Vorlesung

Cloud- und Web-Technologien

(CS3140)

Stromverarbeitung mit Flink

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

Nr	Thema	
1	Einleitung	
2	Einführung in das Semantic Web, RDF und SPARQL	 Datenmodell
3	Die Semantic Web-Ontologiesprachen RDFS und OWL	
4	Multiplattform-Entwicklung mit Kotlin	 Multiplattform
5	Fortgeschrittene Themen mit Kotlin	
6	Einstieg in Cloud Computing, Hadoop	 Backend
7	Operatoren der relationalen Algebra in Hadoop	
8	Datenverarbeitung mit Pig	
9	Einführung in Spark und Flink	
10	**Stromverarbeitung mit Flink**	
11	Knotenzentrische Algorithmen mit Flink	
12	HTML und CSS	Web
13	Browserprogrammierung mit JS/JQuery und Serverprogrammierung mit PHP Hypertext Preprocessor	
14	Zusammenfassung und Ausblick	

Stromverarbeitung - Motivation 1/4

- Finanzanalysen
  - Heutzutage: Elektronisches Handeln
    - Handelsvolumen steigt ständig
    - Algorithmisches Handeln:
      Detektierung von vorteilhaften Marktbedingungen
      $\rightarrow$ automatisches Ausführen von Handelstransaktionen
      - **Latenz ist Schlüssel**
- Typische Anfragen für Finanzanalysen
  - Volume-Waited Average Price (VWAP)
    - Verhältnis des gehandelnden Wertes zum totalen Volumen gehandelt über einen bestimmten Zeithorizont (typischerweise 1 Tag)
    - 5-Minute Rolling Average
      - VWAP für 5 Minuten (zur Anzeige gemittelter Werte etc.)
  - Vergleich der Durchschnitte eines Branchensektors mit dem eigenen Portfolio über die Zeit
  - Implementierung von Modellen basierend auf quantitativer Analyse

Stromverarbeitung - Motivation 2/4

- Netzwerk Monitoring
  - Rapides Ansteigen des Netzwerkvolumens
  - Aufgaben
    - Entdeckung von fehlerhaften Komponenten zur Minimierung der Ausfallzeiten
    - Überwachen der Auslastung der Komponenten
    - Detektierung von merkwürdigen Netzwerkverkehr (Verdacht auf Hackerangriffen etc.)
  - Maßgeschneiderte Lösungen sind teuer
  - Generische Infrastruktur für diese Art von Monitoring Applikationen basierend auf Stromverarbeitung
    - Unterstützung von Ad-Hoc Anfragen

Stromverarbeitung - Motivation 3/4

- Sensornetzwerke/Internet of Things
  - Analyse des Datenstroms von Sensoren,

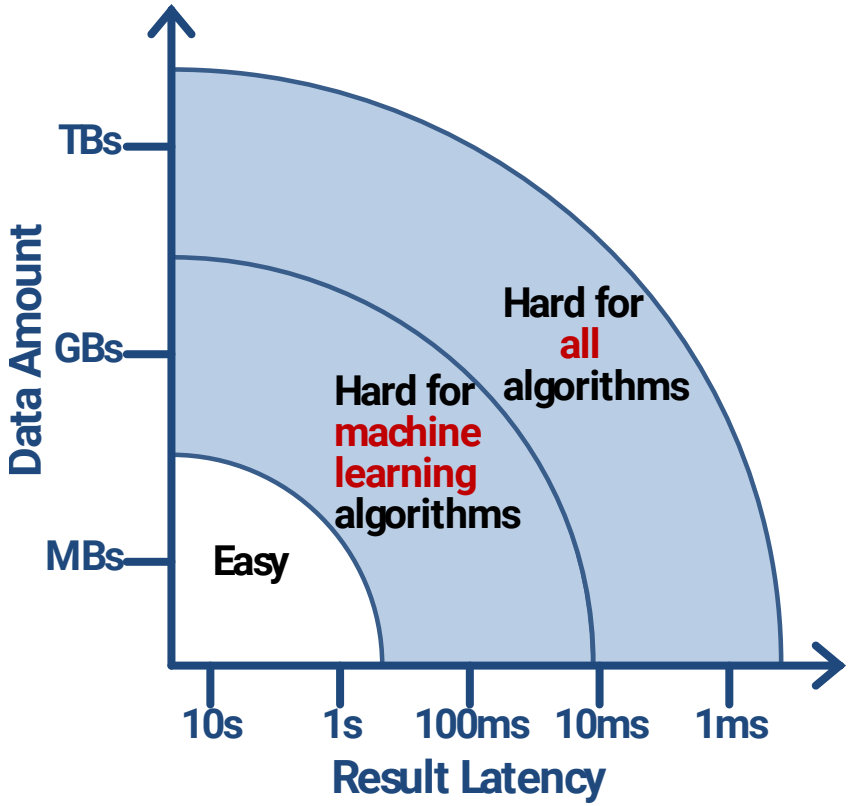
    Beispiele:
    - Militärische Applikationen:
      "Real-Time Command and Control"
    - Medizinische Überwachung von Patienten
    - Habitat Monitoring
      - Erkennung von Notfallsituationen
      - Smart-Home-Anwendungen
    - Industrie 4.0-Anwendungen
      - Überwachung und Steuerung des Herstellungsprozesses von Produkten

Stromverarbeitung - Motivation 4/4

- Real-Time Decision Support
  - Zu langsame Turnaround-Zeit traditioneller Data Warehouses
  - "Business Activity Monitoring" (BAM)
    - Analysen und Präsentationen über zeitrelevante Geschäftsprozesse in Organisationen
- Detektion von Betrug/Schwindel
  - Ausgeklügelter übergreifender Betrug
  - Real-time
- Online Gaming
  - Detektierung von Mogeln
  - Überwachung des Quality of Service

Designraum von Big Data-Systemen

- Performanz versus algorithmische Ausdrückbarkeit



Datenbank versus Stromverarbeitung

- Datenbanksysteme
  - Zumeist recht statische Daten
  - One-Time Queries
  - "Feuern" der Anfragen auf die Daten
  - "Speichere und frage an"
  - Fokus: Nebenläufiges Lesen und Schreiben, effiziente I/O, Maximierung des Transaktionsdurchsatzes, ACID-Eigenschaften der Transaktionen, Historische Analysen
- Stromverarbeitung
  - zumeist flüchtige Daten, die nicht (alle/unverarbeitet) gespeichert werden
    - Datenraten überschreiten oft den Festplattendurchsatz
  - kontinuierliche Anfragen
  - "Feuern der Daten auf die Anfragen"
    mit inkrementellen Aktualisieren des Resultates

Datenstrom

- (möglicherweise) unendlich lange Sequenz von (Tupel, Zeitstempel)-Paaren
  - (Tupel, Zeitstempel)-Paare werden nur angehängt und können **nicht** gelöscht werden
  - Zeitstempel definieren eine totale Ordnung der Tupel im Strom
- Zumeist sofortige Verarbeitung des Datenstromes ($\rightarrow$ kontinuierliche Anfragen)
- Teilweise Archivierung des Datenstromes für spätere Analysen
  - Meist Speicherung nur eines Teilausschnittes des Datenstromes
    - Aggregation von Daten
    - Speichern bei wichtigen Ereignissen (z.B. bei Überschreiten von Thresholds)

Kontinuierliche Anfragen

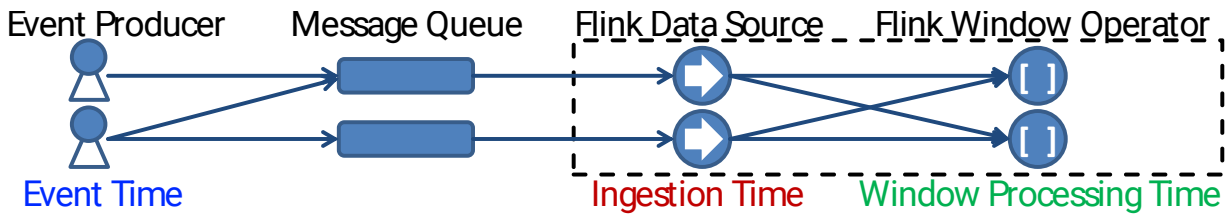
- Resultat einer kontinuierlichen Anfrage ist unbeschränkter Strom
  - und **keine** endliche Relation
    wie bei One-Time-Anfragen (in DBMS)
- Ströme sind unbeschränkt

  $\rightarrow$ Verarbeitung der Anfragen nur auf beschränkten Teilausschnitten - sogenannten **Fenstern** - der Ströme
  - Wende `Window`-Operator an, um den Strom periodisch in Teilausschnitte für die weitere Verarbeitung aufzuteilen

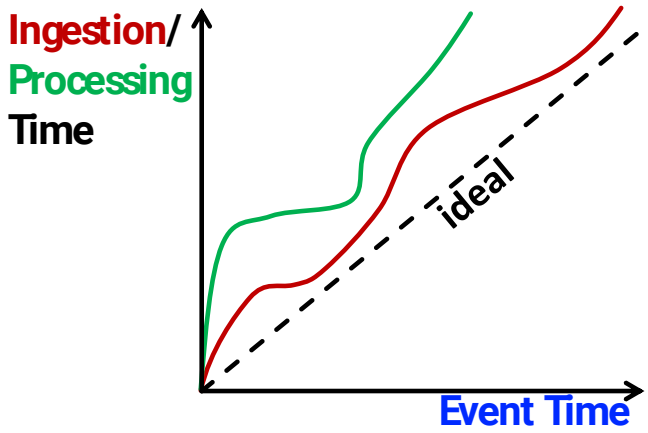
Time Window

- Aufteilung des Stromes in Teilausschnitte der Daten aus Zeitintervallen gleicher Größe
  - z.B für Aggregation über einen gewissen Zeitraum
    - z.B. Durchschnittstemperatur der letzten Stunde
  - Aktualisierung des Fensterresultates in regelmäßigen Abständen

Zeitarten



1. **Processing Time**: Zeitstempel werden vergeben sobald Tupel bei dem das Fenster verarbeitende Rechner ankommen
   - 1-Minute-Fenster verarbeiten die Tupel exakt für 1 Minute
2. **Event Time**: Es gelten Event-Zeitstempel im Fenster
   - Zeitstempel von Log-Einträgen, Sensordaten etc.
3. **Ingestion Time**: Zeitstempel werden vergeben sobald Tupel von Flink entgegengenommen werden



Vorteile Event-Time

- Entkopplung der Bereitstellungsgeschwindigkeit der Events von der Verarbeitungsgeschwindigkeit
- Verhindert falsche Ergebnisse
  1. beim Eintreffen der Events in falscher Reihenfolge (in verteilten Szenarien),
  2. bei der Verarbeitung von historischen Daten mit maximaler Geschwindigkeit,
  3. Rückstau von Events und
  4. Verzögerung bei Wiederherstellungen im Fehlerfall
- Aber: Erfordert Out-Of-Order Tupel-Verarbeitung

  $\Rightarrow$ Aktualisierung der Fensterergebnisse notwendig

Codebsp. zum Setzen der Zeitart

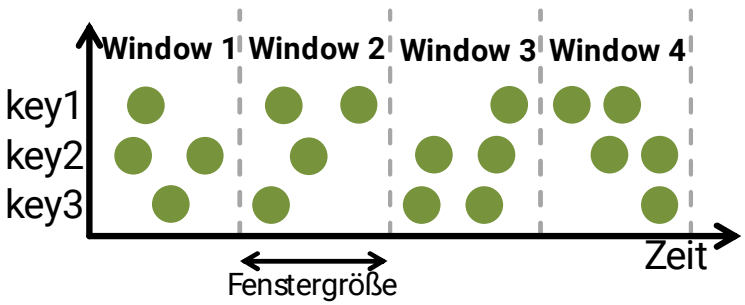
```
final StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();

DataStream<MyEvent> stream = env.addSource(
    new FlinkKafkaConsumer09<MyEvent>(topic, schema, props));

stream
    .keyBy( (event) -> event.getUser() )
    .window(SlidingEventTimeWindows.of(Time.seconds(10), Time.seconds(5)))
    // alternatively:
    // .window(SlidingProcessingTimeWindows.of(Time.seconds(10),Time.seconds(5))
    .timeWindow(Time.hours(1))
    .reduce( (a, b) -> a.add(b) )
    .addSink(...);
```

Count Window

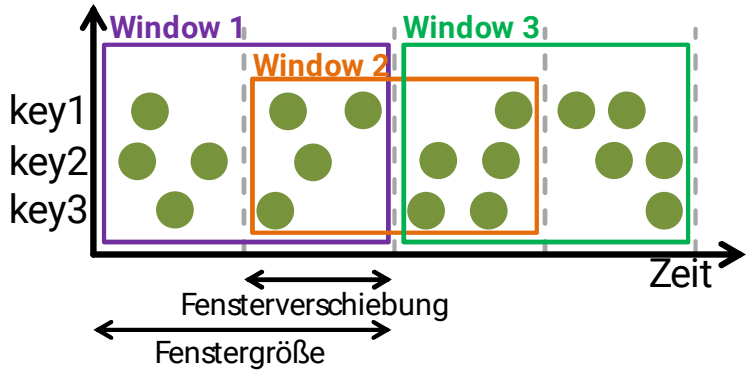
- Aufteilung des Stromes in Teilausschnitte der Daten gleicher Größe/mit derselben Anzahl Tupel
  - z.B. für Aggregation über die gleiche Anzahl von Events
  - Abhängigkeit der Aktualisierungszeitpunkte des Fensters von der Bereitstellungsgeschwindigkeit der Events

Apache Flink API – Window Assigners 1/2 (Nur `DataStream`)

- Tumbling Windows
  - Verarbeitung nicht überlappender Ausschnitte der Daten (als Time oder Count Window)

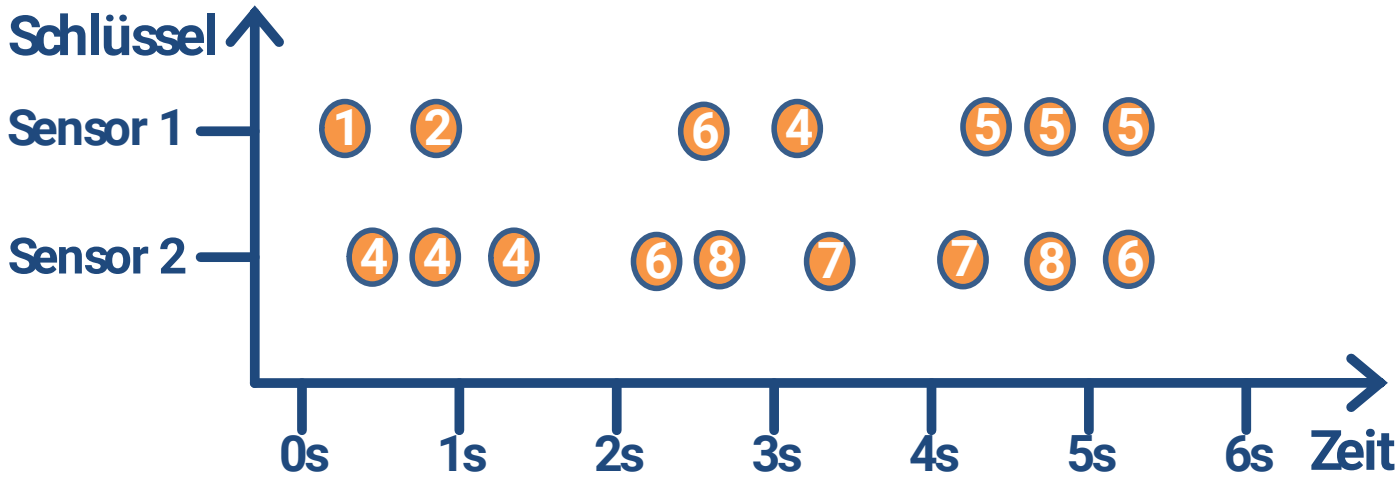


- Sliding Windows
  - Verarbeitung gleitender und überlappender Ausschnitte der Daten (als Time oder Count Window)



Übung Tumbling/Sliding Windows

- Wie lauten die Durchschnittswerte zweier Sensoren der Fenster der Art
  - Tumbling Windows (mit 2s Fenstergröße)?
  - Sliding Windows (mit 4s Fenstergröße und 2s Fensterverschiebung)?

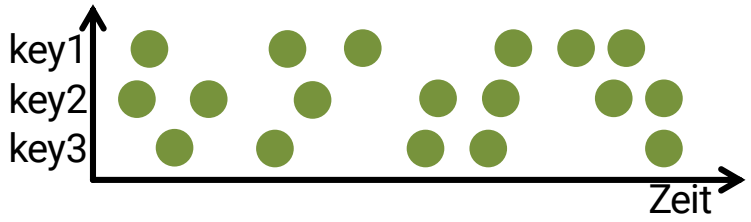


Codebsp. für Tumpling/Sliding Window × Time/Count Window

```
// Stream of (sensorId, carCnt)
val vehicleCnts: DataStream<Int, Int> = ...

val tumblingCnts: DataStream<Int, Int> = vehicleCnts
  // key stream by sensorId
  .keyBy(value -> value.f0)
  // tumbling time window of 1 minute length
  .timeWindow(Time.minutes(1))
  // alternative to timeWindow:
  // tumbling count window of 100 elements size
  // .countWindow(100)
  // compute sum over carCnt
  .sum(1)

val slidingCnts: DataStream<Int, Int> = vehicleCnts
  .keyBy(value -> value.f0)
  // sliding time window of 1 minute length and 30 secs trigger interval
  .timeWindow(Time.minutes(1), Time.seconds(30))
  // alternative to timeWindow:
  // sliding count window of 100 elements size and 10 elements trigger interval
  // .countWindow(100, 10)
  .sum(1)
```

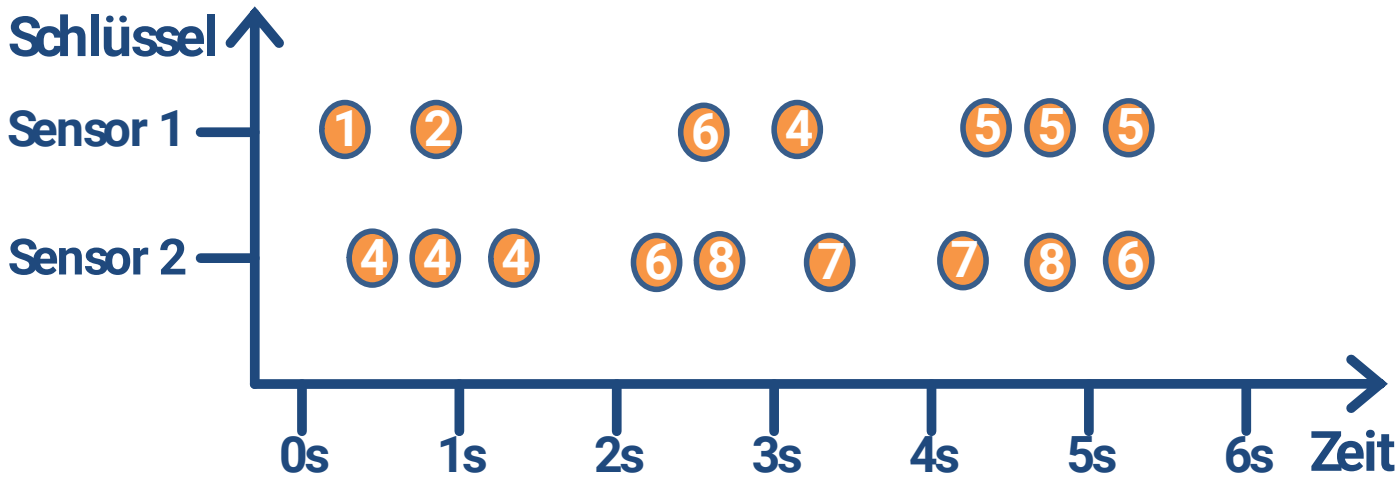
Apache Flink API – Window Assigners 2/2 (Nur `DataStream`)

- Session Windows
  - Gruppiert Events bzgl. Aktivität
  - Fenster schließt falls für eine gewisse Zeit keine Events empfangen werden
- Global Windows
  - Alle Events mit demselben Schlüssel landen in ein globales Fenster
  - nur sinnvoll bei Angabe eines benutzerdefinierten Triggers (zum Sammeln des Fensterinhaltes)



Übung Session/Global Windows

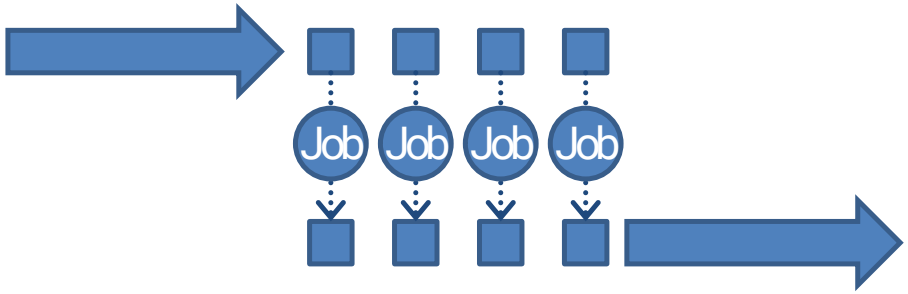
- Wie lauten die Durchschnittswerte zweier Sensoren der Fenster der Art
  - Session Windows (Schließen nach >1s Inaktivität)?
  - Global Windows?



Verarbeitung von kontinuerlichen Anfragen

- Mini-Batches (Apache Spark)

```
while(true){
// get tuples of next window
// issue batch computation
}
```



  - leicht verzögerte Verarbeitung
  - einfach zu implementieren
  - einfache Konsistenz & Fehlertoleranz
  - Schwierige Aktualisierung bei verspäteten Tupeln (bei Event Time)

- Nativ Iterativ (Apache Flink)

```
while(true){
// process next tuple
}
```

  - nahezu "Real-Time"

Sicherungspunkte

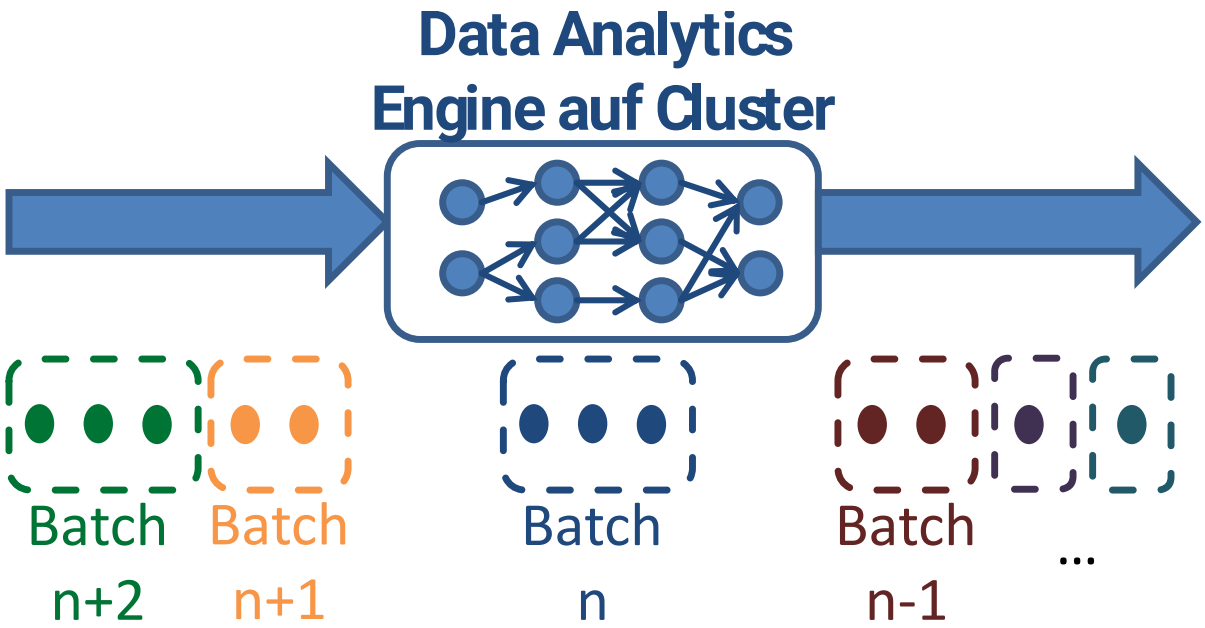
- Sicherungspunkt enthält den Zustand der Stromverarbeitung zu einem Zeitpunkt t, so dass dieser Zustand später wieder eingespielt & die Abarbeitung ab t fortgesetzt werden kann
  - Wiederanlauf nach Absturz von Komponenten
  - Aktualisierung der laufenden Anwendung
  - Neu-Skalierung des Clusters

Datensicherheit & Verfügbarkeit

- Stelle sicher, dass alle Operatoren alle Events sehen (Apache Storm)
  - "Zumindest" einmal
  - Gelöst durch Wiedereinspielen eines Stromes von einem Sicherungspunkt
  - evtl. inkorrekte Resultate
- Stelle sicher, dass Operatoren keine Duplikate verarbeiten (Apache Flink)
  - "Genau einmal"
  - Mehrere Lösungen dafür
- Stelle sicher, dass der Job (des Mini-Batches) Fehler überlebt (Apache Spark)

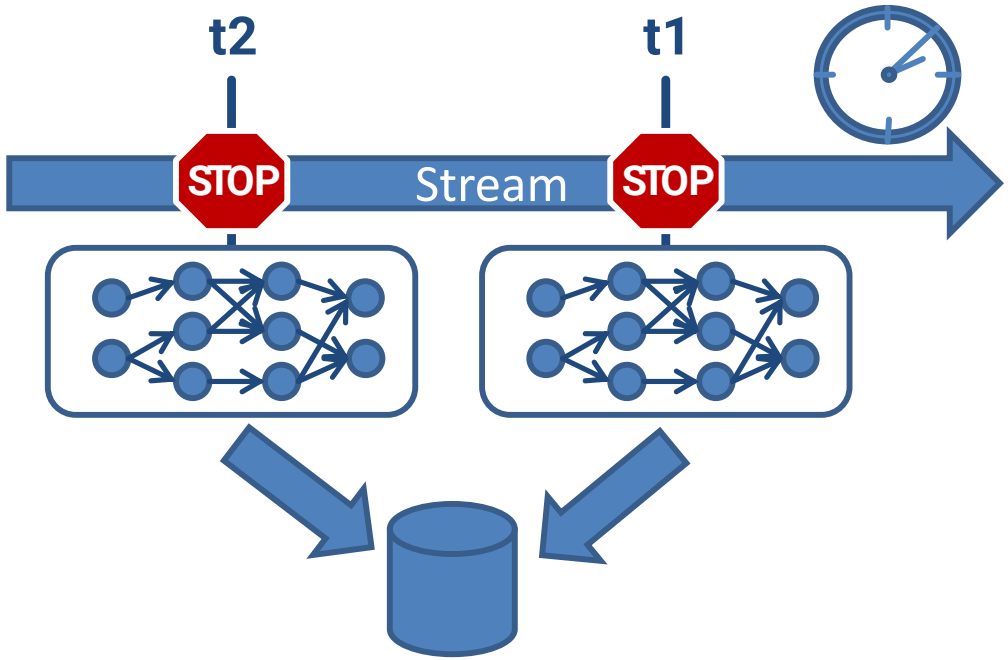
Wiederanlauf bei Mini-Batches

- Im Fehlerfall wiederhole Verarbeitung des aktuellen (& der nachfolgenden) Batches jeweils als "Transaktion"
- Transaktionsrate ist ca. konstant



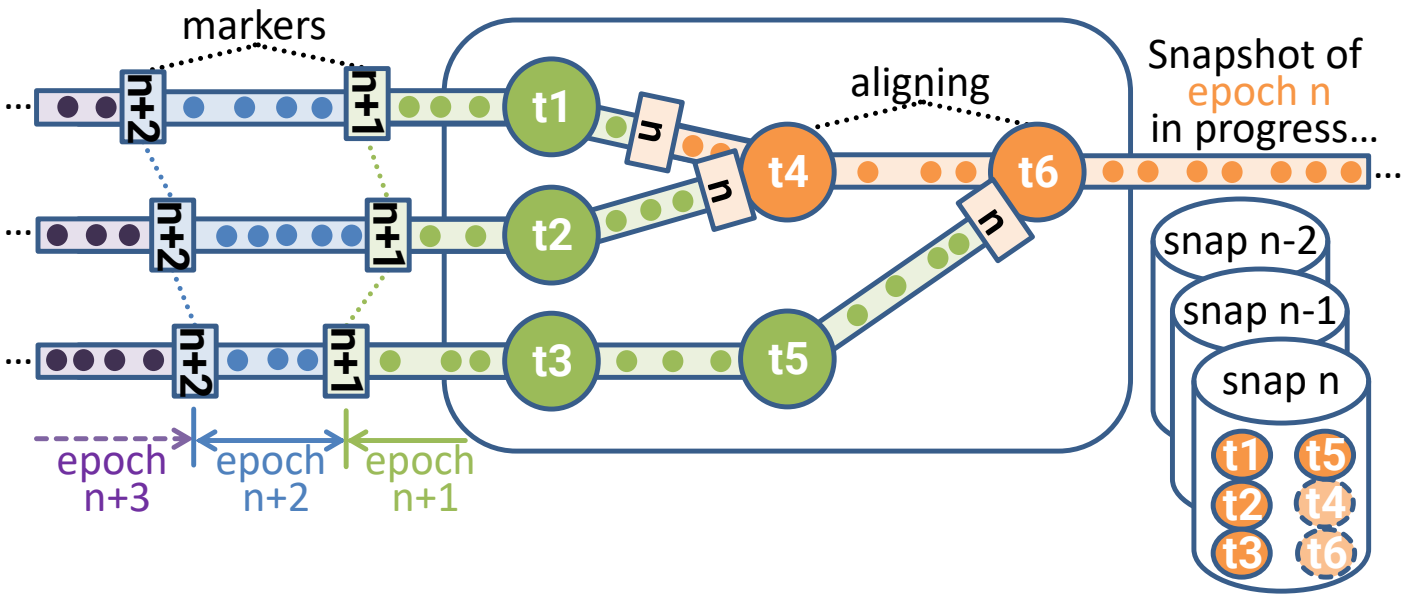
Sicherungspunkte in Flink - naiver Ansatz

- Pausiere Verarbeitung & sammle die Zustände der Operatoren im (verteilten) Verarbeitungsgraph
  - Bei Wiederanlauf spiele Zustände der Operatoren zum (letzten) Zeitpunkt tx ein und wiederhole Events des Stromes ab tx



Asynchrone Sicherungspunkte in Flink

- Loggen der Eingabeströme
- Marker zum Auslösen des Speicherns von Sicherungspunkten der Operatoren
  - Speichern der Zustände wenn an jeder Eingabekante ein Marker desselben Sicherungspunktes



Zusammenfassung

- Datenstrom & kontinuierliche Anfragen
  - Fenster als Teilausschnitt der zu verarbeitenden (aktuellen) Daten
    - Mini Batch-Verarbeitung (Spark) versus nativ iterativ (Flink)
    - Time Windows: Def. des Ausschnittes basierend auf Event/Ingestion/Processing Time
    - Count Windows: Def. des Ausschnittes bzgl. letzte x Events
    - Tumbling/Sliding/Session/Global Windows
- Datensicherheit & Verfügbarkeit
  - synchrone versus asynchrone Sicherungspunkte