"LET'S MIGRATE OUR ENTERPRISE APPLICATION TO BIG DATA TECHNOLOGY IN THE CLOUD" - WHAT DOES THAT MEAN IN PRACTICE?

SUMMERSOC 2015

JUNE 30, 2015, ANDREAS TÖNNE

NOVATEC | SummerSoc Service Oriented Computing

About the Author and NovaTec Consulting GmbH

Andreas Tönne

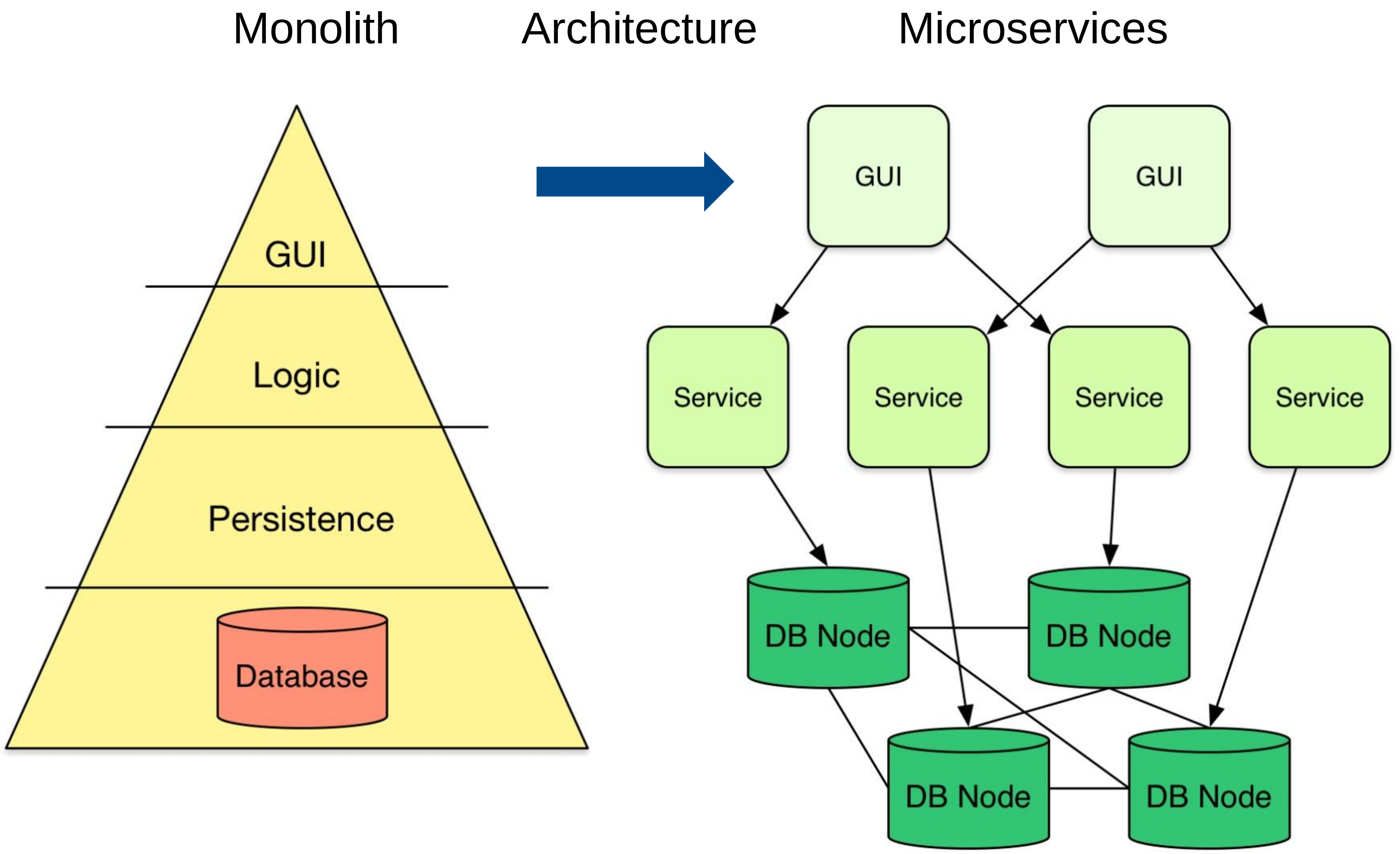
Director R&D and Senior Managing Consultant in the Frankfurt/M office

NovaTec Consulting GmbH

IT-Consulting company with a strong focus on Java enterprise application development, business process management, SOA and agile methodologies

Headquarters in Leinfelden-Echterdingen and Offices in Berlin, Munich and Frankfurt

Goal: Transform a Java EE Architecture to a BigData Architecture



- Practical experiences of a Big Data transformation project
- Cloud transformation strategy and consequences for the application requirements

Why Bother? OLTP and Big Data are Two Different Beasts

Big Data is a diagnosis of high Volume, Variety, Velocity

- Online Transaction Processing applications feel the pressure of Big Data
  - Growing scope of the applications
  - Modern sources of data with heterogeneous data formats
  - Swift competition

- Big Data architectures and technology are a promising solution to handle this
- At a price!

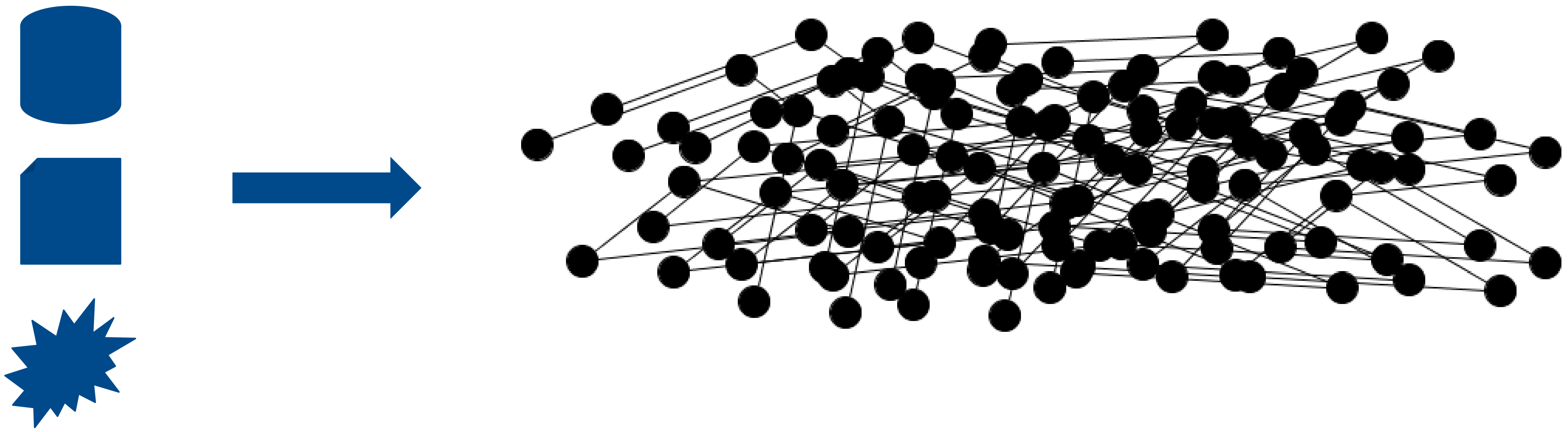
- Business thinks transactional with ACID properties
  - Consistency constraints and isolation are used to hedge against concurrent execution errors

- Cloud-native microservices architecture puts an emphasis on concurrent execution

Big Data Transformation Example Project

Semantic middleware to provide information logistics to enterprises

- Consolidates heterogeneous data sources like relational DB, file shares, websites or tweets in a uniform data model
- Computes weighted relations using syntactical and semantic algorithms
- Provides access through queries and contextual traversal
- Samples: search on steroids, lightweight ETL, MDM, document annotation



Big Data Transformation Example Project - Woes

The previous version of the product did not scale at all!

  - System breakdown at moderate data sizes instead of up to petabytes of data (**Volume**)
  - Inacceptable low throughput of the import (**Velocity**)

- Previous architecture: best practices Java EE with clustering and Lucene as index
  - Mapping of graph model to relational store considered the bottleneck
  - Replace the persistence layer by one using a graph database and all is good?

- Measured up to 95% of execution time spent in commits (DB and Lucene)
- Why?

Facing two killer problems

1. Storage architecture does not scale to the customer requirements
2. Application design does not scale at all

Big Data Transformation Strategy

Analyze, Understand, Plan!

- Jumping right to action is a seriously bad idea (proven)

1. Architecture workshops with the original developers and business
2. Runtime analysis using profiling and tracing tools (inspectIT.eu)
3. Code reviews to confirm / understand the runtime findings
4. Lock the original developers in a cupboard ☺
5. Plan the new architecture concept based on requirements
6. Pilot implementation

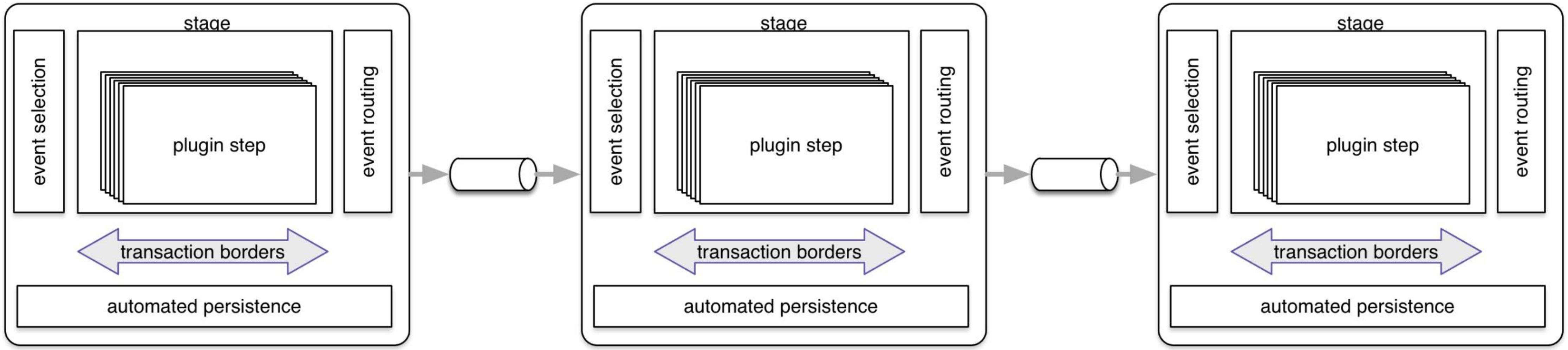
New Architecture Strategy

Paramount requirements: 1. Scalability, 2. Scalability, 3. Scalability, 4. Analysis quality

Strategy:

1. Decomposition of services: client API, import, content agents, periodic jobs
2. Decompose the import service into a chain of independently scalable computing steps
   E.g.: dispatch of update/delete, storage, indexing, statistical information, various semantic analysis
3. Allow analysis plugins
4. Compose a import workflow topology using the SEDA model
5. Put everything in the cloud (public or private)

Staged Event Driven Architecture (SEDA) – Pipes & Filter Pattern



- Chains of microservices based on Spring Boot and the JMS template
- ActiveMQ provides load balancing and persistence of the workflow
- Shared Titan / Cassandra / Elasticsearch cluster

- Analysis plugins configured as Spring beans

Benefits of This New Architecture

- Group plugin steps by their resource needs
- Adapt the number of instances of each stage to the current needs
- Achieve the best possible data locality (graph DB caches)
- Use different technology in the stages
- Have a single source of truth for transaction optimization

Benefits for the project:

- Small components are developed independently
- Short backlogs per service instead of everything mixed in one big project
- Experimentation friendly – Each service is cheap to come and go
- Effective isolation of problems

Big Data Transformation Example Project – Requirements Woes

New architecture highlights scalability bottleneck: strict consistency requirements to assure optimal quality

Uniqueness of relations

- Concurrent import of related documents could produce duplicate relations
- Example: my <user> matches your <author> and vice versa
- Uniqueness constraint on the DB required lots of short transactions

Quality of weight computation

- Business mandates perfect counts of e.g. words in the DB
- Synchronized counters create a perfect bottleneck

Decision: Drop ACID properties for BASE

Replace strong consistency by eventual consistency (and compensation)

How to Get Away With "Eventually Consistent"

The CAP theorem is a weak excuse since most of the time there are no partitions

Real reason: availability and horizontal scalability on a Big Data scale

Stronger models of consistency require a lot of agreement of the distributed nodes

- The system seems to stop until agreement is achieved

OLTP applications often do not admit a share-nothing architecture

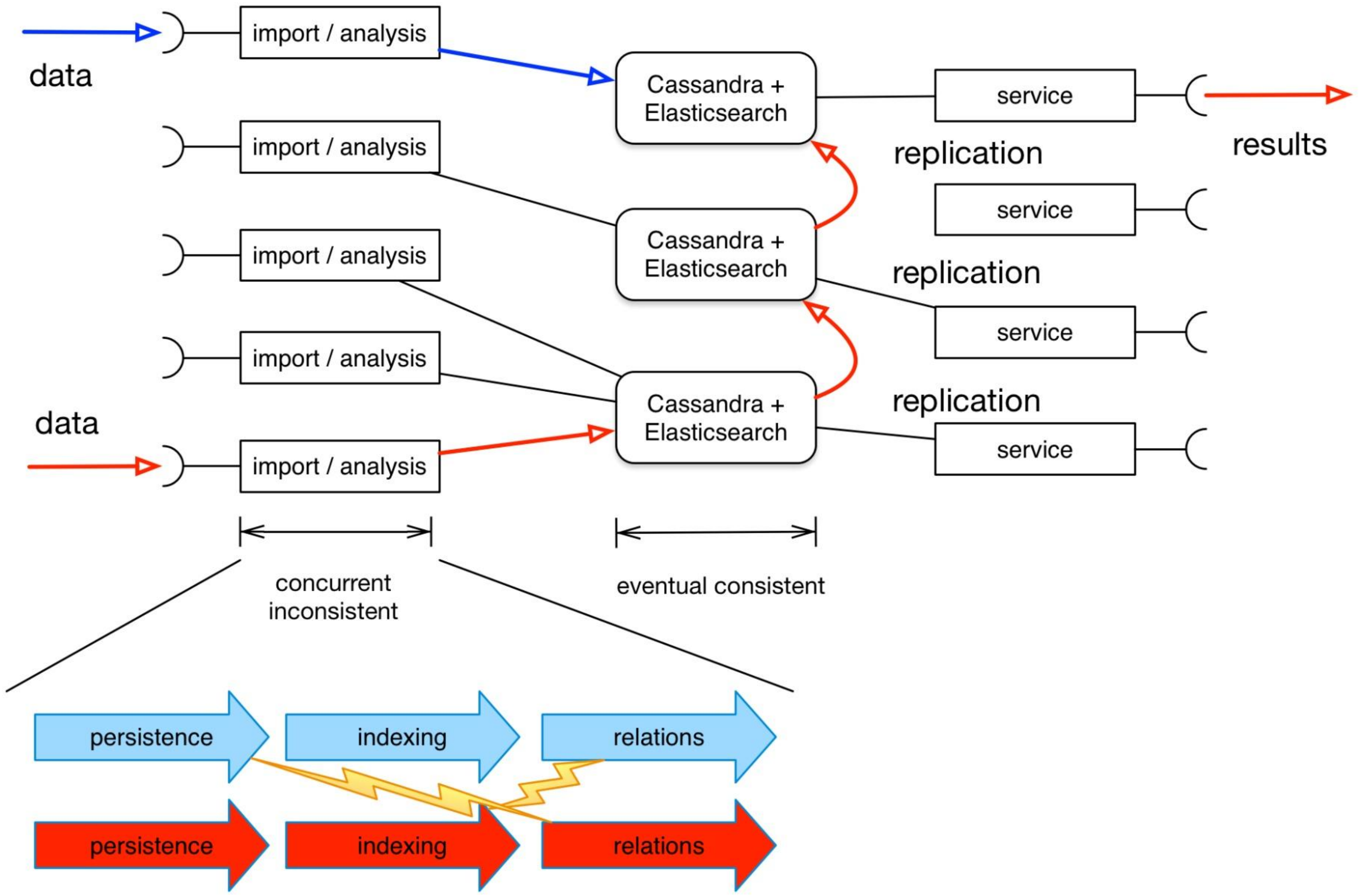
Business requirements demand logical or technical locking

Measurement: lock-aquisition in Cassandra took up to 40ms!

-> we could create only 25 unique keywords per second

Eventually Consistent is No Law of Nature But a Reasoned Decision

Eventual consistency creates a timeframe of uncertainty



Eventual Consistency Requires a Re-Evaluation of Requirements

Expect long and tiring discussions with business

Invariants of concurrency need to be understood

Example (Uniqueness constraints on relations):

- How frequent is the possible violation of uniqueness?
- What is the impact of this violation on the quality of relations?
- Are other requirements and services impacted by the violation
- Do these violations build up over time?
- Should we filter the duplicate relations on the API layer or should we perform periodic compensation jobs?

Difficult questions if ACID transactions protected you in the past

We favoured scalability over a possible reduction of the quality of relations

Conclusions

Transactional enterprise applications can be transformed to a Big Data architecture

Splitting a monolith into independent microservices is a good match for Big Data

The need for a Big Data architecture usually comes with a strong requirement for scalability

Some consistency focused requirements can no longer be uphold

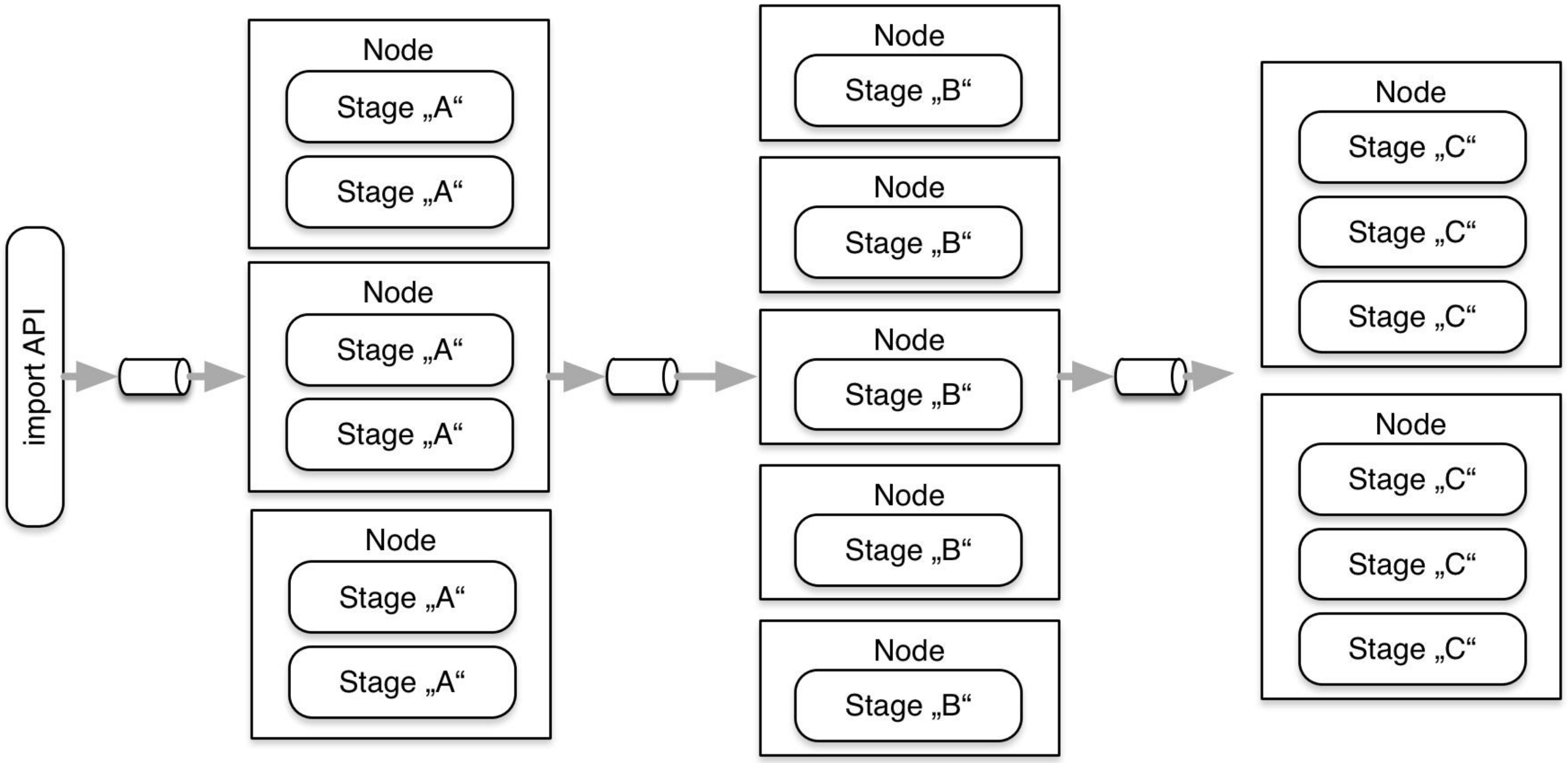
Business needs to re-evaluate these requirements and accept compromises

NOVATEC

THANK YOU FOR YOUR ATTENTION

JUNE 30, 2015, ANDREAS TÖNNE

Sample Stage Topology



NOVATEC

Background

ACID – Anyone?

BASE

- Basically available, soft state, eventually consistent

Eventually Consistent

- Single value consistency! (sometimes called atomic consistency)
- New value written to a node is replicated in the background
- An unbound, finite time later all available nodes show the same version of the value

CAP Theorem

- In a distributed system you can have only two of:
  - consistency
  - availability
  - partition tolerance

Practical Consequences of Choosing Scalability Over Consistency

Phantom reads

- Concurrent update of two records does not see the updated other record
- The relations graph becomes locally random
- Solution: Keep a journal of updated records and re-analyse in an overlap

Read not your own writes

- Chains of import/analysis steps become dependent on the replication speed of the eventual consistent storage
- Results of previous step may not be immediately accessible
- Solution: Exponential back off retries or brute-force delays between steps

Zombies

- Sequences of update and delete of the same record could overrun each other
- Solution: Timestamps for monotonic order of import and writing of tombstones

Practical Consequences of Choosing Scalability Over Consistency Cont.

Creating uniqueness

- Locks are too expensive
- Violation of uniqueness (e.g. keywords) was not admissible
- Would create a partitioning of the relation graph
- Solution: Prefabricated unique records based on sample data in a warm-up run

Accuracy of tallies and other statistical data

- Accuracy of these numbers strongly affect the quality of the relation weights
- Global synchronized counters slowed down the system to a great extent
- Periodically refreshed local caches introduced an unknown error rate
- Solution: Use probabilistic cardinality estimator like HyperLogLog
  This yields a very good performance with a known error probability