The ~~art~~ SCIENCE of debugging

Nati Cohen (@nocoot)
Avishai Ish-Shalom (@nukemberg)

What’s in it for me?

- Debugging is a skill, not an innate talent
- Debugging has a well defined methodology
- We can train ourselves in this methodology

Cognitive biases ahead!

Complexity matters

- Cognitive biases help us deal with data overload
- But they are often misleading
- We need to actively avoid them

The methodology:

0. Triage
1. Define and narrow down the symptoms
2. Build a (mental) model of the system
3. Deconstruct, create/revise a theory
4. Corroborate the theory
5. Reconstruct and validate
6. Rinse and Repeat

0. Triage

- What is the business impact?
- Is it actually a problem?
- Should this be handled?
- When should we handle this?

1. Define and narrow down the symptoms

- Can you recreate the issue?
- Isolate the offending conditions
- **Operational definition** of the problem

When? Where? to Whom? Under Which conditions?

Examples

- GET /bla -> returns 500 from all countries, for any headers, in 5% of cases
- p99 of transaction X is consistently over 500ms since 1 hour ago, should be under 100ms
- Transaction Y for user XXX returns empty records, should return 100 records

2. The Mental Model

- We have one for anything we interact with
- Implicit assumptions about how things work
- Sometimes wrong

We need to make the model explicit!

The Mental Model (example)

Motorcycle

counter-steering

3. Deconstruct, create/revise a hypothesis

- Disassemble the system to sub-systems
- How are they connected? what is the input/output of each one?
- Which sub-system(s) is the cause of the problem?
- Or maybe the connection is the problem?

Expand the metal model by drawing more diagrams!

4. Corroborate the hypothesis

- Define what metrics/experiments you need to prove/disprove the hypothesis
- Define the expected results
- Get the data

5. Reconstruct and validate

- Validate the results
    - Using: bounds, system invariants (e.g. Little’s Law), ...
- Compare the results to expectation
- Reconstruct the system from sub-systems
- Remember the integration points!

When things don’t make sense

3 options:

- We are missing data
- The data we have is wrong
- **Our mental model is wrong**

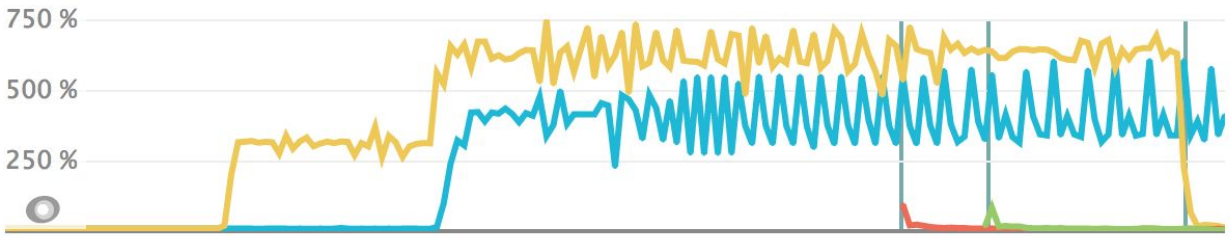
6. Rinse and repeat

- Problem found? excellent
- Problem narrowed down to sub-system? you are now debugging it. Go back to #1
- Problem not found? revise your hyposthesis/model, go back to #2

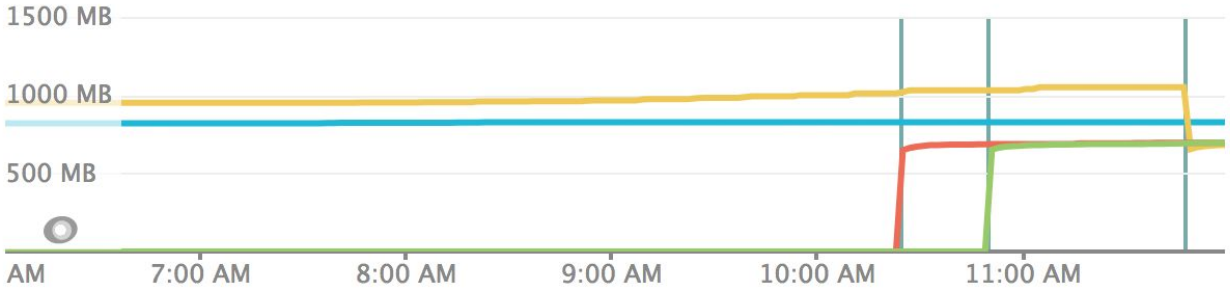
Bonus round (a war story)

OMG, it’s broken!

CPU usage



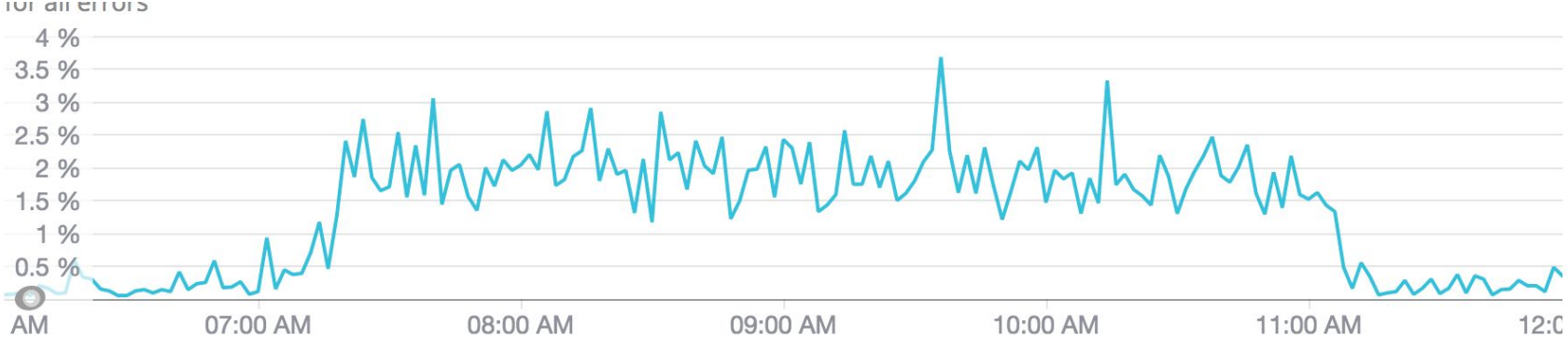
Memory

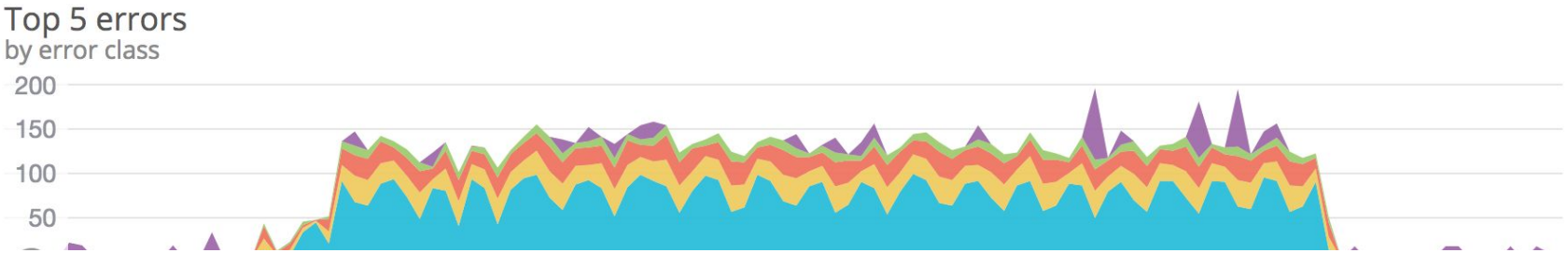


When in doubt, reboot

- Recurring issue, service rebooted instead of debugged
- “Known” issue, but nobody understood why

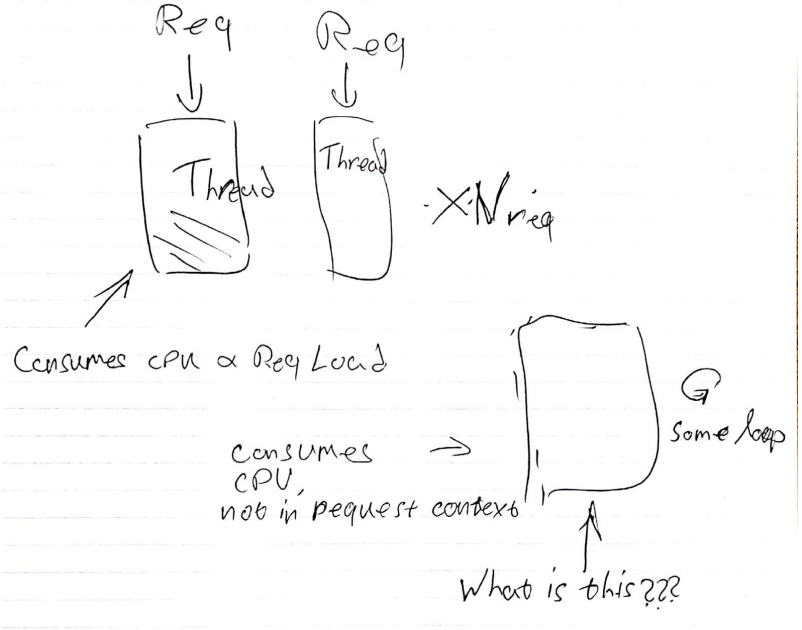
Amazing correlation!



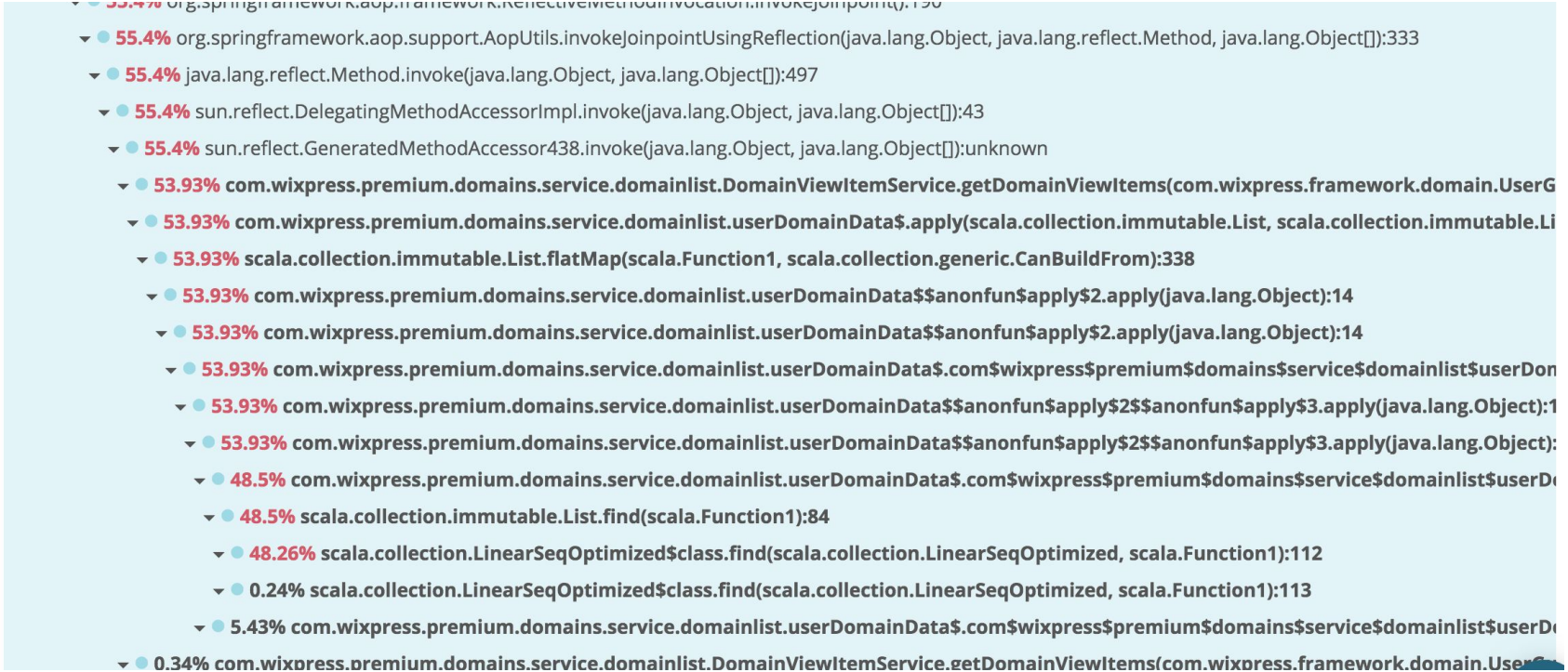


Once again, with methodology

- **Operational definition**: “Transient high CPU, without load”
- Hypothesis “caused by errors”
- Hypothesis disproved
- Let’s **deconstruct** -> we need more data!



Getting data (profiler)



```
@tailrec
private def isConnectedToASite(userDomains:
List[DomainDTO])(domain: DomainDTO) =
domain.redirectDomain match {
    case Some(redirectToDomain) =>
      val redirectToInfoOpt =
userDomains.find(_.domainName == redirectToDomain)
      redirectToInfoOpt match {
        case Some(redirectToInfo) =>
isConnectedToASite(userDomains)(redirectToInfo)
        case None => false
      }
    case None => domain.siteGuid.isDefined
 }
```

But but but….

- We don’t have redirect loops!
- And even if we did, why didn’t CPU released after timeout?
- Why didn’t we get StackOverflowError?

We needed to revise our mental model!

```
@tailrec
private def isConnectedToASite(userDomains:
List[DomainDTO])(domain: DomainDTO) =
domain.redirectDomain match {
    case Some(redirectToDomain) =>
      val redirectToInfoOpt =
userDomains.find(_.domainName == redirectToDomain)
      redirectToInfoOpt match {
        case Some(redirectToInfo) =>
isConnectedToASite(userDomains)(redirectToInfo)
        case None => false
      }
    case None => domain.siteGuid.isDefined
 }
```

The Hypothesis

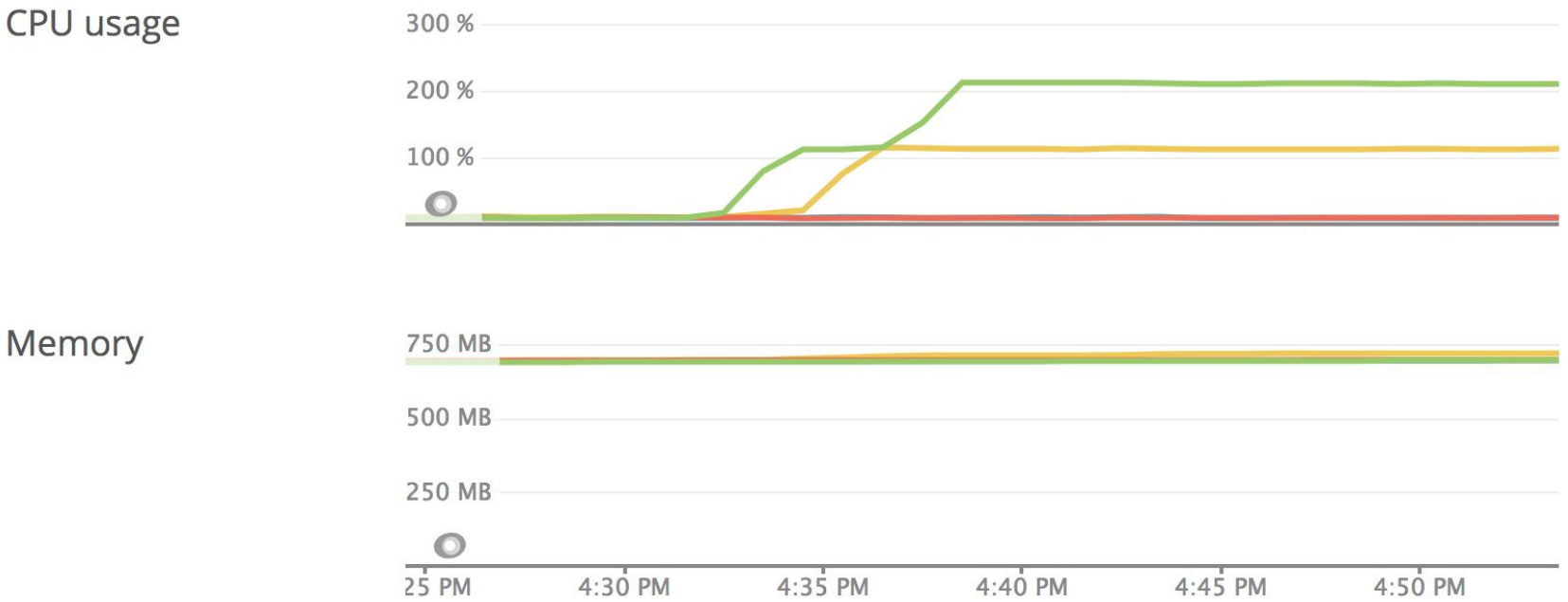
- We have redirect loops!
- Timeout terminates request, not the loop
- @tailrec converts recursion into infinite loop

Validating the hypothesis

- Queried 400k records
- Wrote program that traversed redirects and detects for loops
- Redirect loops found! (~30)

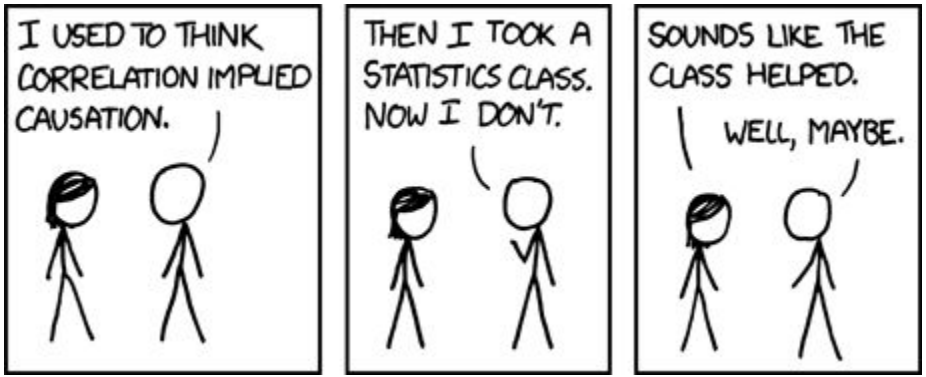
date_created (Local (UTC+2))	src	evid	evid_desc	browser_name	app_url
2018-01-17 07:02:22.635	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 07:03:11.449	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 07:04:44.834	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 07:08:09.226	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 07:08:22.701	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 07:09:33.336	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:08:07.558	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:08:22.608	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:09:40.970	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:10:13.064	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:11:58.265	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:12:11.970	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:13:05.201	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 08:18:15.184	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 10:18:57.918	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 16:33:15.221	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 16:35:18.847	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	
2018-01-17 16:37:34.472	17: billing	1500	V_MpDomainTabLoadSuccess	Chrome 63	

Reproducing the error



Recap

- The methodology proved itself (again)
- Correlation is not causation
- In data we trust (unless we have a good reason not too)



Debugging Kata

Let’s practice

Kata

WIKIPEDIA
The Free Encyclopedia

***Kata**, a Japanese word,*
are detailed choreographed patterns of movements
practiced either solo or in pairs.

Ex. 1: Triage

Player 1: https://bit.ly/2wmn8Qg
Player 2: https://bit.ly/2wmP7j0
Cheat sheet: https://bit.ly/2wybscz

Ex. 1: Triage

Recap

Real life is messy

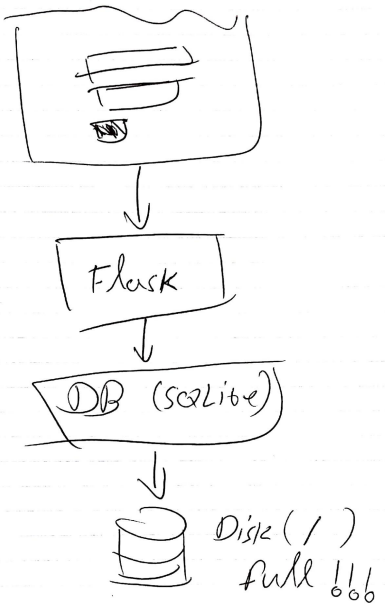
- Triage isn’t always clear cut, but we **have** to make a decision
- What is the cost of being wrong?
- It’s mostly about asking the right questions, fast.

Ex. 2: Debugging

AMI (eu-central-1):
ami-0c82122f26934f541

Ex. 2: Debugging

Recap



Mental model

- `df /` shows filesystem free space
- `du /` aggregates space consumed on filesystem

Mental model

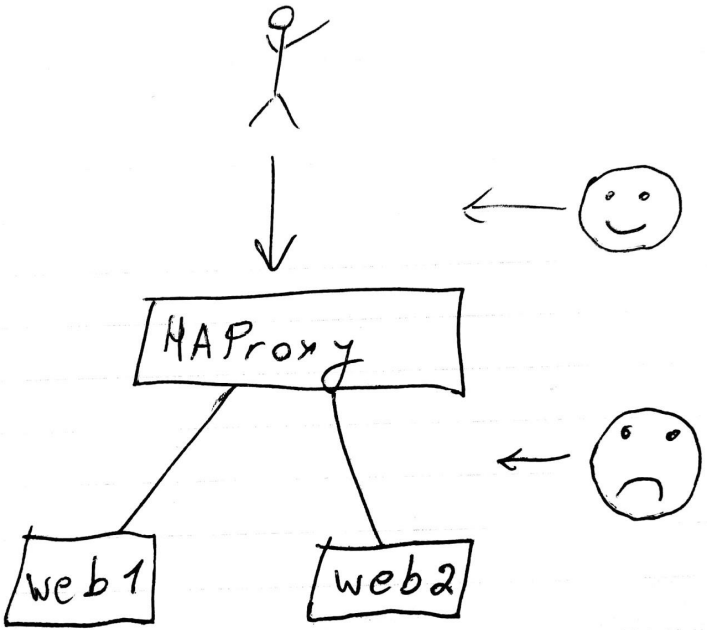
- `df /` shows filesystem free space
- `du /` aggregates space consumed on **VFS**
- VFS has different files than the filesystem mounted on /

Ex. 3: Debugging

AMI (eu-central-1):
ami-097261df74acf8910

Ex. 3: Debugging

Recap



Mental model

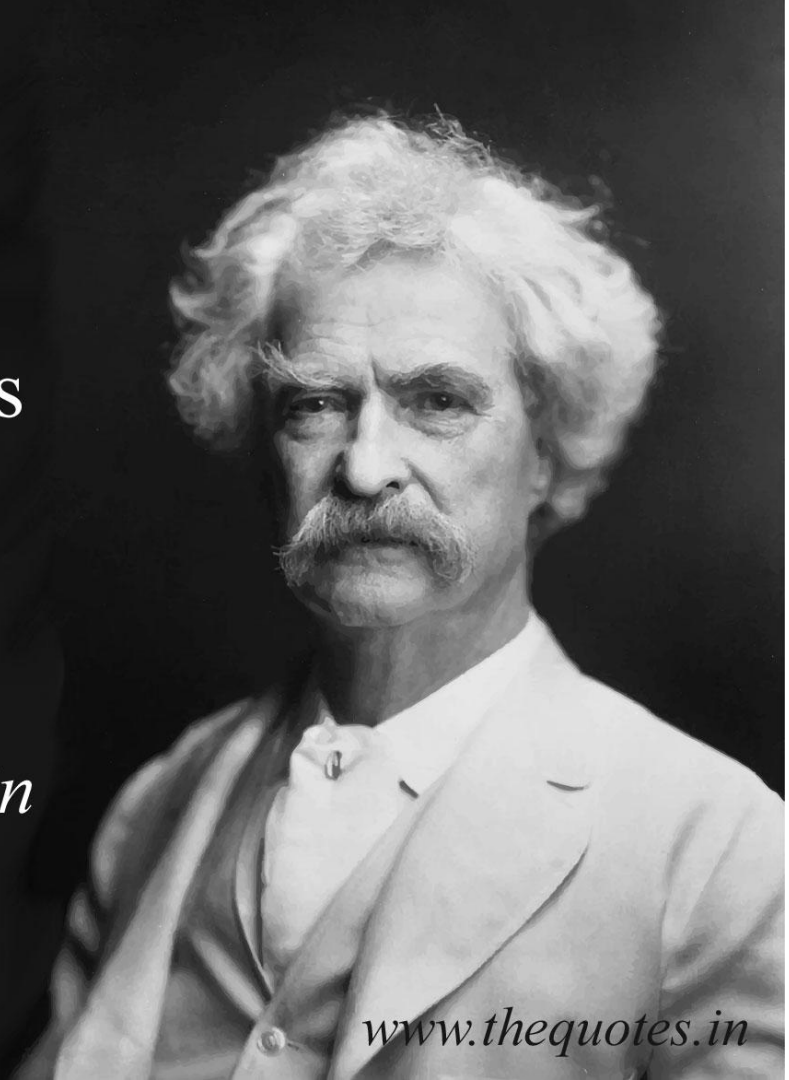
- haproxy resolves backends from **`/etc/hosts`**, **`/etc/resolv.conf`**
- **`/etc/hosts`**, **`/etc/resolv.conf`** managed by docker

Mental model

- Haproxy statically linked
- Alternate libc, alternate resolver
- Resolves from **`/usr/local/etc/host, /usr/local/etc/resolv.conf`**
- **`/etc/hosts, /etc/resolv.conf`** managed by docker

It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so.

Mark Twain



The ~~art~~ SCIENCE of debugging

Thank you!

Nati Cohen (@nocoot)
Avishai Ish-Shalom (@nukemberg)