luminis
Conversing worlds

Microservices for Mortals

@BertErtman

About me

- Fellow at Luminis (Netherlands)
- Background in all things Java since 1995
- Java Champion, JavaOne Rockstar Speaker, and a Duke's Choice Award Winner
- Involved in architecting and implementing dozens of large scale systems over the past 20 years or so
- Book author for O'Reilly, speaker at many conferences

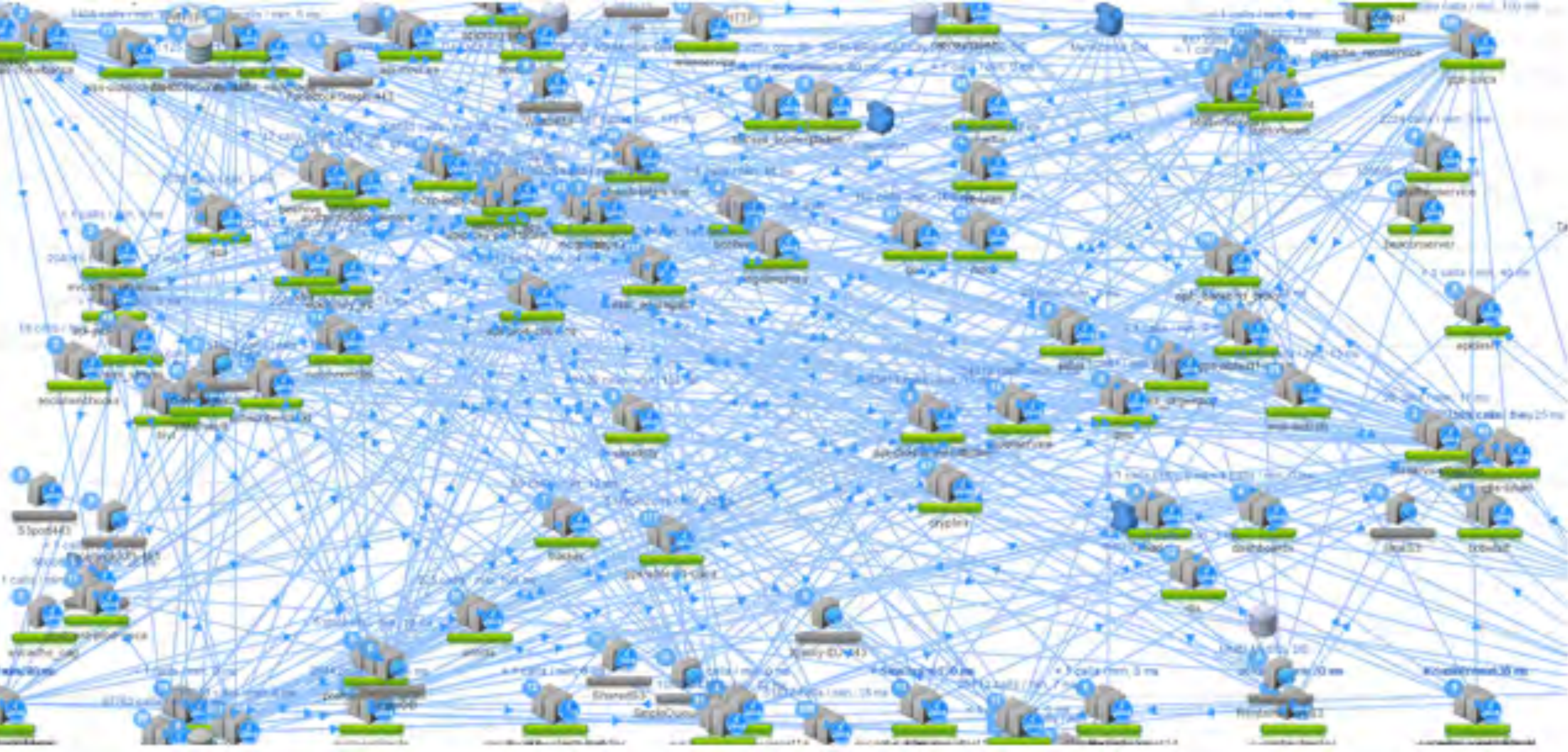
Italian Food Analogy

Before SOA: Spaghetti

SOA: Lasagna

Microservices: ???

Pick a side…



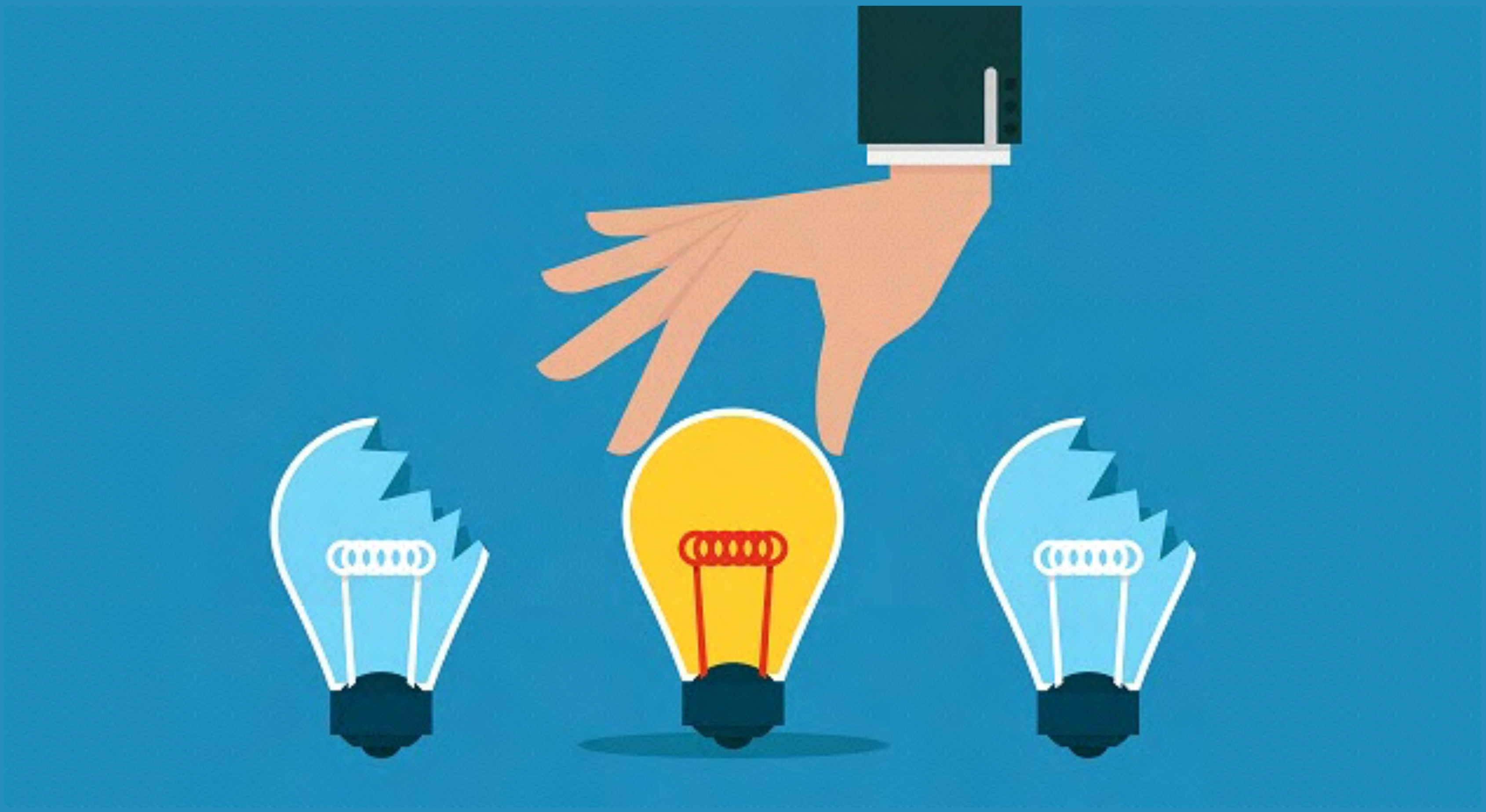
What if I’m allergic to Italian food? Can I still do Microservices?

Where did it come from?

- People have been doing AJAX, NoSQL, SOA, etc before they even got a name
- Microservices style architectures are a response to adjust software architecture to an ever-evolving spectrum. It addresses Business Agility through technology:
    - Usage of cloud-based infrastructure and services
    - DevOps
    - The need to scale up the number of people/teams
    - Client-side revolution both in technologies and devices

Microservices are about Business Agility

Microservices are like SOA, but only the good parts



What happened to SOA?

- SOA quickly turned into the rape victim of a vendor infested lock-in massacre, excessively complicating all good advice into giant overpriced hairballs sold as fake middleware, ESBs, and appliances.

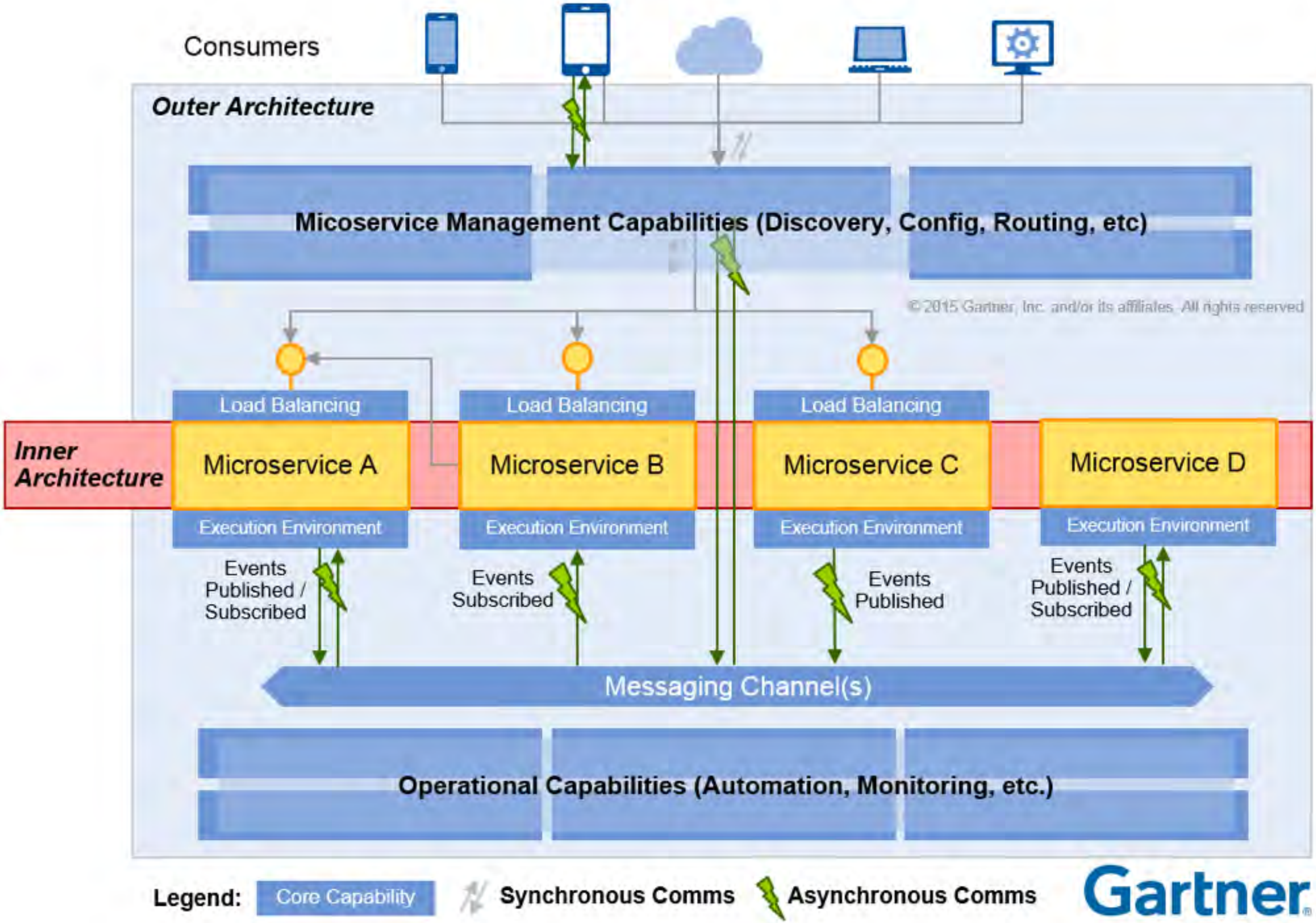
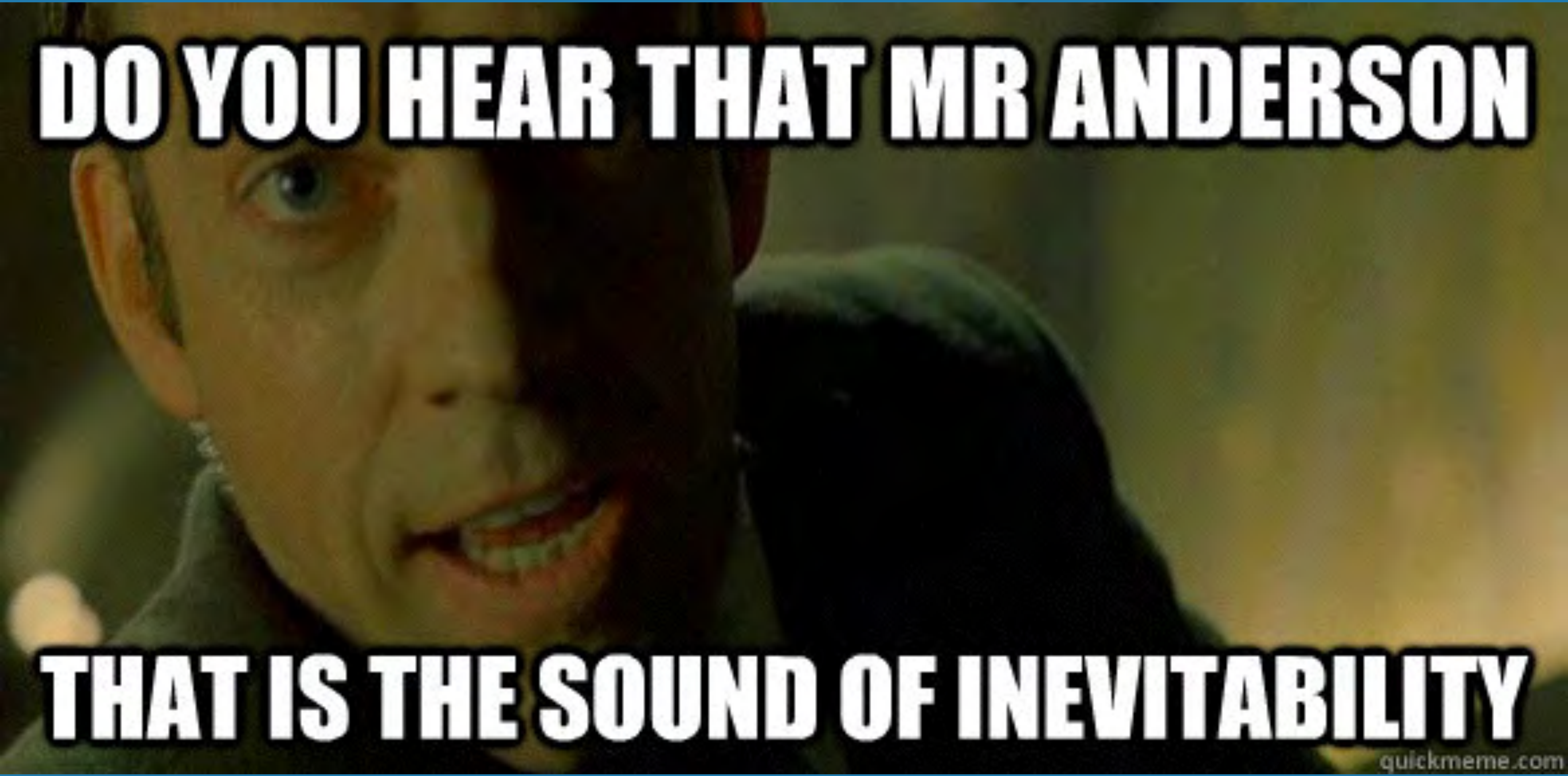


Image credit: http://blogs.gartner.com/gary-olliffe/files/2015/01/InnerOuterMSA.png

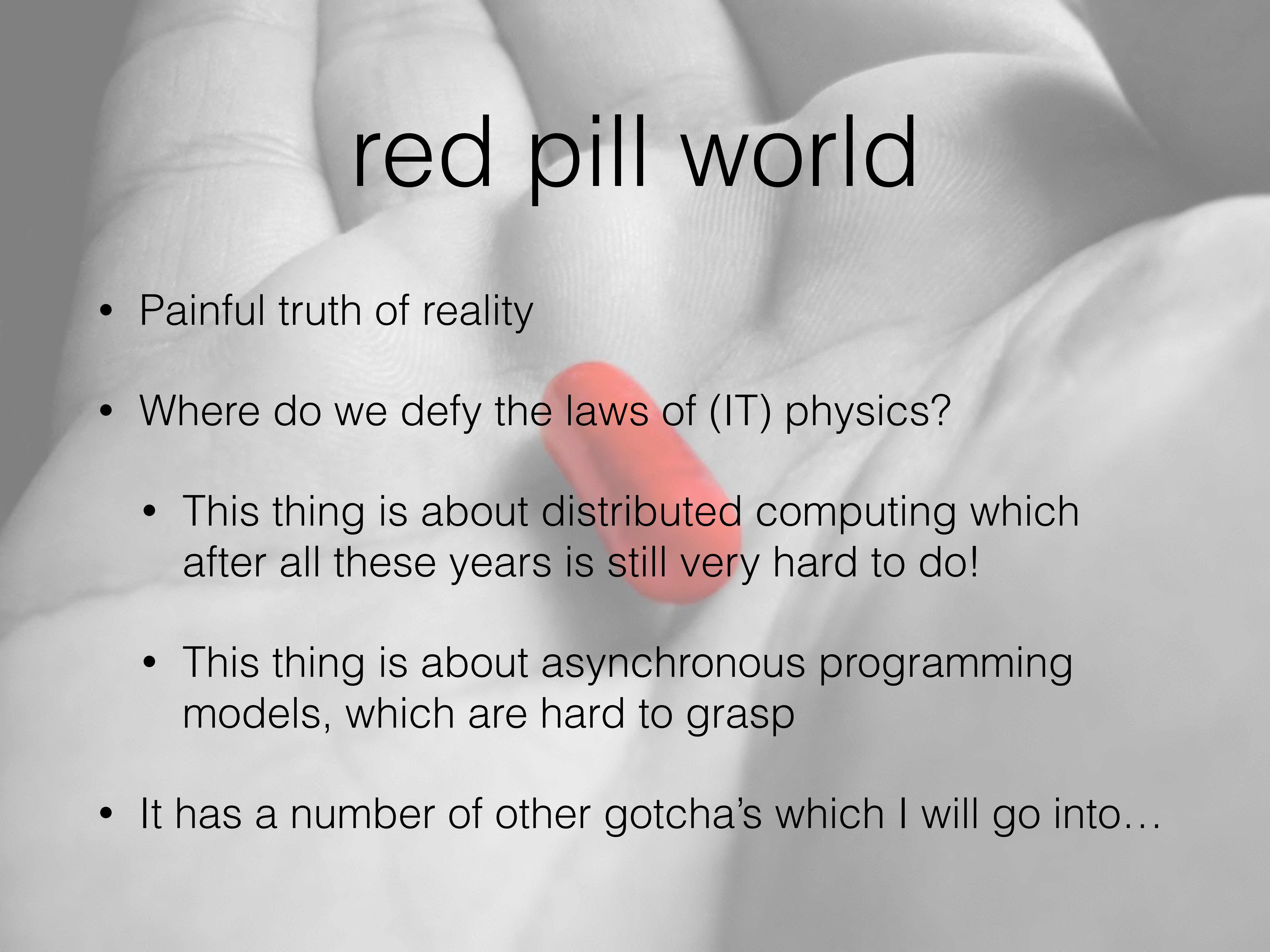

DO YOU HEAR THAT MR ANDERSON
THAT IS THE SOUND OF INEVITABILITY
quickmeme.com

Which pill do you take?

blue pill world

- Blissful ignorance
- Hey, since everyone and their mother seems to be tweeting about this microservices thing, it must be cool, right?
- Heck, even Martin Fowler is blogging about it. Now it’s definitely mainstream
- Dude, it’s on the freakin’ Technology Radar!
- Btw, this seems to be a killer way to introduce Node.js to my customer, yeah!

Which pill do you take?



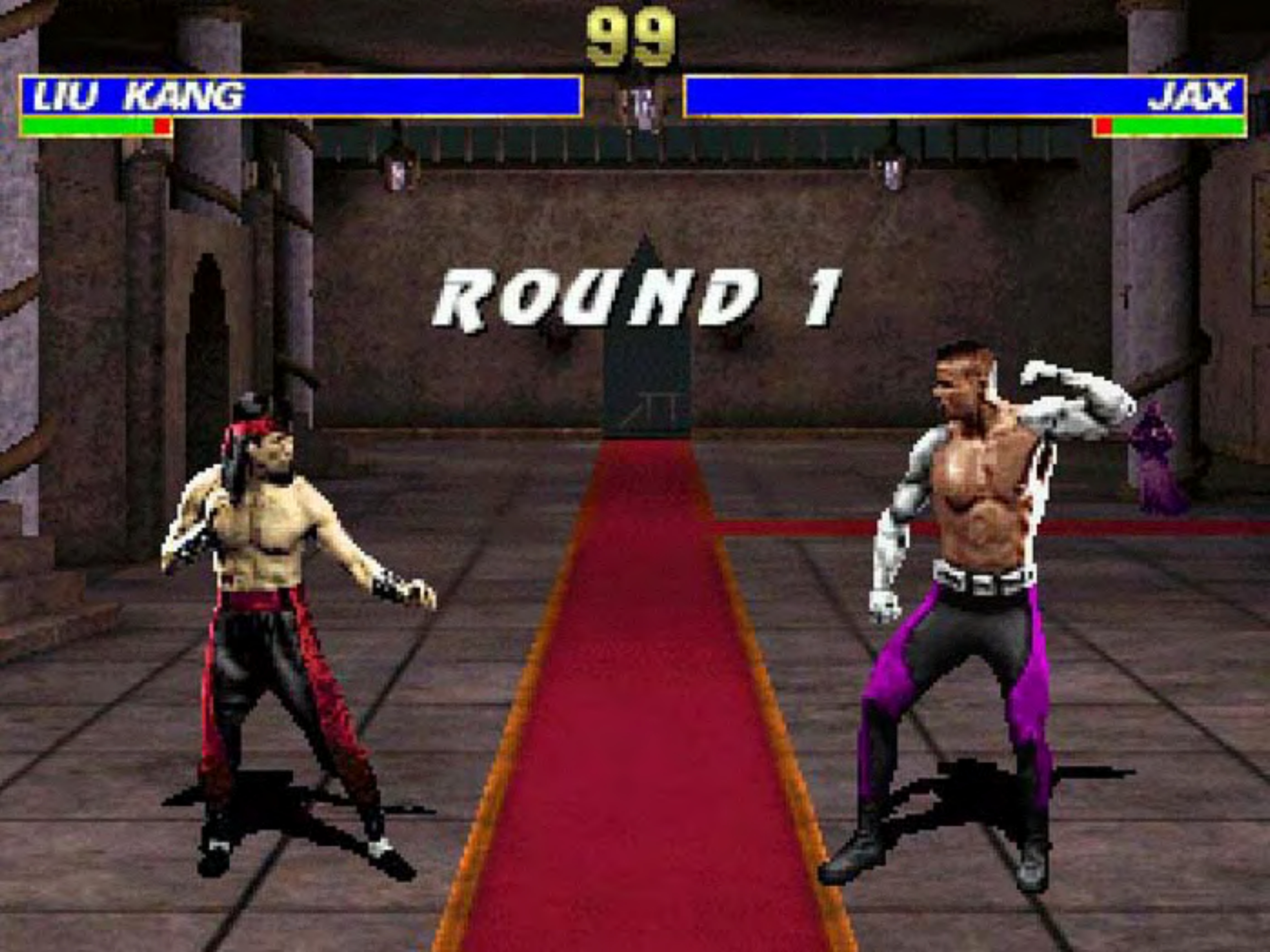
red pill world

- Painful truth of reality
- Where do we defy the laws of (IT) physics?
  - This thing is about distributed computing which after all these years is still very hard to do!
  - This thing is about asynchronous programming models, which are hard to grasp
- It has a number of other gotcha's which I will go into…

There is no spoon

- Forget synchronous programming models
- Forget a single Enterprise Domain Model
- Forget ACID transactions
- Forget Relational Integrity

Welcome to red pill world


99
LIU KANG
JAX
ROUND 1

SELECT YOUR FIGHTER
2
KOMBAT ZONE: MOONWALK TOWER

Picture credit: @ParisPic

About Monoliths

- The word ‘monolith’ has a negative connotation
- Not all non-microservices apps are bad applications
- What do you call an application that everyone wants to interface with but was not designed to do?

Monolith?

a successful application?

When monoliths are bad

- But… such systems can end up as monoliths with all the negative connotations to it as they:
  - Start to build up massive technical debt
  - Become hard to change without breaking stuff
  - QA and test cycles take lots of time (expensive)
  - When heavy internal coupling starts to take over control of the application
  - Become married to the underlying technical stack

Monolith to Microservices approach

- As was recently suggested by Martin Fowler, et al. the Monolith-First approach is a way of “strangling off services from a monolithic application”.

- This makes great theory, but is pretty hard to do in practice

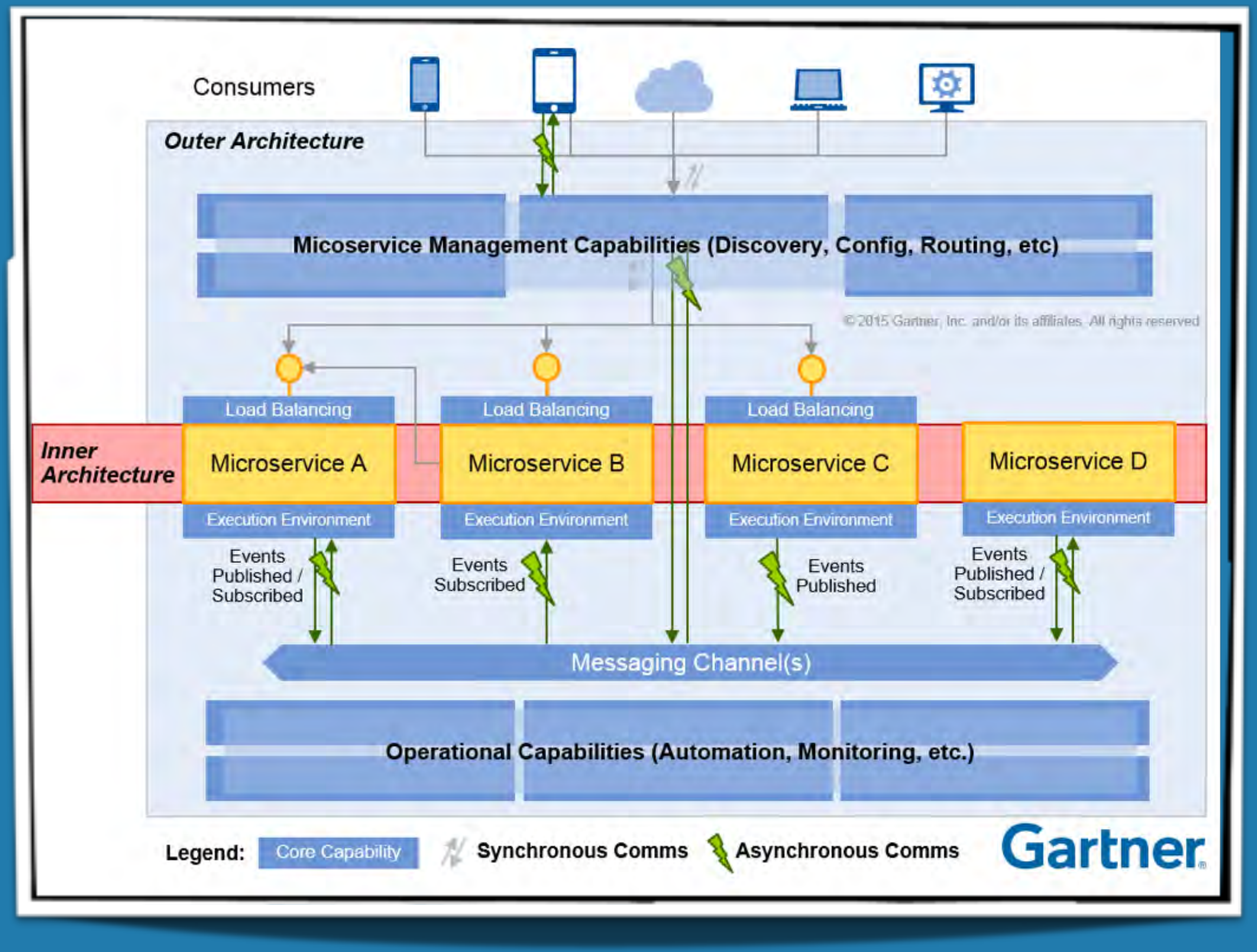
- Often times building blocks on the inside of an application are not suitable as building blocks outside of an application

Monolith to Microservices Struggles

- Initial Investment
- Data Strategy
- Synchronous vs Asynchronous
- Conway's Law
- Re-use Traps
- Dealing with Failure

Initial Investment

- So-called “Outer Architecture”
    - Service Discovery
    - Logging
    - Metrics and Analytics
- DevOps process in-place
    - Solid CI/CD practices
- Impact on testing strategy



Data Strategy

Data Strategy

- Don’t have multiple microservices share the same database model and perform updates on it
    - this results in unwanted coupling
- Separate at least read/write access if you must
- Better: separate data stores for each service

Data Strategy

- Q: What if I have common data?
- A: Either perform a call to another service or just copy the data
- Q: How do I deal with referential integrity?
- A: Move it up in the application layer

Synchronous vs Asynchronous

- Within the monolith, most communication will be synchronous
    - Your interfaces have been designed with synchronous, in-process, interactions in mind
    - May be chatty (fine-grained)
- Rethinking interaction patterns is essential
- Rethink the communication protocol as well

Service Communication

- Standardize on a common communication protocol
- Oftentimes people choose REST, but there are others
  - protobuf, thrift, zeromq, mqtt, …
  - Is REST fast enough to do massive fan-out?
- Maybe have two: synchronous and asynchronous

Avoid re-use traps

- Q: What is the best strategy for reusing common functionality between microservices?

- A: Copy it in the beginning of the project if you must. Never look back. Microservices are designed to be TOTALLY independent of each other, remember?

Conventions over abstractions!

Conway’s Law

So here is the obligatory reference to Conway’s Law:

> “Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"
>
> —M. Conway 1968

What it actually means

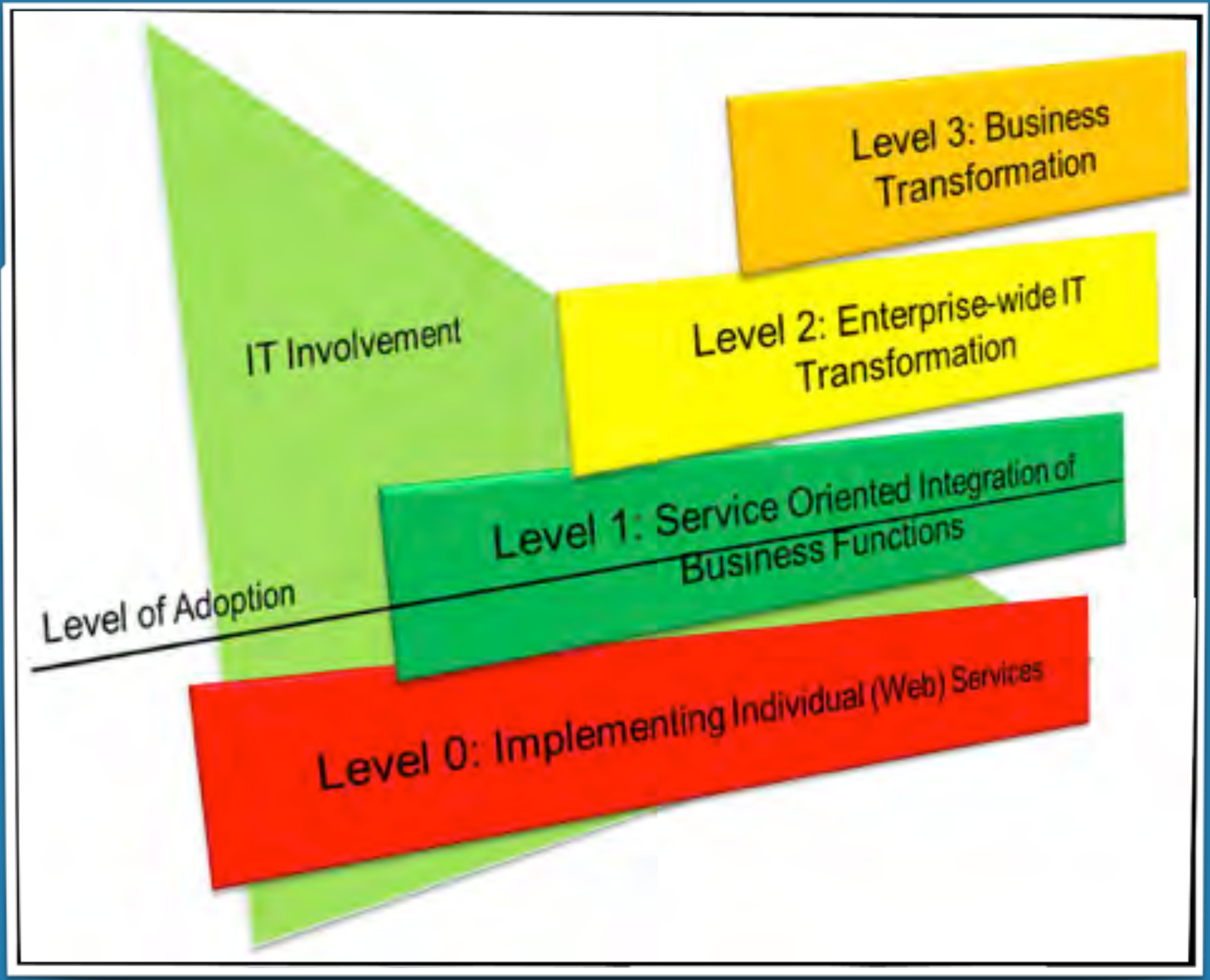
- Make sure the organization is compatible with the software architecture

- If your (microservices) architecture does not reflect the way your organization is structured, don’t even bother going that way!

- It also means that your team should be cross-functional. Everyone you need to build, maintain and get it into production must be part of the team

This is hard!

SOA Adoption Model


Level 3: Business Transformation
Level 2: Enterprise-wide IT Transformation
IT Involvement
Level 1: Service Oriented Integration of Business Functions
Level of Adoption
Level 0: Implementing Individual (Web) Services

02 WINS
99
PUSH START
REPTILE
RAYDEN
ROUND 2
SQUAKENET.COM

Failure will ALWAYS happen

Design for Failure

- Dependent services may be unavailable or too slow to respond
  - Minimize human intervention
    - failure happens all the time, so it shouldn’t be a big deal
    - fail faster sooner than later (prevent cascading)
  - Horizontal clustering to the rescue (multiple services)
  - Resilience Patterns to the rescue
    - CircuitBreaker, Bulkhead, Caching, Retry, Messaging, etc
    - This is complicated stuff. It is not just about throwing Hystrix or some other library in

Some take-aways

- The essence of Microservices is about structuring systems differently
    - It’s about Modularity
    - It’s about Separation of Concerns
    - It’s about Single Responsibility Principle
    - It addresses Business Agility through technology
- Those are not bad things!

However…

- Everything comes at a price
- Aligning the architecture to the organization is surprisingly hard
- It is not just a matter of throwing in a couple of frameworks, you have to think things through thoroughly before going this direction

In the end…

- Keep on educating yourself as more war stories see the light of day

- Don’t just listen to one vendor’s version of the story, all they care about is locking you in

- Have a rational thought process trump the hype, however difficult that is - think for yourself rather than following just the latest blogs and technologies

And one more thing…

- We are not all Netflix or Amazon
- Just like we’re not all Twitter and Facebook with the Big Data and Web Scale hypes…
    - not all of us have billions of calls floating around at any given day
    - if you pretend you are, you will have all their infrastructural problems to deal with for free and even at a minor scale they are just as hard

SUB-ZERO WINS
FLAWLESS VICTORY
LEGUMES

luminis
Conversing worlds

Thanks!

@BertErtman