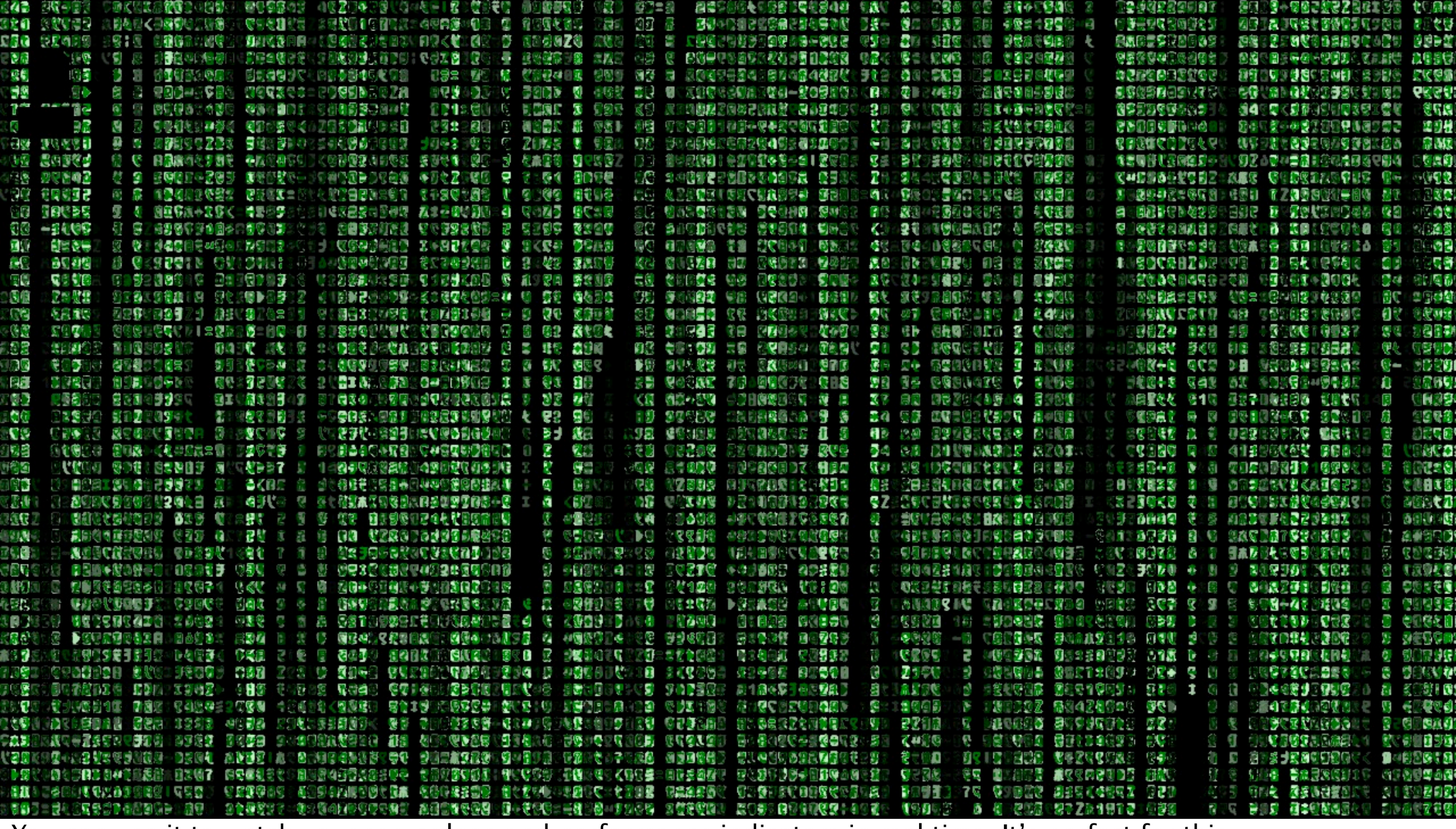
Web-Sockets

And I think I've got something that is. And I'd like to show it to you to day. It's called the iPad^W^WServer Sent Events. It enables you to do fantastic things

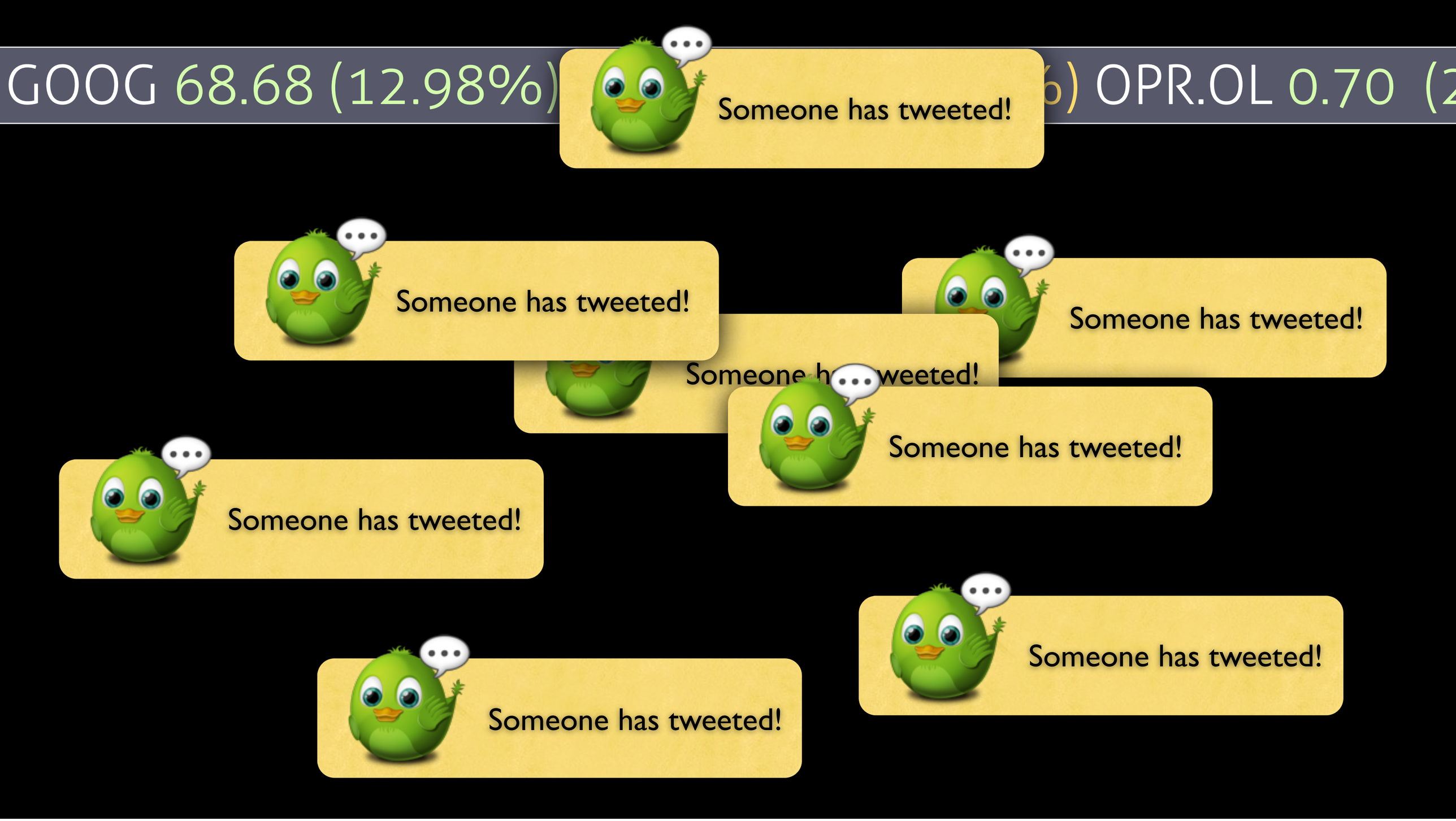
Real-time chat



You can create real-time chat application. A posted message can be pushed to everyone in the chat, instantly.



You can use it to watch your server logs and performance indicators in real time. It’s perfect for this.



You can do boring stuff, like up-to-date stock tickers. And you can do more interesting stuff, like instant notifications on your pages whenever a message is posted (as soon as sender presses Send). API for all of this is very easy:

Very simple API

```
var source = new EventSource("url");

source.onmessage = function(event) {
  alert(event.data);
}
```

You create new event source, attach your event handler and read the data. That’s it!

- Automatically reconnects/resumes
- Reuses connections to the same URL
- Avoids blocking other HTTP traffic

The important thing is all the things you DON’T have to do. You don’t have to handle errors! Browser will the keep connection alive for you. You don’t have to manage resources. If you create lots of EventSource objects, browser is allowed to open just one connection and distribute events to all objects.
Browser knows that these are special long-lived connections and will ensure they’re handled properly and don’t hit limits or block queues for other resources.

text/event-stream

```
data: Hello World!\n
\n
```

How does the stream look like? Couldn't be simpler. It's a text file! Lines prefixed data: set text to send (no length limit), and empty line fires the event.

The protocol can do a bit more, but you don't need to read a 200-page RFC.

```
data: Hello
data: World!
id: 123
```

Last-Event-Id: 123

ms

```
retry: 1000

event: bark
data: Woof!

: comment
```

The whole thing fits on one slide. You don't have protocol switching, framing, packet fragmentation or masking. To send multiline data, prefix each line with data:. If you set ID, browser will send it back to you when it reconnects. This way you can avoid sending same data twice (e.g. send scrollback in a chat app to new users only). Retry lets you control how long to wait before reconnect. You can name events and use addEventListener('name') to avoid huge onmessage handler. Comments are for debug and heartbeat.

- Works with regular web servers
- Works with HTTP proxies
- Supports long polling (Comet)

SSE is just a download of a text file. You can use any server that can slowly generate a text file. You don't need special servers and complicated libraries for it!
Since it's regular HTTP, it works with all proxies just fine. Even connection drops are not a problem, because it can gracefully reconnect.

```
res.writeHead(200, {"Content-Type":
                    "text/event-stream"});

res.write("data: " +
          JSON.stringify(stuff) +
          "\n\n");
res.flush();
```

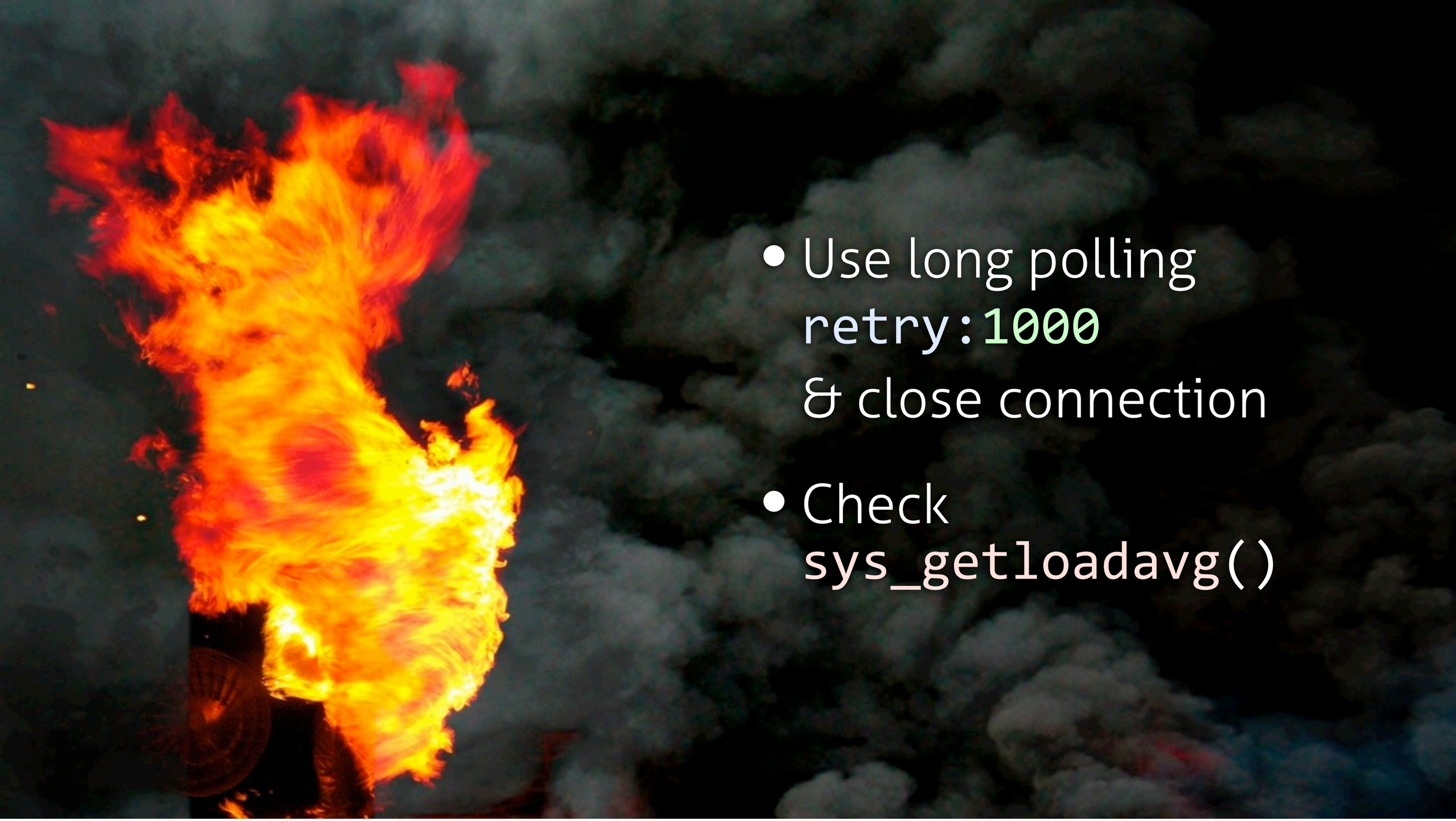
On the server it is very simple as well. Set content type and print lines. Note that I'm not including any special library and I'm not doing any handshake.
Bonus tip here is to use JSON.stringify and JSON.parse, so you can use JS end-to-end. Flush is needed to actually send data, instead of it sitting somewhere in buffers.

```php
header("Content-Type: text/event-stream");

do {
 echo "data :".json_encode($stuff)."\n\n";
 flush();
 sleep(1);
} while(!connection_aborted());
```

If you don’t have node.js I’m not sure what you’re doing on londonjs meetup, but you can generate event stream even with clunky old PHP. You might be thinking – if I unleash this on millions of visitors, is my server going to look like…

is my server going to look like this? Yes, yes it will! Two ways around it: limit number of users. Use it for yourself in admin, give it to few special users. Or, if you're brave, you can do it on larger scale:



Instead of keeping connections open all the time, just close them from time to time. Browser will reconnect without fuss. You control length of connection and control delay between reconnection, so you can have full spectrum between persistent connection and polling. You can even be smart about this and watch server load – if it's low, then use persistent, when it's too high, switch to polling and increase back-off time.
So, SSE can work with most servers. What about browsers?

11	6	5	4	6
(9)				beta

Peachy! Firefox has been lagging behind, but finally SSE is almost there.

There is something missing here, isn't it? A certain browser built-in into an outdated OS of an American corporation…

Android of course! The other one doesn’t work either. But fear not! Fallback is easy. Since SSE is just a text file, syntax is trivial to parse and it can fall back to polling, you can read it with XHR. It’s easy to write yourself, you can find polyfill for this too. I’m pretty sure that Pusher guys have infiltrated the room by now, and they have dealt with some server and client problems. Speaking of problems, there are a couple:

- Same-origin limitation
  (even port has to be the same!)
- CORS in the future
- No binary data (UTF-8 only)

You must use host and port for serving the page itself and the event stream. It's annoying if you want to use Node.js for the stream and have something else for regular pages. You can work around it by rewriting+proxying just the stream URL with nginx or Varnish (this is another place where proxy compatibility pays off!)

History

First version was on Hixie's blog in 2004 in response to "wouldn't it be cool if server could fire any event?" And it was really any event—you could remotely control page by firing mouse clicks and keyboard events!

<event-source>

You could just hook stream with <event-source> element and the server could play with the page. Unsurprisingly, it turned out to be difficult implement. Opera released first implementation in 2006, but Firefox had a patch for it a bit earlier. However, it took years for the patch to be reviewed, fixed, reviewed, updated, etc. In 2009 the spec has been simplified to only allow custom events, having element for JS-only API felt stupid. This broke Firefox's patch. Updated patch missed Fx4 deadline. Fx5 wasn't making big changes. Finally Fx6 has it. I'd go mad if my patch took so long to land, but Wellington Fernando de Macedo—who wrote most of it—in the meantime wrote WebSockets patch too!

Future

- SMS
- Push notifications

Mozilla hesitated adding SSE, which is a bit redundant with XHR multipart/replace hack and WebSockets, but eventually decided to do it, hoping for the future: SSE is protocol-agnostic. It doesn't have to work over HTTP. It might work over SMS or Apple-style push notifications. An incoming event could launch Firefox mobile, which would open appropriate tab and fire event there—a real push! It's still far off thought, there isn't even spec for it yet.
In short term, EventSource is coming to SharedWorkers, so you'll be able to e.g. fire notification only in one tab that user is looking at.

http://pornel.net/sse

- http://www.flickr.com/photos/gauri_lama/2672901420/
- http://www.flickr.com/photos/wheatfields/116783486/
- http://arrioch.deviantart.com/