

Getting a System to Production

... and keeping it there

SATURN 2016

Eoin Woods
Endava

Who Am I?



- ✦ Eoin Woods - CTO at Endava
 - ✦ 2005 - 2014 in capital markets (UBS, BGI)
 - ✦ 2000 - 2004 in product engineering & consultancy (Bull, Sybase, InterTrust, independent)
- ✦ Author, editor, speaker, community-guy



Who are Endava?

- ✦ Software Engineering & IT Services Firm

- ✦ 2800+ people
- ✦ UK, US, Germany, Romania, Moldova, Serbia, Macedonia

- ✦ Agile and Digital Transformation

- ✦ Consulting, Architecture, Development, Testing
- ✦ Data and Analytics
- ✦ Application Management, Infrastructure, DevOps



Content

- ✦ Introducing Production Systems
- ✦ What Goes Wrong in Production?
- ✦ Solutions for Production Systems
- ✦ Conclusions



Production Systems



What is a production system?



Any system
being used
for real work

Why is Productionisation Hard?

- ✦ No one teaches you about production
 - ✦ who do you talk to?
 - ✦ what do they want?
 - ✦ what is the definition of “done” ?
- ✦ Production is difficult for developers
 - ✦ hard to access, interrogate, debug, change, ...

A new cast of characters

Development



Developers



Users

A new cast of characters

Production



Auditors



Operations



Developers



Infrastructure



Business
Management



Users



Acquirers

Production is constrained

- ✦ Highly controlled
- ✦ Content is all valuable
- ✦ Change can be difficult

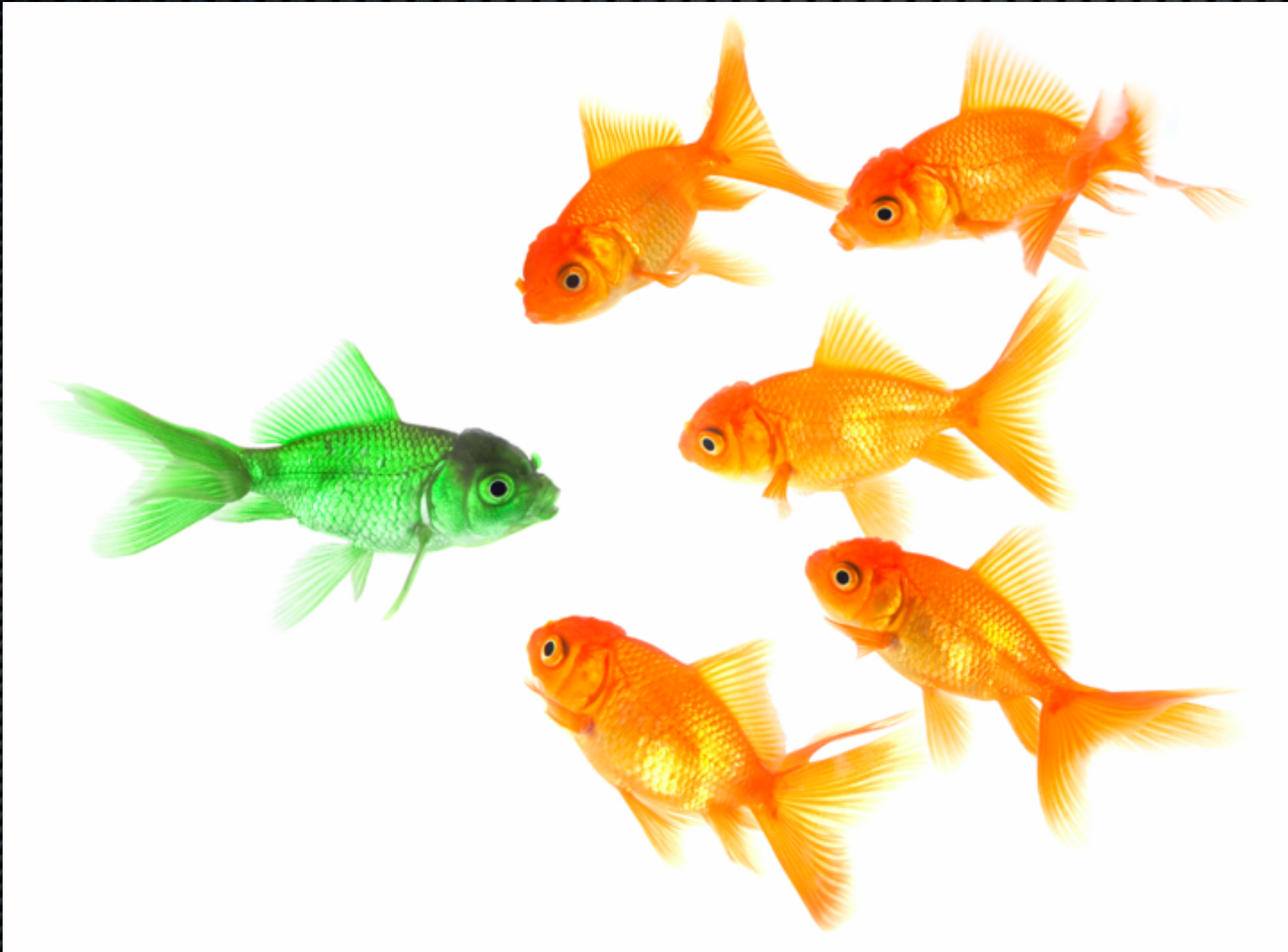


Production is unpredictable



© Alamy

Production is highly visible!



You don't own production

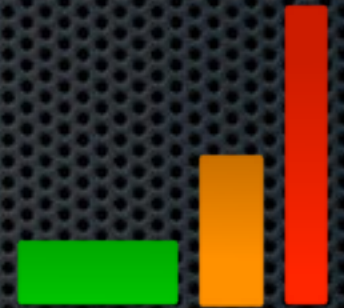


What goes wrong?



Performance surprises

- ✧ Interactive load
- ✧ Batch time surprises
- ✧ System abusers!
 - ✧ “all transactions this year”,
 - ✧ “average since 1967”, ...



Environment bombshells

- ✧ Constraints and contention



- ✧ Unexpected behaviour



- ✧ Integration points



Failures happen

- ✦ Software defects
- ✦ Platform failures
- ✦ Environment failures



Security tangles

- ✦ Security is **simple** in **Development**
- ✦ Much more **complex** in **Production!**



Finding Solutions



Architects Know This - Right?



Architectural Heresy

- ✦ Architects obsess about system qualities
 - ✦ usually results in good production characteristics
- ✦ However teams just find this all a bit hard
 - ✦ too many qualities, need to get functions delivered
- ✦ ... and we must empower teams
 - ✦ architects can't be responsible for all of the software being “production ready”


Key requirements for production

- ✦ Functionally **correct**
 - ✦ does what the business process requires
- ✦ **Stability**
 - ✦ behaves predictably in all situations
- ✦ **Capacity**
 - ✦ can process the workload required (at all times)
- ✦ **Security**
 - ✦ limits access to those who are authorised to have it

Solution Framework

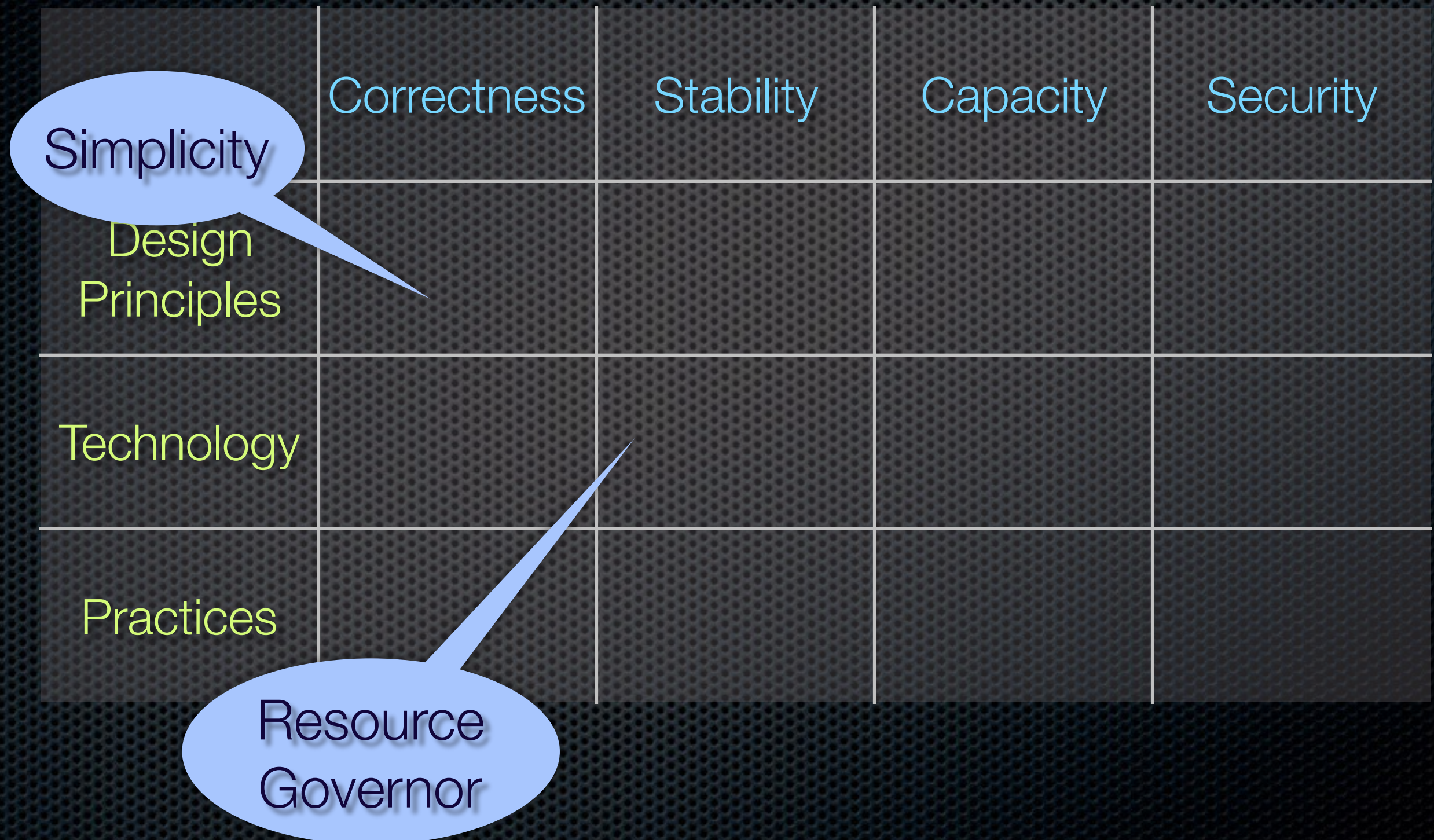
	Correctness	Stability	Capacity	Security
Design Principles				
Technology				
Practices				

Solution Framework

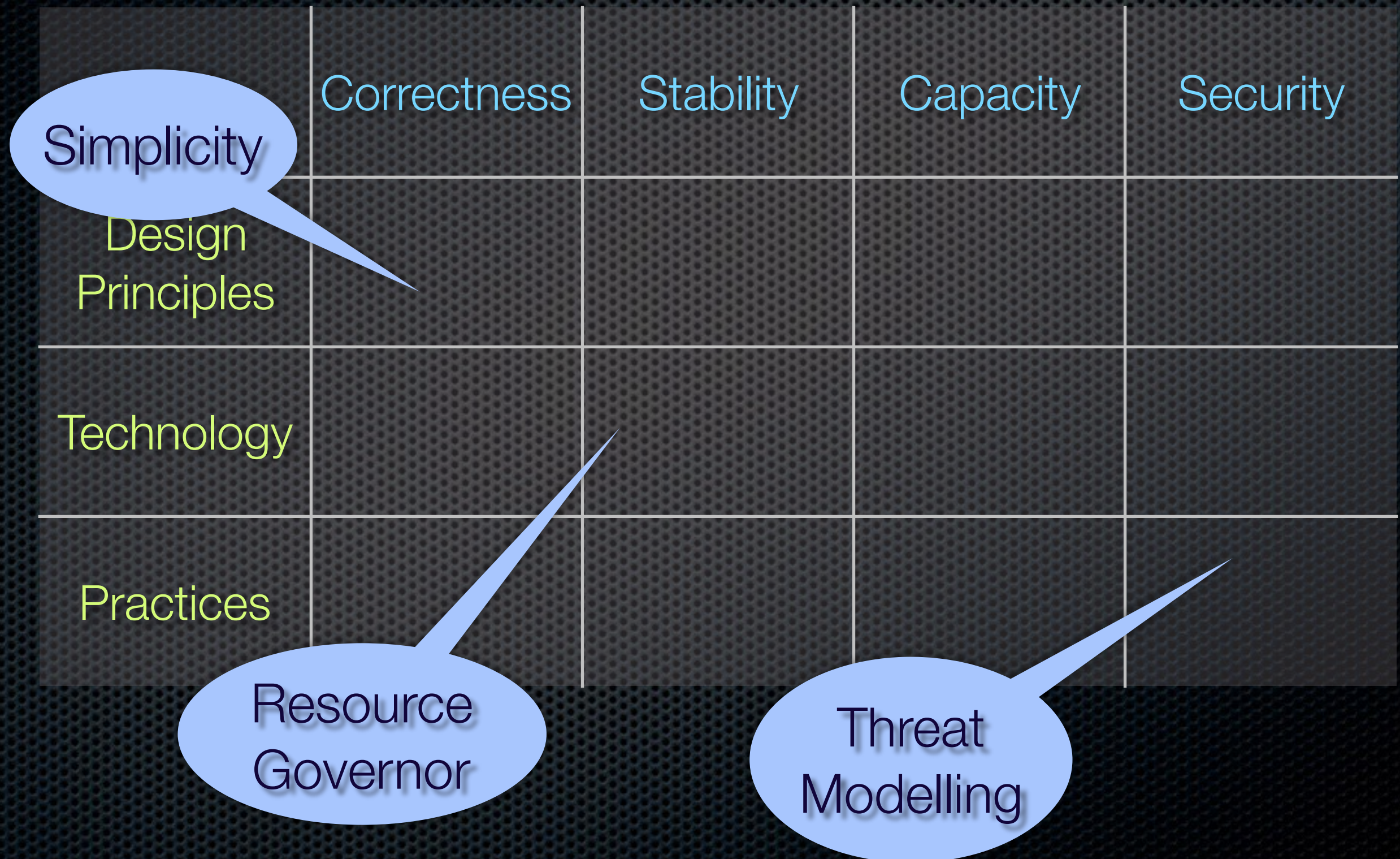


	Correctness	Stability	Capacity	Security
Design Principles				
Technology				
Practices				

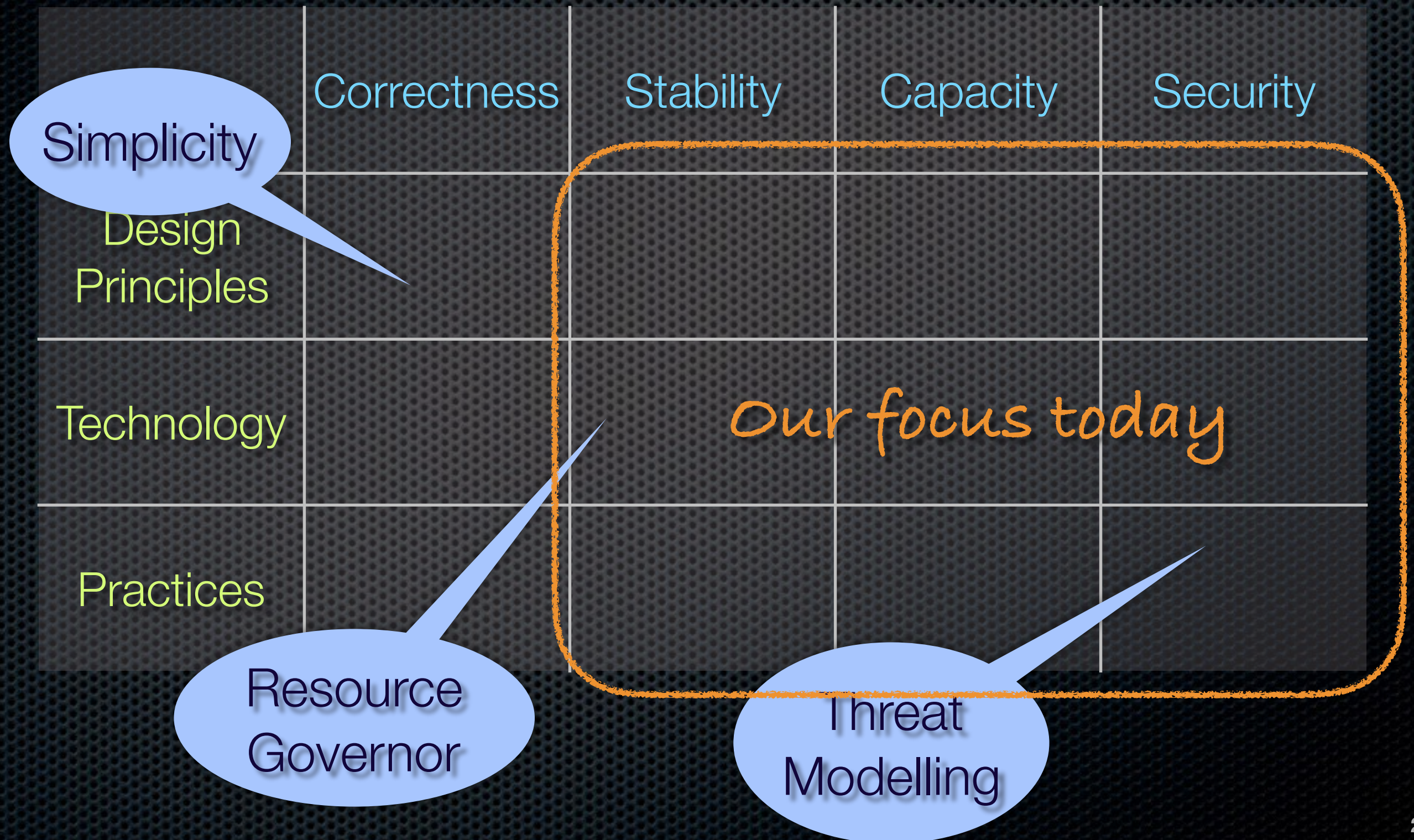
Solution Framework



Solution Framework



Solution Framework



General Principles

- ✦ One Team
- ✦ Automate
- ✦ Measure and Improve (feedback loops)
- ✦ Good Enough over Perfection

Timeless principles ... that led to CD and DevOps

So How About DevOps?

- ✦ DevOps helps get code **to** production
 - ✦ not much about whether it is **ready** for production
- ✦ Developers still need to “productionise”
 - ✦ make sure the software meets the requirements for production operation
- ✦ Relatively few developers get much training to prepare them for this

DevOps Principles

- ✦ **C**ommunication
- ✦ **A**utomation
- ✦ **L**ean thinking
- ✦ **M**easurement
- ✦ **S**haring

CALMS - itrevolution.com/devops-culture-part-1

Solutions: Achieving Stability



Stability - design principles

- ✧ Fail quickly

- ✧ fail fast, timeouts

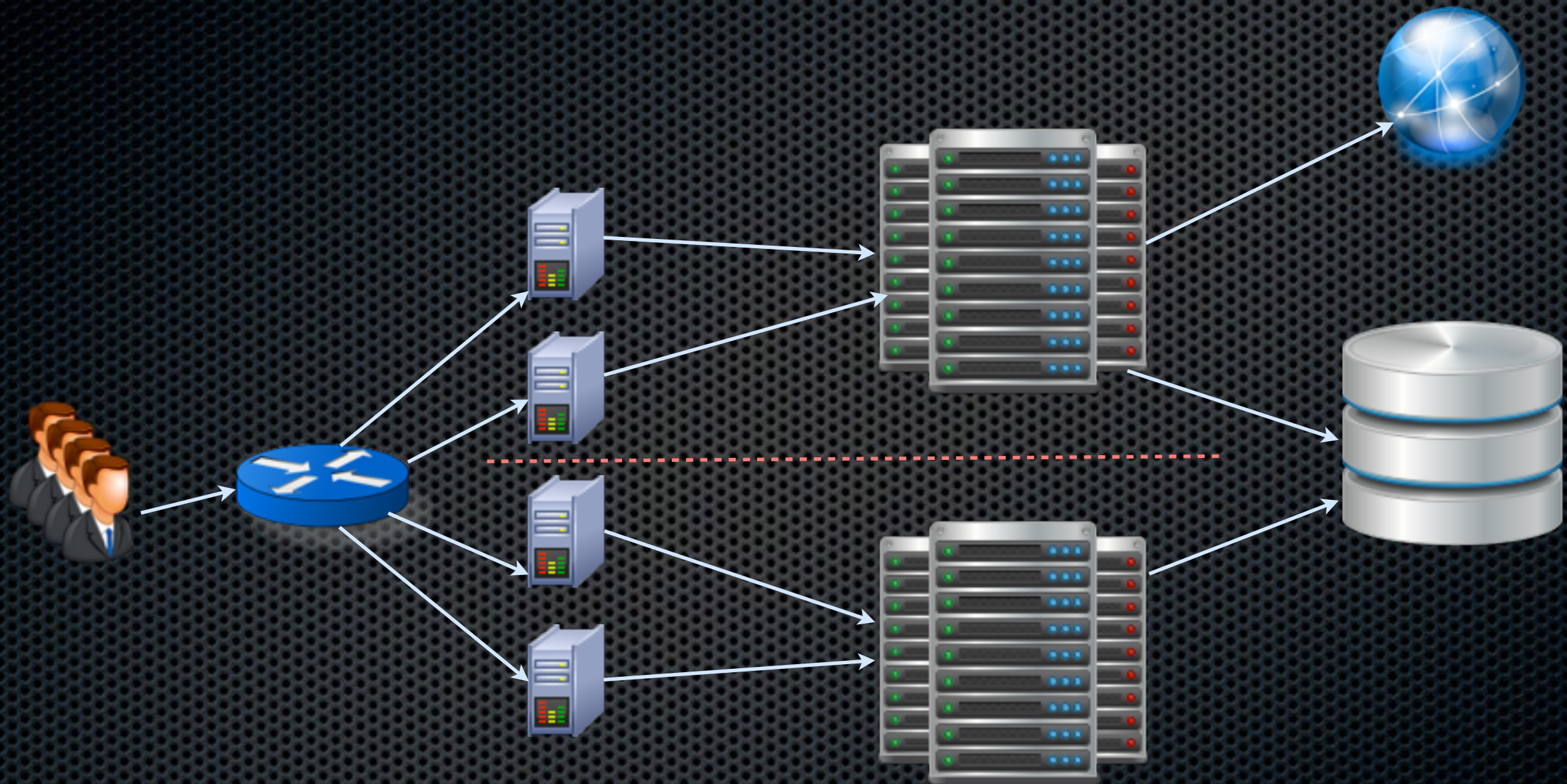
- ✧ Isolate problems

- ✧ flow control, circuit breakers, bulkheads, asynchronous integration

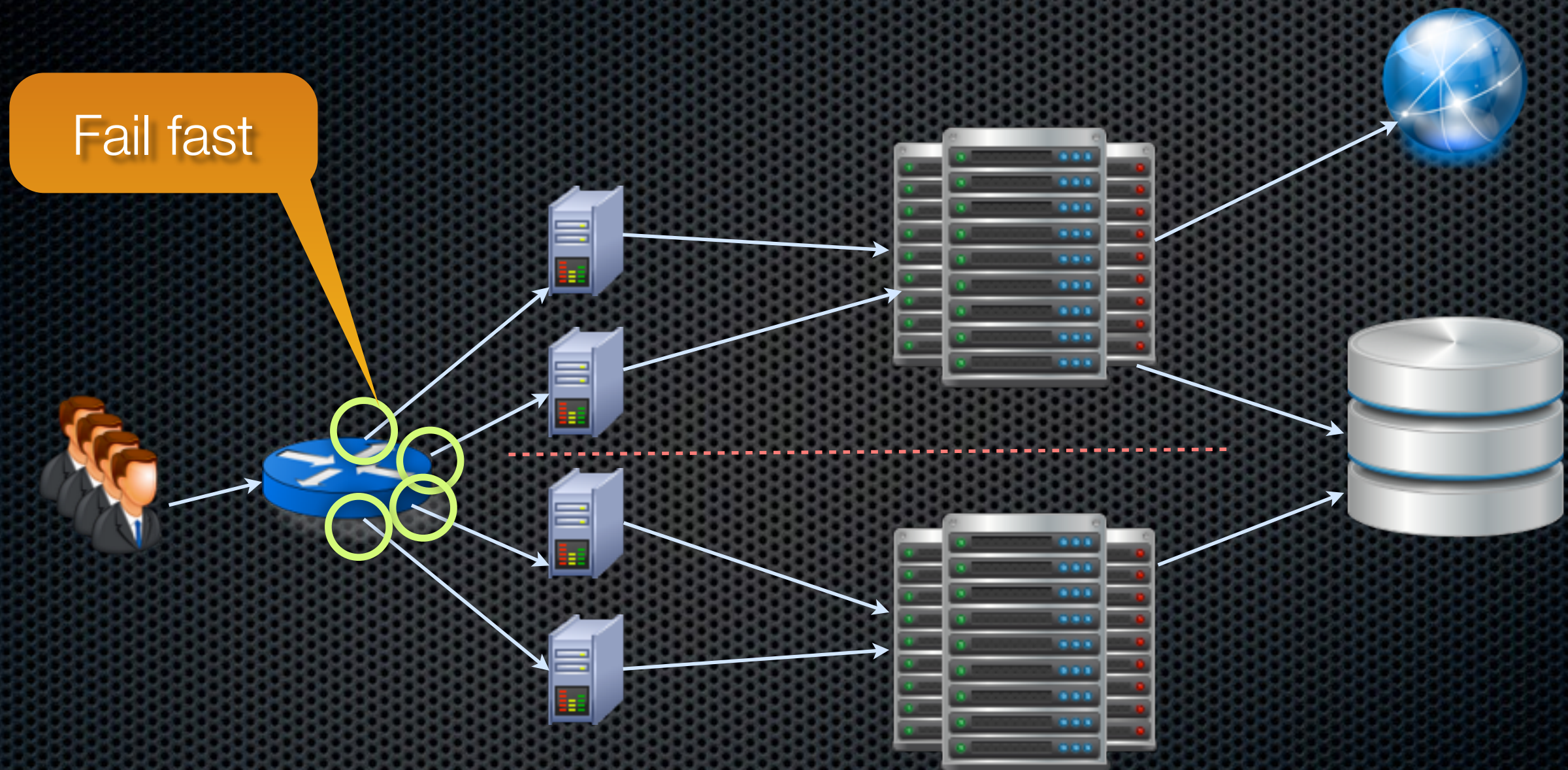
- ✧ Ensure steady state operation

- ✧ housekeeping, predictable resource allocation, governors, throttling

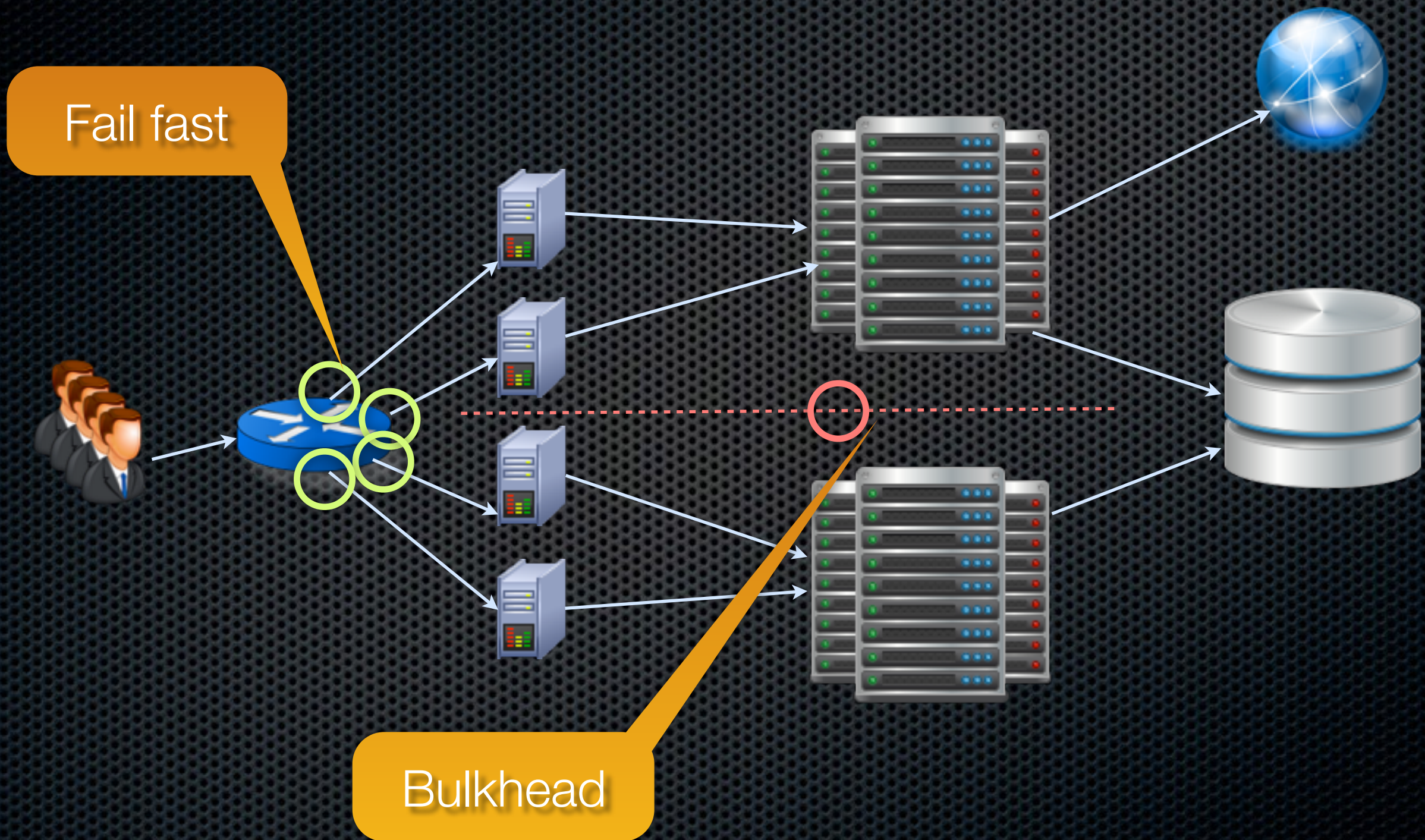
Stability - technology solutions



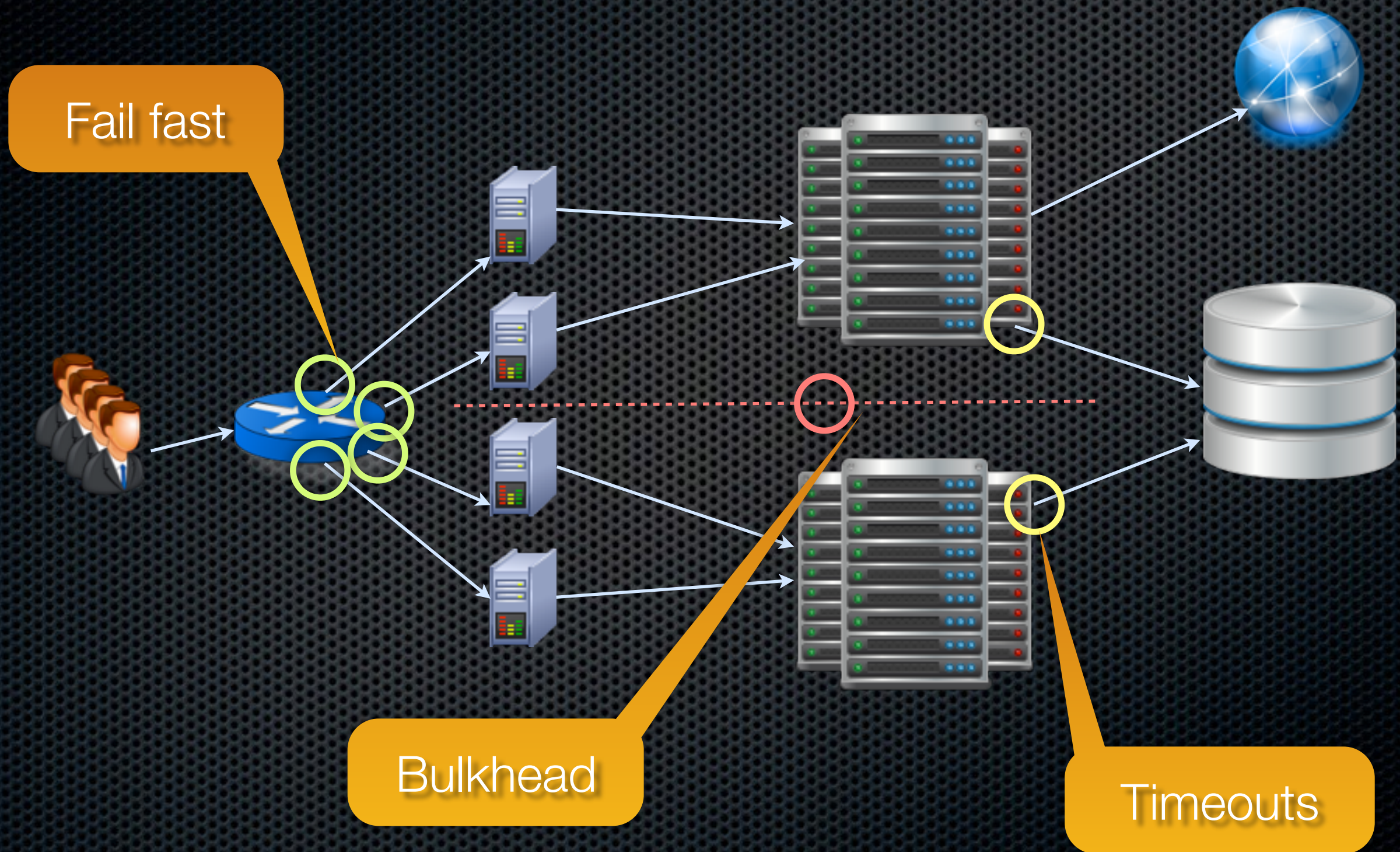
Stability - technology solutions



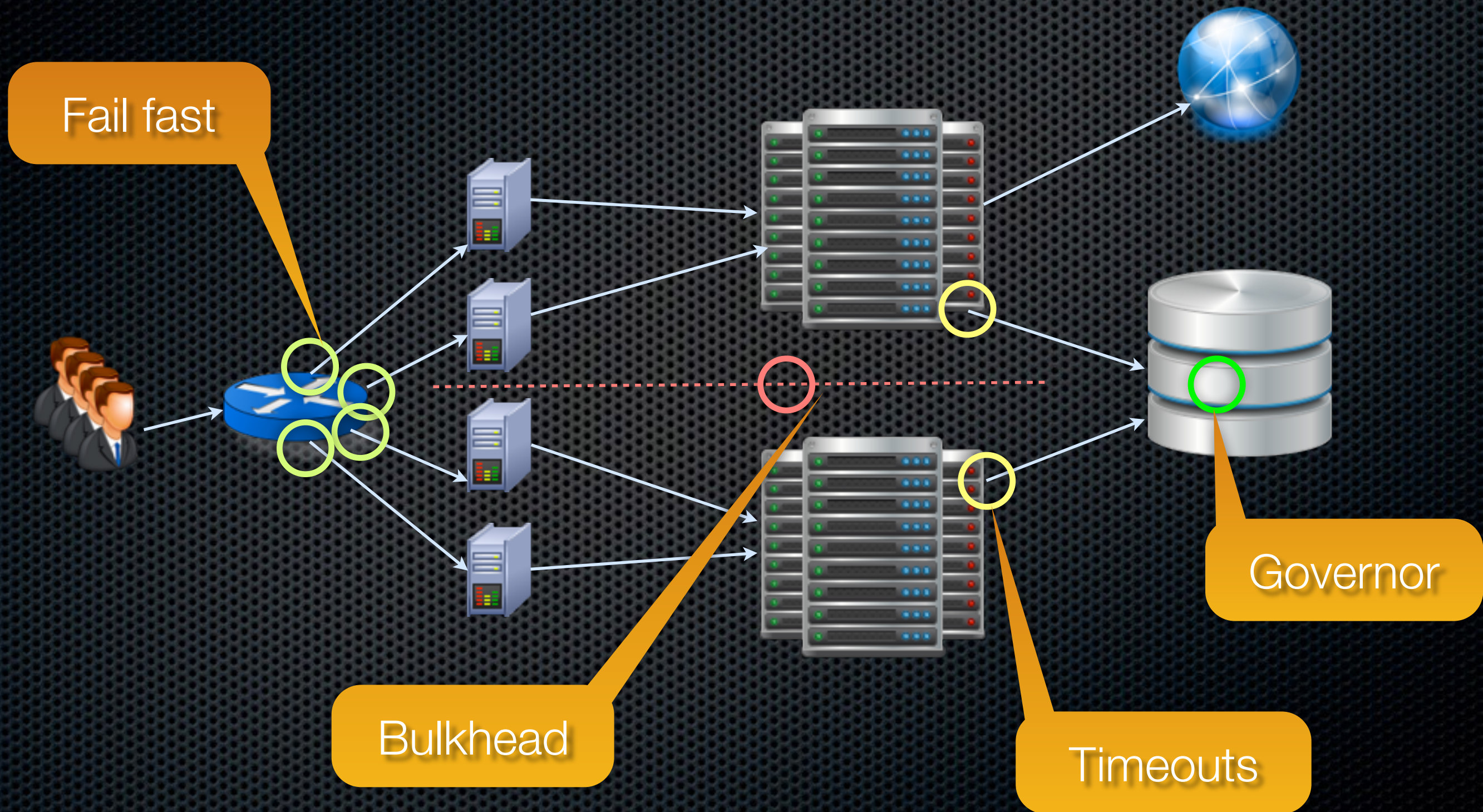
Stability - technology solutions



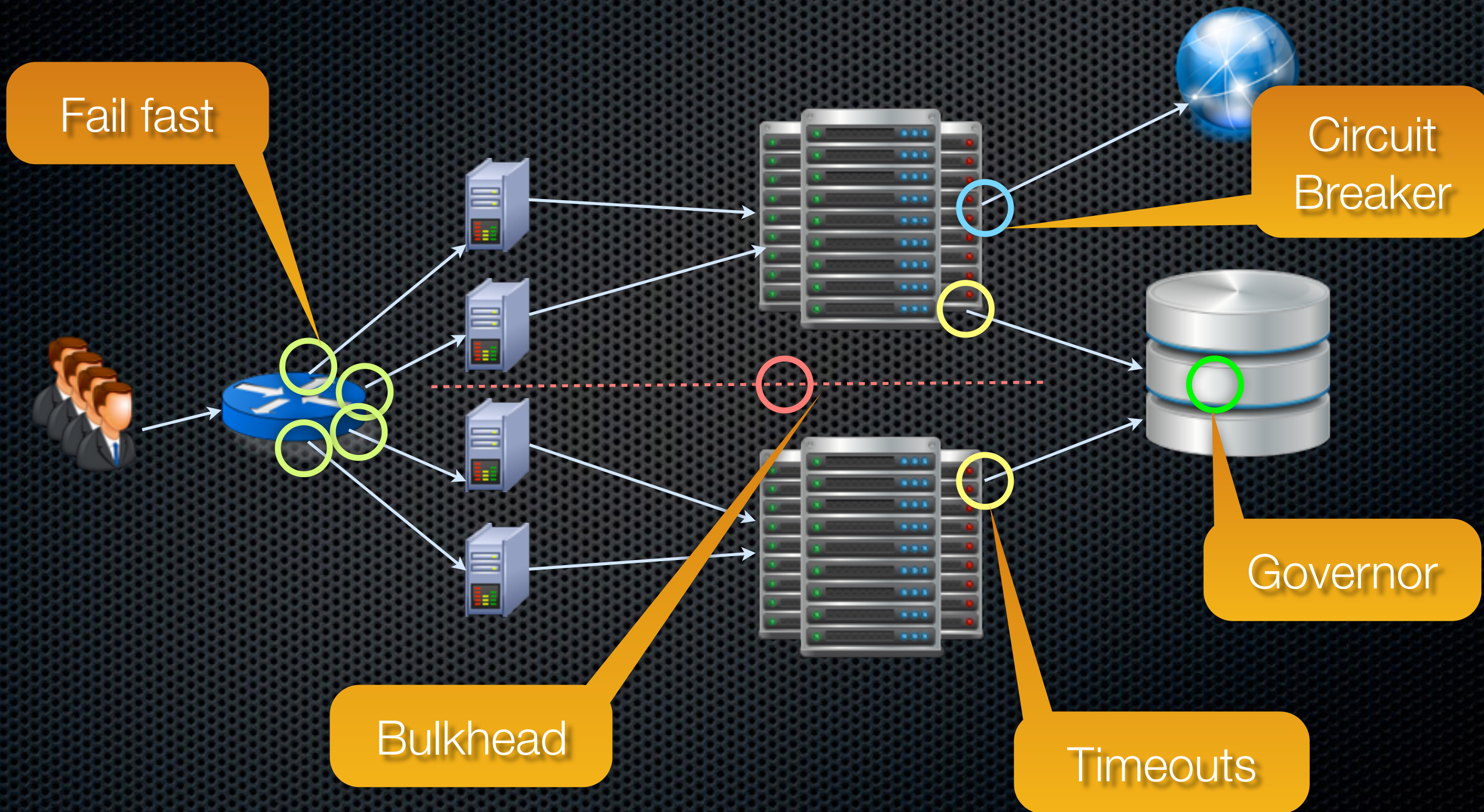
Stability - technology solutions



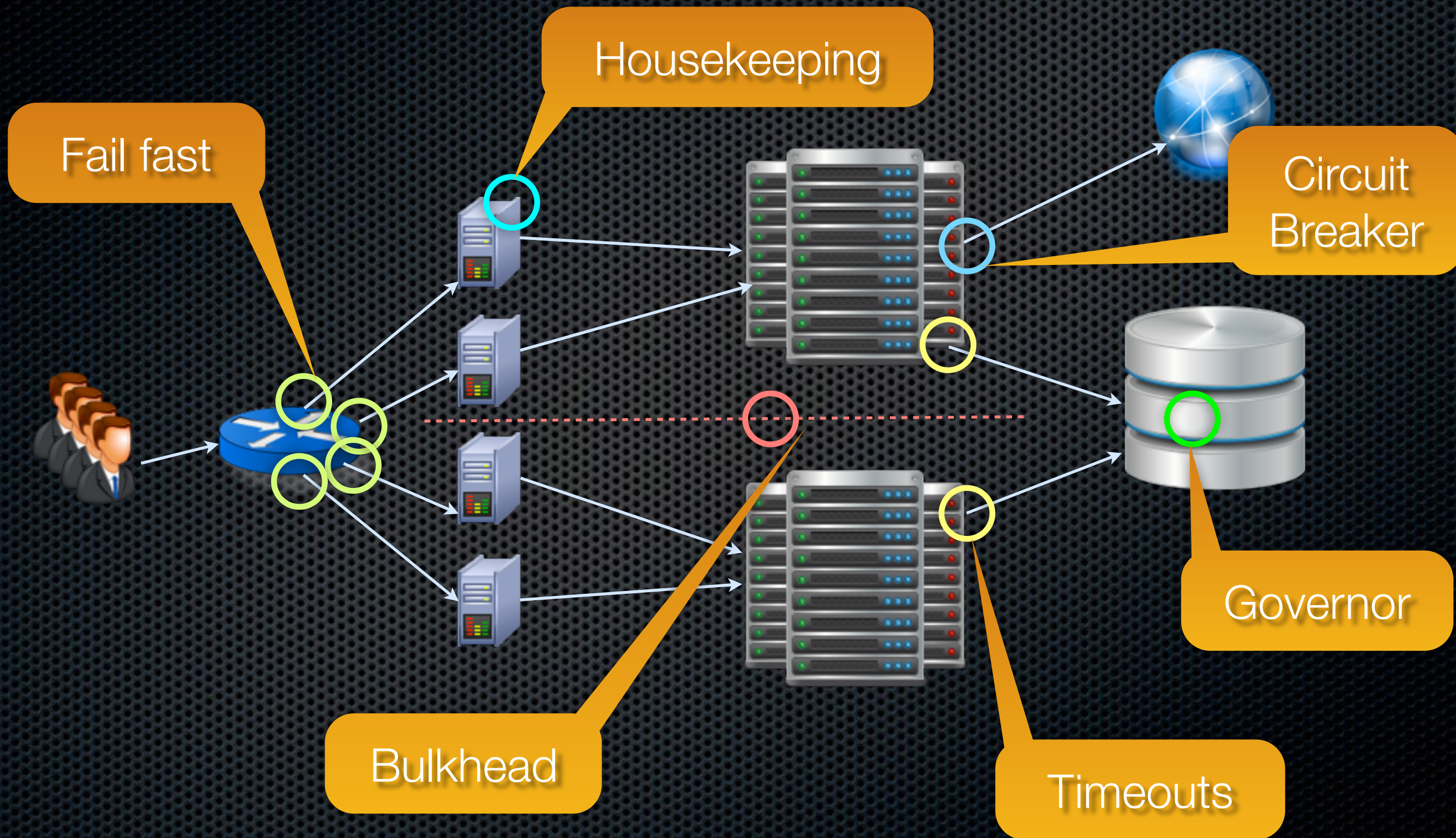
Stability - technology solutions



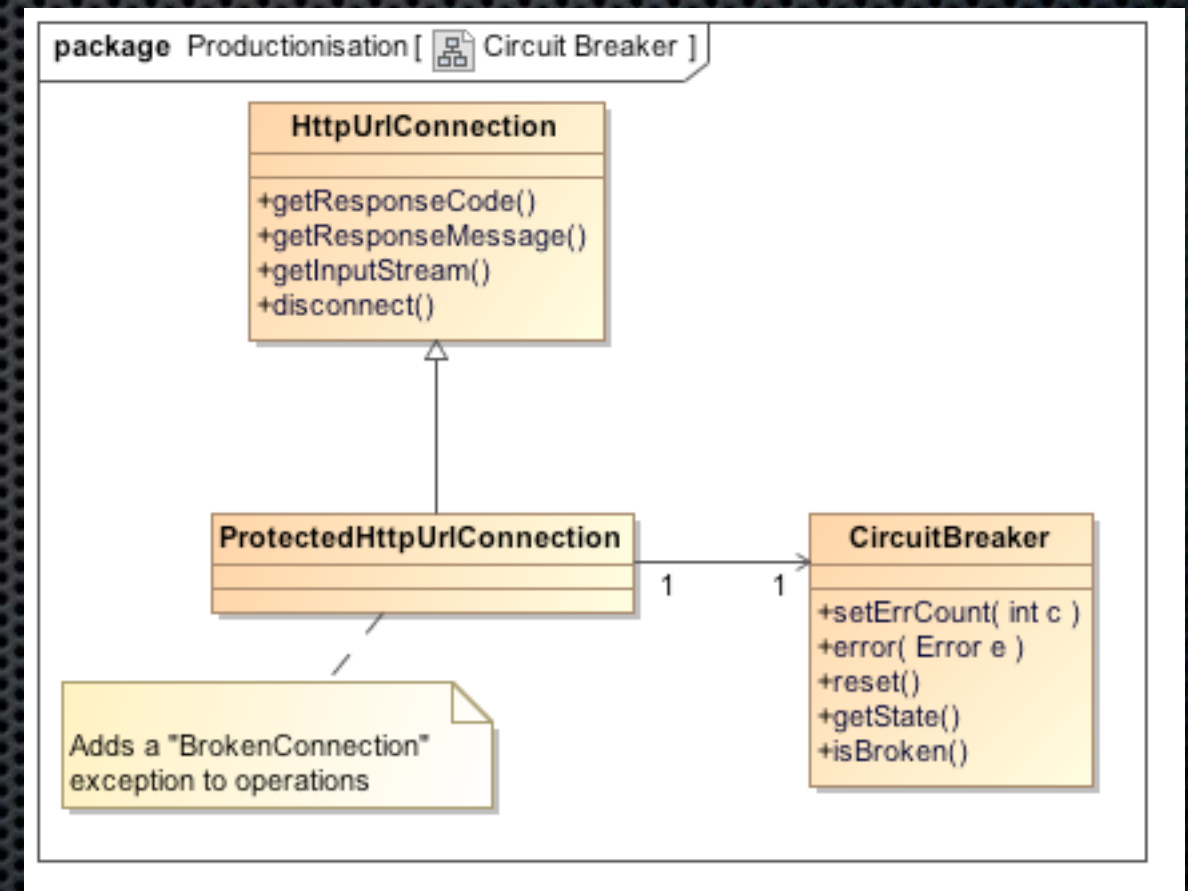
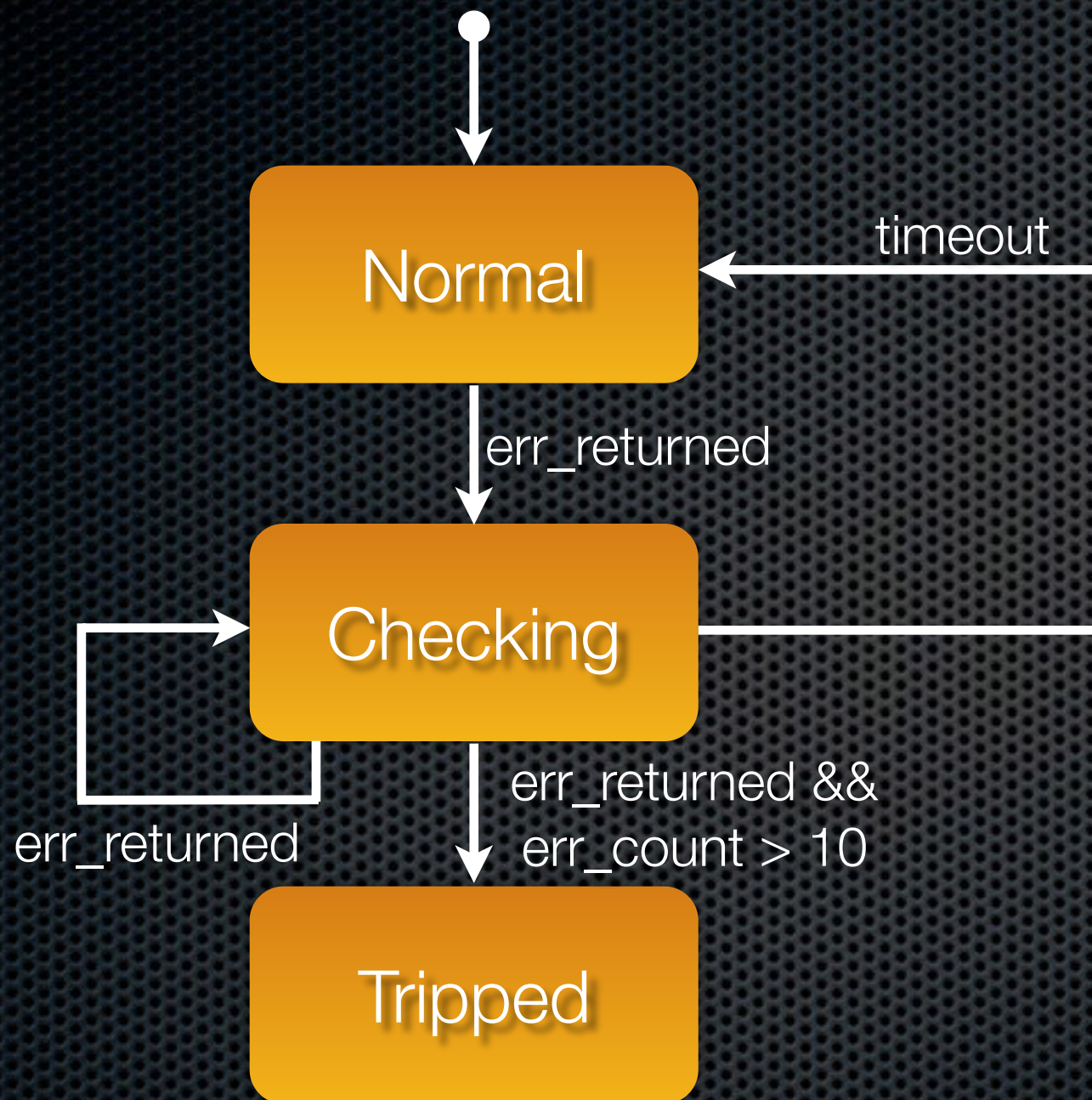
Stability - technology solutions



Stability - technology solutions



Example - Circuit Breaker



Stability - practices

✧ Repeatability

- ✧ defined processes, practice scenarios, prelive environments

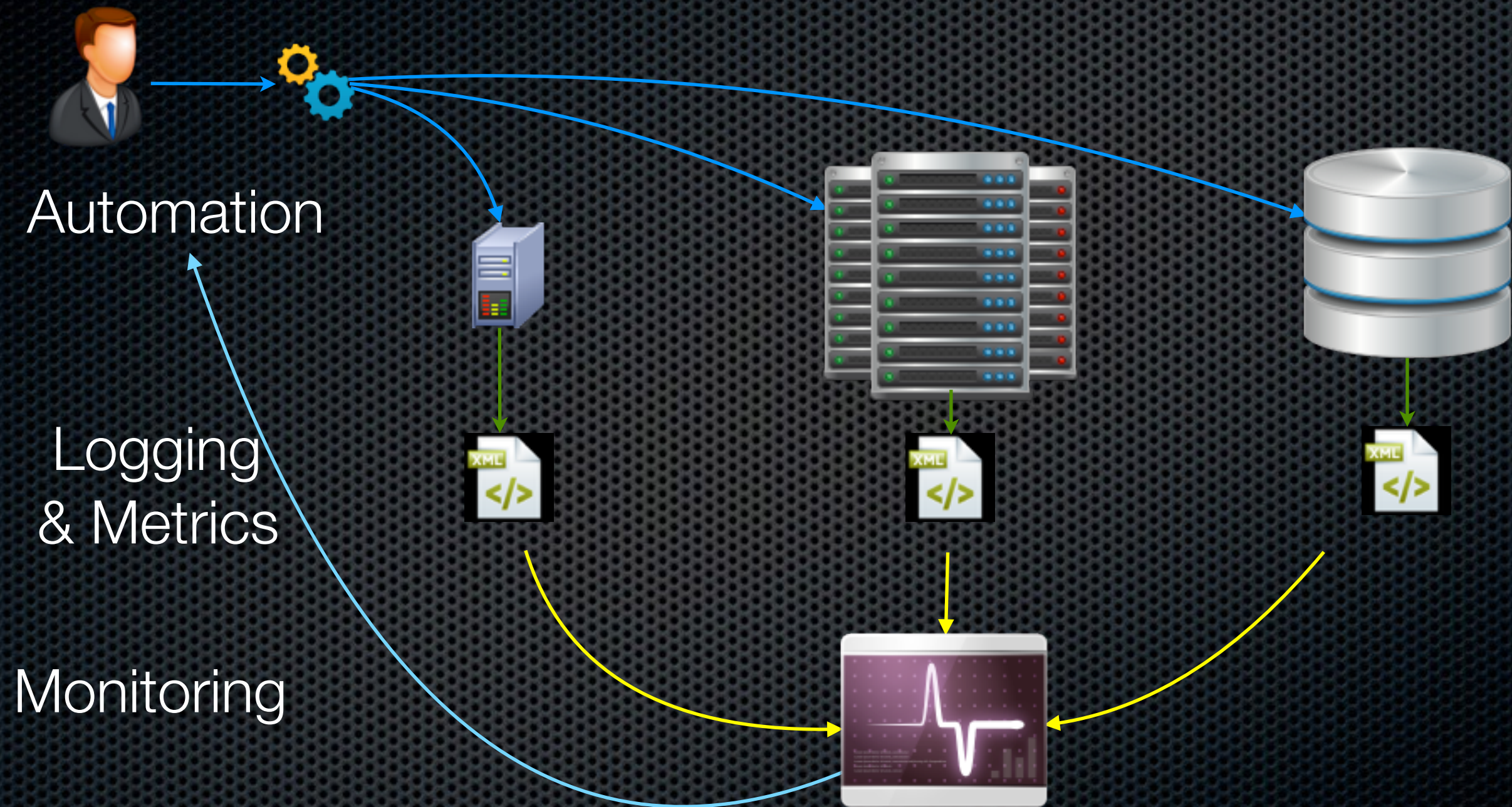
✧ Automation

- ✧ automate the routine, automate the difficult
- ✧ allow the human back in the loop on demand

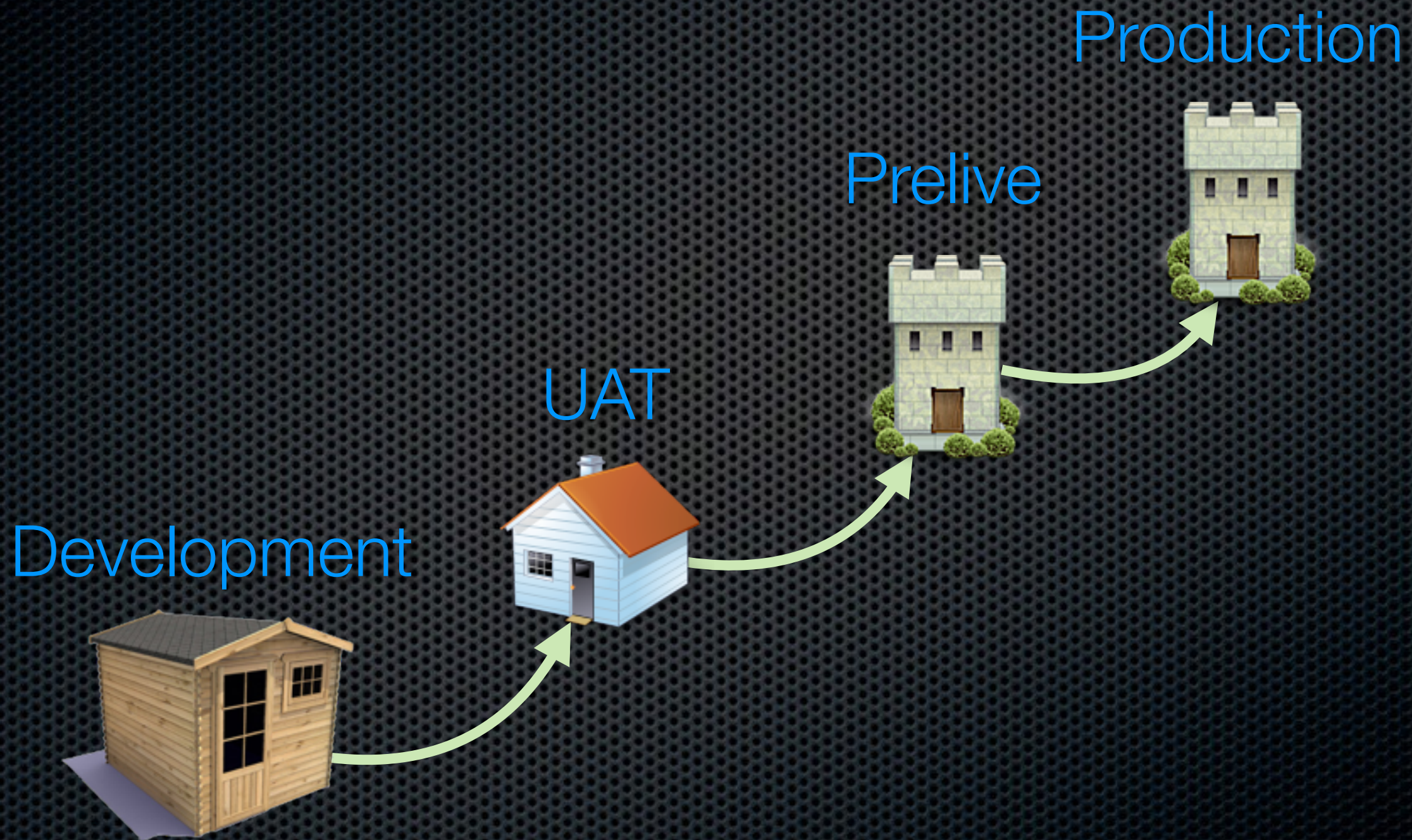
✧ Transparency

- ✧ logging, monitoring, alerts, trends

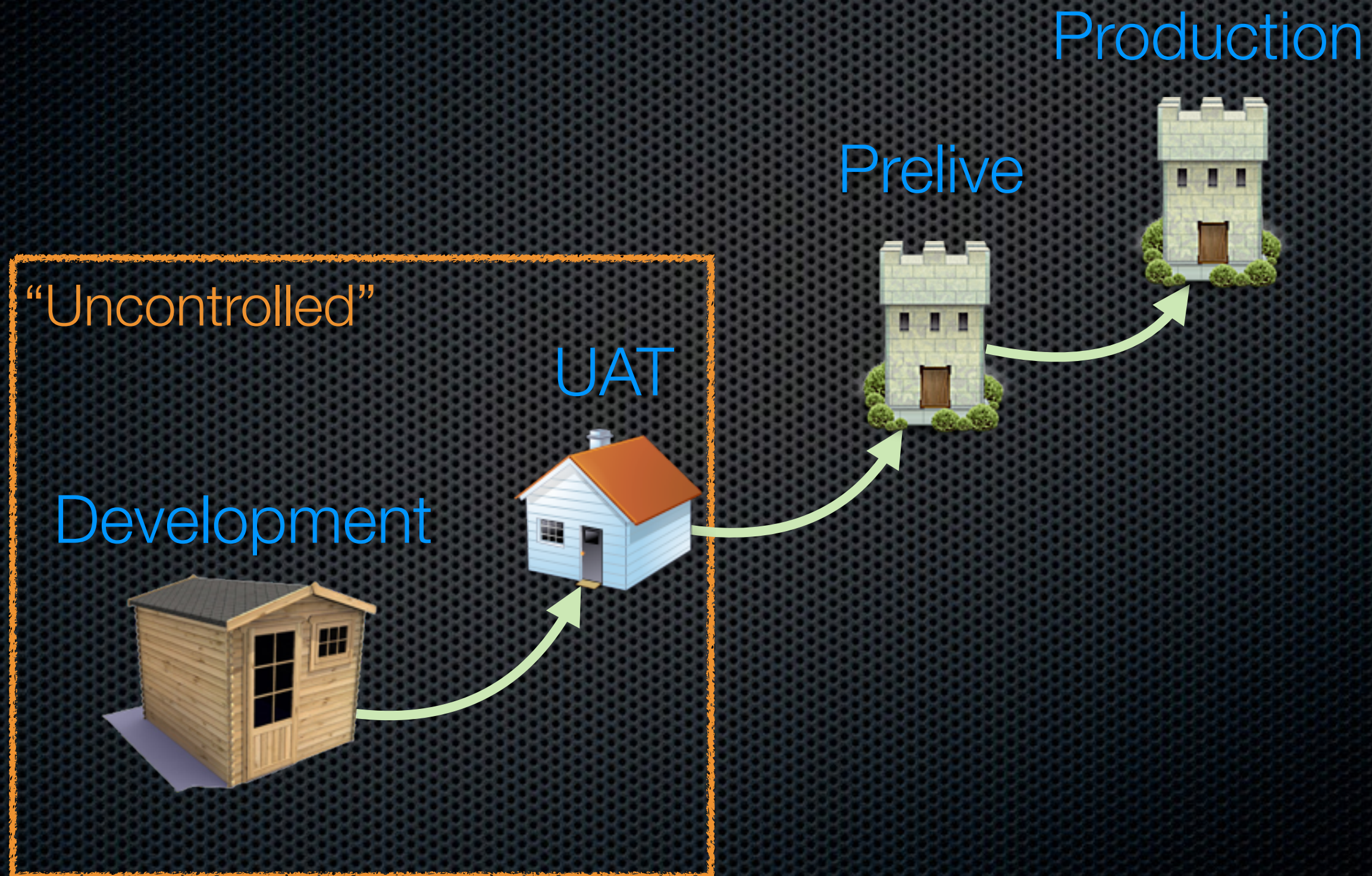
Stability - process automation



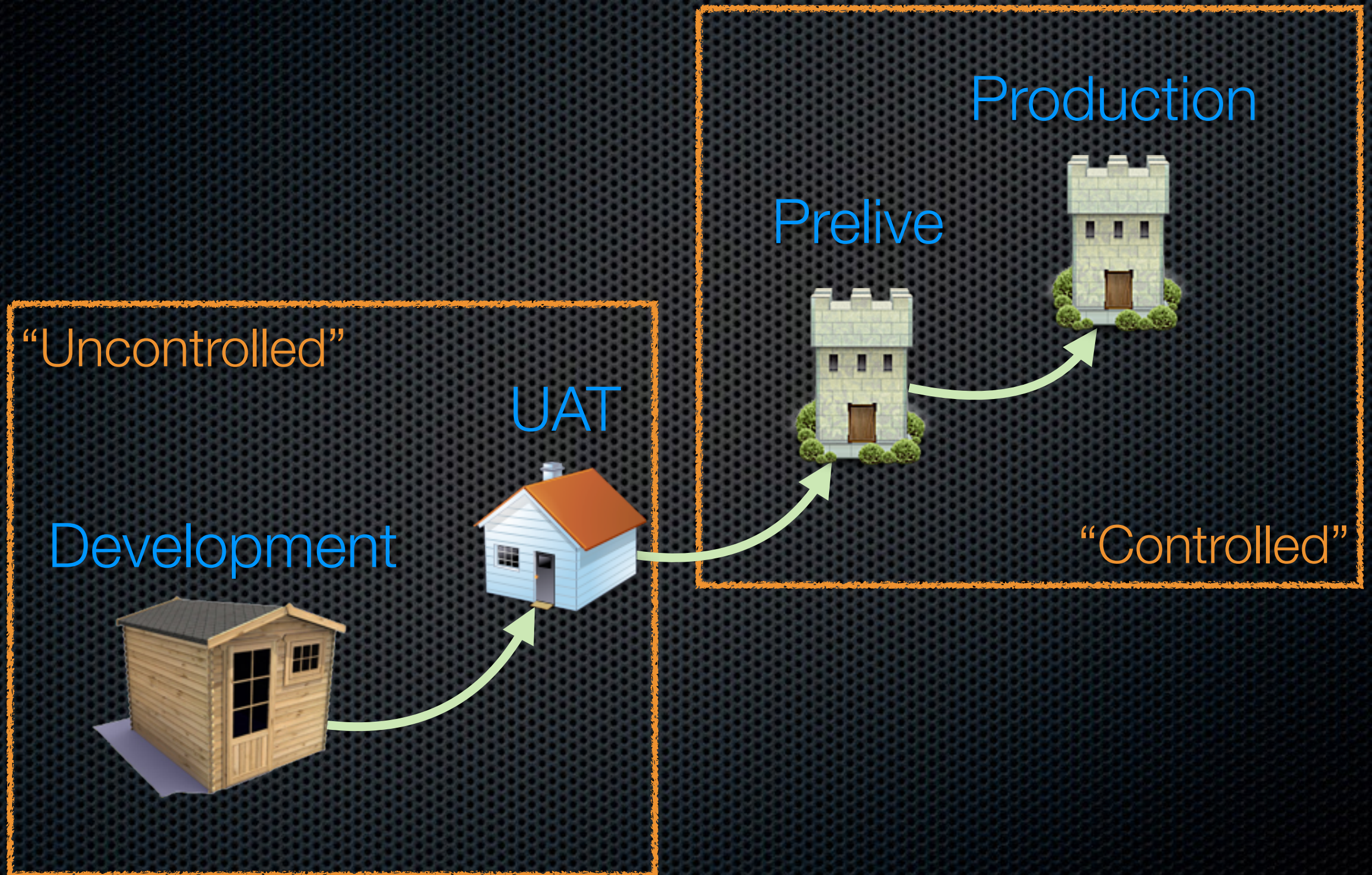
Stability - environments



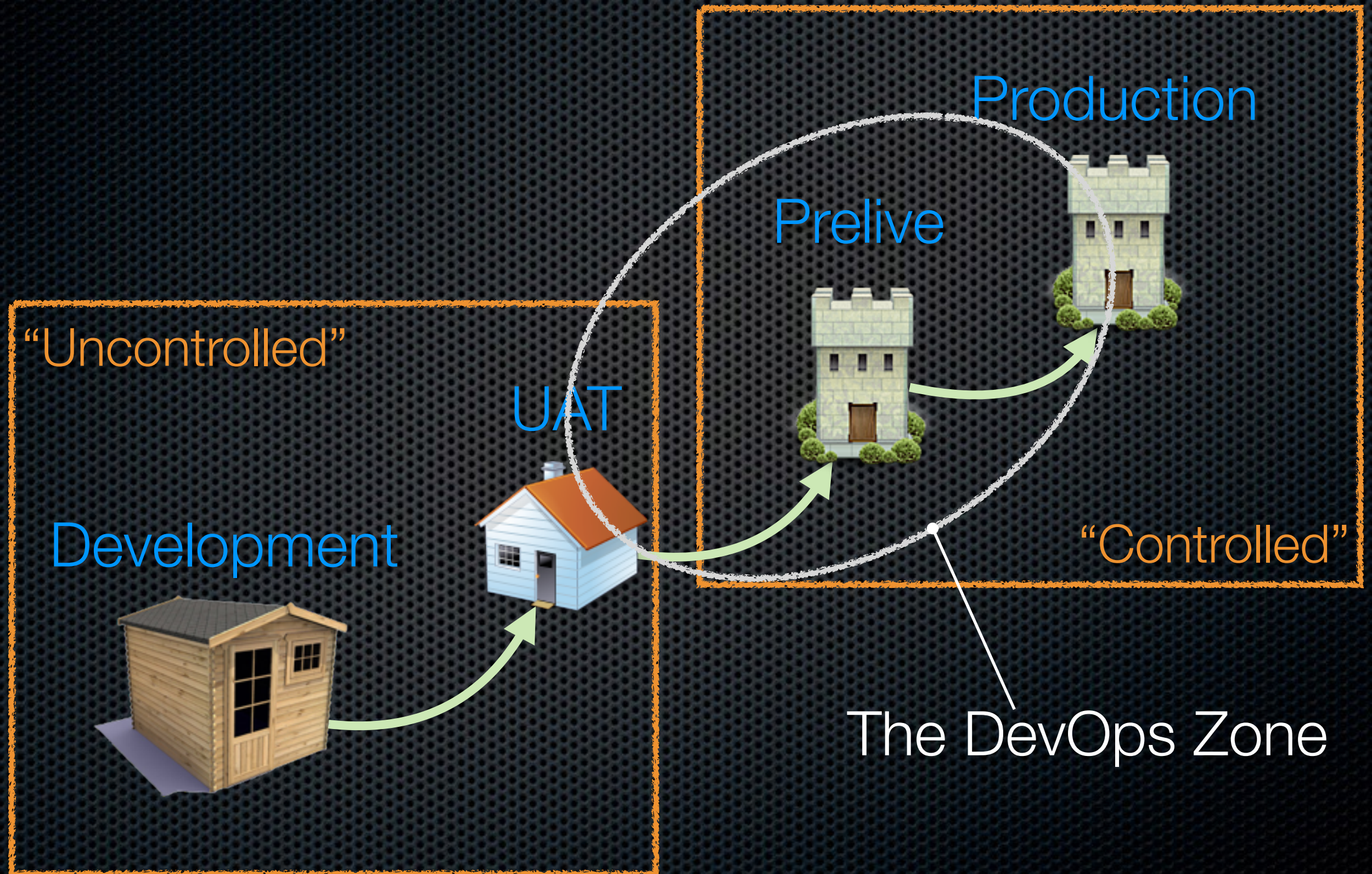
Stability - environments



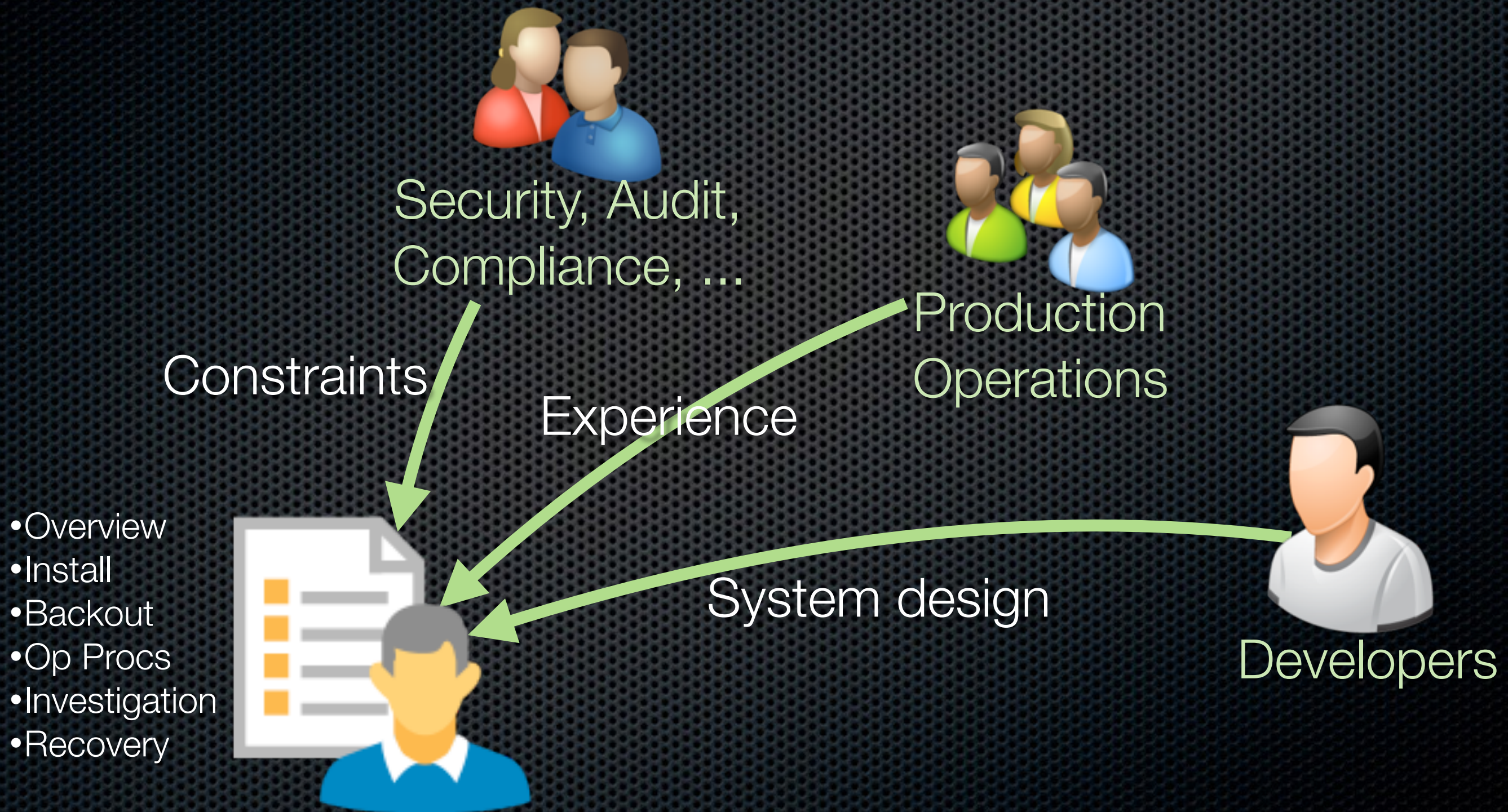
Stability - environments



Stability - environments



Stability - production runbooks



Solutions: Achieving Capacity



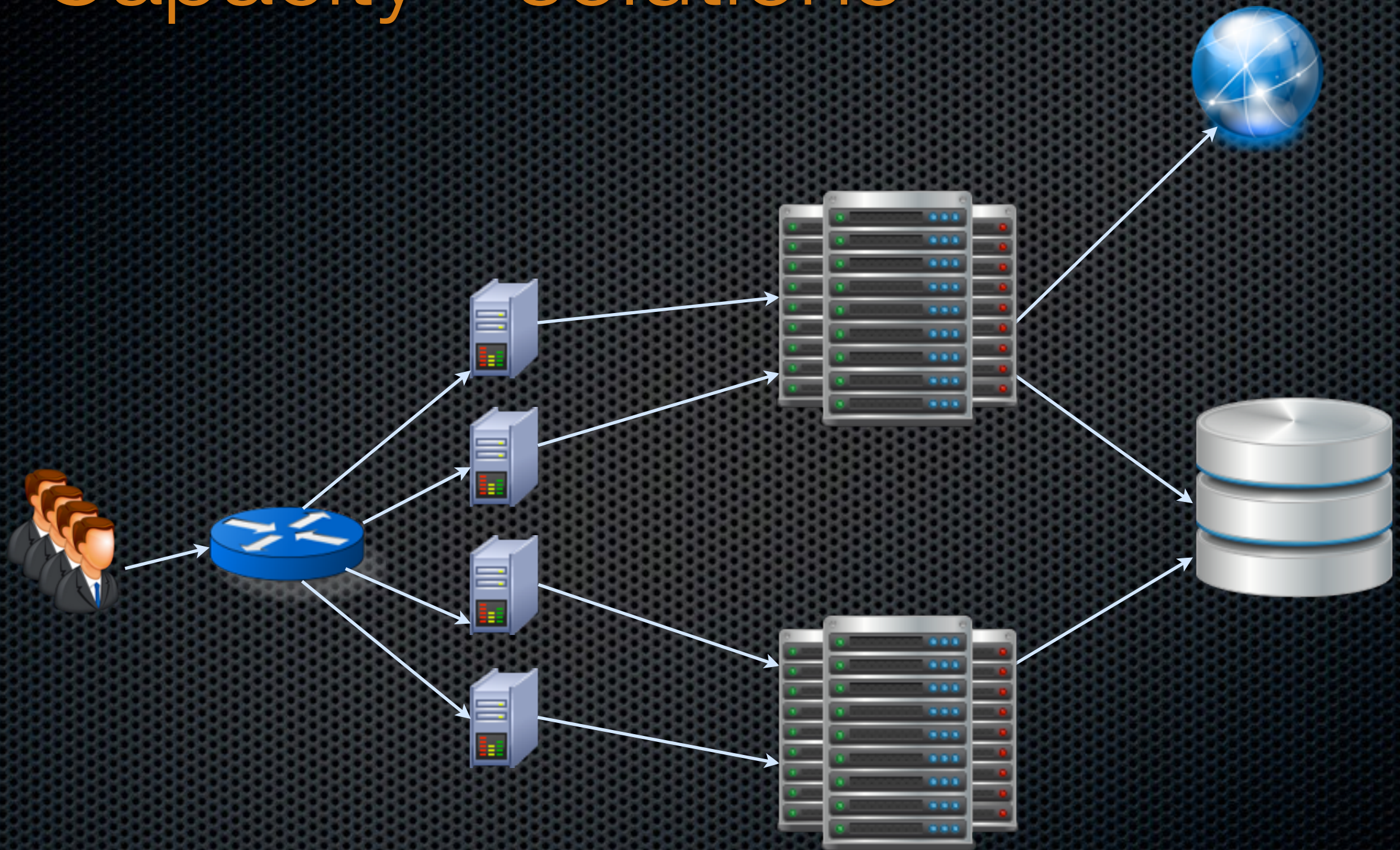
Capacity - design principles

- ✦ Minimise workload
 - ✦ efficiency is important
- ✦ Flatten the peaks
 - ✦ move workload around
- ✦ Design for the large (scalability)
 - ✦ understand where the time goes
 - ✦ multiply by a million

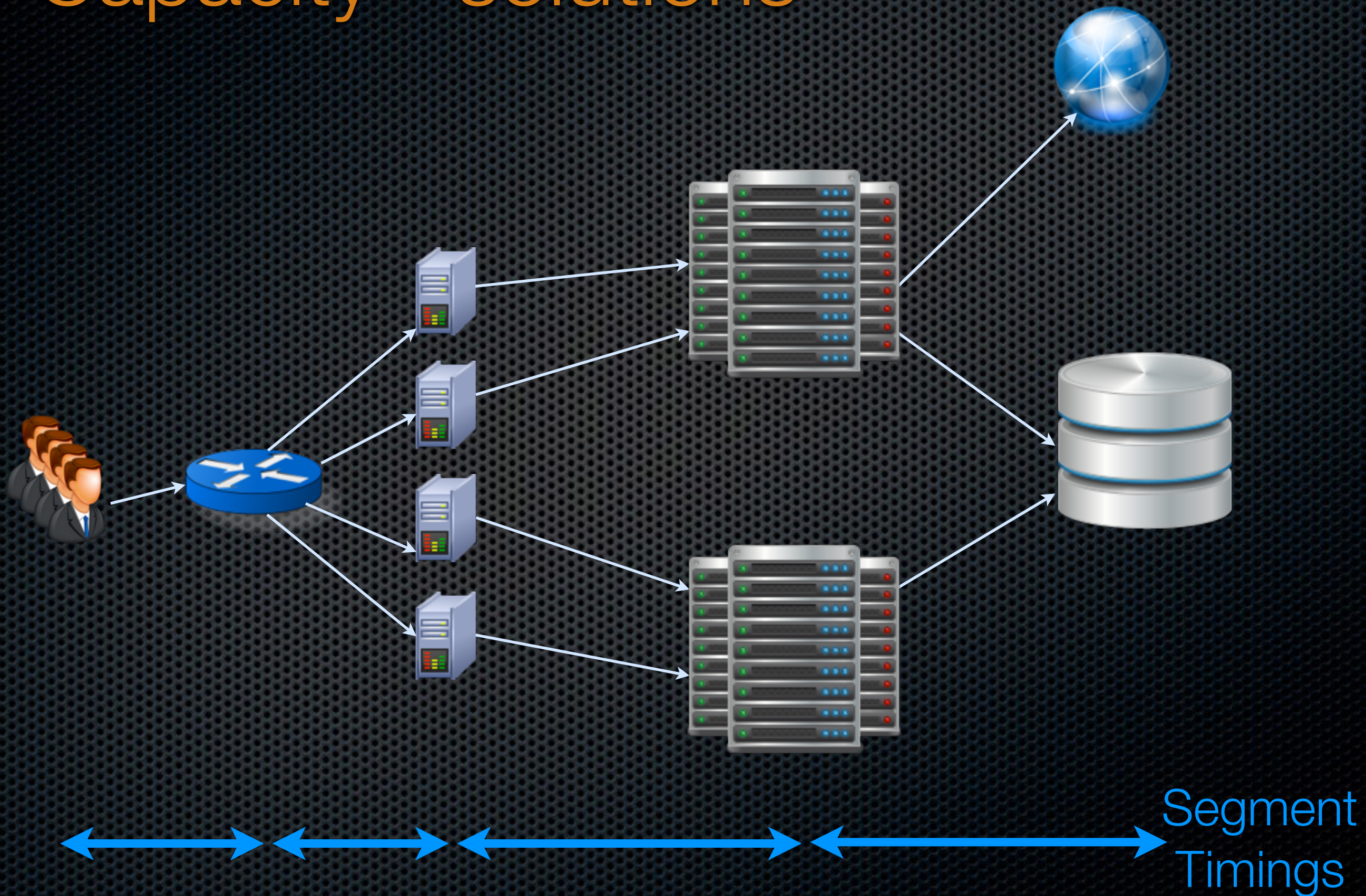
Capacity - technology solutions

- ✦ Measure and minimise
 - ✦ understand where the work is
- ✦ Caching and pre-computing
 - ✦ reduce the work to be done
- ✦ Sharding and partitioning
 - ✦ separate workload to allow scale

Capacity - solutions

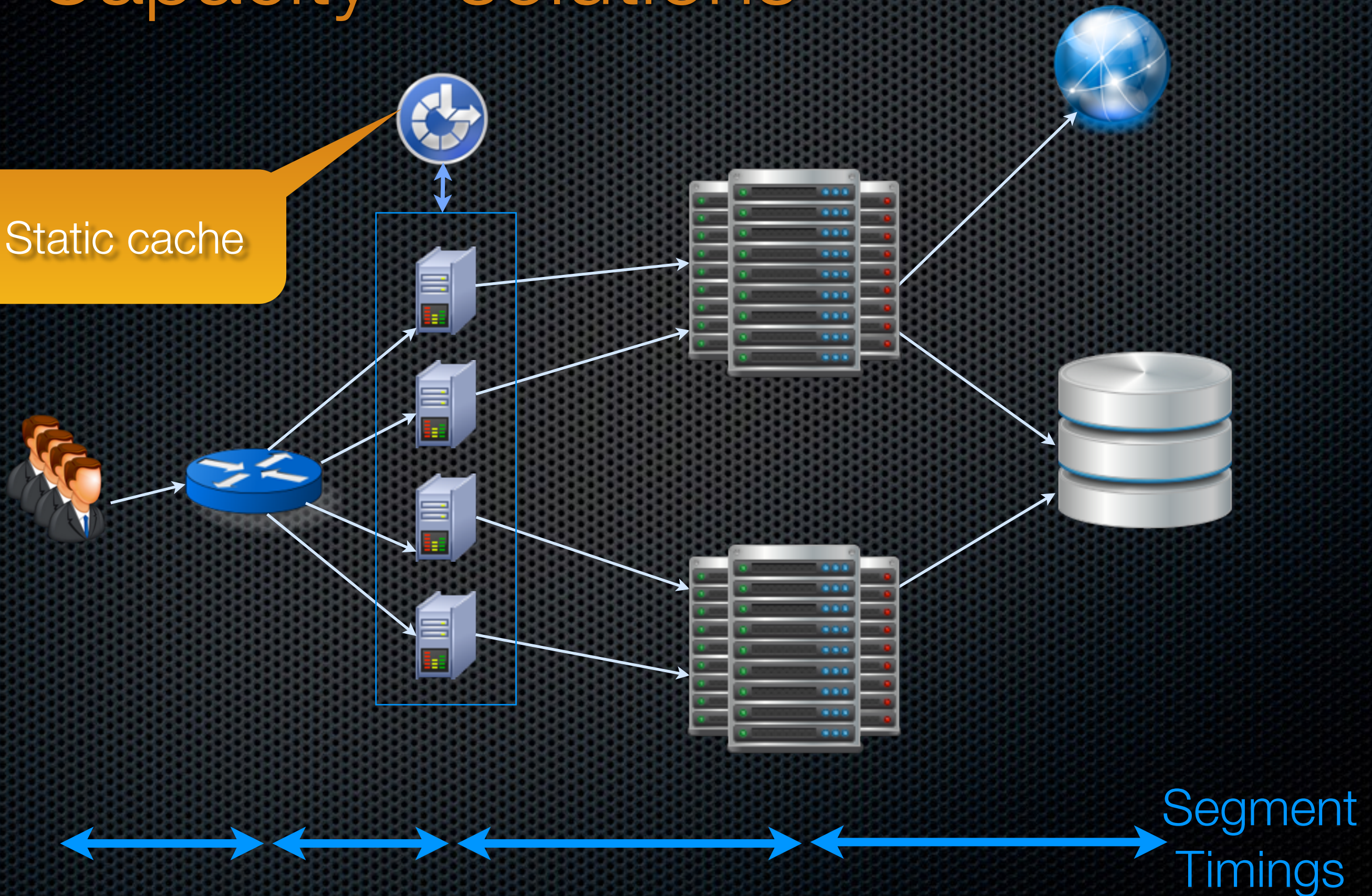


Capacity - solutions



Capacity - solutions

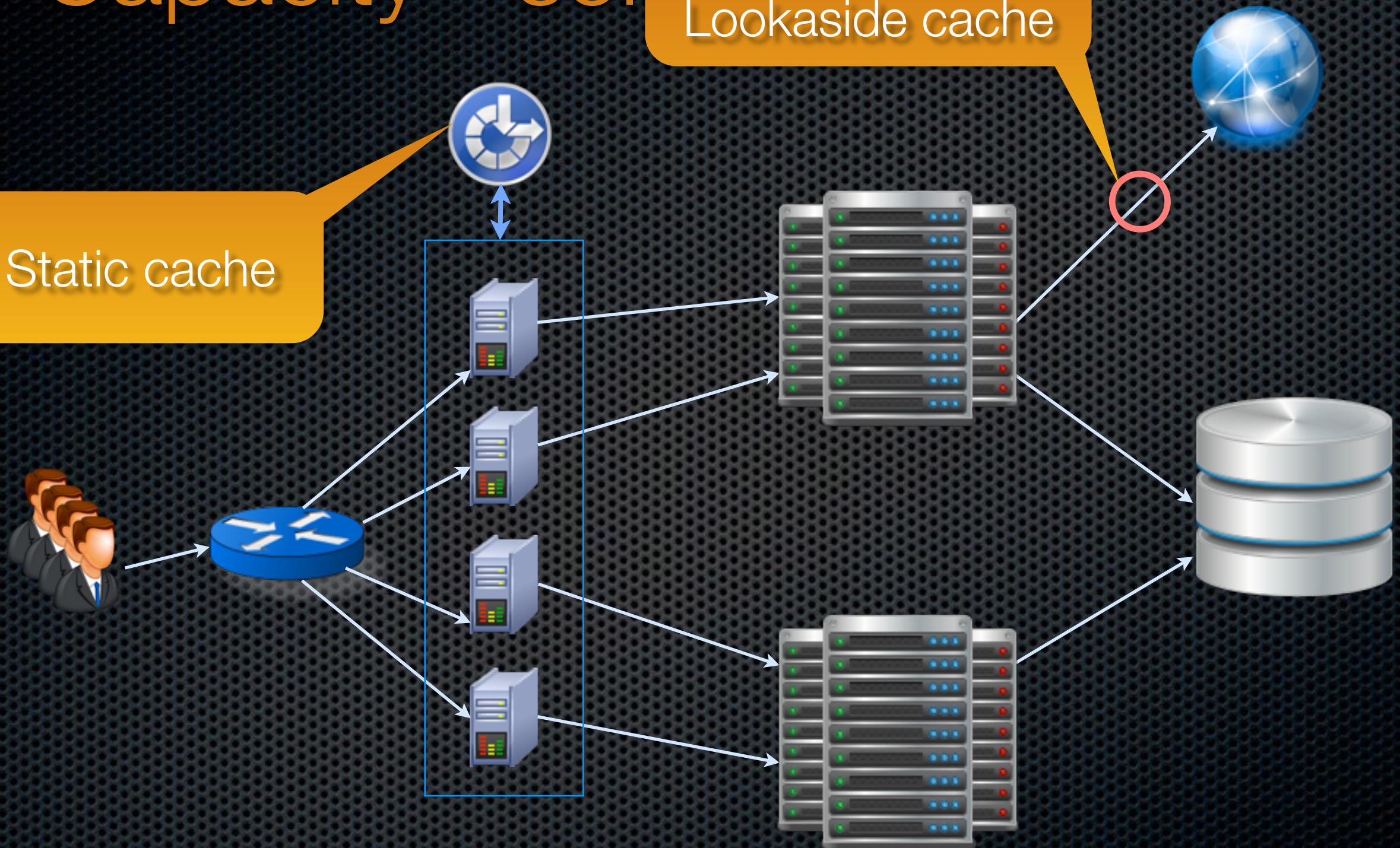
Static cache



Capacity - solutions

Lookaside cache

Static cache

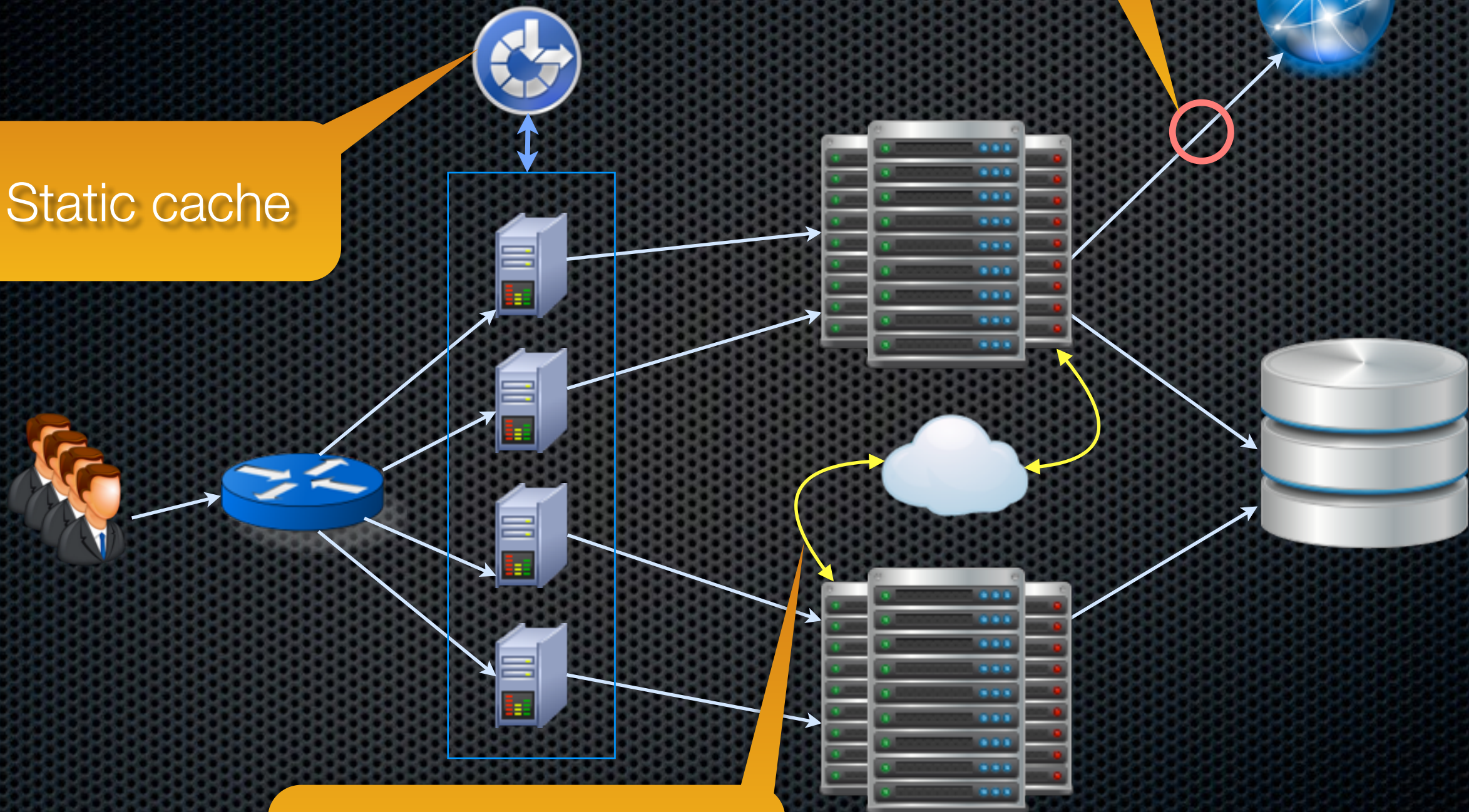


Segment
Timings

Capacity - solutions

Lookaside cache

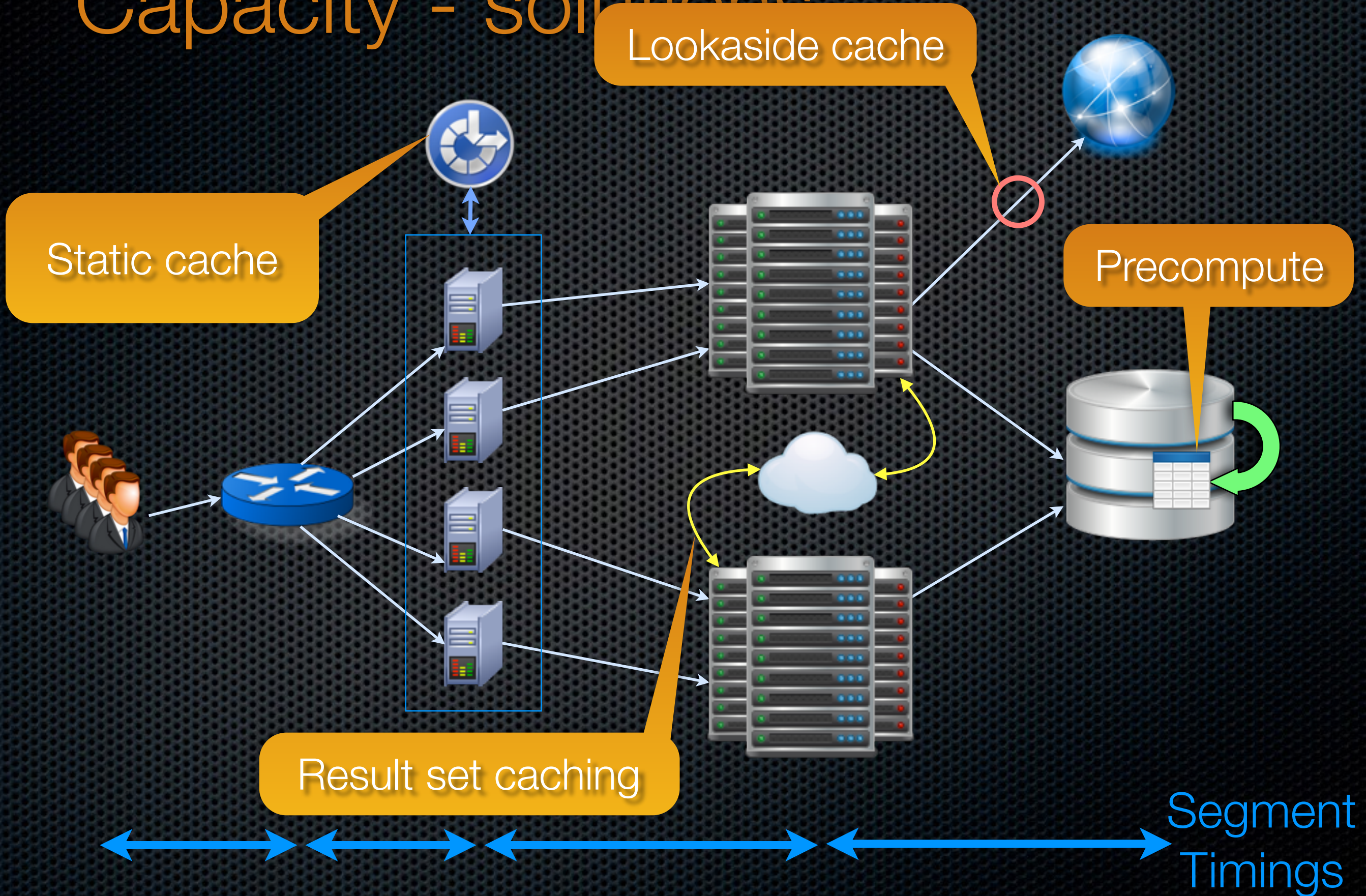
Static cache



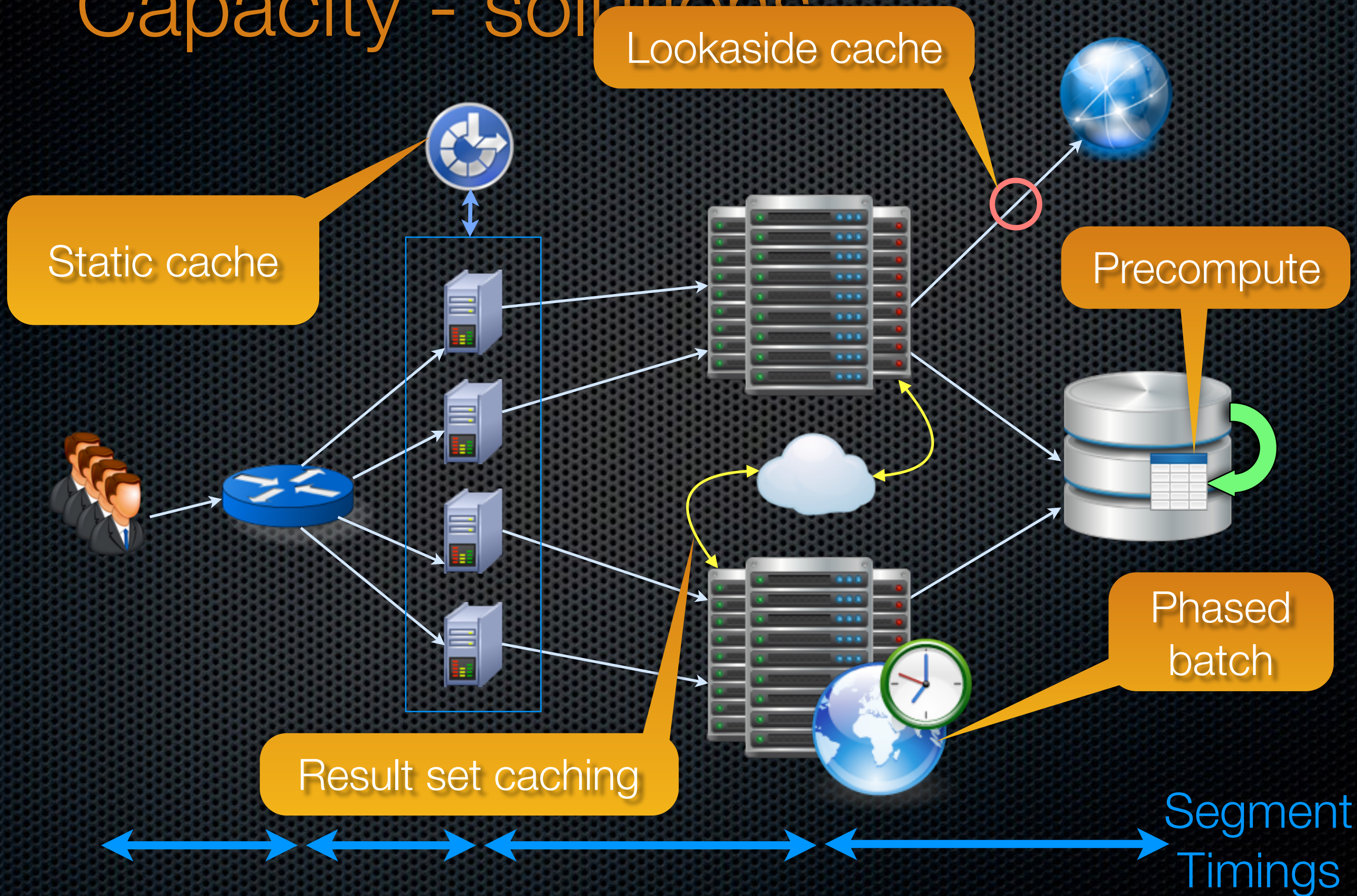
Result set caching

Segment
Timings

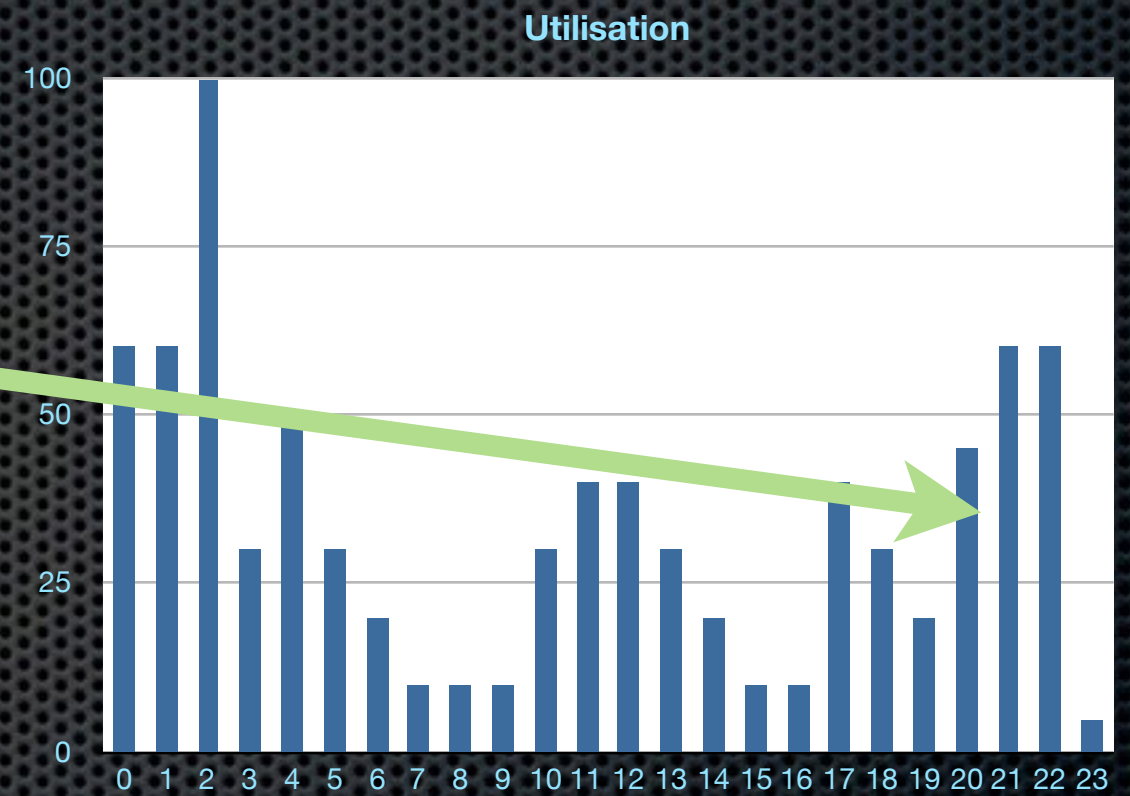
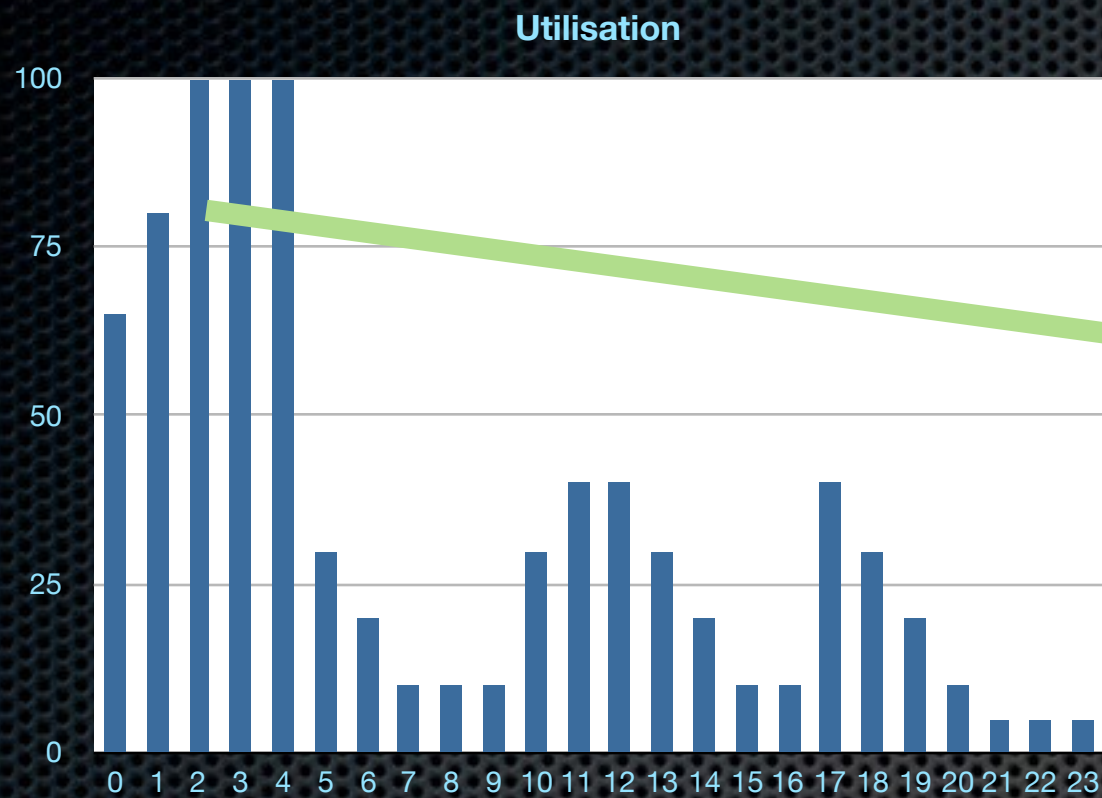
Capacity - solutions



Capacity - solutions



Moving Work Around



Capacity - practices

- ✧ Model and estimate
- ✧ Test capacity on realistic environments
 - ✧ allows model calibration
- ✧ Monitoring and trend analysis
 - ✧ tests theory against reality
 - ✧ spots impending storms before they hit

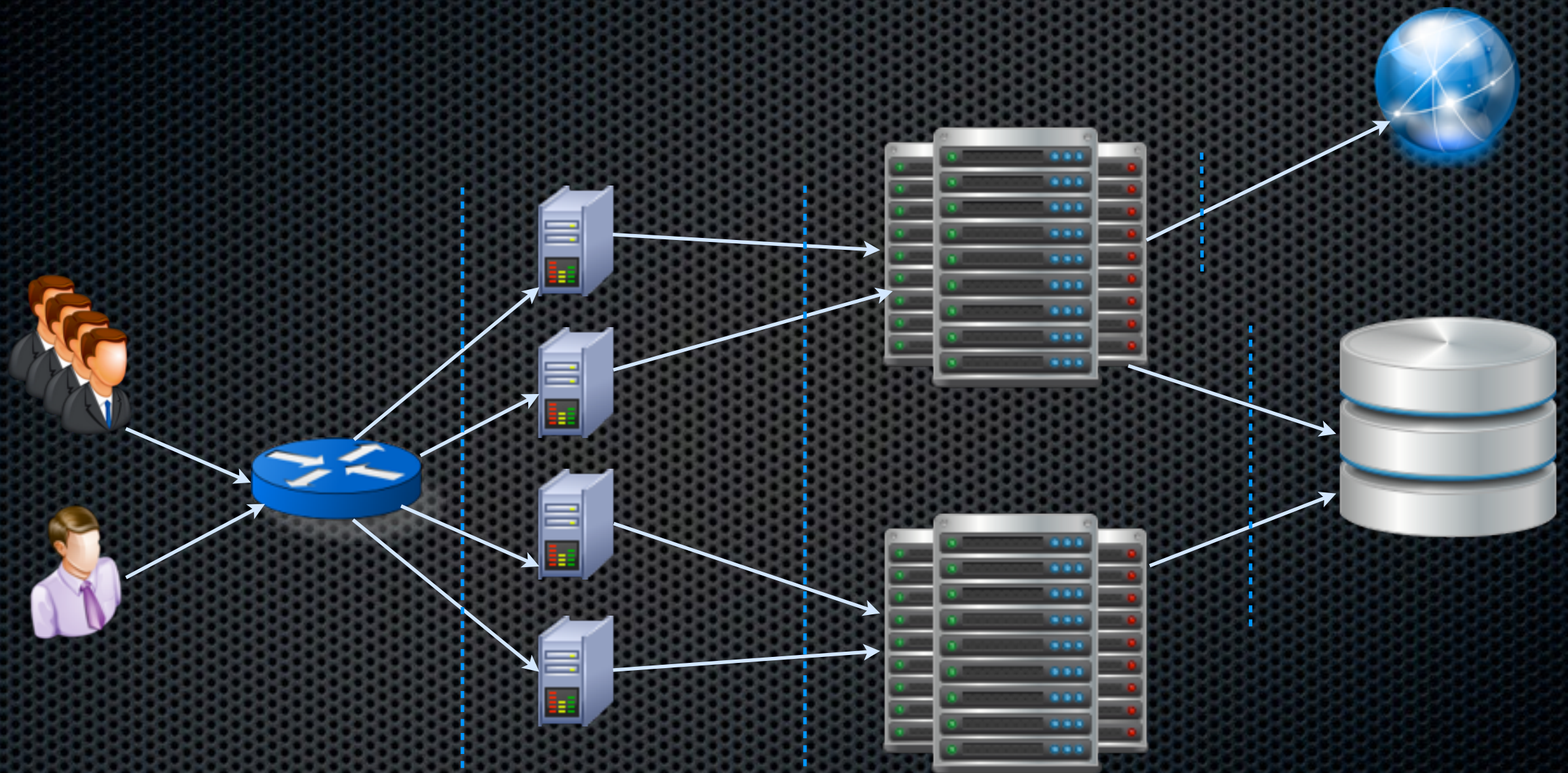
Solutions: Achieving Security



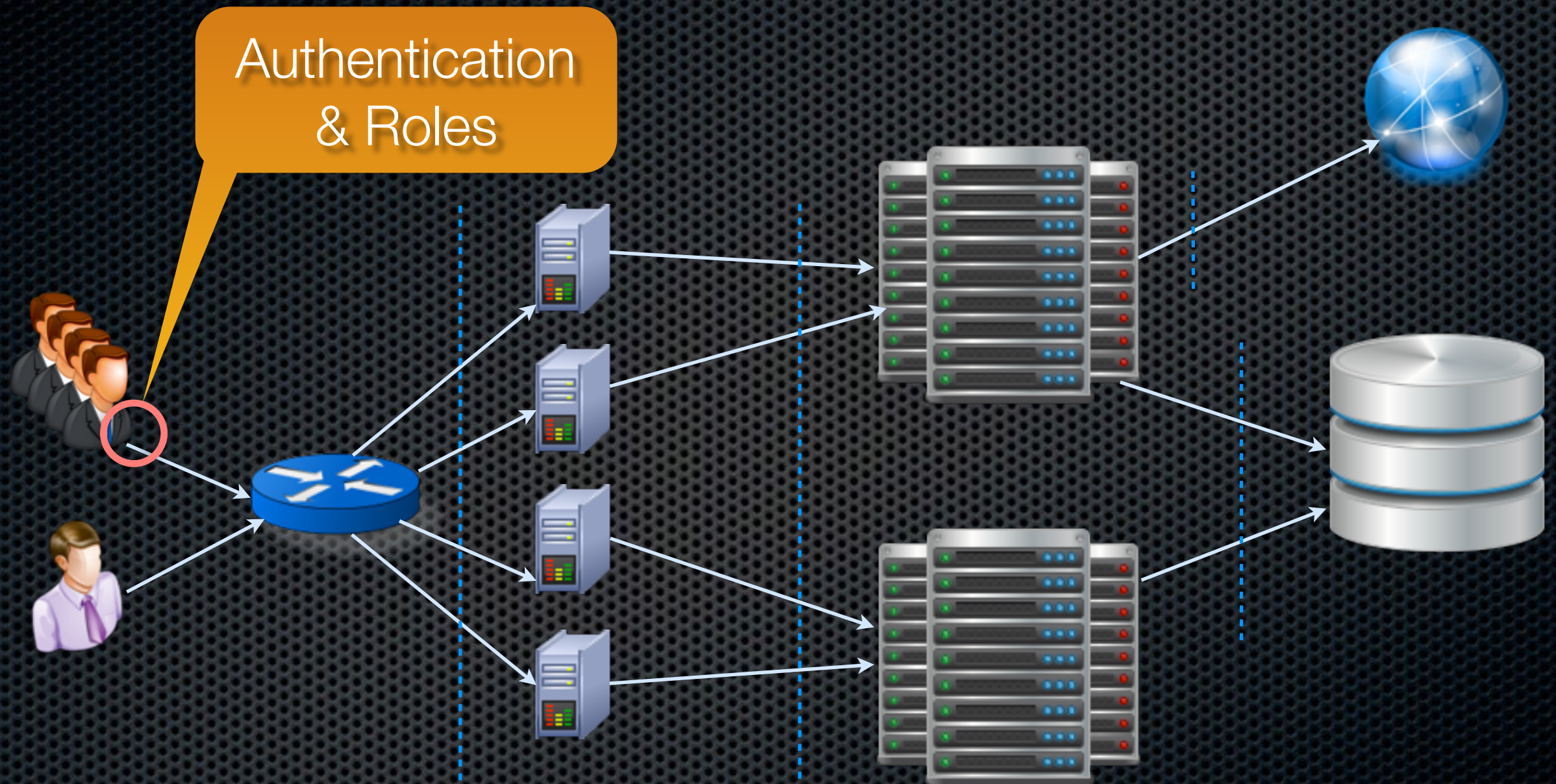
Security - key design principles

- ✦ What they don't have won't hurt you
 - ✦ least privilege - grant the minimum needed
- ✦ Security needs simplicity
 - ✦ what you can't analyse you can't be sure about
- ✦ Don't put your eggs in one basket
 - ✦ separate privileges to avoid total breaches
- ✦ Fail safely

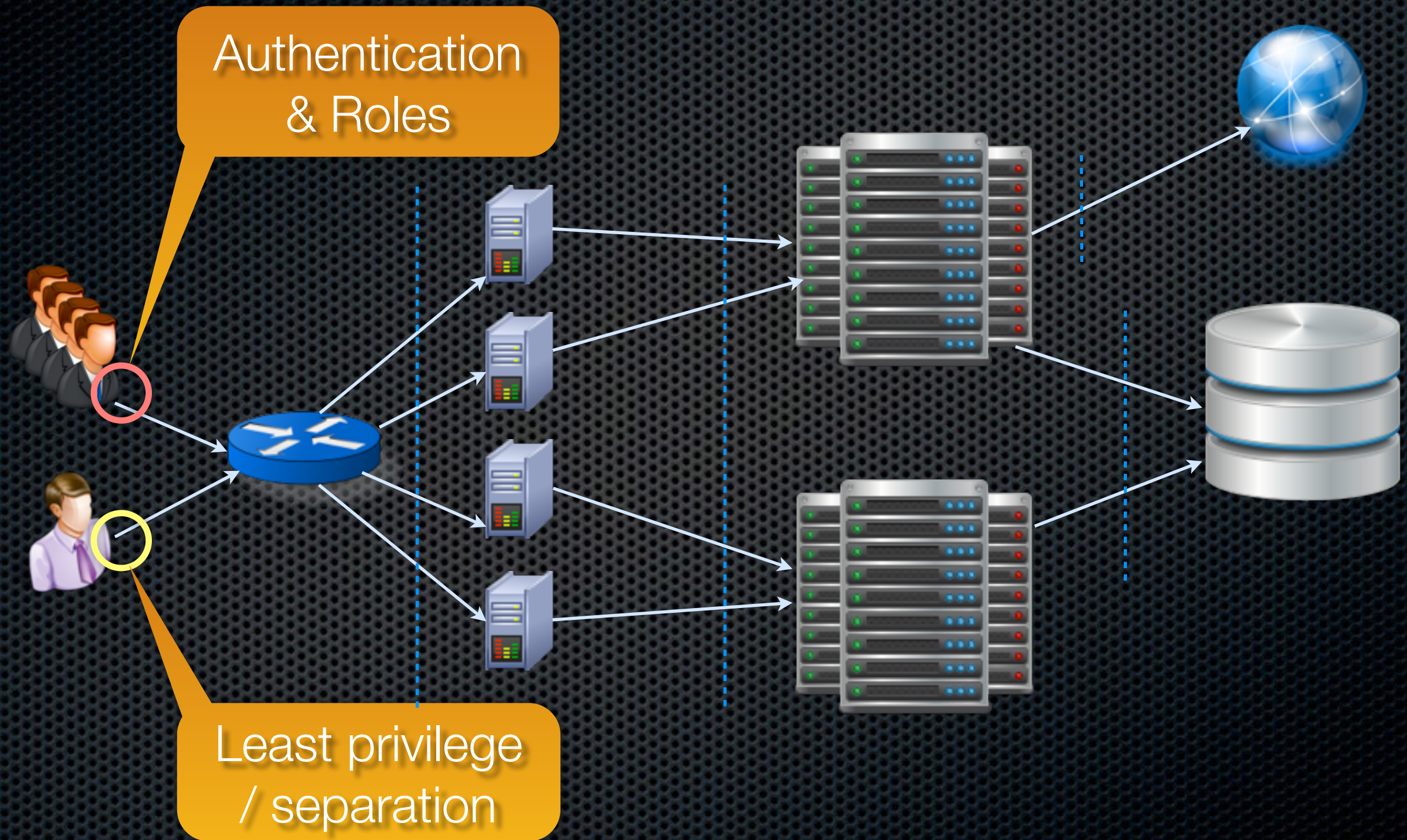
Security - solutions



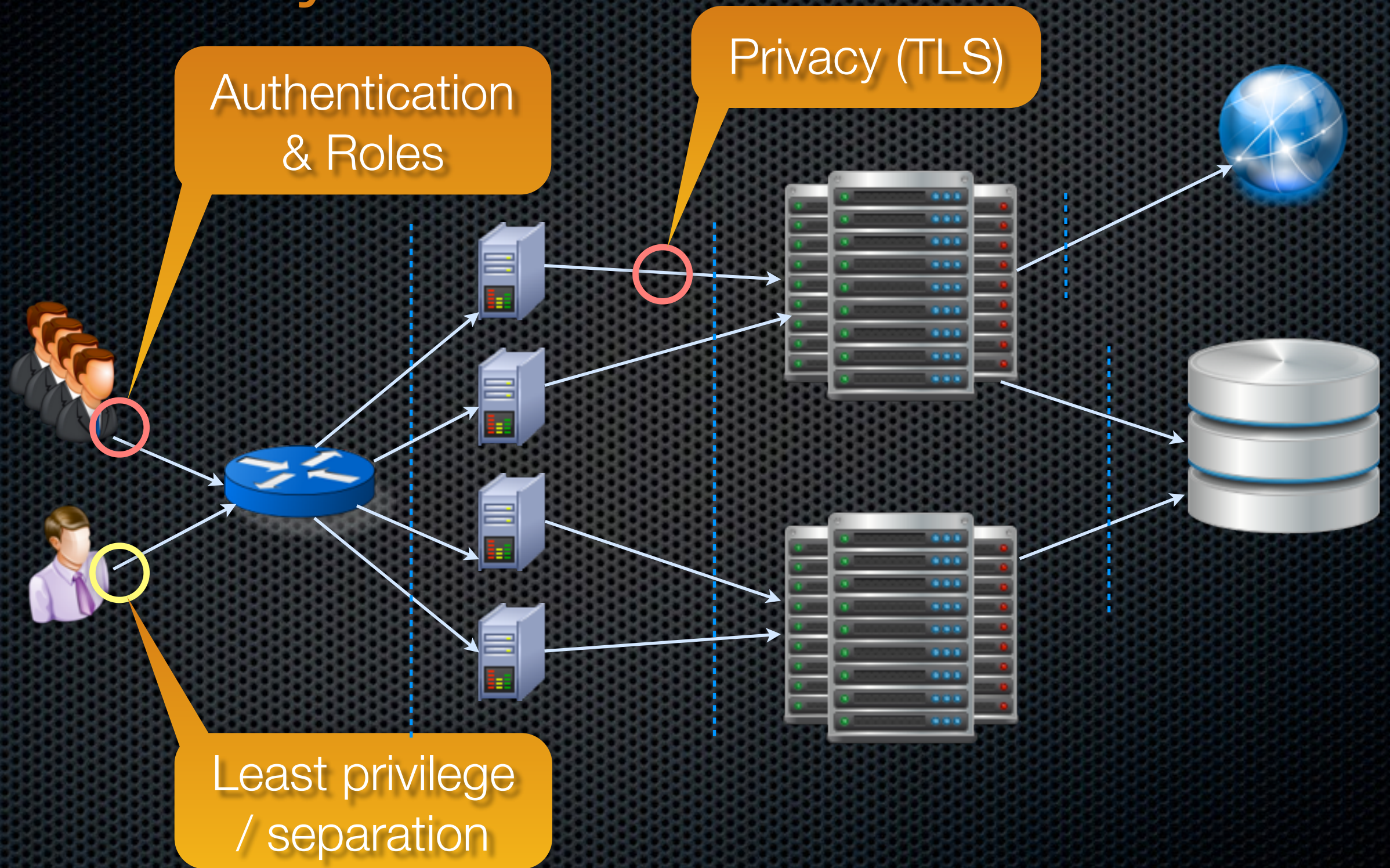
Security - solutions



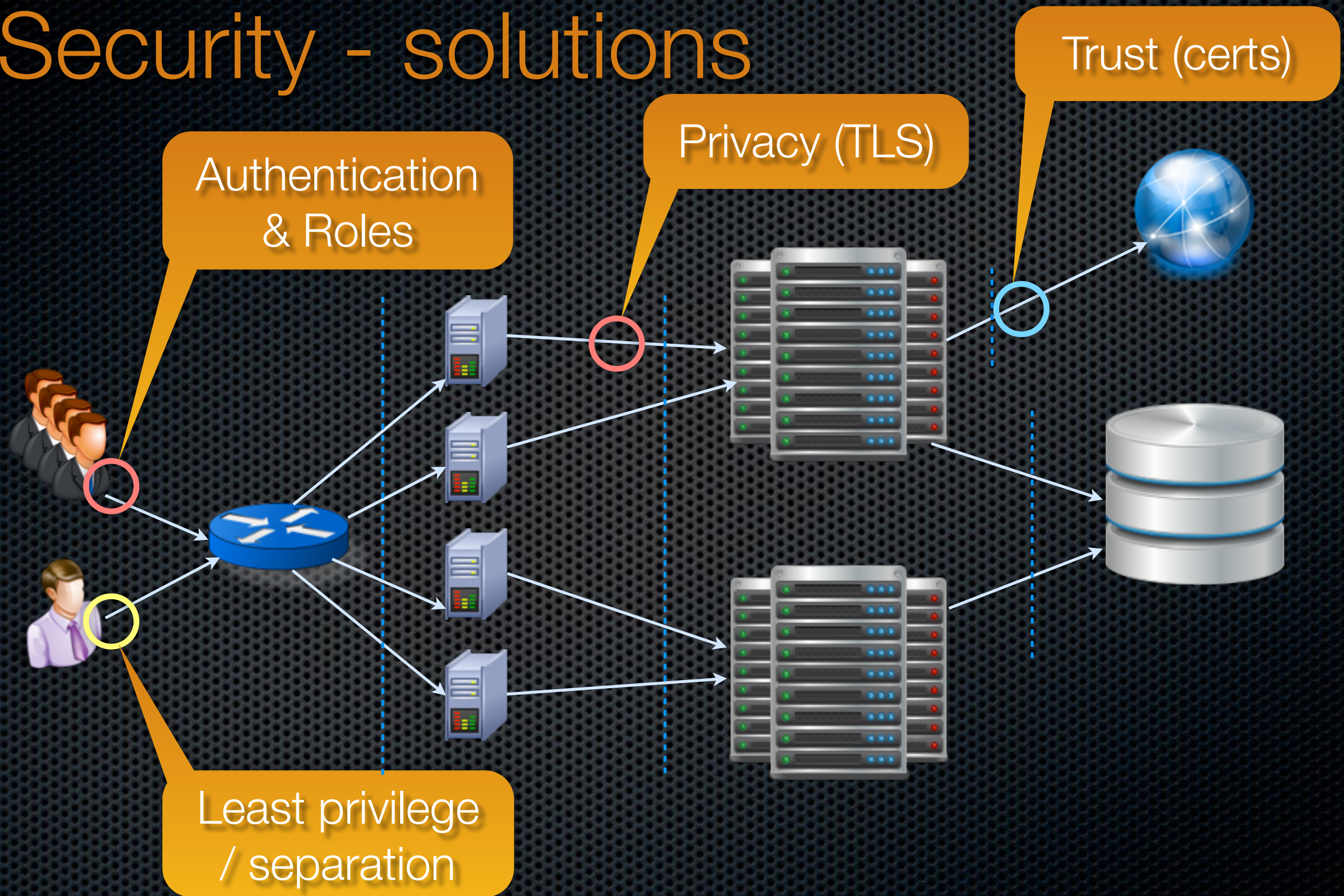
Security - solutions



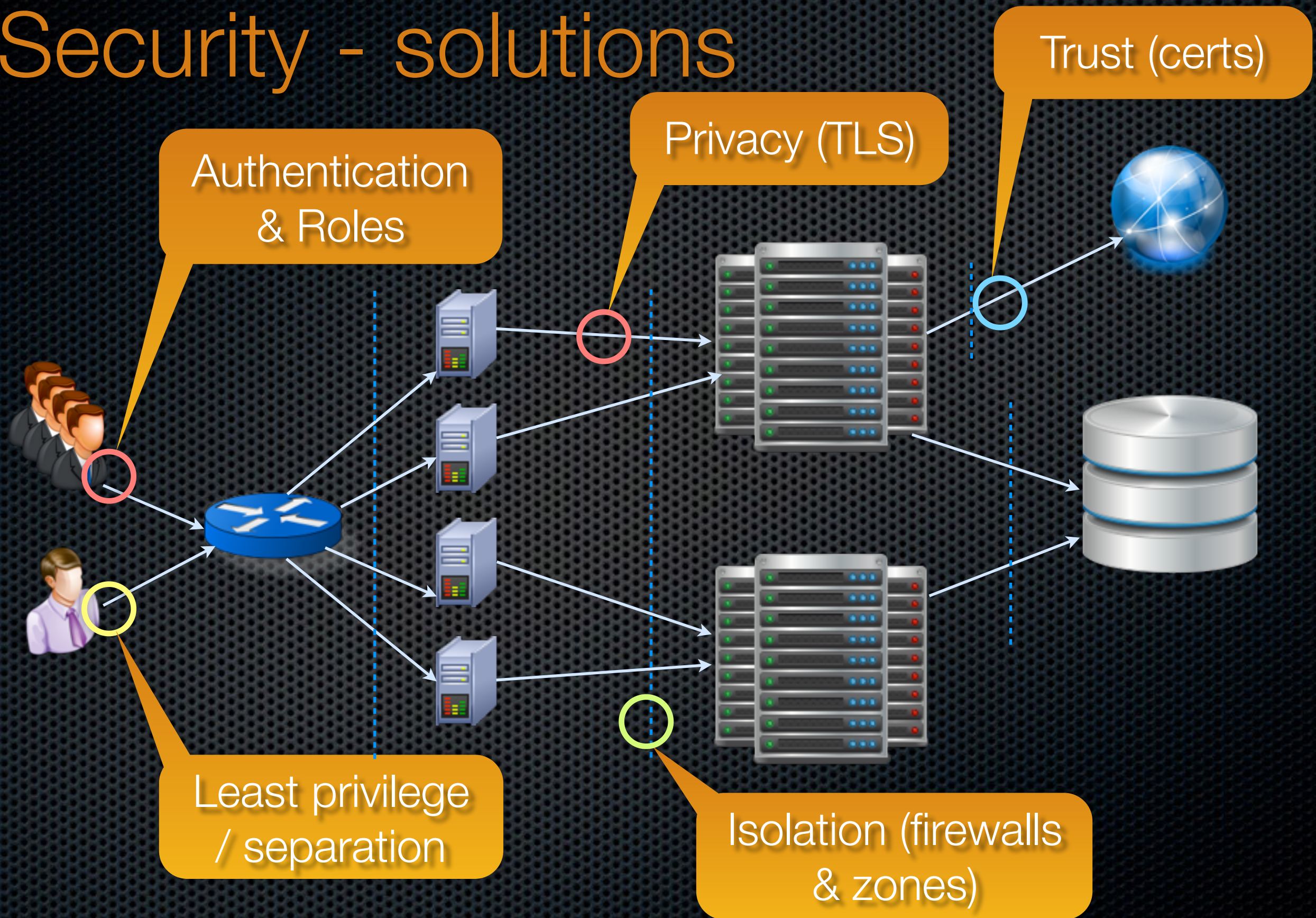
Security - solutions



Security - solutions



Security - solutions

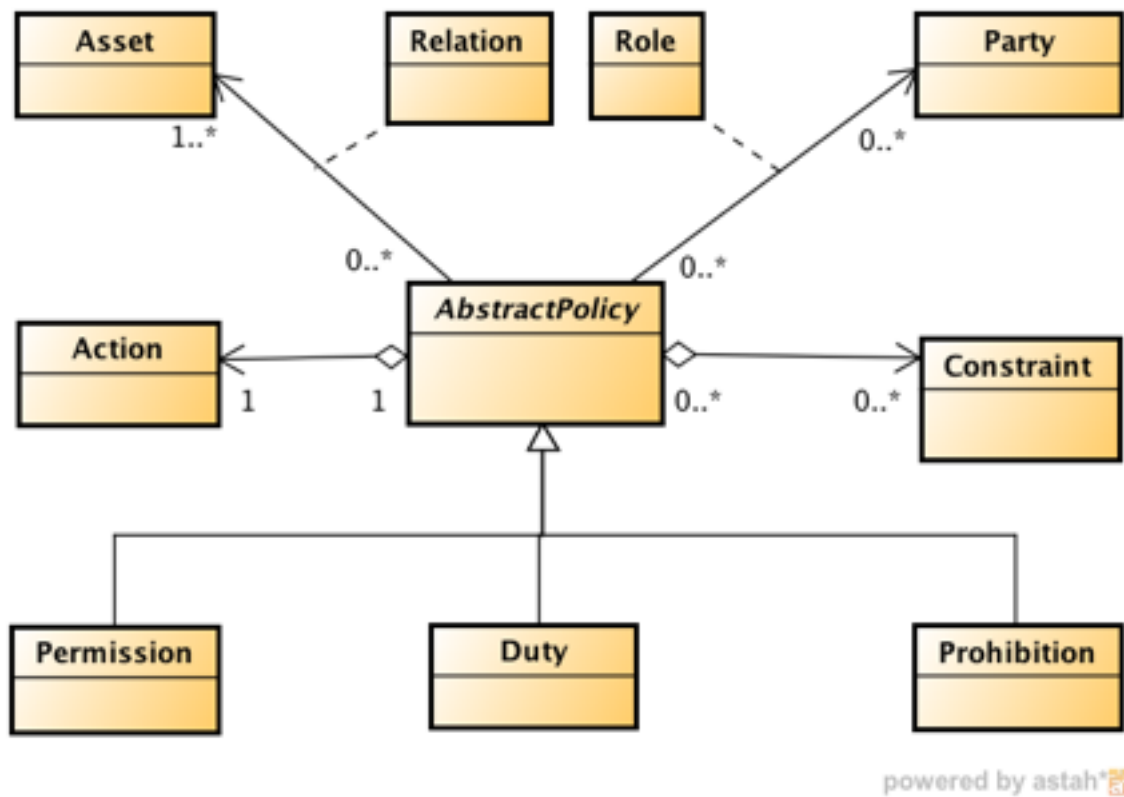
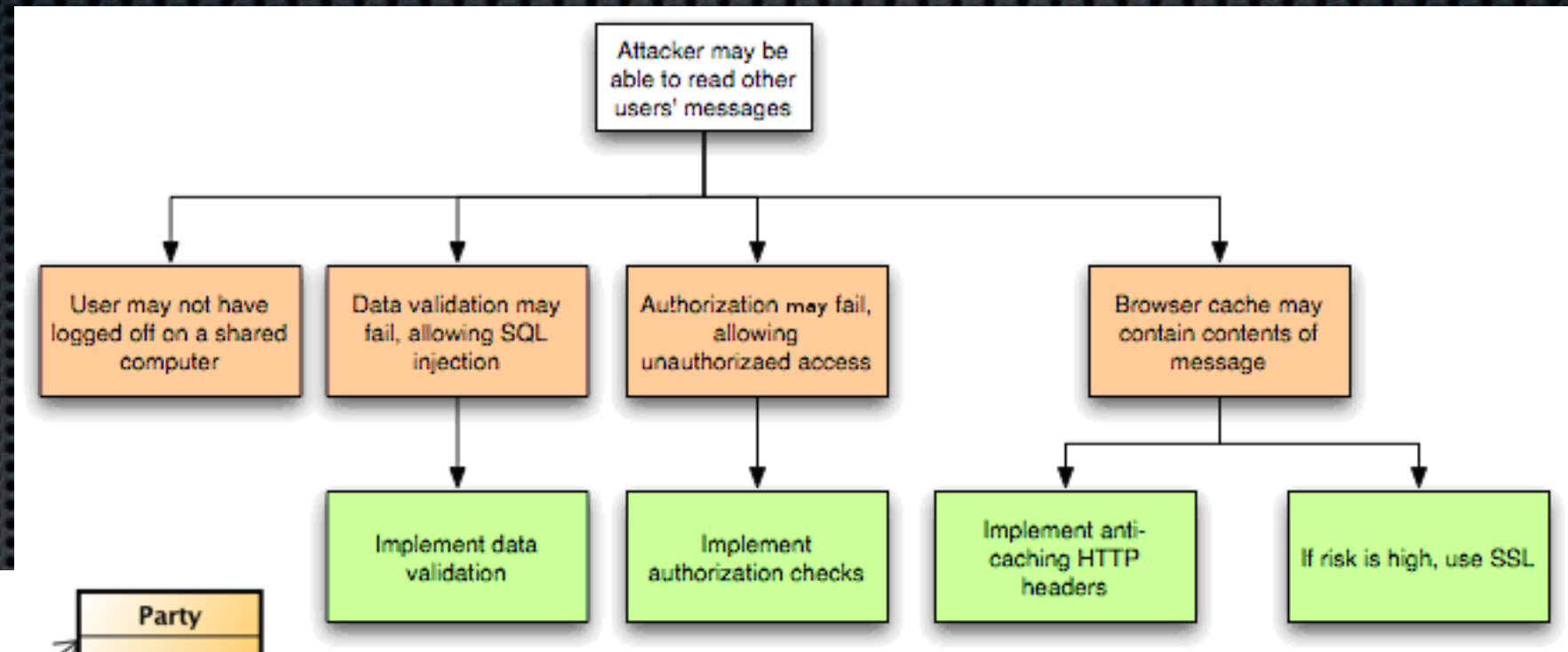


Security - key practices

- ✦ Model **threats** to identify mitigation
- ✦ Define **policy** to know what to protect
- ✦ Apply **mechanisms** to **mitigate** threats
- ✦ **Test** security as well as functions

Security - techniques

Threat Model



Security Model

Summary

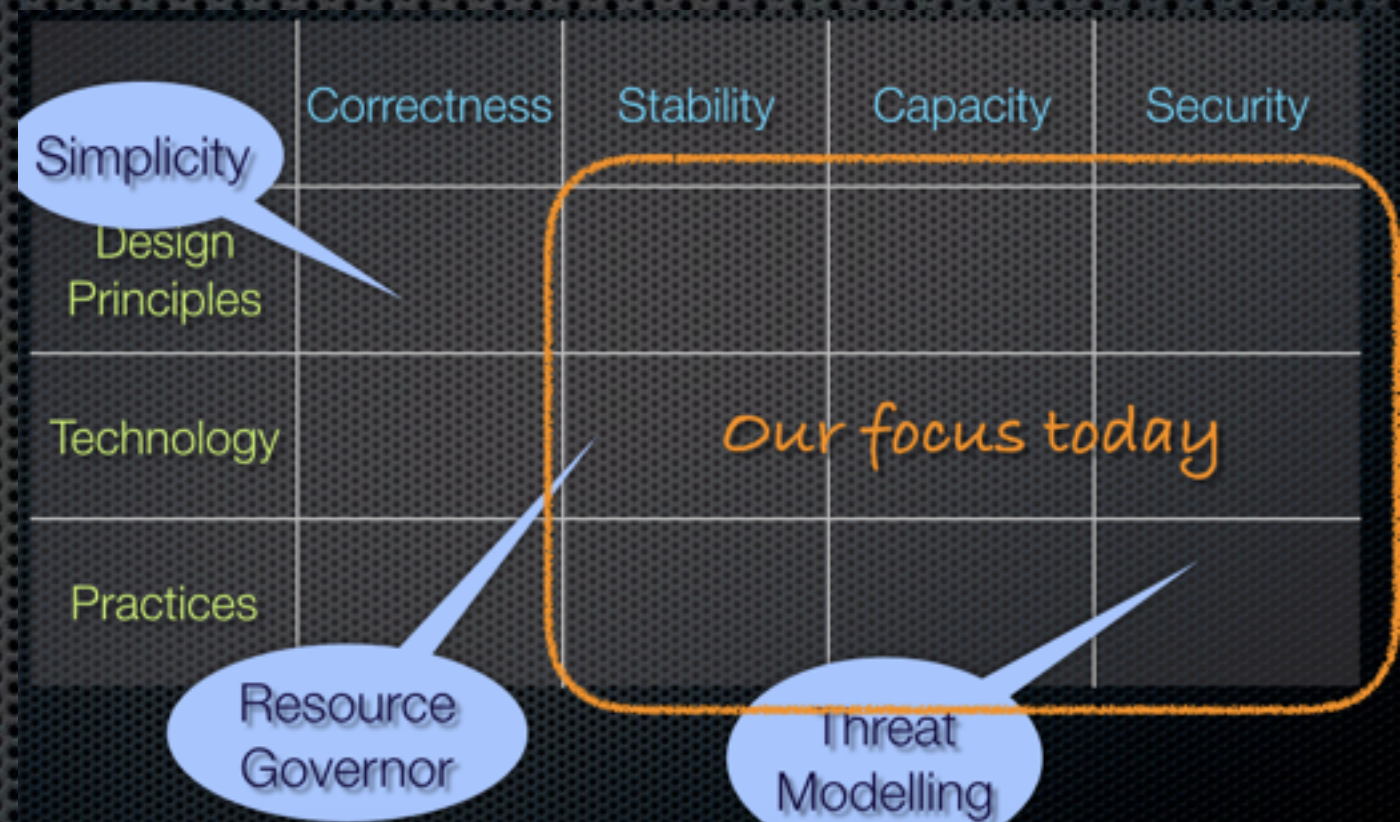


Summary

- ✧ Production is just different
 - ✧ it's not yours and you need to respect that
- ✧ Production is demanding
 - ✧ Correctness
 - ✧ Stability
 - ✧ Capacity
 - ✧ Security

Summary (ii)

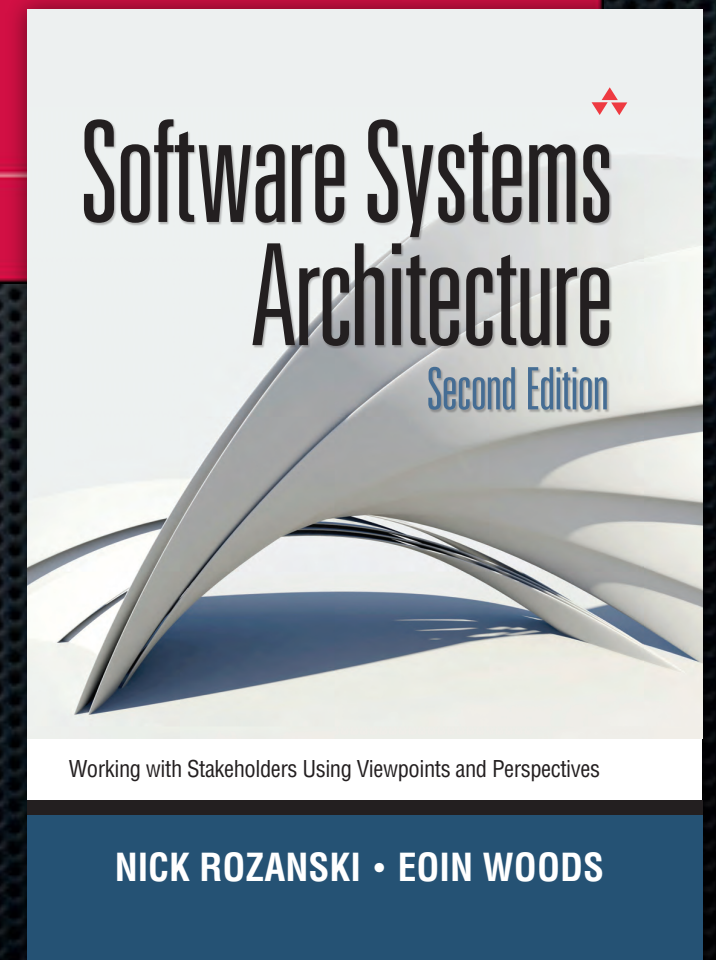
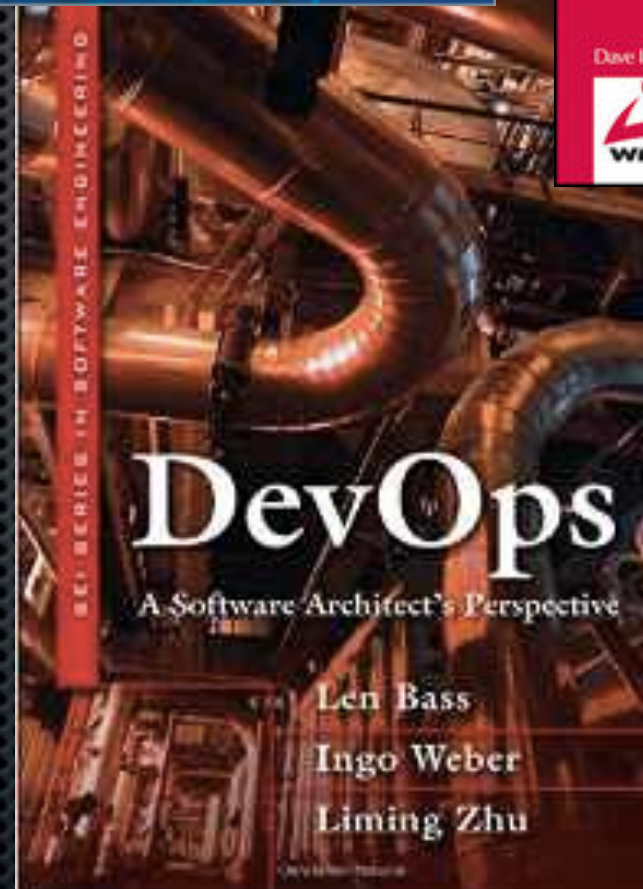
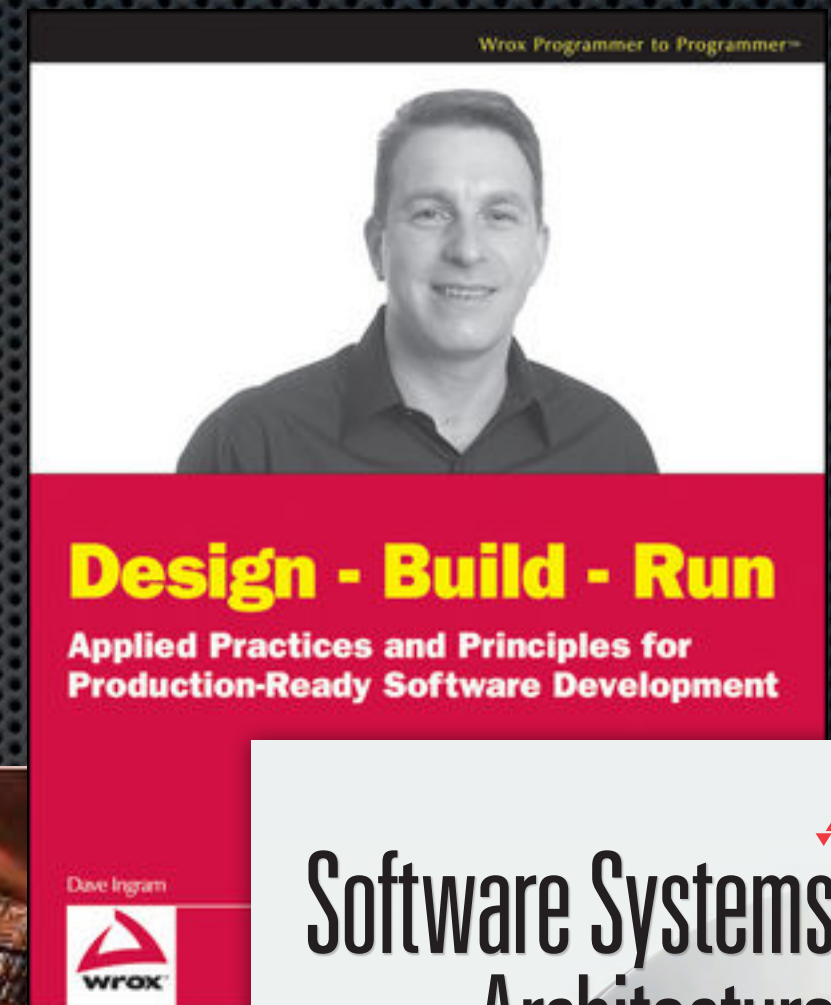
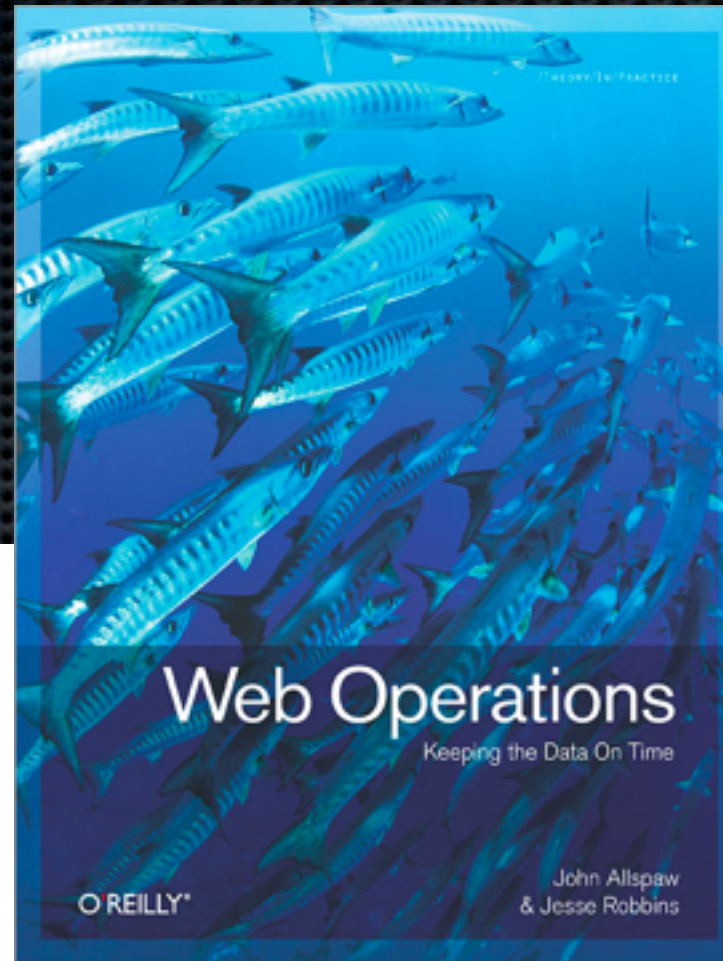
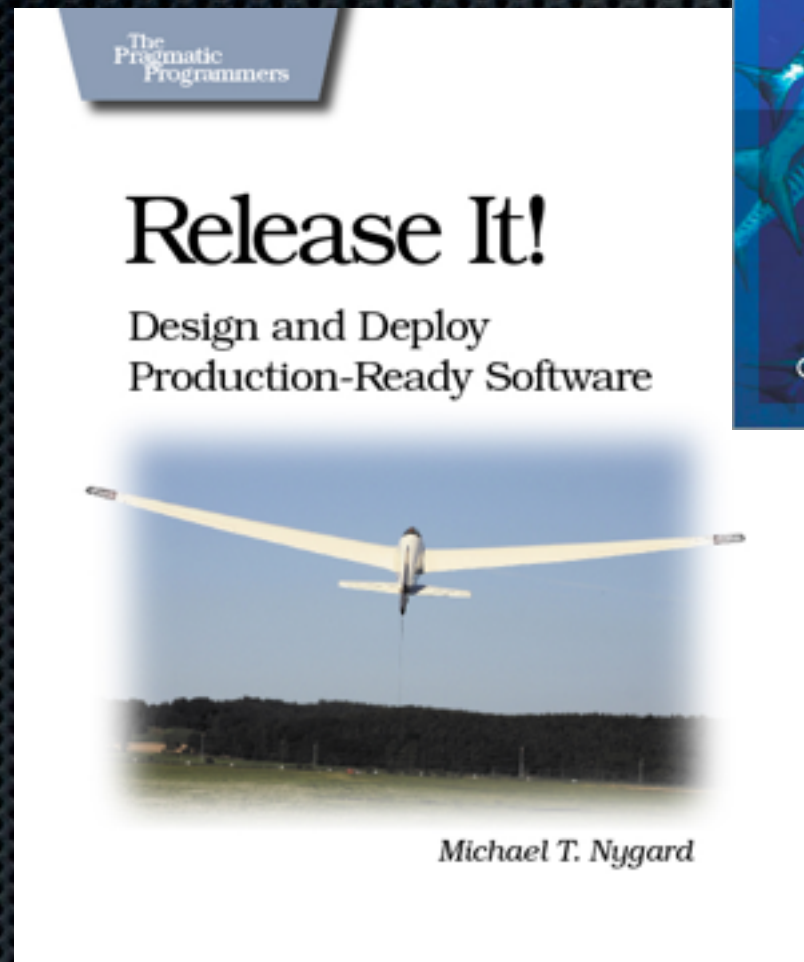
- ✦ Identify solutions by requirement & area
 - ✦ principles
 - ✦ technologies
 - ✦ practices



Summary (iii)

- ✦ Production requirements and principles go back to the age of the mainframe
- ✦ CD and DevOps makes another step
 - ✦ welcome attention from developers
 - ✦ new tech enabling new possibilities
 - ✦ breaking down silos to make it happen

Books



Thank you.

Questions?

Acknowledgements

<http://www.icons-land.com>

<http://www.alamy.com/>

<http://www.42u.com>

Eoin Woods

`eoin.woods@endava.com`

`www.eoinwoods.info`

`@eoinwoodz`