I am doing HTTP wrong

— a presentation by Armin Ronacher

@mitsuhiko

FIRETEAM™

THE WEB DEVELOPER'S EVOLUTION

echo

```
request.send_header(…)
request.end_headers()
request.write(…)
```

```
return Response(…)
```

Why Stop there?

What do we love about HTTP?

Text Based

REST

Cacheable

Content Negotiation

Well Supported

Works where TCP doesn't

Somewhat Simple

Upgrades to custom protocols

Why does my application look like HTTP?

everybody does it

Natural Conclusion

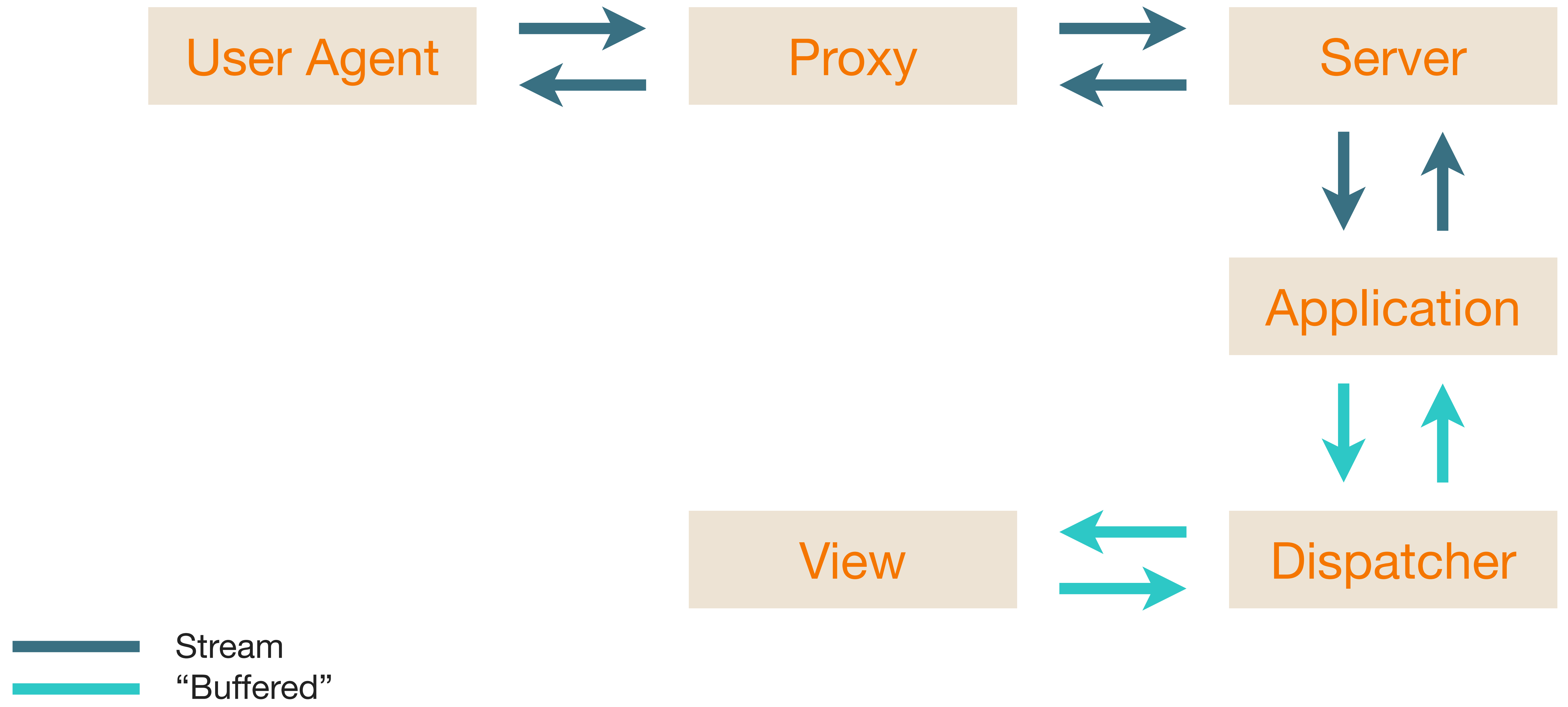
we can do better!

we're a level too low

Streaming: one piece at the time, constant memory usage, no seeking.

Buffering: have some data in memory, variable memory usage, seeking.

TYPICAL REQUEST / RESPONSE CYCLE



IN PYTHON TERMS

```python
def application(environ, start_response):

    # Step 1: acquire data
    data = environ['wsgi.input'].read(...)

    # Step 2: process data
    response = process_data(data)

    # Step 3: respond
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [response]
```

ONE LEVEL UP

```
s = socket.accept()
f = s.makefile('rb')
requestline = f.readline()
headers = []
while 1:
    headerline = f.readline()
    if headerline == '\r\n':
        break
    headers.append(headerline)
```

WEIRD MIXTURE ON THE APP

```
request.headers                <- buffered
request.form                   <- buffered
request.files                  <- buffered to disk
request.body                   <- streamed
```

HTTP'S LIMITED SIGNALLING

Strict Request / Response

The only communication during request from the server to the client is closing the connection once you started accepting the body.

BAILING OUT EARLY

```
def application(request):
    # At this point, headers are parsed, everything else
    # is not parsed yet.
    if request.content_length > TWO_MEGABYTES:
        return error_response()

    ...
```

BAILING OUT A LITTLE BIT LATER

```
def application(request):
    # Read a little bit of data
    request.input.read(4096)

    # You just committed to accepting data, now you have to
    # read everything or the browser will be very unhappy and
    # Just time out.  No more responding with 413
    ...
```

REJECTING

Form fields -> memory
File uploads -> disk

What's your limit? 16MB in total? All could go to memory. Reject file sizes individually?
Needs overall check as well!

THE CONSEQUENCES

How much data do you accept?
Limit the overall request size?
Not helpful because all of it could be in-memory

IT'S NOT JUST LIMITING

Consider a layered system
How many of you write code that streams?

What happens if you pass streamed data through your layers?

A new approach

Dynamic typing made us lazy

we're trying to solve both use cases in one
we're not supporting either well

HOW WE DO IT

Hide HTTP from the apps
HTTP is an implementation detail

PSEUDOCODE

```
user_pagination = make_pagination_schema(User)

@export(
    specs=[('page', types.Int32()),
           ('per_page', types.Int32())],
    returns=user_pagination,
    semantics='select',
    http_path='/users/'
)
def list_users(page, per_page):
    users = User.query.paginate(page, per_page)
    return users.to_dict()
```

TYPES ARE SPECIFIC

```
user_type = types.Object([
    ('username', types.String(30)),
    ('email', types.Optional(types.String(250))),
    ('password_hash', types.String(250)),
    ('is_active', types.Boolean()),
    ('registration_date', types.DateTime())
])
```

WHY?

Support for different input/output formats
keyless transport
support for non-HTTP
no hash collision attacks :-)
Predictable memory usage

COMES FOR FREE

Easier to test
Helps documenting the public APIs
Catches common errors early
Handle errors without invoking code
Predictable dictionary ordering

Strict vs Lenient

RULE OF THUMB

Be strict in what you send,
but generous in what you receive
— variant of Postel's Law

BEING GENEROUS

In order to be generous you need to know what to receive.

Just accepting any input is a security disaster waiting to happen.

SUPPORT UNSUPPORTED TYPES

```
{
  "foo": [1, 2, 3],
  "bar": {"key": "value"},
  "now": "Thu, 10 May 2012 14:16:09 GMT"
}

foo.0=1&
foo.1=2&
foo.2=3&
bar.key=value&
now=Thu%2C%2010%20May%202012%2014:16:09%20GMT
```

SOLVES THE GET ISSUE

GET has no body
parameters have to be URL encoded
inconsistency with JSON post requests

Where is the streaming?

THERE IS NONE

there are always two sides to an API

If the server has streaming endpoints —
the client will have to support them as well

For things that ***need*** actual streaming we have separate endpoints.

streaming is different

but we can stream until we need buffering

DISCARD USELESS STUFF

```
{
  "foo": [list, of, thousands, of, items, we don't, need],
  "an_important_key": "we're actually interested in"
}
```

What if I don't make an API?

modern web apps **are** APIs

Dumb client?

Move the client to the server

Q&A

FIRETEAM™

Oh hai. We're hiring

http://fireteam.net/careers