
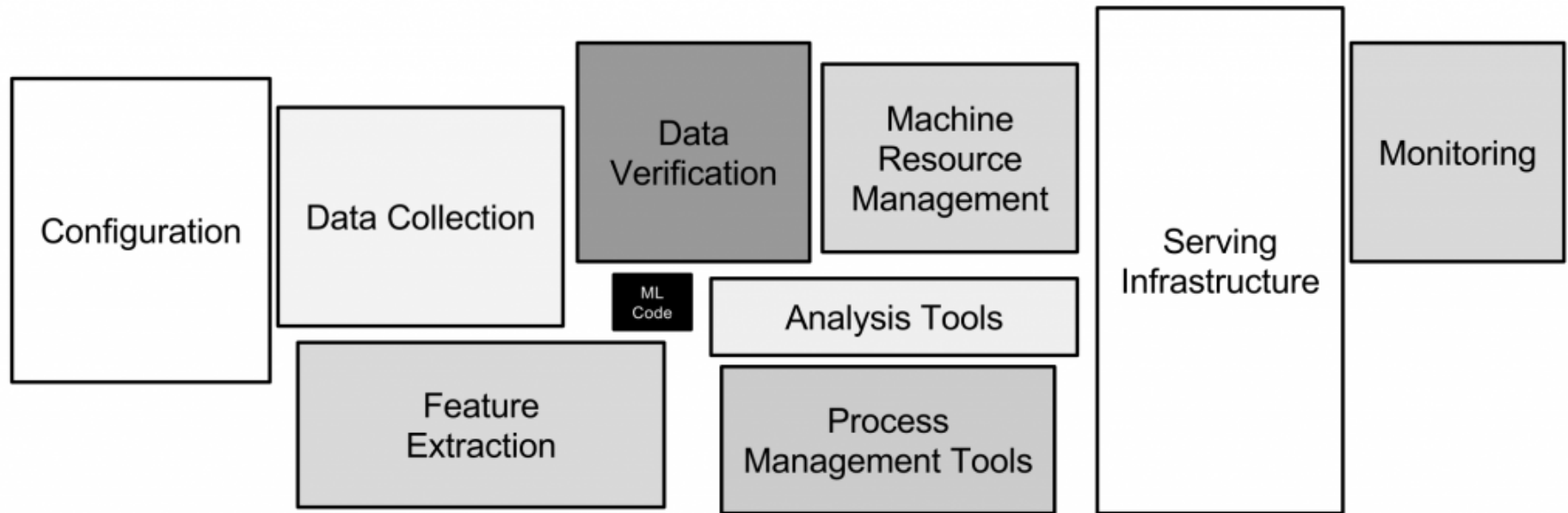


# Создание и архитектура ML-систем

Гущин Александр  
DMIA Production ML  весна 2021

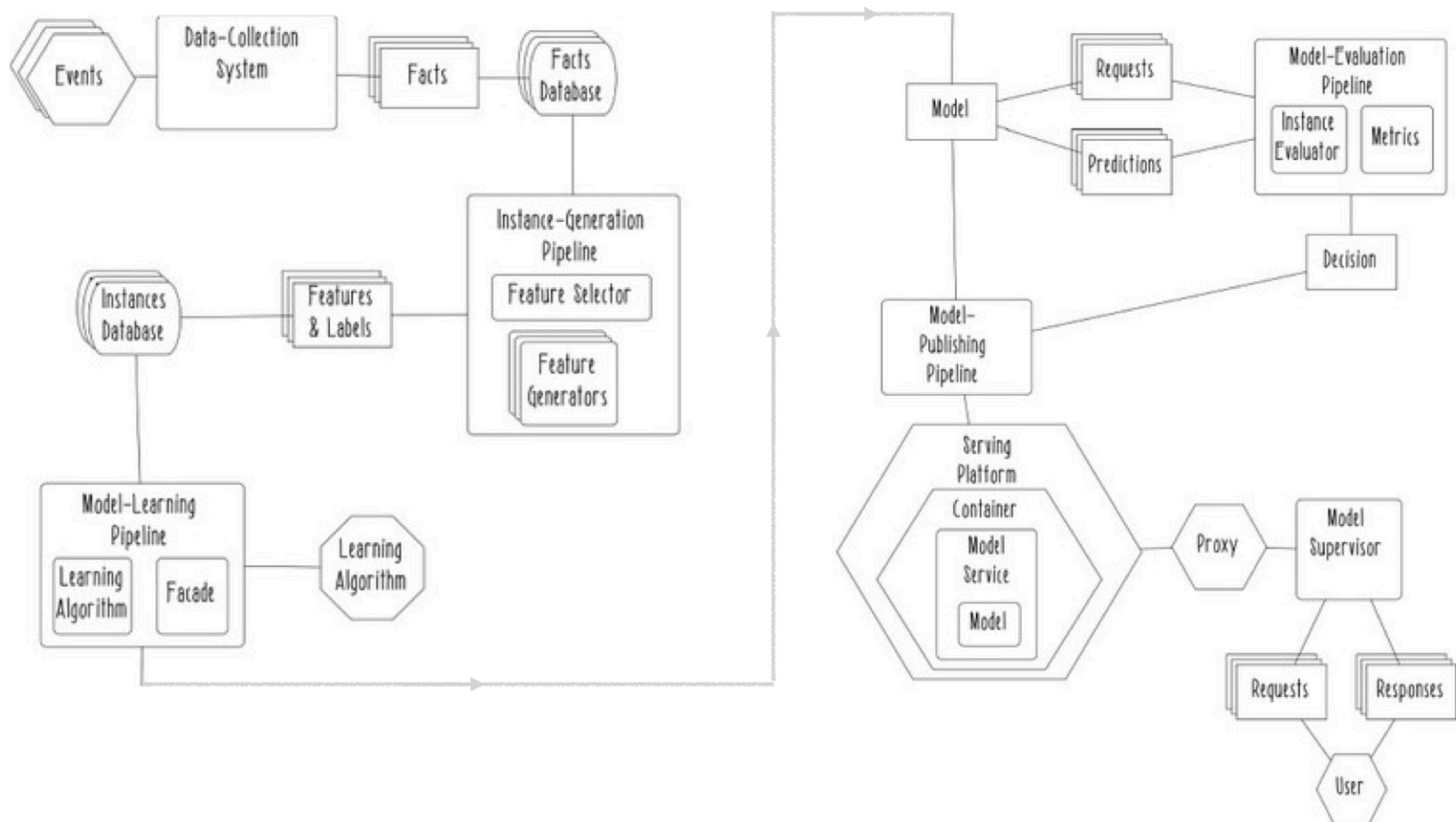
# Архитектура ML решений



Все это вместе образует **Систему**.

**Архитектура** - это устройство таких систем.

["Hidden Technical Debt in Machine Learning Systems" paper](#)



## Machine Learning Systems by Jeff Smith

## Зачем нам думать об этом?

0. Архитектура описывает конечный предмет, который мы должны построить
1. Выбор архитектуры диктует слабые и сильные стороны решения и пути его модификации в будущем
2. Плохая архитектура породит технический долг и приведет к проблемам в её работе, поддержке, и доработке
3. Для эффективной работы в команде нужно общее понимание, как именно устроена ваша система

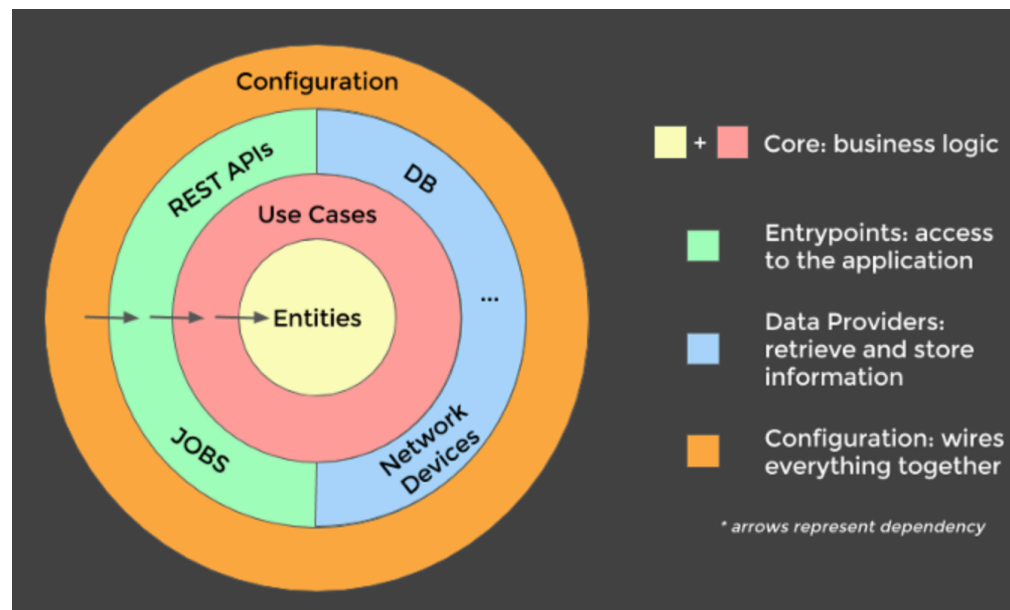
**Технический долг** — накопленные в программном коде или архитектуре проблемы, связанные с пренебрежением к качеству при разработке программного обеспечения и вызывающие дополнительные затраты труда в будущем.

**ML-системы** имеют дополнительные “возможности” для создания технического долга. ML Engineer сталкивается с этим **в первую очередь**.

Например — unstable data dependencies, feedback loops, glue code, pipeline jungles, dead experimental codepaths, fixed thresholds in dynamic systems, entanglement (CACE principle), etc.

[“Hidden Technical Debt in Machine Learning Systems” paper](#)

Для начала, MLE приобретают все проблемы & best practices из Software engineering



- Clean Architecture: A Craftsman's Guide to Software Structure and Design, Robert C. Martin; [Short video about it](#)
- A Philosophy of Software Design (2018), John Ousterhout
- Figure from <https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/>

# Создание ML систем

Best Practices for ML Engineering by Martin Zinkevich

To make great products: **do machine learning like the great engineer you are, not like the great machine learning expert you aren't.** 😂

- Make sure your pipeline is solid end to end.
- Start with a reasonable objective.
- Add common-sense features in a simple way.
- Make sure that your pipeline stays solid.

ML system development is divided in the following phases:

- Before Machine Learning
- ML Phase I: Your First Pipeline
- ML Phase II: Feature Engineering
- ML Phase III: Slowed Growth, Optimization Refinement, and Complex Models

<https://developers.google.com/machine-learning/guides/rules-of-ml>



# От требований к архитектуре

- Требования к расчету фичей
  - на лету или предварительно?
  - с помощью каких программных средств?
- Требования к обучению
  - как часто?
  - на каких ресурсах?
- Требования к выдаче прогнозов
  - каким способом?
  - как быстро?
  - минимальное допустимое качество?
- Tradeoff скорость/надежность разработки
- Требования к автономности
- Требования к доступной кастомизации
- Требования к обработке приватных данных

И прочее, прочее, прочее

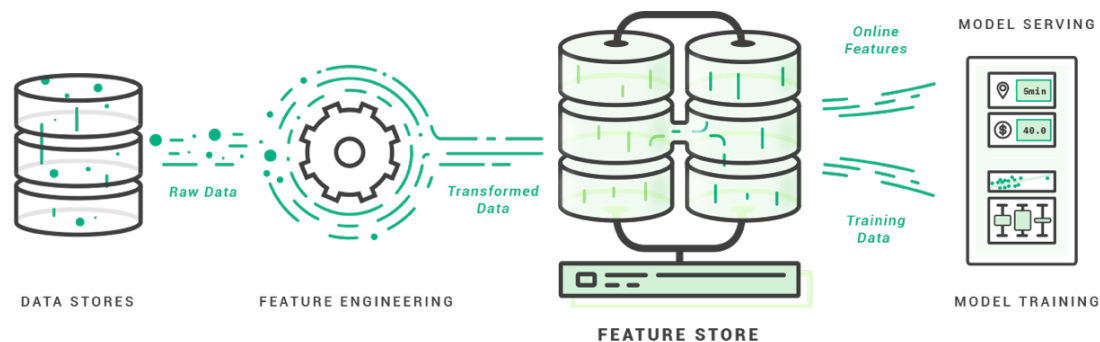
# Препроцессинг фичей

Обычно есть два подхода:

- Считаем фичи в режиме реального времени
- Требуется рассчитывать некоторые фичи заранее

Использование имеющихся источников порождает задачи интеграции и накладывает на них ограничения - например, по времени ответа.

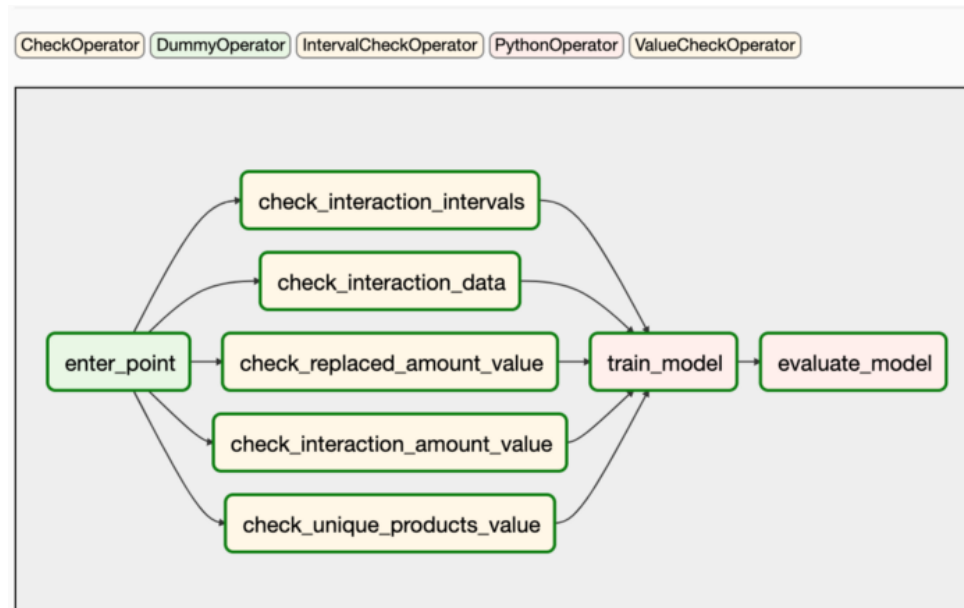
Feature Store - служит для сохранения фичей. Используется и во время обучения, и во время расчета прогнозов



Например, <https://docs.hopsworks.ai/latest/>

# Обучение

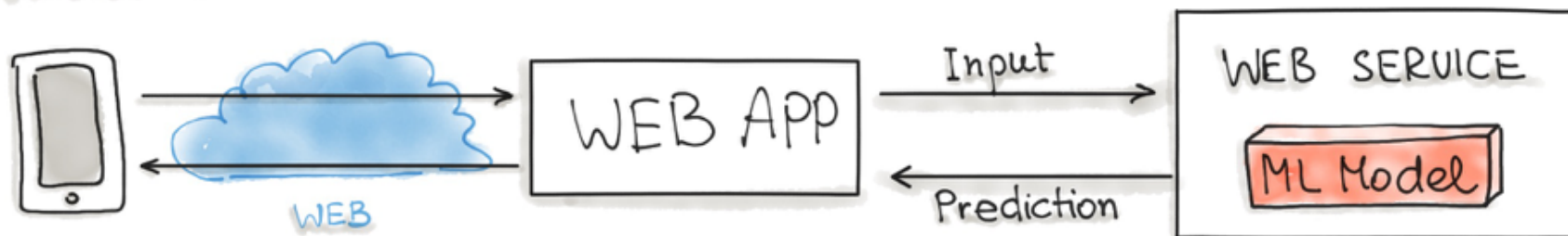
- Наиболее распространено переобучение по расписанию - Запускаем код с помощью cronjob, планировщиков задач <— **практика на нашем курсе**
- Другой вариант - обучение на лету (например, с помощью Kafka и Tensorflow)



Например, <https://towardsdatascience.com/machine-learning-in-production-using-apache-airflow-91d25a4d8152>

# Выдача прогнозов

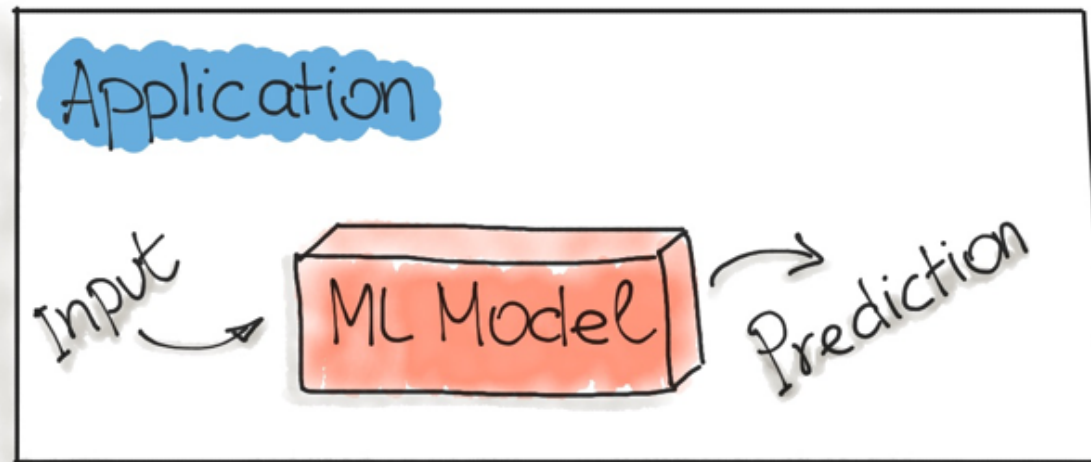
## MODEL-as-SERVICE



- Просто: Пишем REST API обертку на питоне <— **практика на нашем курсе**
- Быстро: Переписываем код на Go/C++, используем gRPC

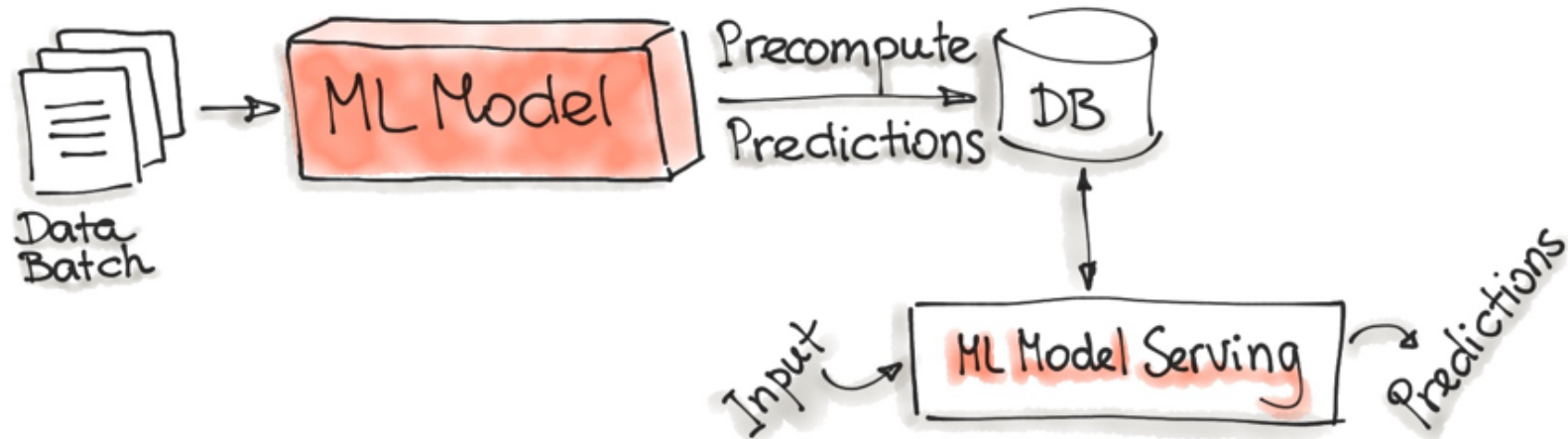
[Figure source](#)

# MODEL-as-DEPENDENCY



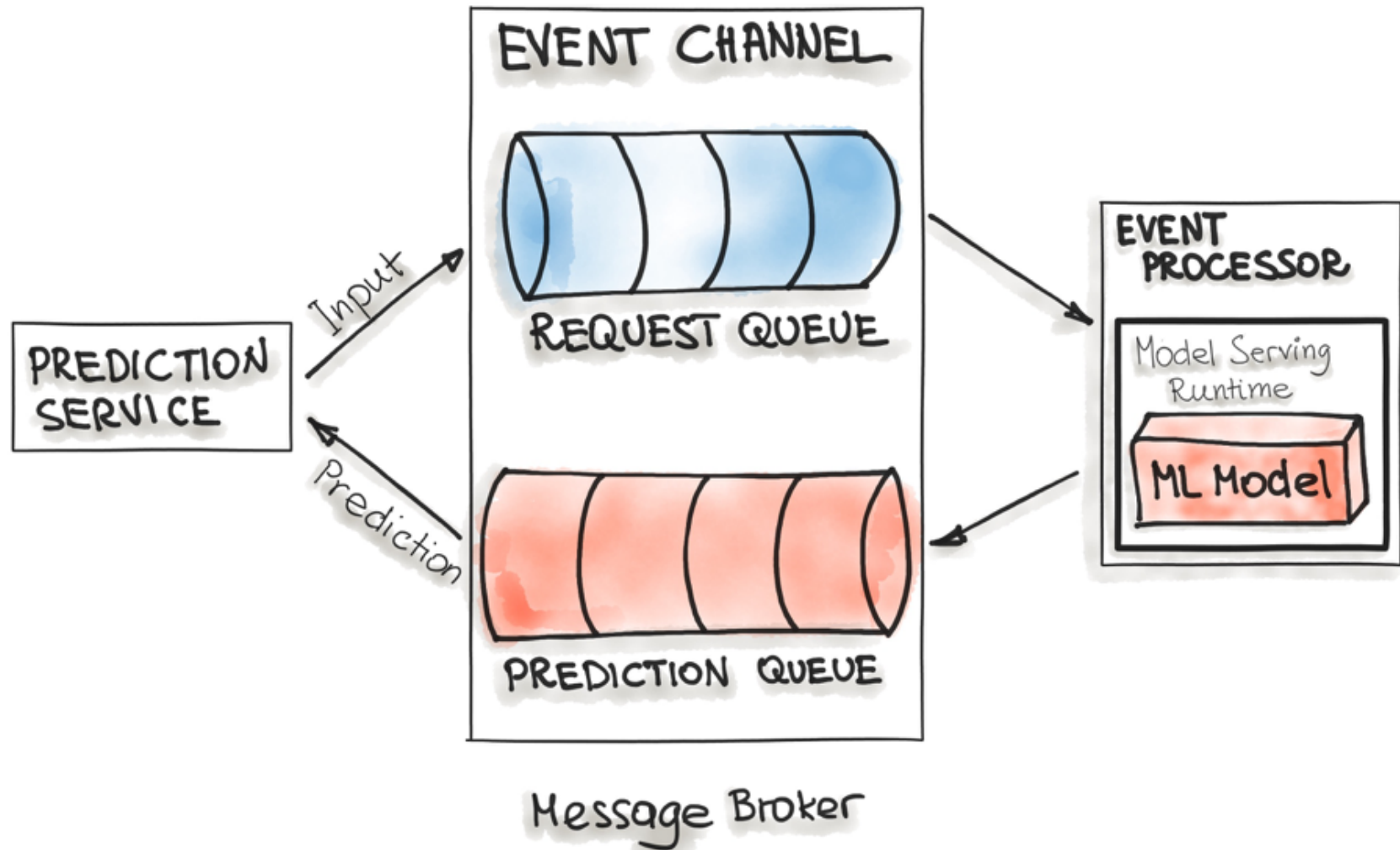
- В приложениях с монолитной архитектурой
- В приложениях на телефон

# PRECOMPUTE SERVING PATTERN





# MODEL-ON-DEMAND





# Примеры

- Пример задачи: практика на нашем курсе
- Архитектура
  1. Данные обновляются ежедневно и доступны для скачивания из Object-storage
  2. Обучение происходит по расписанию
  3. Прогноз в режиме реального времени с помощью запросов по REST API

## Пример 2

- Требование: периодичность ответного действия на прогноз фиксирована.
- Примеры задач
  - ежедневный прогноз LTV пользователей для предотвращения оттока
  - ежечасный прогноз погоды
  - ежечасный прогноз потребления электроэнергии
- Архитектура
  1. Данные собираются из различных источников и складываются в хранилище, обновляются постоянно (DWH)
  2. Обучение по расписанию - скачиваем свежие данные (SQL), готовим датасет, обучаем модель
  3. Прогноз по расписанию, загружаем результат в базу данных

## Пример 3

- Требования: big data, сервисы с большой нагрузкой, где требуется высокая скорость ответа
- Примеры задач
  - карты - время в пути из точки А в точку Б
  - рекламная крутилка
  - классификация спама в почте
- Архитектура
  1. Данные собираются из различных источников (например, HDFS), фичи рассчитываются (например, Spark) и складываются в Feature Store
  2. Обучение по расписанию - скачиваем свежие данные, готовим датасет, обучаем модель
  3. Прогноз в режиме реального времени с помощью запросов по gRPC, код может быть переписан на компилируемых языках

# Когда ML - дополнение к имеющимся сервисам, а не центральная часть

Часто ML-системы возникают не на пустом месте, а поверх уже существующей инфраструктуры:

1. Уже есть Data Warehouse
2. Уже есть бэкенд и устоявшиеся подходы к разработке

В таком случае типично использование имеющихся инструментов и подходов. Пример - использование Airflow для запуска пайплайнов по обучению моделей, когда он используется в компании для ETL. Или деплой на AWS Sagemaker/OpenShift/Heroku, когда они используются разработчиками, и так далее.

Это может быть как плохо (бывают неудобные инструменты), так и хорошо (не нужно возводить все с нуля).

# Саммари

1. Архитектура - это описание и устройство программных систем
2. Технический долг — накопленные в программном коде или архитектуре проблемы, связанные с пренебрежением к качеству при разработке программного обеспечения и вызывающие дополнительные затраты труда в будущем.
3. ML системы имеют свойство накапливать особенный технический долг, который требует особенной работы по его выплате (помимо обычного техдолга разработки) 😄
4. Начинать разговор об архитектуре удобно с набора требований к системе. Одни из основных ML-специфичных - это требования к расчету фичей, к обучению моделей и к выдаче прогнозов.