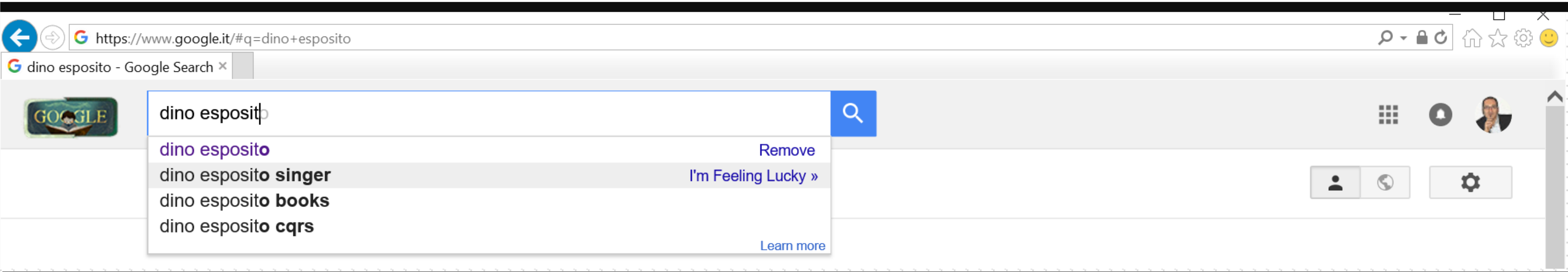
Life as a Service:
Scalability and Other Aspects

Dino Esposito

ARCHITECT, TRAINER AND CONSULTANT

@despos facebook.com/naa4e

Who's this guy?



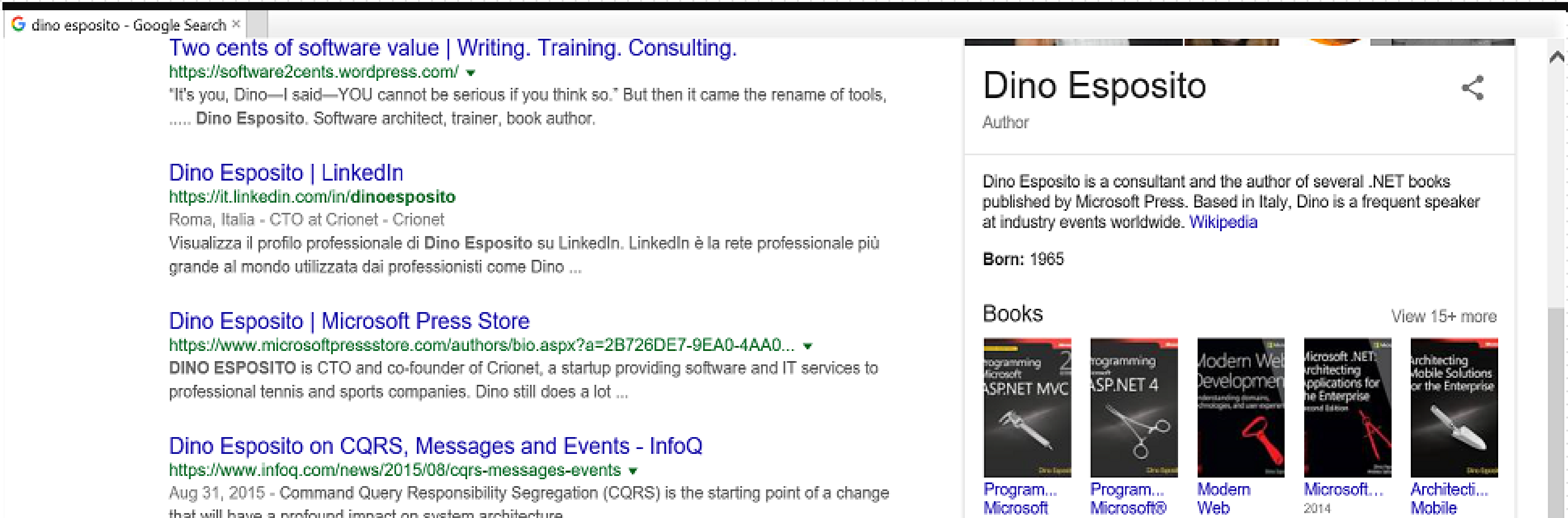
https://www.google.it/#q=dino+esposito

dino esposito - Google Search

dino esposit

- dino esposito — Remove
- dino esposito singer — I'm Feeling Lucky »
- dino esposito books
- dino esposito cqrs

Learn more



dino esposito - Google Search

Two cents of software value | Writing. Training. Consulting.
https://software2cents.wordpress.com/
"It's you, Dino—I said—YOU cannot be serious if you think so." But then it came the rename of tools, ... Dino Esposito. Software architect, trainer, book author.

Dino Esposito | LinkedIn
https://it.linkedin.com/in/dinoesposito
Roma, Italia - CTO at Crionet - Crionet
Visualizza il profilo professionale di Dino Esposito su LinkedIn. LinkedIn è la rete professionale più grande al mondo utilizzata dai professionisti come Dino ...

Dino Esposito | Microsoft Press Store
https://www.microsoftpressstore.com/authors/bio.aspx?a=2B726DE7-9EA0-4AA0...
DINO ESPOSITO is CTO and co-founder of Crionet, a startup providing software and IT services to professional tennis and sports companies. Dino still does a lot ...

Dino Esposito on CQRS, Messages and Events - InfoQ
https://www.infoq.com/news/2015/08/cqrs-messages-events
Aug 31, 2015 - Command Query Responsibility Segregation (CQRS) is the starting point of a change that will have a profound impact on system architecture

Dino Esposito
Author

Dino Esposito is a consultant and the author of several .NET books published by Microsoft Press. Based in Italy, Dino is a frequent speaker at industry events worldwide. Wikipedia

Born: 1965

Books — View 15+ more

Program... Microsoft | Program... Microsoft® | Modern Web | Microsoft... 2014 | Architecti... Mobile

PART I

Scalability and Measurable Tasks

SCALABILITY

Scalability is the ability of a system to expand to meet your business needs. You scale a system by adding extra hardware or by upgrading the existing hardware without changing much of the application.

Microsoft

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth.

Wikipedia
The Free Encyclopedia

SCALABILITY

Everyone talks about it

SCALABILITY

Nobody really knows how to do it

SCALABILITY

Everyone thinks everyone else is doing it

SCALABILITY

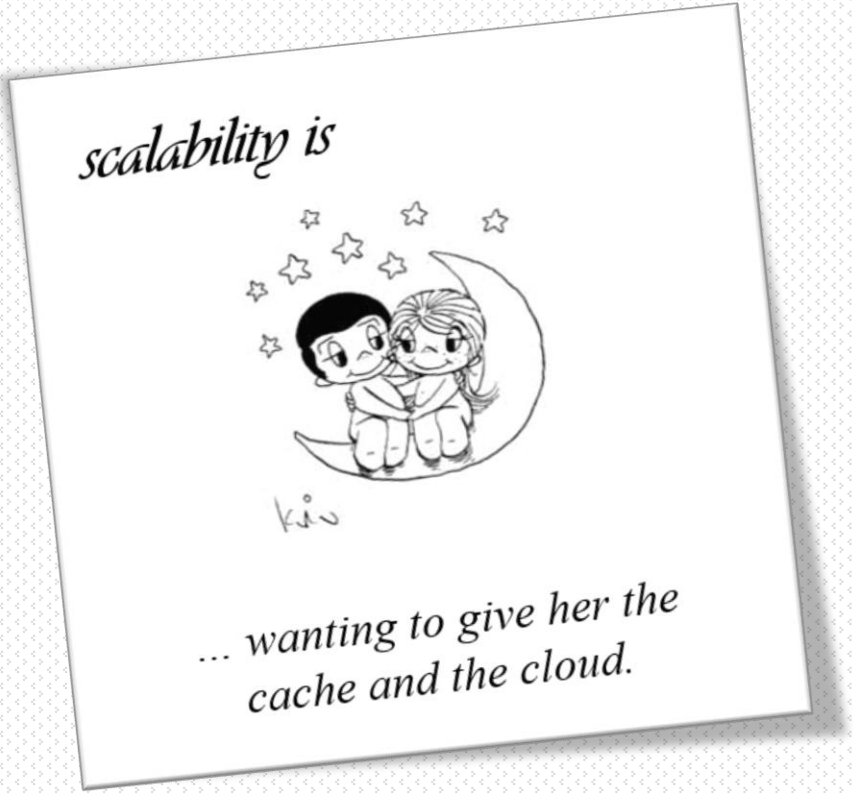
So everyone claims they are doing it

SCALABILITY

So everyone claims they *want* to do it

- Customers only

SCALABILITY ≈ teenage sex



Attributes of Scalability

Measurable

Growing factor

Context

Not everything that matters can be measured and not everything that can be measured matters

READ ON stackoverflow

COOL

PONDERING

SEXY

I’M NOT SCALABLE

NOT
SCALABLE

10 EUR

12 EUR

TROUBLES

Fine-tuning | Resiliency | Performance

5 EUR

5 EUR x N

JOY

SCALABLE

Queuing theory

A queue forms when frequency at which requests for a service are placed exceeds the time it takes to fully serve a pending request.

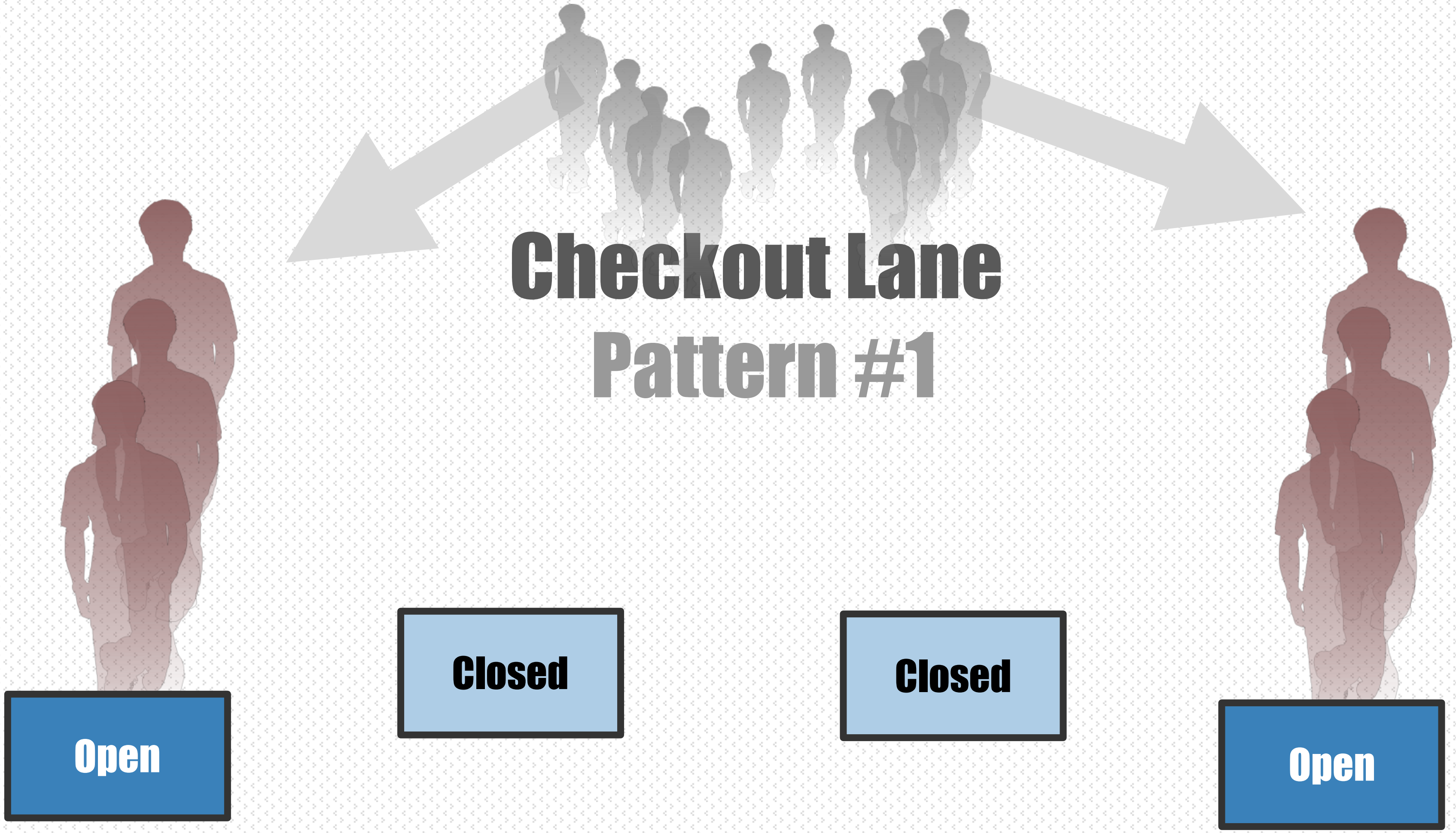
- ❑ It’s about the **performance** of a single task
- ❑ It’s about **expanding** the system to perform more tasks at the same time

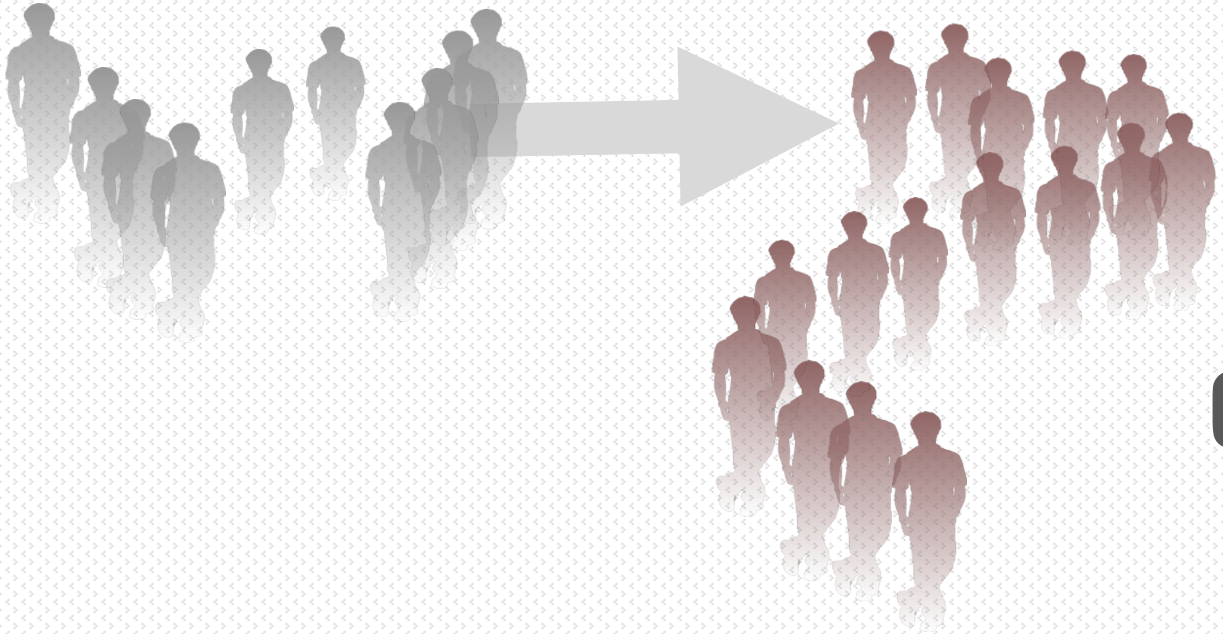
SCALABILITY

Became a problem as the web became popular

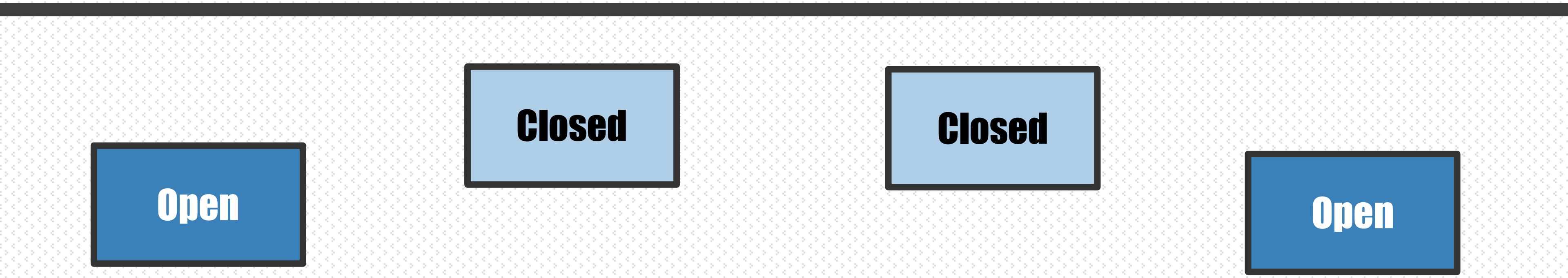
Checkout Lane

Pattern #1





Checkout Lane
Pattern #2



SCALABILITY

Context is king

Common Tradeoffs

Scalability in Space	vs	Scalability in Time
Vertical Scalability	vs	Horizontal Scalability

Scalability in Space

Configuration of systems for particular users

Big data

Time-traveling data

Data sharding

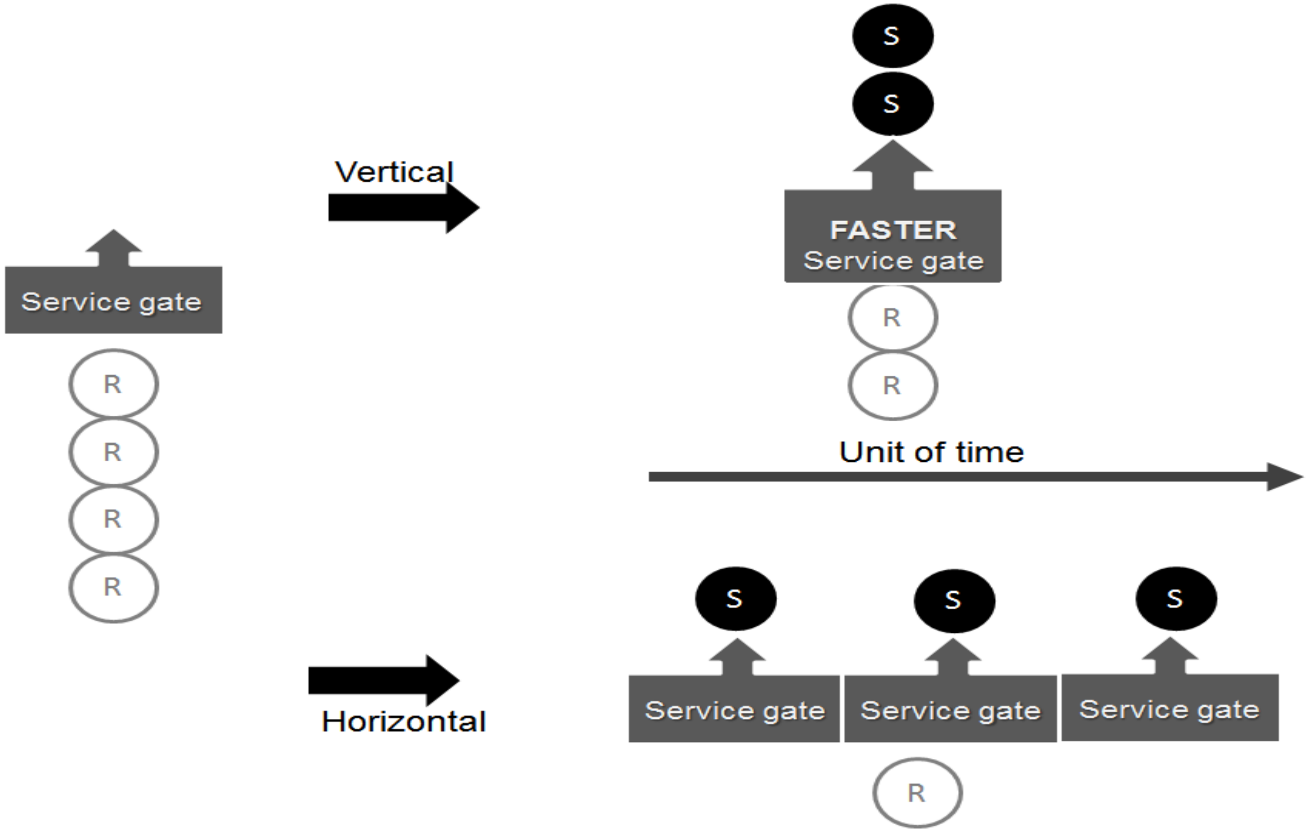
Scalability in Time

Infrastructure and network balancing

Cloud

Geographic distribution of data/service centers

Vertical vs. Horizontal



Vertical

Norm for 20 years – so long as DB was central point

Doesn’t scale beyond a point

Front caching is a good/cheaper way to achieve it

- Proxy servers with load balancing capabilities
- Working outside the core code of the application
- Squid, Varnish, Nginx

Horizontal

Mostly an architectural point

Critical parts can be expanded without

- Damaging other parts
- Introducing inconsistencies / incongruent data

Horizontal

MULTIPLE INSTANCES

LOAD BALANCING

DATA SHARDING

Real-world

Cloud apps are probably the easiest and most effective way to achieve forms of scalability today.

But you can have well responsive apps without re-architecting for the cloud.

PART II

Caching and Measurable Performance

Operational Practice #1

Remove bottlenecks

- Convoluted queries
- Long initialization steps
- Inefficient algorithms

HIGH throughput

MEDIUM cost

TIME consuming

DELICATE

Operational Practice #2

Move "some" requests to other servers

- CDN for static files
- Geographically distributed sites

LOW throughput

LOW cost

Quick

Improves the user's perception of the system

Operational Practice #3

Output Caching

- By param
- By locale
- By custom data

  for example, multi-tenant sites

MEDIUM throughput

LOW cost

Quick

MEDIUM risk

Operational Practice #4

Data Caching

- Problematic with farms
- Auto-updatable internal cache
- Use of distributed caches

    Redis, ScaleOut, NCache

HIGH throughput

MEDIUM cost

Relatively Quick

DELICATE

Operational Practice #5

Proxy caching for example **Varnish**

- Installed in front of any web site
- Fully configurable
- Cache and load balancer in one

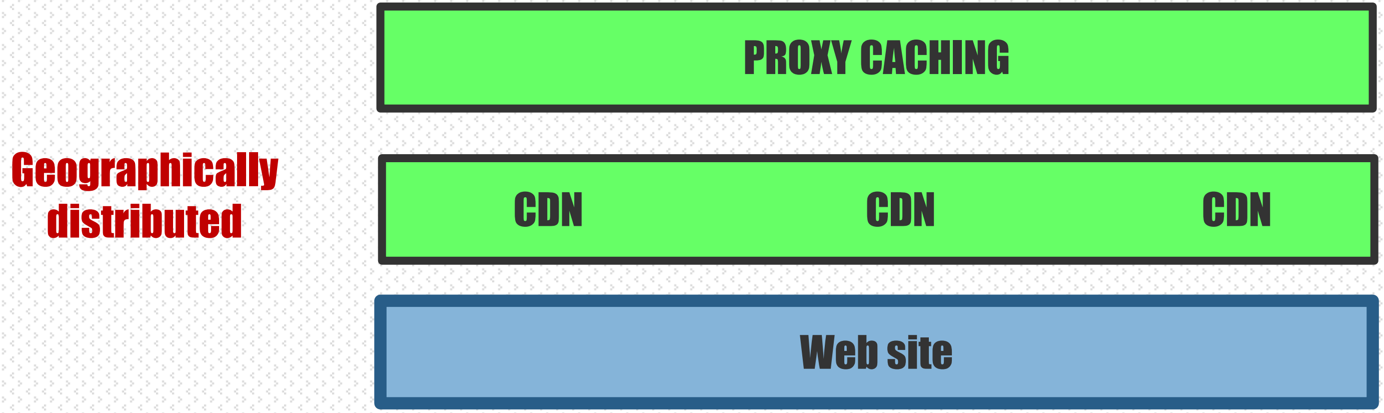
HIGH throughput

Relatively LOW cost

Relatively Quick

CDN vs. Proxy Caching

Not an either/or choice; often go together.



Architectural Practice #1

CQRS Architecture

- Optimize stacks differently

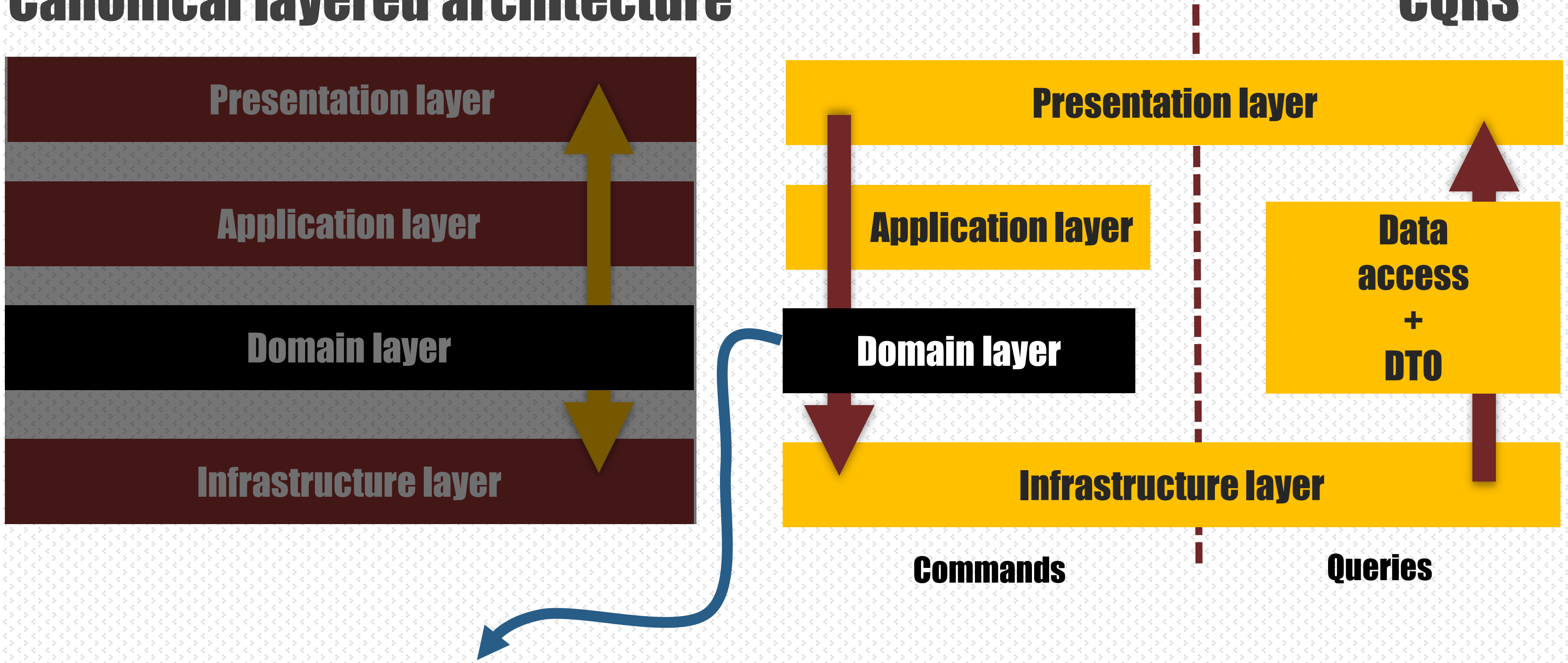
HIGH throughput

HIGH cost

Time consuming

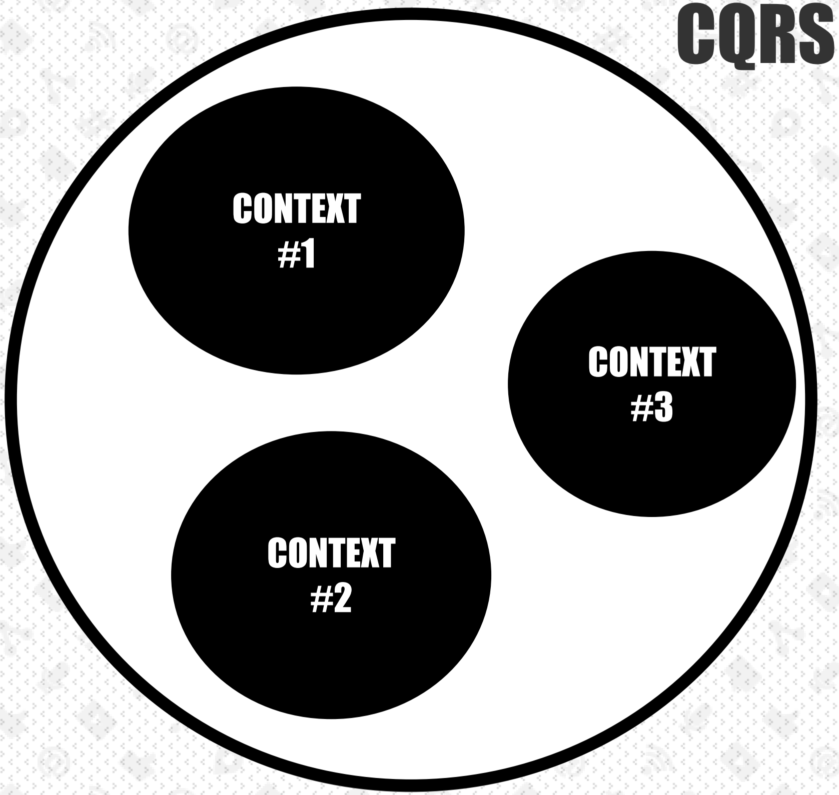
Canonical layered architecture

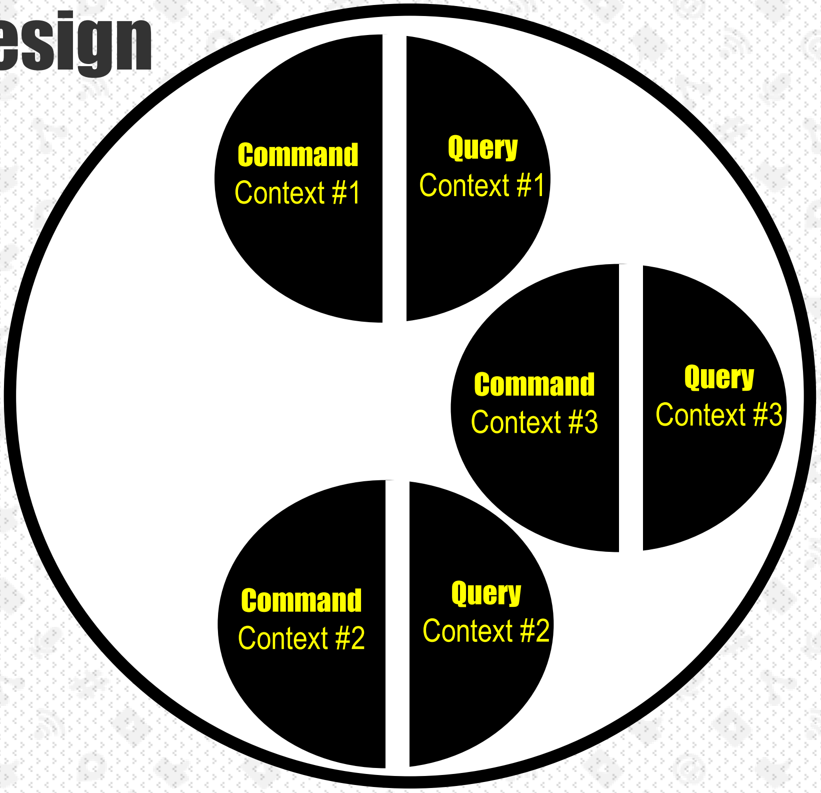
CQRS



Message-based business logic implementation

Requirements
DDD Analysis
CQRS Design





Architectural Practice #2

Single-tier and stateless server

- One server
- No session state
- Quick and easy to duplicate
- Behind a load balancer

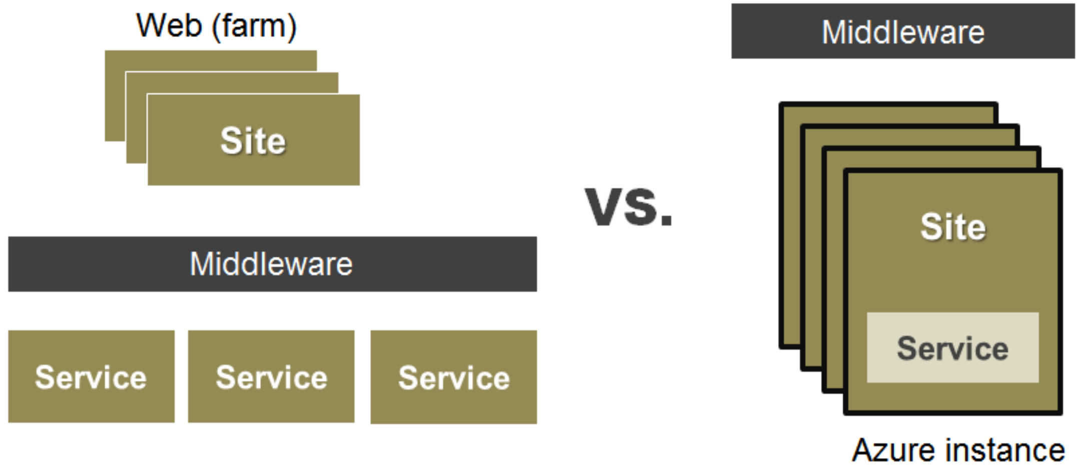
HIGH throughput

Low cost

Quick

Cloud support

- On-demand servers
- Pay per use
- Configure easily
- No middleware costs
- Better failure policies



Architectural Practice #3

Be aware of NoSQL and polyglot persistence

- Relational is OK ... until it works
- Sharding/growth of data

Azure SQL
+ Many small tables <500GB each
+ No extra license costs
+ Zero TCO
+ HA automatically on

SQL Server in a VM
+ Fewer large tables >500GB each
+ Reuse existing licenses
+ More machine resources
+ HA and management is your own

Case-studies

StackOverflow

Top50 most trafficked web sites

500M page views in a month and growing

- **< 10** Visual Studio projects
- **100,000** lines of code
- Nearly no unit tests: code coverage of **0.25%**
- Web projects based on **ASP.NET MVC**
- Move fast and break things (that users don’t see)
- Deploy **5** times a day

Caching is the secret weapon

5 levels of data caching

- Network (i.e., CDN and browsers)
- Server (ASP.NET cache)
- Site (Redis)
- SQL Server (1 terabyte of RAM)
- Solid state disks

For more information: http://speakerdeck.com/sklivvz

Globo.com

Largest portal and ISP in South America

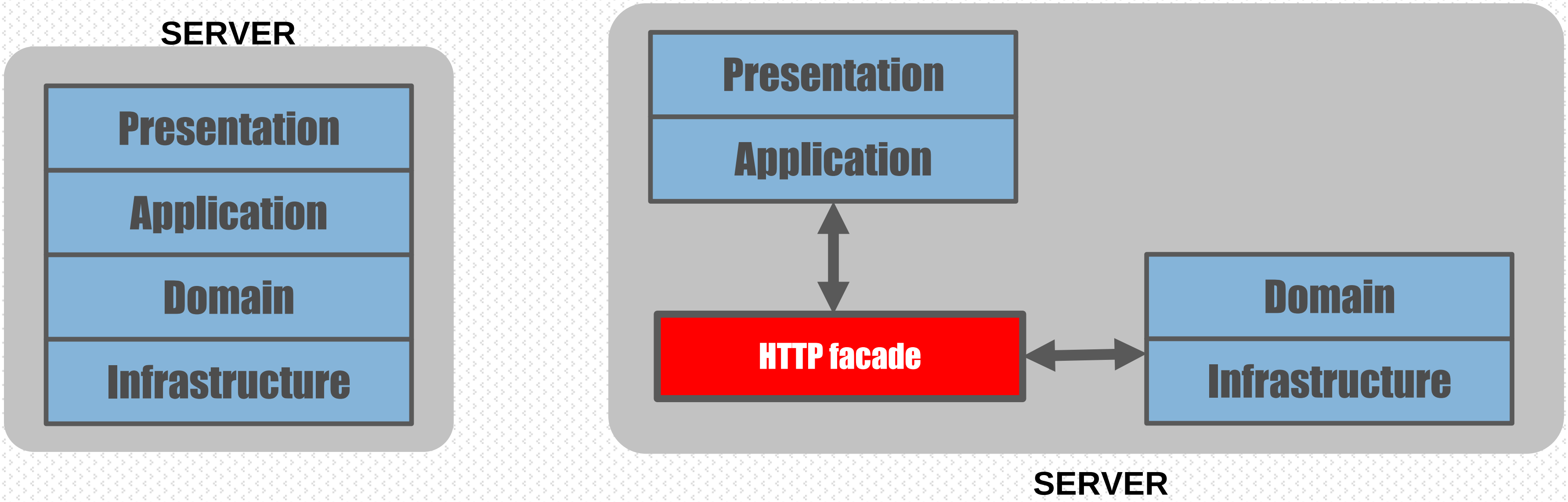
45 million views per day **2x** StackOverflow

- 1 Scrum master, 1 designer, 1 lead architect
- TDD, peer programming, CI, 10 days sprints
- WordPress with PHP
- Python/Django for server-side code
- Data storage is MySQL, MongoDB & Memcached
- Reverse proxy caching

Talking CMS that grow BIG

1. Your database grows **really** BIG
2. Monolithic code **hard** to cluster
3. Content **generation** become EXPENSIVE
4. Site **slows** DANGEROUSLY down

My personal horror story



MOST OF THE TIME … YOU DON'T HAVE SCALABILITY PROBLEMS.

YOUR CODE JUST SUCKS.

That's All Folks!

FOLLOW @despos

FOLLOW @udev_tech_events

facebook.com/naa4e

facebook.com/plarium

software2cents.wordpress.com

developers.plarium.com

dino.esposito@jetbrains.com