Building Software Systems

Lecture 7.1

Microservices Style

SAURABH SRIVASTAVA

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT (ISM) DHANBAD

Architectural Styles

Solutions to commonly occurring problems

Can be *reused* multiple times – not specific to a particular usage

- Abstracted enough to be applied to a wider range of similar problems

Help achieve certain *Quality Attributes*

- As always, may be at the cost of other attributes (trade-off between QAs)

Service Oriented Architecture

An *Architectural Style* for building applications with distributed components

Components are modelled as stand-alone *Services*

Each service offer an interface to other services to access it

- For example, an API or an HTTP Endpoint

The services can be built using different technologies

- They must still provide a standardized interface though

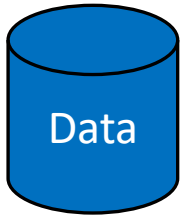
Services may communicate via different kinds of *Connectors*

- For example, REST calls, Message Queues or a specific protocol called SOAP

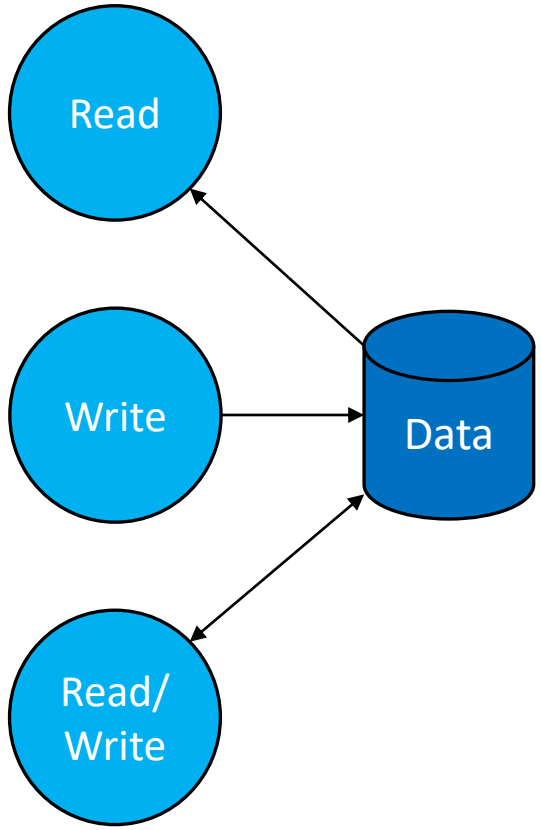
Microservices – An Architectural Style

How do we build Systems ??

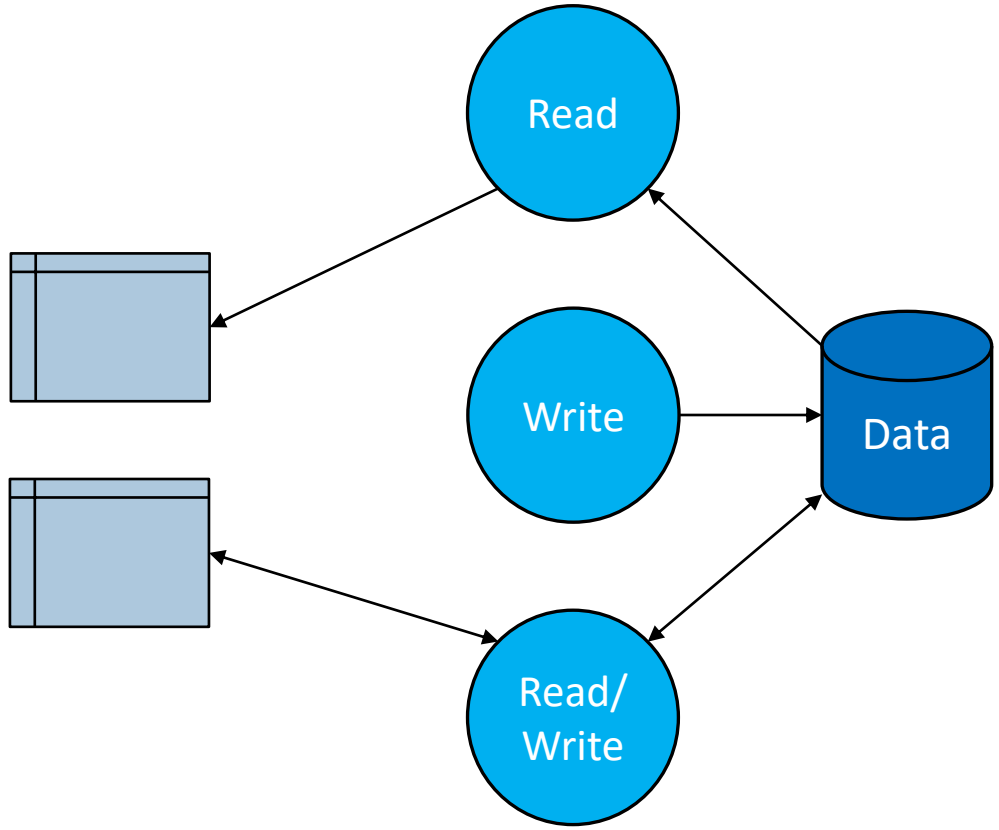
We Have Data

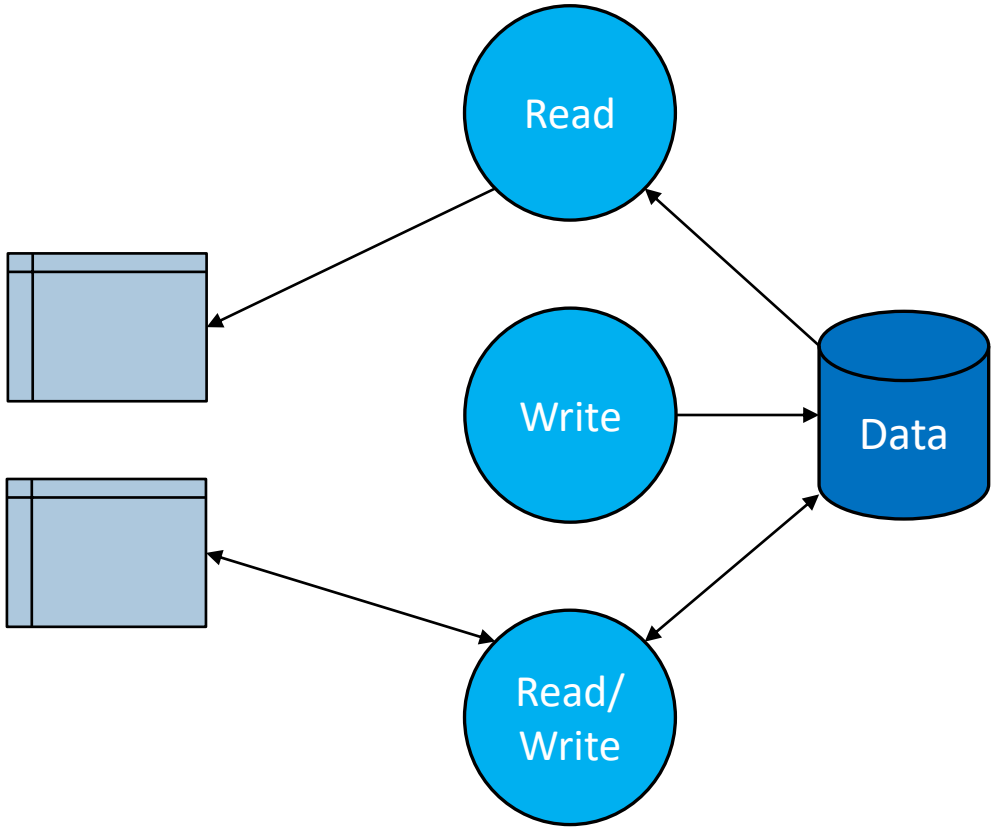


We write code
to access that data



We then provide interfaces to view or edit this data

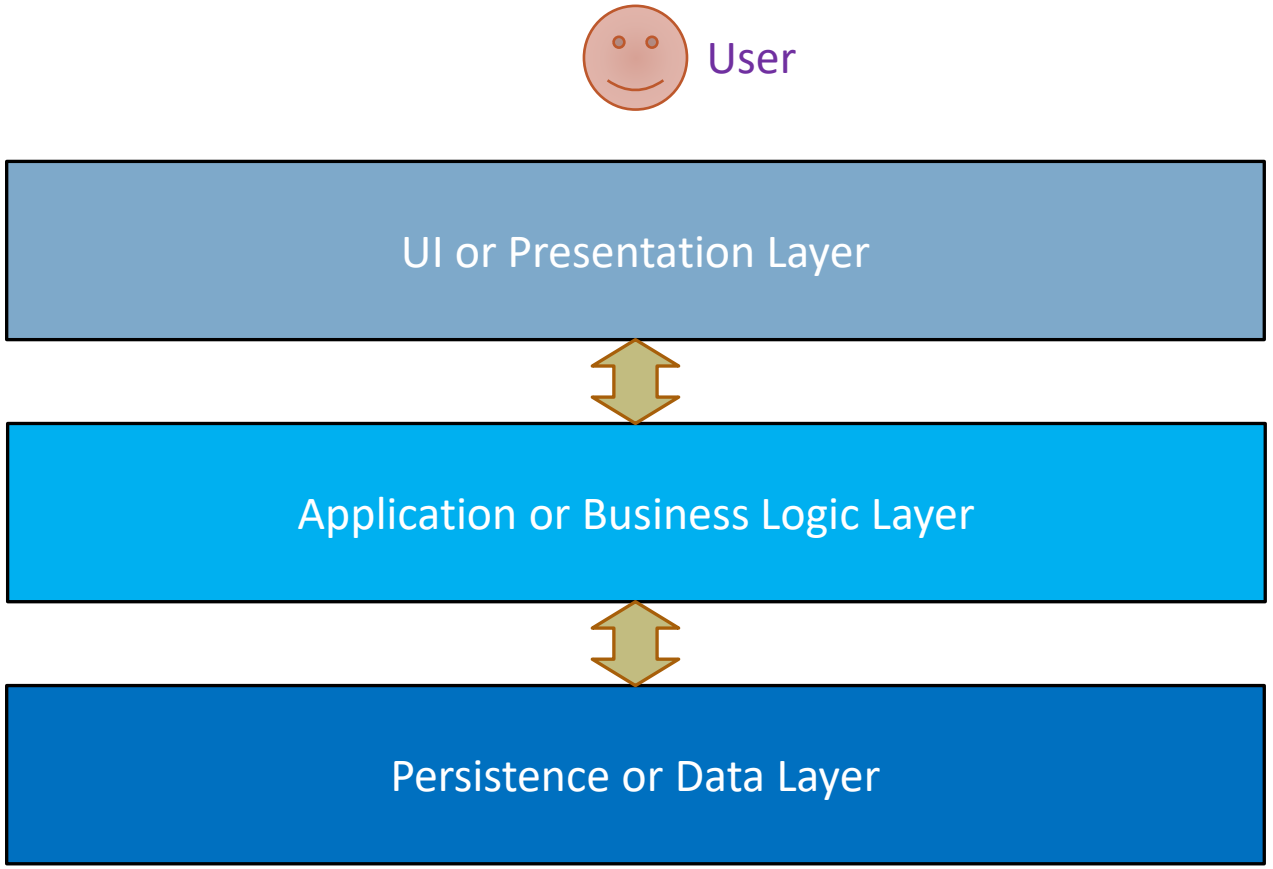




This is a common way to build applications, since very long

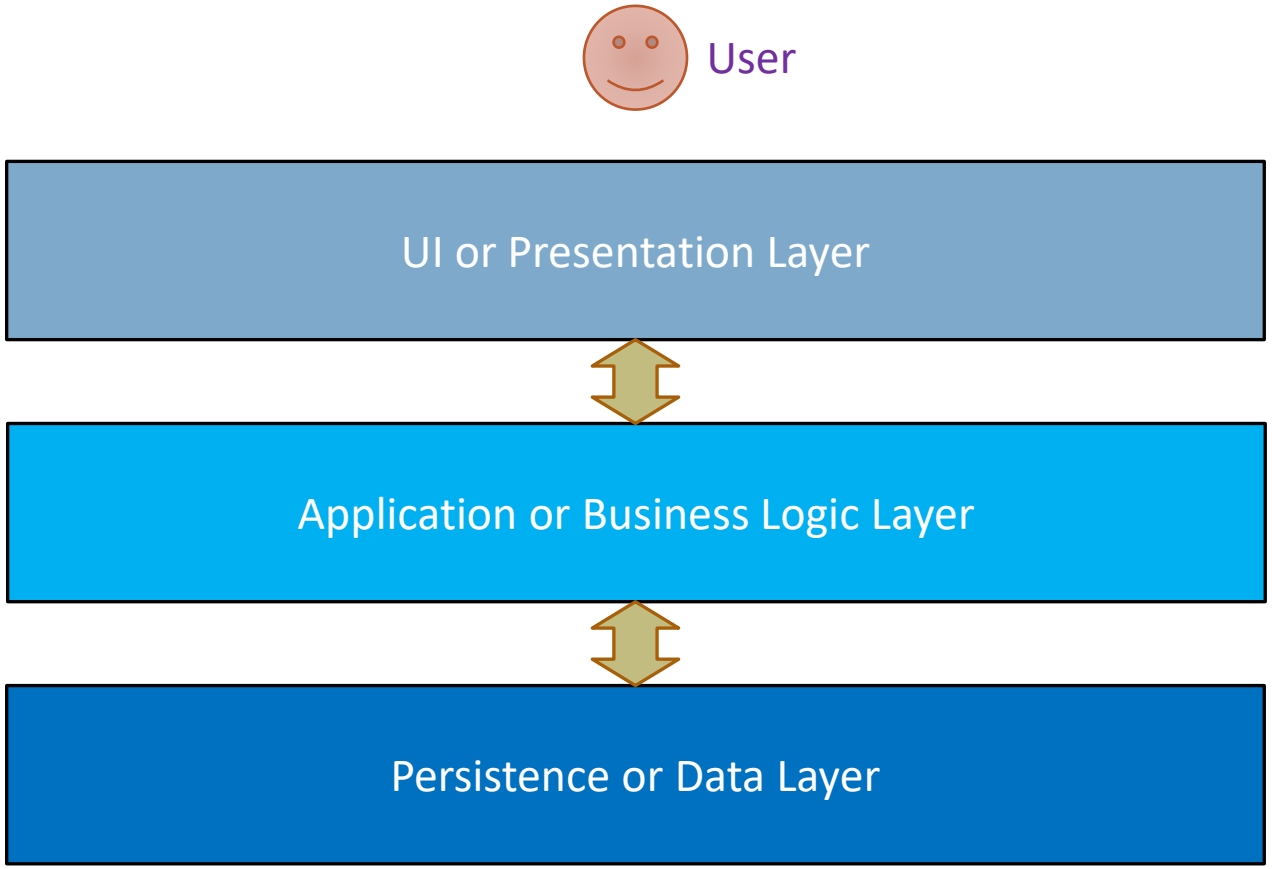
The methodology is based on the *3-Tier Architecture*

The methodology is based on the *3-Tier Architecture*

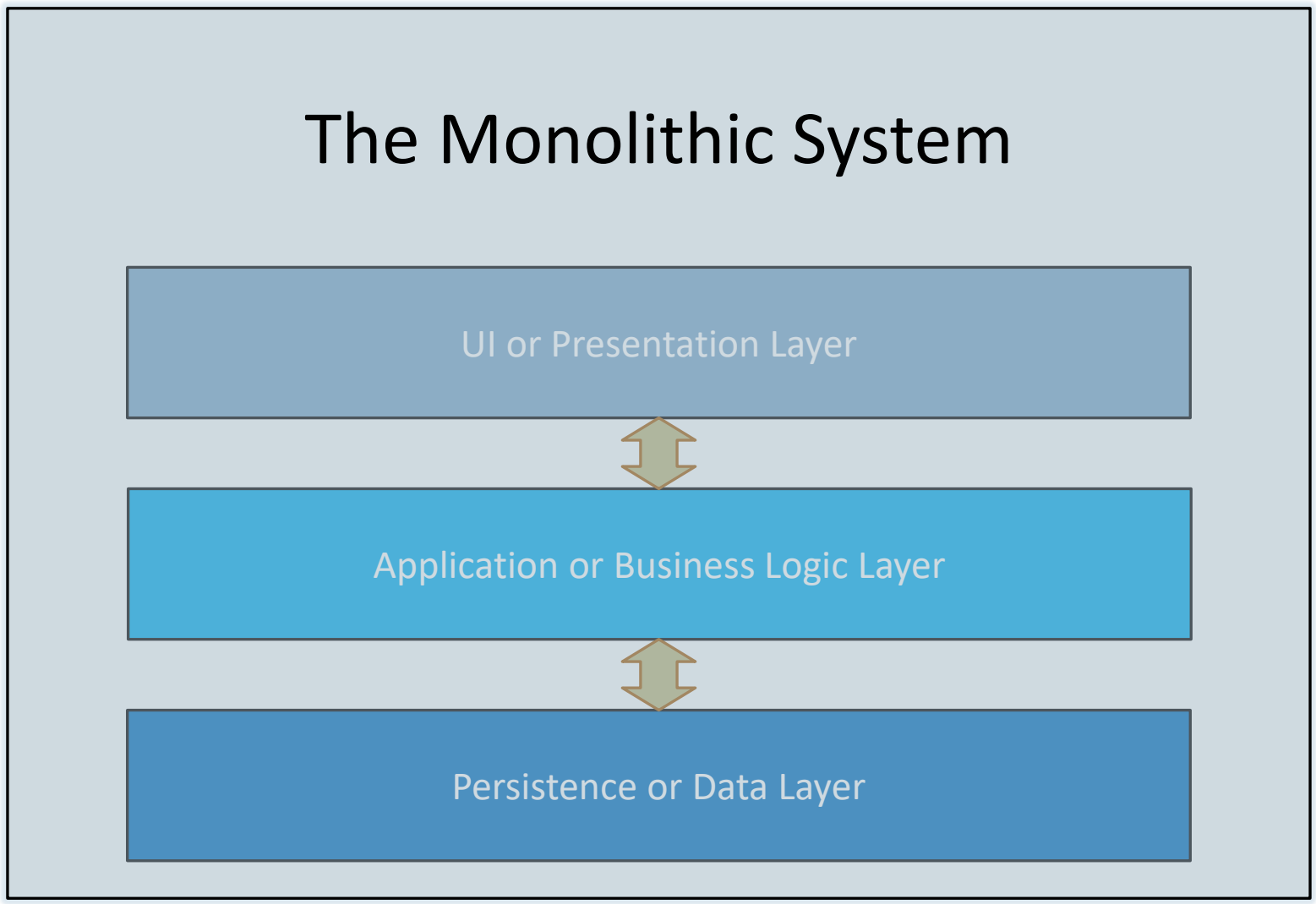


The methodology is based on the *3-Tier Architecture*

Coined around 3 decades ago !!



The Monolithic System



Do we need a change in how we build systems?

“If it ain't broke, don't fix it.”

Bret Lance ‘1977

Do the Enterprises care to change if everything is well?

Not really !!

So, is everything well?

Not really !!

So, what’s wrong with current architectures?

Time to Market !!

A typical enterprise software sees 2-6 releases a year

- That is about 2 months to release one version, at best !

New Features must be thoroughly tested before they can go live

- Unit Tests, Integration Tests, Regression Tests, User Acceptance Tests
- The testing itself may take a month or more to complete

Contrast that to applications on the web

- Facebook makes changes to the live code on a daily basis [1]
- Even multiple releases a day, is fairly common

Enterprises would love to decrease the time it takes to release new versions

- The current architecture may involve rebuilding a lot of things – may be everything – for a small change
- Even if it doesn't the coupling between the components may mandate regression testing

Operating on scale ??

Internet is an utter chaos

- So much heterogeneity – from protocols to data formats to unreliable networks
- Yet some internet applications can somehow make it work, that too, on a large scale

Enterprise Solutions may need to go online (or may be already online)

- What if our systems are to be accessed by customers *over the internet* ?
- Can our *monoliths* handle the scale of internet?

Scaling vertically is an option, or is it?

- We can add more hardware to our servers, and they can handle more load (vertical scaling)
- But what if the application is *inherently incapable* to use new cores (no parallelism) ?

We can run multiple instances of the application

- Some parts of the application may be rarely used, but had to be replicated any way

Can’t we do better?

How about if we *split the monolith* ?

Instead of one system, have a *system of systems*

Each smaller system, is a system on its own

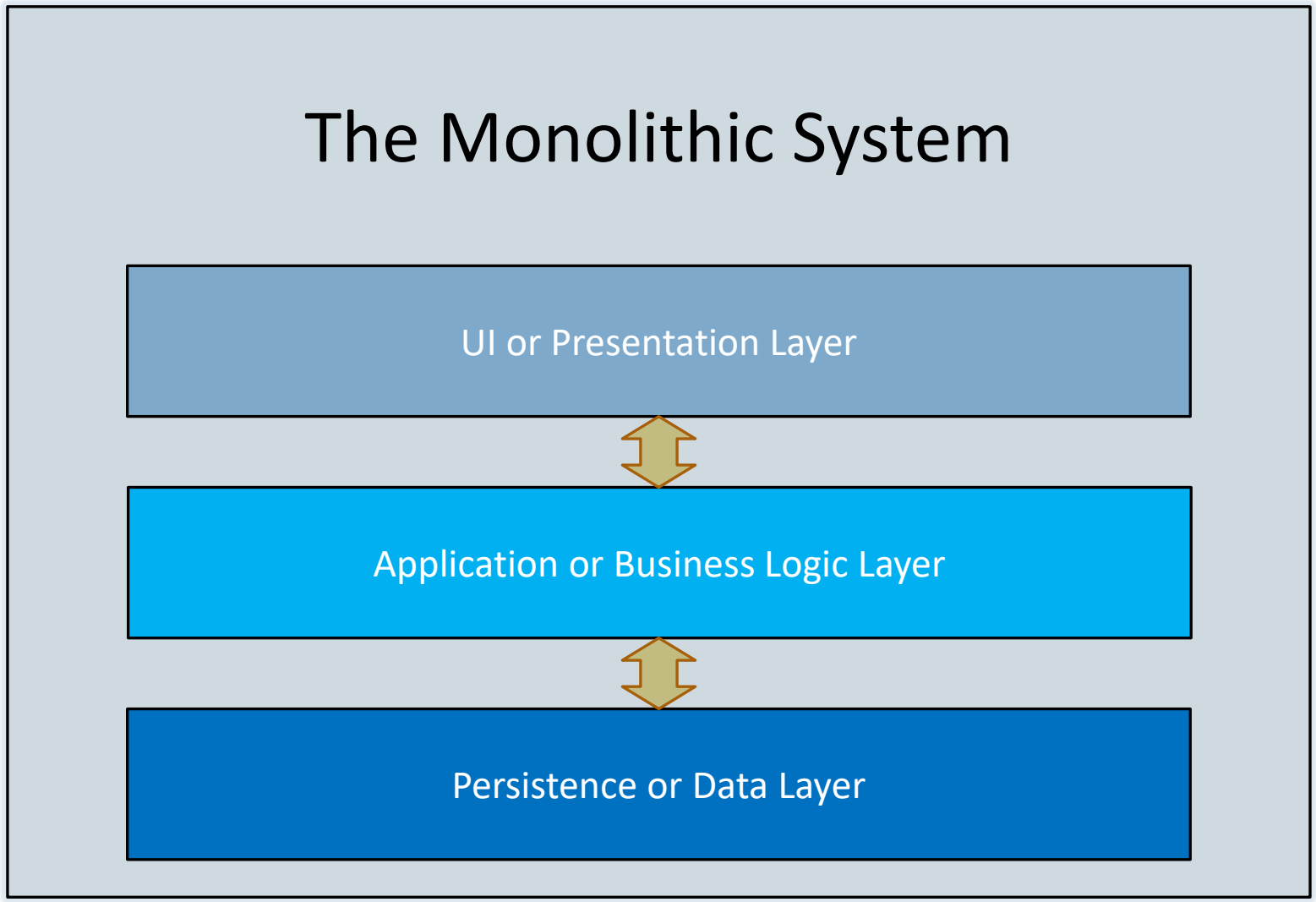
It keeps its own data

Probably has its own UI too

Exposes an interface for others to use

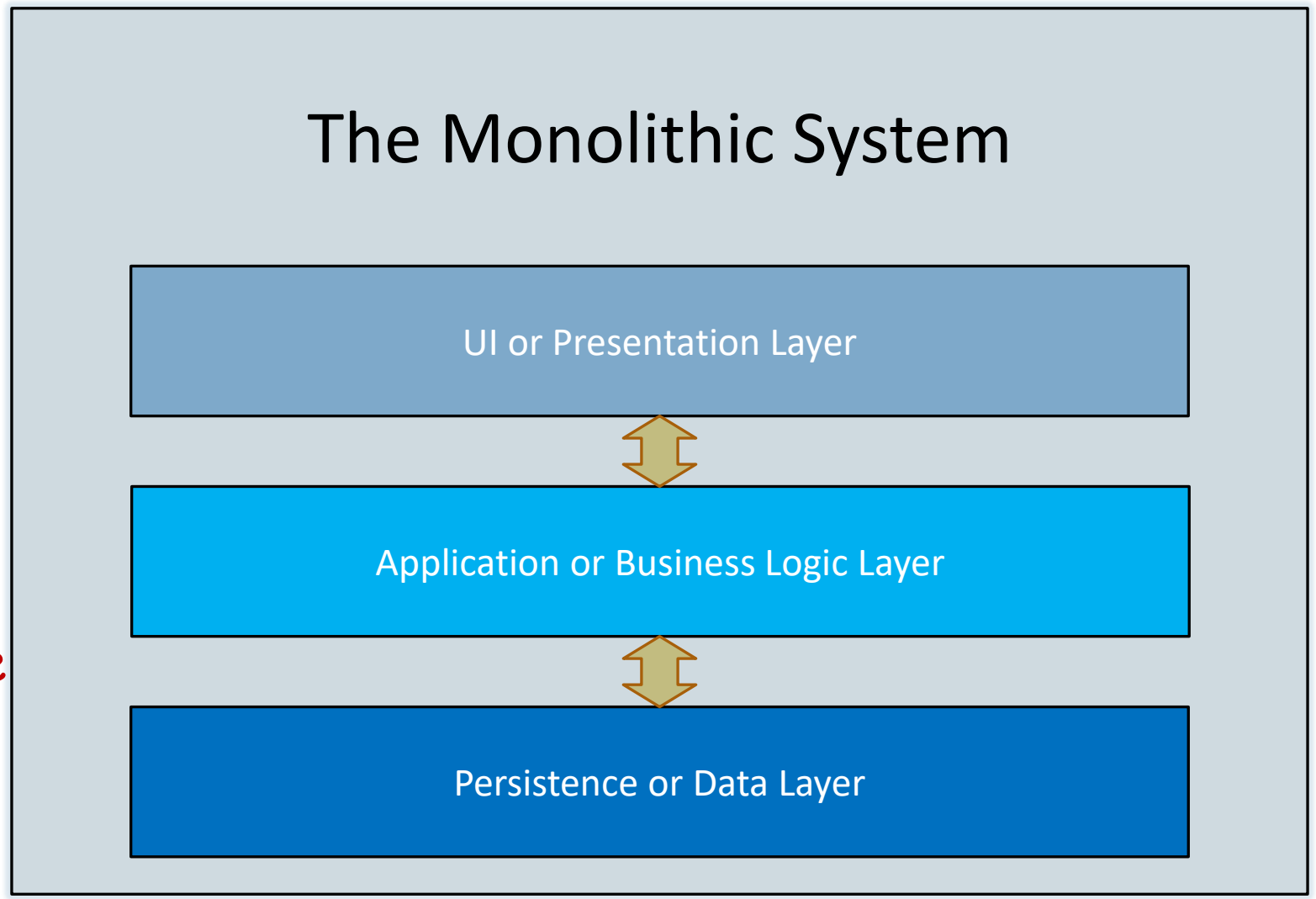
The Monolithic System

The Monolithic System



Right now
we are cutting it
horizontally

We see it from a technical perspective



Right now we are cutting it *horizontally*

“Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations.”

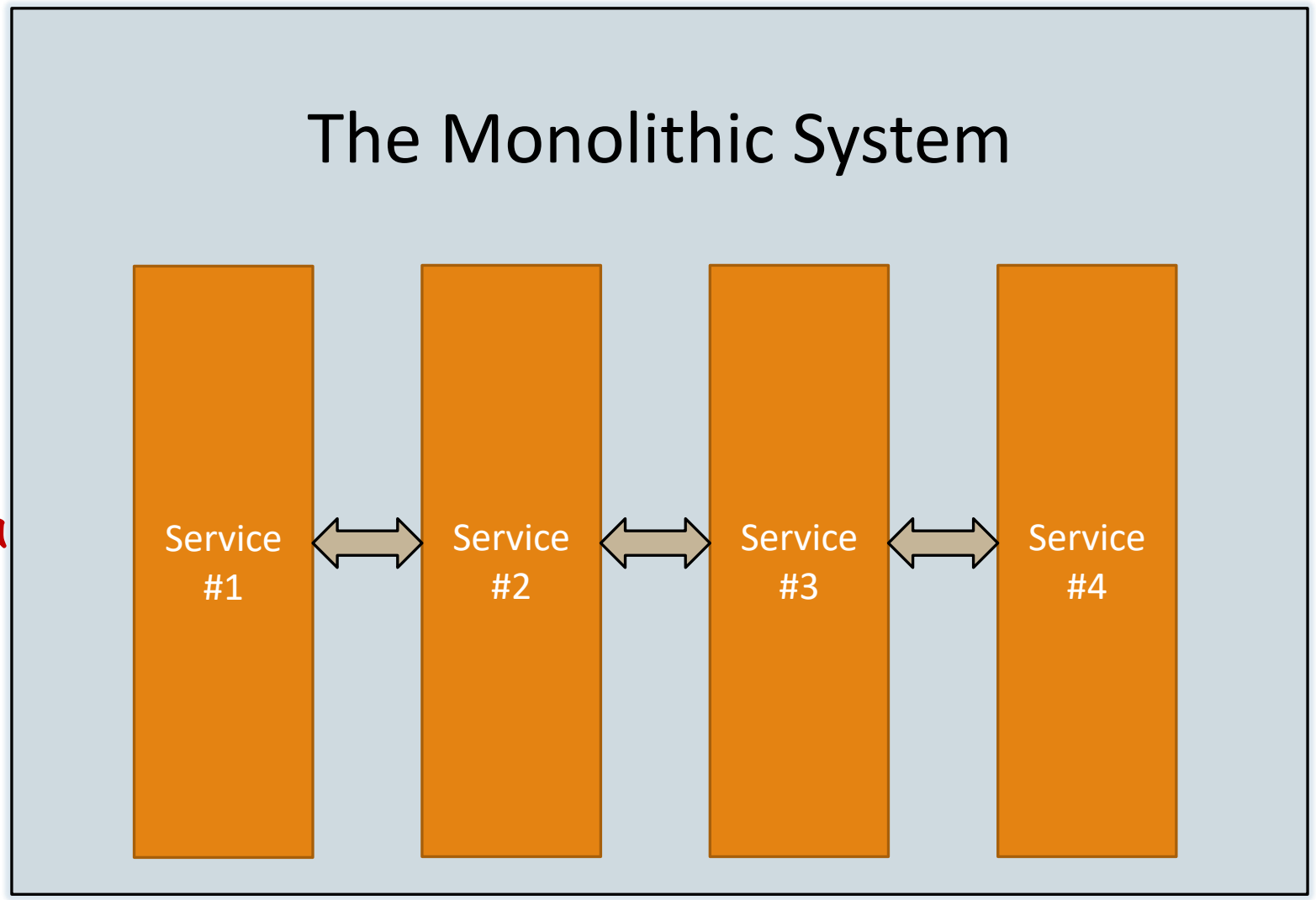
Conway’s Law ‘1968

The Monolithic System

The Monolithic System

How about if we cut it *Vertically* ?

Services represent a complete, coherent task



How about if we cut it *Vertically*?

What are “Microservices” ??

Microservices are

Small, Autonomous services, that work together

- The *smaller systems* we talked about

Microservices are

Small, Autonomous services, that work together

- The *smaller systems* we talked about

How small is small?

- There's no clear definition
- *Lines of code* is not very apt to calculate
  - 100 lines of **C** code ≠ 100 lines of **Python** code
- Something that could be re-written in roughly two weeks time [2]
- Small enough, and no smaller [3]

Microservices are

Small, Autonomous services, that work together

- The *smaller systems* we talked about

Capable to exist autonomously

- Shall be independently deployable
- *Ideally* on a separate *host*
- All communications to outside world are via network calls
- Interface to communicate should *ideally* be technology agnostic

What do we gain by using Microservices ??

Technology Heterogeneity

Sometimes we choose technology that is the lowest common denominator

- Some things may be done better in some technologies
- A use-case may fit more to a NoSQL engine, than to an RDBMS engine

The learning curve for new technologies could be steep

- There may not be enough time for proper training of the staff
- It may be too risky to try them with little experience

The interfaces may restrain our choices

- What if a component we need to talk only knows RMI?
- We are forced to use Java elsewhere too !

Technology Heterogeneity

Since all the *smaller systems* are autonomous, they can be built using separate technologies

- As long as they conform to certain loose restrictions
- Like “being able to run on Linux”
- Lesser the restrictions, better it is

If we fail, we don’t lose much

- Since microservices are small in size, if we screw up, or realize that the technology change is adversarial, we still don’t lose much
- We can revert to old technology and rebuild the part from scratch

Low risk, more scope to experiment

- We can pick up technologies that we think are better, to build one or few services at a time
- If they work, we can do the same with others

Resilience

When something goes wrong in the monolith

- It can have a cascading effect
- One problem can bring down the whole system

Microservices are isolated from each other

- If one service is down, the others may still keep going
- Supports the idea of graceful degradation much better

Fault isolation and repair becomes easier

- The service that is down, needs to be looked into
- Smart logging can make it easier to find bugs, and repair them quickly

Ease of Deployment

In monolithic systems, changes are not easy

- Every change may involve a complete re-build, and then a re-deployment
- The application most probably will be offline for this time

So changes accumulate

- To avoid re-deployment, changes are accumulated till they become significant
- Higher the delta between releases, higher the risk

Since all microservices are deployed independently

- Changes to one can be deployed without affecting others (changes which are *not breaking* changes)
- If the new release doesn't work, a quick restore can be made to previous version

Time to market is reduced with microservices

- New changes can be rolled out easily, and with relatively lesser risk

Organizational Alignment

A small team is a happy team

- Smaller teams are generally more productive
- Reduces involvement of too many brains
- Better communication *within the team*

One microservice can be owned by one team

- No external involvement, teams can be responsible for the full lifecycle of the service, including operation and maintenance

Teams can be built around business contexts rather than technology

- One team can cater to one business subdomain, via one or more microservices
- Can also interact directly with the customers to get better feedback

Optimized for Replacability

Bigger systems are hard to modernize or replace

- Building a newer version from scratch may involve too much effort, and risk

Microservices are optimized for replacement

- Since they are small, it is easier to write them from the scratch and replace
- For example, a custom made single-sign on service can be replaced by sign-in via Facebook easily

What’s the catch then in using them ??

Not Magic Wands

So why don’t we all start using microservices?

- Just like any other Architectural Style, microservices have trade-offs
- Not everything associated with them make life easier

Not Magic Wands

So why don't we all start using microservices?

- Just like any other Architectural Style, microservices have trade-offs
- Not everything associated with them make life easier

Network Calls are slower

- Out-of-process call, like HTTP request is always slower than in-process call like method invocation

Networks can fail

- It may be difficult to figure out if a service is down, or is out-of-reach

Networks are insecure

- If the data travels over a network, say internet, it needs to be properly guarded against adversaries

Not Magic Wands

So why don't we all start using microservices?

- Just like any other Architectural Style, microservices have trade-offs
- Not everything associated with them make life easier

More *hosts* to manage

- Typically, each microservice runs on a different host
- A host could either be a physical machine, a virtual machine or even a docker container
- More microservices mean more hosts to manage

Not Magic Wands

So why don’t we all start using microservices?

- Just like any other Architectural Style, microservices have trade-offs
- Not everything associated with them make life easier

Distributed Transactions

- With one database or process, transactions are easier to implement
- A rollback in case of failure is easy to achieve
- With multiple processes and database instances, each keeping track of some part of an overall business transaction, it is difficult to revert changes in case of failures

Not Magic Wands

So why don’t we all start using microservices?

- Just like any other Architectural Style, microservices have trade-offs
- Not everything associated with them make life easier

Distributed Monitoring and Logging

- Logging and monitoring one host is significantly simpler than a cluster of hosts
- Without automation, things can quickly become unmanageable

Wrapping up !!

Summary

Microservices Architecture

- is an Architectural Style
- meant to build scalable, resilient, easily updateable applications
- using small teams
- by building small, autonomous microservices,
- which are loosely coupled with smarts in the end-points, not in the network

However,

- they bring with them issues associated with
- managing distributes resources,
- and sub-optimal testing processes,
- along with duplication of code and effort at times

Thanks !

References

[1] Release engineering and push karma, **Chuck Rossi**, https://www.facebook.com/note.php?note_id=10150660826788920

[2] Micro services, what even are they?, **Jon Eaves**, http://techblog.realestate.com.au/micro-services-what-even-are-they/

[3] Building Microservices – Designing Fine Grained Systems, **Sam Newman**, O'Reilly