Your Benchmark is Invalid!

Christoph Mönch-Tegeder

2ndQuadrant
http://www.2ndquadrant.com/

2016-11-03

2ndQuadrant®
PostgreSQL

At every other Customer

- Is my server fast enough?
- How much CPU/RAM/IO do we need?
- Can we measure that?

Let's run pgbench

- tps = 382.827727

Let's run pgbench

- tps = 3817.126237

Let's run pgbench

- tps = 1869.938199

Let's run pgbench

- tps = 382.827727
- tps = 3817.126237
- tps = 1869.938199
- Look at that precision!

Understanding pgbench

- results depend on data set size, connections, . . .
- benchmark for internal locking, buffer or IO efficiency
- can use custom scripts for application workload
- and check out the logging options

Lesson 1

- naive use of pgbench does not show anything

Lesson 1

- naive use of pgbench does not show anything
- understand your workload

Lesson 1

- naive use of pgbench does not show anything
- understand your workload
- and benchmark for that!

Response time matters

- 0.1 s to 1 s for interactive workflows (Nielsen)
- user gives up after a few seconds
- industrial applications, payments, . . .
- contracts & SLAs
- end-to-end measurements

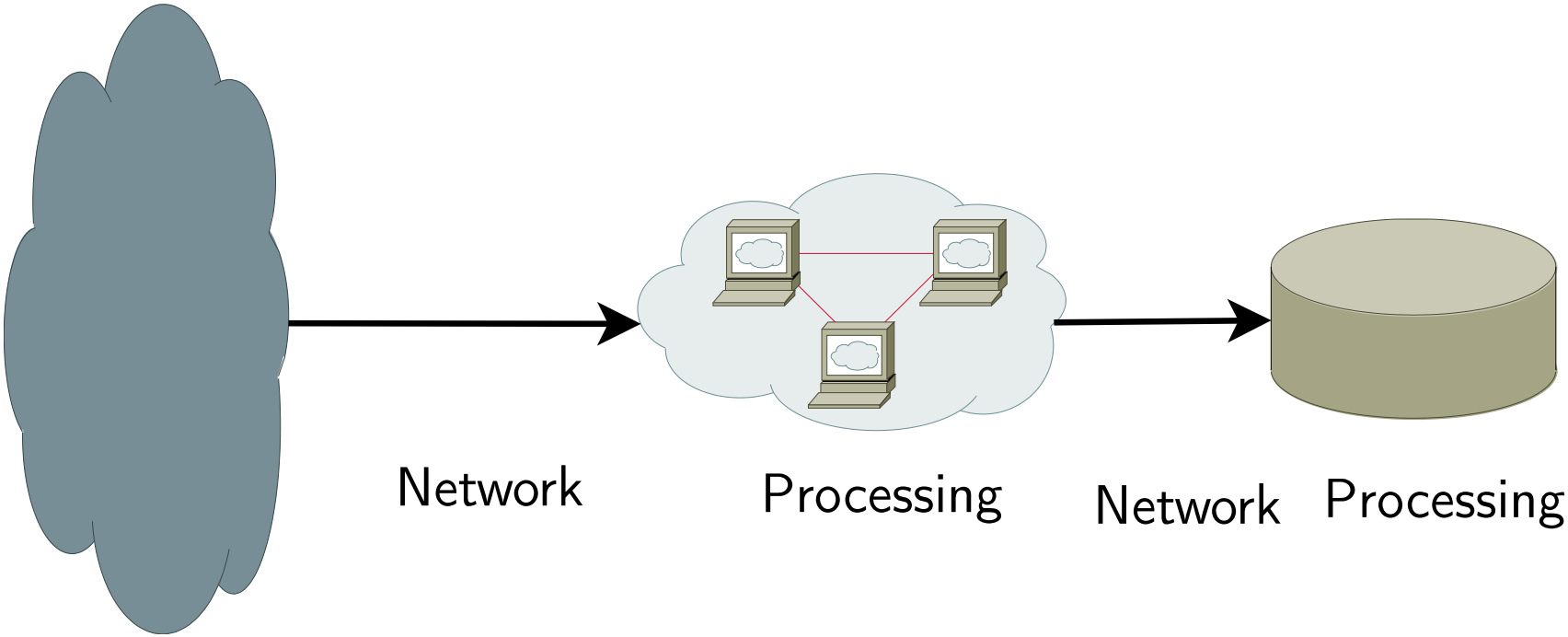
How to measure

- of course it's fast on your computer

How to measure

- of course it's fast on your computer
- with production-sized data set?
- production configuration?
- production load?
- realistic network?

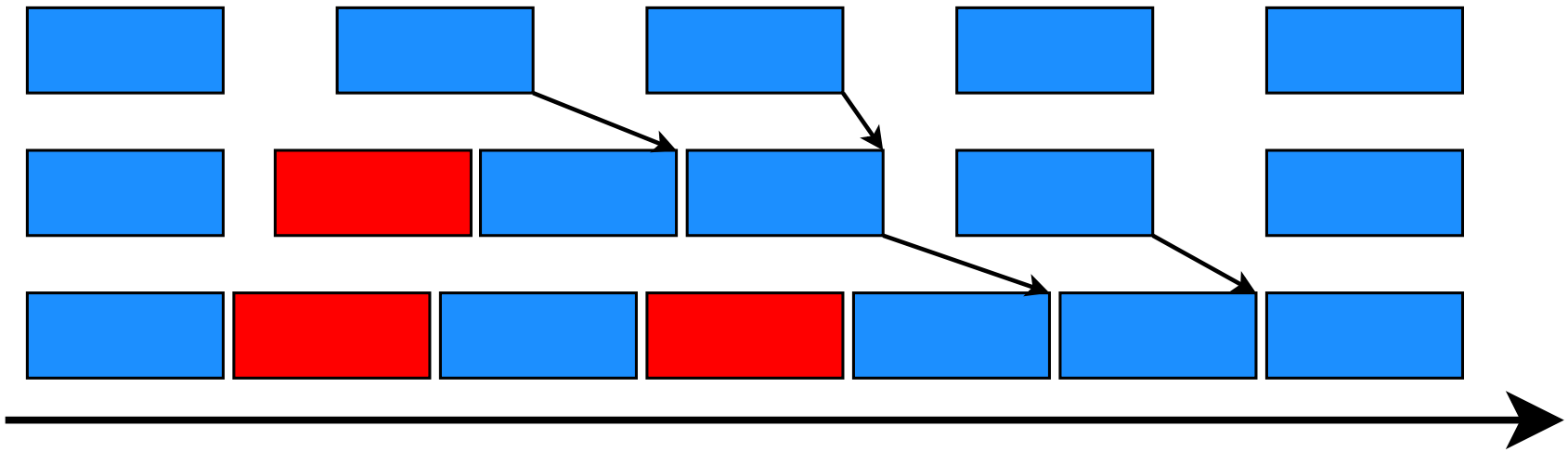
Standard Web App



Bottlenecks

- if network and application already take 150 ms. . .
- . . . no database magic will save your day
- higher utilisation: response time jitter

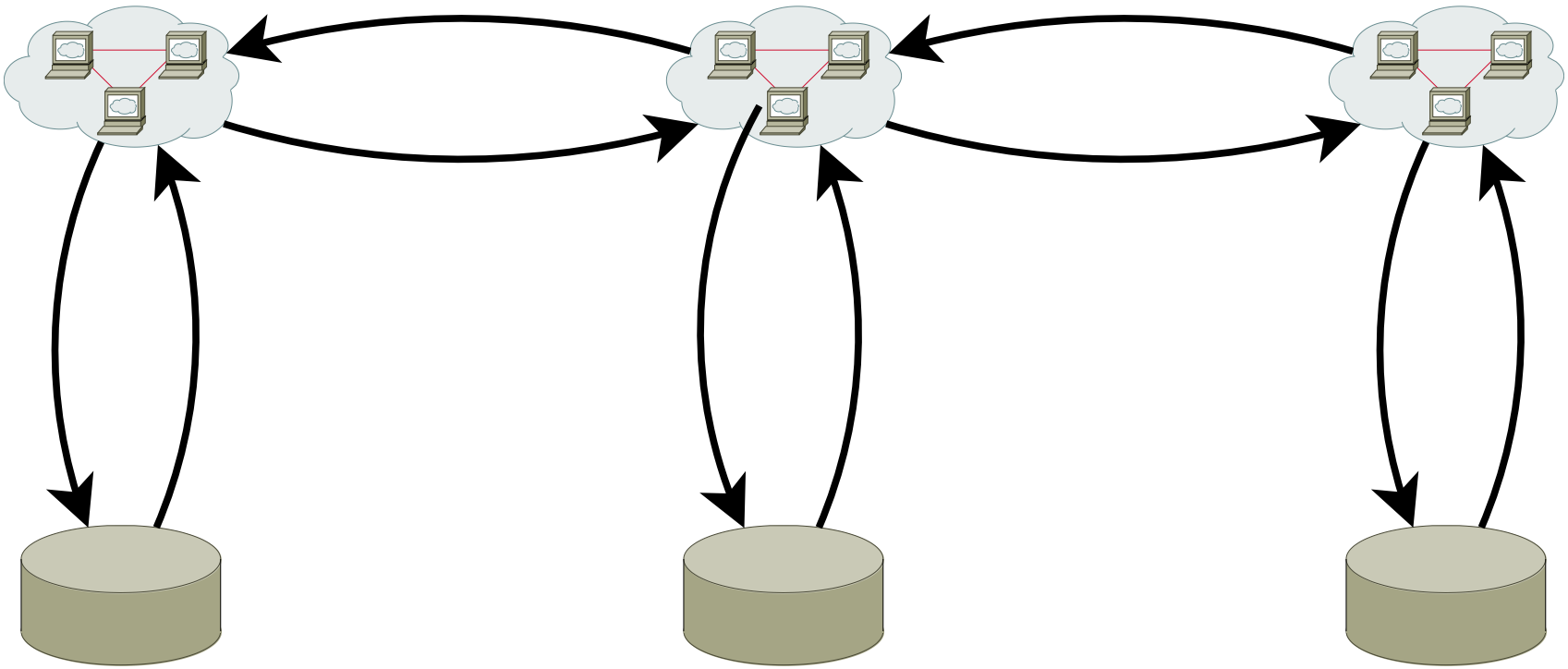
Queueing



Queueing

- even below 100% load, adding work increases response times
- when hitting the capacity limit, response time skyrockets
- *The Hockey Stick*

Microservices



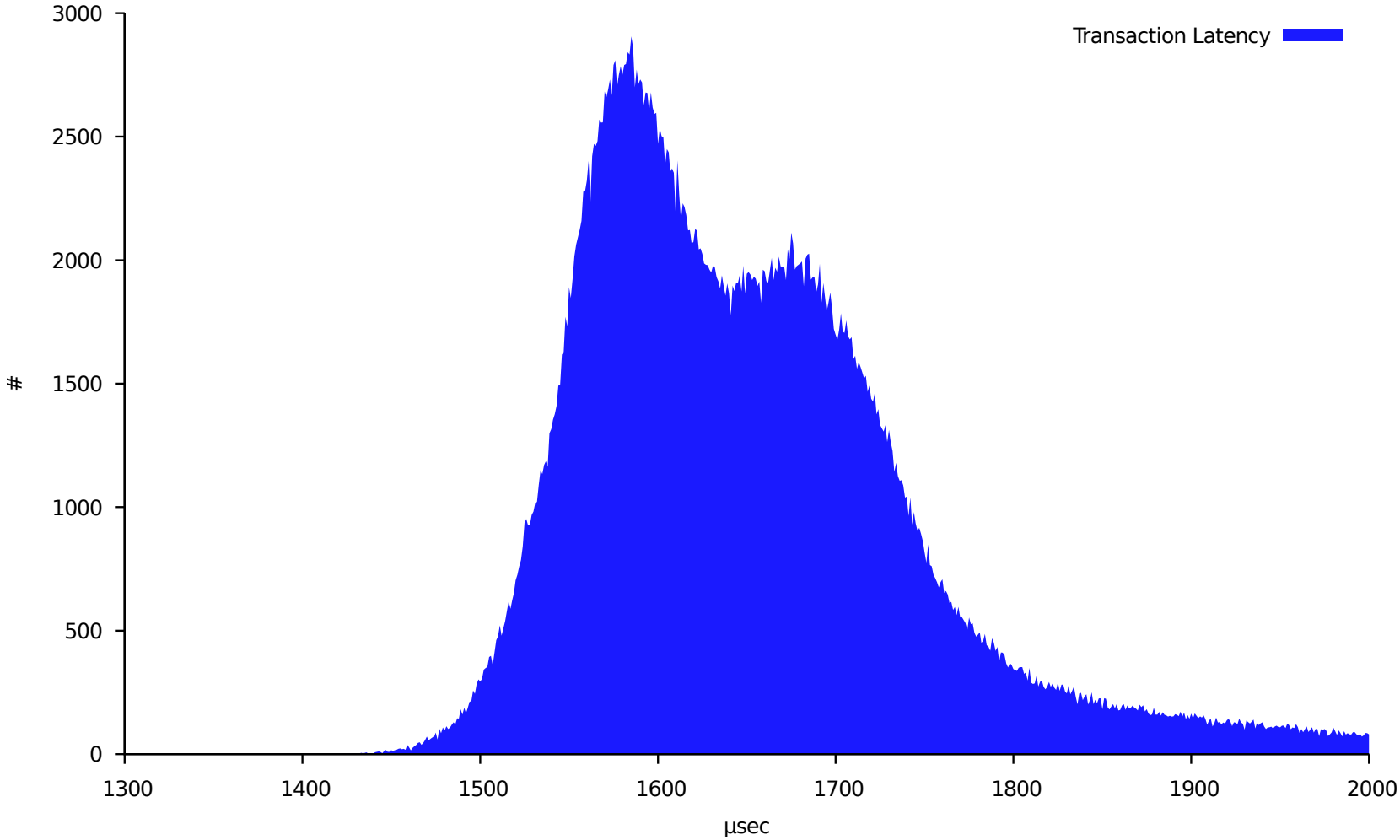
Microservices

- more components, more network traversals
- more jitter

Microservices

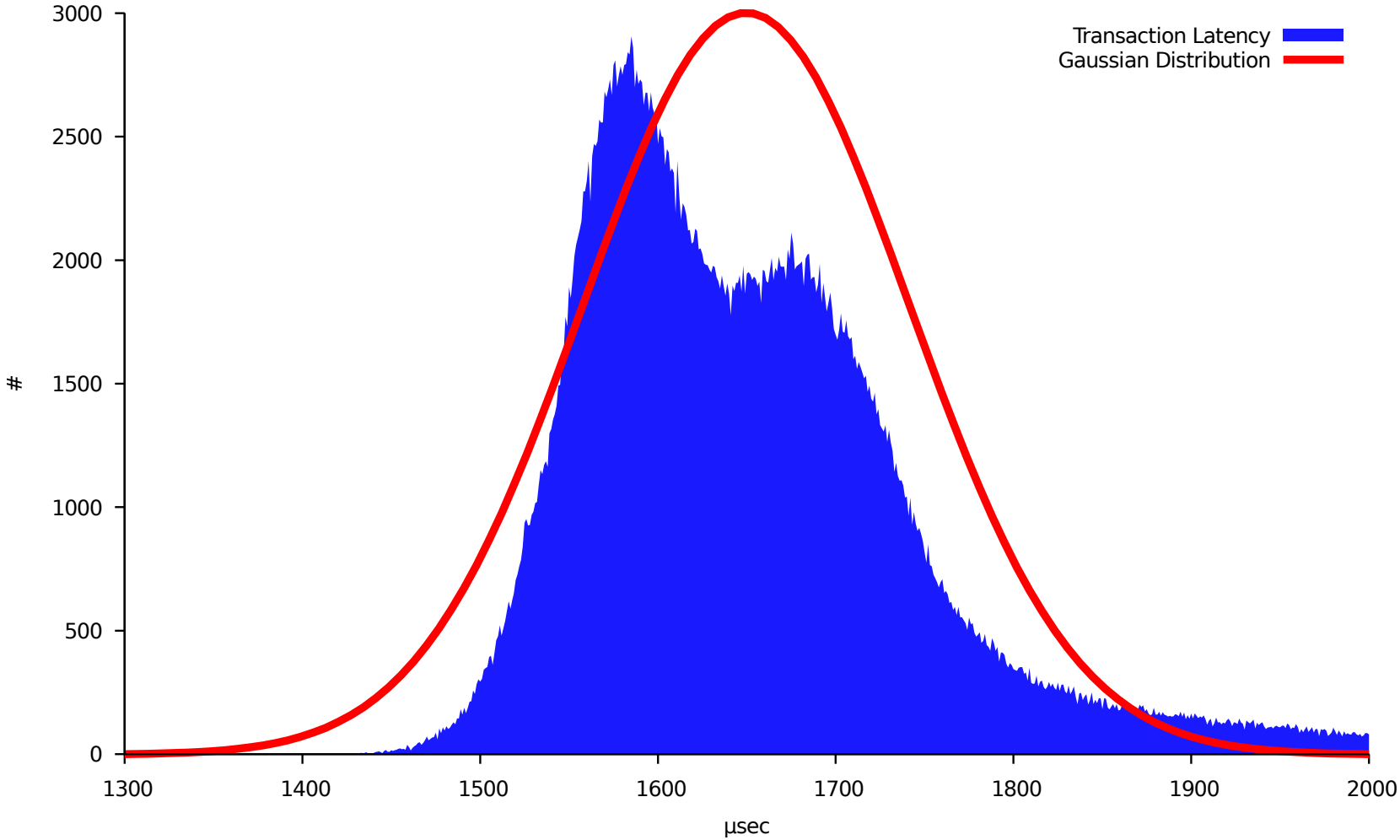
- more components, more network traversals
- more jitter
- What's Jitter?

Jitter



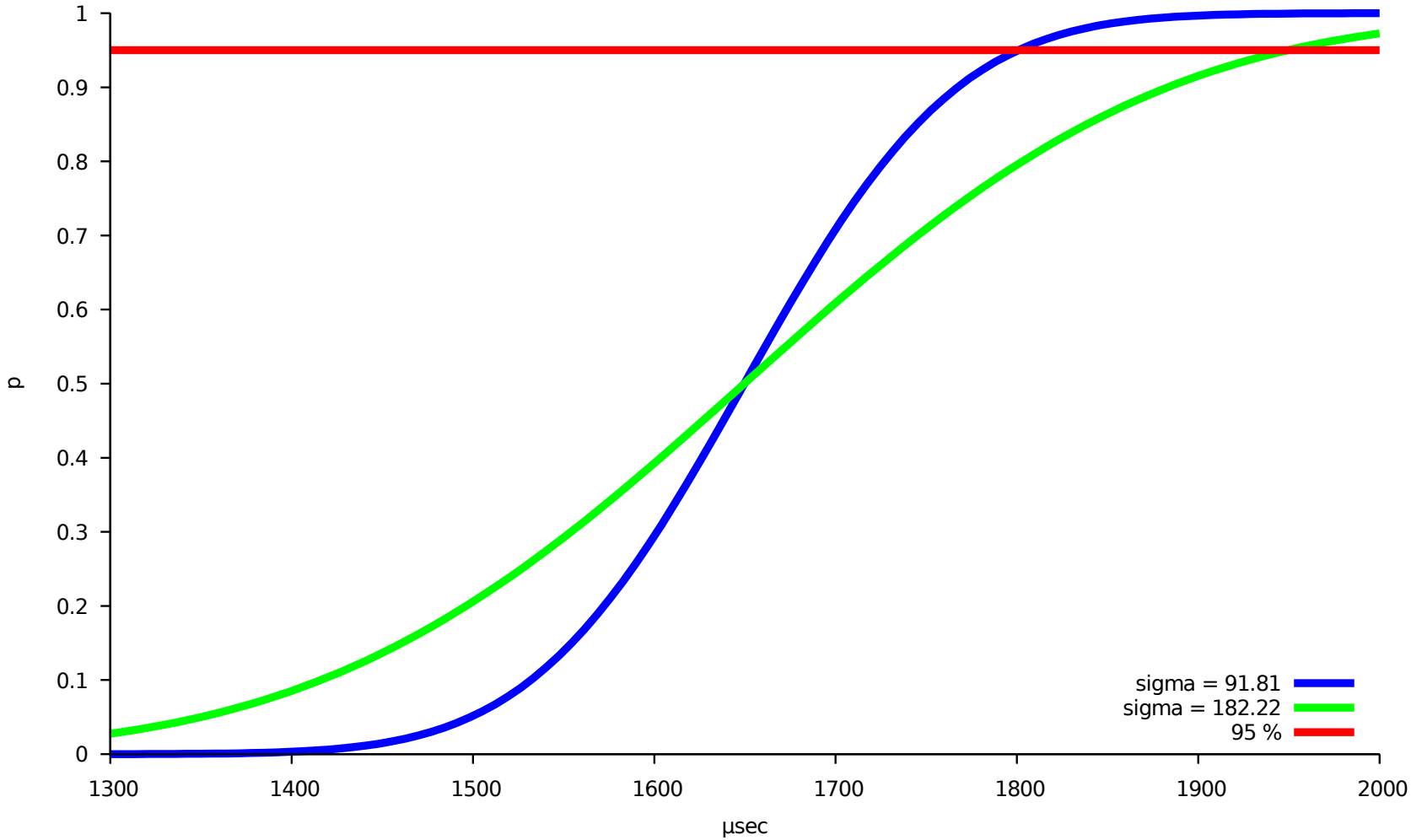
2ndQuadrant®
PostgreSQL

Jitter



2ndQuadrant®
PostgreSQL

Jitter



2ndQuadrant®
PostgreSQL

Response Time Jitter

- assume SLA: 98% queries faster 100 ms
- avg time 100 ms: 50% slower than 100 ms: BAD
- avg time 50 ms, σ 20: 0.8% slower than 100 ms: GOOD

Lession 2

- Response Time is User Experience (and money)

Lession 2

- Response Time is User Experience (and money)
- Check where time is spent

Lession 2

- Response Time is User Experience (and money)
- Check where time is spent
- higher standard deviation means more requests are too slow

Building for Performance

- Let's add more CPUs!
- Flash for Everyone!
- Can you use all those chips?

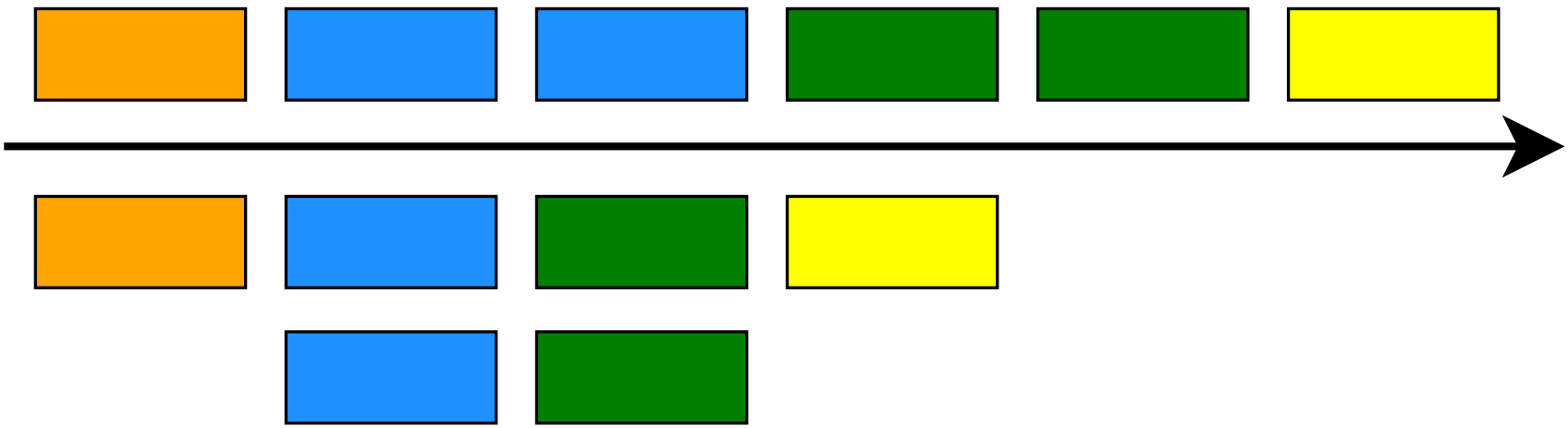
Standard Request Handling

```
function handle_request() {
    get_parameters()
    get_some_data()
    process_data()
    more_processing()
    do_output()
    }
```

Request Handling

- request processed serially
- single request performance does not benefit
- more requests at the same time . . .
- . . . if requests are independent
- what do you gain by parallelism?

Parallelism



Amdahl's Law

$$S_t(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

$$\lim_{s \to \infty} S_t(s) = \frac{1}{(1-p)}$$

Speedup is limited by:

- ratio of parallelizable work (p)
- speedup of parallel work (s)

What about Locking?

- processes have to wait on each other
- the more concurrent processes, the more interaction
- more data shared between processes: more locks
- run enough processes and all you do is waiting on locks

Gunther's Universal Scalability Law

$$C(N) = \frac{N}{1 + \alpha(N-1) + \beta N(N-1)}$$

- Capacity C as a function of the number N of parallel processes
- limited by contention α and coherency β ($\alpha > 0$, $\beta < 1$)
- constants can be determined by experiment

Gunther's Universal Scalability Law

$$C(N) = \frac{N}{1 + \alpha(N-1) + \beta N(N-1)}$$

- Capacity C as a function of the number N of parallel processes
- limited by contention α and coherency β ($\alpha > 0$, $\beta < 1$)
- constants can be determined by experiment
- *Neil Gunther, Guerilla Capacity Planning*

Intermission: Regression Analysis

- Non-Linear Least Squares Method
- run multiple tests with increasing number of *clients*
- get *capacity* measurement $C_i(N_i)$
- find parameters α, β minimizing

$$r = \sum_i (C(N_i) - C_i(N_i))^2$$

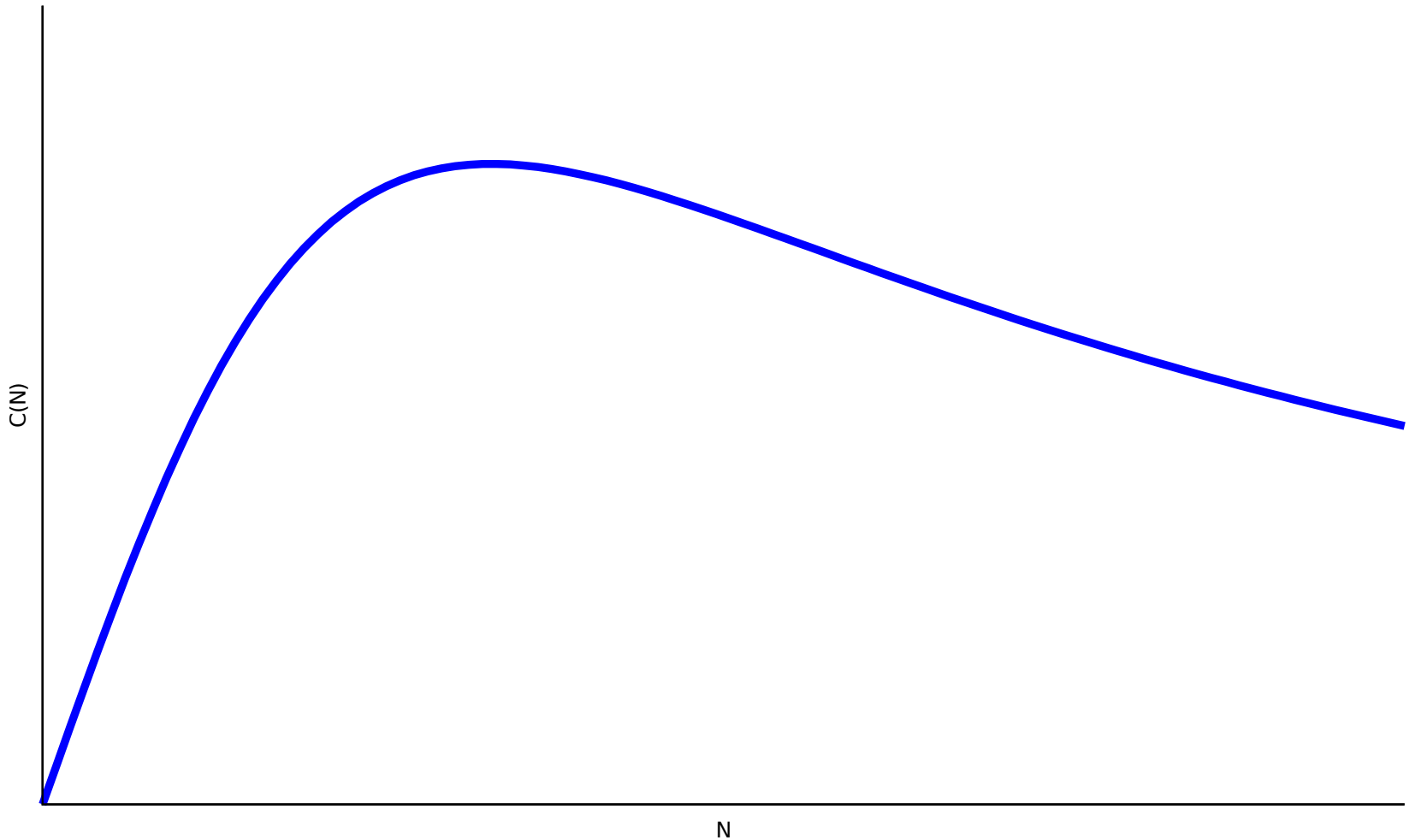
Intermission: Regression Analysis

- Non-Linear Least Squares Method
- run multiple tests with increasing number of *clients*
- get *capacity* measurement $C_i(N_i)$
- find parameters α, β minimizing

$$r = \sum_i (C(N_i) - C_i(N_i))^2$$

- Scared of formulas? No time?
- Software will help yout (spreadsheets, statistics packages, . . .)

Universal Law of Scalability



Lession 3

- adding hardware helps, initially

Lession 3

- adding hardware helps, initially
- but it's not a magic bullet

Lession 3

- adding hardware helps, initially
- but it's not a magic bullet
- build for capacity and speed

Lession 3

- adding hardware helps, initially
- but it's not a magic bullet
- build for capacity and speed
- Test, Calculate, Check with Reality

Tools

- pg_stat_statements

```
-[ RECORD 1 ]-------------------------------------
query       | UPDATE accts SET ab = ab + ? WHERE aid = ?;
calls       | 561867
mean_time   | 0.027920600426791 1
min_time    | 0.009
max_time    | 15.52
stddev_time | 0.099706221895680 1
```

- careful - response time not always normally distributed

Database Testing

- pgbench with custom scripts
- tsung
- pgreplay, playr
- Build-Your-Own

Application Monitoring & Testing

- Application Instrumentation (JMX, . . .)
- Client-Side Measurements (JMeter, tsung, . . .)
- Logging and Monitoring
- Watch for changes over time
- once your customers complain, it's too late

Lession 4

- Tools exist

Lession 4

- Tools exist
- Learn to use them

Lession 4

- Tools exist
- Learn to use them
- Monitoring & Math prevent the Meltdown

Lession 4

- Tools exist
- Learn to use them
- Monitoring & Math prevent the Meltdown
- Visualization helps

Thanks

2ndQuadrant®
PostgreSQL