CS 4530: Fundamentals of Software Engineering

Module 10.1: Distributing Processing

Adeel Bhutta, Jan Vitek and Mitch Wand

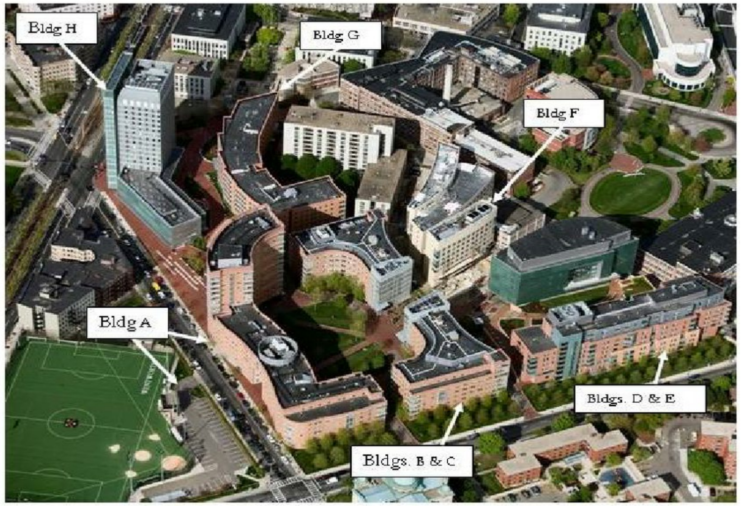
Khoury College of Computer Sciences

Learning Goals for this Lesson

- By the end of this lesson, you should be able to…
  - Recognize a few common software architectures
  - Discuss some of the tradeoffs of scalability, performance, and fault tolerance between these architectures

Distributed Software Architectures

- Goal: abstract details away into reusable components
- Enables exploration of design alternatives
- Allows for analysis of high-level design before implementation
- Match system requirements to quality attributes of common architectural patterns



Review: Challenges of Distributed Systems

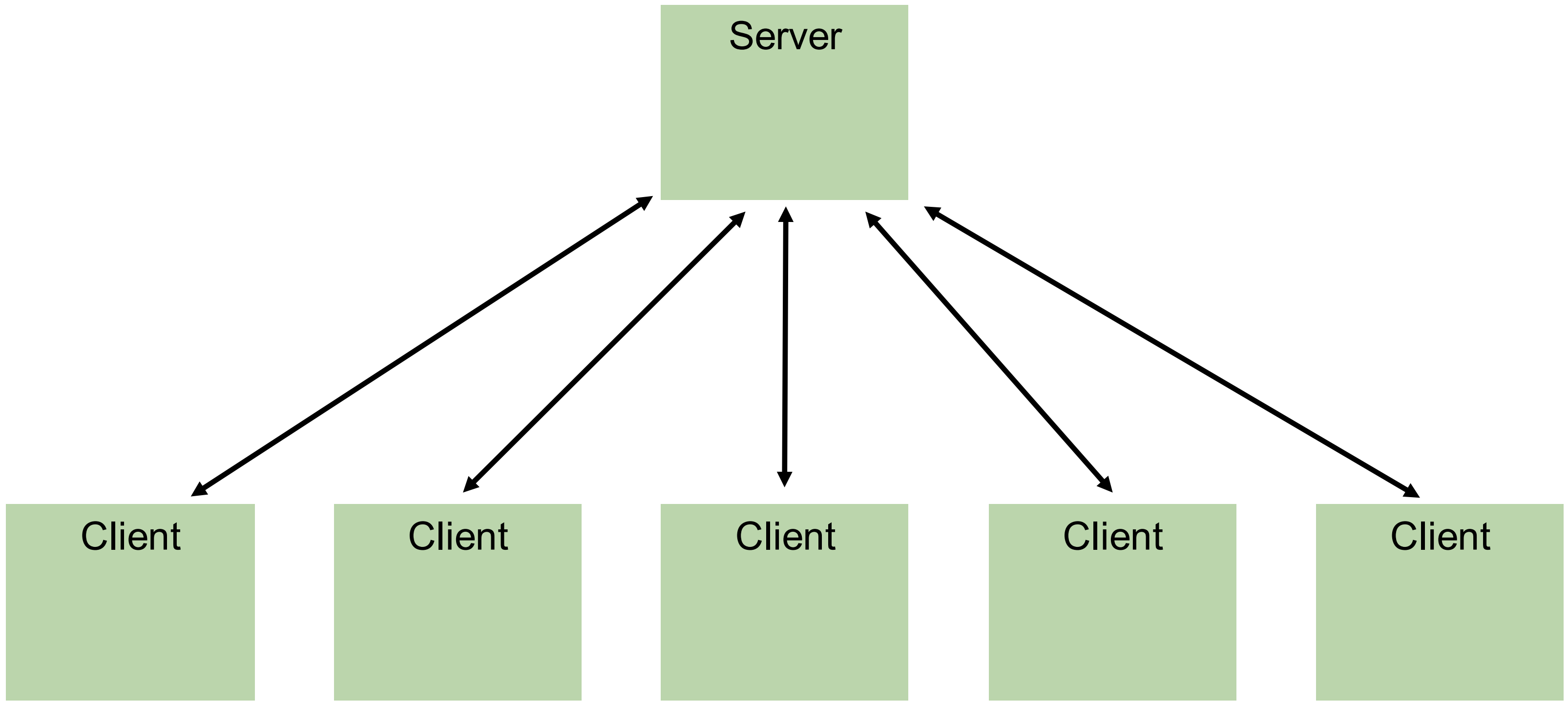
- More machines mean more links that can fail
- Networks introduce delays
- Networks still fail, intermittently and for long periods
- Networks rely on fallible external administrators
- Sequential consistency is impossible

A brief survey of distributed architectures

1. Monolithic server
2. Tiered architectures
3. Pipeline architectures
4. Event-driven architectures
5. Microservice architectures

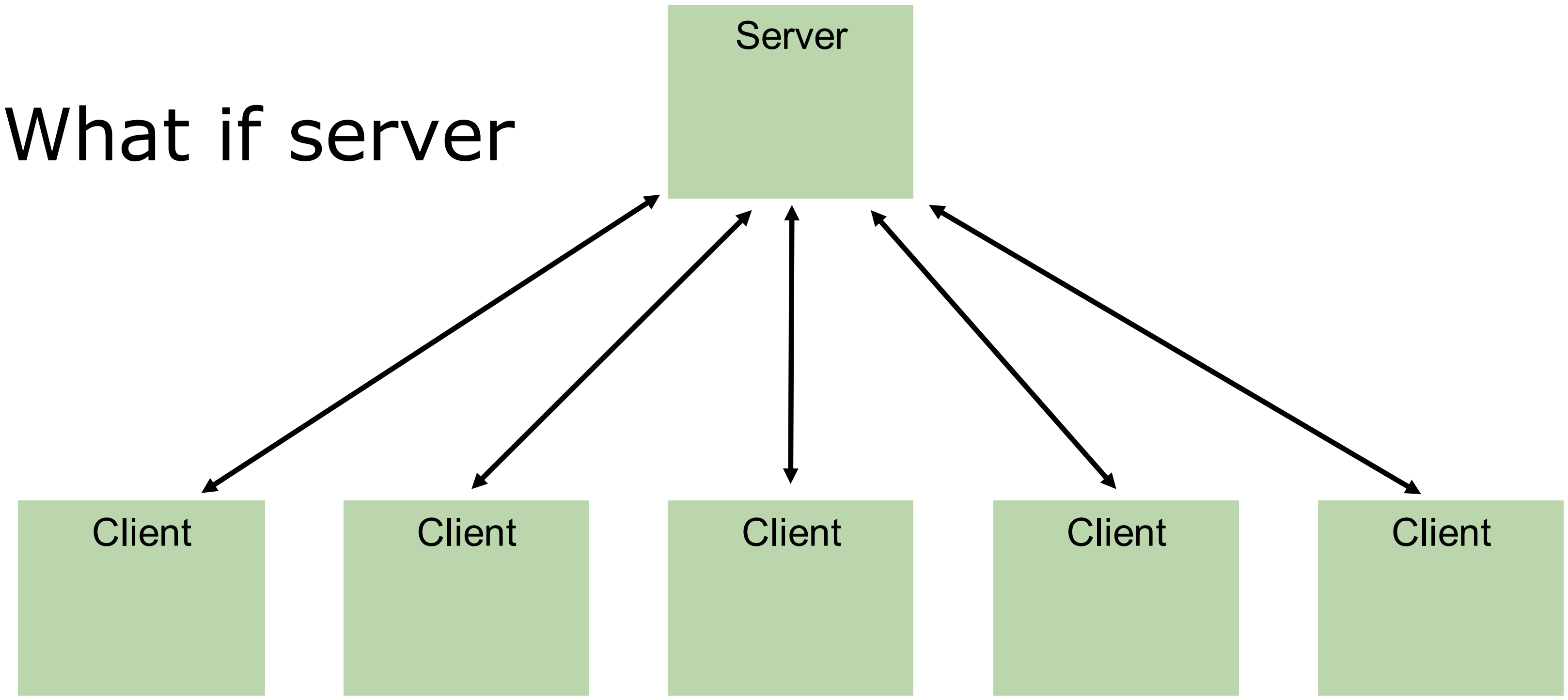
1. The Monolith Architecture Relies on a Single Server

- Simplest answer to consistency problem: have only one server, one source of truth
- Still “distributed” in that we have many clients
- Sacrifices:
  - Scalability
  - Performance
  - Fault tolerance



Monolithic Architectures Struggle to Scale

- Scalability - How to go from 10 to 100 to 1,000 clients?
- Performance - How to access 100’s of GB of data concurrently?
- Fault tolerance - What if server crashes?

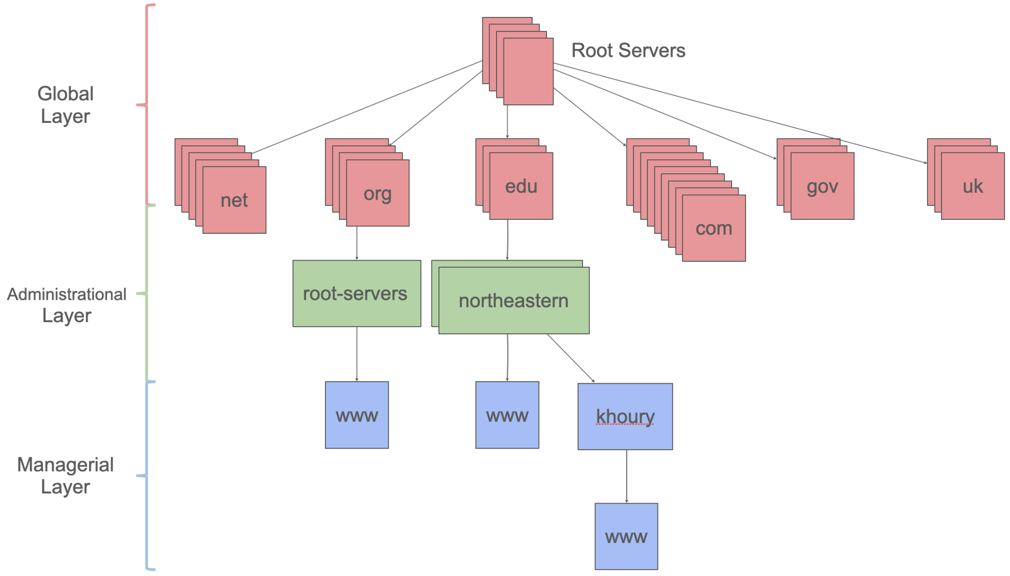


Replication Alone is Not The Answer

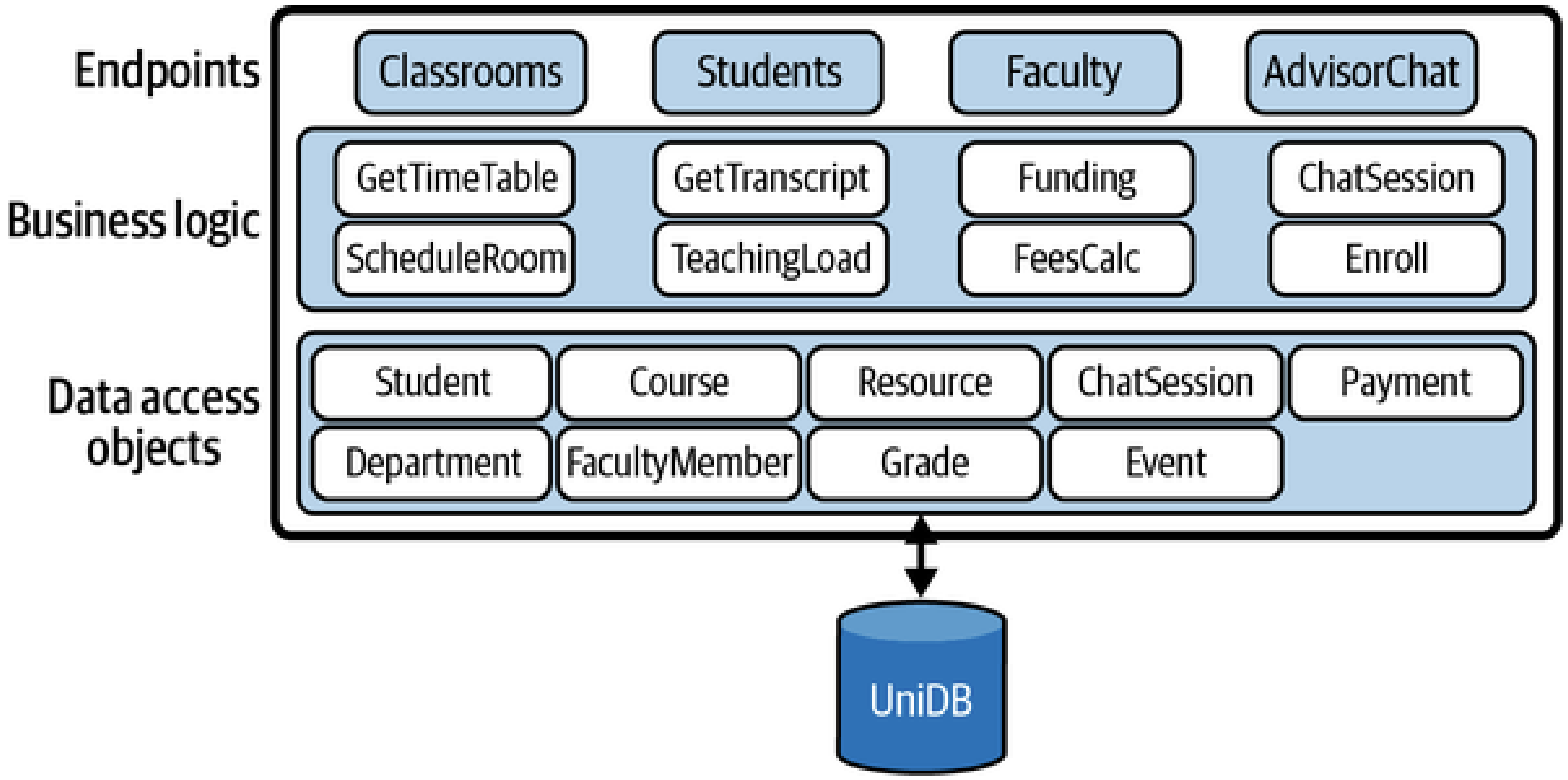
- Constraints:
  - Latency: Speed of light (~1ns/ft)
  - Throughput: Long-distance links between servers are relatively low throughput (10's of Gbps, compare to 100's of Gbps within a single server)
- Tradeoffs for replication, particularly over long distances:
  - Replication will *add* latency, not reduce it
  - Usually not enough bandwidth to maintain replication of all data across all nodes

Tiered Architectures

- Key idea: Partition the system into distinct tiers based on responsibilities
- Each tier scales independently of the others - .com need not know about .org
- Satisfying a single request may require multiple tiers
- DNS is a tiered architecture
  - Example: scale .com differently from .gov

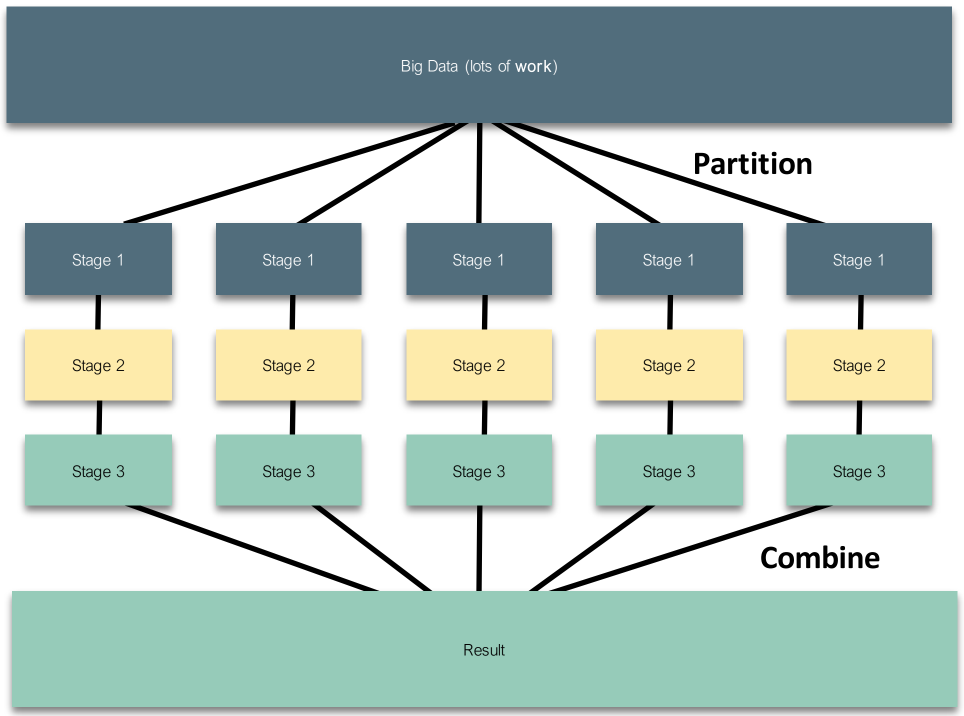


A tiered architecture is like a layered architecture, only distributed



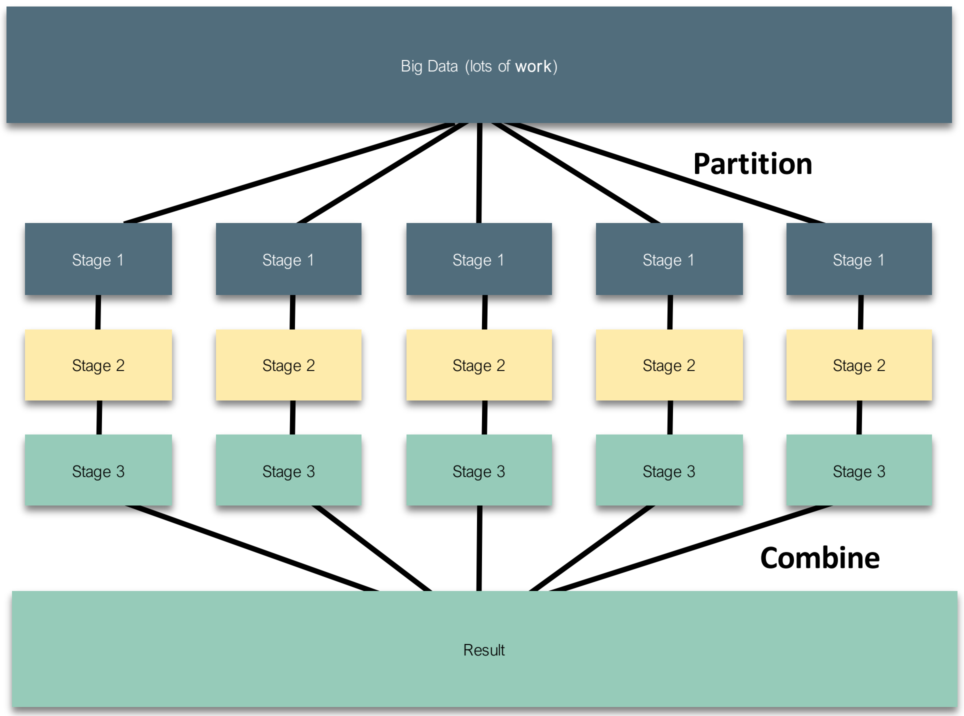
Pipeline Architectures

- The pieces correspond to stages in the transformation of data in the system
- Good for complex straight-line processes where multiple stages applied to different data, concurrently
- Each stage in the pipeline takes an input, produces an output: otherwise *stateless*
- Example: Map/Reduce splits data, filters it through stages, then combines
- Pipeline architecture allows flexibility in mapping stages to physical servers



Pipeline Architectures

- Scalability/Performance:
  - Add more machines to process more data in parallel
  - Limited by bandwidth to transfer inputs/outputs between stages
- Fault tolerance: Each stage in pipeline is stateless. If one fails, it can be repeated elsewhere.

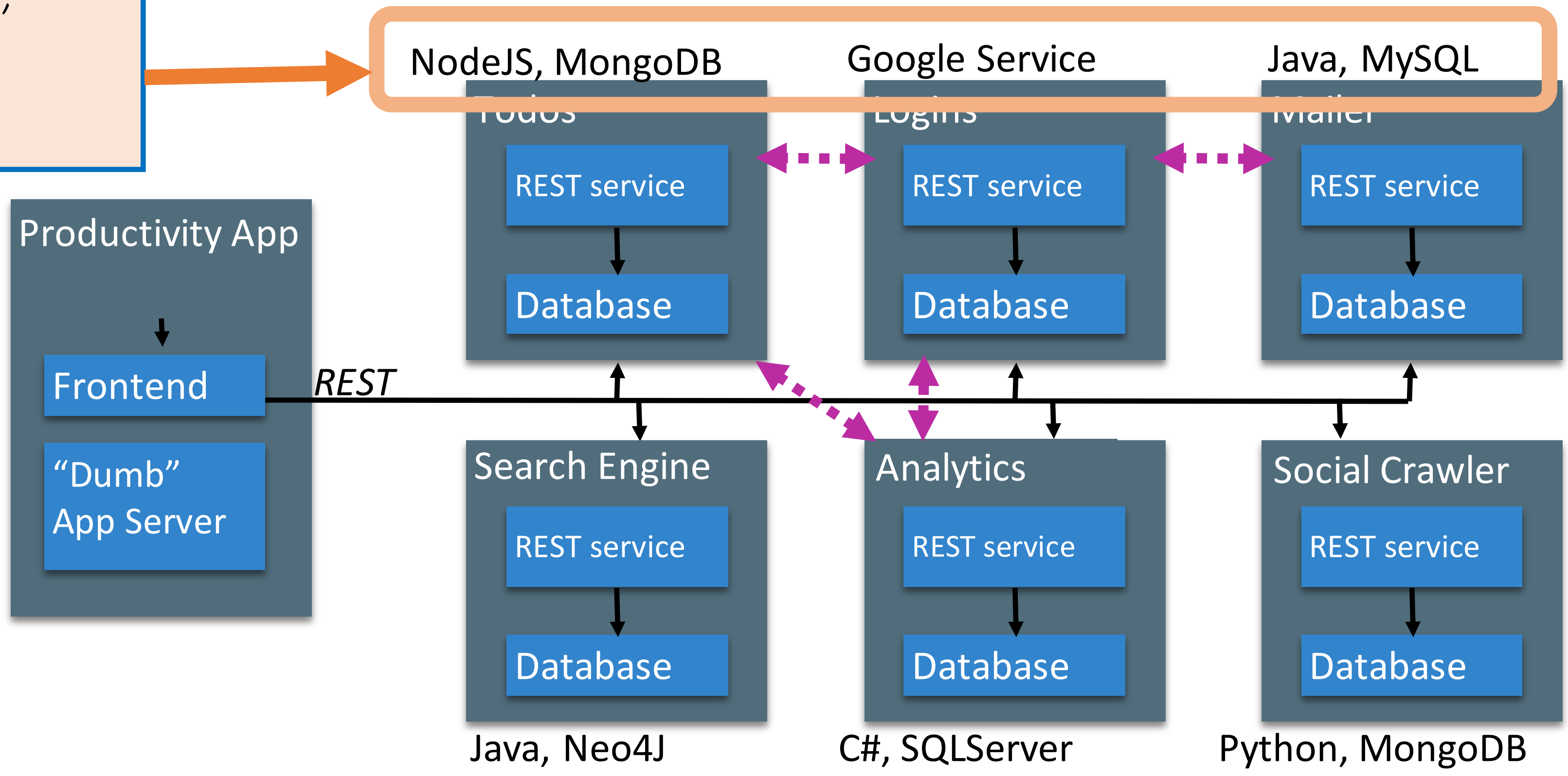


Microservice Architectures

- Organize implementation around components (responsibilities)
- Each component is implemented independently
- Each component is
  - independently replaceable,
  - independently updatable
- Components can be built as libraries, but more usually as web services
- Services communicate via a well-defined protocol (typically REST/http, though others are possible)

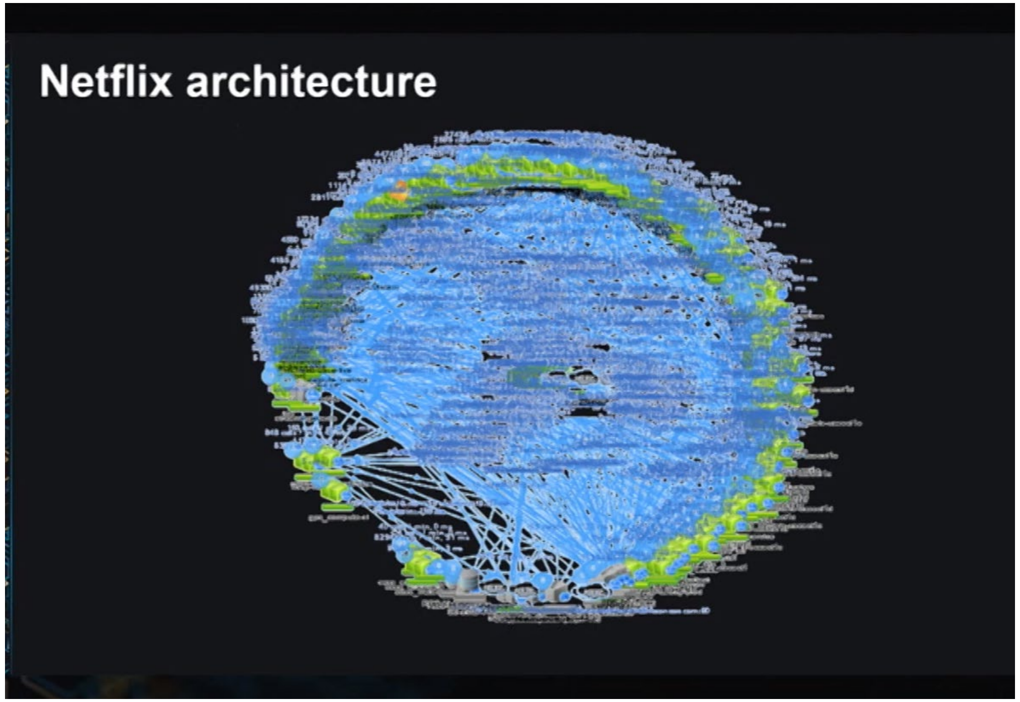
Microservices: Schematic Example

Different languages, different operating systems



Microservices are (a) highly scalable and (b) trendy

- Microservices at Netflix:
  - 100s of microservices
  - 1000s of daily production changes
  - 10,000s of instances
  - BUT:
  - only 10s of operations engineers



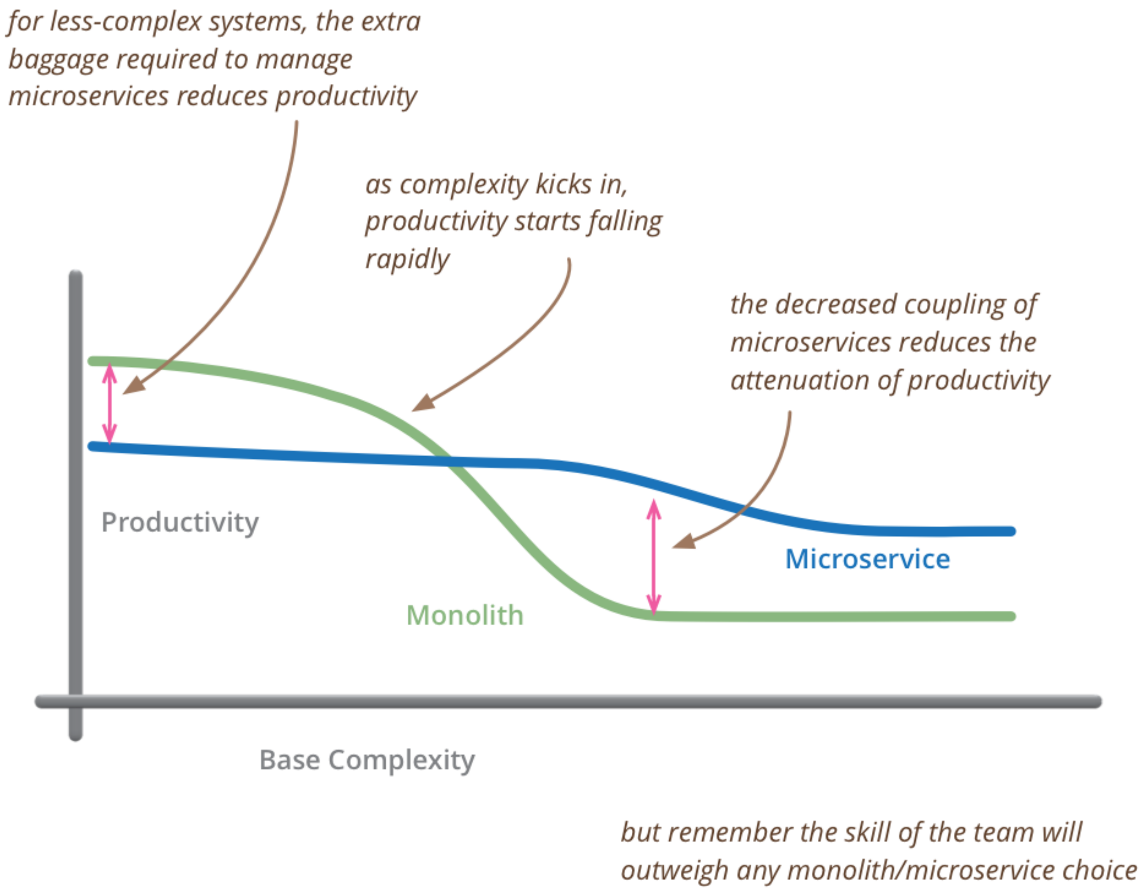
https://medium.com/refraction-tech-everything/how-netflix-works-the-hugely-simplified-complex-stuff-that-happens-every-time-you-hit-play-3a40c9be254b

Microservice Advantages and Disadvantages

- Advantages
  - services may scale differently, so can be implemented on hardware and software appropriate for each
  - services are independent (yay for interfaces!) so can be developed and deployed independently
- Disadvantages
  - Shared data?
  - Requires high availability
  - Service discovery?
  - Data consistency?
  - Overall system complexity

Microservices vs Monoliths

higher is better



https://martinfowler.com/microservices/

Learning Goals for this Lesson

- You should now be able to
  - Recognize a few common software architectures
  - Discuss some of the tradeoffs of scalability, performance, and fault tolerance between these architectures