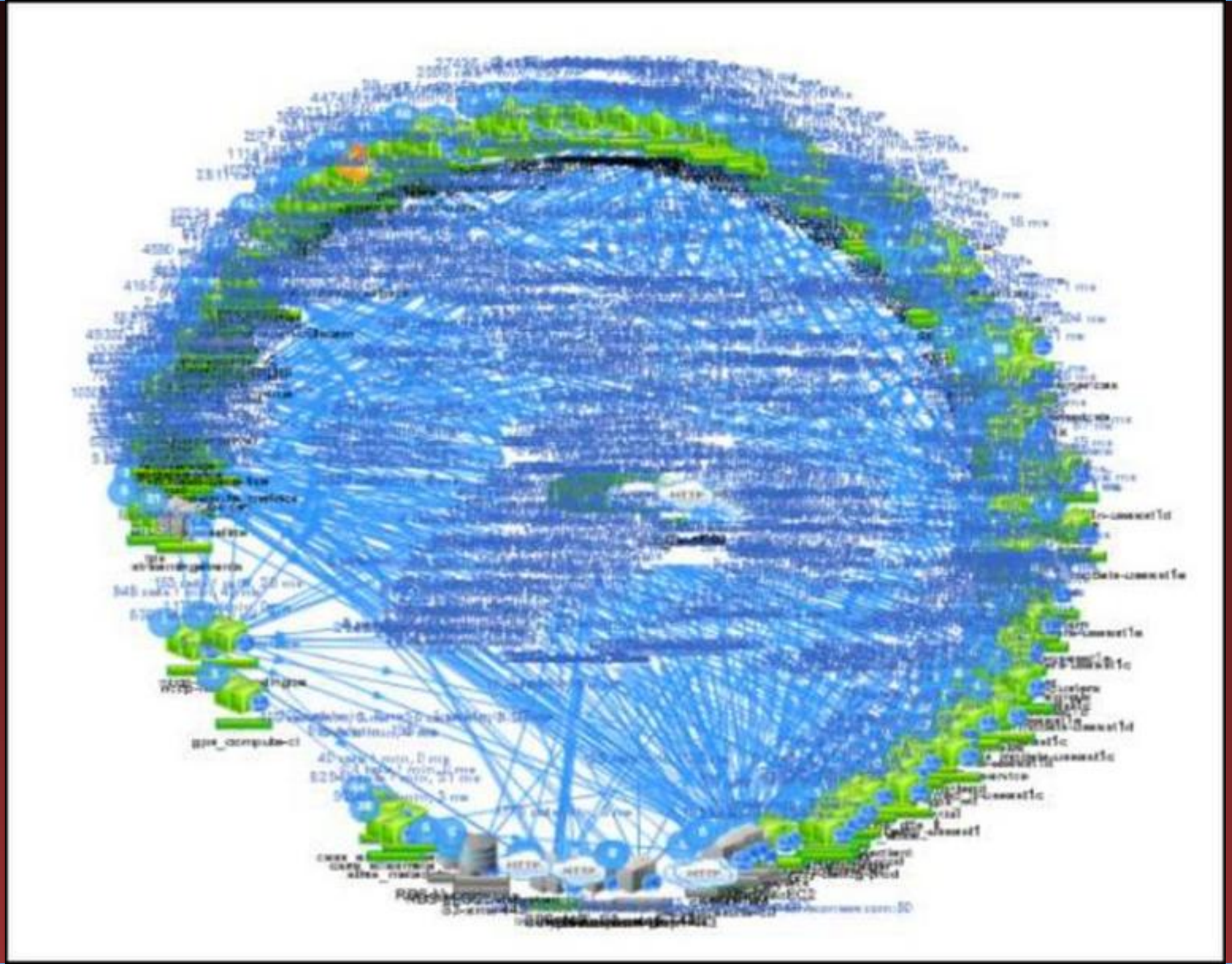
Spring Boot Microservices

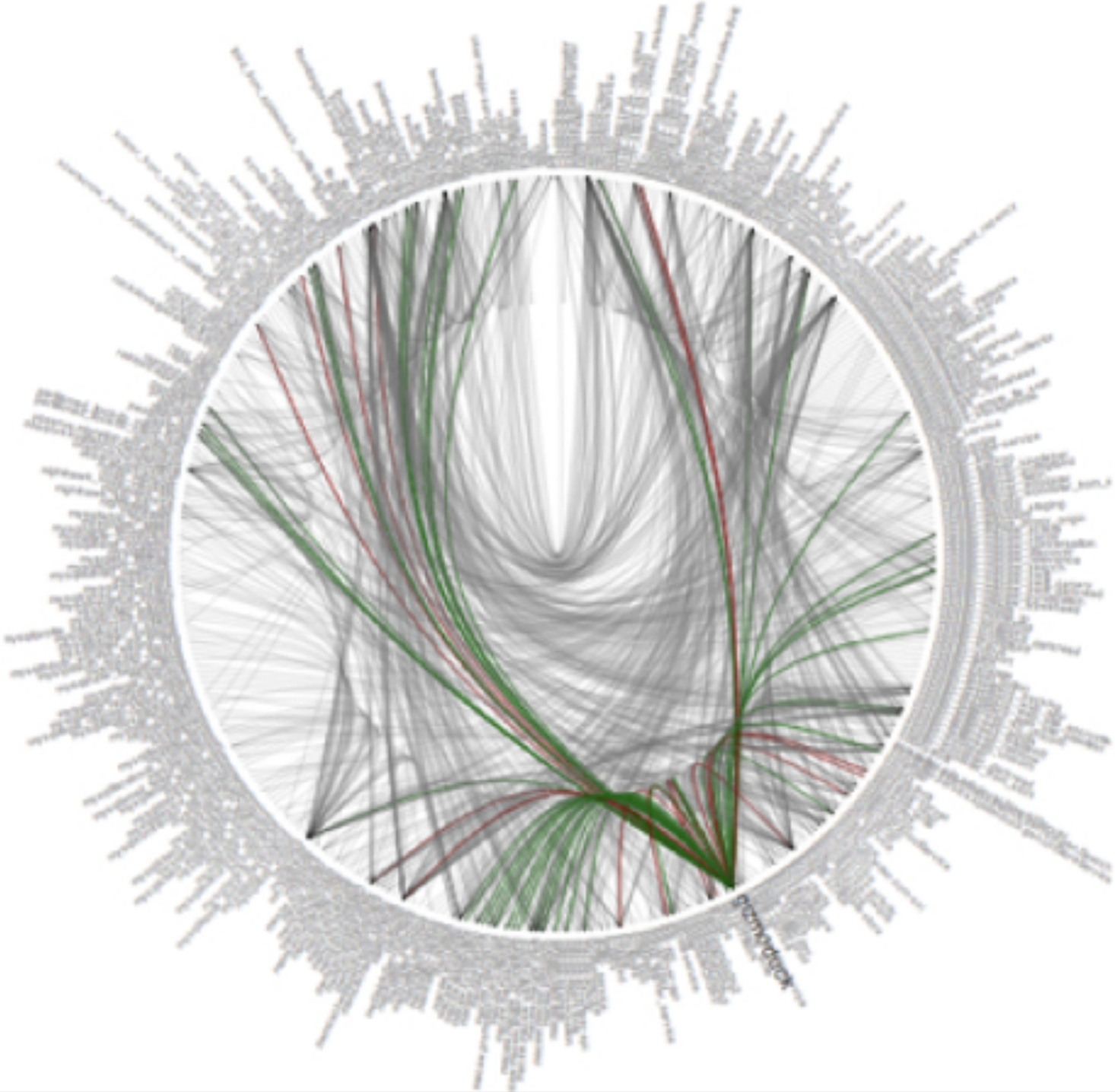
Beginner to Guru

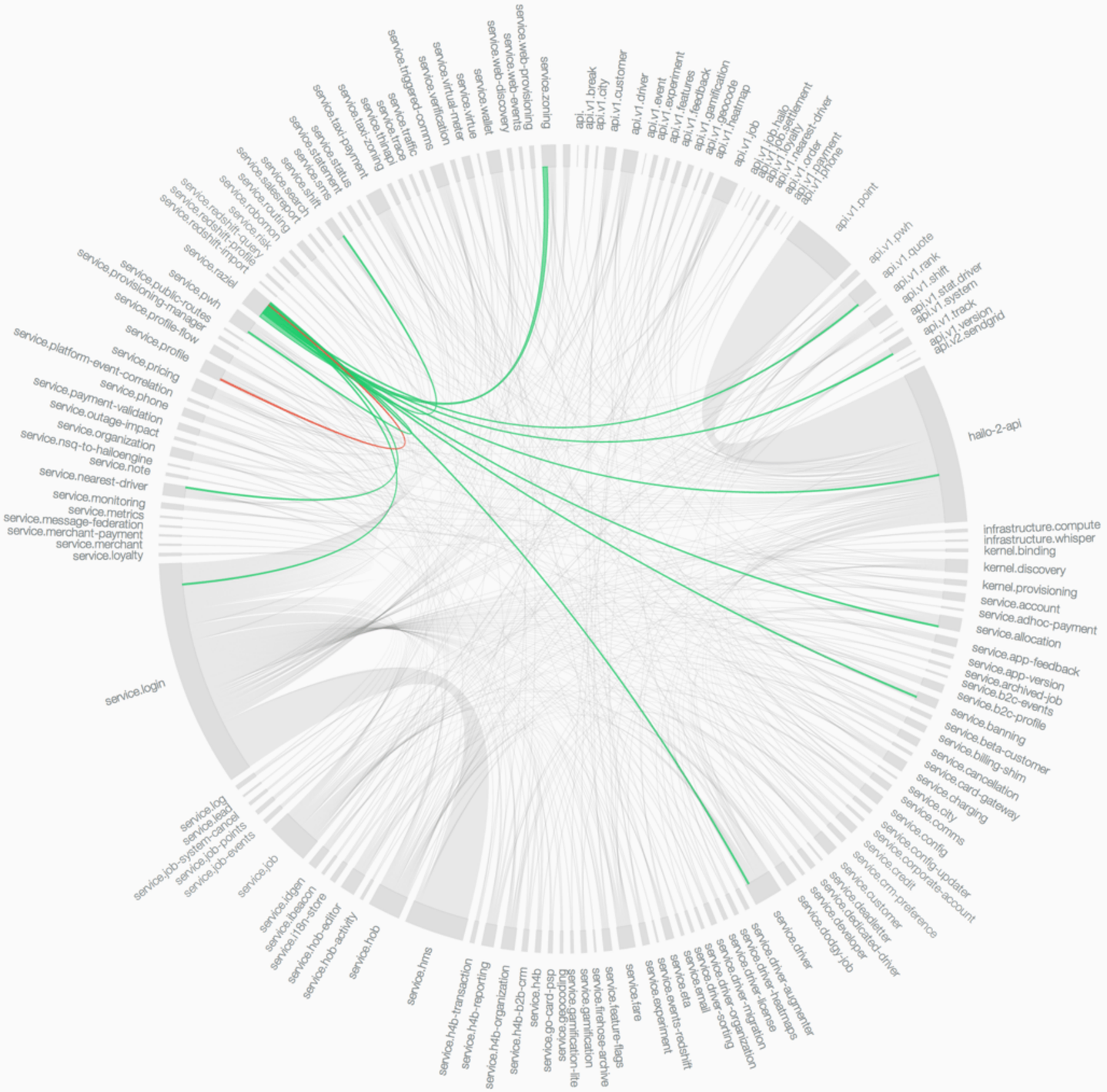
The Need for Sagas

The Microservice Death Star

- As the number of services grows, so does complexity
- Death Star - Examples
  - Netflix
  - Twitter
  - Uber





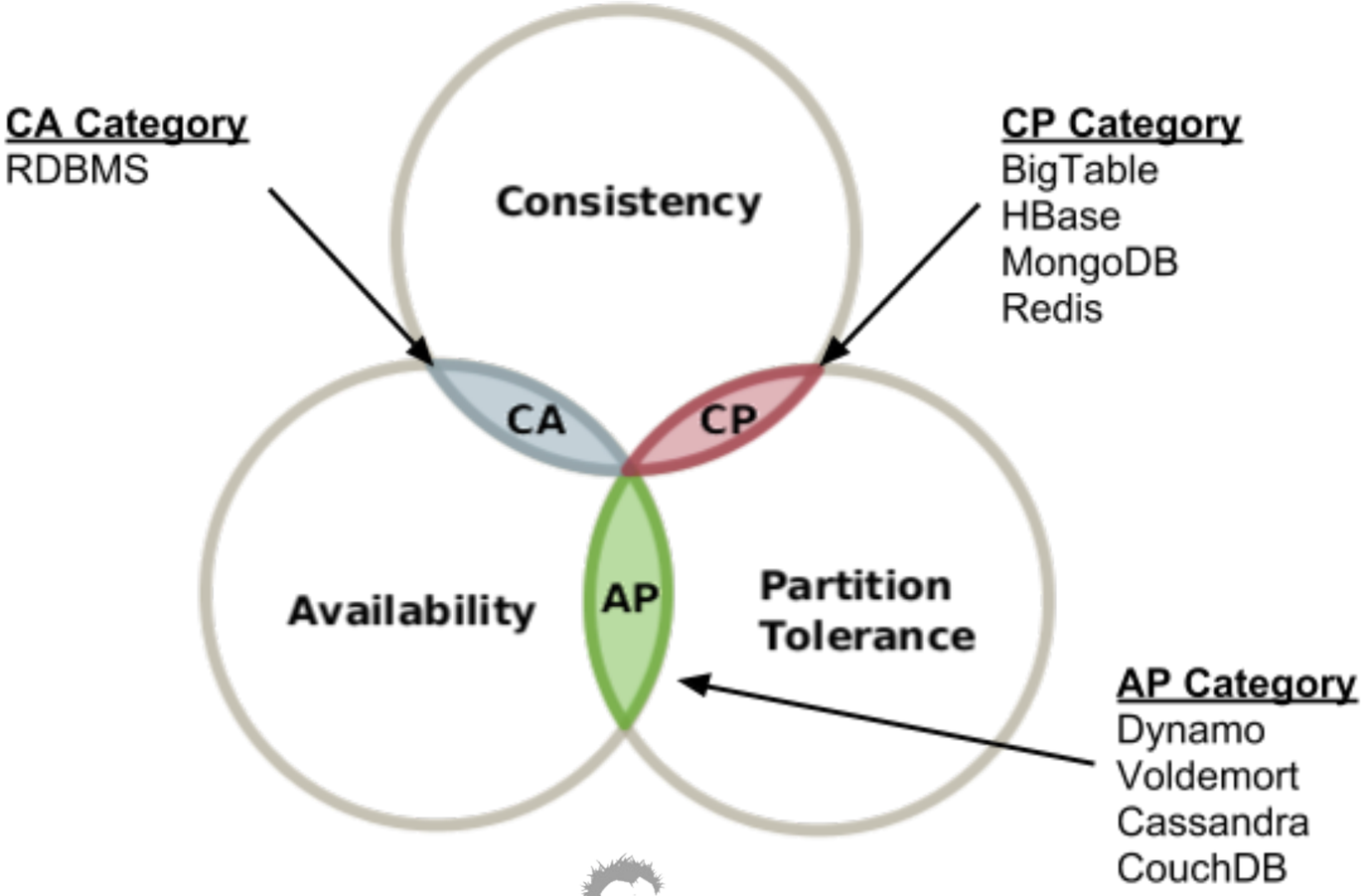

service.zoning
service.web-provisioning
service.web-events
service.web-discovery
service.wallet
service.virtue
service.virtual-meter
service.verification
service.traffic
service.triggered-comms
service.thinapi
service.taxi-zoning
service.taxi-payment
service.status
service.statement
service.sms
service.shift
service.search
service.salesforce
service.routing
service.robomon
service.risk
service.redshift-query
service.redshift-profile
service.redshift-import
service.razial
service.pwh
service.public-routes
service.provisioning-manager
service.profile-flow
service.profile
service.pricing
service.platform-event-correlation
service.phone
service.payment-validation
service.outage-impact
service.organization
service.nsq-to-hailoengine
service.note
service.nearest-driver
service.monitoring
service.metrics
service.message-federation
service.merchant-payment
service.merchant
service.loyalty
service.login
service.log
service.lead
service.job-system-cancel
service.job-points
service.job-events
service.job
service.idgen
service.ibeacon
service.i18n-store
service.hob-editor
service.hob-activity
service.hob
service.hms
service.h4b-transaction
service.h4b-reporting
service.h4b-organization
service.h4b-b2b-crm
service.h4b
service.geocode
service.gamification-lite
service.gamification
service.firehose-archive
service.feature-flags
service.fare
service.experiment
service.events-redshift
service.eta
service.email
service.driver-sorting
service.driver-organization
service.driver-migration
service.driver-license
service.driver-heatmaps
service.driver-augmenter
service.driver
service.dodgy-job
service.dedicated-driver
service.developer
service.deadletter
service.customer
service.crm-preference
service.credit
service.corporate-account
service.config-updater
service.config
service.comms
service.city
service.charging
service.card-gateway
service.cancellation
service.billing-shim
service.beta-customer
service.banning
service.b2c-profile
service.b2c-events
service.archived-job
service.app-version
service.app-feedback
service.allocation
service.adhoc-payment
service.account
kernel.provisioning
kernel.discovery
kernel.binding
infrastructure.whisper
infrastructure.compute
hailo-2-api
api.v2.sendgrid
api.v1.version
api.v1.track
api.v1.system
api.v1.stat.driver
api.v1.shift
api.v1.rank
api.v1.quote
api.v1.pwh
api.v1.point
api.v1.phone
api.v1.payment
api.v1.order
api.v1.nearest-driver
api.v1.loyalty
api.v1.location
api.v1.job.hailo
api.v1.job
api.v1.heatmap
api.v1.geocode
api.v1.gamification
api.v1.feedback
api.v1.features
api.v1.experiment
api.v1.event
api.v1.driver
api.v1.customer
api.v1.city
api.v1.break
api.

Challenges

- Business transactions often span multiple microservices
- ACID transactions are not an option between services
- Distributed Transactions / Two Phase Commits
  - Complex and do not scale
- Microservices should be technology agnostic
  - Making 2PC even more difficult to implement

CAP Theorem

- **CAP** - Consistency, Availability, Partition Tolerance
  - **Consistency** - Every read will have the most recent write
  - **Availability** - Each read will get a response, but without the guarantee data is most recent write
  - **Partition Tolerance** - System continues in lieu of communications errors or delays
- **CAP Theorem** - States a distributed system can only maintain two of three
  - Consistency, Availability, Partition Tolerance


CA Category
RDBMS
Consistency
CP Category
BigTable
HBase
MongoDB
Redis
CA
CP
Availability
AP
Partition
Tolerance
AP Category
Dynamo
Voldemort
Cassandra
CouchDB

BASE - An ACID Alternative

- **BASE** - Coined by Dan Pritchett of EBay in 2008
- **BASE** = **B**asically **A**vailable, **S**oft state, **E**ventually consistent
  - The opposite of ACID
- **Basically Available** - Build system to support partial failures
  - Lost of some functionality vs total system loss.
- **Soft State** - Transactions cascade across nodes, it can be inconsistent for a period of time
- **Eventually Consistent** - When processing is complete, system will be consistent

Feral Concurrency Control

- *Feral Concurrency Control: An Empirical Investigation of Modern Application Integrity*
  - Published by Peter Baillis in 2015
- **Feral Concurrency Control** - Are application level mechanisms for maintaining database integrity
  - Relational Databases can enforce a variety of constraints - such as foreign key constraints
  - Not available within a distributed system
  - Thus up to the application to enforce constraints

Introducing Sagas

- Concept introduced in 1987 by Gracia-Molina / Salem of Princeton University
- Originally was looking at Long Lived Transactions (LLTs) within a single database
  - LLTs hold on to database resources for an extended period of time
  - In 1987 computational resources were much more scarce
- Paper proposed rather than long complex processes to break up into smaller more atomic transactions
- Introduced the concept of compensating transactions to correct partial executions

Sagas

- Sagas are simply a series of steps to complete a business process
- Sagas coordinate the invocation of microservices via messages or requests
- Sagas become the transactional model
- Each step of the Saga can be considered a request
- Every step of the Saga has a compensating transaction (request)
  - Semantically undoes the effect of the request
  - Might not restore to the exact previous state - but effectively the same

Saga Steps

- Each step should be a message or event to be consumed by a microservice
- Steps are asynchronous
- Within a microservice, it's normal to use traditional database transactions
- Each message (request) should be idempotent
  - Meaning if same message / event is sent there is no adverse effect on system state
- Each step has a compensating transaction to undo the actions

Compensating Transactions

- Compensating Transactions
  - Effectively become the 'Feral Concurrency Control'
  - Are the mechanism used to maintain system integrity
  - Should also be Idempotent
  - Cannot abort - need to ensure proper execution
  - Not the same as a 'rollback' to the exact previous state
    - Implements business logic for a failure event

Sagas are ACD

- **A** - **Atomic**
  - All transactions are executed or compensated
- **C** - **Consistency**
  - Referential integrity within a service by the local database
  - Referential integrity across services by the application - 'Feral Concurrency Control'
- **D** - **Durability**
  - Persisted by database of each microservice

Sagas & Eventually Consistent

- **BASE** = **B**asically **A**vailable, **S**oft state, **E**ventually consistent
- During execution of the Saga system is in the 'Soft State'
- Eventually consistent - meaning system will be consistent at the conclusion of the Saga
  - Consistency achieved via normal completion of the Saga
  - In the event of an error, consistency is achieved via compensating transactions

Summary

- **Saga Definition** - "*A long, involved story, account, or series of incidents*"
- Microservices by nature run in a distributed environment, across many computers
- Sagas are used to address the challenges faced when operating in a distributed environment
- Sagas are a tool used to coordinate a series of steps for a business transaction across multiple services
- Sagas not only prescribe the series of necessary steps, they also maintain system integrity

SPRING FRAMEWORK

GURU