Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Descartes research

How we built a scalable micro-service application

- lessons learned & tooling -

Nikolas Herbst,
Jóakim von Kistowski, Simon Eismann, André Bauer,
Norbert Schmitt, Johannes Grohmann, Marwin Züfle,
Samuel Kounev

ScrumScale Workshop, Oslo, Norway
June 5, 2018

Slides available: descartes.tools

We are

Chair of Software Engineering (a.k.a. Descartes Research Group) at the University of Würzburg, Germany, Franconia (part of Bavaria)

- Performance Modeling and Benchmarking,
  Data Center Resource Management,
  Self-Aware Computing, Data Analytics
- New: IoT, CPS, I4.0, Block chain, Ethical hacking, …

On my research

- Started research after diploma in 2012 at Karlsruhe Institute of Technology (KIT)
- Research Interests:
  - Cloud Computing
  - Elasticity and Scalability
  - Auto-Scaler Benchmarking
  - Forecasting
  - …

LIMBO

Cloud Elasticity Benchmark BUNGEE

Telescope

Chameleon

SPEC Research

Mission Statement

- Provide a **platform for collaborative research efforts** in the area of quantitative system evaluation and analysis
- Foster interactions and **collaborations** between industry and academia
- **Scope**: computer benchmarking, performance evaluation, and experimental system analysis
- **Focus on** standard scenarios, metrics, benchmarks, analysis methodologies and tools

Working groups:

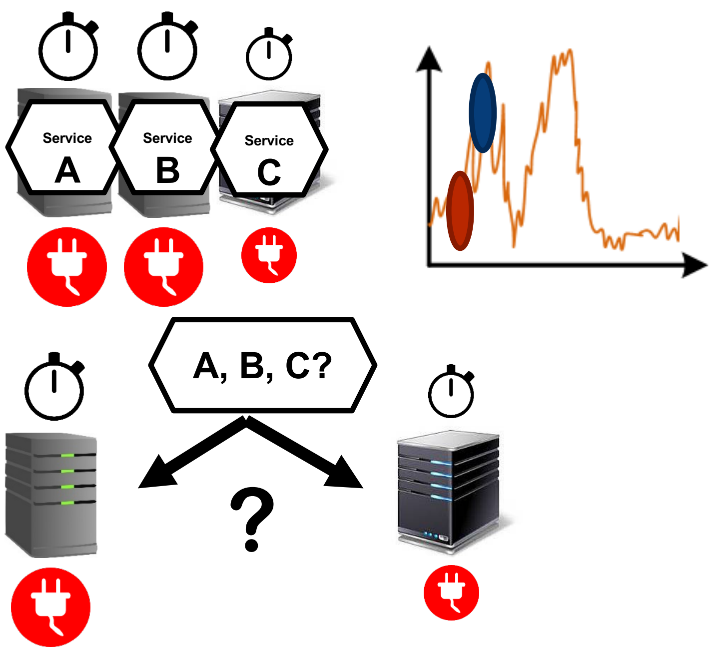
Cloud, DevOps Perf., Power, IDS & Security, Big Data

Find more information on: http://research.spec.org

Why TeaStore ? Our Motivation

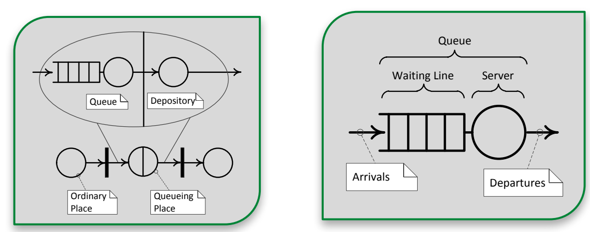
Auto-Scaling and Placement

- Placement at run-time



Performance Modeling

- An approach for the auto-scaling + placement problem
- Build or extract model



- Use Model for placement decision

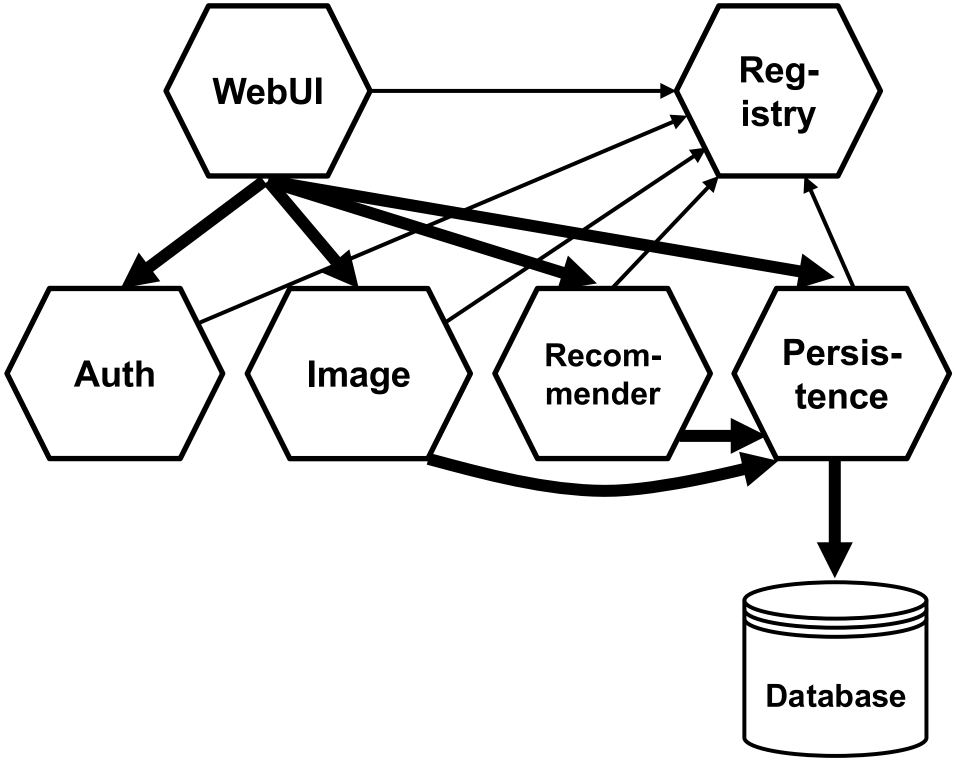
Requirements for a Reference Application

- **Highly scalable**
- Deployment flexibility at run-time
- Reproducible performance results
- Complex performance behavior
- Failover and reliable
- Online monitoring
- Load Profiles for realistic stress
- Simple setup
- Modern technology stack

The Descartes TeaStore

Micro-Service test application

- Five Services + Registry
- Uses Netflix "Ribbon" client-side load balancer
  - Swarm/Kubernetes supported, not required
- Pre-instrumented version with Kieker application monitoring
- Docker Images
  - Alternatively: manual deployment in application server (documentation available)



Services I

Registry

- Simplified Eureka
- Service location repository
- Heartbeat

RegistryClient

- Dependency for every service
- Netflix “Ribbon”
- Load balances for each client

WebUI

- Servlets/Bootstrap
- Integrates other services into UI
- **CPU + Memory + Network I/O**

Authentication

- Session + PW validation
- SHA512 + Bcrypt
- **CPU**

Services II

PersistenceProvider

- Encapsulates DB
- Caching + cache coherence
- **Memory**

ImageProvider

- Loads images from HDD
- 6 cache implementations
- **Memory + Storage**

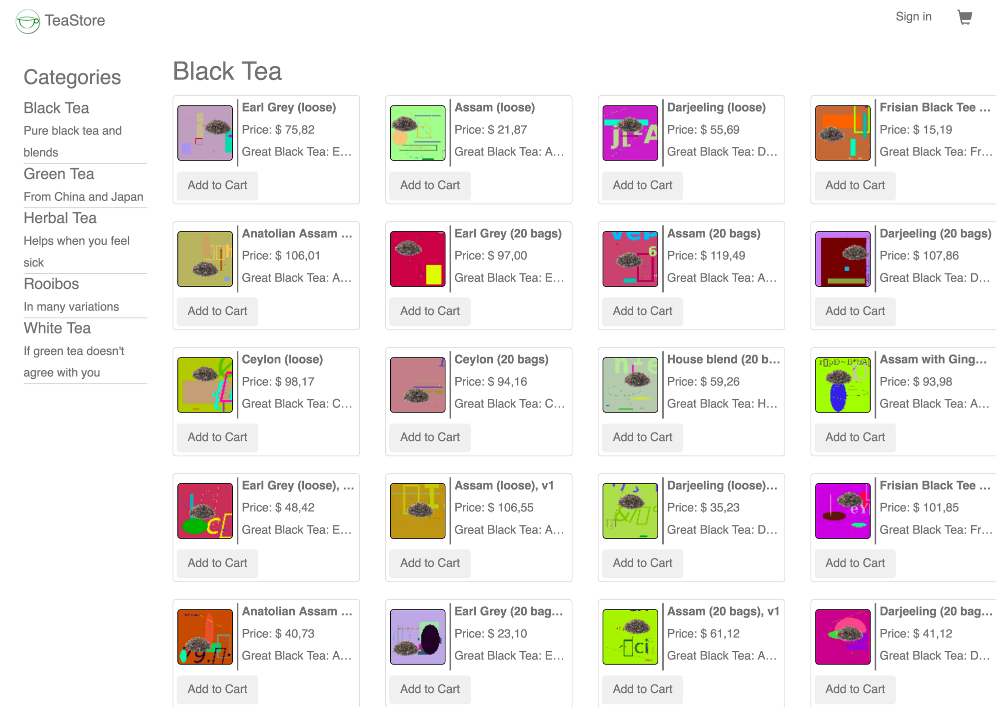
Recommender

- Recommends products based on history
- 4 different algorithms
- **Memory or CPU**

TraceRepository

- AMQP Server
- Collects traces from all services

TeaStore Demo



Open Source – Apache License v2

https://github.com/DescartesResearch/TeaStore

Performance: Characteristics & Configurations

Two types of caches

- Black-box persistence cache
- White-box image provider cache

Different load types

- CPU
- I/O
- Network

Internal state

- Database size influences resource demands

Load independent tasks

- Periodic recommender retraining (optional)

Startup behavior

- Auth and WebUI start "instantly"
- Recommender needs training on startup
- Image Provider creates images on startup

Configuration options

- Recommender algorithms
- Recommender retraining interval
- Image Provider cache implementations
- Database size

Load and Usage Profile

HTTP load generator

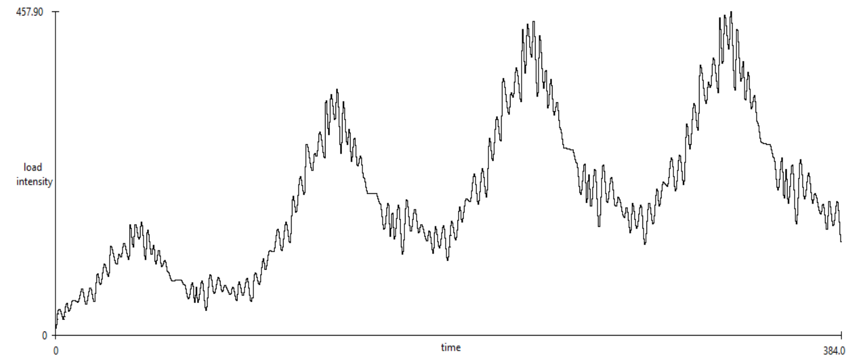
Supports load intensity profiles

- Can be created manually
- Or using LIMBO (more later)

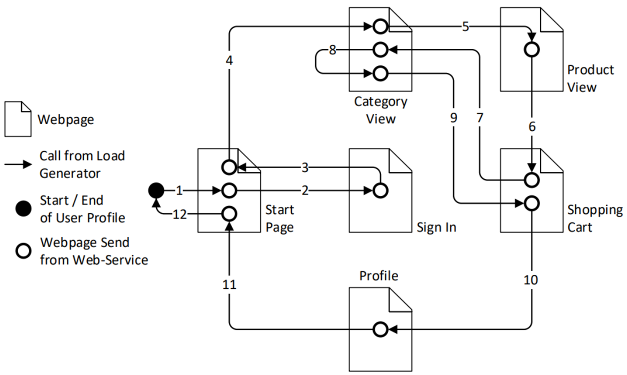
Scriptable user behavior

- Uses LUA scripting language
- e.g. "Browse" Profile on Github

Example load intensity profile:



"Browse" user profile:



1. GET Start Page
2. GET Sign In
3. POST Send login data for random user
4. GET Category View for random category and 30 items per page
5. GET Product View for random product on first page
6. GET Shopping Cart; add currently viewed product
7. GET Category View with previous category
8. GET Category View with random page number
9. GET Shopping Cart; add random product from current category view
10. GET Profile
11. GET Start Page and Logout
12. Start Over

Does it scale?

First stress tests:

- Very **limited scalability** due to communication overhead !
- Image provider service was network bound (no caching)
- All services: running out of ports and connections due to standard Java networking (connections, sockets)

→ Okay, let us reuse connections via connection pooling

→ Introduce image caching (service instance & client side)

Does it scale? (II)

Second version stress test:

- Somewhat better scalability, still not sufficient
- Performance variability
- Connection pool size configuration important, but specific for service type, platform and load

  → not a good idea to set a default in a service container image

→ Okay, think and re-implement one more time…

Does it scale? (III)

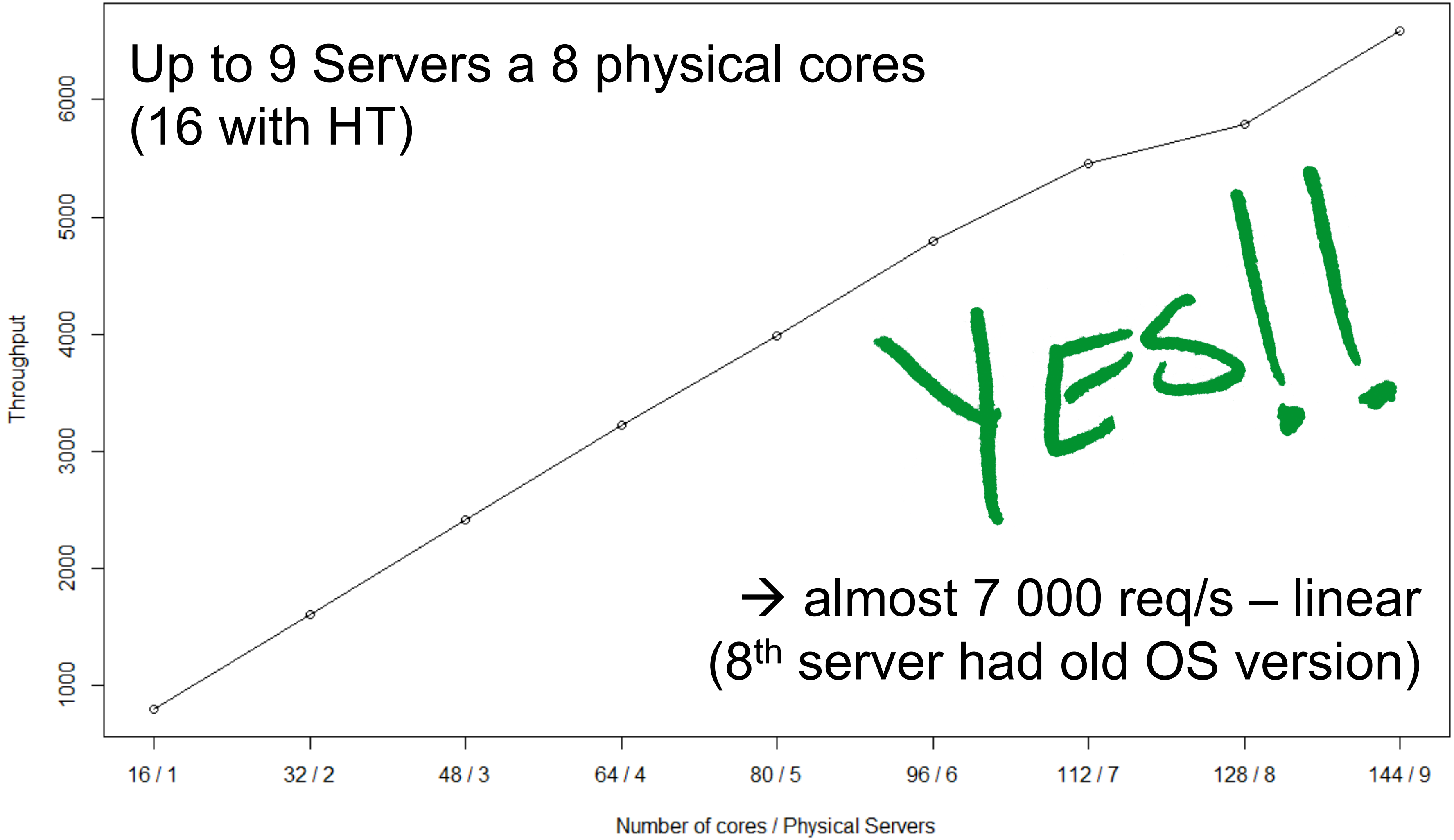
Third version towards scalability:

- Asynchronous communication

- Based on Java NIO APIs
  (multi-plexed, non-blocking I/O)
  - Leverages network card HW features
  - Managed buffers, worker and thread pools
  - Channel listener concept for Java servelets

Frameworks: Undertow (JBoss) or Grizzly NIO (Glassfish)
https://javaee.github.io/grizzly/

Does it scale? (IV)

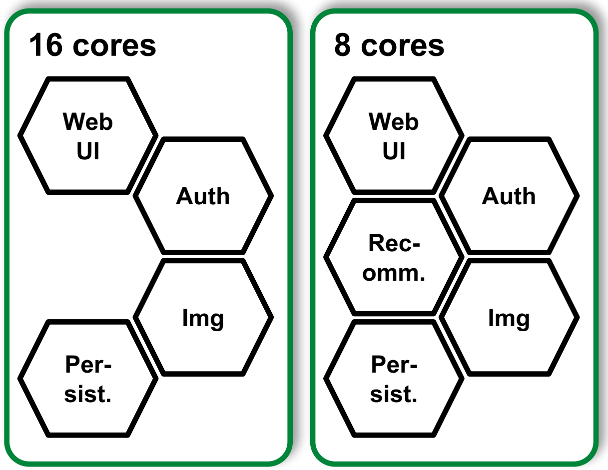


Up to 9 Servers a 8 physical cores (16 with HT)

→ almost 7 000 req/s – linear (8th server had old OS version)

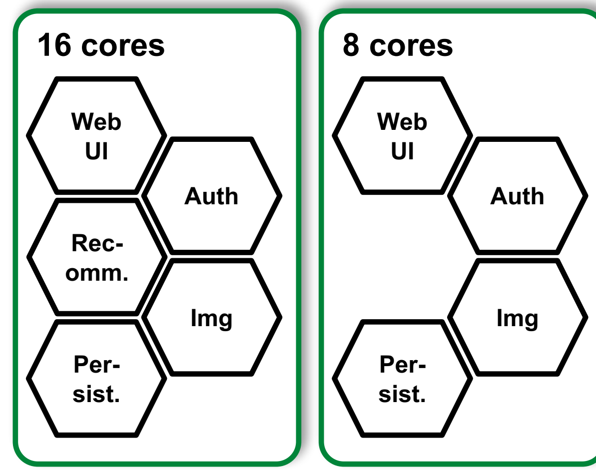
Example: Energy Efficiency of Placements

Placement 1



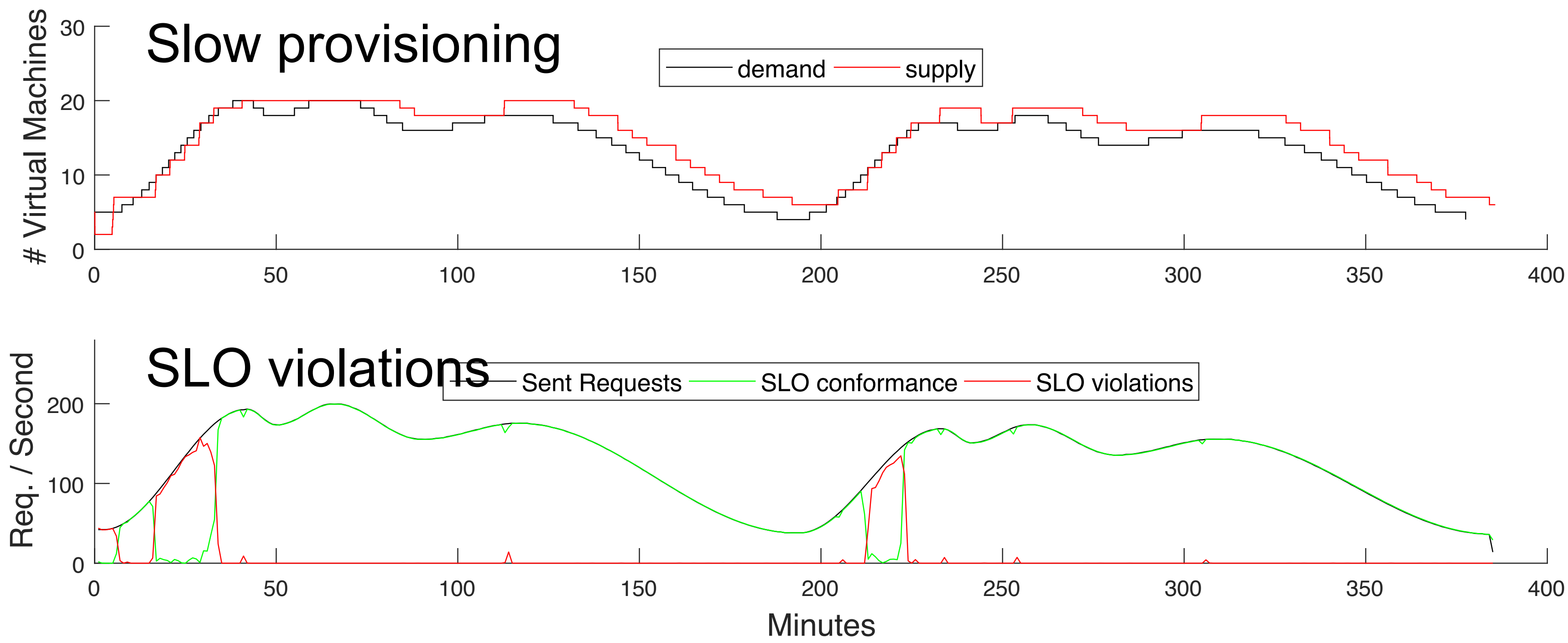
Max	1011.9 Tr/s
Max	179.6 W
Geo	4.4 Tr/J

Placement 2

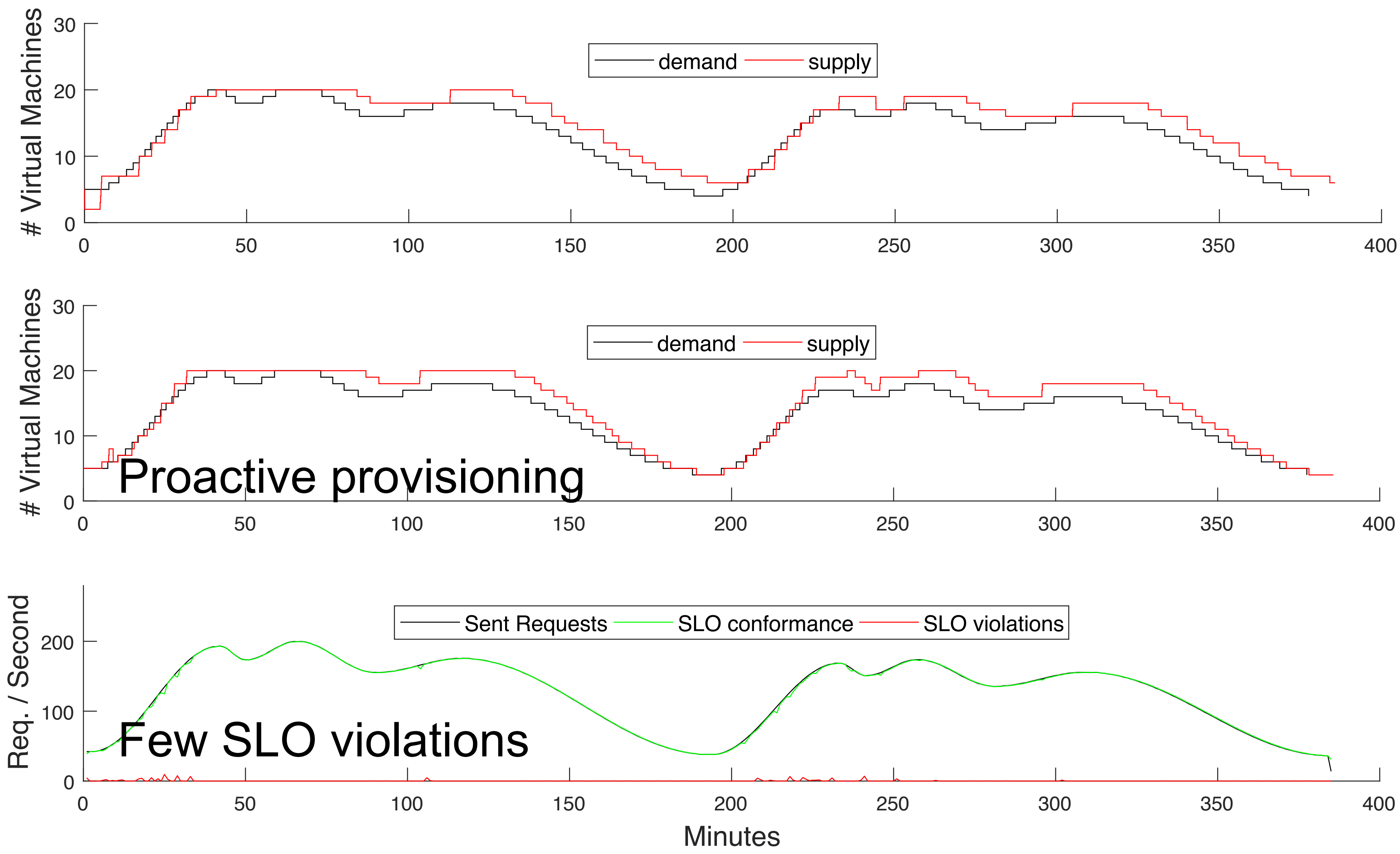


Max	1067.7 Tr/s
Max	187.0 W
Geo	4.3 Tr/J

Auto-Scaling TeaStore



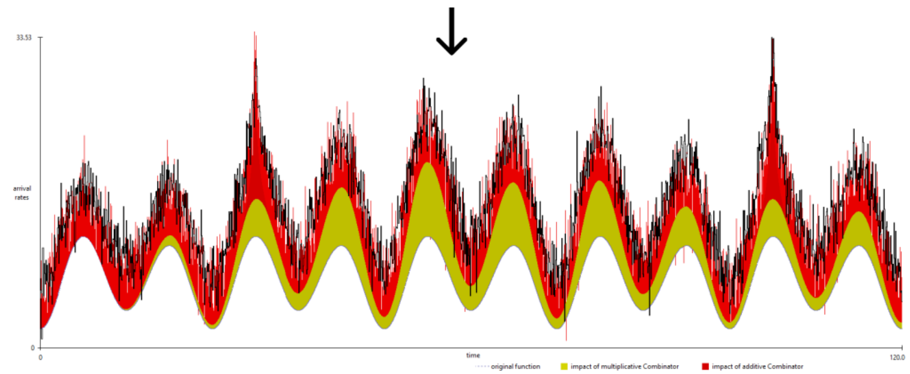
Auto-Scaling TeaStore



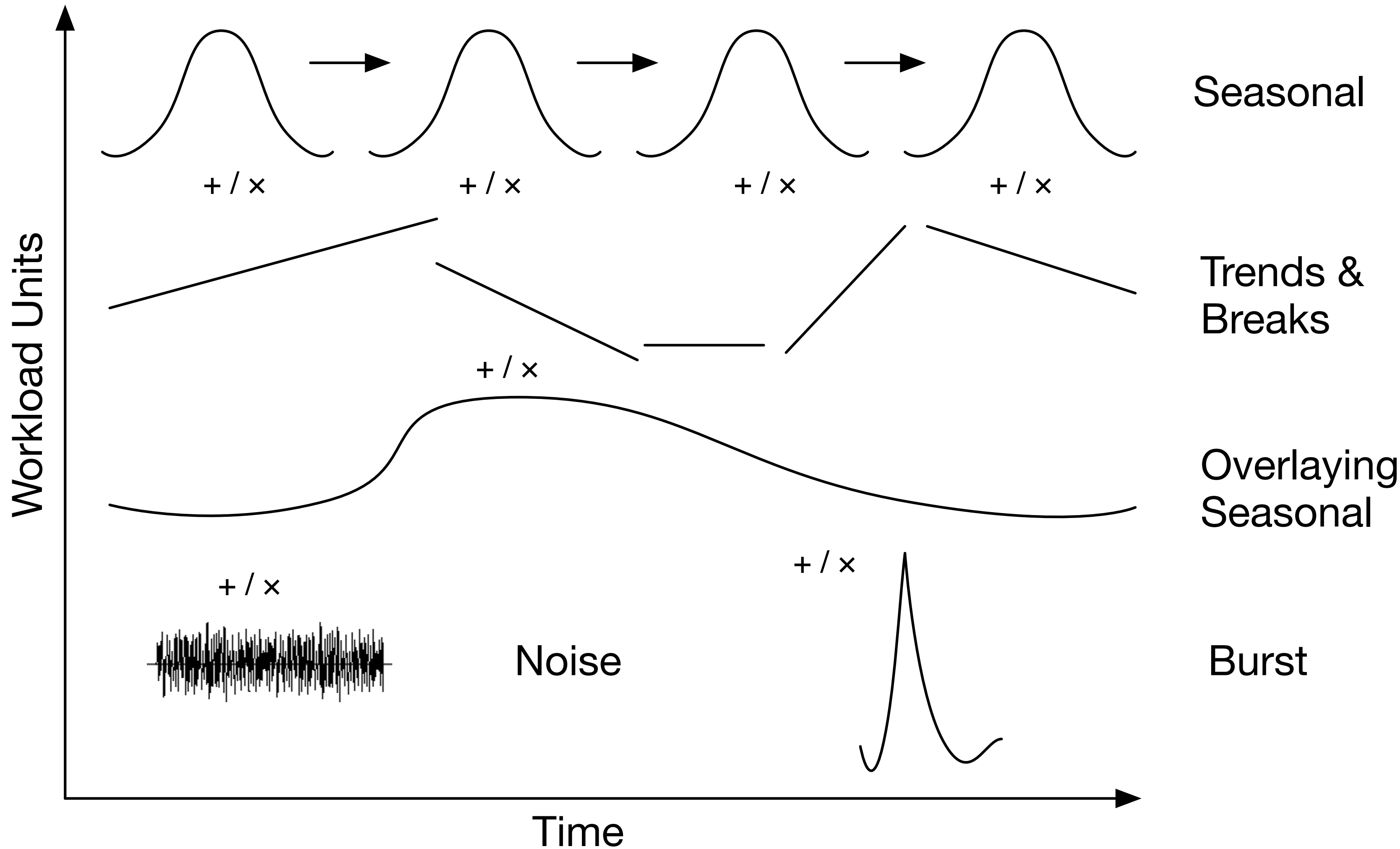
http://descartes.tools/limbo

Load Profile Models

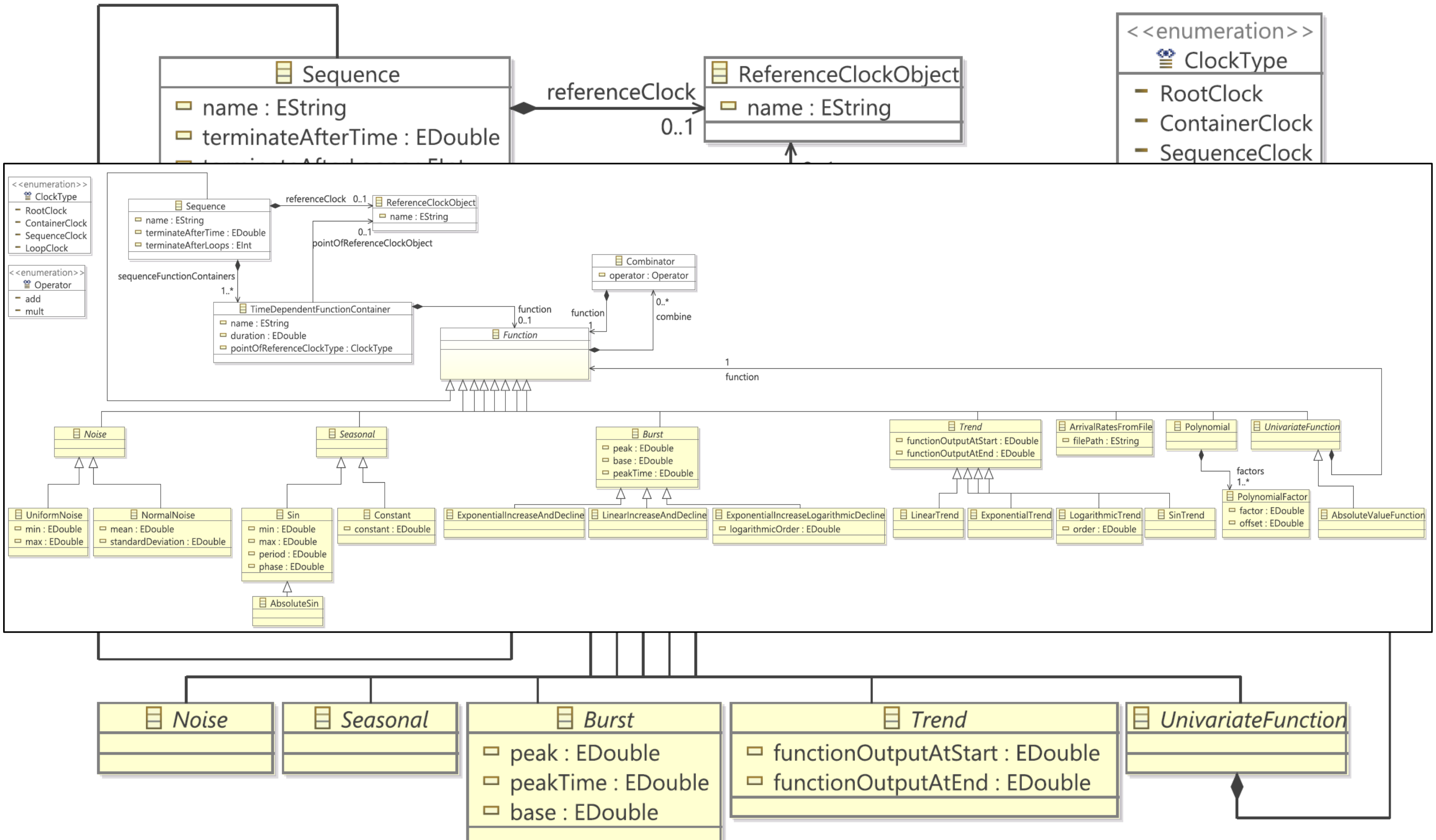
LIMBO



Load Profile Description

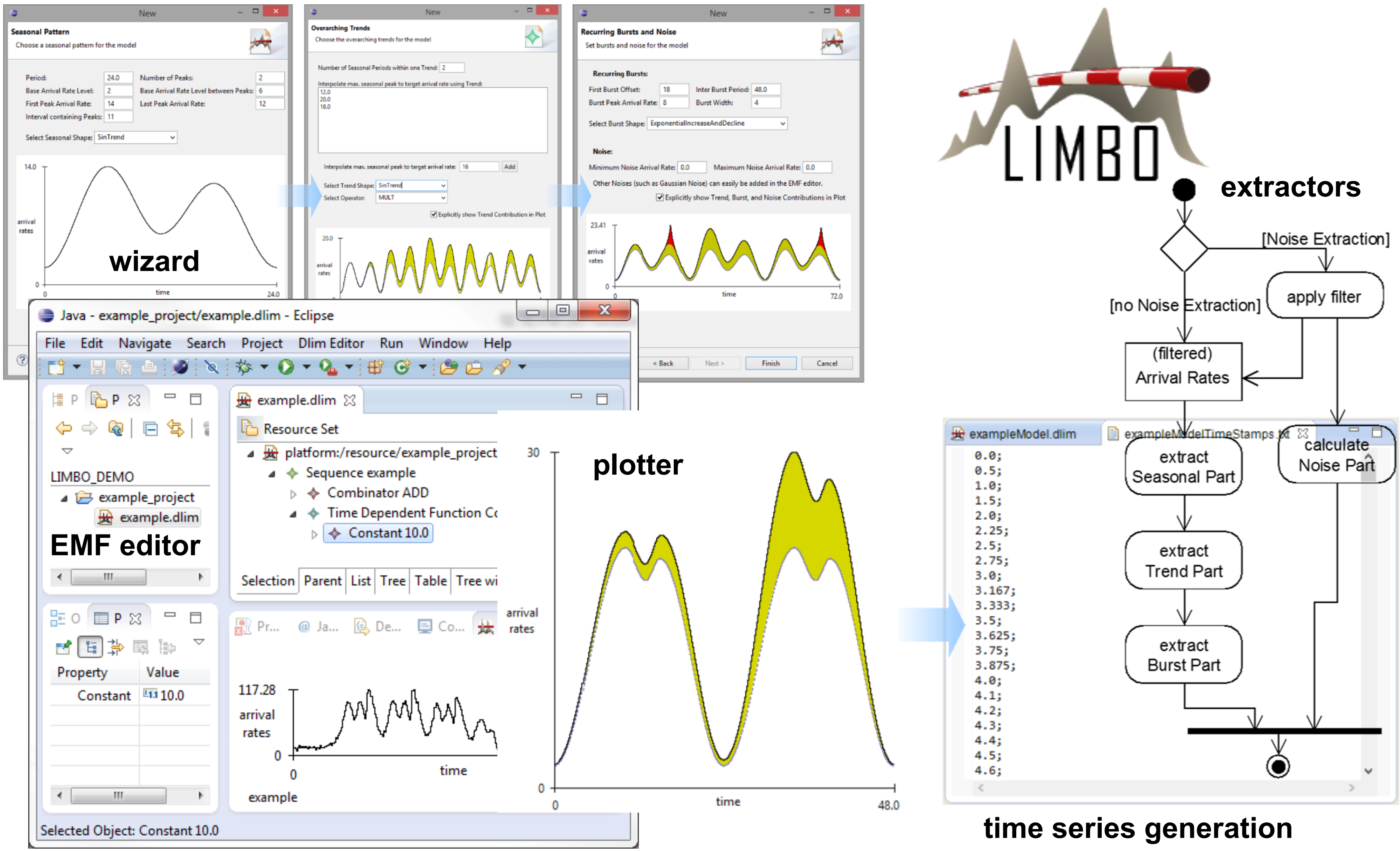


Descartes Load Intensity Model



LIMBO toolkit

http://descartes.tools/limbo



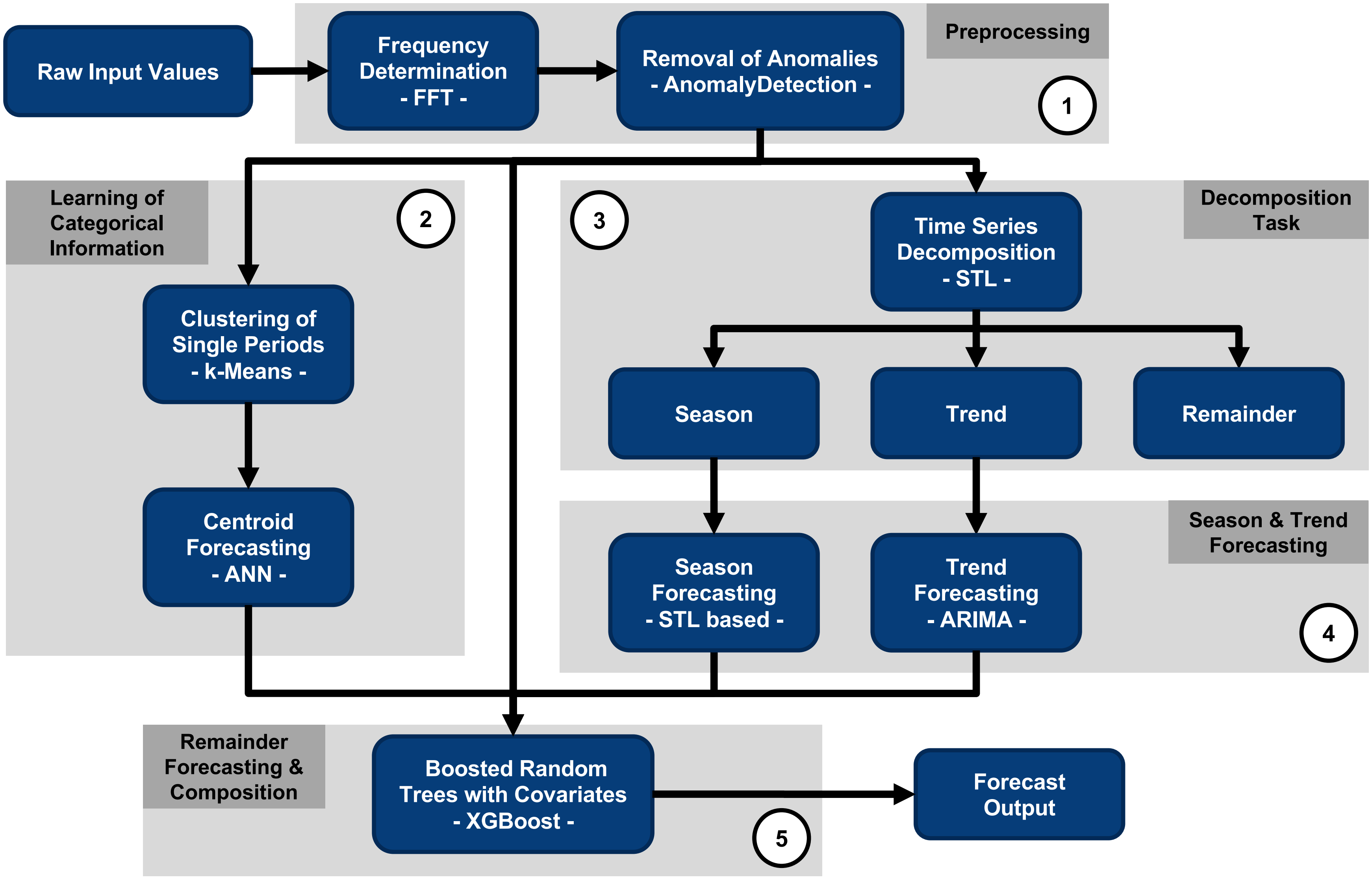
http://descartes.tools/telescope

Forecasting the future workload

TELESCOPE

→ Released in May 2018 as R package on Github ←

Telescope Approach

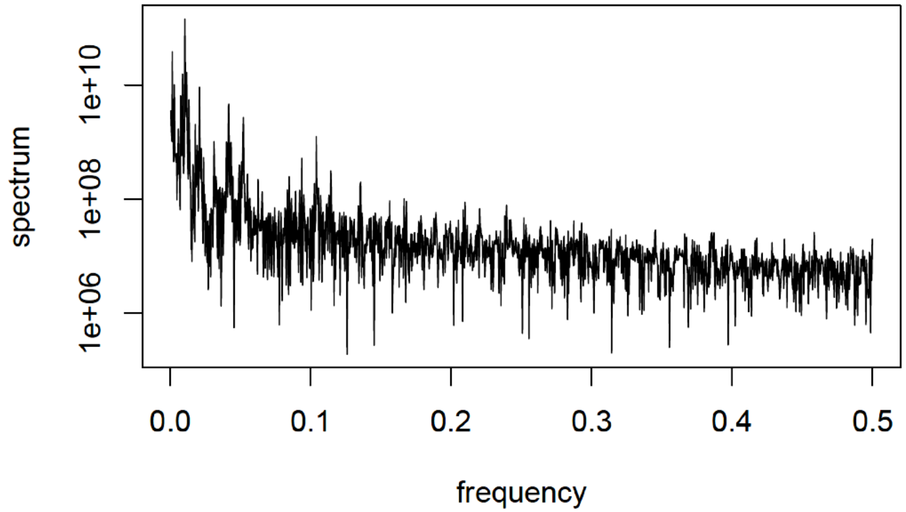


Preprocessing

1

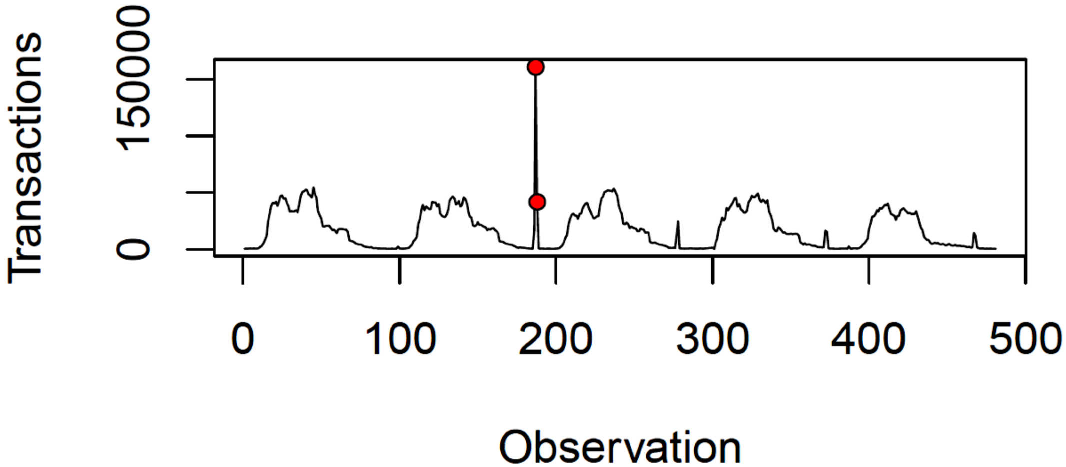
Frequency Estimation:

- Periodograms for rough estimation
- List of common frequencies

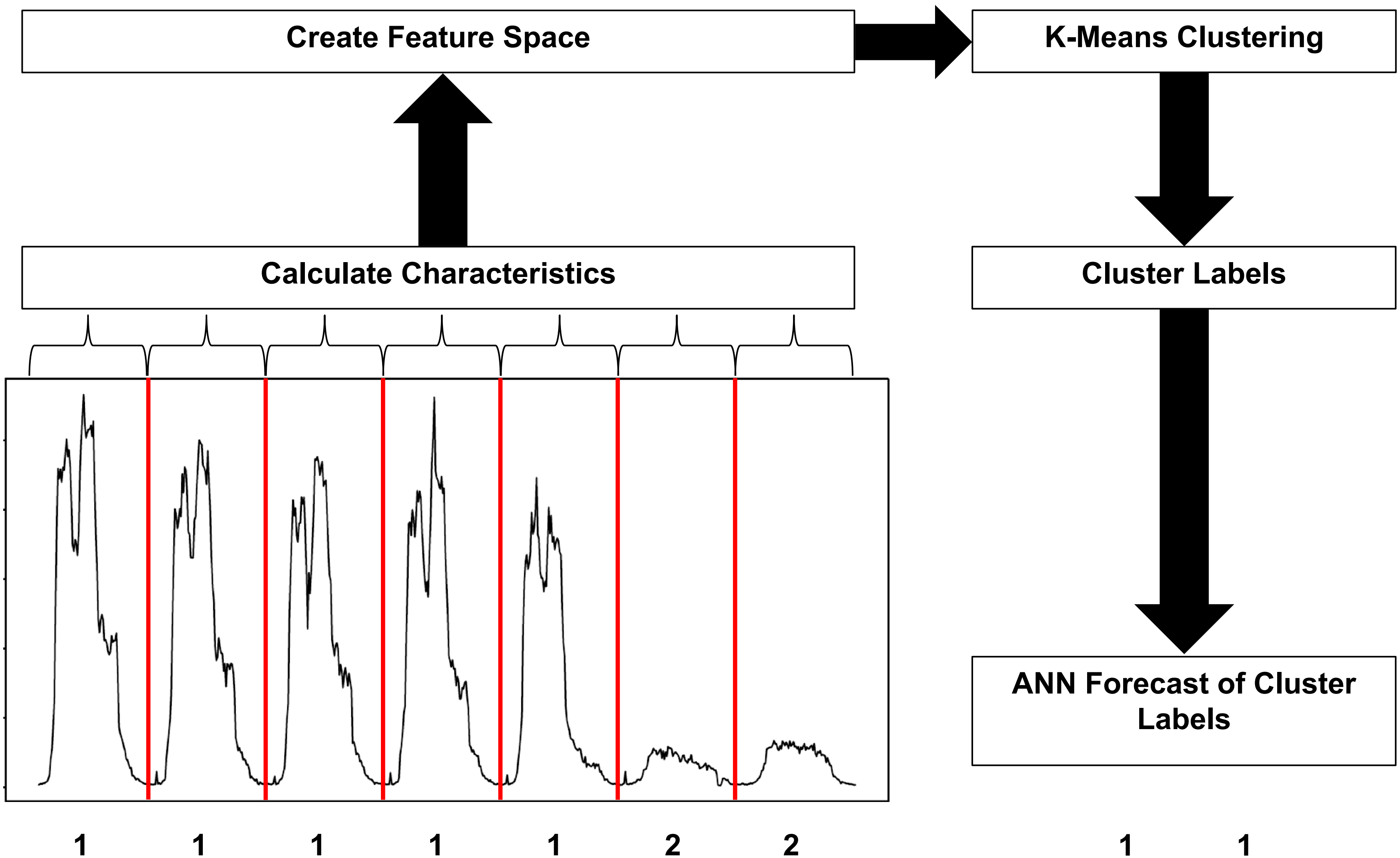


Anomaly Detection:

- Generalized extreme studentized deviate test (ESD) on the remainder
- Replace anomaly by mean of non-anomaly neighbors



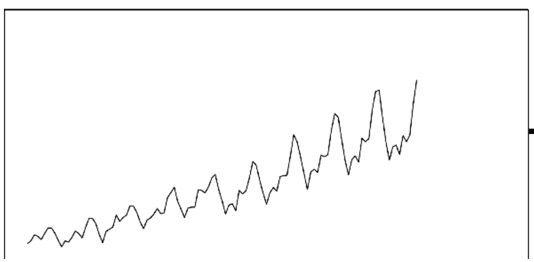
② Learning Categorical Information



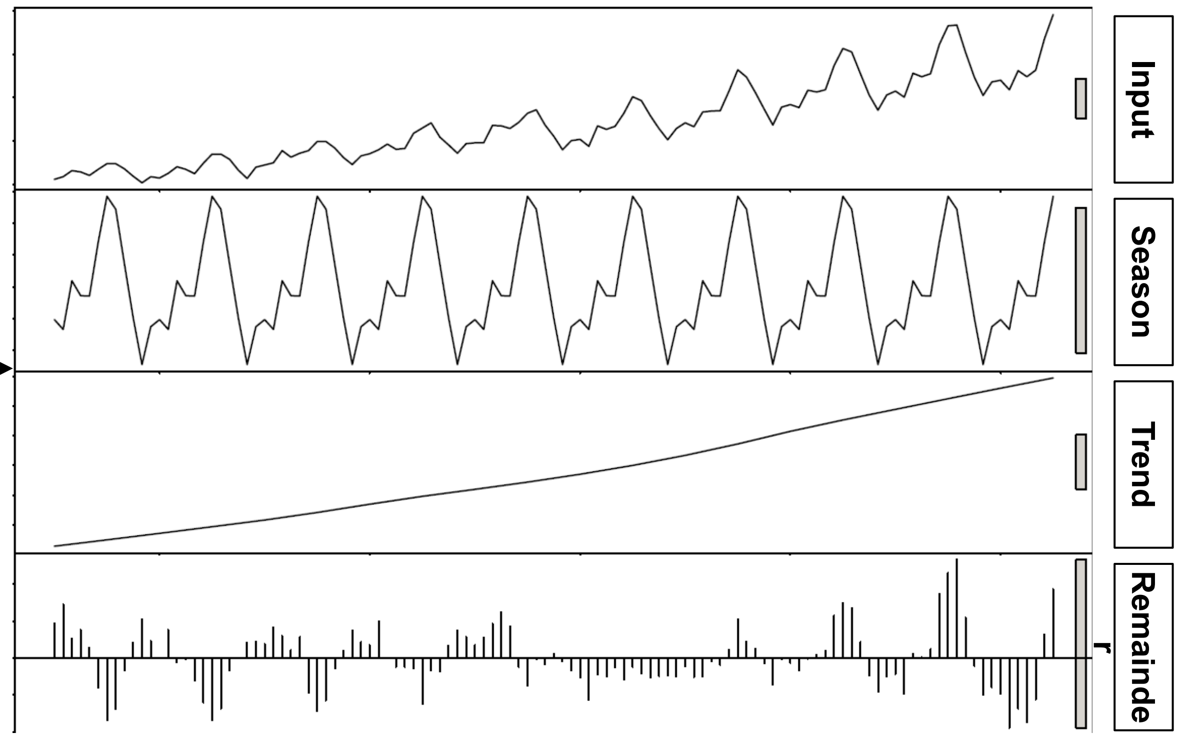
3 4 5

Decomposition & Forecasting

Learning of Categorical Information → **Cluster Label Forecast**



Time Series History



STL Decomposition

3

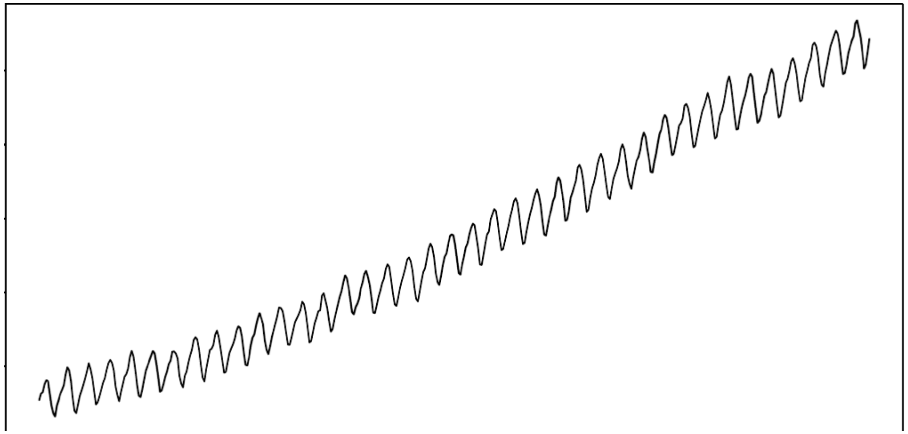
Estimating Decomposition Type

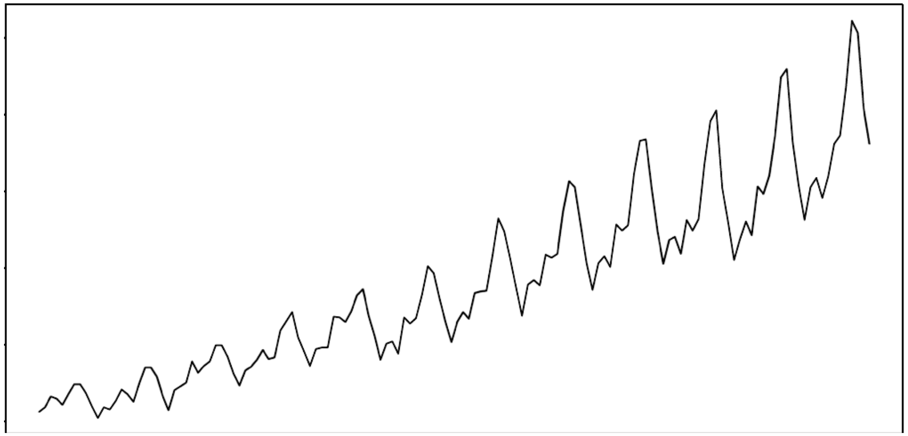
STL once on original and once on logarithmized time series

Calculate:

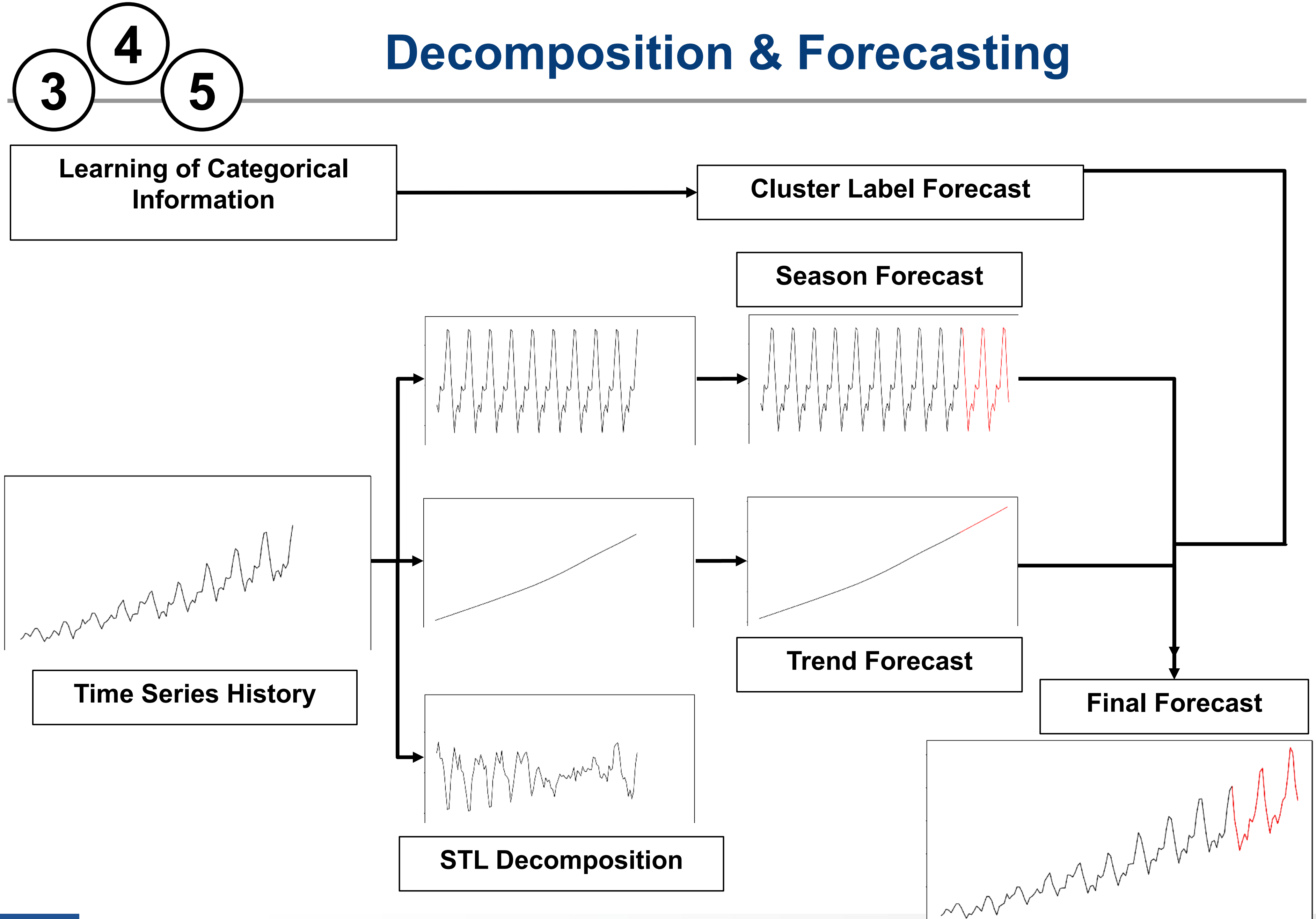
- Sum of squares of the auto-correlation on remainder
- Range between first and third quantile of the remainder
- Sum of squares of the remainder

Majority decision

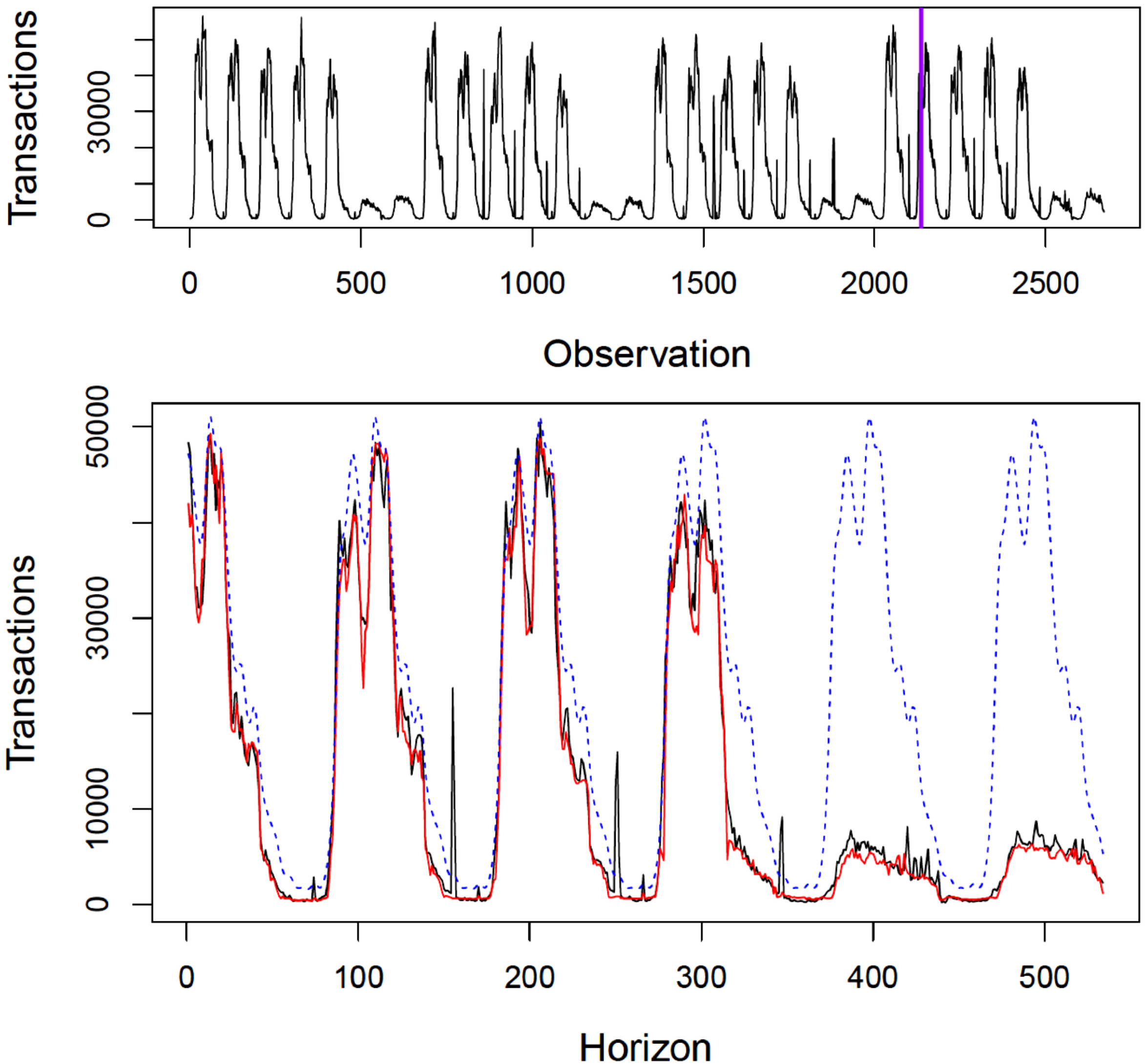




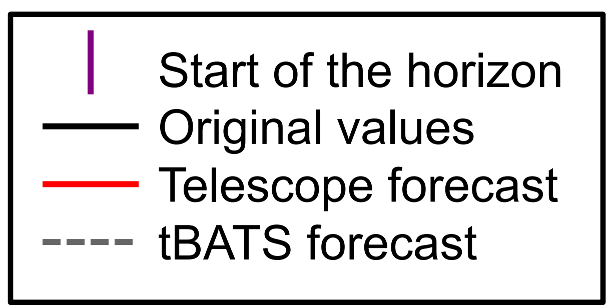
Decomposition & Forecasting



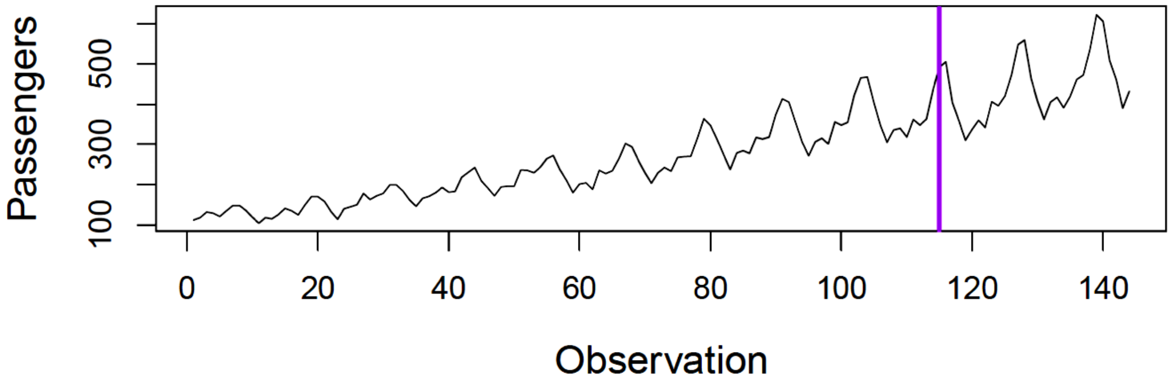
Example: IBM Trace

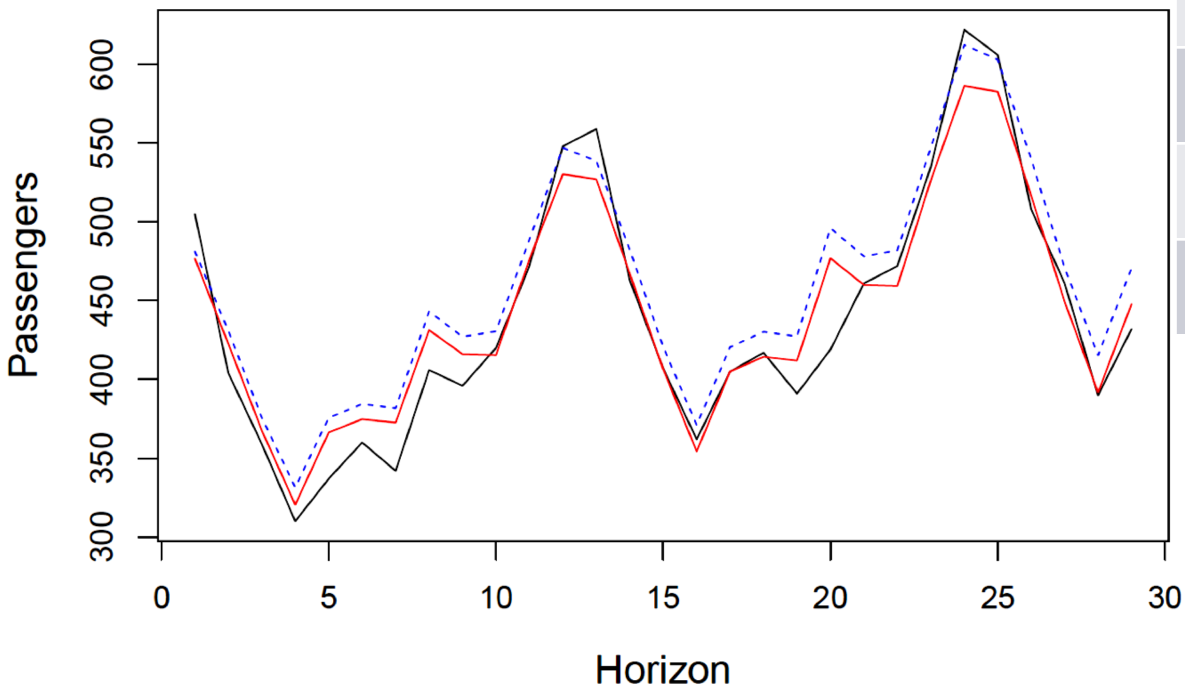


Forecaster	MASE	Time
Telescope	**0.842**	6.248
tBATS	4.547	33.360
SVM	6.557	2.344
XGBoost	7.683	**0.172**
ARIMA	7.828	87.016
ANN	18.678	10.938
ETS	23.389	0.984

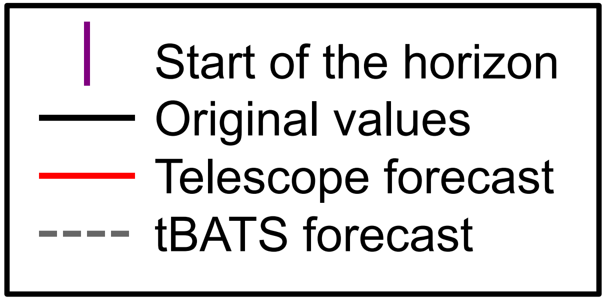


Example: Airline Passengers Trace





Forecaster	MASE	Time
Telescope	**0.353**	1.671
tBATS	0.520	11.641
ARIMA	0.638	3.248
ETS	0.652	2.266
ANN	0.711	0.375
XGBoost	1.261	0.102
SVM	6.758	**0.094**



Measures for 56 Time Series

- High and stable accuracy for multi-step forecasting
- Comparably short time-to-result

Forecaster	Ø MASE	σ MASE	Ø MAPE	Ø Time
Telescope	**1.503**	**1.619**	25.217	9.032
tBATS	1.791	3.112	**25.107**	56.334
ARIMA	2.022	2.405	43.194	177.288
ANN	2.072	3.206	67.176	77.948
XGBoost	2.251	2.017	47.779	**0.167**
ETS	2.638	4.288	81.816	2.184
SVM	5.334	6.254	64.306	24.608

http://descartes.tools/LibReDE

Estimating Resource Demands

LIBREDE

“A ***resource demand*** is the time a unit of work (e.g., request or internal action) spends obtaining service from a resource (e.g., CPU or hard disk) in a system.” S. Spinner 2015

How to quantify resource demands?

Direct Measurement

Requires specialized infrastructure to monitor low-level statistics.

Examples:

- TimerMeter [Kuperberg09]
  + ByCounter [Kuperberg08]
- Brunnert et al. [Brunnert13]
- Magpie [Barham04]

Statistical Estimation

Use of statistical techniques on high-level monitoring statistics.

Examples:

- Linear regression [Kraft09]
- Kalman filtering [Wang12]
- Nonlinear optimization [Kumar09]
- Maximum likelihood estimation [Kraft09]

Why should I use statistical estimation?

Direct measurements infeasible

- Only aggregate resource usage statistics available
- Unaccounted work in system or background threads

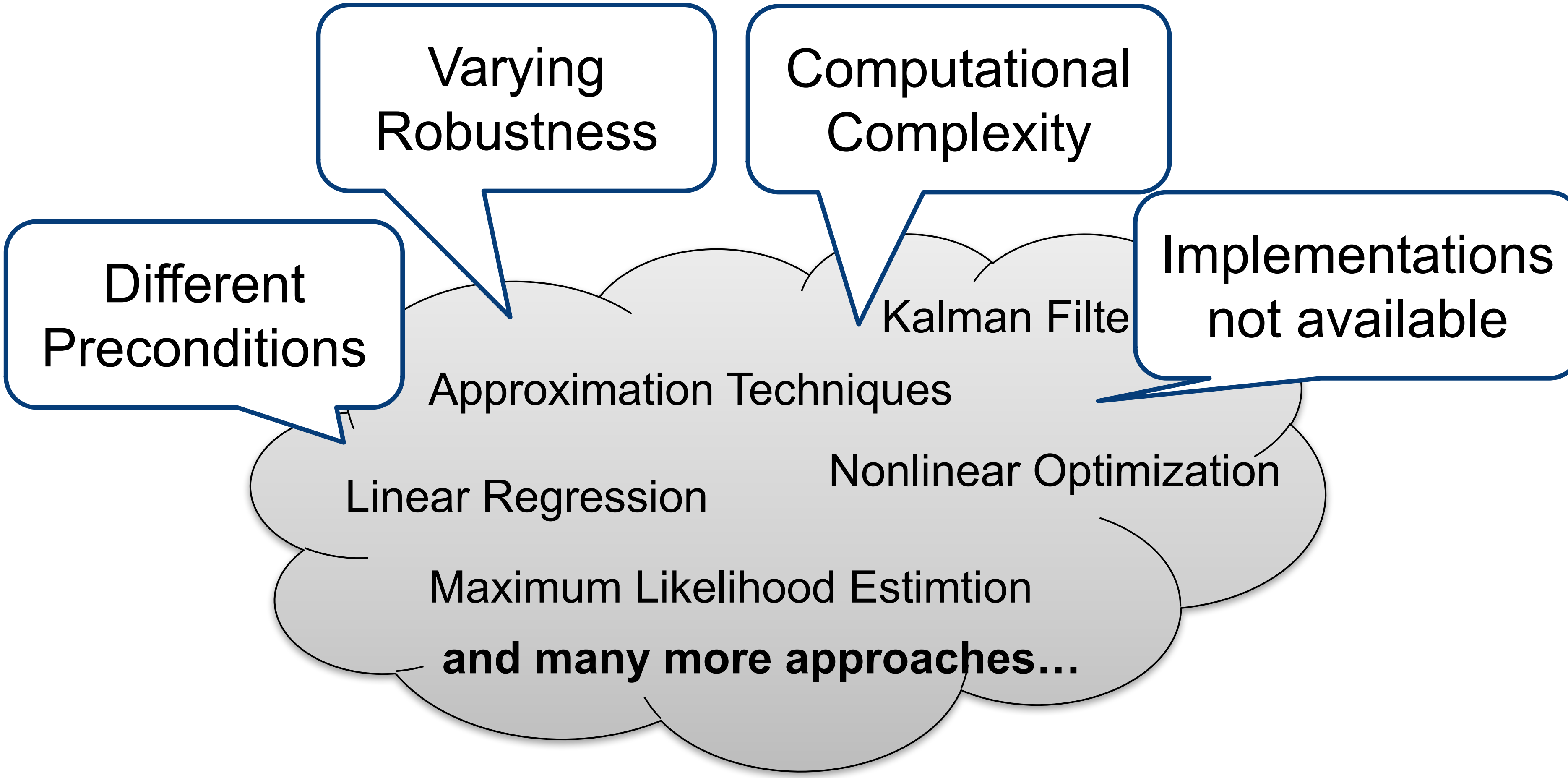
Direct measurements too expensive

- Monitoring of production system
- Heterogeneous software stacks

Coarse-grained models

- Trade-off analysis speed vs. prediction accuracy
- Usage of performance models at system runtime

Challenges



What is the best approach for a given scenario?

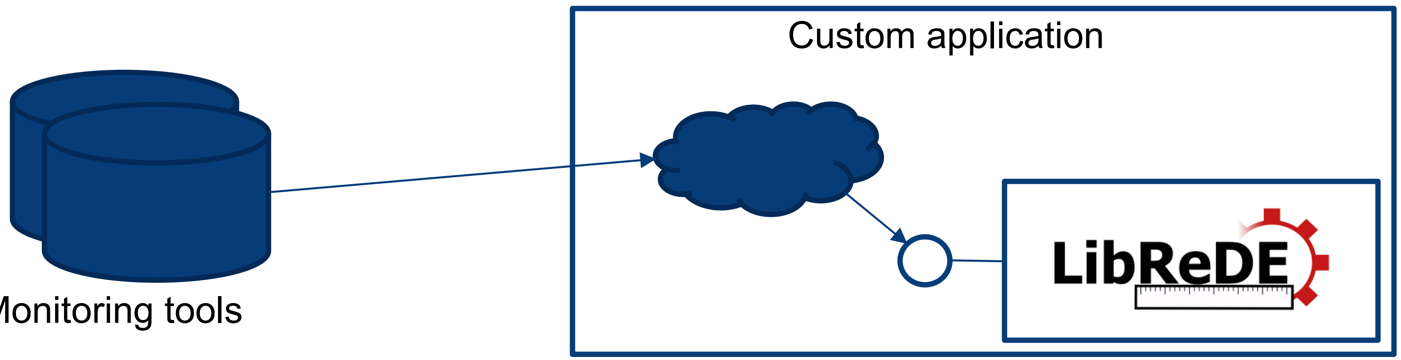
LibReDE Usage

Standalone version for offline analysis

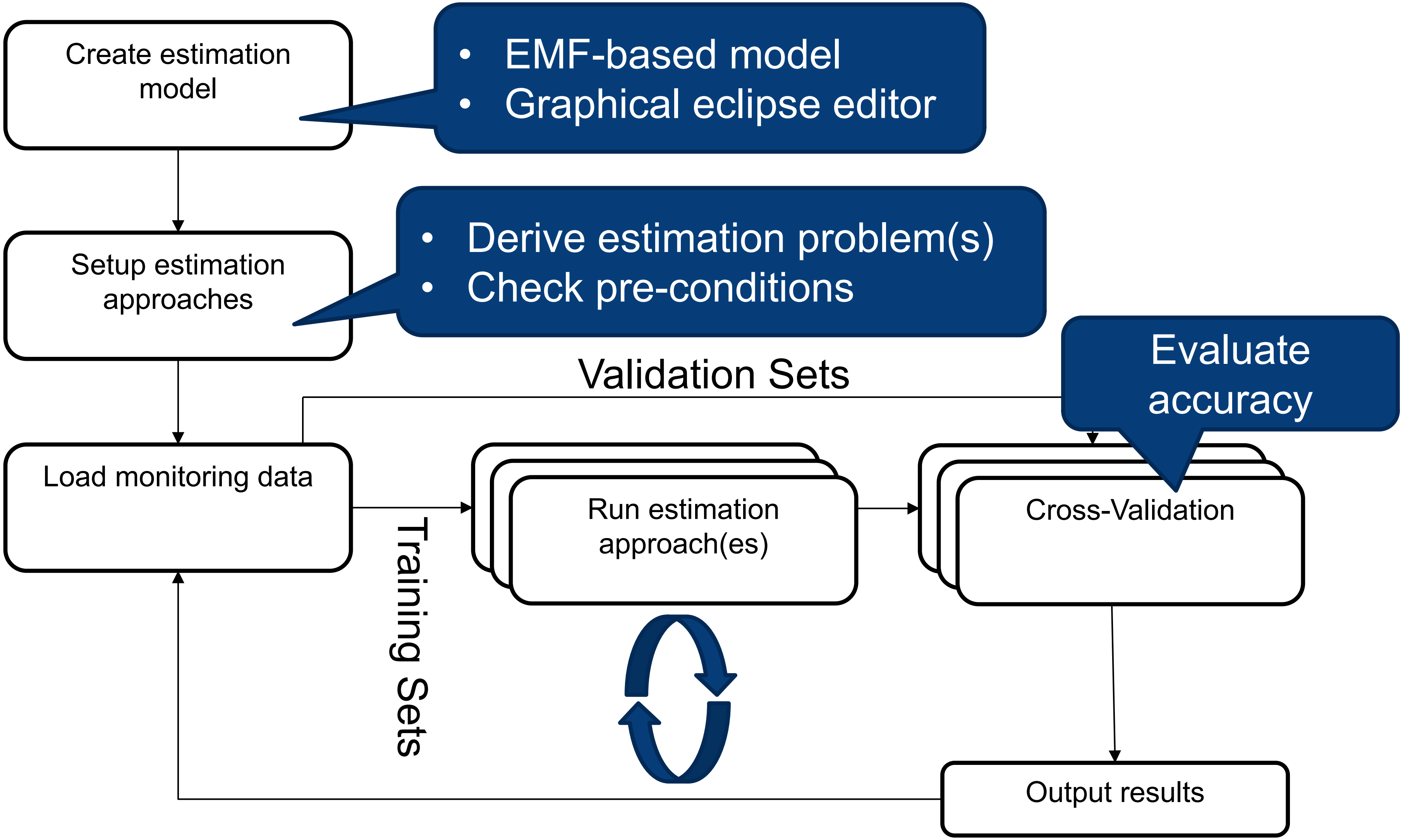
Measurement traces

Estimated Demands

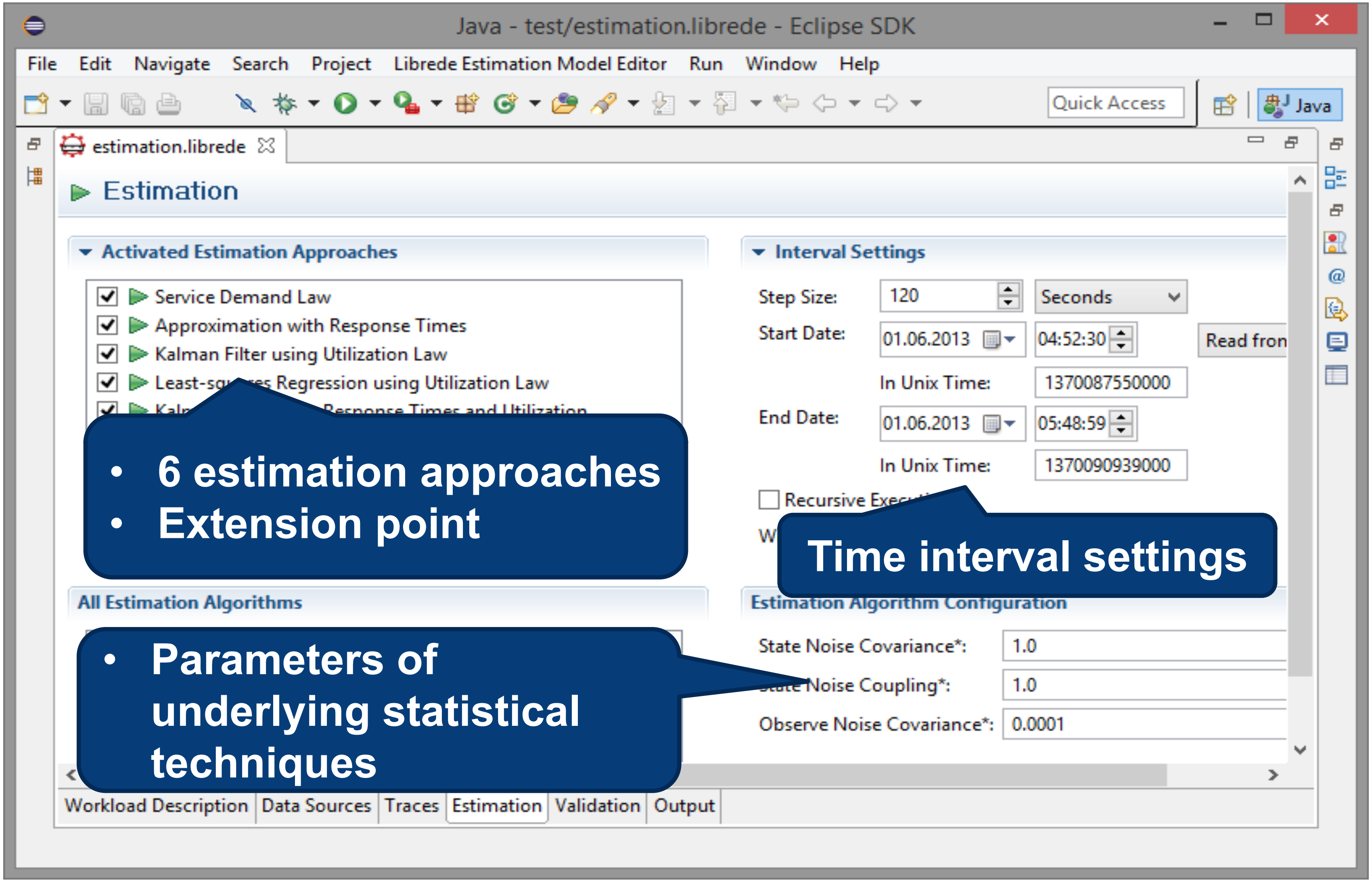
Java library for online analysis



Estimation Process



Estimation



- 6 estimation approaches
- Extension point

Time interval settings

- Parameters of underlying statistical techniques

Key take away points

If you can, build you application from micro-services with restful interfaces

- Flexibility, portability of containers
- Maintainability, reusability

Netflix offers a state of the art software stack

- Netflix Eureka service registry
- Netflix Ribbon service load-balancer with reliability features

Asynchronous communication frameworks in high demand

- E.g. Java NIO implementations:
  JBoss Undertow or Glassfish Grizzly

Thank You!

https://github.com/DescartesResearch/TeaStore

Contact:

nikolas.herbst@uni-wuerzburg.de

https://go.uni-wuerzburg.de/herbst