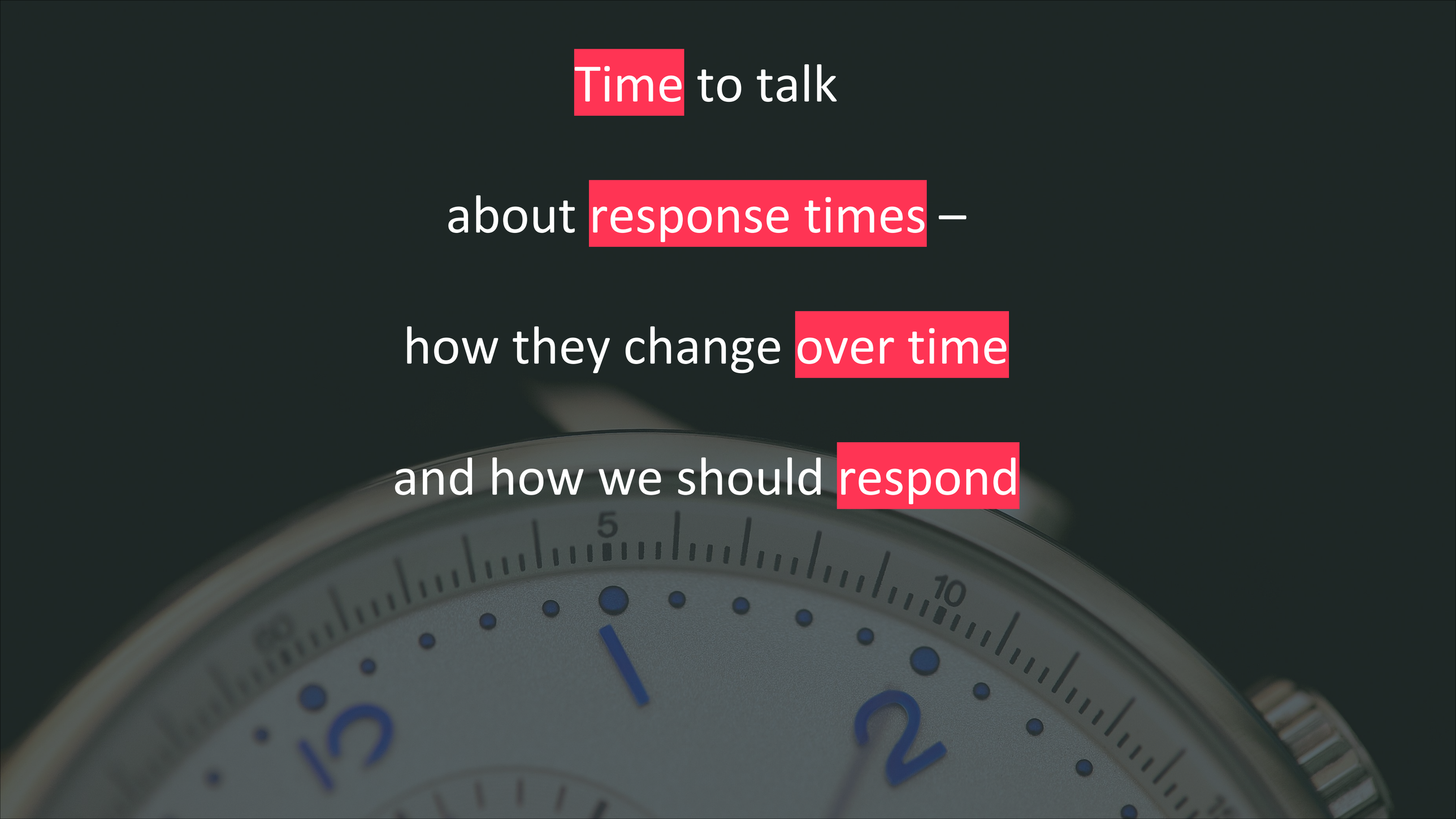
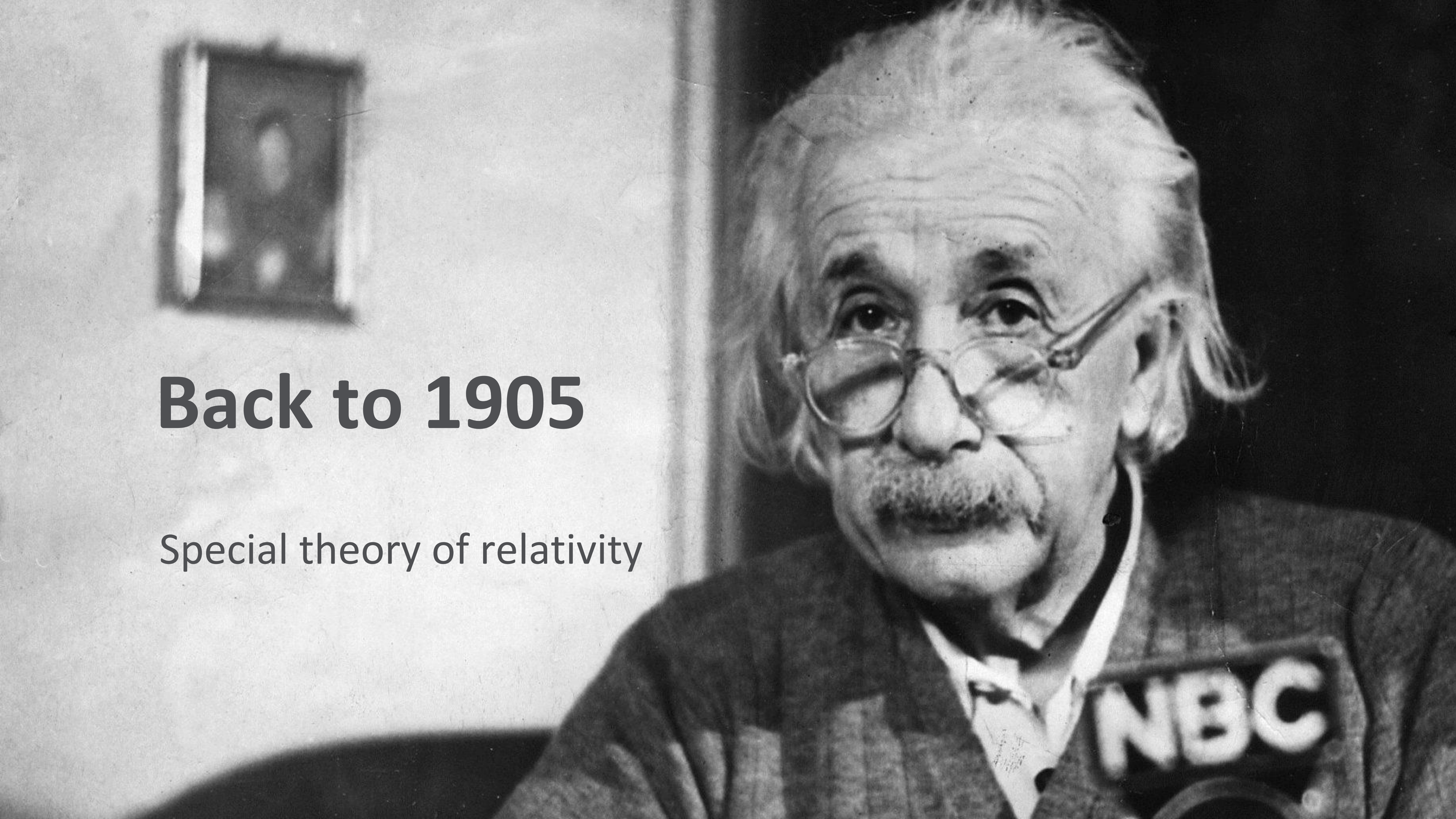
DEVBRIDGE GROUP

Addressing the performance (before someone else does it for you)

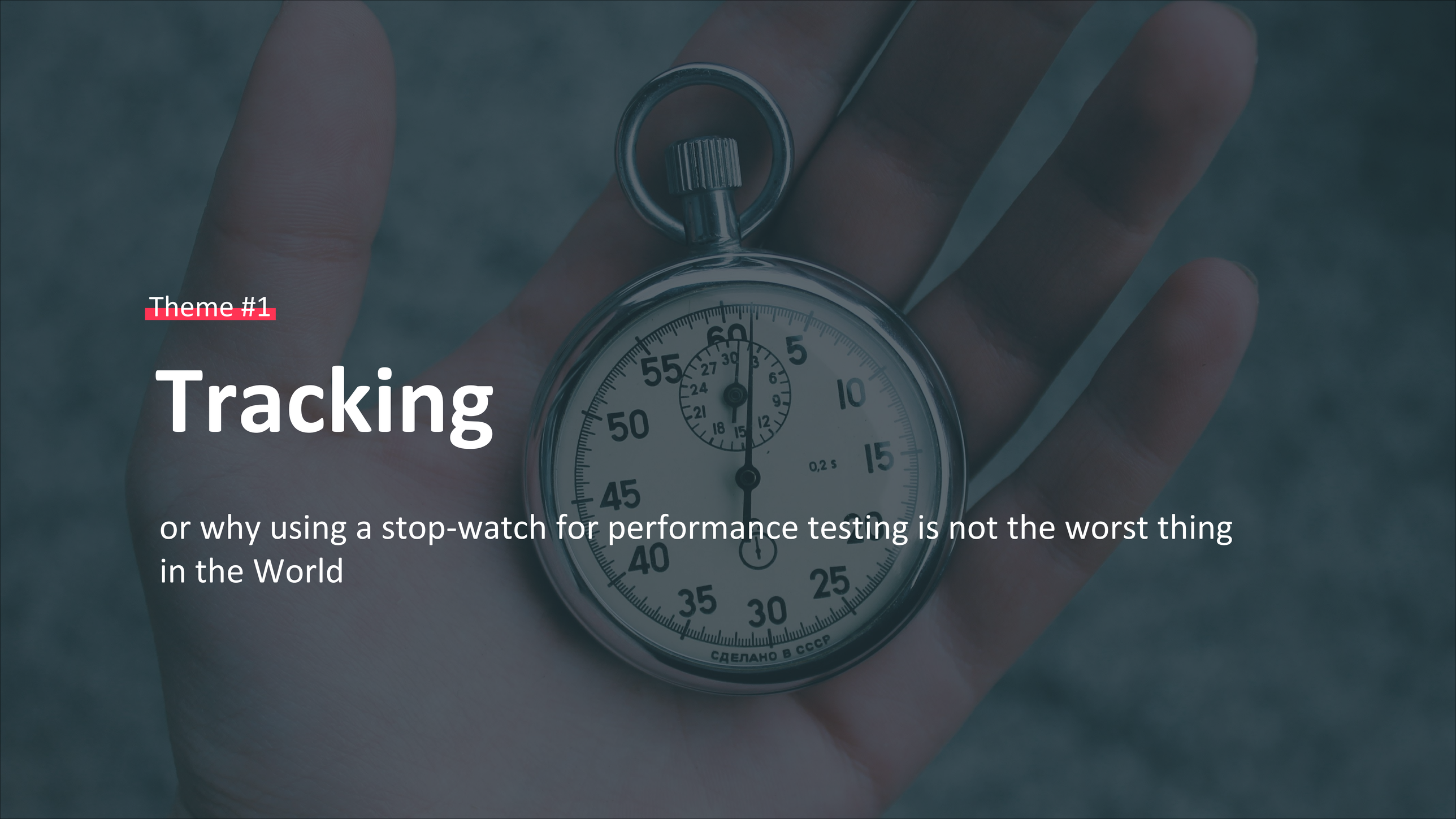
Justas Laužadis


Time to talk
about response times –
how they change over time
and how we should respond



Back to 1905

Special theory of relativity



Theme #1

Tracking

or why using a stop-watch for performance testing is not the worst thing in the World

Changes cause changes

- Architectural/infrastructural changes
- Configurational/operational changes
- Calculation logic changes
- Data pulling/design changes
- Traffic distribution changes
  - due to new features
  - due to design/UX improvements
  - due to user learning

…

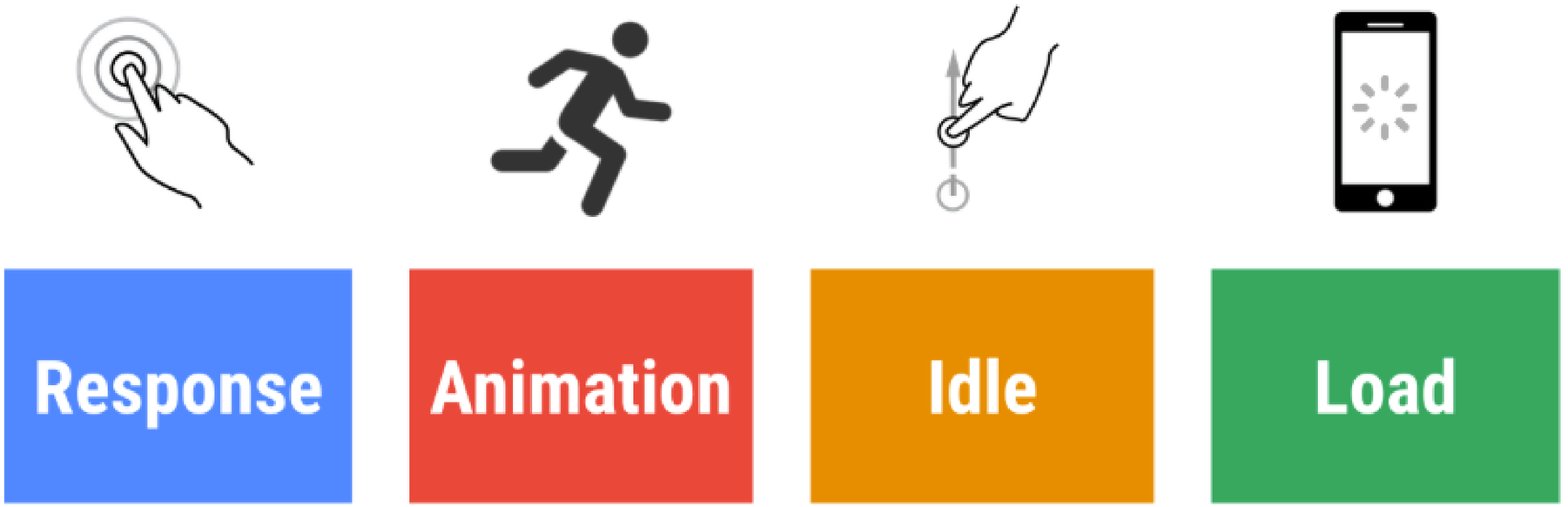
IT’S IMPOSSIBLE TO TRACK THEM ALL

High-level user-centric metrics

User Perception Of Performance Delays	
0 to 16ms	Users are exceptionally good at tracking motion, and they dislike it when animations aren't smooth. They perceive animations as smooth so long as 60 new frames are rendered every second. That's 16ms per frame, including the time it takes for the browser to paint the new frame to the screen, leaving an app about 10ms to produce a frame.
0 to 100ms	Respond to user actions within this time window and users feel like the result is immediate. Any longer, and the connection between action and reaction is broken.
100 to 300ms	Users experience a slight perceptible delay.
300 to 1000ms	Within this window, things feel part of a natural and continuous progression of tasks. For most users on the web, loading pages or changing views represents a task.
1000ms or more	Beyond 1000 milliseconds (1 second), users lose focus on the task they are performing.
10000ms or more	Beyond 10000 milliseconds (10 seconds), users are frustrated and are likely to abandon tasks. They may or may not come back later.

Following the RAIL model

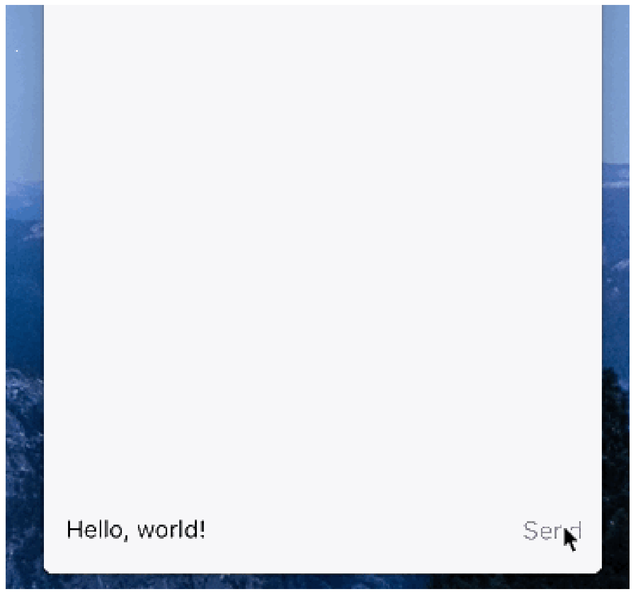
- https://developers.google.com/web/fundamentals/performance/rail

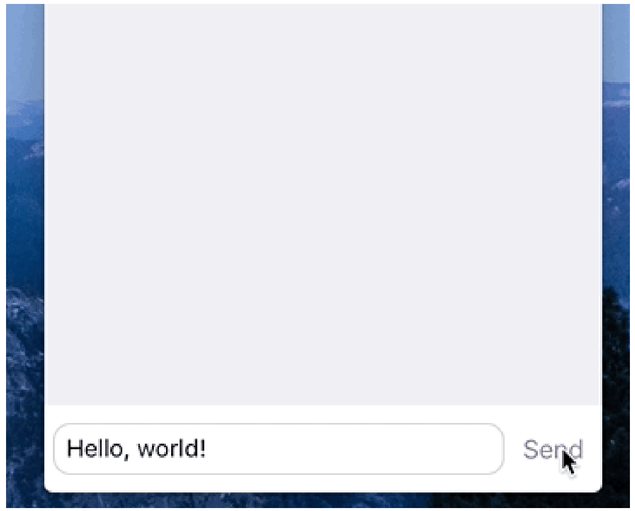


Response

- User Timing API
  (Performance.Mark, Performance.Measure)

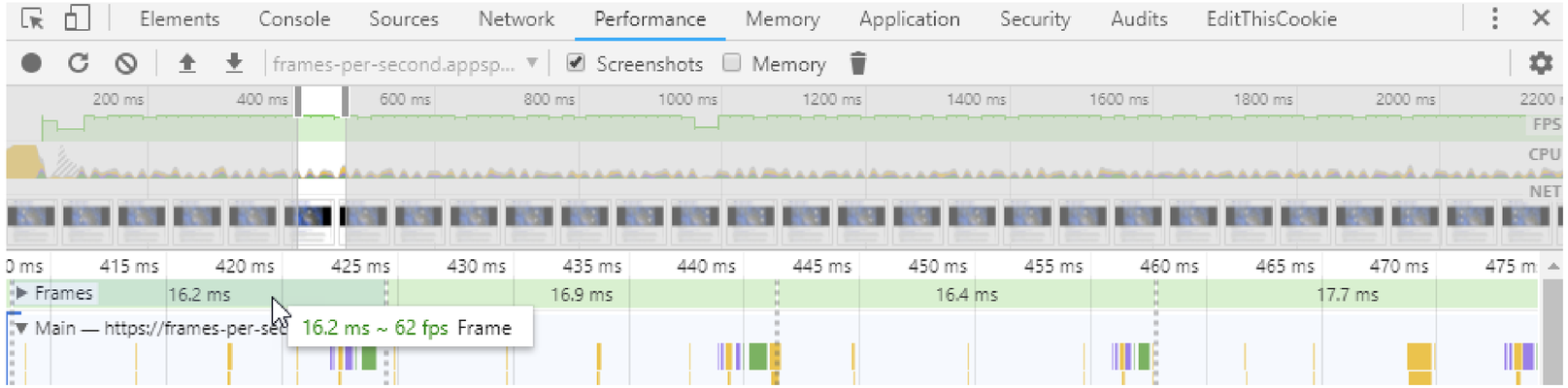
- Optimistic UI

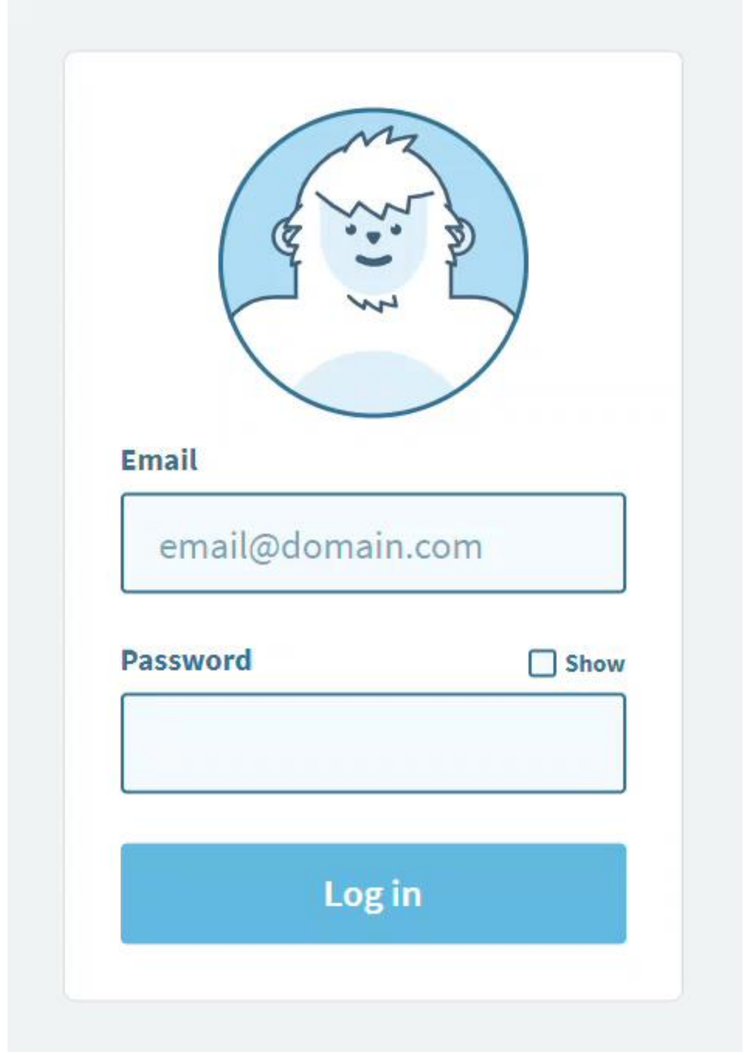




Animation

- Do not block the flow
- Parallel animations are great





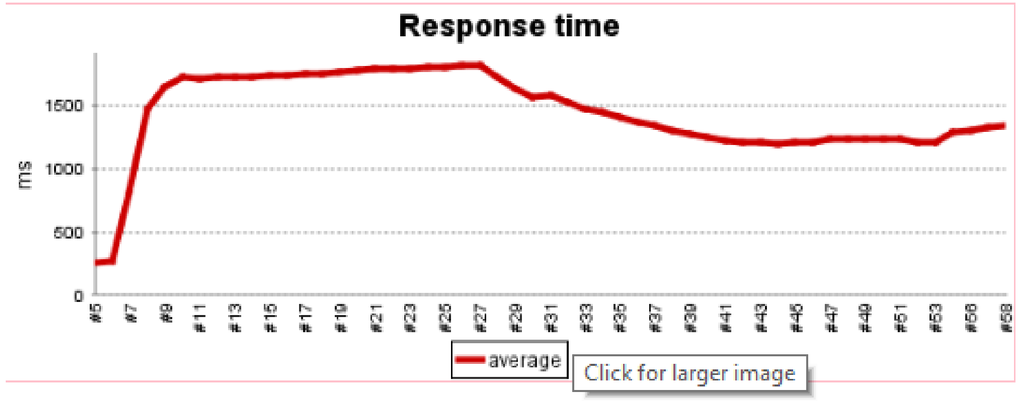
Idle

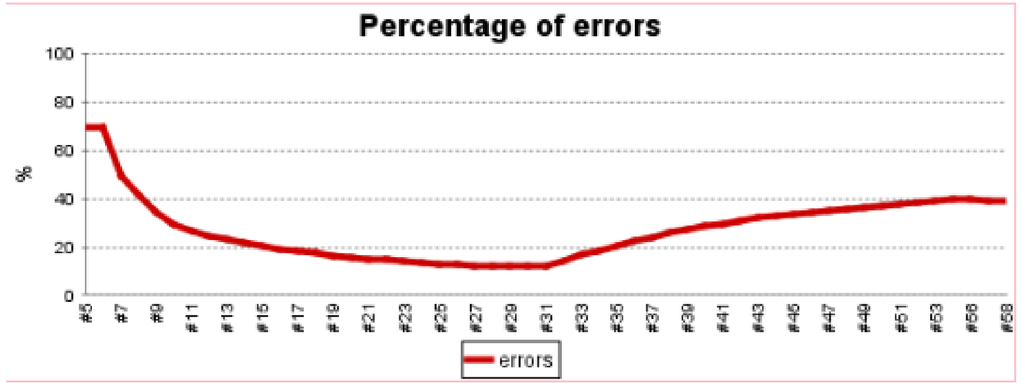
- Deliver first meaningful paint ASAP
- Maximize system’s idle time
- Run audits (Lighthouse, etc)

#LaisvesTV
„JAV žino – lietuvių programuotojai yra aukštesnė klasė“ – Aurimas Adomavičius || Laikykitės ten

Load

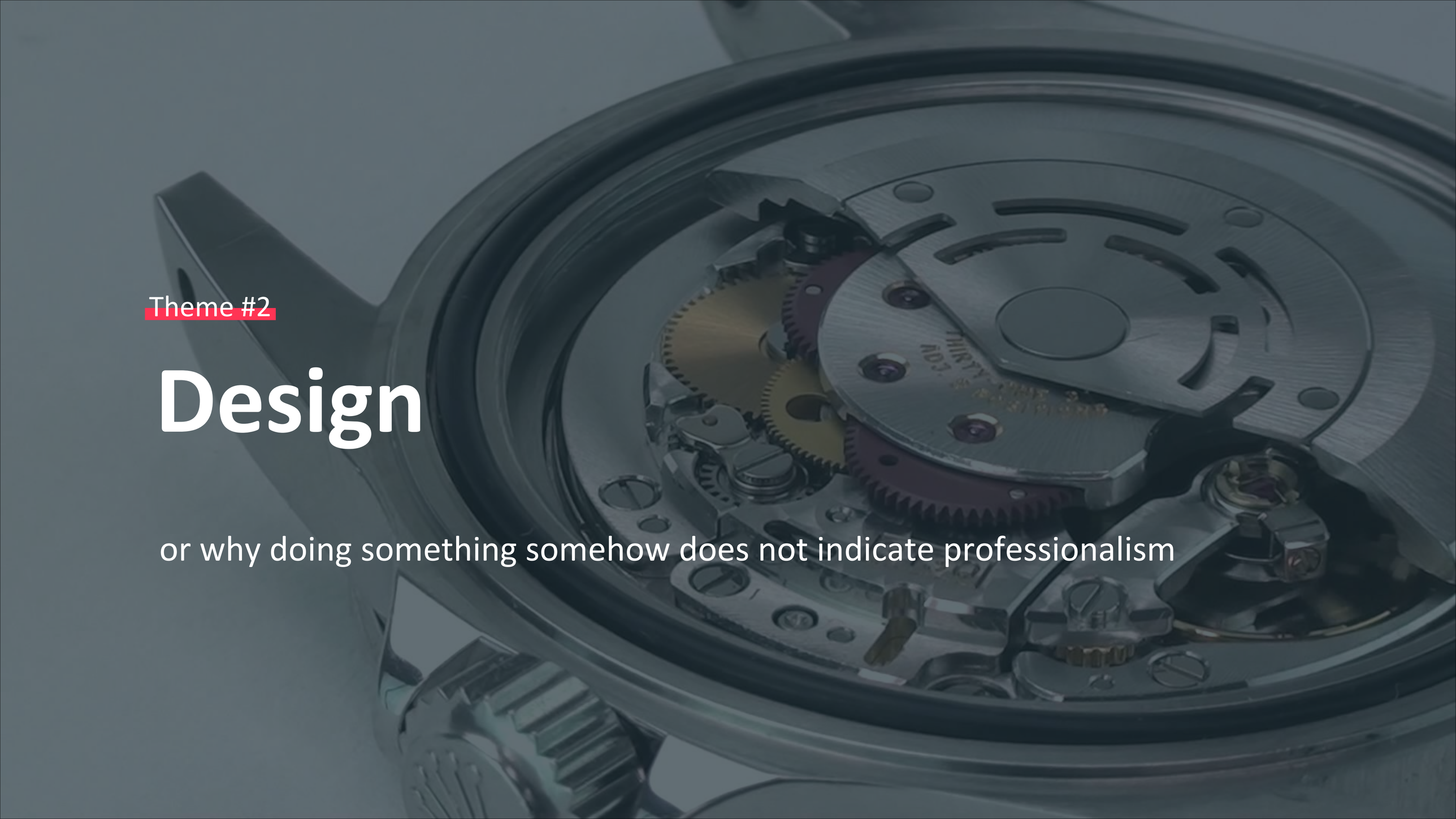
Performance Breakdown by URI: resultsBSS.jtl





Response time trends for build: "New DevOps - Test Performance BlueSkyQA #58"

URI	Samples	Average (ms)	Min(ms)	Median(ms)	Line 90.0(ms)	Max(ms)	Http Code	Errors (%)	Average (KI
Approve project	70 +8	23573 +368	3172 0	19714 +696	40634 +28	48971 0	500,200	18.571 % +2.442 %	
Check filing count	68 +8	2291 +776	515 0	1138 -3	2710 0	58535 +53624	200	73.529 % -3.138 %	
Check the potential filing number	2572 +0	323 0	6 0	31 0	756 0	16619 0	404,200,400,500	52.527 % 0.0 %	
Create a broker	2810 +9	448 +6	7 0	66 +2	603 +20	27046 0	400,500,200	50.534 % -0.162 %	
Create a class	2734 +8	202 +1	11 0	91 +1	338 +4	21673 0	400,200	52.085 % -0.079 %	
Create a client	2842 +9	330 +11	18 0	114 0	465 +19	31563 0	200,400,500,Non HTTP response code: java.net.SocketException	50.317 % -0.16 %	
Create a portfolio	2767 +8	189 +4	11 0	89 0	283 +1	23957 0	400,500,200	52.186 % -0.079 %	
Create a prospectus	2740 +8	165 +2	9 0	49 +2	220 0	23151 0	404,200,201,400,500,Non HTTP response code: java.net.SocketException	52.190 % -0.079 %	



Theme #2

Design

or why doing something somehow does not indicate professionalism

Evidence of time relativity

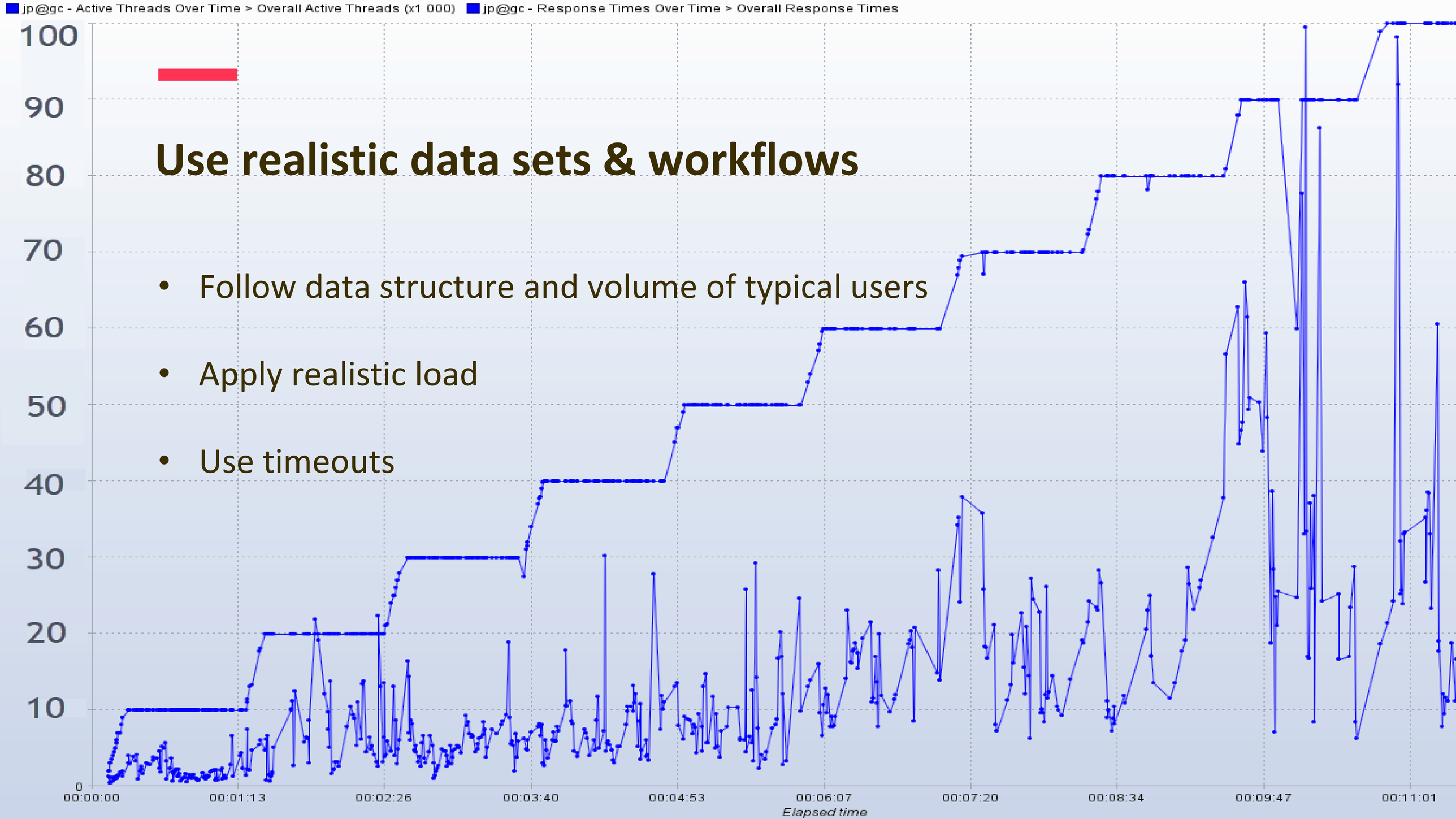
- Existing performance test framework
  - response times checked and compared every sprint
  - average times measured using one user load
- Longer response times were registered by performance tests

API request name	Before (ms)	After (ms)	Change	Change %
API Add Order Room Items	1659	1873	-214	-12,90%
API Orders - Expand order by 3 days	3166	4056	-890	-28,11%
API Orders - Decrease order by 3 days	3507	4591	-1084	-30,91%

Results after release to production

- **Timeouts** while editing relatively small orders
- Made **impossible** to use the system until hotfix introduced
- Had to **revert** functionality and investigate the causes

BE REAL(ISTIC)

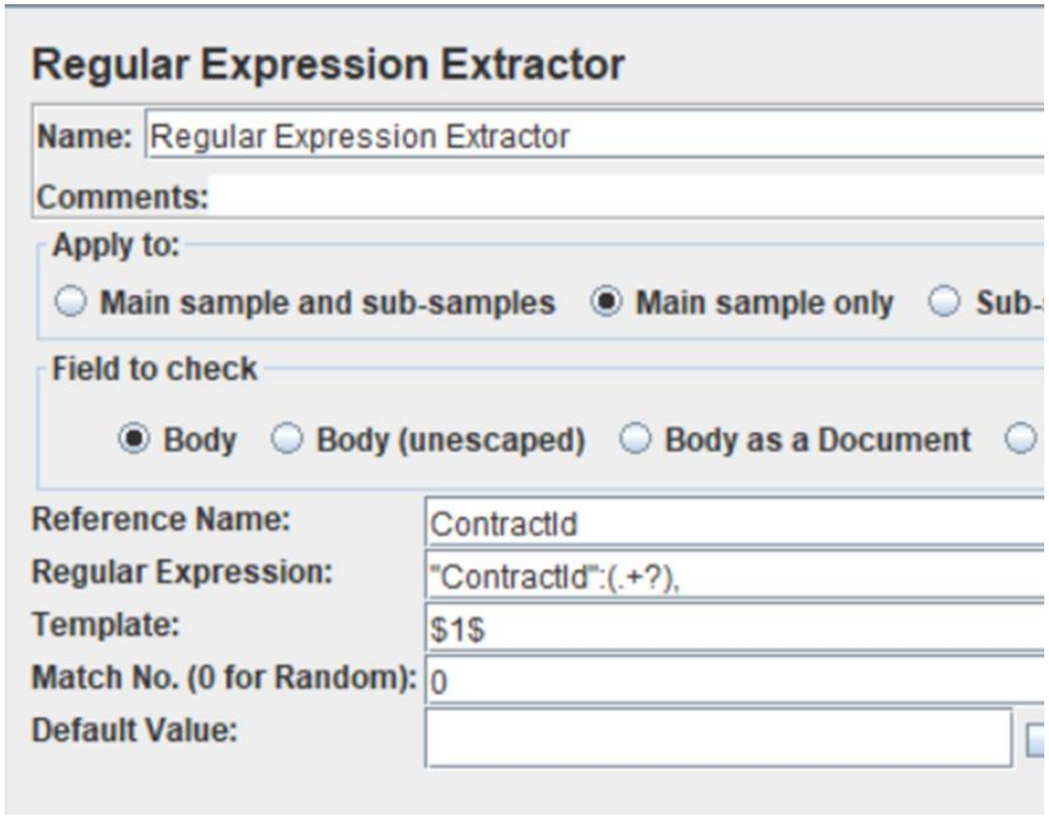


Use realistic data sets & workflows

- Follow data structure and volume of typical users
- Apply realistic load
- Use timeouts

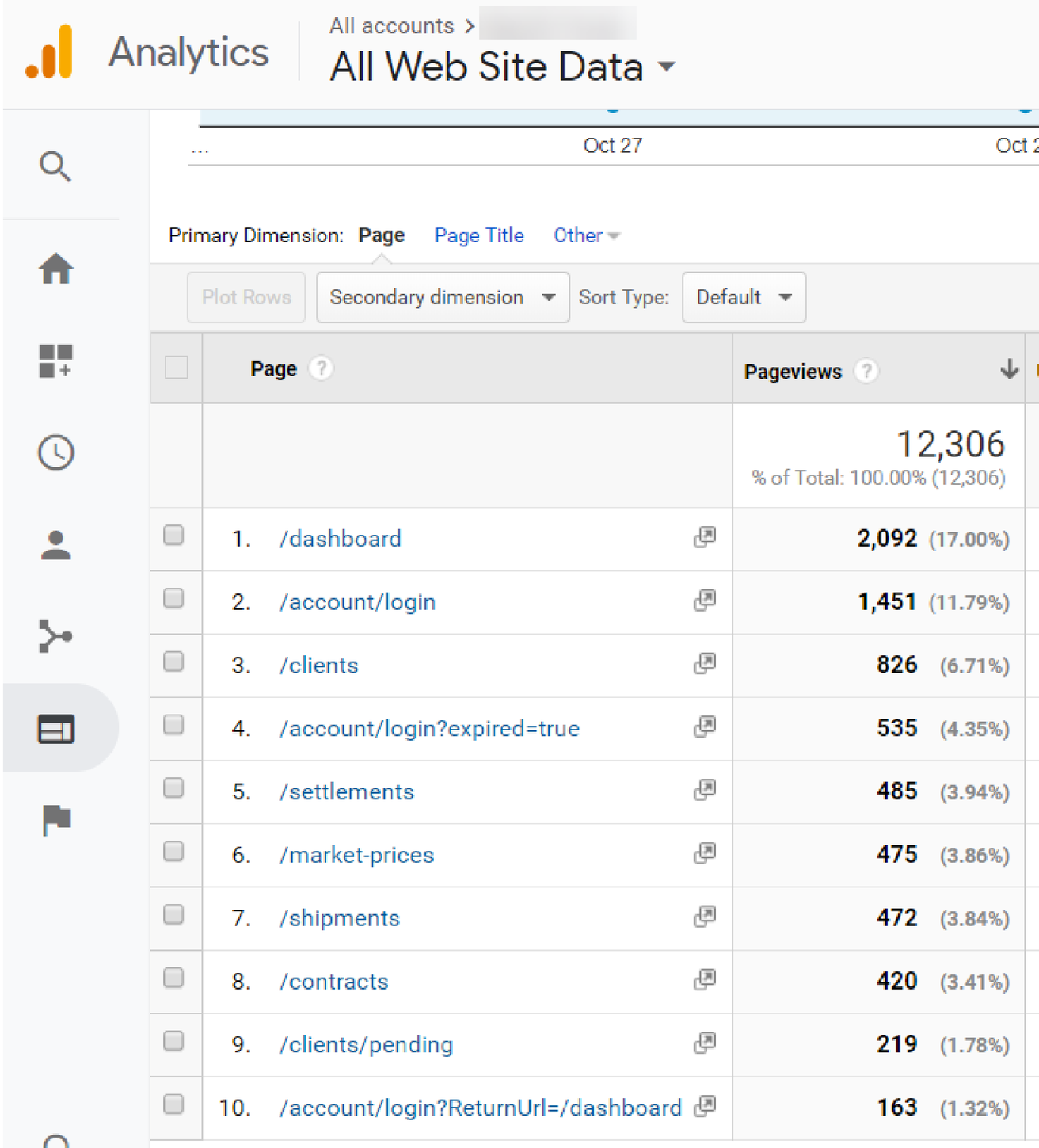
Randomize

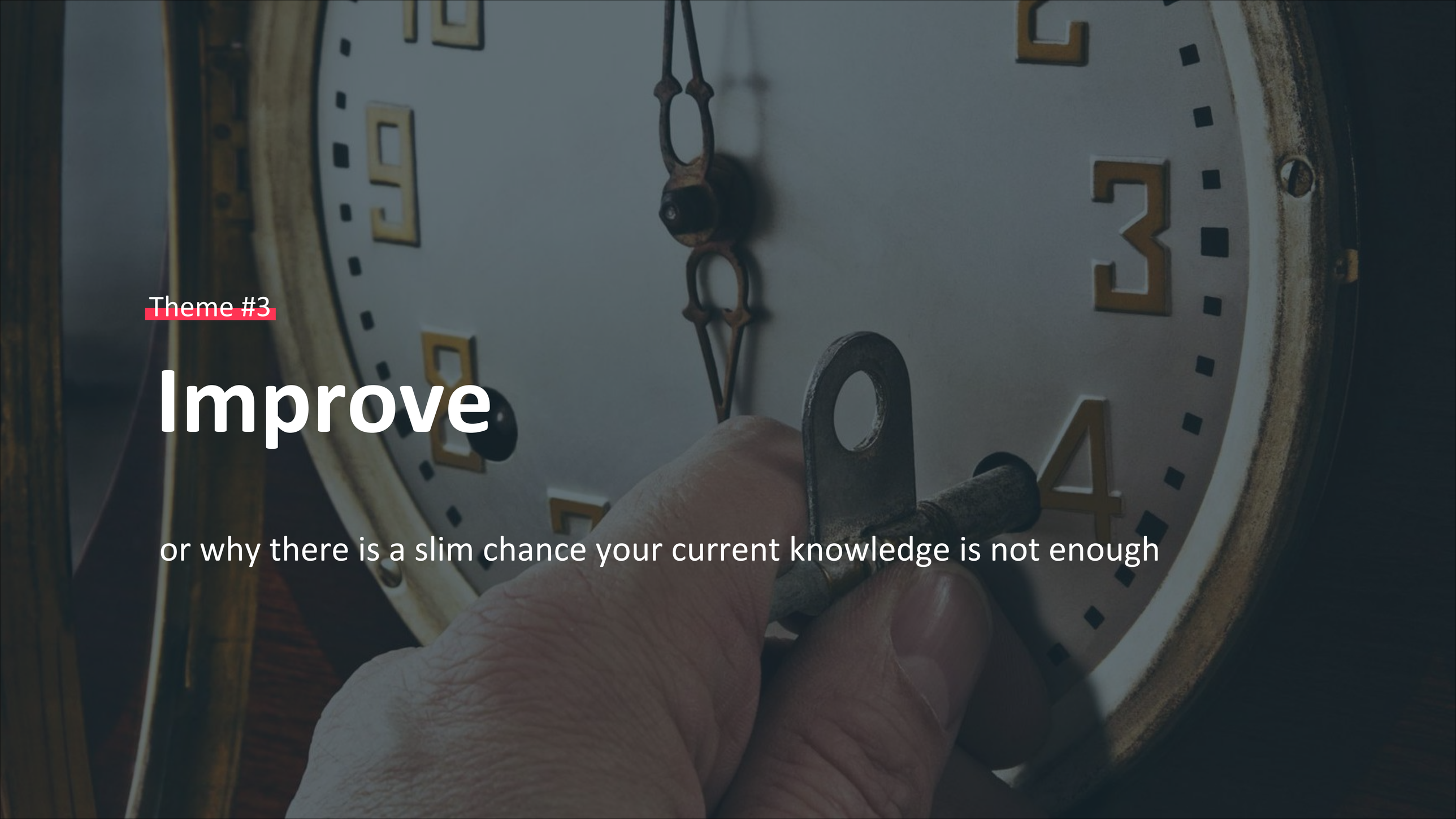
- Avoid server-side caching, but respect browser cache
- Randomizing the order → more realistic flows
- Randomizing URLs → higher coverage



Coverage

- Cover endpoints, parameters, roles
- Simulate realistic traffic distribution
- Test both API and WEB layers





Theme #3

Improve

or why there is a slim chance your current knowledge is not enough

Ask & Listen

- Ask early - as soon as you have test design in mind
- Overhear conversations
- Ask about the fix – what was the cause?

Knowing the infrastructure is critical

- Explore every corner of your environment
- Sneak into other environments

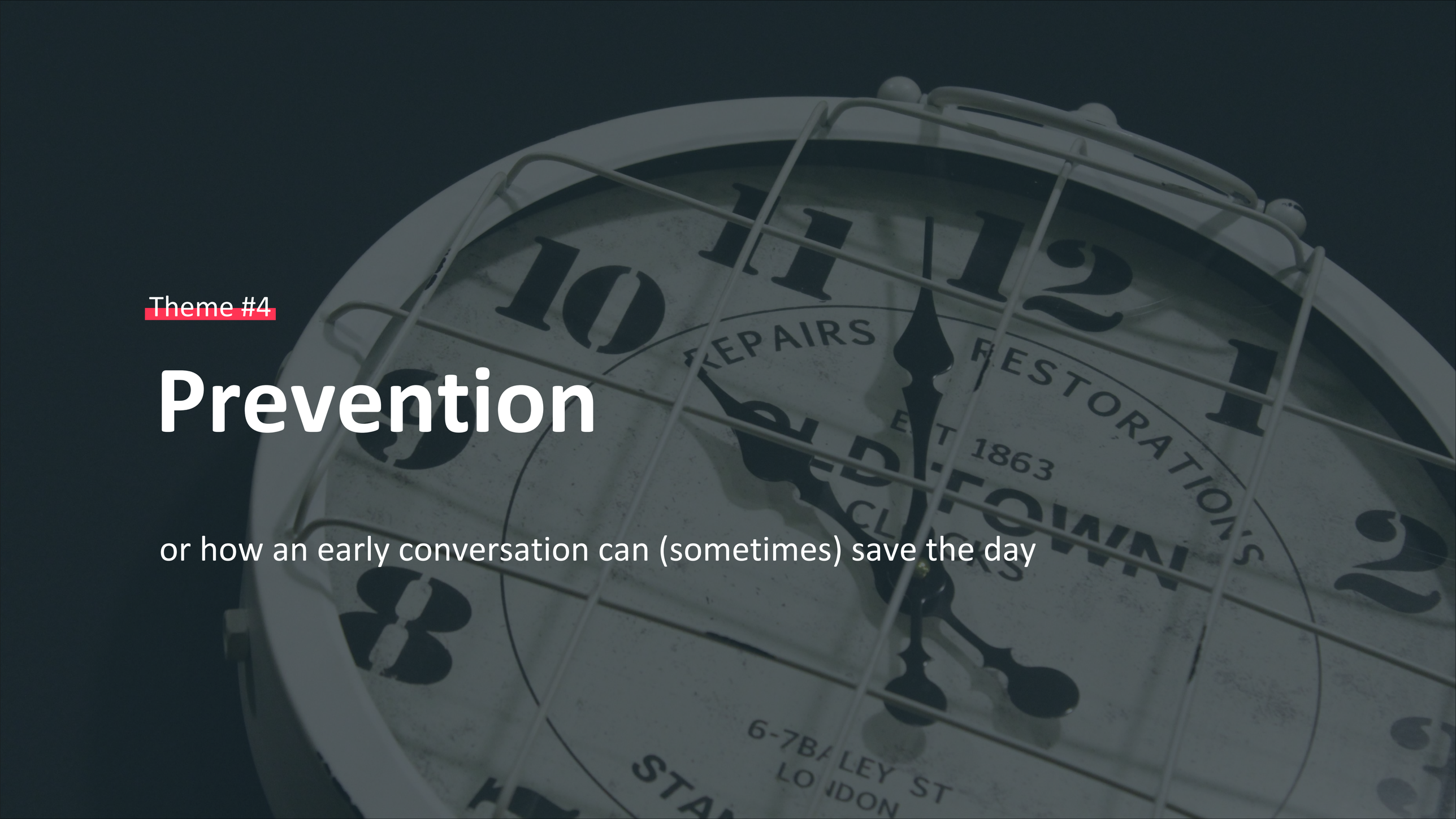
Changes between environments might give understanding

Environment #1

Label	OLD API 60 users	NEW API 60 users	DIFF
GET Contracts: all sales	4488	1948	-56.60%
GET Contracts: all sales by clientId	4140	1745	-57.85%
GET Contracts: all sales by clientId (all parameters)	4860	1785	-63.27%
GET Contracts: all sales by commodity	3042	1952	-35.83%
GET Contracts: all sales by date range	6852	1911	-72.11%
GET Contracts: all sales by status = Closed	6495	1953	-69.93%
GET Contracts: all sales by status = Historical	6655	1808	-72.83%
GET Contracts: all sales by status = Open	6707	1868	-72.15%
GET Contracts: all sales by status = Overdue	6671	1783	-73.27%
GET Contracts: all sales by status = Ready	6624	1850	-72.07%
GET Contracts: all sales by status = Settled	6729	1867	-72.25%
GET Contracts: all sales search	4448	1884	-57.64%
GET Contracts: all sales sorted asc by status	5289	1855	-64.93%
GET Contracts: all sales sorted desc by status	5318	1855	-65.12%
GET Contracts: overview	8011	2702	-66.27%
GET Contracts: unsigned	5049	1521	-69.88%

Environment #2

Label	OLD API 60 users	NEW API 60 users	DIFF
GET Contracts: all sales	1288	1439	11.72%
GET Contracts: all sales by clientId	1333	1357	1.80%
GET Contracts: all sales by clientId (all parameters)	1217	1400	15.04%
GET Contracts: all sales by commodity	1443	1439	-0.28%
GET Contracts: all sales by date range	1194	1301	8.96%
GET Contracts: all sales by status = Closed	1215	1321	8.72%
GET Contracts: all sales by status = Historical	1229	1297	5.53%
GET Contracts: all sales by status = Open	1207	1319	9.28%
GET Contracts: all sales by status = Overdue	1278	1339	4.77%
GET Contracts: all sales by status = Ready	1258	1347	7.07%
GET Contracts: all sales by status = Settled	1306	1377	5.44%
GET Contracts: all sales search	1345	1321	-1.78%
GET Contracts: all sales sorted asc by status	1290	1588	23.10%
GET Contracts: all sales sorted desc by status	1261	1452	15.15%
GET Contracts: overview	1752	1919	9.53%
GET Contracts: unsigned	1286	1452	12.91%



Theme #4

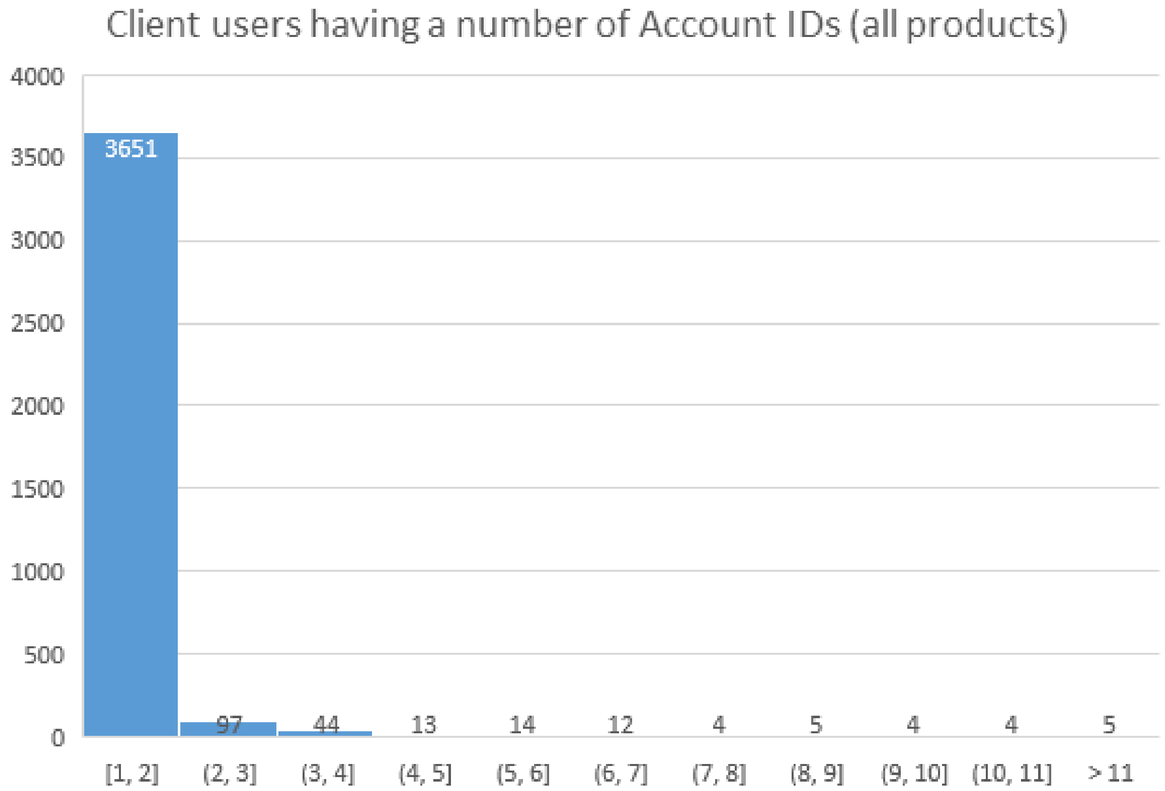
Prevention

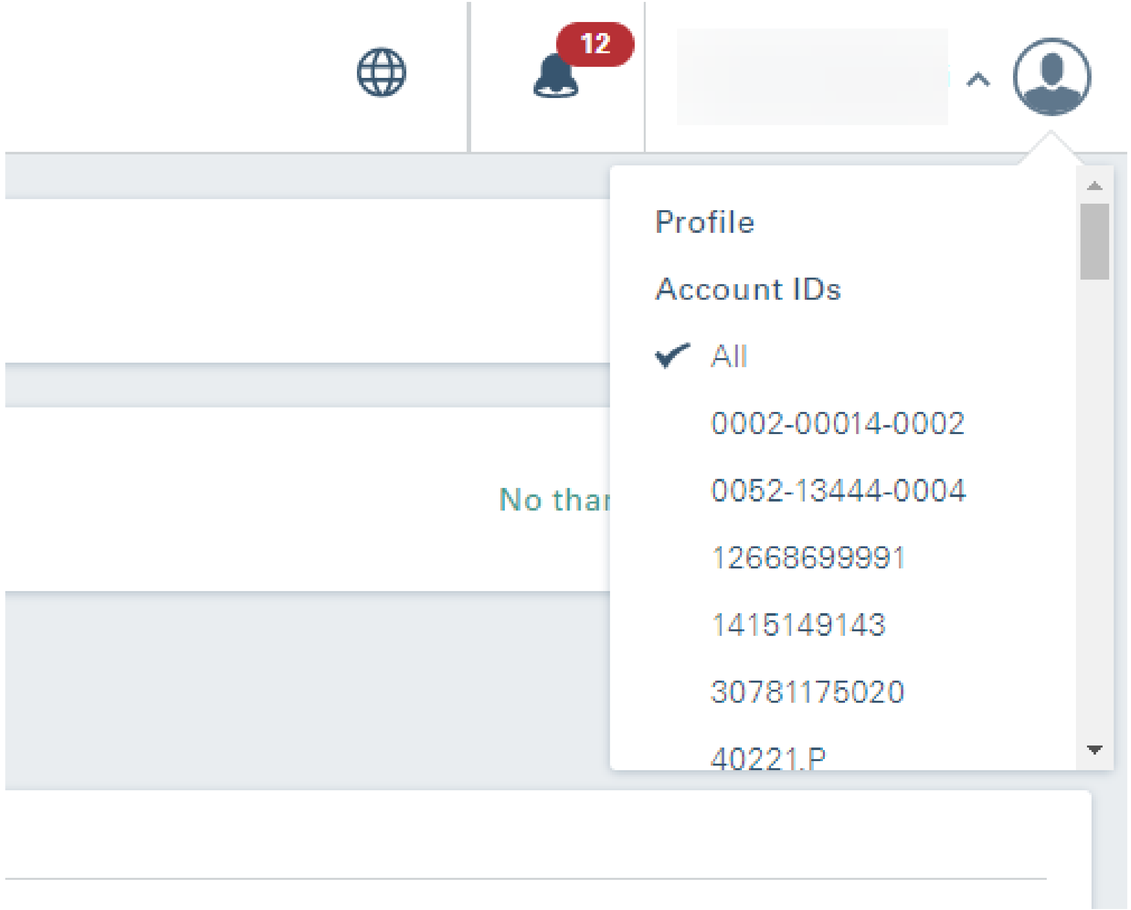
or how an early conversation can (sometimes) save the day

Prevention

- Know the requirements
- Consider performance impact as early as possible
  - initial workshop
  - backlog refinements
  - design is critical
- Run audits
- Instruct the developers about the issue and how to test it

Story about a developer who tested with heavy data





Advocating for tech debt

- Think of how to demo tech debt/performance improvements
- Think of how to demo newly introduced metrics
- Educate the team and the clients about the performance

Finally

Summa summarum

or what I was trying to prove

Summary

- Performance should be tracked continuously
- Test design is critical – be realistic
- Learning from DEVs and infrastructure is key to improvement
- TEs should take ownership of addressing the performance

Supporting tools

- Chrome developer tools
- User Timing API (or Stopwatch lib)
- Lighthouse (Chrome add-on)
- JMeter
- Jenkins

QUESTIONS?

The End