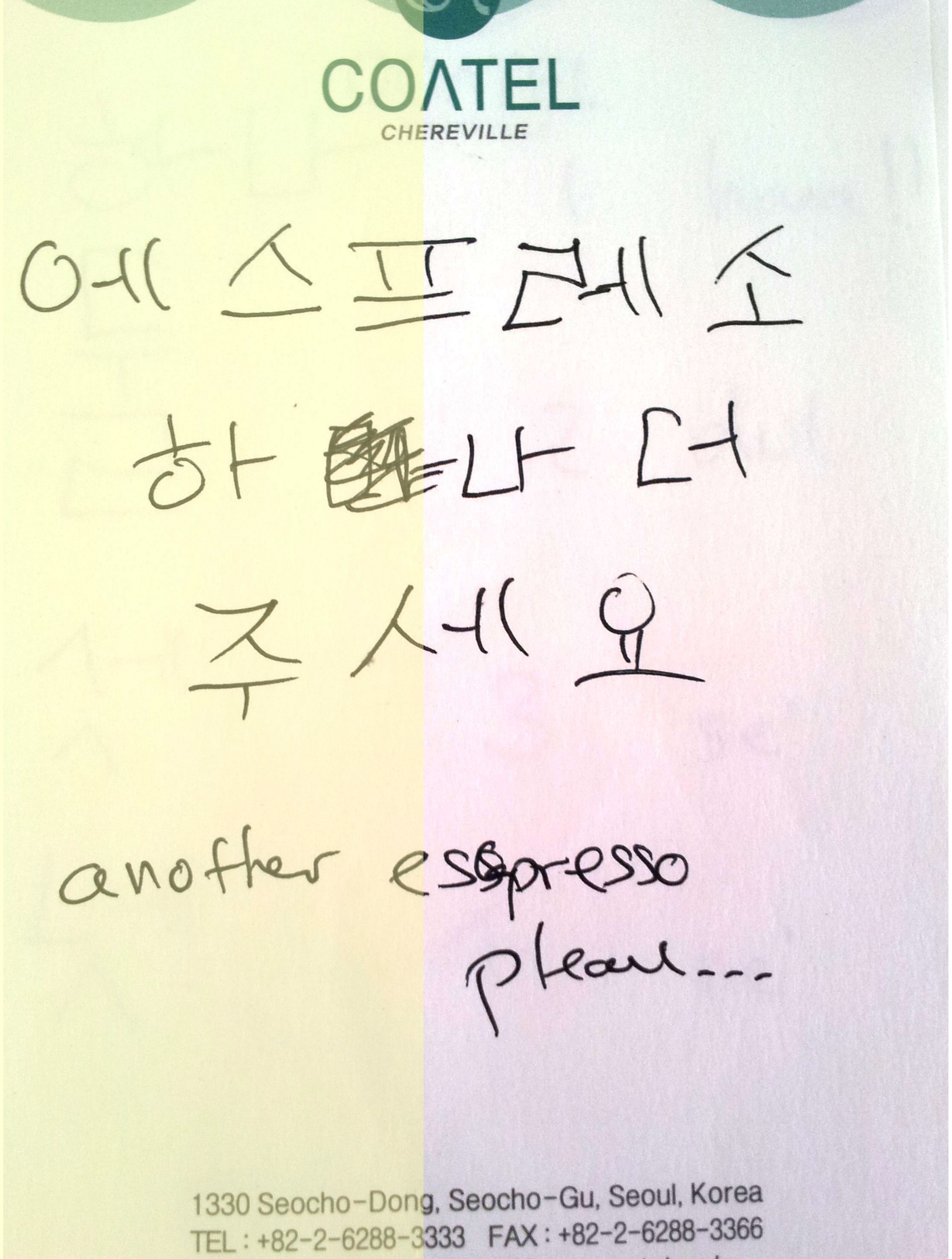
Software Architecture using ØMQ

by
Pieter Hintjens
Strange Loop 2012

A complex story is best told as a series of vacuous 1-liners


COATEL
CHEREVILLE

에 스 프 레 소
하 [illegible] 나 더
주 세 요

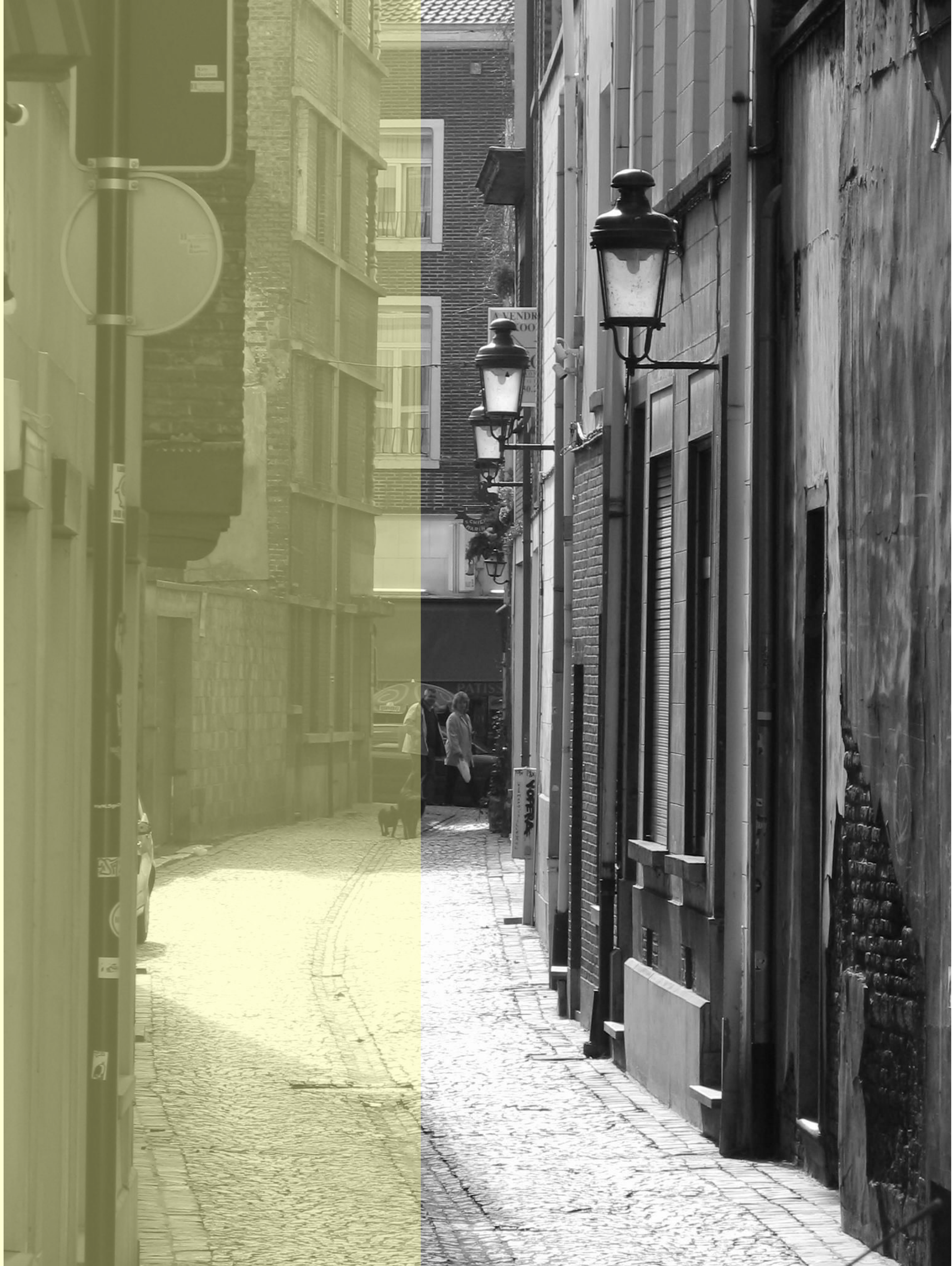
another espresso
please...

1330 Seocho-Dong, Seocho-Gu, Seoul, Korea
TEL : +82-2-6288-3333 FAX : +82-2-6288-3366

90% of software is trash.

90% of the rest will be trash RSN

We basically don't know how to make code that can survive ten, let alone 50 years



The most difficult challenge in our profession is simple accuracy

Future code has to talk to code, has to be chatty, sociable, well-connected

When we can move faster, where we go is more critical than ever.

Writing distributed code is like a live jam session.

It's all about other people

How we connect to each other matters more than who we are

The physics of software is the physics of people

Ideas are cheap.

Execution is the hard part

Making perfect software is easy, once you learn the trick (which is kinda hard)

Simplicity always beats functionality

Problems are not all equal, and most are illusions

When you know the real problem you have done half the work

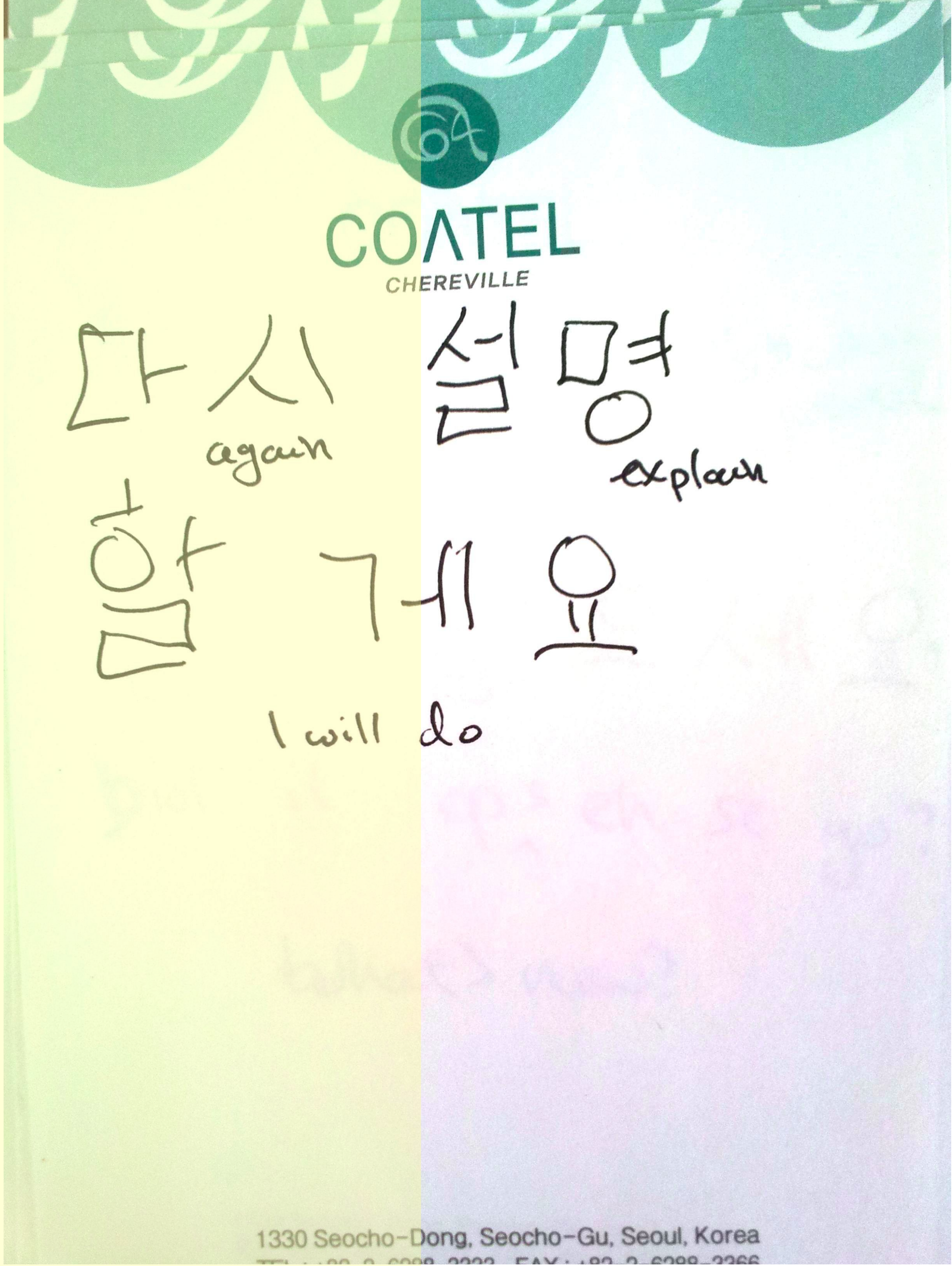
Do nothing that is not a minimal, plausible answer to a well-defined problem

Every commit should be shippable

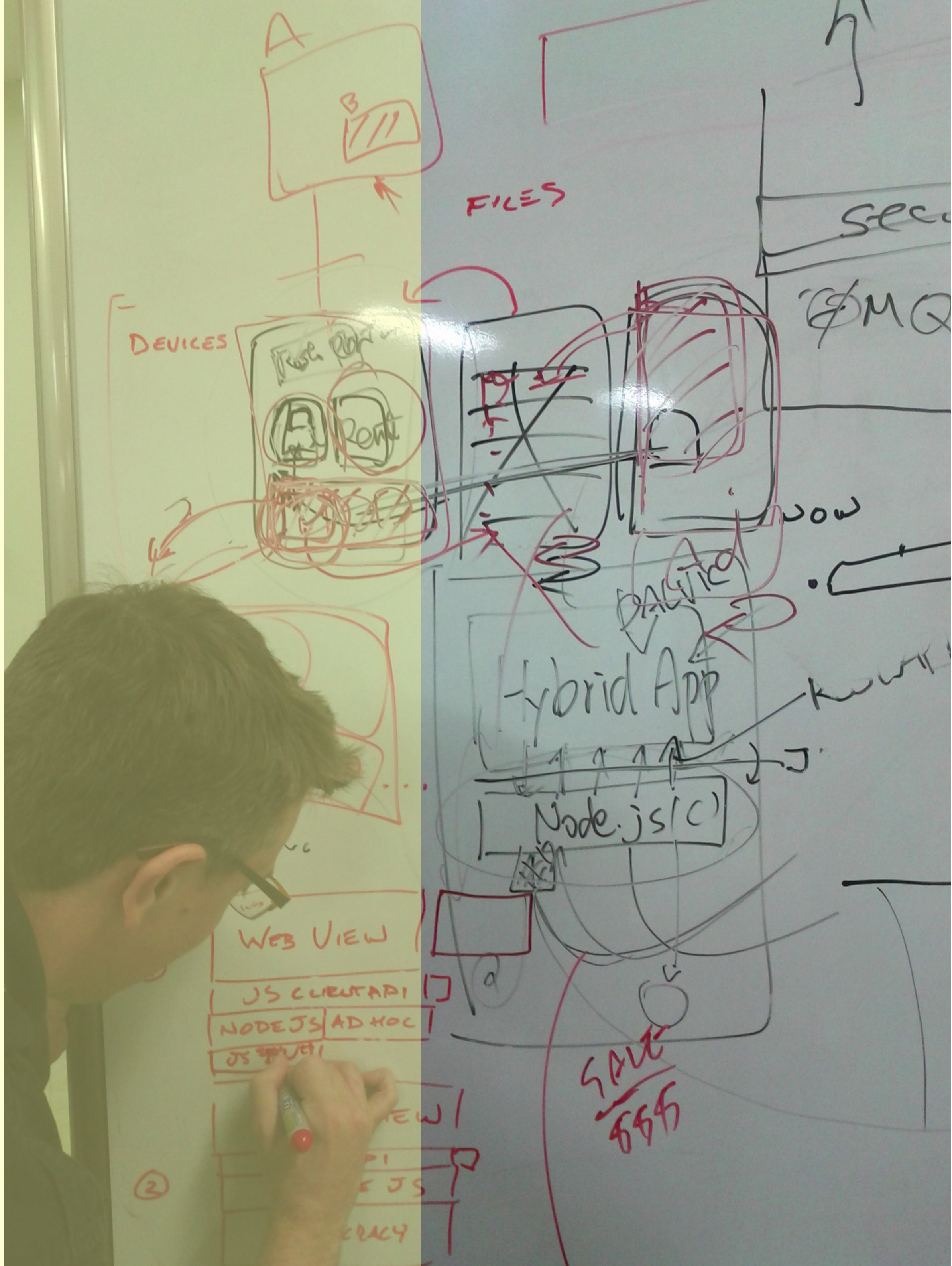
Design by removing problems, not adding features

Five Steps to Satori: Learn, Draw, Divide, Conquer, Repeat

1.
Learn the language before you write a poem


COATEL
CHEREVILLE
다시 설명
again
explain
할게요
I will do
1330 Seocho-Dong, Seocho-Gu, Seoul, Korea

2.
If it looks pretty, it's more likely to work

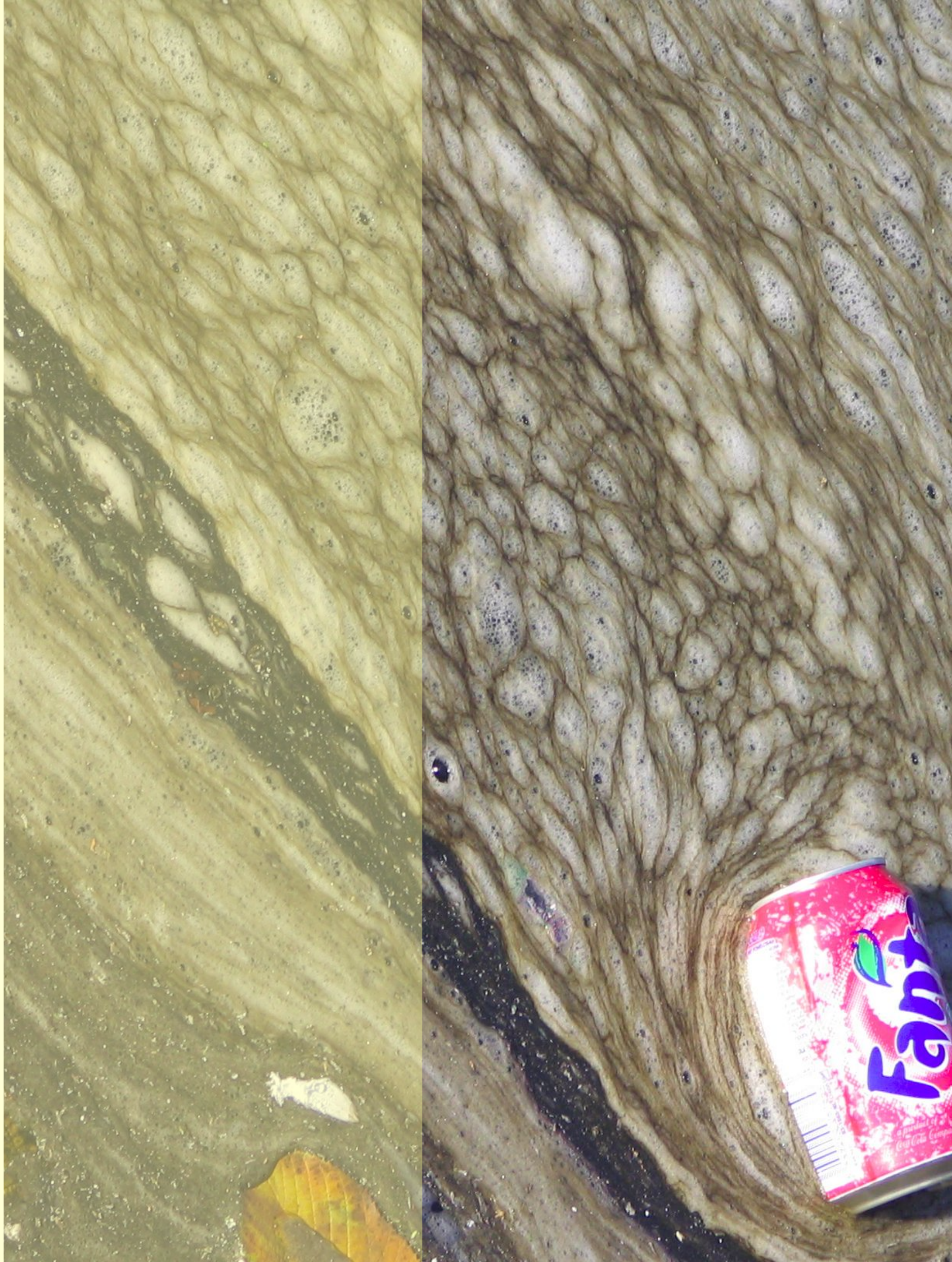


3.
A good contract is worth a thousands assumptions

4. When you take small steps, it hurts less when you fall

5.
Solve one problem, and repeat until you run out of time or money

Distributed software lives or dies by its protocols



Protocols are contracts that describe the rights and obligations of each party

An unprotocol takes minutes to explain, hours to design, days to write, weeks to prove, months to mature, and years to replace

Use human language in your unprotocols.

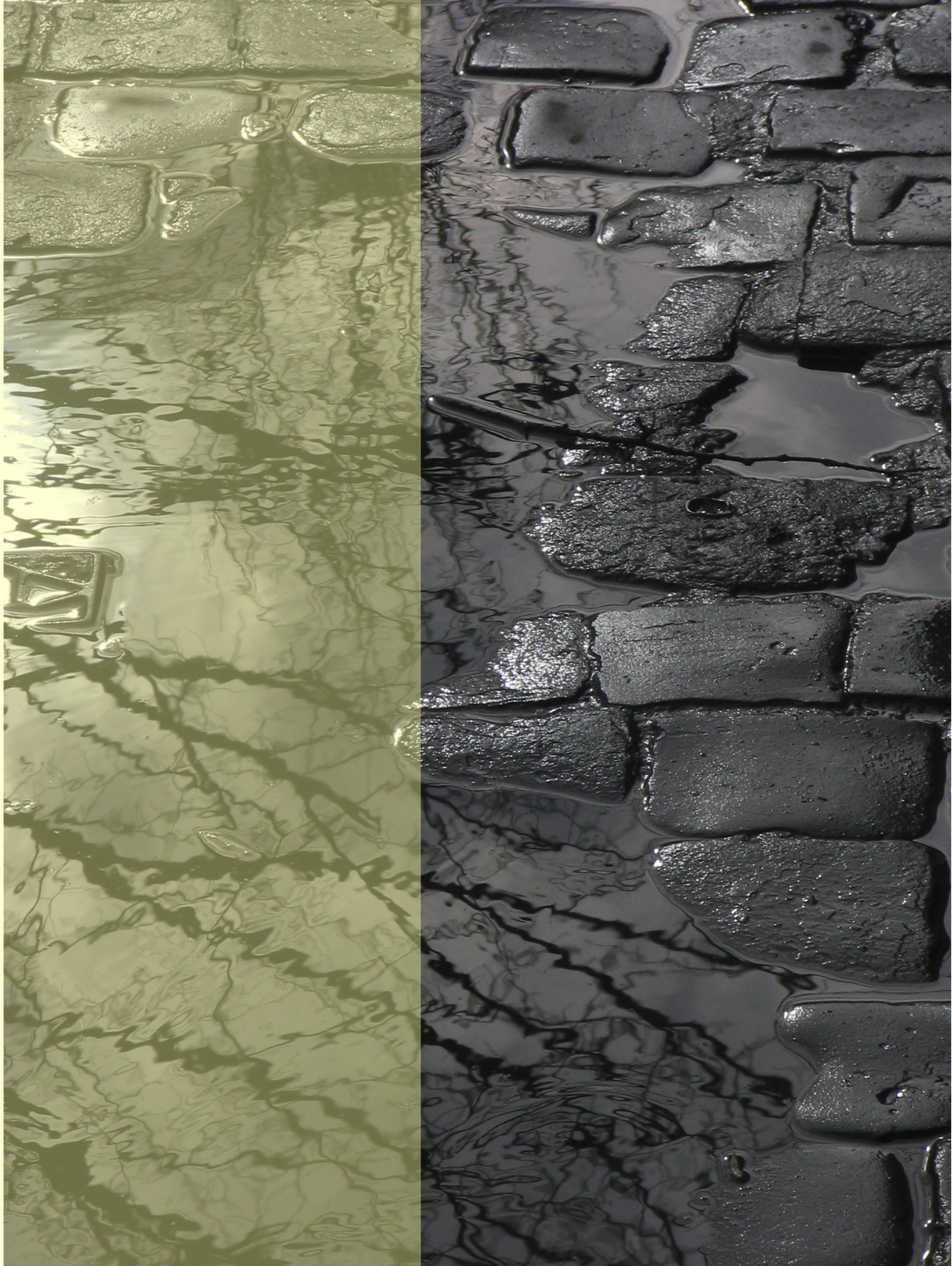
ORLY? YARLY!

```
nom-protocol =
   open-peering
  *use-peering

open-peering =
   C:OHAI
 ( S:OHAI-OK / S:WTF )

use-peering  =
   C:ICANHAZ
 / S:CHEEZBURGER
 / C:HUGZ S:HUGZ-OK
 / S:HUGZ C:HUGZ-OK
```

Use GPLv3 for your open specs.

Remixability is freedom

If you're willing to give up flexibility for speed you deserve neither flexibility nor speed

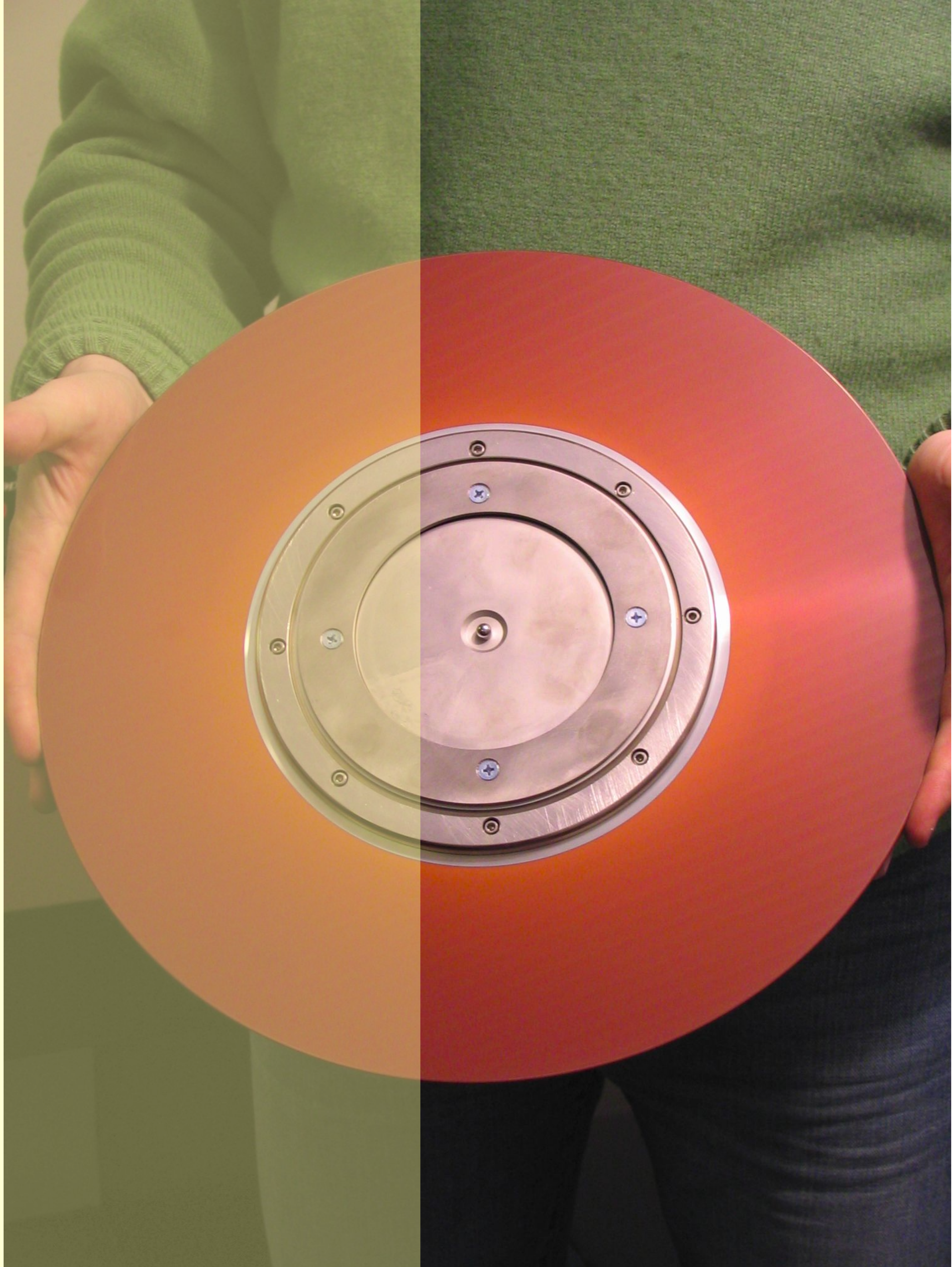


Use cheap text for the low-volume chatty control commands

Use nasty hand-coded binary for the high-volume data

ØMQ framing makes a lousy codec but a great separator

A hand-crafted codec can always beat a generic serializer



A code-generated codec can always beat a hand-crafted one

iMatix GSL: technology so dangerous we had to lock it up for years

File transfer is the zombie problem of distributed applications

Router sockets are the beating heart of every real ØMQ protocol engine

The world needs a chunked, flow-controlled, restartable, cancellable, async, multicast file transfer ØMQ protocol

No matter how hard you push, a file will not just go down a socket

```
C: fetch
S: chunk 1
S: chunk 2
S: chunk 3
```

That annoying pause after you finish your beer, before you catch the waiter's eye

```
C: fetch chunk 1
S: send chunk 1
C: fetch chunk 2
S: send chunk 2
C: fetch chunk 3
S: send chunk 3
C: fetch chunk 4
```

You can, and I've tested this, order a new beer before your old one is empty

```
C: fetch chunk 1
C: fetch chunk 2
C: fetch chunk 3
S: send chunk 1
C: fetch chunk 4
S: send chunk 2
S: send chunk 3
```

Request-reply is just a vulgar subclass of publish-subscribe

```
C: subscribe
C: send credit
S: send chunk
S: send chunk
C: send credit
S: send chunk
```

On a router socket, you should never hit the high-water mark

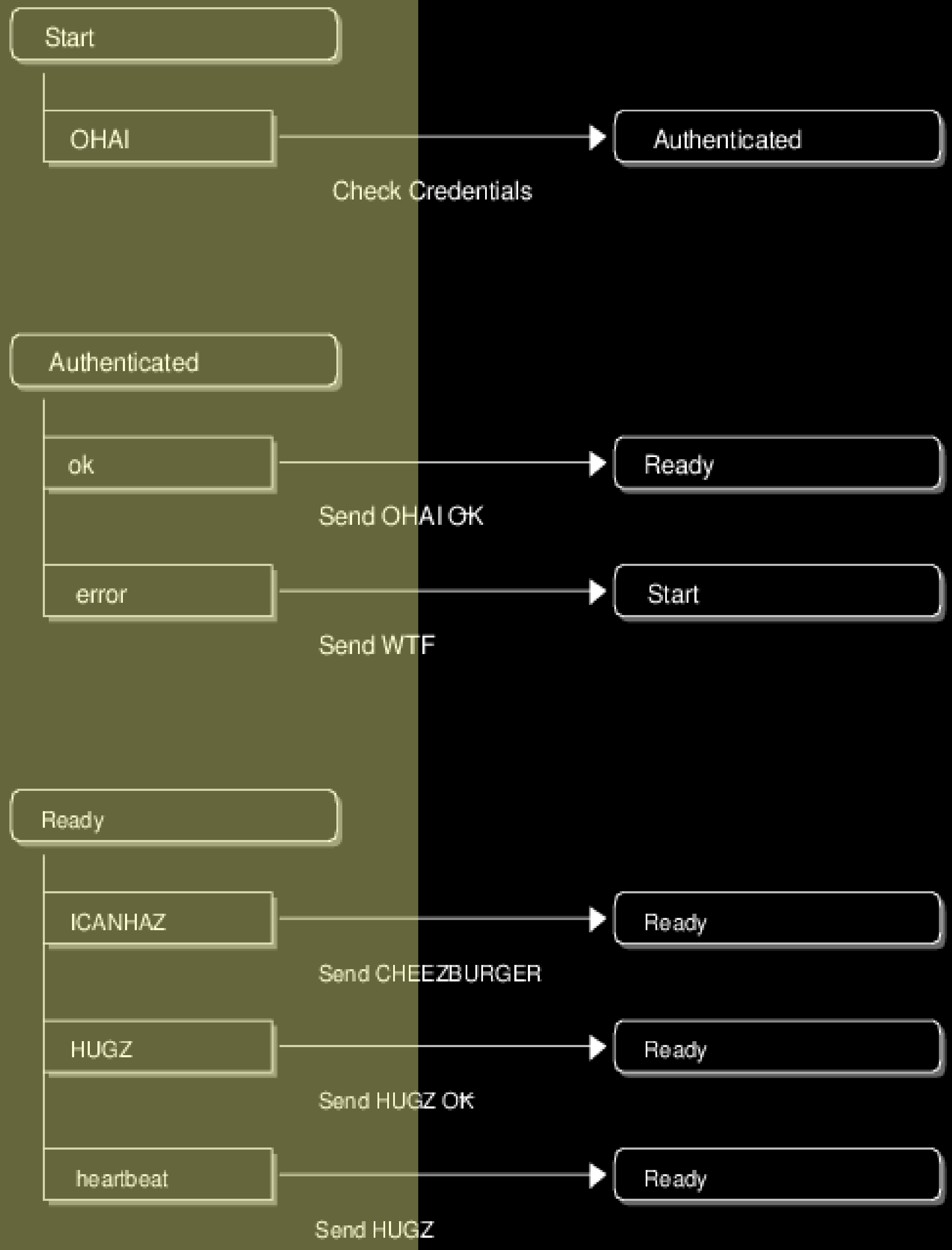
Heartbeats are our protocol's way of asking if we still care

Protocol stack = message codec + protocol engine

```
command_t
  *request =
command_decode
  (socket)

execute_engine
  (command)
```

State machines are a perfect domain language for protocol engines

State machines can be cudly and gentle, when you get to know them



You don't want to bet against a compiler

If you're not thinking of security, security is probably thinking of you

For connected bidirectional protocols over ØMQ use SASL

For loosely connected and one-way protocols over ØMQ, use AES and such

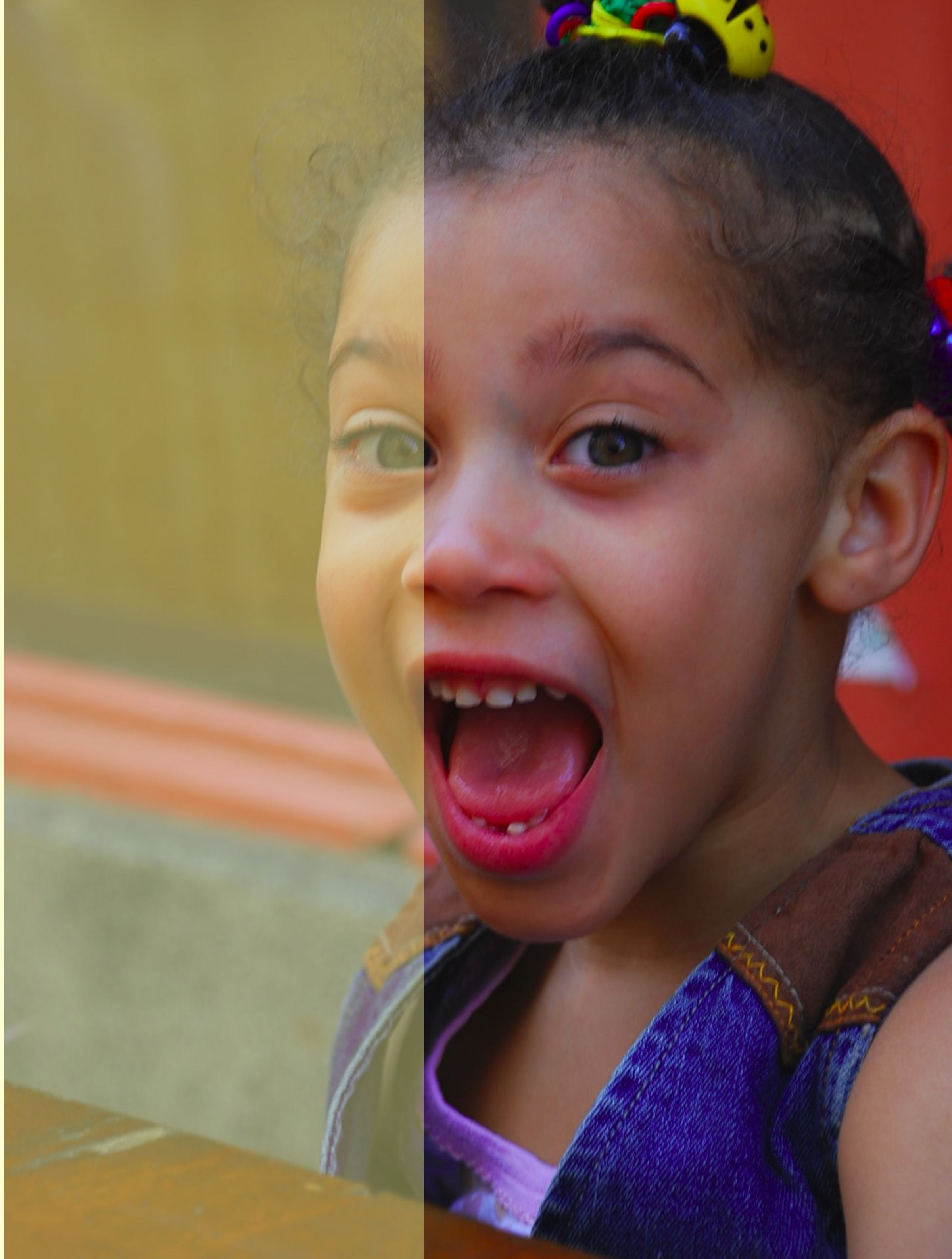
SASL over ØMQ is darned simple

```
secure-nom = open-peering
            *use-peering

open-peering =
    C:OHAI
 *( S:ORLY C:YARLY )
  ( S:OHAI-OK / S:WTF )

ORLY = 1*mechanism
       challenge
mechanism = string
challenge = *OCTET

YARLY = mechanism
        response
response = *OCTET
```

Theory is fine in theory, but in practice, practice is better

FileMQ is a file sharing protocol and stack over ØMQ.

Reusable until 2062



To sum it up:

zero.mq/ch6

The Weird Fish Book, coming soon from O'Reilly

1. Aim for 50 years
2. It's all about people
3. Minimal plausible solutions
4. To real immediate problems
5. Document the contracts
6. Cheap and Nasty codecs
7. Code generation rocks
8. Router sockets rock
9. CBFC > HWM
10. Learn state machines
11. Learn about SASL
12. Worked example: FileMQ