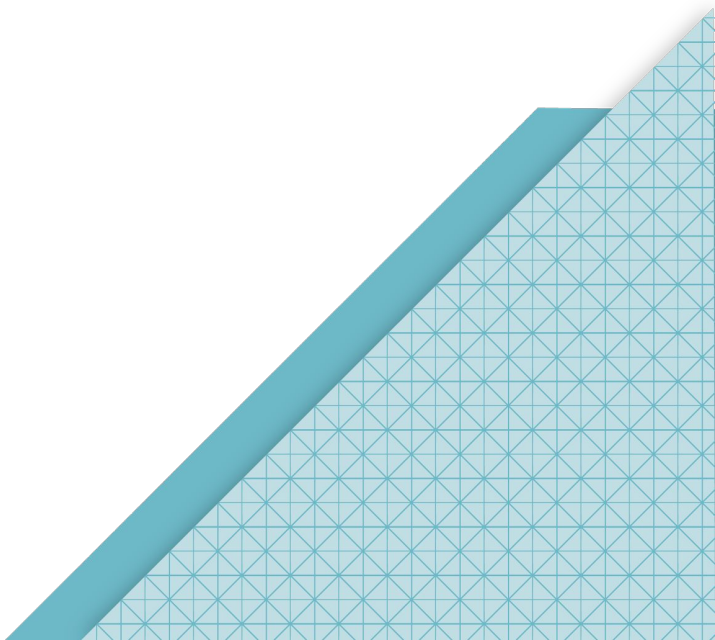
Managing Chaos

When systems get too complicated to understand



Paweł Królikowski

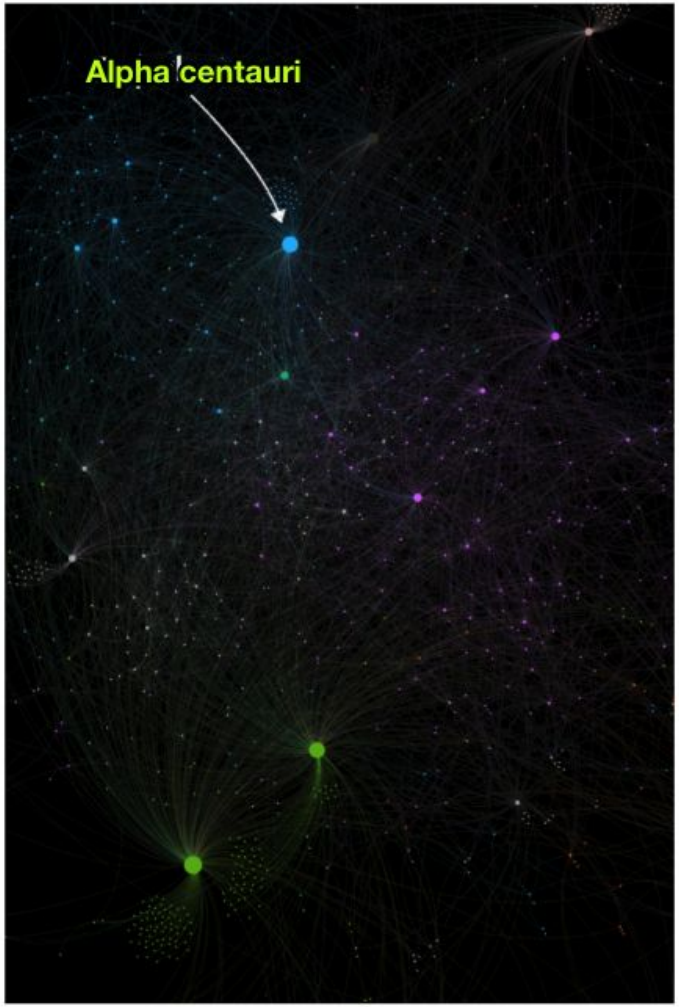
Me

- At Uber for 3 years
- Based in Amsterdam
- Started as a software engineer, moved to SRE 1 year ago
- Moved from a tiny company
- On-call + ring0
- “Still” fascinated by big/complex systems

pawel@uber.com / rabbbit@gmail.com

This presentation

- Define chaos
  - It’s fascinating!
- Present the problems
- Touch on solutions
- No answers, just possibilities
  - What worked, and what didn’t work
- Not an architecture talk - it’s about surfing the chaos, not controlling it


Alpha centauri

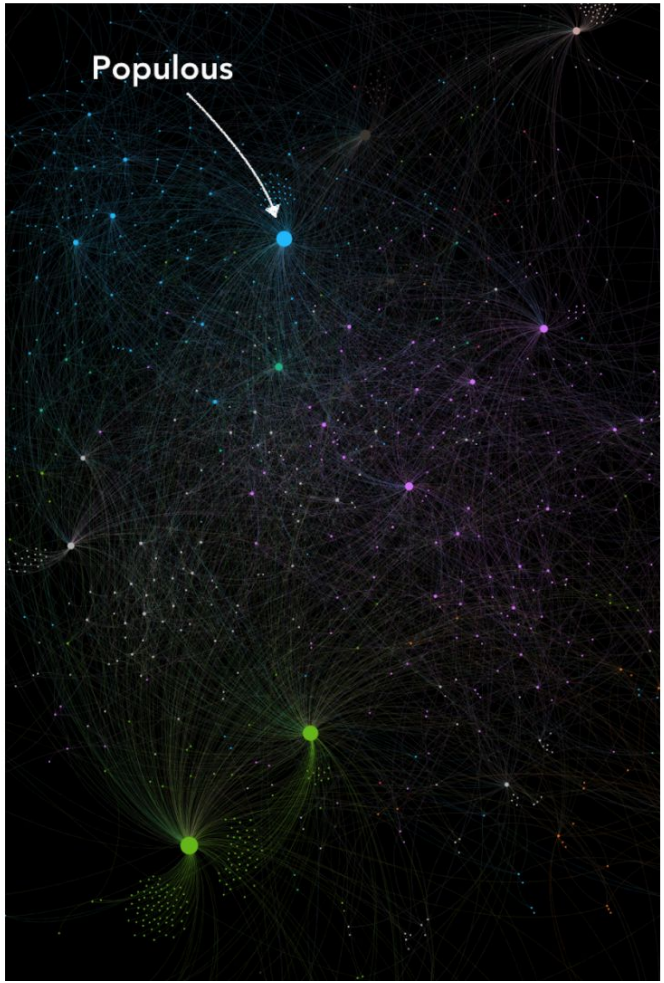
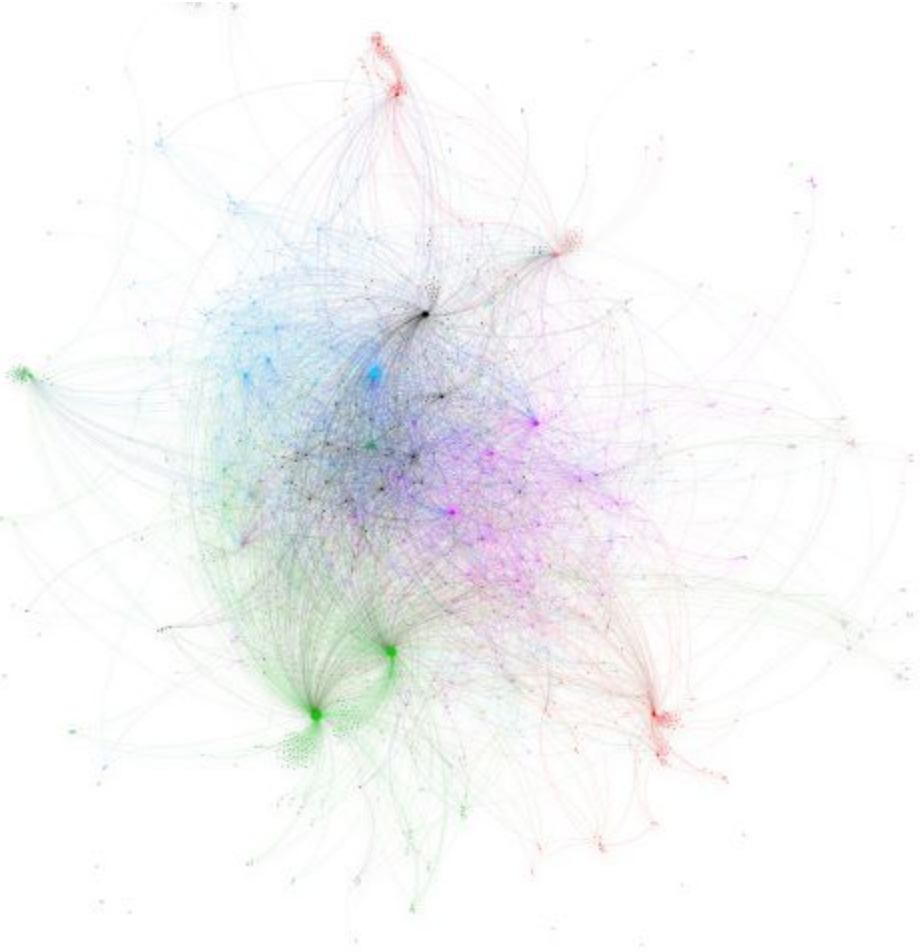
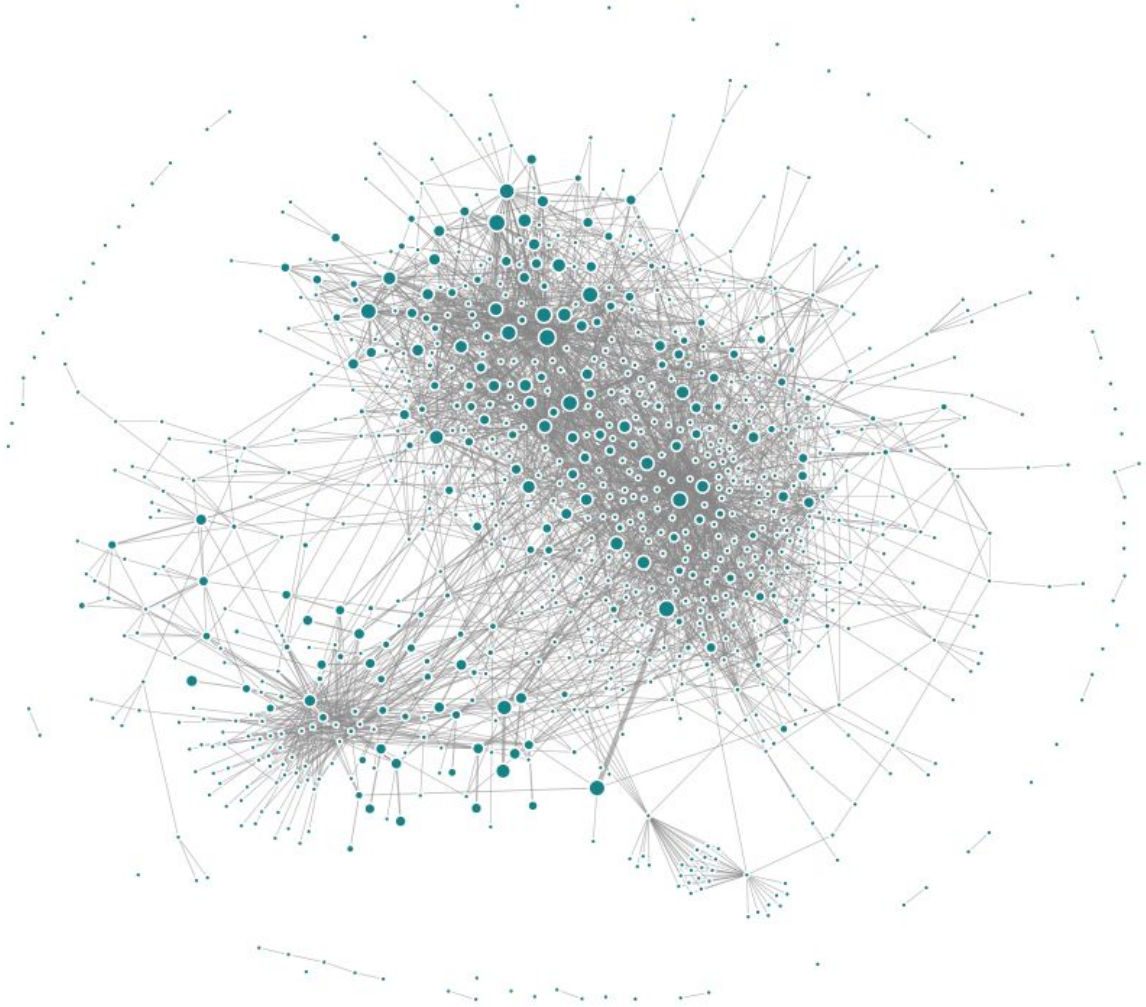
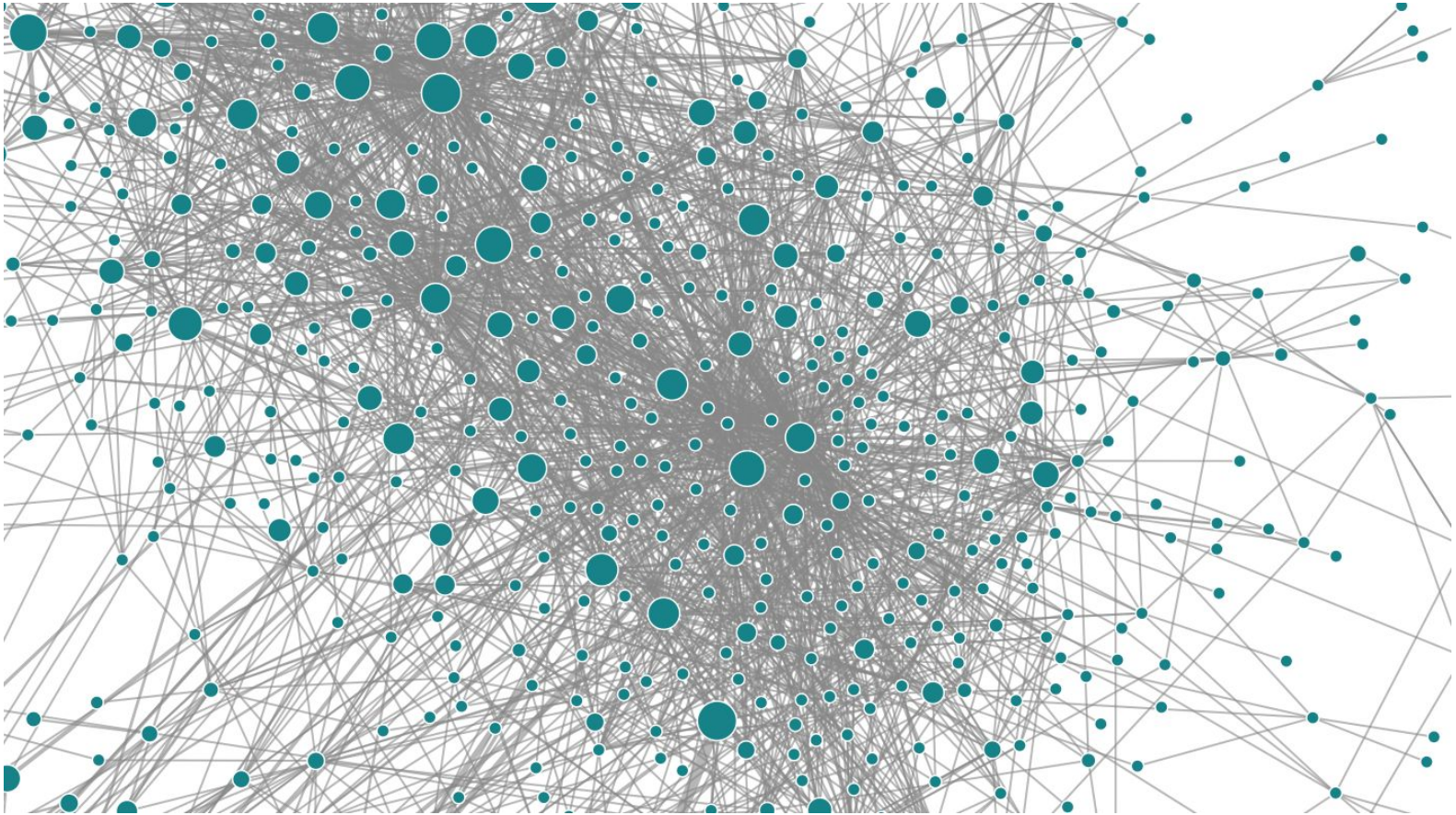
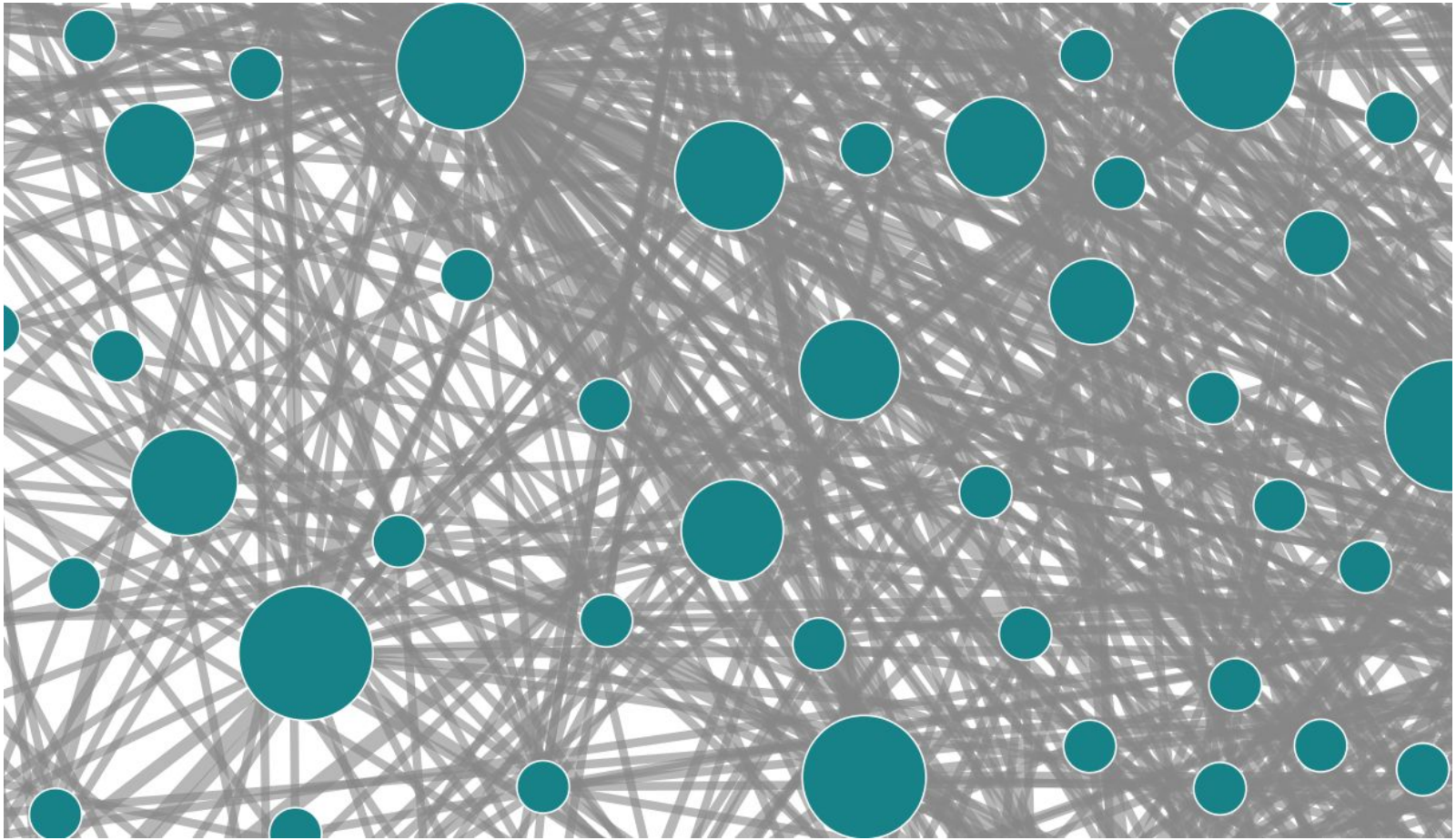


Fig. 2 A rendering of the Uber-wide service call-graph

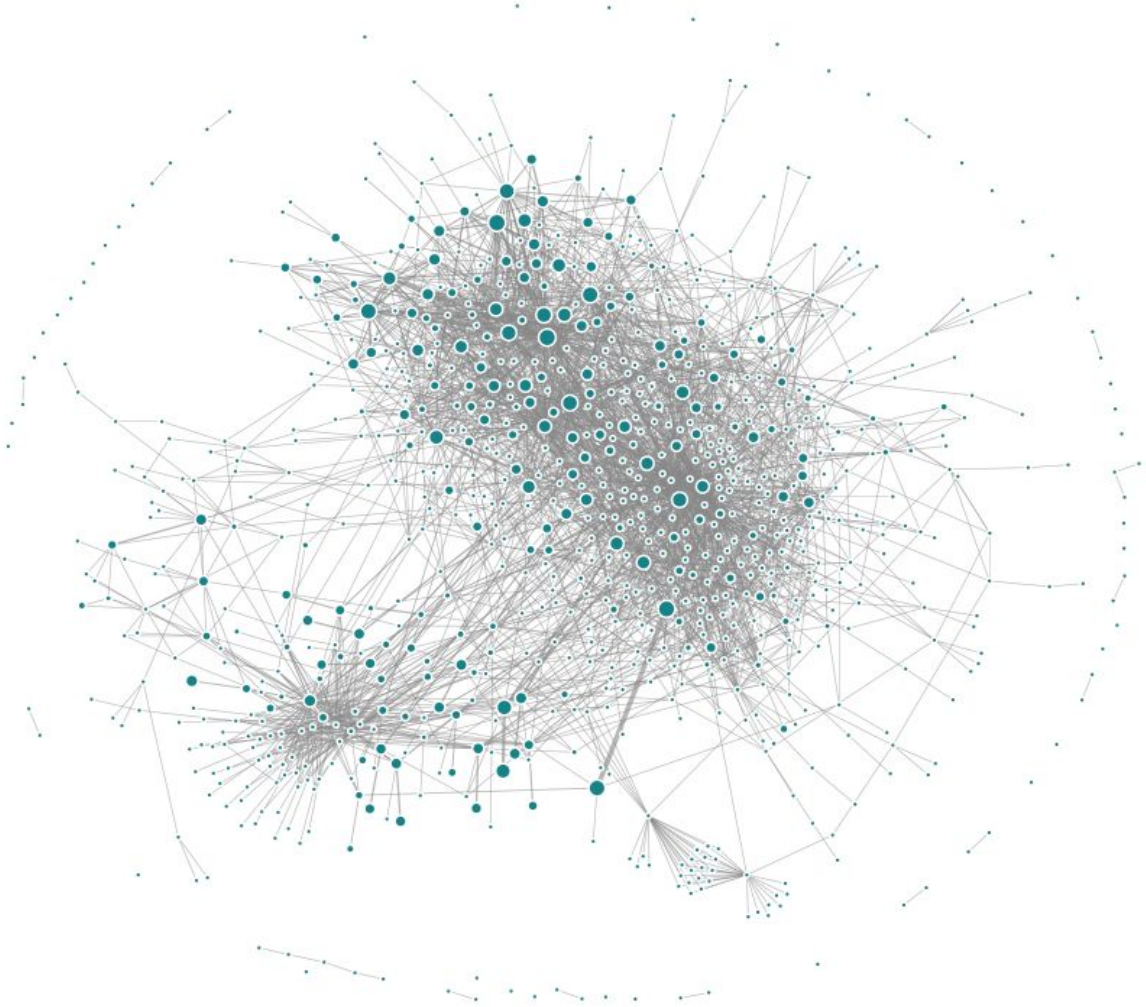


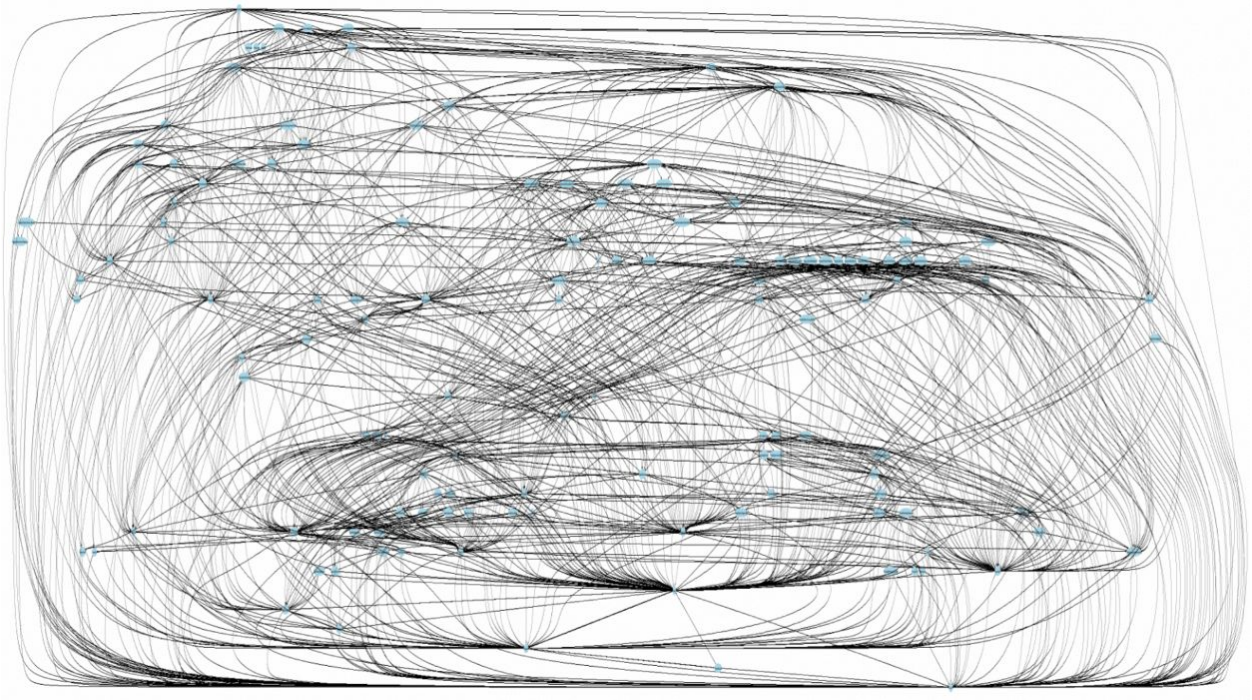







YO DAWG I HEARD YOU LIKE MICROSERVICES
...





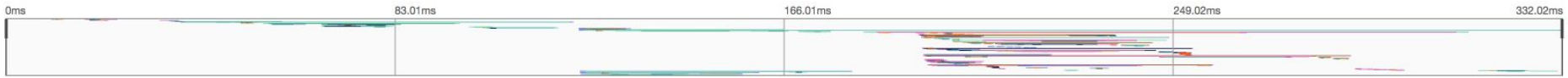
rtapi-group3-gateway: /riders/:riderUUID/app-launch

994.8ms

1879 Spans

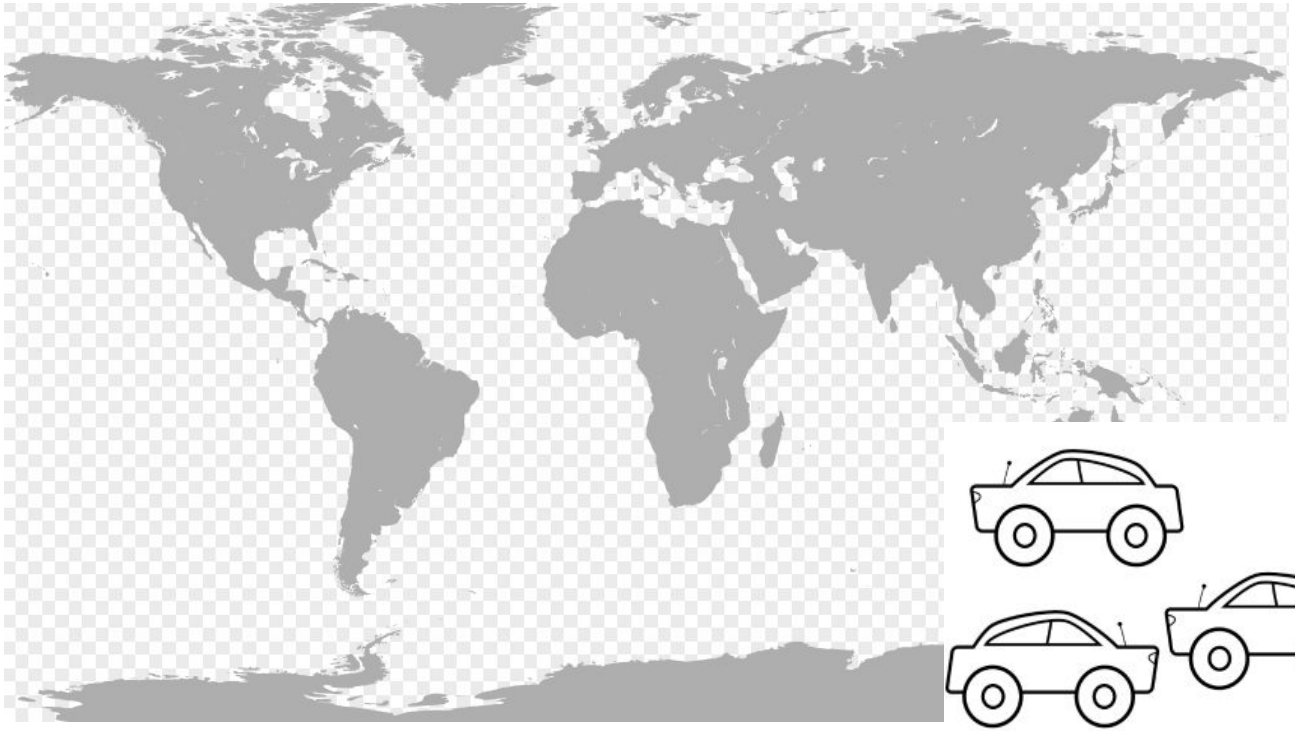
audience (6) banco (10) bankemoji (2) bar (3) cobrandcard (16) colosseum (1) dynamite (24)
dynamitestore (4) dynamitestore-backless (4) eater (1) expiringpayment (5) faresplit (3) feeder (21)
fireball (123) gopro (31) growthstore_udr (2) growthstore_udr-backless (2) hangout (9) jukebox (63)
mcs (40) mezzanine-backless (140) moneyaccess (20) multipass (8) octane (40) octanestore (14)
octanestore-backless (14) onix (196) partnerships-finserv-gateway (30) passport (84)
personalized-rt-offer-engine (2) promotion-feed-card (20) promotions (8) promotionstore (8)
promotionstore-backless (8) punch (28) ridereducation (10) rt-geofence (18) rt-msg-shared (5)
rt-products (4) rtapi-group2-gateway (247) rtapi-group3-gateway (13) safety-gateway (8) scheduled-ride (4)
scheduled-rides-gateway (14) schemaless (152) shared-rides-gateway (26) splash-mountain (4)
streamgate (83) streamgatefe (36) subscription-gateway (6) tier2-gateway (10) treatment (45)
trident (184) trufflestore (3) trufflestore-backless (4) vinyl (2) wallet (5) wormhole (2)
yellowpages (4)

Today | 7:39:55 pm
24 minutes ago



Mandatory tweet


Top Eight 4xx Endpoints
800
600
400
200
0
19:40
19:50
20:00
20:10
20:20
Top Eight 5xx Endpoints
20
15
10
5
0
19:40
19:50
20:00
20:10
20:20



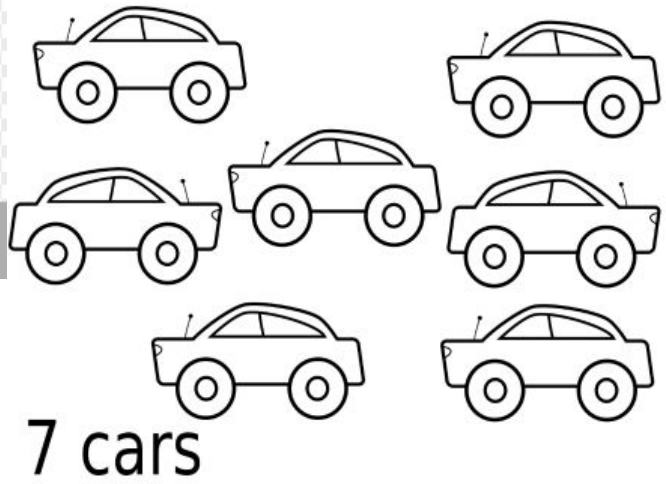


Image from: https://commons.wikimedia.org/wiki/File:Math_For_The_Young_7_cars.jpg

Problems

- How do you deploy a service?
  - Are you sure you didn’t make a breaking change?
- How do you do a breaking change?
- How do you deprecate a service?
- How do you determine the root cause of a page?
- How do you mitigate?

Plan

- Monitoring
- Alerting
- Testing
- Tracing
- Mitigation
- Conclusions
- Questions

Monitoring

- Step 1: Have metrics, have graphs, have dashboards
  - My first job: “ssh …; tail -f /var/log/*”
  - Secretly scp’ing logs nightly, building graphs with matplotlib, sending emails to the chosen ones

- Step 2:
  - Tens of millions datapoints ingested per second
  - Seventy five years of time series data queries per second

Monitoring

- Whitebox metrics
  - Business level monitoring
  - One dashboard to rule them all, then recurse

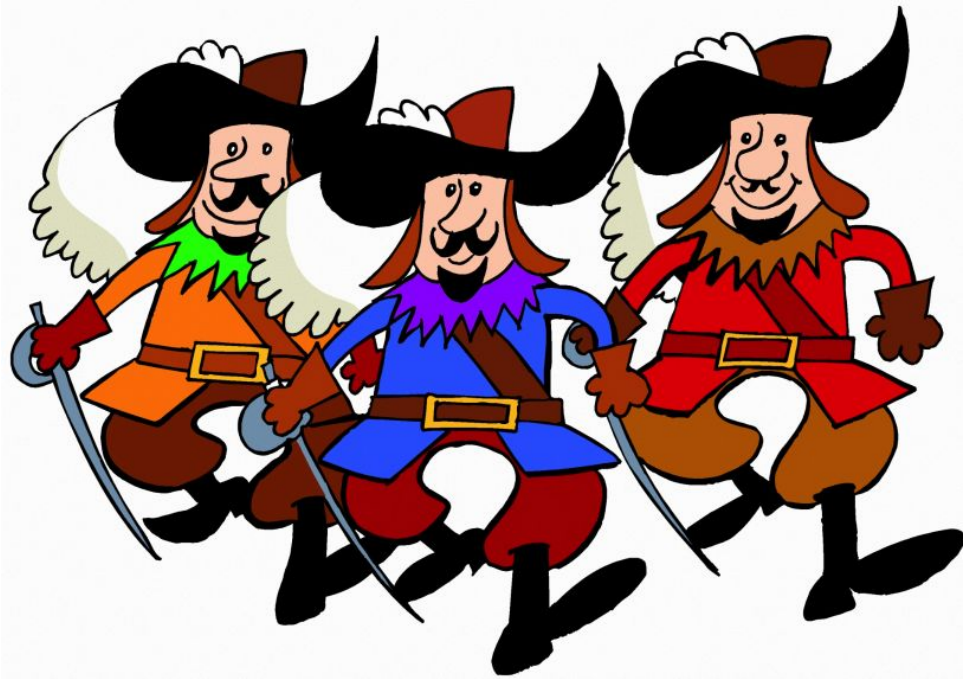
- Blackbox metrics
  - Separate deployment, in multiple 3rd party clouds
  - Continuously testing business flows
  - Per city (!)
  - Sounds easy, but:
    - How do you keep the flows up-to-date
    - How do you differentiate test data
    - How do you not overload small cities

Alerting

- How many alerts do you trigger on an outage?
- How do you prevent floods of alerts?
- How do you page “just the right” team?

- If you get paged:
  - How do you know who caused it?
  - Do you know the “events” (deploys, alerts) of your dependencies?
  - Are their dashboards generic enough for you to understand?

Testing



Staging

- How do you make everyone have staging?
- How do you make sure everyone keeps it up to date?
- With 5 services in a chain:
  - What do you do if someone leaves the master dirty? Is it page worthy? Do you block your own release?
  - What if it’s 15 services?
- Do you maintain the same capacity?
- How do you generate “real” load?

Testing in production

- Port-forwarding is a big no.
- Or is it?
  - What if you have an auth proxy in between?
  - What if it’s only for specific test users?
  - What if it’s only on whitelisted endpoints?
  - What if you can reverse the flow and send traffic to you?

Testing in production

- Can we do better?
  - Tenancies
  - Separate stores, separate flows, separate metrics
  - Testing on actual production instances, with the actual code
  - A lot of work across the stack.

Automated testing

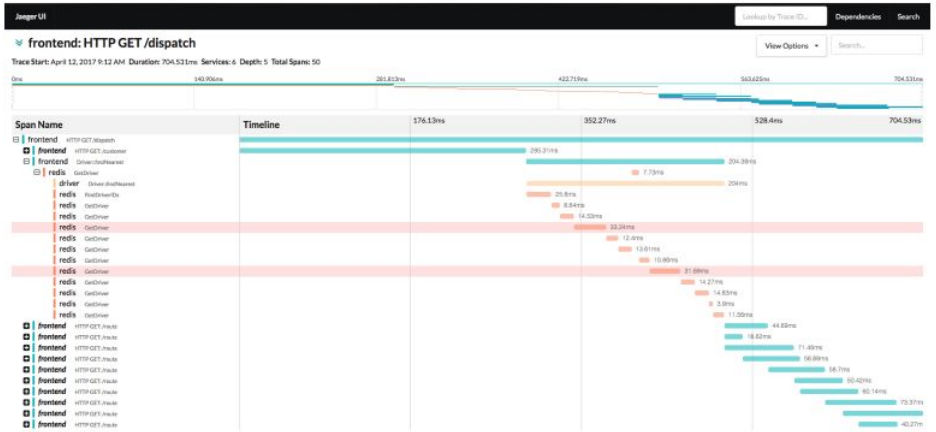
- Ideally, integration + load tests on schedule/deploy
- Can you really though?
  - Do you need tenancies?
  - For load tests, will your downstream be happy about it?
  - If you use production, how do you prevent customer impact? Can you stop automatically?
- What does “on deploy” mean:
  - Is it your deploy? Or all of your downstream services?
  - Do you trigger your tests or your upstream tests?
  - What are your downstream and upstream services?

Solution

- You need it all
- Unit tests for basic correctness
- “Individual testing against prod” for quick development
- Integration testing for confidence
- Staging environment for full coverage
- Automated tests for stability
- “Real testing against prod” for production-only issues

Distributed tracing

- Tracing is awesome
- Not just for tracing itself
- Last piece
  - Automate system knowledge
  - Automate issue discovery
  - Automate alerts
  - Automate testing
  - Automate mitigation



- Takes years to create
- Takes years to integrate
- Takes years to teach to use

Mitigation

- Prevention will fail:
  - red/black deploys, canaries, stable APIs, feature flags, gradual rollouts, spare capacity, automated rollbacks - all necessary, but will fail

- Most undervalued skill
  - Fight your instincts!
  - Most surprising interview question
  - Does it matter if it’s fixed, what matters is its impact on the user
  - Every minute counts
  - Can you fix an issue in a minute, at 3 a.m.?
  - Mitigate!

Mitigation

- Have tools at hand
  - At 3 a.m., you should know what's happening in 60 seconds
  - At every level of the call graph
- Have runbooks
  - Degrade rather than fail, or fail-open
- We use OODA loop
  - When acting, have tools in to act in 30 seconds
- Train:
  - Exercise failures - Ring0

Mitigation - Ring0

- On-call for datacenter failovers
- Paged automatically for Blackbox metrics failure
- Implicit “Incident Commander”
- And yet - impossible to understand the failure - how do you handle this?

- Training, training, training
  - In different “roles”
  - Know your tools/dashboards
  - Weekly exercise

- Our runbooks: lockdown everything, failover data centers

Not covered

- Unit/Chaos testing
- Incidents:
  - Postmortems
  - Postmortem follow ups
  - Incident reviews
- On-call health
- Library upgrades/migrations/depreciation
- Capacity planning

Not covered, but important

- SLA enforcement
- Accountability ⇔ correct incentivisation

Conclusion

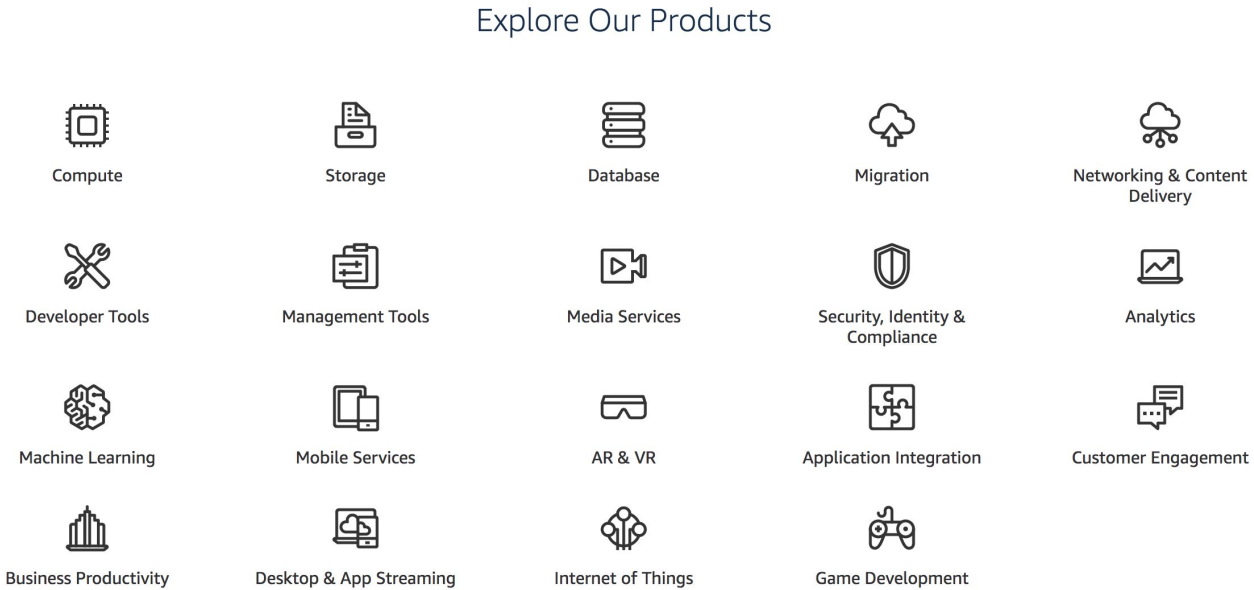
- It’s never boring :)
- Your mileage might vary
- Tools, tools, tools
  - Meta-tools: tools to help you use tools better
- Engineering culture is just as important
- Automate everything (with more tools)

Conclusion

The (maybe?) solution:

Conclusion

The (maybe?) solution:



Questions?

Or:

- pawel@uber.com / rabbbit@gmail.com

Blank slide