Deceived by monitoring

Nikita Salnikov-Tarnovski

@iNikem

Plumbr

Me

- Nikita Salnikov-Tarnovski, @iNikem
- Java developer for 16 years
- 7 years mainly performance problems solving
- Master Developer at Plumbr

Plumbr

What is monitoring

“monitoring and management of performance and availability of software applications [with the goal] to detect and diagnose complex application performance problems to maintain an expected level of service”.

Wikipedia

Plumbr

Huh, WAT?

- Observe the state of the system
- Understand is it “good” or “bad”
  - If “bad” make it “good”
- Make it “better” in the future

Easy Metrics

- CPU usage is 90%
- Free disk space is 34GB
- There is 2M active users on site
- Average response time for application X is 1s
- During last 24h we had 578 errors in our logs
- We have 7 servers died in last 4 hours

Plumbr

Problems

- Lack of context
- Misaligned goals

Goals of the application

- The goal is not to use X% of CPU
- And not to keep disk mostly empty
- And even not to be fast

Real goal

- Satisfy customer’s need
- Meet business goals

Real metrics

- You have to observe application from the point of view of your users
- Can they achieve their goal?

The simplest useful monitoring

- Observe real user’s interactions with your application
- Note failed interactions
- Record response times

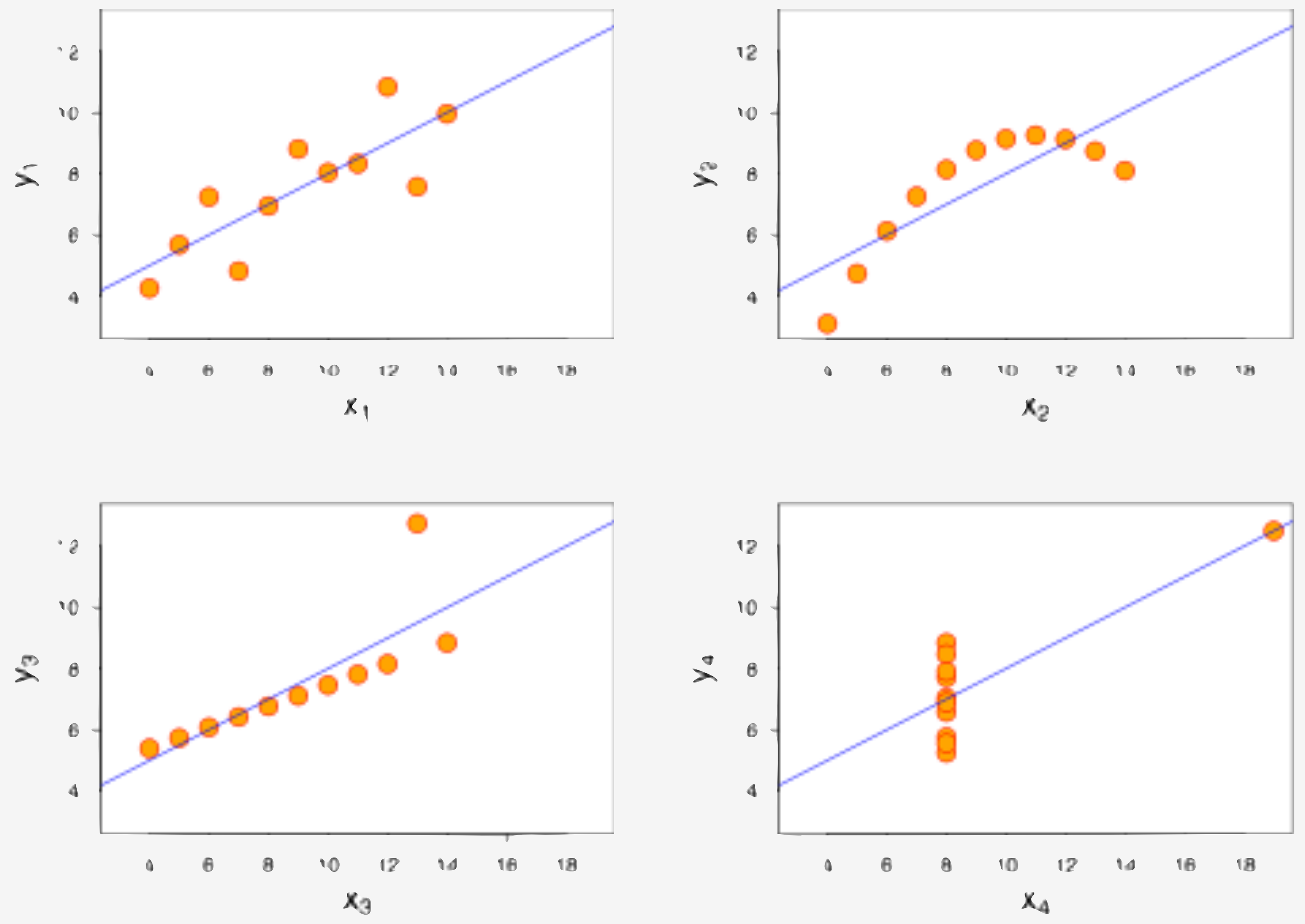
Plumbr

The biggest fallacy

“Average response time is an useful metric”

Plumbr

Anscombe's quartet



Percentiles

Most page loads will experience the 99%'lie server response

Gil Tene, How NOT to measure latency

Percentiles

Q: How many of your users will experience at least one response that is longer than the 99.99%’lie?

A: 18%

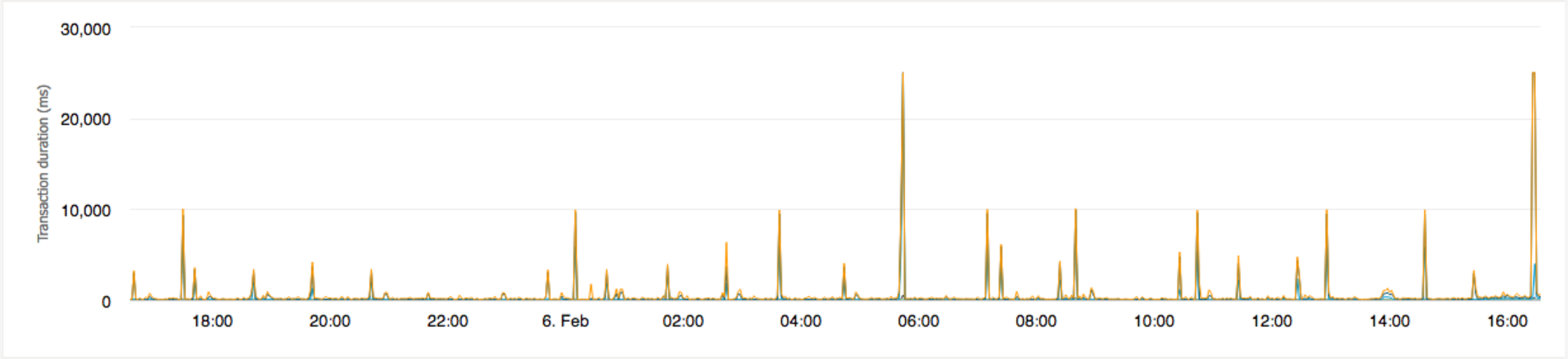
Gil Tene, How NOT to measure latency

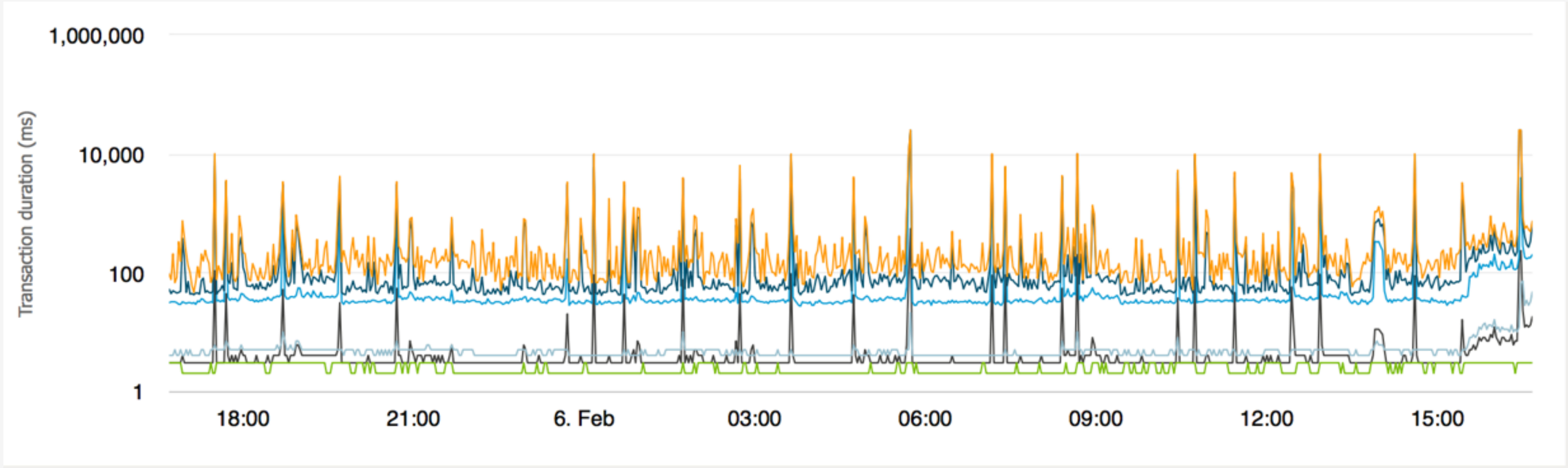
Plumbr

Percentiles

- Always record your maximum value
- Forget about median/average
- Follow your 99%’lie or higher
- Plot them on logarithmic scale

Plumbr


30,000
20,000
10,000
0
Transaction duration (ms)
18:00
20:00
22:00
6. Feb
02:00
04:00
06:00
08:00
10:00
12:00
14:00
16:00


1,000,000
10,000
100
1
Transaction duration (ms)
18:00
21:00
6. Feb
03:00
06:00
09:00
12:00
15:00

Dichotomy of metrics

- Are users happy with your application? - direct metric
  - Great for alerts and health assessment

- CPU/disk usage/errors in logs - indirect metrics
  - Great for debugging and alert prevention

Plumbr

That was about fixing

- What about improving?

Plumbr

Planning performance

- Compete with actual business feature
- Know when to stop

Plumbr

This or that?

- You have to explain to your manager why performance/ resilience is important
- Use your user happiness metric as a proxy

Not all requests are equal

- Group requests by consumed service and initiated user

Plumbr

Suits and beards

- Let business people decide which services and which users are more important
- Then you don’t need to prove the importance of any performance fix any more :)

Plumbr

Suits and beards

- And you have a perfect priority for improvements
- That actually makes sense to your manager!

Plumbr

When you talk to a suit

- “How many operations can fail”
  - “Are you stupid? Of course 0!”
- “How much time can the system be down”
  - “Are you kidding me? No downtime!”
- “How fast must operations be”
  - “What a question is this? As fast as possible!”

Plumbr

Now you have a price tag

- “This errors happens twice a week for 1 user. Should I spend 2 days fixing it?”
- “Can we have 15 minutes downtime every Sunday 3AM when we have 0 users?”
- “Should I spend 100K to move 99.99% latency from 800ms to 500ms?”

Plumbr

Conclusion

- Technical metrics are so indirect they are almost harmful
- User “happiness" is the common ground between engineers and managers

Plumbr

Solving performance problems is hard.
We don’t think it needs to be.

Plumbr

@JavaPlumbr/@iNikem
http://plumbr.eu