

KAPITEL 1: EINLEITUNG

LERNZIELE

- Unterschied zwischen Event-basierten Ansätzen und Nachrichten-basierten Ansätzen erklären
- Architektur-Begriffe voneinander abgrenzen und vergleichen
- Batch- und Stream-Processing abgrenzen
- Cloud- und Edge-Processing erklären

1.1 MOTIVATION

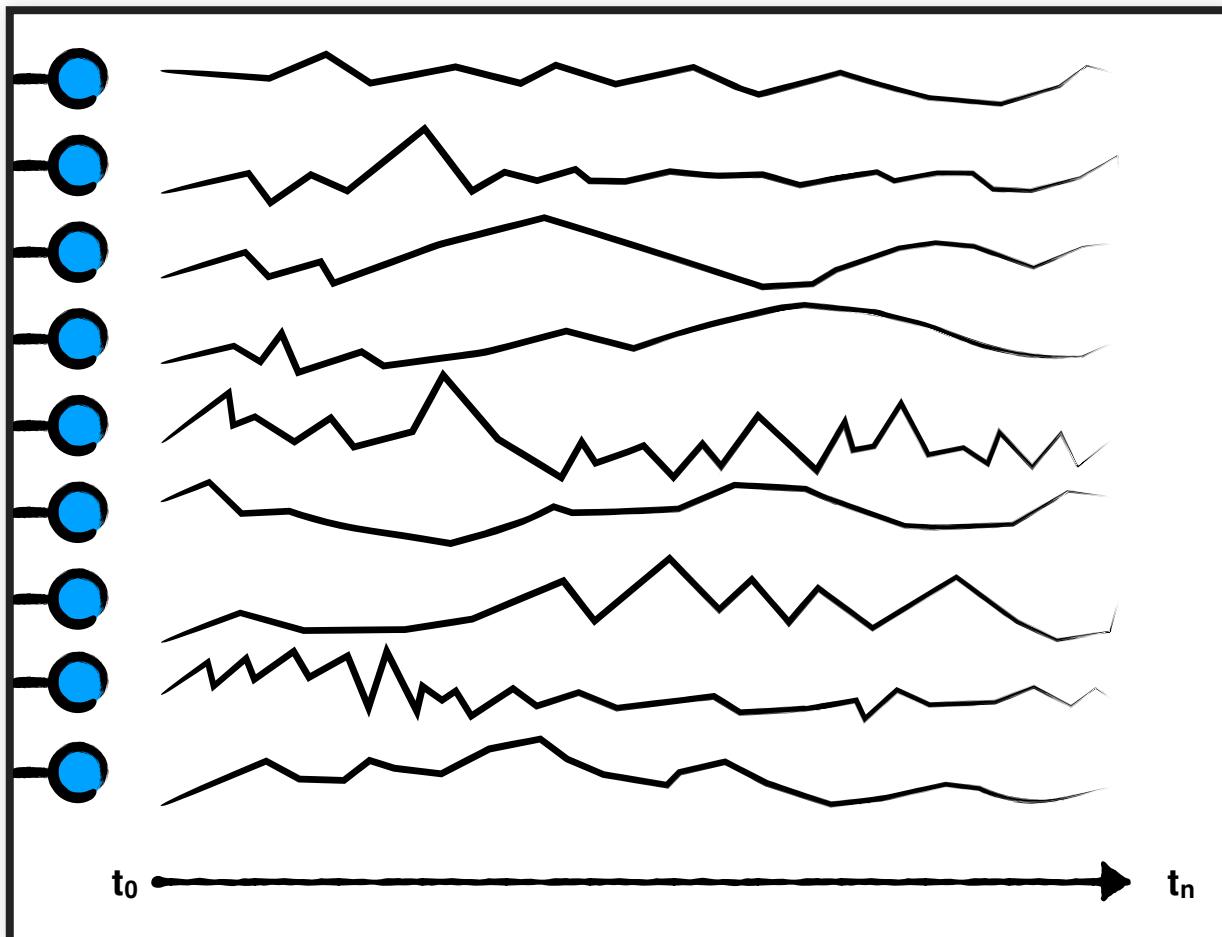


Wozu sind ereignisbasierte Dateninfrastrukturen notwendig?

Folgend ein Beispiel zur Erhebung von Messwerten, wie Stromverbrauchsdaten an verschiedenen Standorten

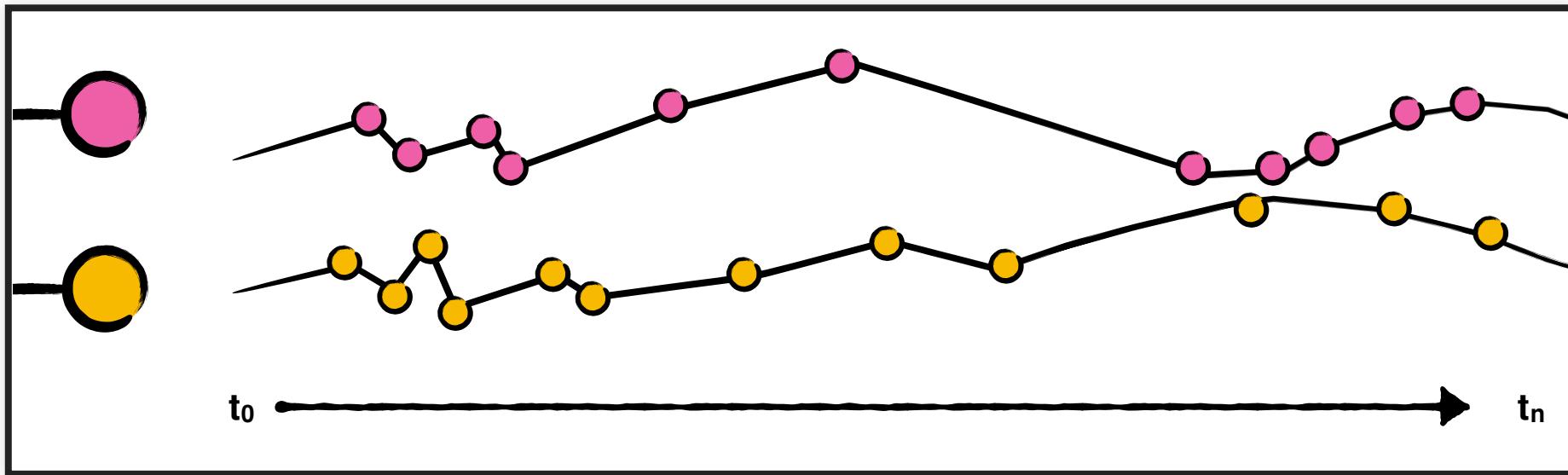


Verschiedene voneinander getrennte Standorte mit eigenständigen Installationen, welche Messwerte erzeugen - z.B. Verbrauchsdaten von einem Smart Meter.

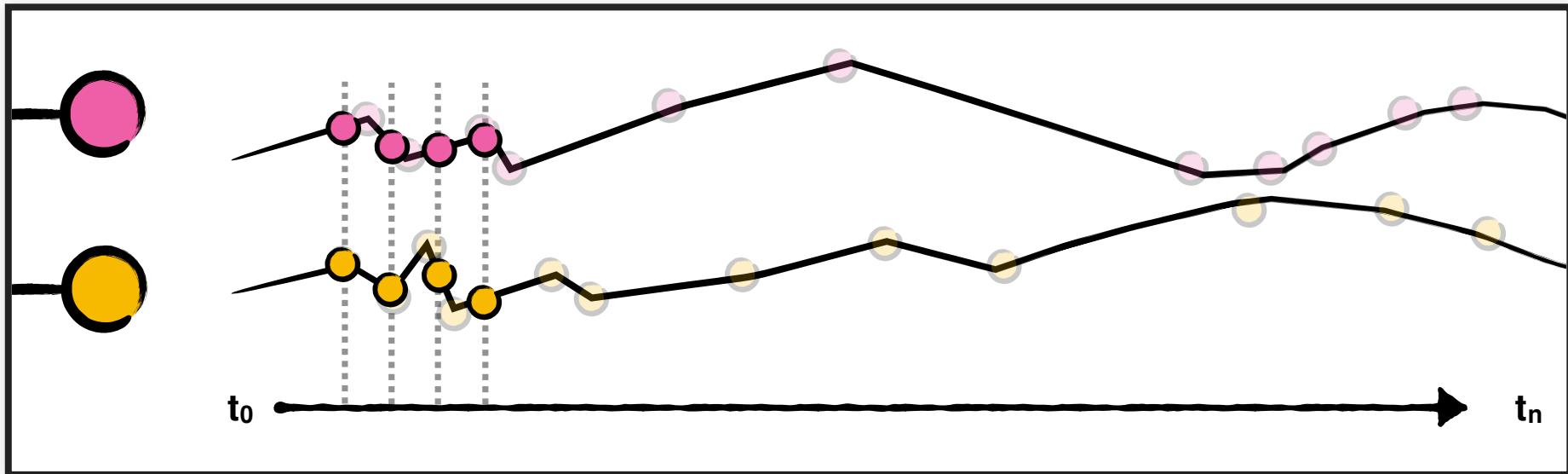


Aus den Installationen an den verschiedenen Standorten entstehen kontinuierliche Zeitreihen mit Messwerten.

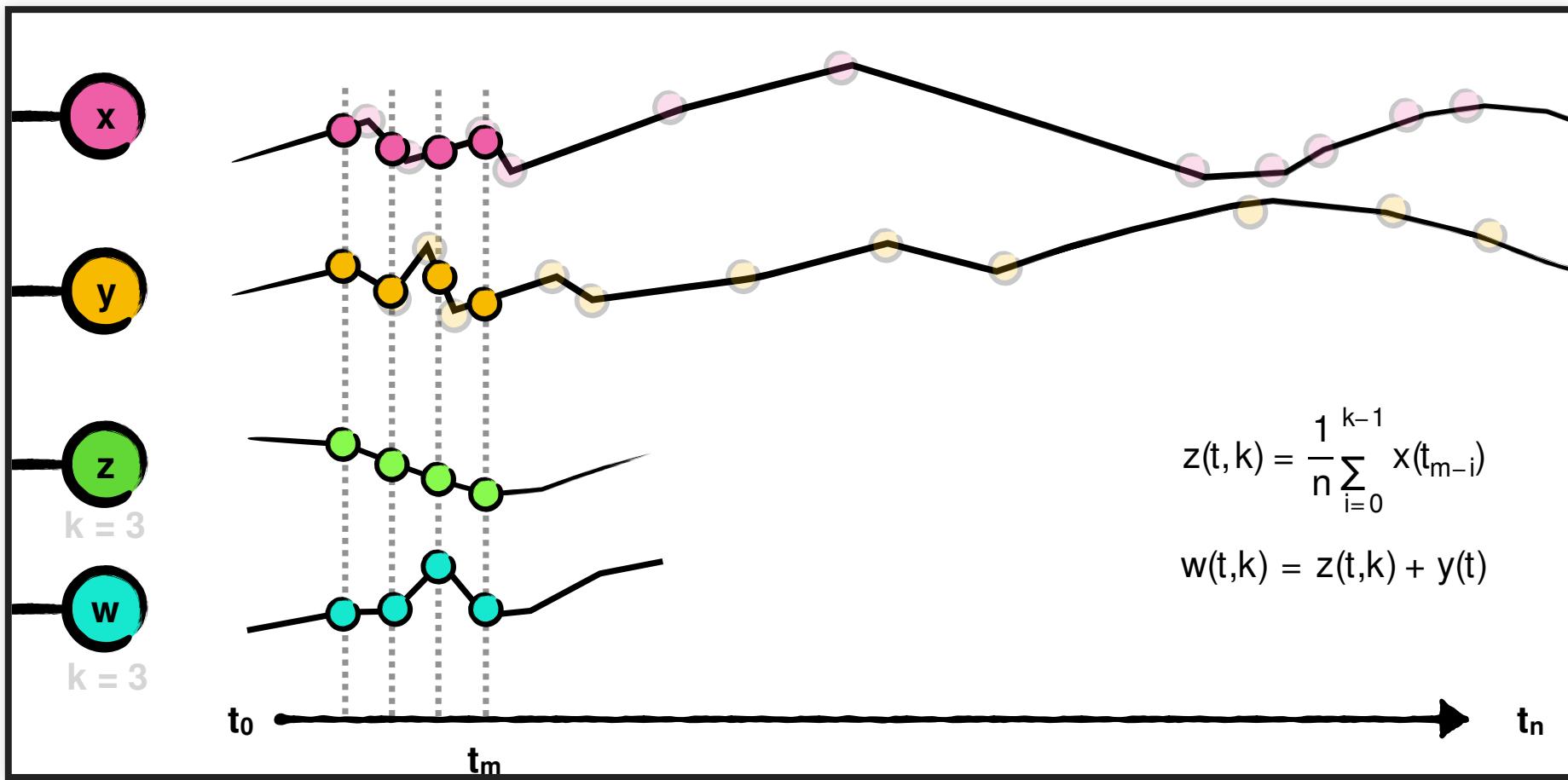
Diese müssten erfasst und gegebenfalls verarbeitet sowie gespeichert werden.



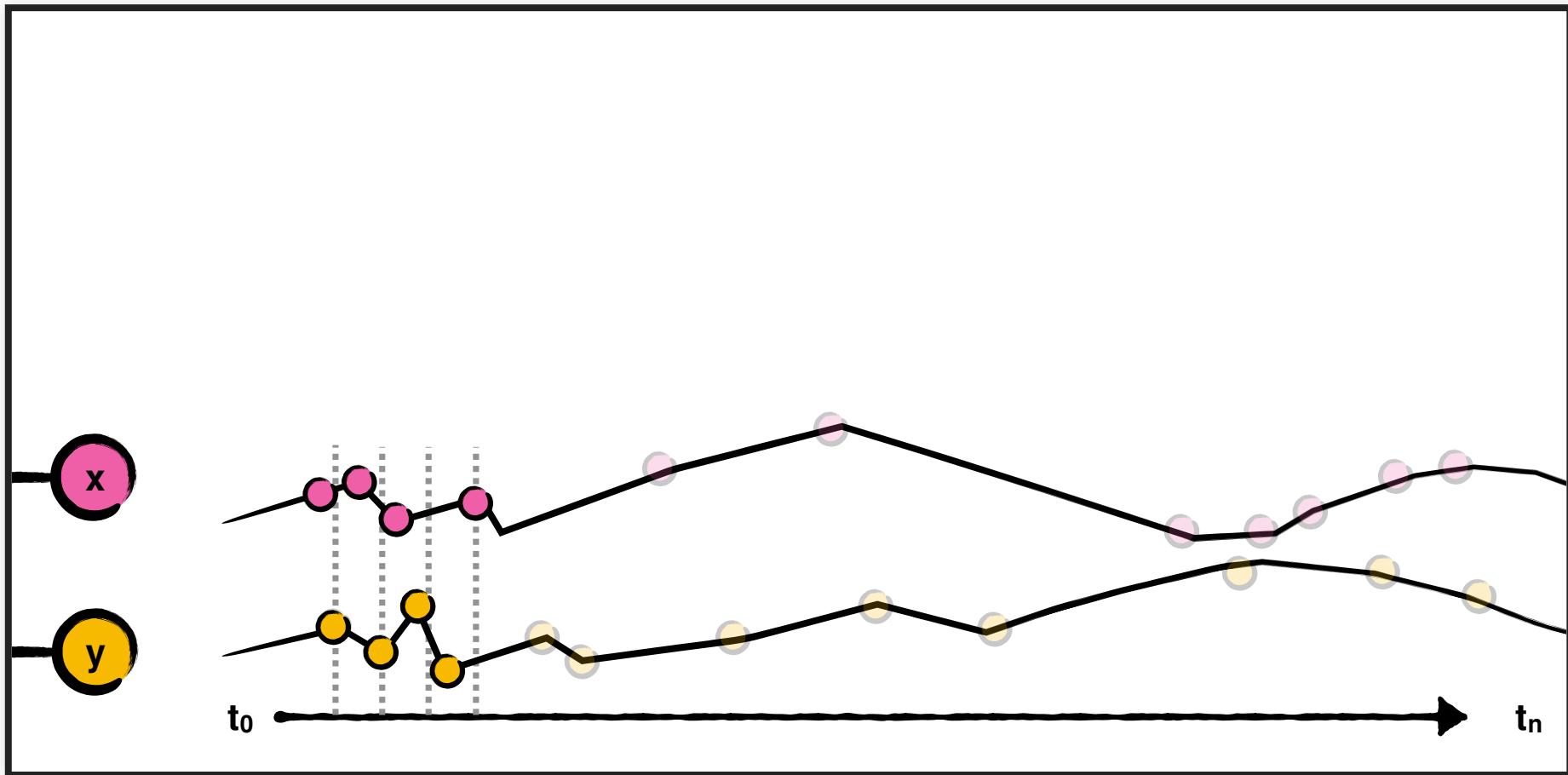
Jede Messreihe ist über ihre eigenen Messdaten repräsentiert und Messpunkte entstehen basierend auf der jeweiligen Taktung.



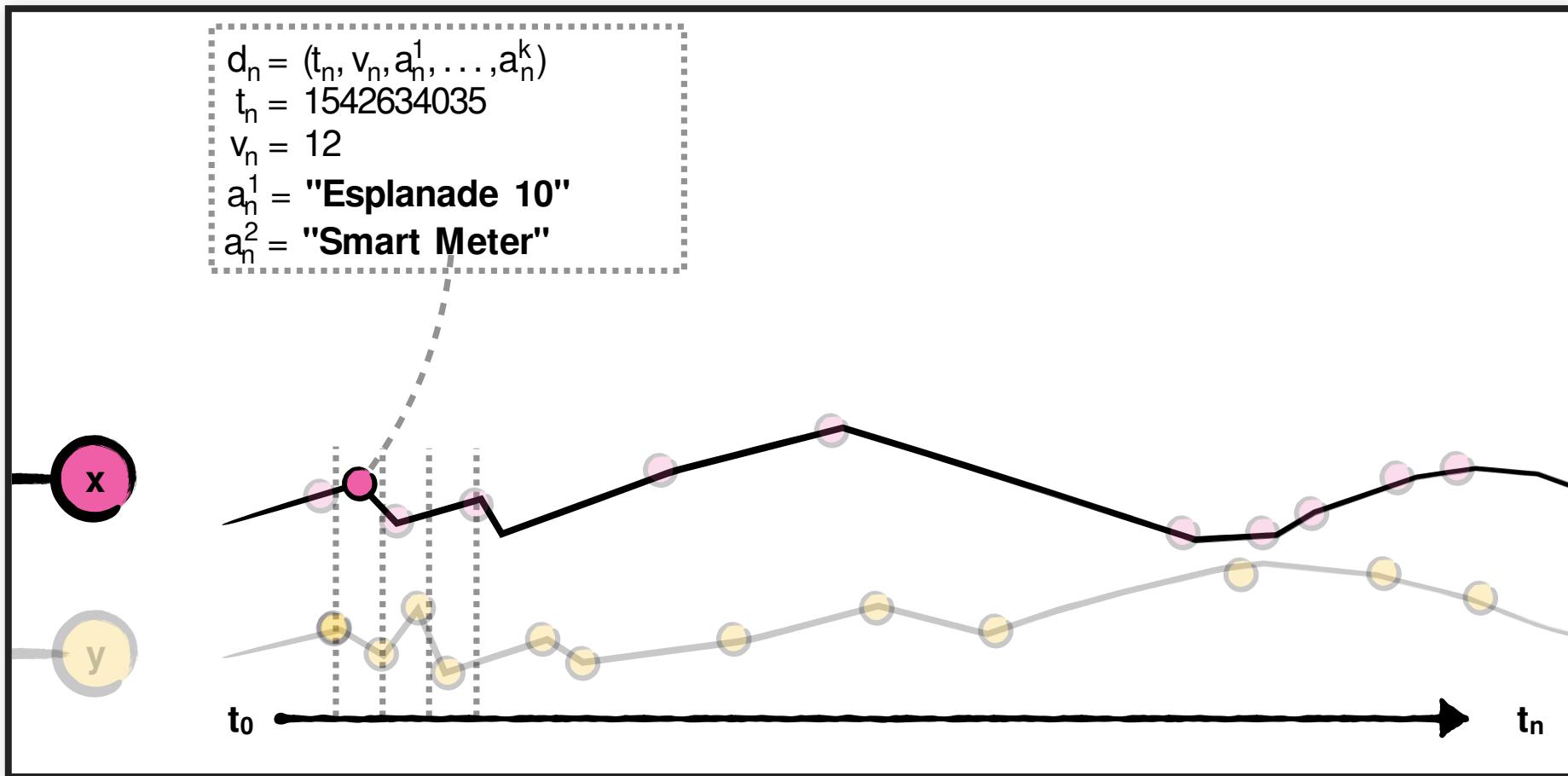
Für eine Verarbeitung ist daher zum Beispiel die zeitliche Synchronisation spannend



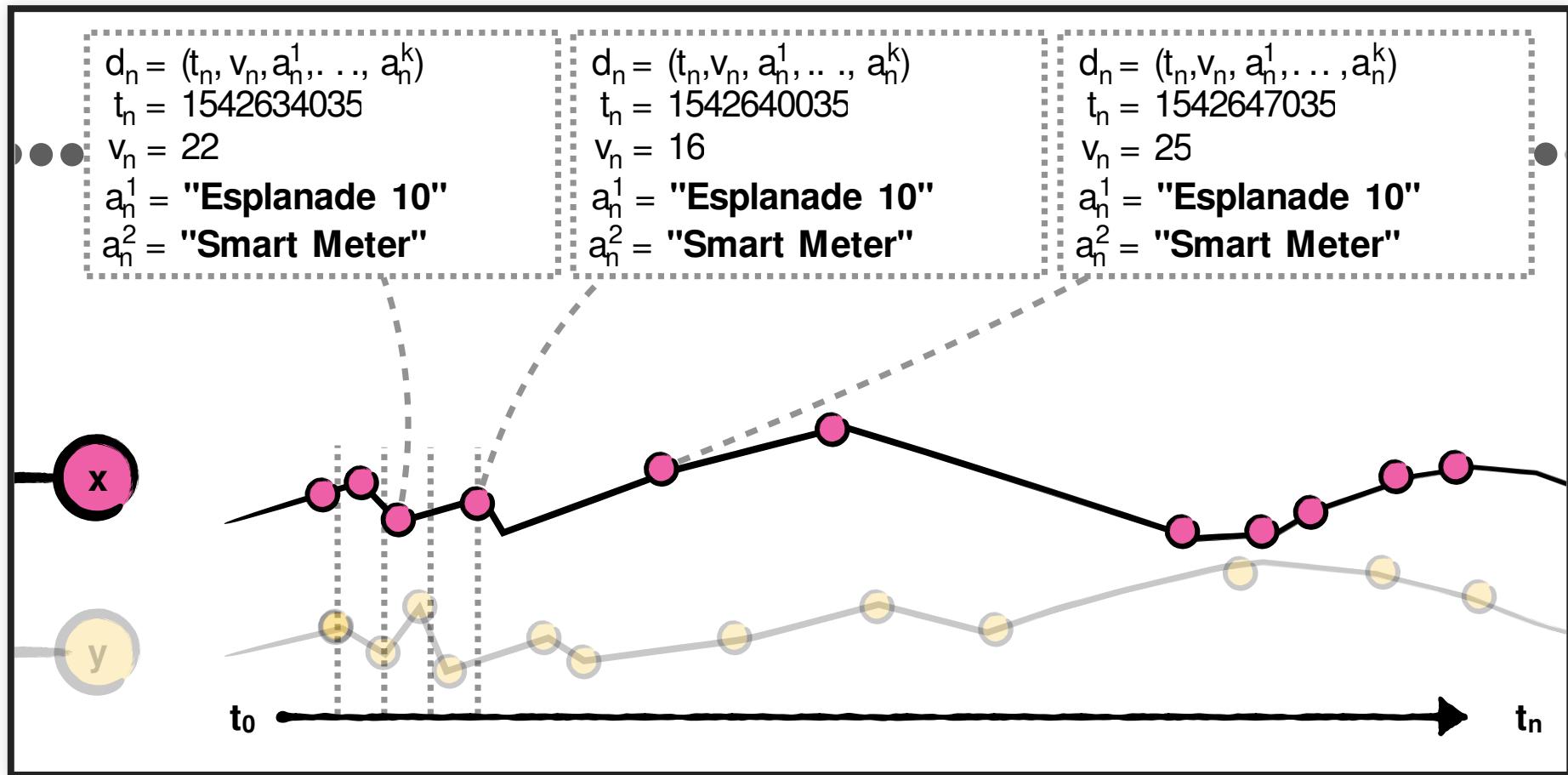
*Analyse der Datenströme und Berechnen höherwertiger Ereignisse und Kennzahlen
wäre ein weiteres Szenario für die Nutzung dieser Messreihen*



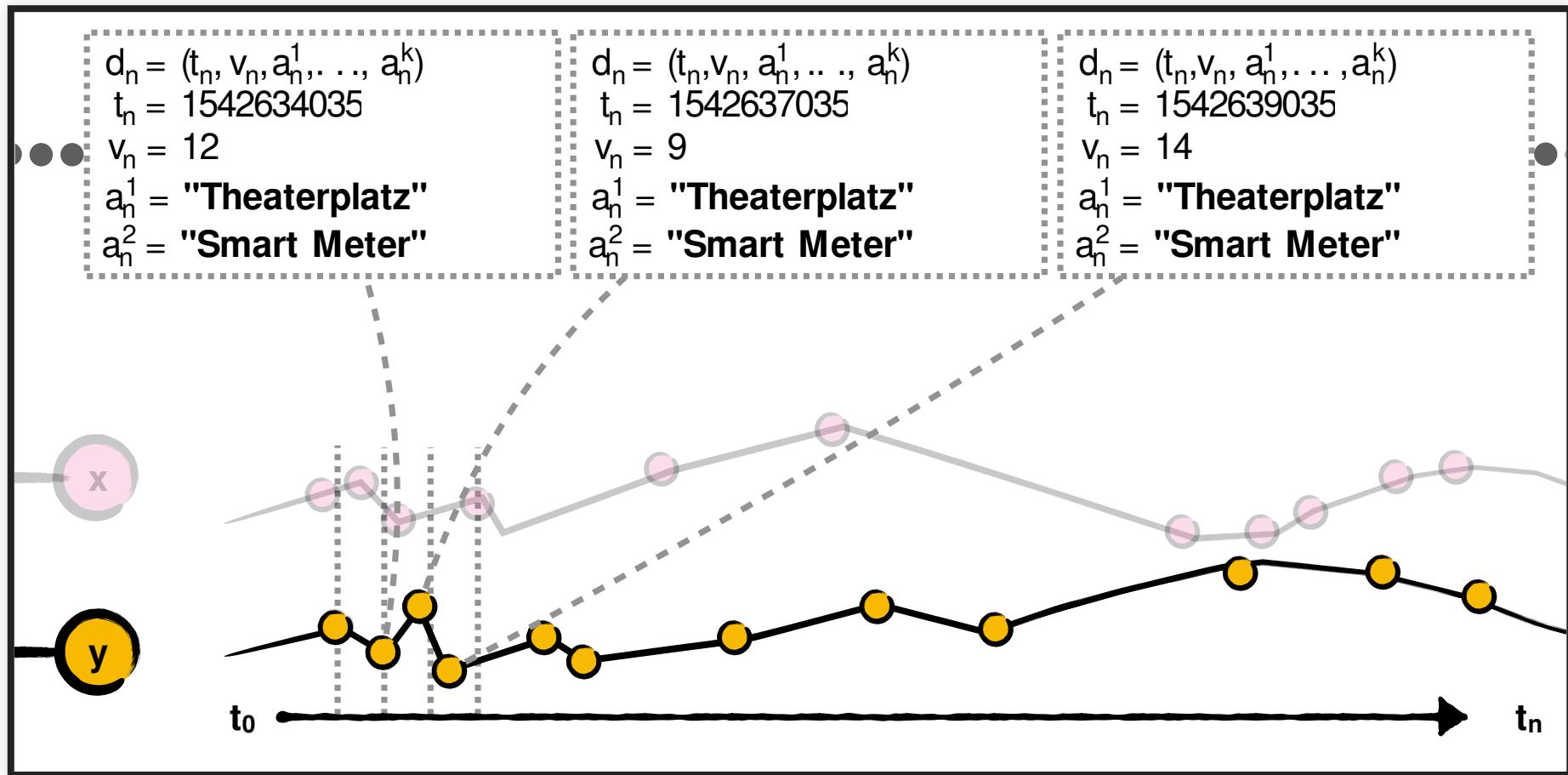
Woraus bestehen solche Daten? Wie lassen sich diese beschreiben?



Einen Messpunkt kann man durch einen Zeitbezug, den eigentlichen Wert und zusätzlichen Tags als Tupel beschreiben.



Eine Messreihe ergibt sich dann durch die konstanten Tags.



Eine weitere Zeitreihe für einen Messwert weist dann andere Tags auf.



***Data moves.** Almost all sources of data have an element of dynamism and motion about them. Even data at rest in some form of archival storage tier will have previously have led a more fluid life moving between applications, devices and network backbones and, inevitably, will have also moved to its resting place via a transport mechanism.*

But although almost all data moves, not all data moves at the same speed, at the same cadence and with the same kind of system circularity, size or value.

Quelle: <https://www.forbes.com/sites/adrianbridgwater/2022/09/22/why-modern-business-runs-on-data-streaming>

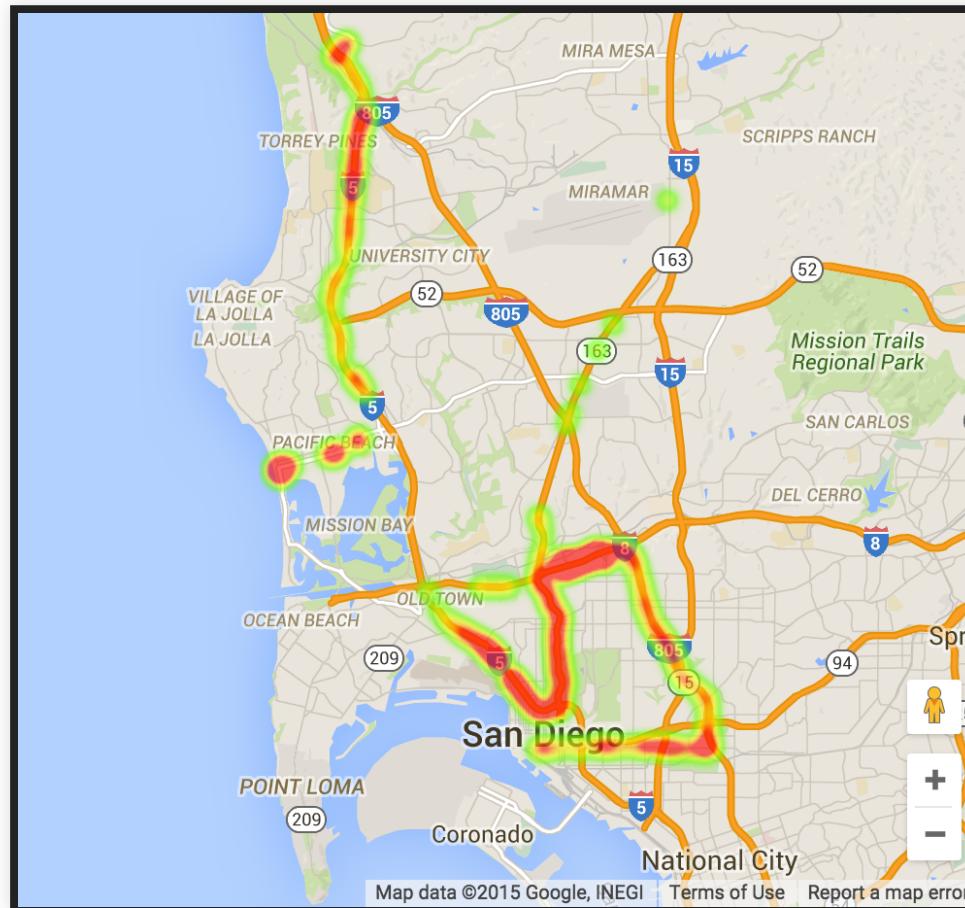


Zentrale Themen in den folgenden Kapiteln sind Daten in Bewegung, deren Erfassung, Verteilung, Verarbeitung und Verwendung, um damit Geschäftsprozesse zu beeinflussen. Dabei stehen Software-Architekturen und Software-Entwicklung im Vordergrund.

Technologien und Werkzeuge die betrachtet werden, spielen eine Rolle im Cloud-Context und lassen sich häufig als provisionierbare Dienste wiederfinden.

BEISPIEL: TELEMETRIE

Telemetrie und Visualisierung von Daten



Quelle: <https://cloud.google.com/architecture/scalable-geolocation-telemetry-system-using-maps-api?hl=de>



Telemetrie ist der Prozess der automatischen Erfassung regelmäßiger Messungen von Remote-Geräten. Wenn Sie zum Beispiel eine Fitness-Tracking- oder Mitfahrdienst-App neu erstellen, erfassen Sie wahrscheinlich Geolocation- und andere Sensordaten von einer großen Anzahl von bewegten Objekten wie Personen oder Fahrzeugen.

Telemetrie- und Telematics-Daten werden in der Regel als regelmäßige Meldung der GPS-Position zusammen mit einigen zusätzlichen Sensordaten wie Geschwindigkeit und Zeit gesendet.

Quelle: <https://cloud.google.com/architecture/scalable-geolocation-telemetry-system-using-maps-api?hl=de>



Zum Beispiel könnte ein Datensatz wie folgt aussehen:

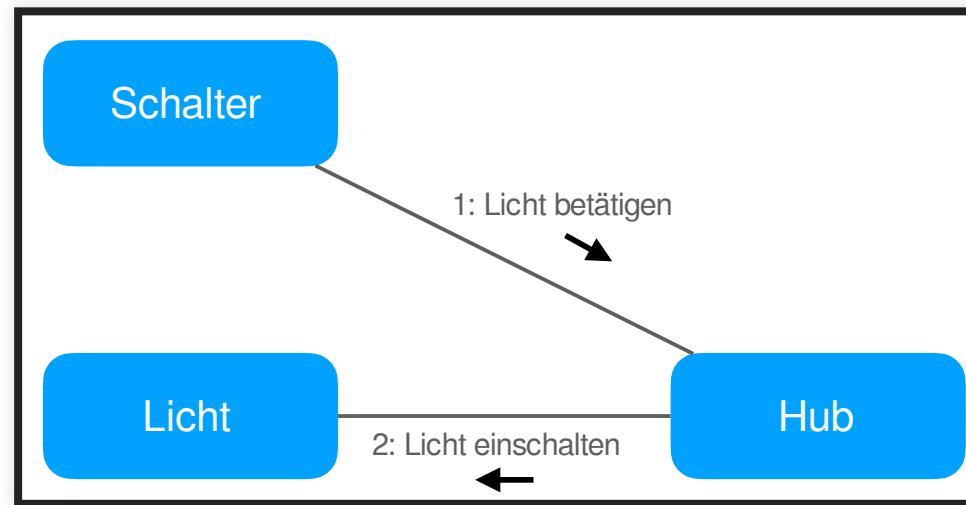
Timestamp	Latitude	Longitude	Speed	Bearing
97197600000	51.2345678	-0.1234556	34	261

Ziel wäre es diese Informationen über Endgeräte zu sammeln und geeignet zu verarbeiten, um daraus folgende Geschäftsprozesse zu realisieren. Zum Beispiel Navigationsanwendungen, welche eine Streckenführung realisieren die Staus berücksichtigen.

Jedes Informationspaket wäre dann ein Ereignis, welches in einem zeitlichen Kontext auftritt und verarbeitet werden muss.

BEISPIEL: SMART-HOME

Beispiel eines Smart-Home-Szenarios (1)



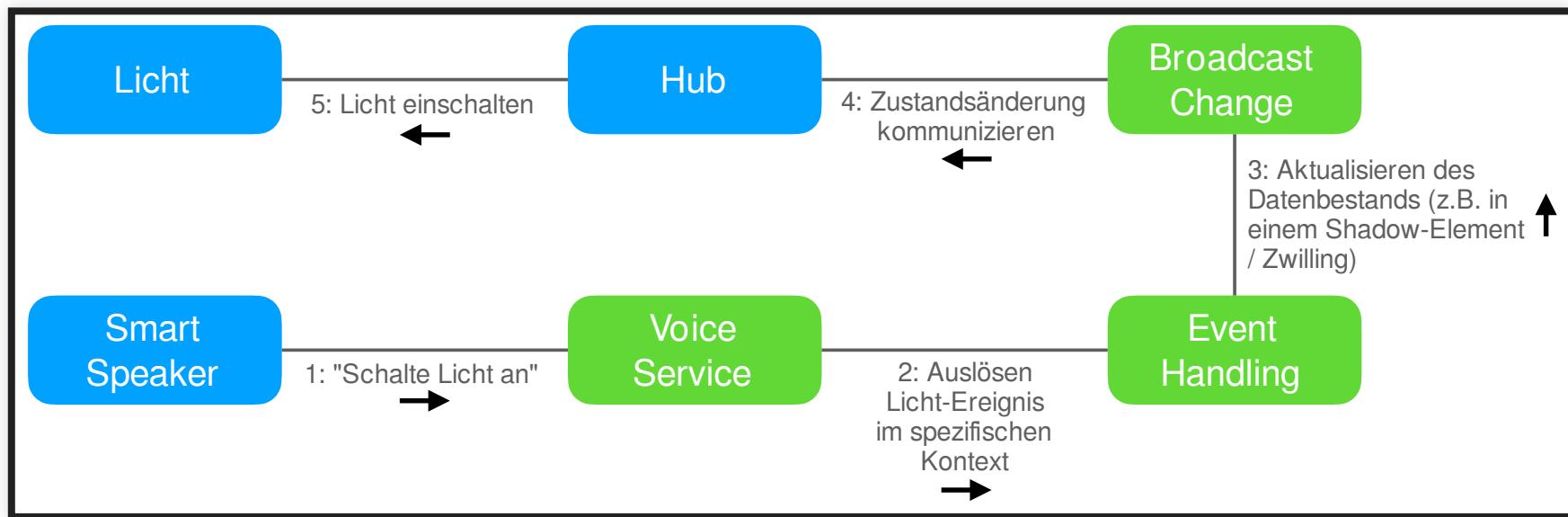
Das Beispiel zeigt einen typischen Schalter (Sensor) und ein typisches Licht (Aktor) wie es heute häufig in Smart Home Szenarien zu finden ist und Daten fließen. Wird der Schalter aktiviert oder deaktiviert könnte das Licht an bzw. ausgeschaltet werden.



Im Kontext einer ereignisbasierten Infrastruktur wäre die Frage, wie man mit dem Ereignis "*Schalter wurde aktiviert*" oder "*Schalter wurde deaktiviert*" umgeht. Kennt der Schalter das zu schaltende Licht? Überwacht das Licht den Schalter? Wie wird das Ereignis transportiert, damit das Licht reagieren kann?

Was passiert wenn mehrere Lichtschalter existieren? Wie sieht es aus, wenn mehrere Lampen geschaltet werden? Wie lässt sich das System erweitern ohne existierende Elemente zu modifizieren?

Beispiel eines Smart-Home-Szenarios (2)



Was passiert wenn zum Beispiel Smart Speaker in die Infrastruktur ergänzt werden?
Was bedeutet dies für eine Architektur?

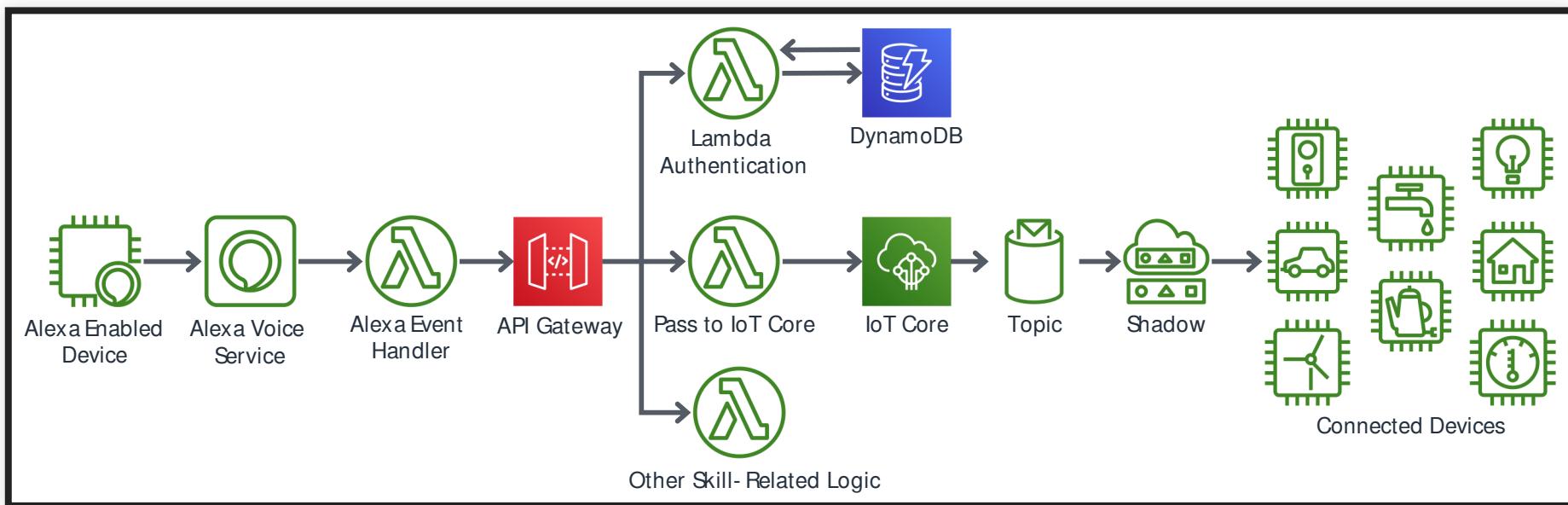


Ein aktueller Smart Speaker erfordert (meist) eine aktive Verbindung ins Internet, um die Sprachaufzeichnung auswerten zu lassen. Wenige führen dies lokal durch. Hierfür kommt ein Voice Service zum Einsatz, welcher die gesprochene Sprache in ein Kommando übersetzt. Dieses wird anschließend von einer Logik verarbeitet (im Kontext von Amazon Alexa zum Beispiel einem Skill, welche z.B. mit einem AWS Lambda gekoppelt ist).

Im Rahmen der Verarbeitung könnte es dann zu Ereignissen, wie z.B. "Schalte das Licht im Wohnzimmer an", welches zurück in das ursprüngliche Smart Home muss.

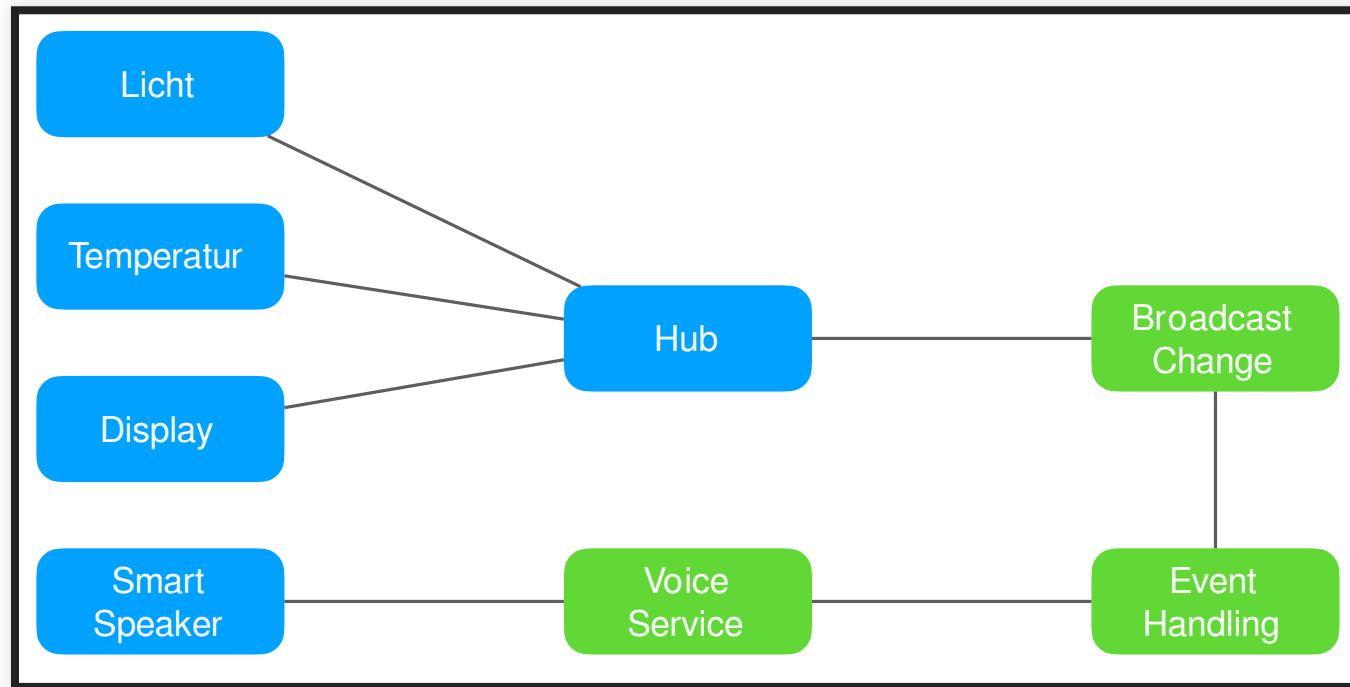
Hier wird z.B. mit digitalen Zwilligen gearbeitet, die die Zustandsänderung aufnehmen und über die verfügbaren Protokolle wird die Zustandsänderung an die realen Geräte zurückgespielt.

Verarbeitung eines Kommandos mit Alexa-Enabled Devices



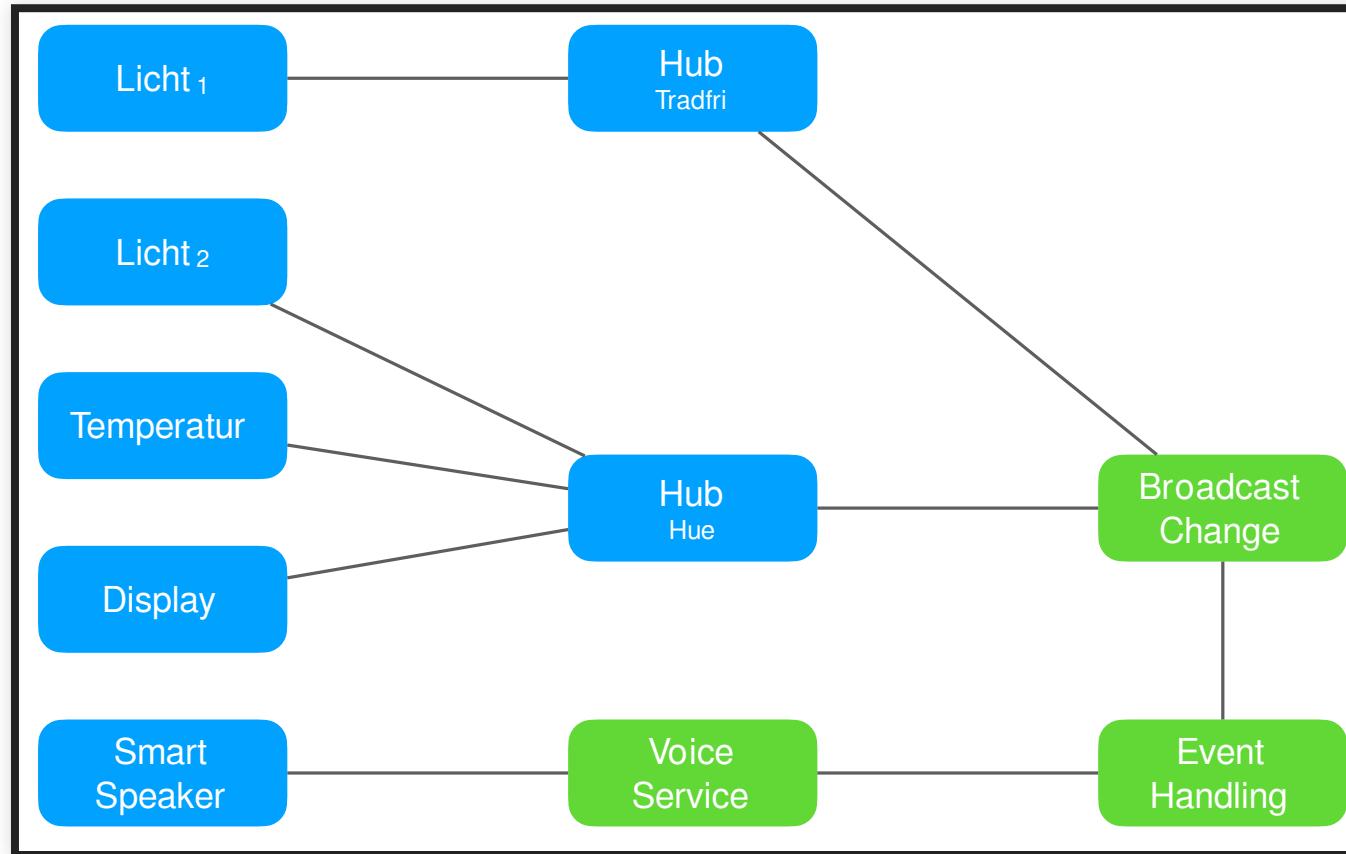
Quelle: <https://aws.amazon.com/de/blogs/iot/implement-a-connected-building-with-alexa-and-aws-iot/>

Beispiel eines Smart-Home-Szenarios (3)



Erweitert man das Szenario um verschiedene Endgeräte, ergeben sich vielfältige Aktoren und Sensoren. Nutzer können durch Displays das System steuern oder über Sensorik das System beeinflussen.

Beispiel eines Smart-Home-Szenarios (4)



Selbst heterogene Setups mit mehreren Herstellern lassen sich für ein solches Szenario finden.



DATENSTRÖME

Data streaming is a computing principle and operational system reality where (typically small in size) elements of data travel through an IT system in a time-ordered sequence.

Quelle: <https://www.forbes.com/sites/adrianbridgwater/2022/09/22/why-modern-business-runs-on-data-streaming>



Beispiele für Datastreaming

Quelle: <https://aws.amazon.com/de/streaming-data/>

- Sensoren in Transportfahrzeugen, Industriemaschinen und Landwirtschaftsmaschinen senden Daten an eine Streaming-Anwendung. Die Anwendung überwacht die Leistung, erkennt potentielle Defekte im Voraus und bestellt automatisch Ersatzteile, damit die Geräte nicht für längere Zeit ausfallen.
- Ein Finanzdienstleister verfolgt Änderungen am Börsenmarkt in Echtzeit nach, führt Value-at-Risk-Berechnungen durch und gleicht Portfolios automatisch entsprechend der Änderungen am Börsenmarkt aus.

- Eine Website für Immobilien verfolgt eine Untermenge an Daten von den Mobilgeräten der Verbraucher nach und schickt ihnen, basierend auf ihrem Standort, Vorschläge für potentielle Objekte zur Besichtigung in der Nähe zu.
- Ein Solarunternehmen muss seinen Kunden immer genug Strom zur Verfügung stellen oder es werden Strafen fällig. Es implementierte eine Streaming-Daten-Anwendung, die alle Solarzellen auf dem Gelände überwacht und in Echtzeit Wartungsmaßnahmen einplant, wodurch die Zeiträume niedriger Stromleistung für jede Zelle und die damit verbundenen Strafzahlungen minimiert werden.



- Ein Medienunternehmen streamt Milliarden Clickstream-Aufzeichnungen über seine Online-Präsenzen, aggregiert und bereichert die Daten mit demografischen Informationen der Benutzer und optimiert die Platzierung von Inhalten auf seiner Website, um relevantere Informationen und ein besseres Benutzererlebnis für das Zielpublikum bereitzustellen.
- Ein Online-Gaming-Unternehmen sammelt Streaming-Daten über die Interaktionen der Spieler im Spiel und implementiert diese Daten in seine Gaming-Plattform. Das Unternehmen analysiert die Daten dann in Echtzeit und schafft Anreize und dynamische Erfahrungen, um die Spieler bei Laune zu halten.

1.2 EVENT UND MESSAGE



Die Begriffe "Ereignis" und "Nachricht" werden beide im Zusammenhang mit dem Informationsaustausch verwendet, haben aber leicht unterschiedliche Bedeutungen.

Wie unterscheiden sich **ereignisbasierte** und **nachrichtenbasierte** Systeme?

Was ist ein **Event** und was ist eine **Nachricht**?

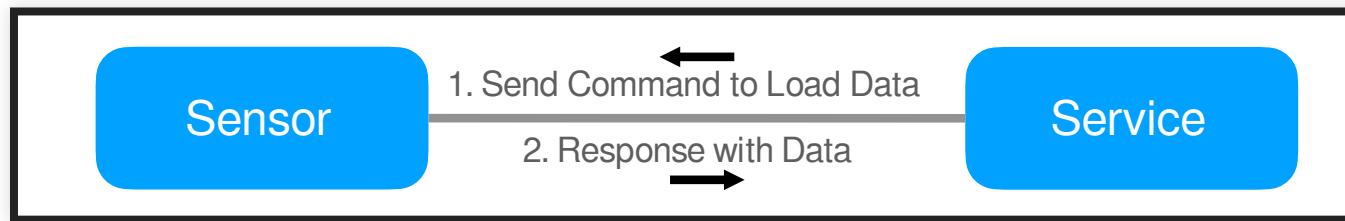


*A **message** is an item of data that is sent to a specific destination. An **event** is a signal emitted by a component upon reaching a given state. In a message-driven system addressable recipients await the arrival of messages and react to them, otherwise lying dormant. In an event-driven system notification listeners are attached to the sources of events such that they are invoked when the event is emitted.*

This means that an event-driven system focuses on addressable event sources while a message-driven system concentrates on addressable recipients. A message can contain an encoded event as its payload.

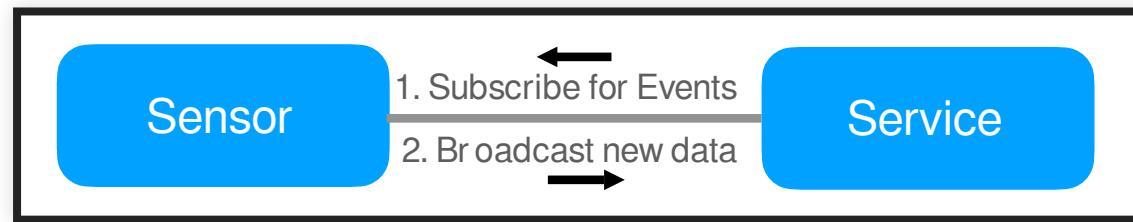
Quelle: <https://www.reactivemanifesto.org>

Messages in einem Message-Driven-System



In einem **nachrichtengesteuerten** System warten adressierbare Empfänger auf das Eintreffen von Nachrichten und reagieren darauf, andernfalls bleiben sie untätig. Als Beispiel könnte ein Temperatur-Sensor verwendet werden, welcher auf ein Kommando (1) wartet, um die Temperatur einmal zu liefern (2).

Events in einer Event-Driven-System



In einem **ereignisgesteuerten** System werden Listener an die Quellen von Ereignissen angehängt, so dass sie aufgerufen werden, wenn das Ereignis ausgelöst wird. Als Beispiel könnte ein Temperatur-Sensor verwendet werden, ein Service registriert sich einmalig für Ereignisse (1) und erhält anschließend die erfassten Werte (2) bei Aktualisierung.



Bei der Unterscheidung zwischen Event und Messages, sowie Event- und Message-Driven-System ist die Rolle eines Brokers wie Apache Kafka ein weiteres Thema und wird erst später aufgegriffen.

1.3 ARCHITEKTUR-BEGRIFFE

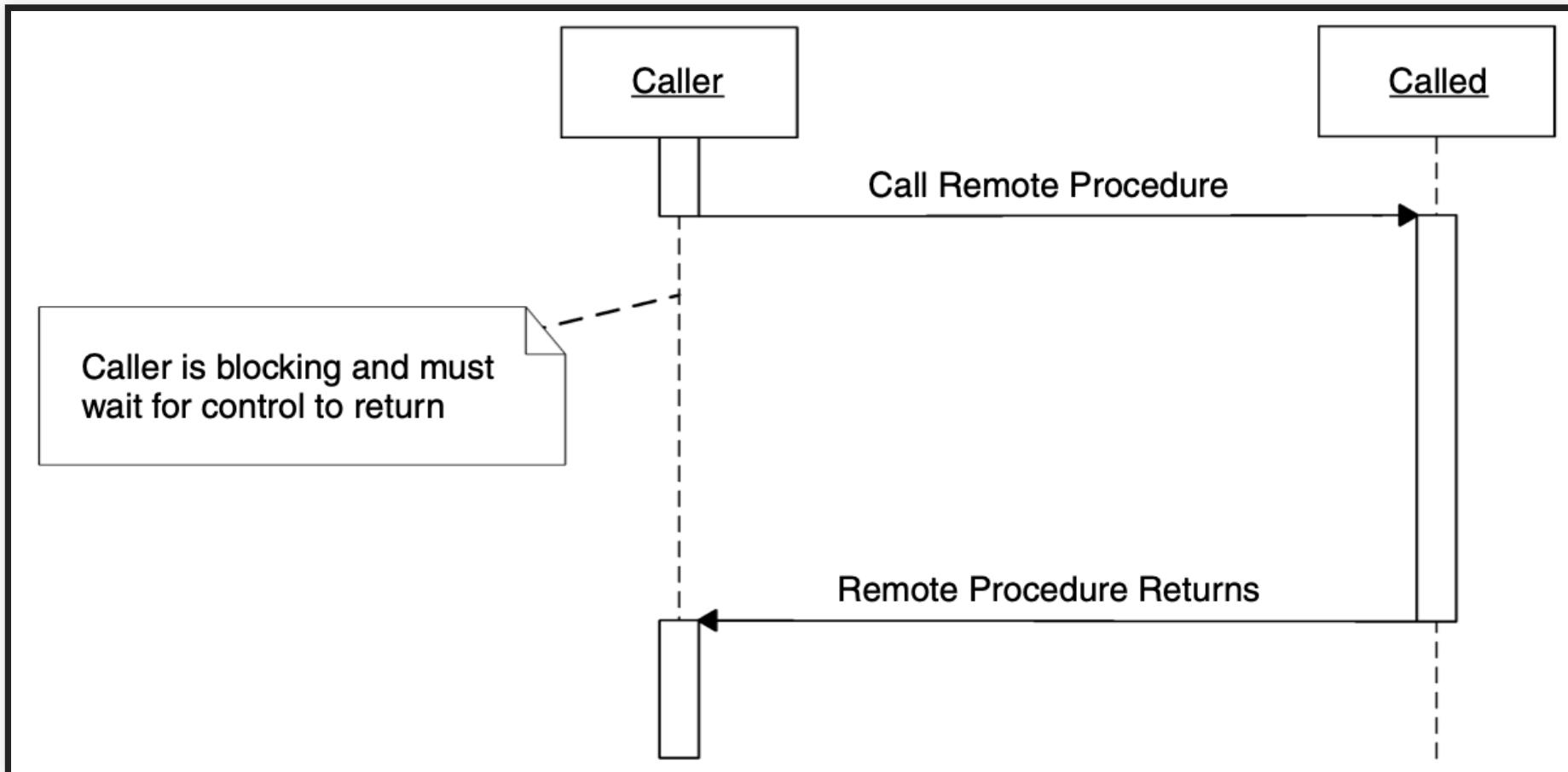


Für Ereignis- und Nachrichten-basierte Systeme finden sich verschiedene Architekturbegriffe die im Vorfeld eingesortiert werden sollen. Hierfür stehen im Folgenden im Vordergrund:

- Message-oriented Middleware (MOM)
- Event-driven Architecture
- Service-oriented Architecture
- Enterprise Service Bus
- Microservice Architecture

Hierfür ist zuvor eine Unterscheidung zw. synchronem und asynchronem Interaktionsmodell notwendig.

Synchronous Interaktionsmodell



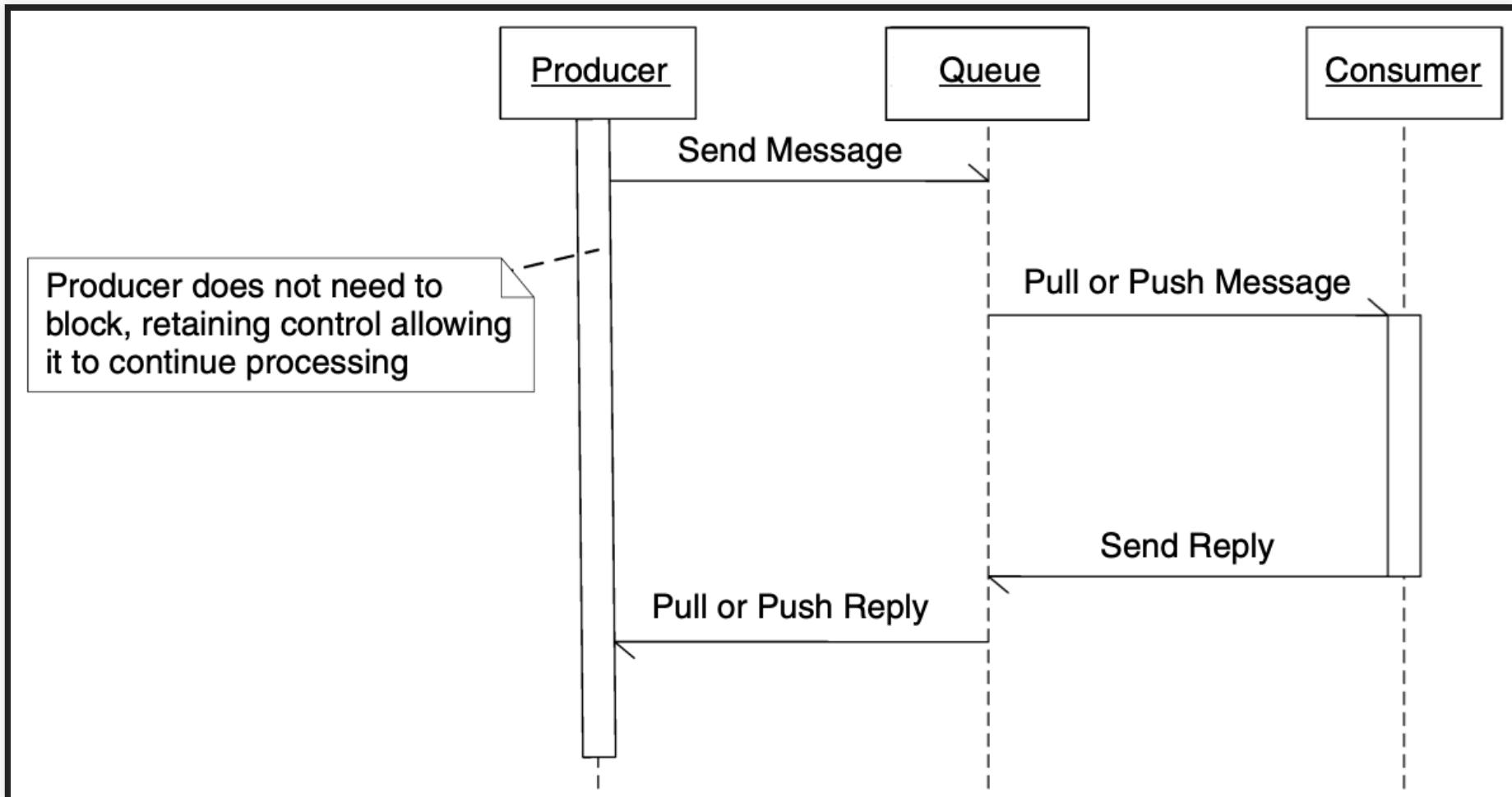
Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



- Wenn eine Prozedur/Funktion/Methode nach dem synchronen Interaktionsmodell aufgerufen wird, muss der aufrufende Code blockieren und warten (die Verarbeitung unterbrechen), bis der aufgerufene Code die Ausführung beendet und die Kontrolle an ihn zurückgibt
- Der aufrufende Code kann mit der Verarbeitung fortfahren wenn die Antwort eingegangen ist
- Bei Verwendung des synchronen Interaktionsmodells sind die Systeme nicht unabhängig von der Verarbeitungskontrolle, sie sind auf die Rückgabe der Kontrolle durch die aufgerufenen Systeme angewiesen

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf

Asynchronous Interaktionsmodell



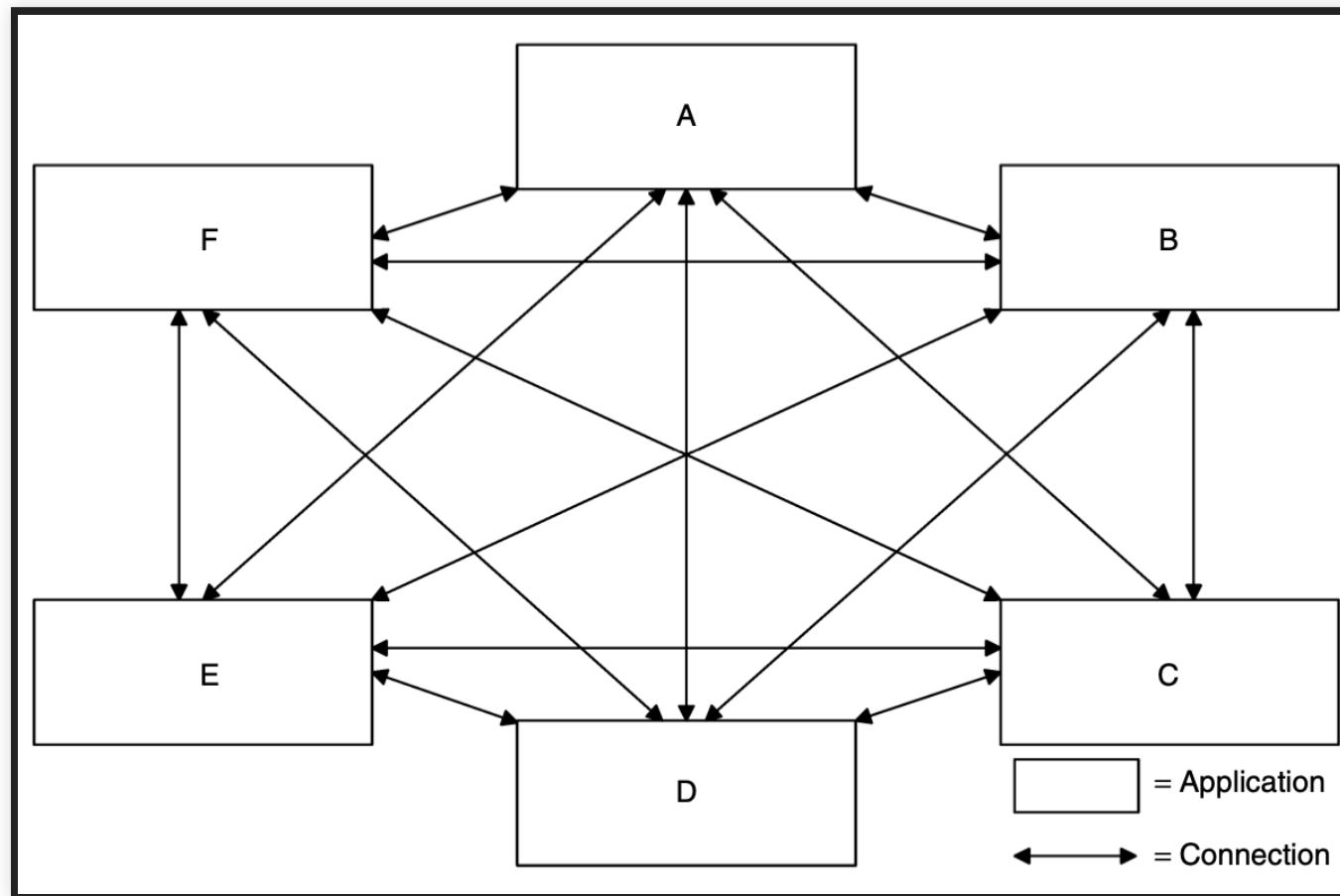
Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf

- Asynchrone Interaktionsmodell ermöglicht es dem Aufrufer, die Kontrolle über die Verarbeitung zu behalten
- Der aufrufende Code braucht nicht zu blockieren und auf die Rückkehr des aufgerufenen Codes zu warten
- Dieses Modell ermöglicht es dem Aufrufer, die Verarbeitung unabhängig vom Verarbeitungsstatus der aufgerufenen Prozedur/Funktion/Methode fortzusetzen
- Bei asynchroner Interaktion wird der aufgerufene Code möglicherweise nicht sofort ausgeführt
- Dieses Interaktionsmodell erfordert einen Vermittler, der den Austausch von Anfragen abwickelt, normalerweise ist dieser Vermittler eine Art Nachrichtenwarteschlange.

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf

REMOTE PROCEDURE CALL

- traditionelles Modell mit Remote Procedure Call als Konzept zur verteilten Datenverarbeitung
- Middleware-Plattformen wie CORBA, Java RMI, Microsoft DCOM und XML-RPC können hier eingesetzt werden
- Ziel von RPC ist es, die Interaktion zwischen zwei Prozessen zu ermöglichen
- RPC schafft die Möglichkeit, beide Prozesse glauben zu lassen, sie befänden sich im selben Prozessraum (d. h., sie sind ein und derselbe Prozess)
- Auf der Grundlage des synchronen Interaktionsmodells ähnelt RPC einem lokalen Prozederaufruf, bei dem die Kontrolle sequenziell und synchron an die aufgerufene Prozedur übergeben wird, während die aufrufende Prozedur blockiert und auf eine Antwort auf ihren Aufruf wartet



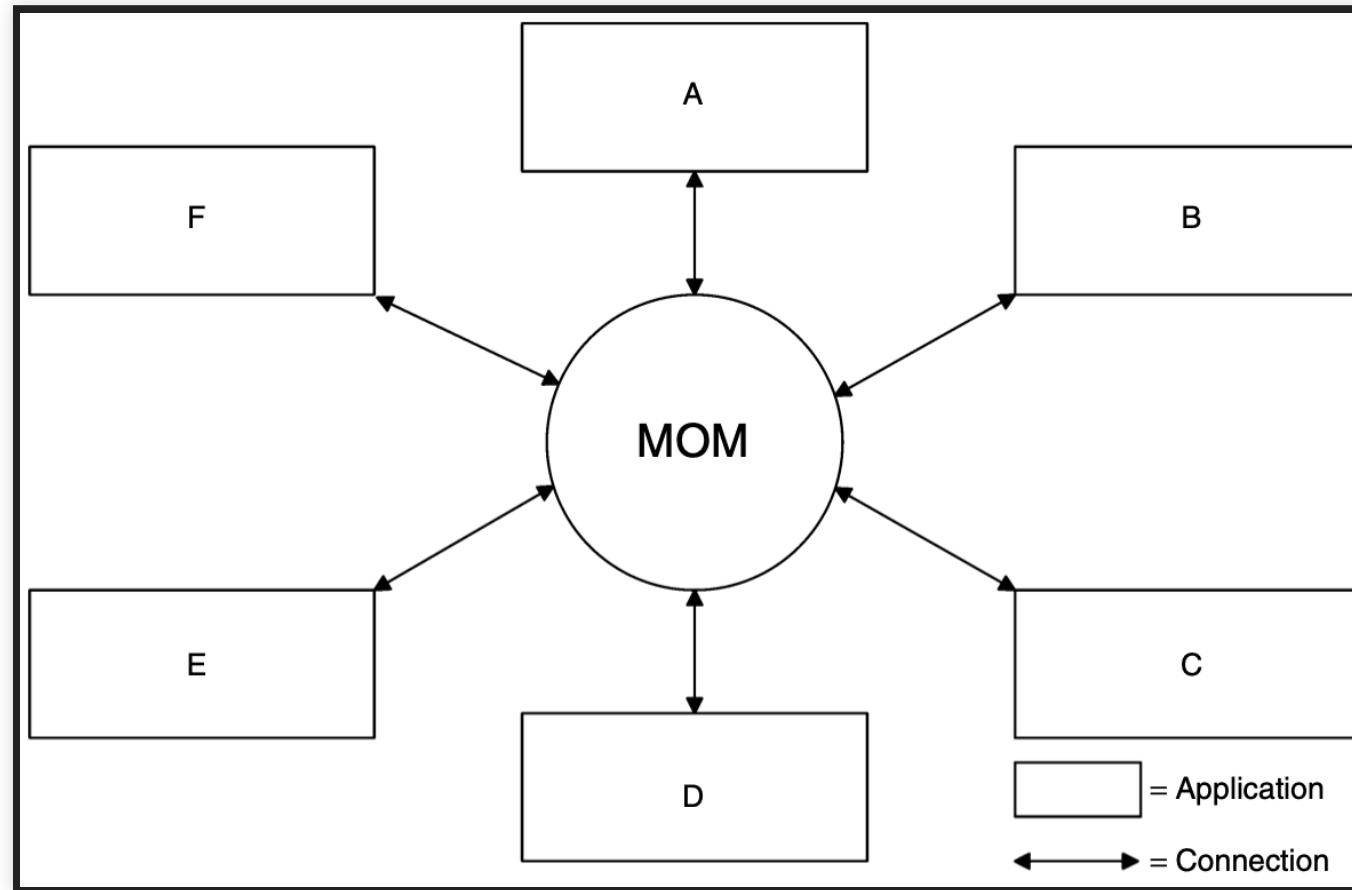
Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



MESSAGE-ORIENTED MIDDLEWARE

Bezeichnet Architekturen, bei denen einzelne Komponenten über eine Messaging Middleware entkoppelt werden.

- Ein Client eines MOM-Systems kann Nachrichten an andere Clients des Nachrichtensystems senden und von diesen empfangen
- Jeder Client stellt eine Verbindung zu einem (oder mehreren) Servern her, die als Vermittler für das Senden und Empfangen von Nachrichten fungieren
- MOM verwendet ein Modell mit einer Peer-to-Peer-Beziehung zwischen einzelnen Clients



Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



- In diesem Modell kann jeder Peer Nachrichten an andere Client-Peers senden und von ihnen empfangen
- MOM-Plattformen ermöglichen die Schaffung flexibler, kohäsiver Systeme (*ein kohäsives System ist ein System, das Änderungen in einem Teil des Systems zulässt, ohne dass Änderungen in anderen Teilen des Systems erforderlich sind*)

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



Systeme mit einer Message-oriented Middleware

- MOM-Systeme bieten eine verteilte Kommunikation auf der Grundlage des asynchronen Interaktionsmodells
- Dieses nicht blockierende Modell ermöglicht es MOM, viele der Einschränkungen zu überwinden
- Die Teilnehmer an einem MOM-basierten System müssen nicht blockieren und auf das Senden einer Nachricht warten, sondern können die Verarbeitung fortsetzen, sobald eine Nachricht gesendet wurde
- Dies ermöglicht die Zustellung von Nachrichten, wenn der Sender oder Empfänger zum Zeitpunkt der Ausführung nicht aktiv oder verfügbar ist, um zu antworten

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



Kopplung

MOM fügt eine Schicht zwischen Sender und Empfänger ein, die es den Sendern und Empfängern von Nachrichten ermöglicht, diese unabhängige Schicht als Vermittler für den Austausch von Nachrichten zu nutzen. Damit ist eine lose Kopplung zwischen den Systemteilnehmern verbunden - die Möglichkeit, Anwendungen miteinander zu verbinden, ohne dass Quell- und Zielsystem aneinander angepasst werden müssen, was zu der erwähnten kohäsiven, entkoppelten Systemstruktur führt.

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



Ausfallsicherheit

Mit MOM wird der Verlust von Nachrichten durch Netz- oder Systemausfälle vermieden, indem ein Store-and-Forward-Mechanismus für die Nachrichtenpersistenz verwendet wird. Diese Fähigkeit von MOM führt ein hohes Maß an Zuverlässigkeit in den Verteilungsmechanismus ein, Store and Forward verhindert den Verlust von Nachrichten, wenn Teile des Systems nicht verfügbar oder ausgelastet sind. Das spezifische Zuverlässigkeitsniveau ist in der Regel konfigurierbar, aber MOM-Nachrichtensysteme sind in der Lage zu garantieren, dass eine Nachricht zugestellt wird und dass sie jedem vorgesehenen Empfänger genau einmal zugestellt wird.

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



Scalability

Neben der Entkopplung der Interaktion von Teilsystemen entkoppelt MOM auch die Performance der Teilsysteme voneinander. Die Teilsysteme können unabhängig voneinander skaliert werden, ohne andere Teilsysteme zu stören. MOM ermöglicht es dem System auch, unvorhersehbare Aktivitätsspitzen in einem Teilsystem zu bewältigen, ohne andere Bereiche des Systems zu beeinträchtigen. MOM-Nachrichtenmodelle enthalten eine Reihe natürlicher Eigenschaften, die einen einfachen und effektiven Lastausgleich ermöglichen, indem sie es einem Teilsystem erlauben, eine Nachricht dann anzunehmen, wenn es dazu bereit ist, anstatt gezwungen zu werden, sie anzunehmen.

Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



Verfügbarkeit

MOM führt Hochverfügbarkeitsfunktionen in Systeme ein, die einen kontinuierlichen Betrieb und eine reibungslose Handhabung von Systemausfällen ermöglichen.

MOM erfordert nicht die gleichzeitige oder "zeitgleiche" Verfügbarkeit aller Teilsysteme. Der Ausfall eines Teilsystems führt nicht dazu, dass sich die Ausfälle auf das gesamte System ausbreiten. Aufgrund der losen Kopplung zwischen den MOM-Teilnehmern kann MOM auch die Reaktionszeit des Systems verbessern. Dadurch kann die Zeit für den Prozessabschluss verkürzt und die Reaktionsfähigkeit und Verfügbarkeit des Systems insgesamt verbessert werden.

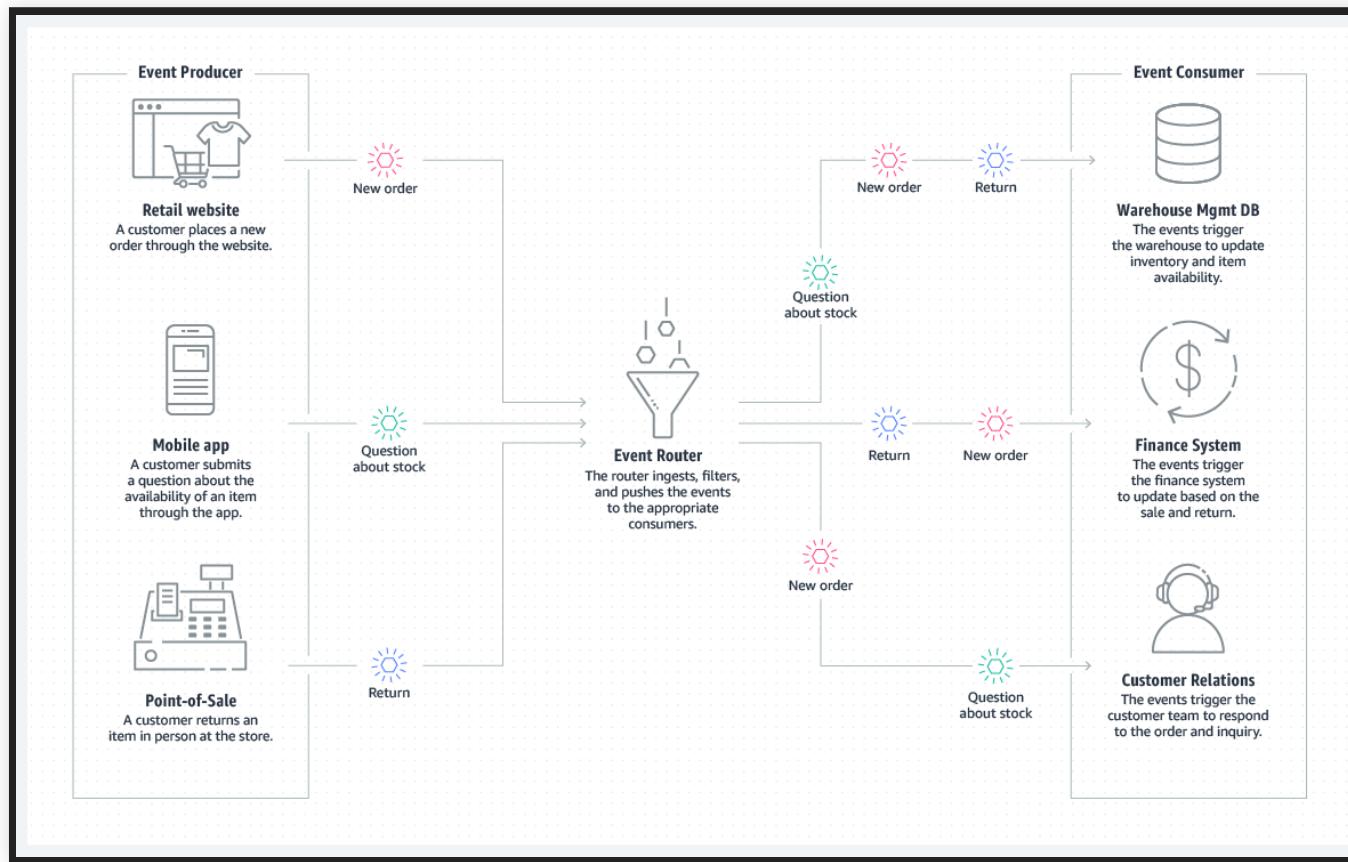
Quelle: https://www.edwardcurry.org/publications/curry_MfC_MOM_04.pdf



EVENT-DRIVEN ARCHITECTURE

- Eine ereignisgesteuerte Architektur ist in modernen Anwendungen, die üblicherweise mit Microservices entwickelt werden
- Sie verwendet Ereignisse zum Auslösen und Kommunizieren zwischen entkoppelten Services
- Ein Ereignis ist eine Zustandsänderung oder eine Aktualisierung, so wie ein Artikel, das auf einer E-Commerce-Website in einen Einkaufswagen gelegt wird
- Ereignisse können entweder den Status enthalten (den gekauften Artikel, seinen Preis und eine Lieferadresse) oder Identifikatoren sein (eine Benachrichtigung, dass eine Bestellung versandt wurde)

Quelle: <https://aws.amazon.com/de/event-driven-architecture/>



Quelle: <https://aws.amazon.com/de/event-driven-architecture/>



Skalieren und Ausfallsicherheit

Durch die Entkopplung Ihrer Services nehmen diese nur den Ereignis-Router wahr, aber nicht einander. Das bedeutet, dass Ihre Services interoperabel sind, aber wenn ein Service ausfällt, laufen die anderen weiter. Der Ereignis-Router fungiert als elastischer Puffer, der Lastspitzen auffangen wird.

Quelle: <https://aws.amazon.com/de/event-driven-architecture/>



Zugriffskontrolle

Ein Ereignis-Router dient als zentraler Ort für die Prüfung Ihrer Anwendung und die Festlegung von Richtlinien. Diese Richtlinien können einschränken, wer einen Router veröffentlichen und abonnieren darf, und steuern, welche Benutzer und Ressourcen die Berechtigung zum Zugriff auf Ihre Daten haben. Darüber hinaus können Sie Ihre Ereignisse sowohl während der Übertragung als auch im Ruhezustand verschlüsseln.

Quelle: <https://aws.amazon.com/de/event-driven-architecture/>



Agilität

Keine benutzerdefinierter Code mehr, um Ereignisse abzufragen, zu filtern und weiterzuleiten; der Ereignis-Router filtert und leitet Ereignisse automatisch an die Verbraucher weiter. Durch den Router entfällt die Koordination zwischen Produzenten- und Konsumenten-Services, was den Entwicklungsprozess beschleunigen kann.

Quelle: <https://aws.amazon.com/de/event-driven-architecture/>



SERVICE-ORIENTED ARCHITECTURE

Die serviceorientierte Architektur (SOA) ist ein unternehmensweiter Ansatz für die Softwareentwicklung von Anwendungskomponenten, der sich wiederverwendbare Softwarekomponenten oder Dienste zunutze macht. In der SOA-Softwarearchitektur besteht jeder Service aus dem Code und den Datenintegrationen, die für die Ausführung einer bestimmten Geschäftsfunktion erforderlich sind. Zum Beispiel die Überprüfung der Kreditwürdigkeit eines Kunden, die Anmeldung auf einer Website oder die Bearbeitung eines Hypothekenantrags.

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>



Die Serviceschnittstellen bieten eine lose Kopplung, was bedeutet, dass sie mit wenig oder gar keinem Wissen darüber aufgerufen werden können, wie die Integration darunter implementiert ist. Aufgrund dieser losen Kopplung und der Art und Weise, wie die Dienste veröffentlicht werden, können Entwicklungsteams durch die Wiederverwendung von Komponenten in anderen Anwendungen im gesamten Unternehmen Zeit sparen. Dies ist sowohl ein Vorteil als auch ein Risiko. Durch den gemeinsamen Zugriff auf den Enterprise Service Bus (ESB) können sich Probleme auch auf die anderen angeschlossenen Dienste auswirken.

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>



SOA bietet vier verschiedene Diensttypen:

- Funktionale Dienste (d. h. Geschäftsdienste), die für Geschäftsanwendungen entscheidend sind.
- Unternehmensdienste, die zur Implementierung von Funktionalität dienen.
- Anwendungsdienste, die für die Entwicklung und Bereitstellung von Anwendungen verwendet werden.
- Infrastrukturdienste, die für Backend-Prozesse wie Sicherheit und Authentifizierung wichtig sind.

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>

Jeder Dienst besteht aus drei Komponenten:

- **Interface:** legt fest, wie ein Dienstanbieter Anfragen von einem Dienstnutzer ausführt
- **Contract:** legt fest, wie der Dienstanbieter und der Dienstnutzer interagieren sollen
- **Implementation:** der Code des Dienstes

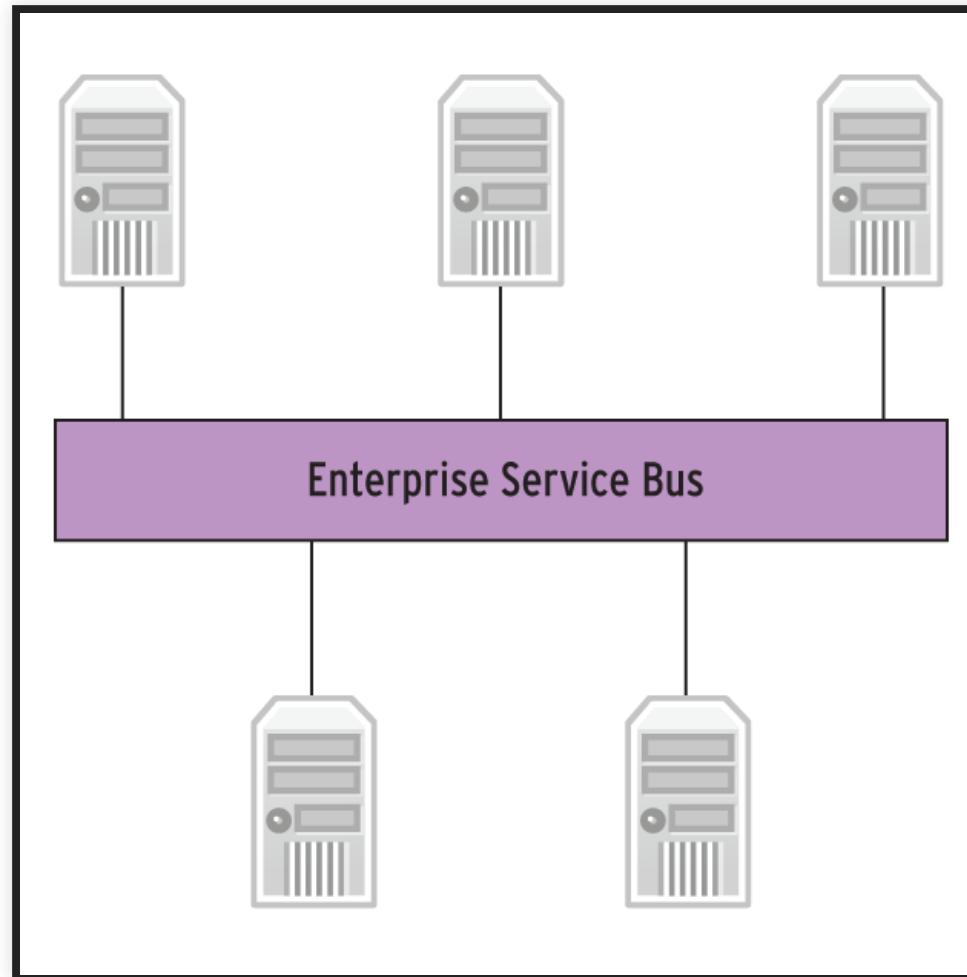
Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>



ENTERPRISE SERVICE BUS

Ein ESB oder Enterprise-Service-Bus ist ein Muster, bei dem eine zentrale Softwarekomponente Integrationen auf Back-End-Systeme (und Übersetzungen von Datenmodellen, tiefe Konnektivität, Routing und Anforderungen) ausführt und diese Integrationen und Übersetzungen als Serviceschnittstellen zur Wiederverwendung durch neue Anwendungen bereitstellt. Das ESB-Muster wird in der Regel mit einer speziell konzipierten Integrationslaufzeit und einem Toolset implementiert, die die bestmögliche Produktivität gewährleisten.

Quelle: <https://www.ibm.com/de-de/cloud/learn/esb>



Quelle: <https://www.linux-magazin.de/ausgaben/2011/12/enterprise-service-bus/>



MICROSERVICE ARCHITECTURE

- Microservices-Architekturen bestehen (wie SOA) aus lose gekoppelten, wiederverwendbaren und spezialisierten Komponenten, die oft unabhängig voneinander arbeiten
- Microservices verwenden auch ein hohes Maß an Kohäsion, auch bekannt als Bounded Context
- Bounded Context bezieht sich auf die Beziehung zwischen einer Komponente und ihren Daten als eigenständige Einheit mit sehr wenigen Abhängigkeiten.

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>



- Anstatt unternehmensweit eingeführt zu werden, kommunizieren Microservices in der Regel über Anwendungsprogrammierschnittstellen (APIs), um individuelle Anwendungen zu erstellen
- Die Services führen eine bestimmte Geschäftsfunktion (oder Funktionen für bestimmte Geschäftsbereiche) auf eine Weise aus
- Dieser Ansatz macht die Services flexibler, skalierbarer und widerstandsfähiger
- Für die Entwicklung eines Dienstes steht Technologieunabhängigkeit im Vordergrund, so kann jeder Dienst mit der für den Anwendungsfall optimalen Technologiestack realisiert werden, z.B. Java mit dem Spring Framework oder auch in Python, NodeJS, usw.

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>



- Microservices sind ein echter Cloud-nativer Architekturansatz, der häufig in Containern betrieben wird, was sie skalierbarer und portabler für die Erstellung unabhängiger Dienste macht
- Teams können Microservices nutzen, um
 - Code einfacher zu aktualisieren
 - unterschiedliche Stacks für verschiedene Komponenten zu verwenden
 - die Komponenten unabhängig voneinander zu skalieren
- Diese Architektur erlaubt es die Kosten zu reduzieren, die mit der Skalierung ganzer Anwendungen verbunden sind, weil eine einzelne Funktion zu stark belastet wird
- Aufgrund ihrer Unabhängigkeit erzeugen Microservices Dienste, die fehlertoleranter sind

Quelle: <https://www.ibm.com/cloud/blog/soa-vs-microservices>

1.4 BATCH- VS. STREAM-PRCESSING



Wie unterscheiden sich die folgenden Beispiele bei der Verarbeitung der Daten?

Nutzen Sie das bereitgestellte Projekt-Archiv `processing-example.zip`, entpacken Sie dieses und öffnen Sie den Ordner in Visual Studio Code.

Im Kontext der Datenverarbeitung trifft man auf die Begriffe das Batch- und Stream-Processings die im Folgenden abgegrenzt werden.



BATCH-PROCESSING

- Klassische Datenverarbeitungsprozesse nutzen das sogenannte Batch-Processing
- deutsche Übersetzung für Batch Processing ist *Stapelverarbeitung*
- Beim Batch Processing werden Daten nach ihrer Entstehung oder ihrem Erhalt von einer Eingangslogik sortiert und strukturiert
- Anschließend erfolgt die Ablage in einer Datenbank, dort verbleiben die Daten, bis diese für eine Weiterverarbeitung oder eine Analyse der Daten notwendig sind

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>

- Bei der Verarbeitung werden die benötigten Daten zu einem definierten Zeitpunkt entnommen und der Analyselogik oder der weiterverarbeitenden Anwendung übergeben
- Die erhaltenen Ergebnisse sind nicht echtzeitfähig und basieren immer auf einem historischen Datenbestand zu einem bestimmten Zeitpunkt
- Während der Analyse oder der Weiterverarbeitung erhaltene Daten fließen nicht in die Analyseprozesse ein und können erst wieder zu einem späteren Zeitpunkt berücksichtigt werden

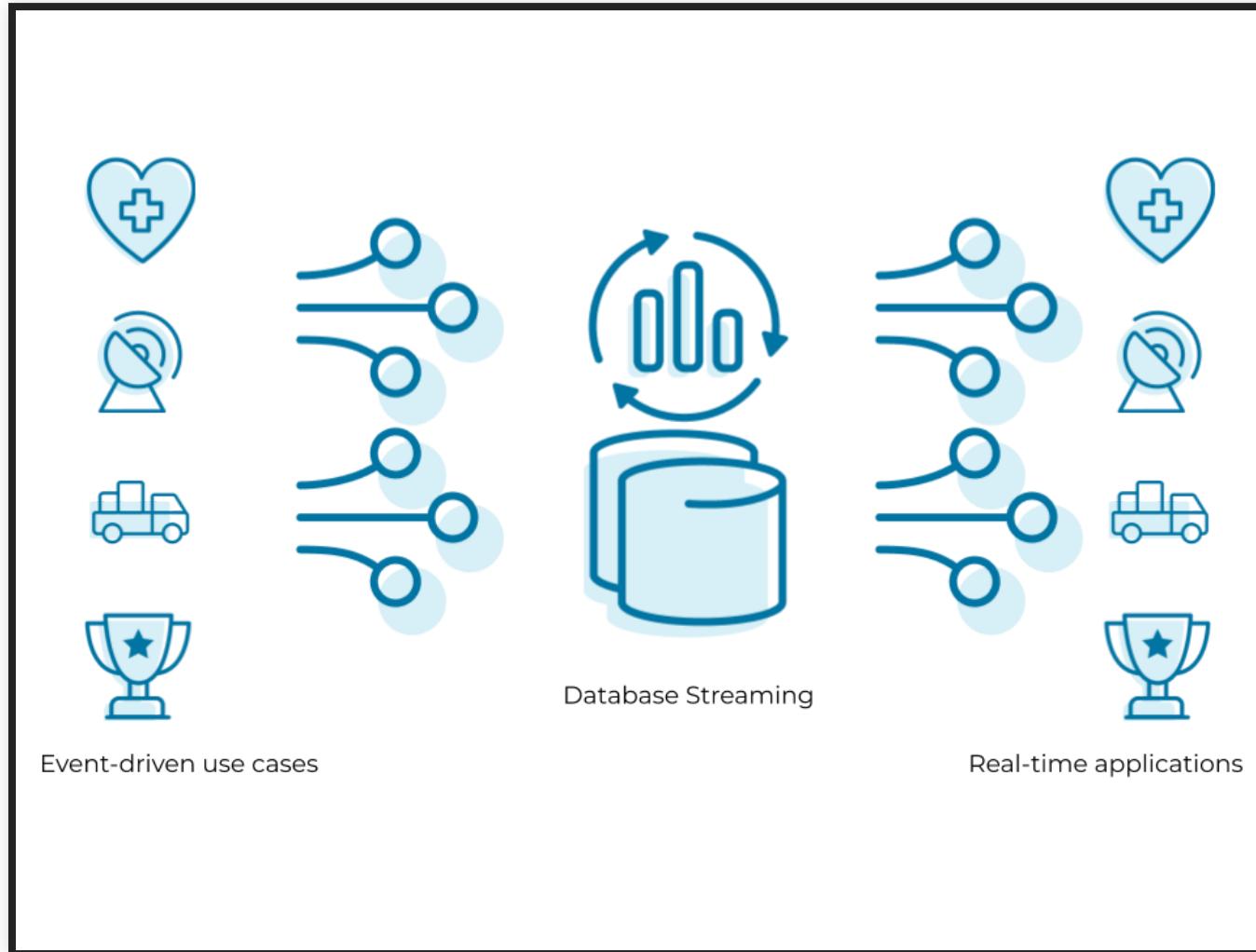
Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>



STREAM-PROCESSING

- Streaming-Processing teilt sich in zwei Ebenen: Speicherebene und Verarbeitungsebene
- Speicherebene unterstützt das Ordnen von Aufzeichnungen und muss äußerst kohärent sein, um schnelle, günstige und wiederholbare Lese- und Schreibzugänge von großen Datenströmen zu ermöglichen
- Auf der Verarbeitungsebene werden die Daten von der Speicherebene verwendet, um Berechnungen mit den Daten durchzuführen
- Anschließend wird die Speicherebene in Kenntnis gesetzt, welche nicht mehr benötigten Daten gelöscht werden können
- Es müssen bei beiden Ebenen Skalierbarkeit, Datenlanglebigkeit und Fehlertoleranz berücksichtigt werden

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>



Quelle: <https://www.confluent.io/learn/data-streaming/>

- Basis des Stream Processings sind kontinuierliche Datenströme.
- Datenquellen, wie Sensoren, erzeugen einen ständigen Datenstrom in einem bestimmten Datenformat.
- Der Datenstrom besteht aus Einzeldaten oder kleineren Paketen mit mehreren Daten.
- Wie viele Daten in die Pakete gefüllt werden, kann zeit- oder mengenabhängig sein.
- Empfänger wie Analyseanwendungen nehmen den Datenstrom entgegen und verarbeiten ihn.

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>

Natives Streaming und Micro-Batching

- Natives Streaming verarbeitet jeden Datensatz sofort, ohne auf weitere Datensätze zu warten
- Micro-Batching hingegen sammelt Daten über kurze Zeitabstände (Zeitfenster) und führt anschließend die Weiterverarbeitung durch

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>

Plattformen im Kontext von Stream-Processing

- Apache Flume
- Apache Spark Streaming
- Apache Storm
- Apache Kafka Streaming
- Amazon Kinesis Data Streams
- Amazon Kinesis Data Firehose

Gründe für das Stream Processing

- Steigende Anzahl von Datenquellen, die kontinuierlich zu analysierende Daten liefern, stößt das Prinzip des Batch Processings an seine Grenzen
- Unternehmen sind mit ihren digitalen Prozessen darauf angewiesen, Daten in Echtzeit zu verarbeiten und binnen kurzer Zeit angemessen auf Analyseergebnisse zu reagieren
- Je mehr Zeit zwischen der Datenentstehung oder dem Datenerhalt und der Auswertung der Daten liegt, desto geringer ist der Wert der Analyseergebnisse

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>



- Zwischenspeicherung großer Mengen an Daten einen riesigen Bedarf an Ressourcen und Speicherbedarf für Datenbanksysteme
- Dynamische Markt- und Produktionsprozesse erwarten zeitnahe Auswertungen in Echtzeit
- Stream-Processing muss im Gegensatz zum Batch Processing kontinuierliche Datenströme nicht unterbrechen und Daten anschließend wieder aufwendig aggregieren, sondern erzeugt einen kontinuierlichen Ergebnis-Output

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>



Typische Anwendungsbereiche des Stream Processings

Anwendungsmöglichkeiten für das Stream Processing ergeben sich im Bereich Künstlicher Intelligenz (KI), maschinellen Lernens, Big Data, Industrie 4.0, Internet der Dinge und anderen Bereichen.

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>

Im Folgenden einige typische Anwendungsbereiche.

- Echtzeit-Analyse des Kundenverhaltens im E-Commerce
- Trigger-Marketing-Kampagnen
- Analysen von Finanztransaktionen
- Patientenüberwachung im Gesundheitswesen
- Monitoring von Produktionsprozessen
- Optimierung von Logistikketten
- Intrusion Detection
- autonomes Fahren
- Smart Grids
- Verkehrsüberwachung
- Userverhalten-bezogene oder Aktions-basierte Werbung
- Vorausschauende Wartung und Instandhaltung (Predictive Maintenance)

Quelle: <https://www.bigdata-insider.de/was-ist-stream-processing-a-863076/>

1.5 BEZUG ZU CLOUD-APPLICATIONS

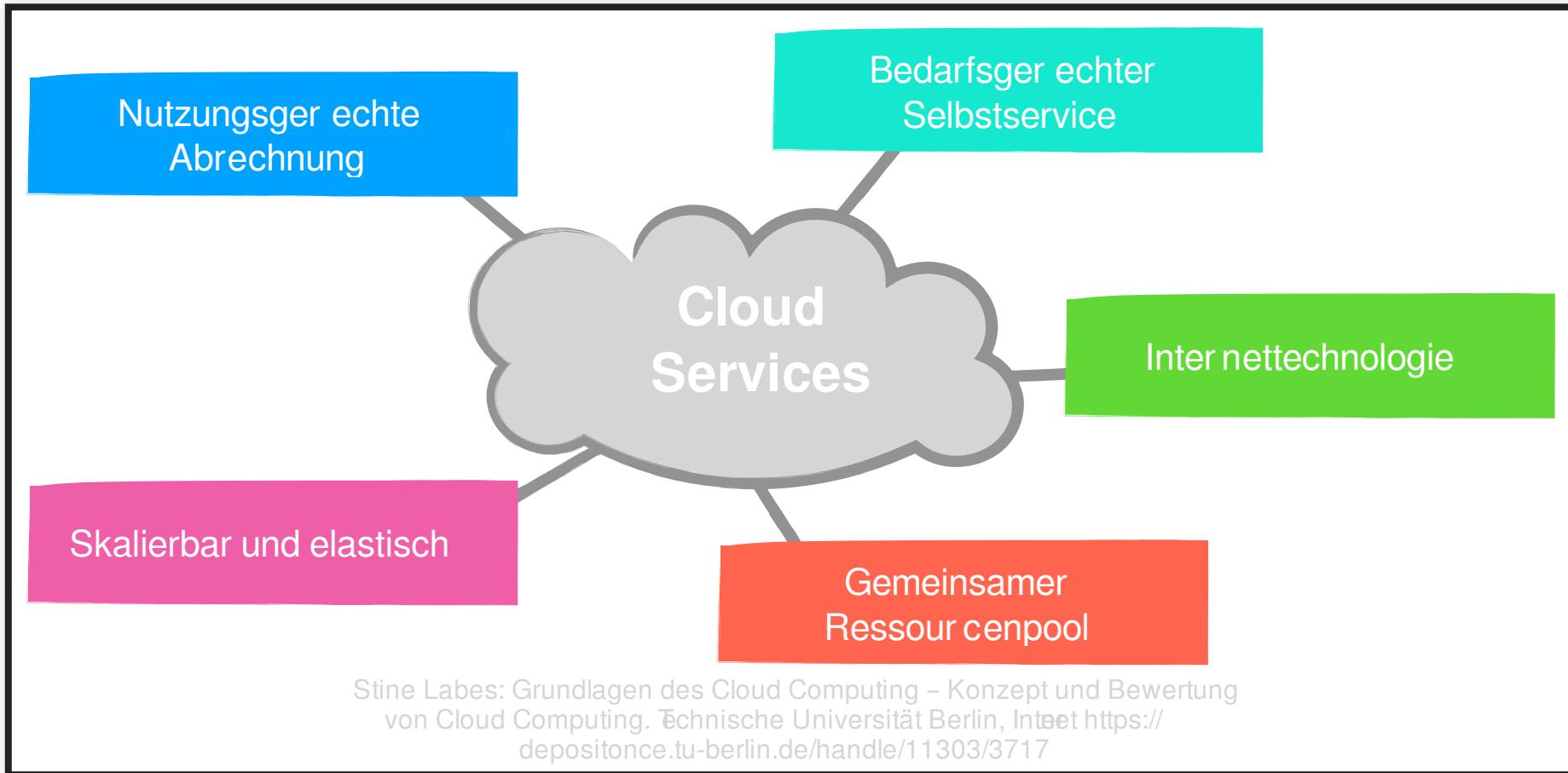


CLOUD-COMPUTING

Cloud Computing ist ein Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.

Quelle: Standardisierungsstelle NIST (National Institute of Standards and Technology)

Technologie-Merkmale von Cloud-Computing





- **Bedarfsgerechter Selbst-Service**

Der Verbraucher kann den benötigten Umfang des Services selbst zusammenstellen, ohne direkte physische Interaktion mit dem Anbieter

- **Internettechnologie**

Die Services werden mit Hilfe von gegebenen Standards über das Internet zur Verfügung gestellt

- **Gemeinsamer Ressourcenpool**

Die Services verfügen über einen gemeinsam nutzbaren Ressourcenpool, so dass der Anbieter von Größenvorteilen profitieren kann

- **Skalierbar und elastisch**

Die Services können bei Bedarf skaliert werden, so dass Ressourcen jederzeit hinzugefügt oder entfernt werden können

- **Nutzungsgerechte Abrechnung**

Die Nutzung der Services wird durch Nutzungskennzahlen protokolliert, so dass eine Kostentransparenz gegeben ist



Bereitstellungsmodelle

- **Private Cloud:** Infrastruktur ist nur für eine Institution betrieben, sie kann von der Institution selbst oder einem Dritten organisiert und geführt werden und kann dabei im Rechenzentrum der eigenen Institution oder einer fremden Institution stehen
- **Public Cloud:** die Services werden von der Allgemeinheit oder einer großen Gruppe, wie beispielsweise einer ganzen Industriebranche, genutzt und von einem Anbieter zur Verfügung gestellt
- **Community Cloud:** die Infrastruktur wird von mehreren Institutionen geteilt, die ähnliche Interessen haben; eine solche Cloud kann von einer dieser Institutionen oder einem Dritten betrieben werden
- **Hybride Cloud:** mehrere Cloud Infrastrukturen, die für sich selbst eigenständig sind, über standardisierte Schnittstellen gemeinsam genutzt

Quelle: Standardisierungsstelle NIST (National Institute of Standards and Technology)



Servicemodelle

- **Infrastructure as a Service (IaaS):** IT-Ressourcen wie z. B. Rechenleistung, Datenspeicher oder Netze als Dienst, Cloud-Kunde bezahlt für diese virtualisierten und standardisierten Services und baut darauf eigene Services zum internen oder externen Gebrauch
- **Platform as a Service (PaaS):** stellt eine komplette Infrastruktur bereit und bietet dem Kunden auf der Plattform standardisierte Schnittstellen an, die von Diensten des Kunden genutzt werden; z. B. Mandantenfähigkeit, Skalierbarkeit, Zugriffskontrolle, Datenbankzugriffe, etc. als Service zur Verfügung gestellt werden; keinen Zugriff auf die darunterliegenden Schichten (Betriebssystem, Hardware); auf der Plattform eigene Anwendungen laufen lassen

Quelle: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html



- **Software as a Service (SaaS):** Angebote von Anwendungen im Cloud-Bereich, fallen in diese Kategorie; Angebotsspektrum sind hierbei keine Grenzen gesetzt; Beispiele wären Kontaktdatenmanagement, Finanzbuchhaltung, Textverarbeitung oder Kollaborationsanwendungen

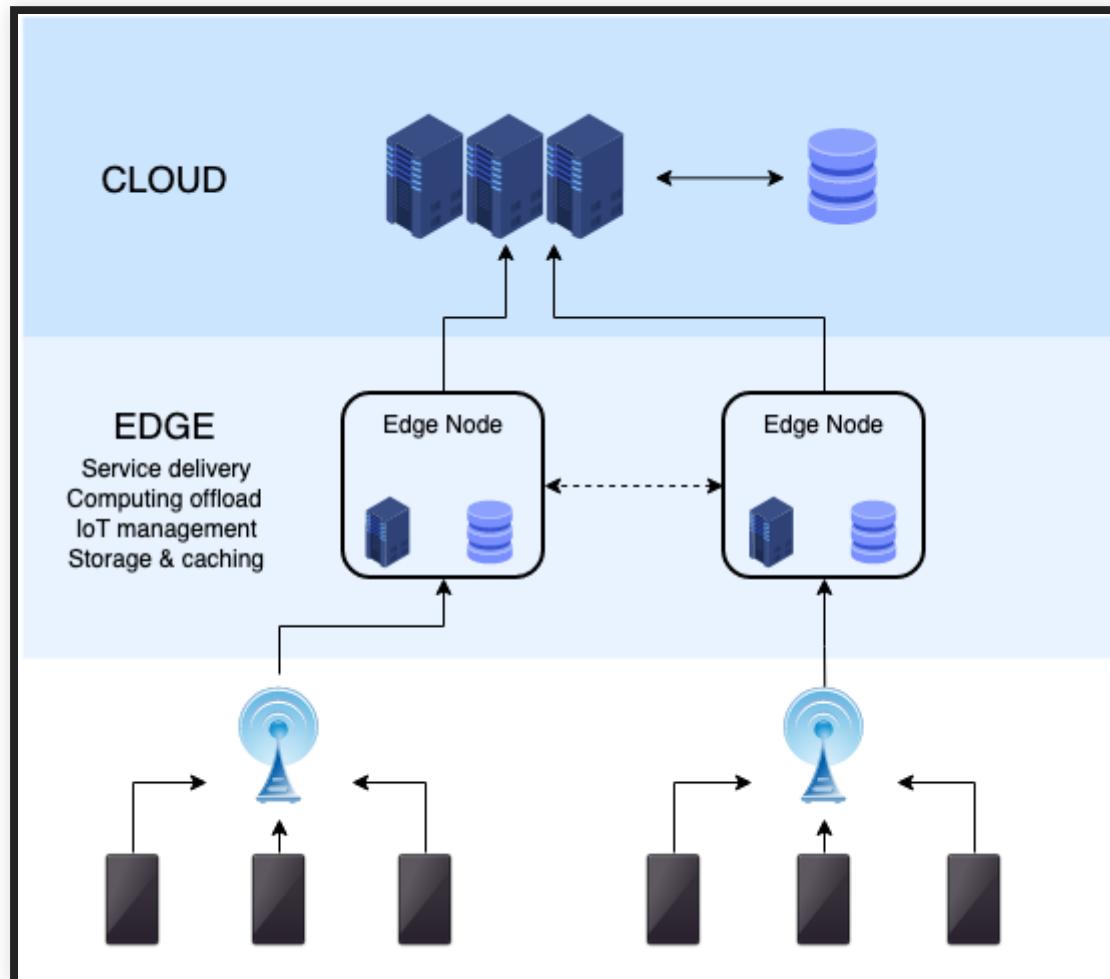
Quelle: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Empfehlungen-nach-Angriffszielen/Cloud-Computing/Grundlagen/grundlagen_node.html



EDGE-COMPUTING

- Edge Computing ist Computing, das nahe am physischen Standort der Nutzende oder der Datenquelle stattfindet.
- Rechendienstleistungen näher zu diesen Standorten gebracht
- Nutzenden profitieren von schnelleren, zuverlässigeren Services und Unternehmen von der Flexibilität des Hybrid Cloud Computing
- Edge Computing ist eine der Methoden, mit der ein Unternehmen einen gemeinsamen Pool an Ressourcen über eine Vielzahl an Standorten verteilen kann

Quelle: <https://www.redhat.com/de/topics/edge-computing/what-is-edge-computing>



Quelle: https://en.wikipedia.org/wiki/Edge_computing



Elemente eines Edge-Netzwerks

- **Anbieter-/Unternehmenskern:** traditionelle Stufen im Besitz von Public Cloud-Anbietern, Telekommunikationsdienstanbietern oder großen Unternehmen, die von diesen betrieben werden. *Sie gehören nicht zum Edge.*
- **Service-Anbieter-Edge:** Stufe befindet sich zwischen dem Kern bzw. den regionalen Rechenzentren und dem Zugang zur letzten Meile, üblicherweise im Besitz eines Telekommunikations- oder Internetdienstanbieters und von diesem betrieben. Von dieser Stufe aus bedient der Anbieter mehrere Kunden.

- **Endbenutzer-Standort-Edge:** Edge-Stufen auf der Endbenutzerseite des Zugangs zur letzten Meile können den Enterprise Edge (z. B. ein Einzelhandelsgeschäft, eine Fabrik, einen Zug) oder den Consumer Edge (z. B. Privathaushalt, Auto) beinhalten.
- **Geräte-Edge:** (Nicht geclusterte) Standalone-Systeme, die Sensoren/Stellantriebe direkt über Nicht-Internet-Protokolle verbinden. Dieser stellt den äußersten Netzwerkrand dar.

Quelle: <https://www.redhat.com/de/topics/edge-computing/what-is-edge-computing>



Fog-Computing

- Fog Computing ist ein Begriff für Computing, das an verteilten physischen Standorten stattfindet, näher an den Nutzern und Datenquellen.
- Fog Computing ist ein Synonym für Edge Computing.
- Außer der Bezeichnung gibt es keinen Unterschied zwischen den beiden Konzepten.

Quelle: <https://www.redhat.com/de/topics/edge-computing/what-is-edge-computing>



CLOUD-DIENSTE IM STREAMING-KONTEXT



Amazon Kinesis ist eine Plattform für das Streamen von Daten in AWS. Es bietet leistungsfähige Services für das einfache Laden und Analysieren von Streaming-Daten und ermöglicht den Aufbau benutzerdefinierter Streaming-Daten-Anwendungen für spezielle Anforderungen. Es bietet drei Services: Amazon Kinesis Data Firehose, Amazon Kinesis Data Streams und Amazon Managed Streaming für Apache Kafka (Amazon MSK).

Quelle: <https://aws.amazon.com/de/streaming-data/>



Amazon MSK ist ein vollständig verwalteter Service, mit dem ganz leicht Anwendungen erstellen und ausführen werden können, die Apache Kafka zur Verarbeitung von Streaming-Daten nutzen. Mit Amazon MSK können native Apache Kafka-APIs verwendet werden, um Data Lakes zu füllen, Änderungen an und von Datenbanken zu streamen sowie Machine Learning und Analyseanwendungen zu unterstützen.

Quelle: <https://aws.amazon.com/de/streaming-data/>



Auf **Amazon EC2** und **Amazon EMR** können individuelle Streaming-Daten-Plattformen installiert und eigenen Speicher- und Verarbeitungsebenen angelegt werden. Wenn Streaming-Daten-Lösung auf Amazon EC2 oder Amazon EMR aufgebaut werden, können Reibungspunkte bei der Bereitstellung der Infrastruktur vermieden werden und es wird der Zugriff auf eine Vielzahl an Speicher- und Verarbeitungssystemen möglich.

Zu den Optionen für Streaming-Daten-Speicherebenen zählen Amazon MSK und Apache Flume. Zu den Optionen für Streaming-Daten-Verarbeitungsebenen zählen Apache Spark Streaming und Apache Storm.

Quelle: <https://aws.amazon.com/de/streaming-data/>



Google Cloud Dataflow ist ein verwalteter Dienst zur Ausführung eines breiten Spektrums an Datenverarbeitungsmustern.

Dataflow ist ein vollständig verwalteter Dienst zum Ausführen von Streaming- und Batch-Datenverarbeitungspipelines. Es bietet eine automatische Bereitstellung und Verwaltung von Rechenressourcen sowie eine konsistente, zuverlässige, genau einmalige Verarbeitung Ihrer Daten. Dataflow unterstützt eine Vielzahl von Anwendungsfällen der Datenverarbeitung, einschließlich Streamanalyse, Echtzeit-KI, Sensor- und Logdatenverarbeitung sowie andere Workflows in Bezug auf die Datentransformation.

Quelle: <https://cloud.google.com/dataflow/docs/about-dataflow?hl=de>



Echtzeitverarbeitung in Azure

- Azure Stream Analytics
- HDInsight mit Spark Streaming
- Apache Spark in Azure Databricks
- HDInsight mit Storm
- Azure-Funktionen
- WebJobs in Azure App Service
- Apache Kafka Streams-API

Quelle: <https://learn.microsoft.com/de-de/azure/architecture/data-guide/technology-choices/stream-processing>

1.6 THEMEN DER VORLESUNG



Allgemeine Kapitel

- **Kapitel 1: Message- und Event-Broadcasting**
Kommunizieren zwischen Ereignisquellen und -senken über proprietäre Ansätze und Message/Event Broker.
- **Kapitel 2: Low-Code-Tools**
Verarbeitung von Daten mit minimalen Programmieraufwand
- **Kapitel 3: Streaming-APIs und Reactive-Programming**
Asynchrone Programmierlogik zur Verarbeitung von Ereignissen
- **Kapitel 4: Enterprise Integration Patterns**
Integration verschiedener Endpunkte basierend auf Patterns unter Verwendung von Apache Camel



- **Kapitel 5: Data Stream Processing**
Verarbeitung von großen Datenmengen in Streaming-Plattformen
- **Kapitel 6: Complex Event Processing**
Vertiefung von Data Stream Processing, Ableiten von höheren, wertvollereren Wissen in Form von komplexen Ereignissen
- **Kapitel 7: Query Languages**
Data Stream Processing und Complex Event Processing mit Abfragesprachen wie KSQL und Siddhi
- **Kapitel 8: Time Series Databases**
Metriken in geeigneten Datenbanken persistieren



Anwendung am Beispiel

Es wird ca. vier Hacking-Veranstaltungen geben, in dem die Themen zur direkten Anwendung kommen.

1. Betrieb und Anbindung von Message Broker in verschiedenen Beispielen
2. Realisierung von Anwendungen mit Apache Camel und Message Brokern, z.B. Spring-REST-Dienste sowie Reactive-Programming
3. Realisierung von Flink- od. Spark-Anwendungen zur Verarbeitung von Ereignissen
4. Verwendung von Query Languages und Zeitseriendatenbanken