An Abridged Guide to Event Sourcing

Tomer Gabel

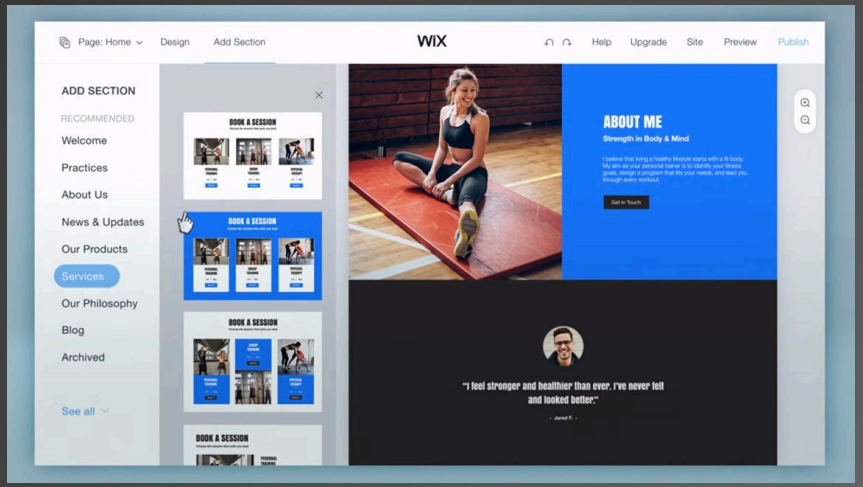
BuildStuff 2017

Vilnius

Image: Jack Zalium, "Abriged" via Flickr (CC-BY-ND 2.0)

Background

- ADI is a *site builder*
- It's pretty nifty
  - Huge web application
  - … and it even works!
- A cool app isn't enough
- Sites have to be *stored* somewhere!



Requirements

- Features:
  - Store "site" *blobs*
  - Store *version history*
  - *Soft-delete* only
- *Not* required:
  - Concurrent editing

Image: ImgFlip

crud

/krəd/

noun *informal*

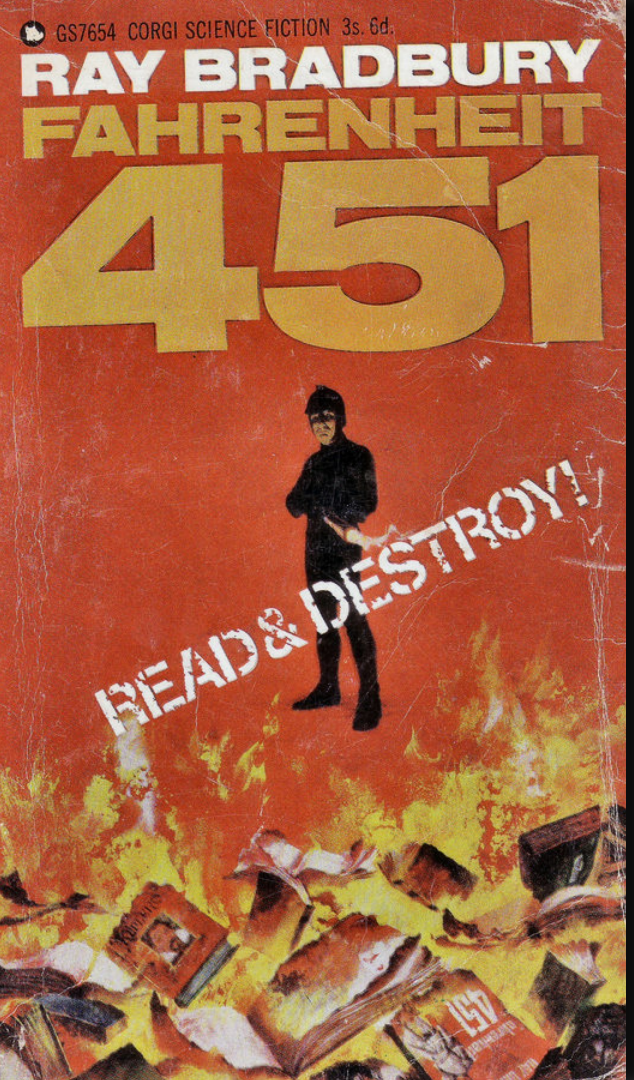
1. A substance which is considered unpleasant or disgusting, typically because of its dirtiness.
   'use a good soap compound to remove accumulated crud'
2. Nonsense.
   'the usual crud which passes itself off as a smart twenty-something comedy'

Source: Oxford Dictionary

1. WHY CRUD SUCKS

Mutation Anxiety

- So what's wrong with CRUD?
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete
- It's all about *mutable state*



Mutation Anxiety

Mutable state is bad.

- Old data is *lost*
- Hard to *debug*
- Can't fix *retroactively*
- No built-in *auditing*

Image: David Bleasdale, "Fahrenheit 451" via Flickr (CC-BY 2.0)

WIX Engineering

Mutation Anxiety

"Who told you
you're allowed to
destroy data?"

-- Greg Young

Image: Code on the Beach speaker profile (source)

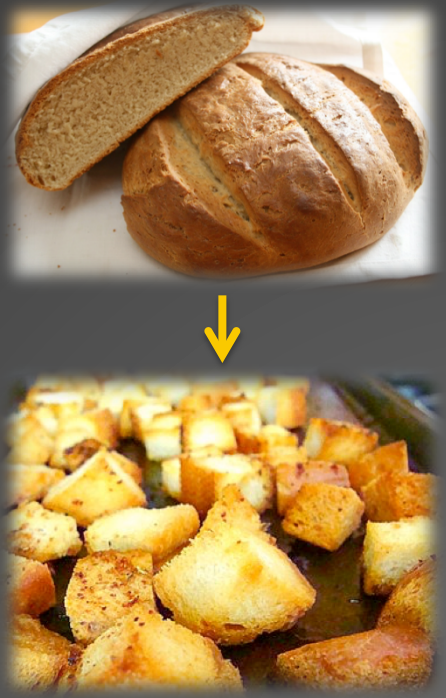
Not To Scale

- CRUD implies:
  - One source of truth
  - Reads, writes against the *same store*
  - Full *consistency* (ACID)
- Difficult to scale!

Image: Jason Baker, "Bank Vaults under Hotels in Toronto, Ontari" via Flickr (CC-BY 2.0)

What's Holding Us Back?

- Strong consistency
  - Assumed with RDBMS
  - Often *not required*!
- Consistency is a *product* concern
  - "Does this have to be 100% fresh?"
  - "No? So how stale can this get?"



Images: "Homemade Bread Freshly Baked" via MaxPixel (CC0, above); Tasha, "Homemade Croutons" via Flickr (CC-BY 2.0, below)

What's Holding Us Back?

- Storage cost
  - "How long must I hold on to data?"
  - "What do you mean forever?!"
- Cost is an *operational* concern
  - "So how much data is there?"
  - "How much will it cost to retain?"

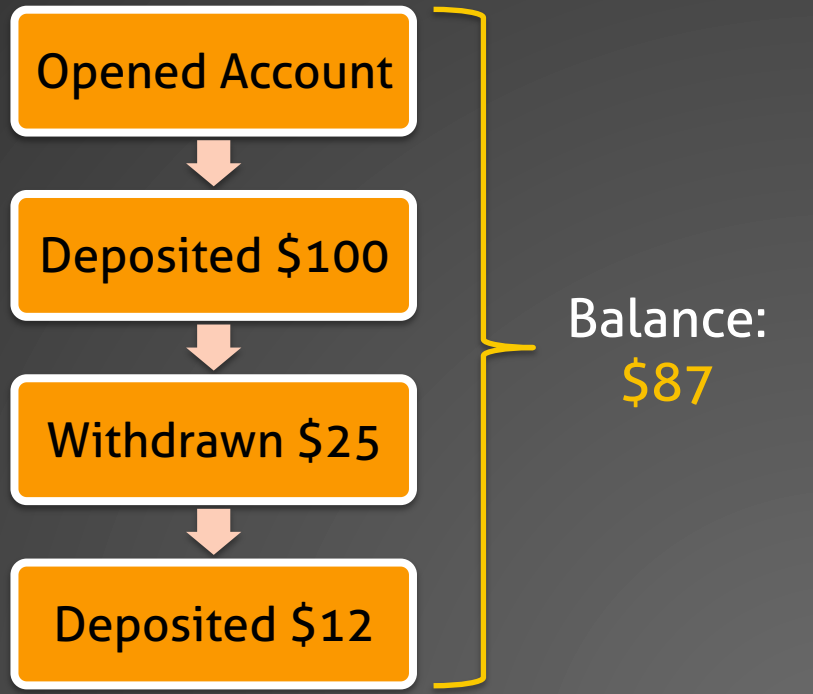
"A database is just a view over its transaction log."

-- Ancient Vulcan proverb

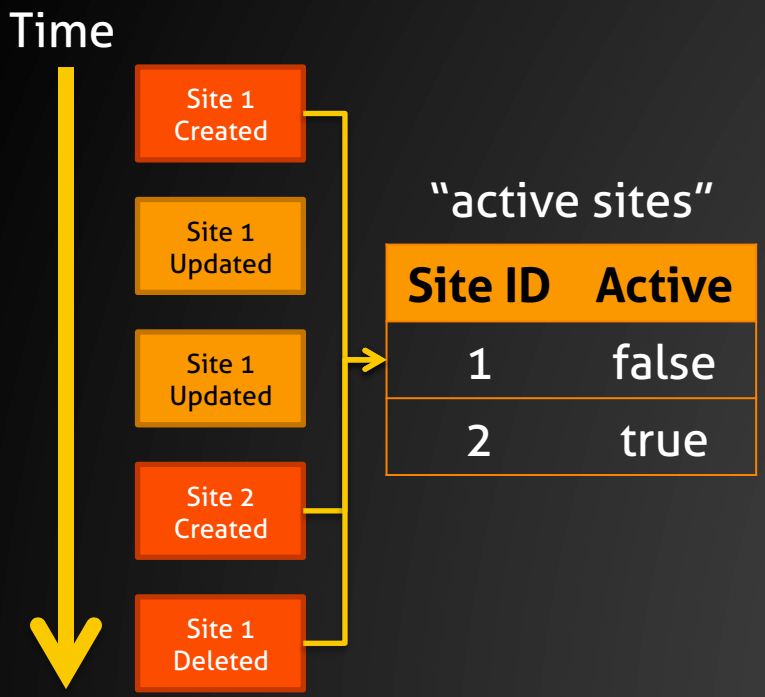
2. A BETTER WAY

Event Sourcing

- A very simple pattern
- Each entity is an *event stream*
  - Events are *facts*
  - Events are *immutable*
  - Events are *forever*



CQRS



- *Writes* simply append events
- But reads are *projections*:
  - Full, partial selects
  - Views, joins
- Two separate concerns!
  - Different *schemas*
  - Different *instances*
  - Even different *data stores*!

Better how?

- *Tunable consistency*
  - Full or eventual
  - Per use-case!
- *Decoupled reads/writes*
  - Better scale
  - Much more flexible!
- *Built-in auditing*
  - Easier to debug
  - Replayable!

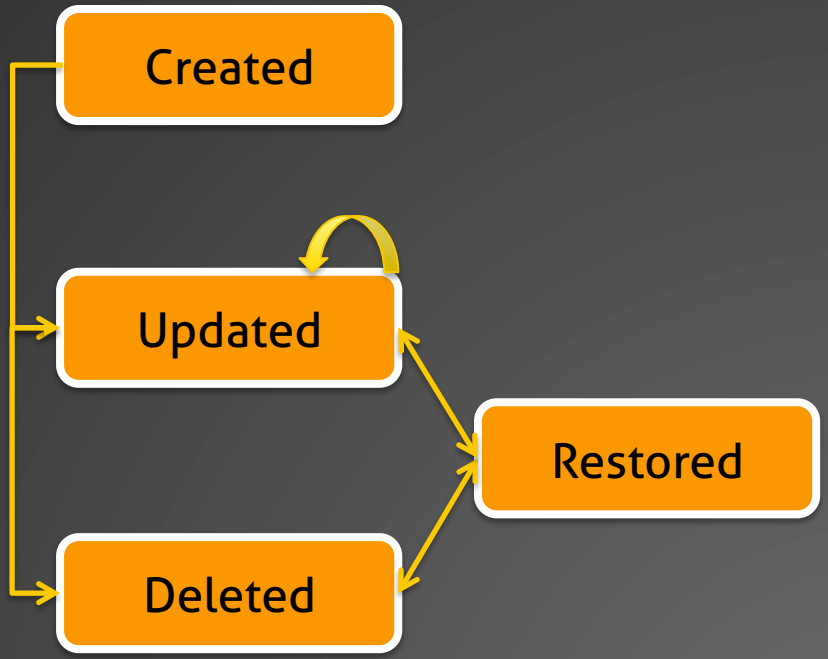
3. WALKTHROUGH

Image: Pöllö via Wikimedia Commons (CC-BY 3.0)

WiX Engineering

An Event Model

- Our business domain: a *website*
- Our use cases:
  - "Let's try this thing out"
  - "Adding more stuff"
  - "Oops! Revert"
  - "OK, I'm outta here"



Storing Events

- What's in an event?
  - The entity (i.e. site) *ID*
  - Some notion of *time*
  - Event *data* (e.g. type)
  - Some *metadata*
- We can model this!

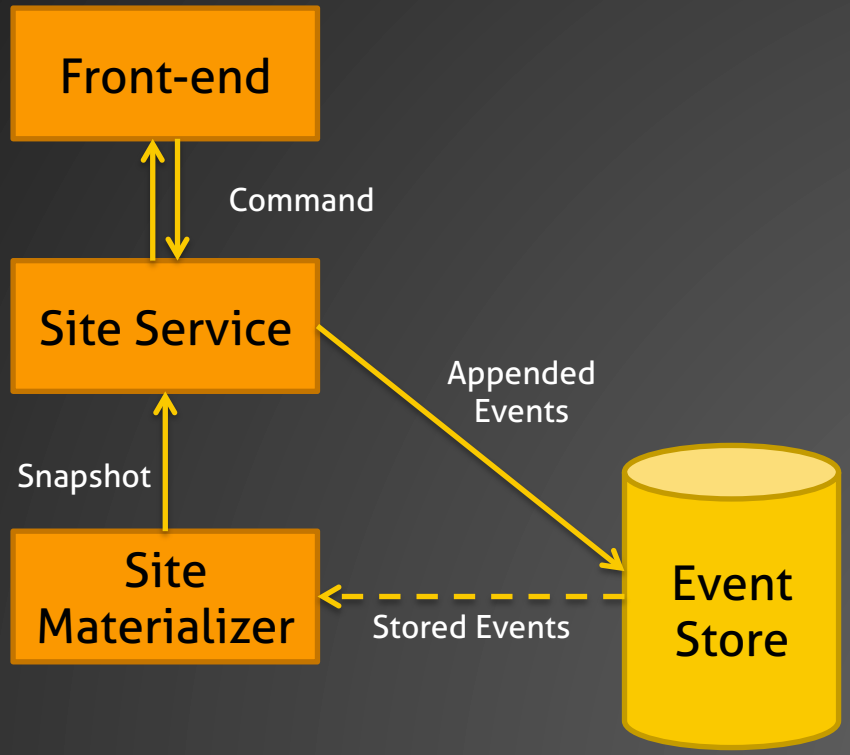
Column	Type	PK?	Null?
site_id	binary	✔	✖
version	int	✔	✖
event_time	timestamp	✖	✖
payload	binary	✖	✖

A Service is Born

- Commands are *wishes*
- Wishes may be *rejected*:
  - Conflicting updates
  - Invalid arguments
  - Faulty dependencies
- Or *granted:*
  - Result: new events!

Command	SLA
Create Site	Reasonable
Get (Latest)	Very fast
Get (Version)	Reasonable
Update	Very fast
Delete	Reasonable
Restore	Reasonable

Architecture 101

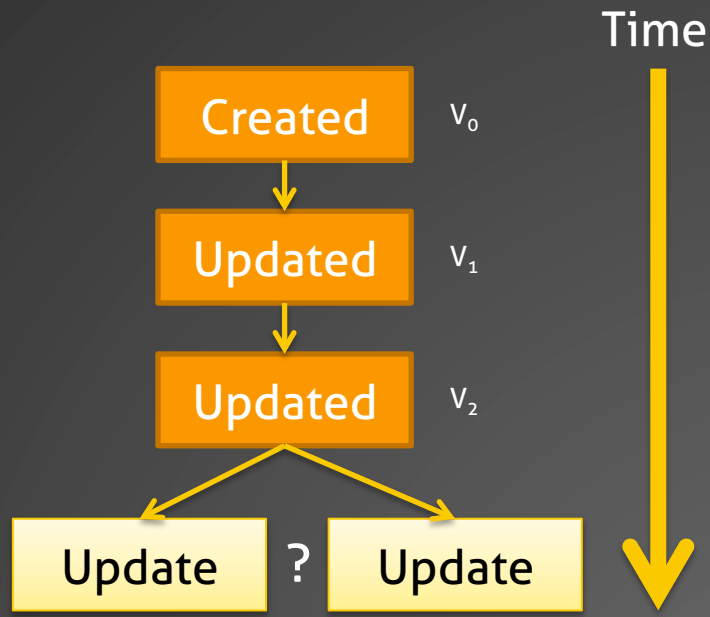


Hold Your Horses

- Still some *concerns*:
  - Conflicting updates
  - Performance
  - Operations
- Let's *sort 'em out*

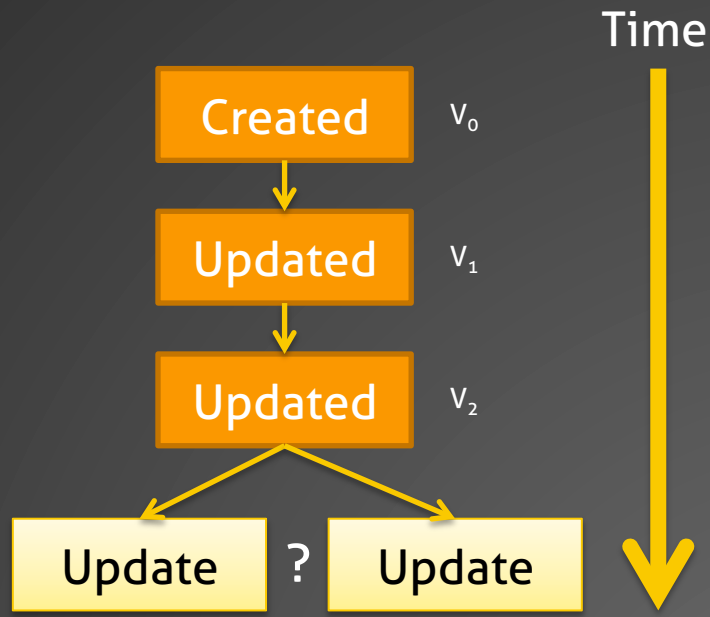
Conflicting Updates

- *Concurrent* operations can *conflict*
  - Rapid site activity ("double click")
  - Concurrent editing ("open in another tab")
  - Normal network issues
- You *have* to deal with it



Conflicting Updates

- Strategies
  - Last-write wins
  - Optimistic locking
  - Smart resolution/merge
- In our case
  - No concurrent editing
  - Simple optimistic locking



Performance

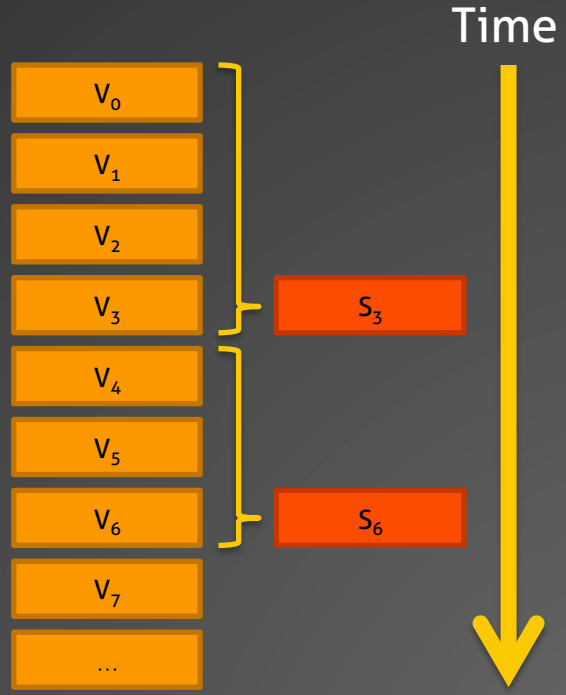
- Updates are easy
- What about *reads*?
  - Load all events for site
  - Feed into materializer
  - Spit snapshot out
- *This can hurt*.

Performance

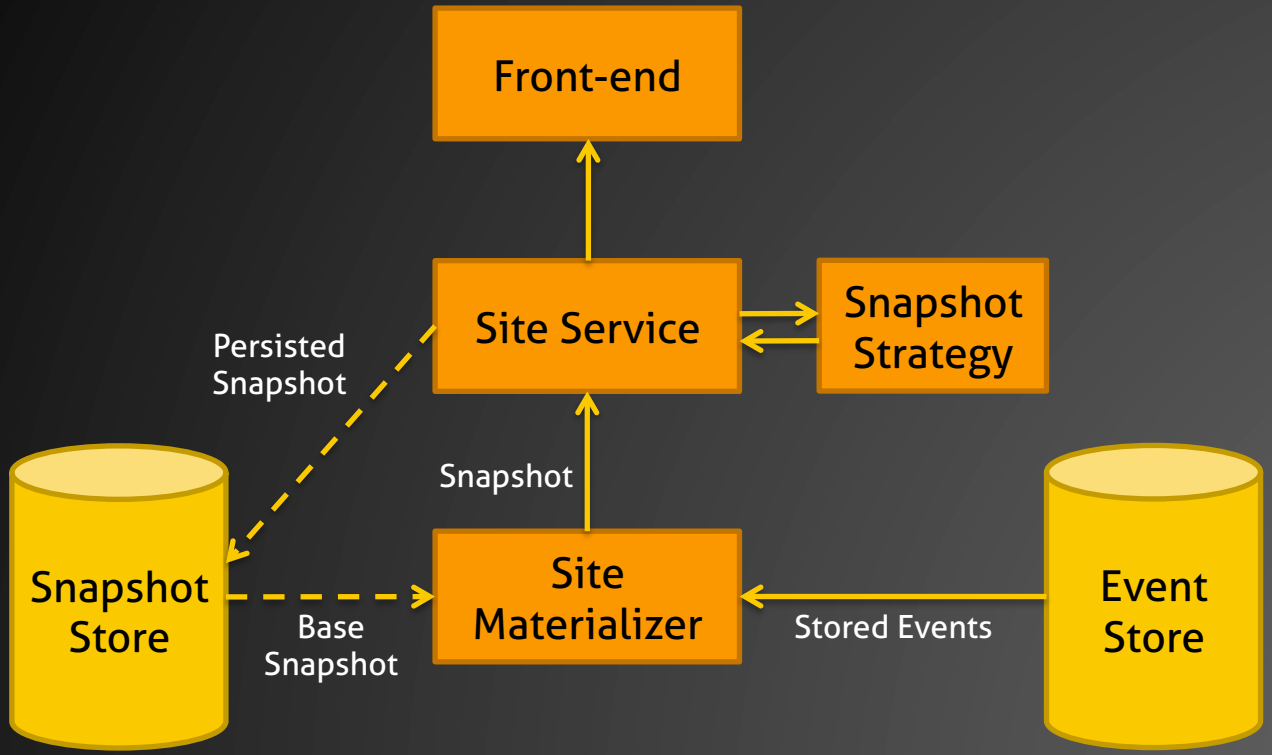
- How *many* events?
  - Domain-specific (for ADI, easily 1000s)
  - But events *never die*
- How *big* are they?
  - Again, domain-specific
  - Let's assume 10-100KB
- Naïve reads *will fail*.

Performance

- Remember *CQRS*?
  - Reads are distinct from writes
  - We can use another store!
- We'll use *snapshots*
  - Immutable
  - Ephemeral
  - Tunable (space/performance)



Architecture Redux



4. LESSONS LEARNED

No Silver Bullet

- Event sourcing is *awesome*
- But it's a *trade-off*
  - Learning curve
  - Increased storage
  - More knobs to turn
- Make it wisely!

Image: Ed Schipul, "silver bullet" via Flickr (CC-BY-SA 2.0)

Eventual Consistency

- Consistency is a *tradeoff*
  - Performance
  - Complexity
  - Cost (tech, support)
- Make it wisely!

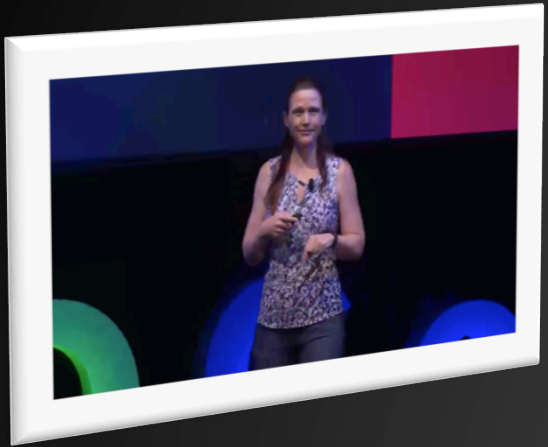
Image: Michael Coghlan, "Scales of Justice - Frankfurt Version" via Flickr (CC-BY-SA 2.0)

WIXEngineering

Forethought

- Define target *SLAs*
  - Latency
  - Consistency
- Place *sanity limits*
  - Stream size
  - Write throttling
- Invest in *tooling*
  - Debug/replay
  - Schema evolution

Further Reading



Scaling Event Sourcing for Netflix Downloads

Phillipa Avery & Robert Reta

How shit works: Time

Tomer Gabel

tomer@tomergabel.com
@tomerg
http://engineering.wix.com

Sample implementation:
http://tinyurl.com/event-sourcing-sample

Thank you for listening

QUESTIONS?

WIXEngineering