

Big Data Engineering

Konstruktion datenintensiver Systeme

christian.zirpins@hs-karlsruhe.de

Big Data Modellierung



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



BDE-Termine — Winter 2018

#	KW	Termin	Block	Thema	Raum
	40	8.10.			
1	41	15.10.		Big Data Paradigma	Hörsaal
2	42	22.10.	Batch Layer	Big Data Modellierung	Hörsaal
3	43	29.10.		Big Data Speicherung	Hörsaal
4	44	5.11.		<i>Übung: Hadoop DFS/Thrift</i>	Labor
5	45	12.11.		Batch Verarbeitung	Hörsaal
6	46	19.11.		<i>Übung: Hadoop Map Reduce</i>	Labor
7	47	26.11.	Serving Layer	Big Data Bereitstellung	Hörsaal
8	48	3.12.		<i>Übung: Cascalog</i>	Labor
9	49	10.12.	Speed Layer	Big Data in Echtzeit	Hörsaal
10	50	17.12.		Queues und Stream Verarbeitung	Hörsaal
	51	24.12.			
	52	31.12.			
11	1	7.1.		<i>Übung: Apache Kafka/Storm</i>	Labor
12	2	14.1.		Micro-Batch Stream Verarbeitung	Hörsaal
13	3	21.1.		Lambda Architektur	Hörsaal
14	4	28.1.		Outro / Q&A / Abgabe	Labor

Raumplan

Hörsaal	E 303
Labor	LI137

Kurzer Rückblick

Themen der letzten Vorlesung

- **Der Big Data Begriff**
- **Grenzen der Skalierbarkeit:** Web Analytics mit klassischer DB
- Noch mal ganz von vorn: Was sind eigentlich **Datensysteme**?
- Die **Lambda Architektur**: Ein Paradigma für Big Data
- **Big Data Technologien** und Open Source Projekte

Datensysteme berechnen Funktionen über Daten

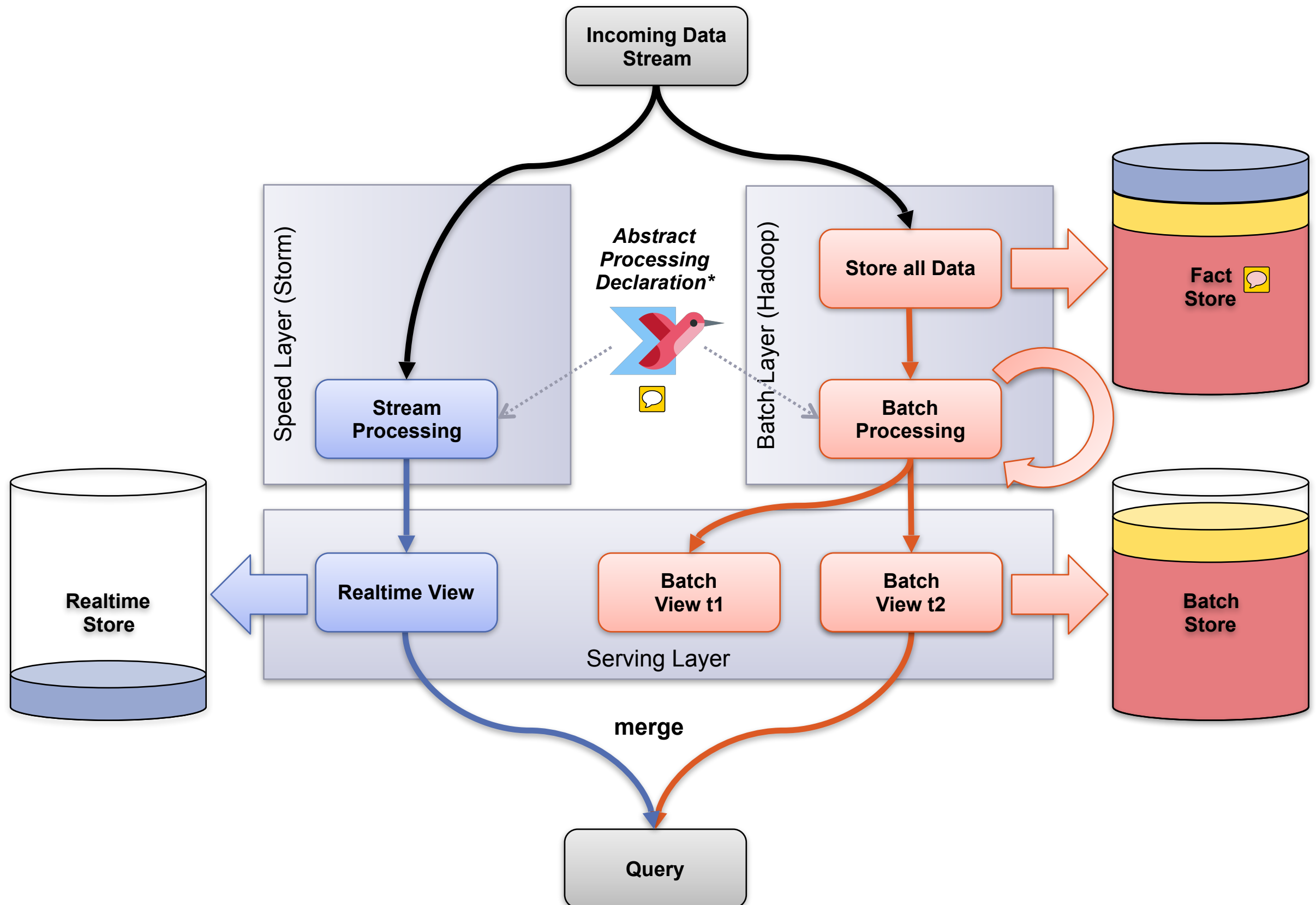
Ein **Datensystem** beantwortet Fragen basieren auf Informationen, die in der Vergangenheit bis in die Gegenwart erworben wurden.

$$Query = Funktion (Datenraum)$$

batch view = *function(all data)* 
realtime view = *function(realtime view, new data)*
query = *function(batch view, realtime view)*


Komplette Lambda Architektur als Gleichungssystem

Lambda Architektur für Datensysteme



Lernziele

Nach dieser Vorlesung können Sie...

- die wichtigsten Eigenschaften von **Daten** **einordnen** 
- **erklären**, wie das Konzept eines **faktenbasierten Datenmodells** die Eigenschaften von Daten bewahrt
- die Struktur eines faktenbasierten Datenmodells durch **Graph Schemas** **ausdrücken**
- Graph Schemas mit **Thrift IDL** **spezifizieren**

Die Eigenschaften von Daten

Einige Begriffe

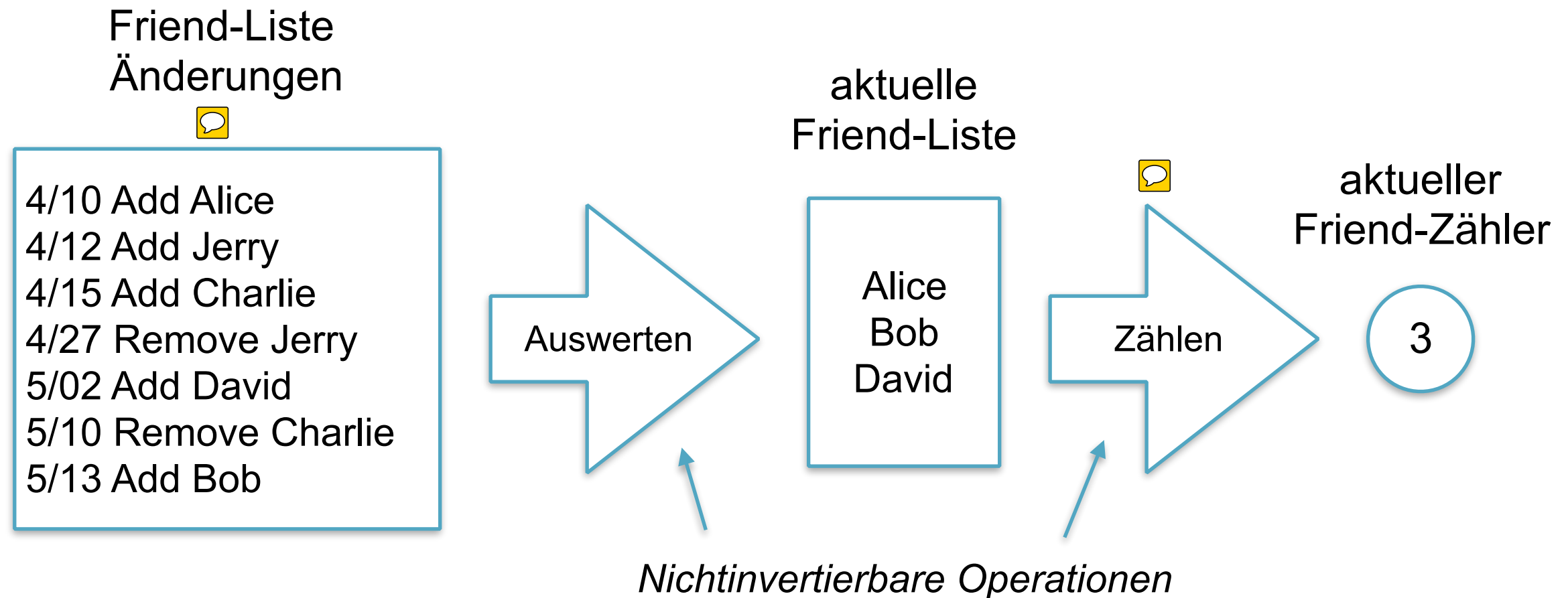
Information ist die allgemeine Wissensbasis im Big Data System

Daten sind Informationen, die nicht abgeleitet werden können

Queries sind Fragen, die mit den Daten beantwortet werden

Views sind Informationen, die von Basisdaten abgeleitet wurden. Sie unterstützen die Beantwortung bestimmter Arten von Fragen 

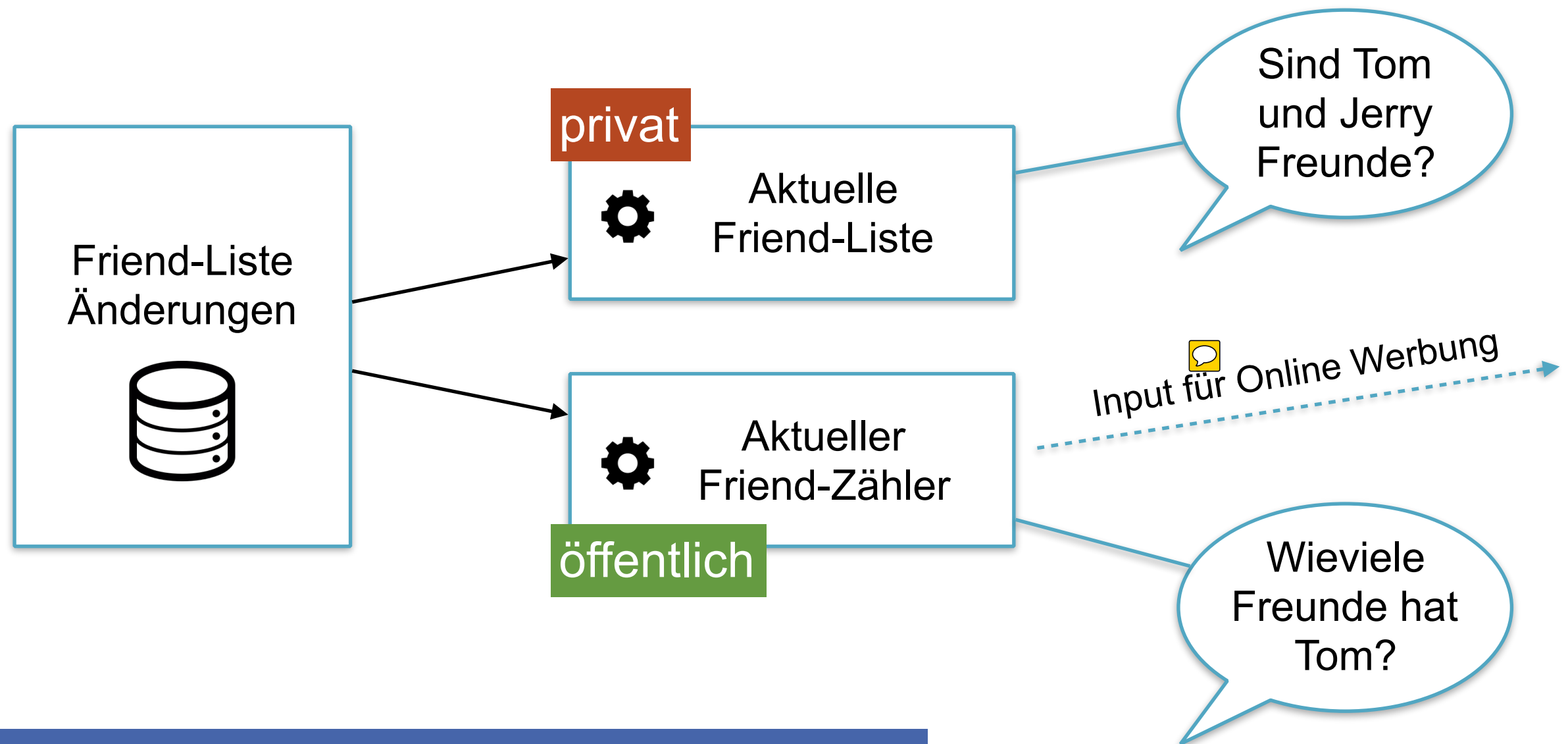
Ein Beispiel: Tom im sozialen Netzwerk



Jede Informationsebene kann aus der davor abgeleitet werden

Frage: Wo sind hier die Daten und wo die Views?

Data Views und Queries



Begriffe sind **relativ**: die Daten des einen können Views des anderen sein

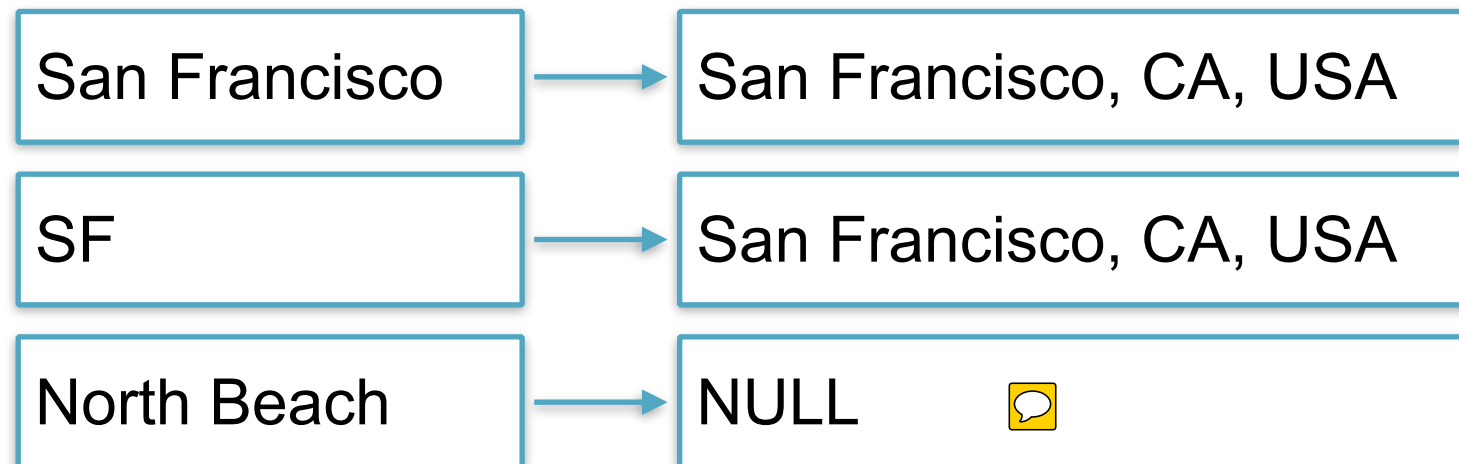
Daten sind "roh" (raw)

Daten sind **roh**, wenn sie ihre ursprüngliche Form haben und nicht verarbeitet wurden. Je **roher** die Daten, desto mehr Fragen kann das System beantworten. 



Die richtigen Rohdaten auswählen

Unstrukturierte Daten sind roher als strukturierte Daten



Semantische Normalisierung
Freiform Informationen in eine
strukturierte Form umformen

Mehr Information führt nicht immer zu roheren Daten

- **Annotierte Texteinträge** sind rohere Daten als **ASCII-Text-Strings** 
- **HTML-Sourcen** sind noch umfangreicher - aber was nützt es?

Frage: was nützen die Annotationen (fett, kursiv, ...)?

Daten sind "unveränderbar" (immutable)

Unveränderbare Daten nicht ändern oder löschen, nur **hinzufügen** 

- **Menschliche Fehler** erzeugen falsche Views, zerstören aber keine Daten. Views werden nach Korrektur einfach neu erzeugt
- **Neue Daten** hinzufügen statt überschreiben - kein Index nötig

Daten unveränderbar machen: **Attributtabellen mit Zeitstempeln**

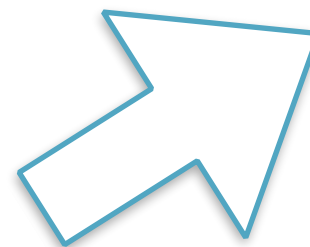
Immutable

Name data		
user id	name	timestamp
1	Alice	2012/03/29 08:12:24
2	Bob	2012/04/12 14:47:51
3	Tom	2012/04/04 18:31:24
4	Charlie	2012/04/09 11:52:30
...

Age data		
user id	age	timestamp
1	25	2012/03/29 08:12:24
2	36	2012/04/12 14:47:51
3	28	2012/04/04 18:31:24
4	25	2012/04/09 11:52:30
...

Mutable

User information					
id	name	age	gender	employer	location
1	Alice	25	female	Apple	Atlanta, GA
2	Bob	36	male	SAS	Chicago, IL
3	Tom	28	male	Google	San Francisco, CA
4	Charlie	25	male	Microsoft	Washington, DC
...





Location data		
user id	location	timestamp
1	Atlanta, GA	2012/03/29 08:12:24
2	Chicago, IL	2012/04/12 14:47:51
3	San Francisco, CA	2012/04/04 18:31:24
4	Washington, DC	2012/04/09 11:52:30
...

Frage: was passiert wenn Tom nach LA zieht?

Daten sind "ewig wahr"

Unveränderbare Daten müssen ewig wahr bleiben.
Ein **Zeitstempel** setzt Daten in Bezug zu ihrer Entstehungszeit.

- Wahre Daten werden nie gelöscht außer bei...
 - **Garbage Collection**: die Daten haben *zu wenig Wert*. 
 - **Regulation**: die Daten *dürfen nicht* gespeichert werden. 

Ein faktenbasiertes Datenmodell

Fakten sind atomar und zeitbezogen



Fakten sind atomar

Rohe
Daten über
Tom

Tom arbeitet für Google
(2012/04/04 18:31:24)

Tom wohnt in San Francisco, CA
(2012/04/04 18:31:24)

Tom wohnt in Los Angeles, CA
(2012/06/17 20:09:48)

Tom ist befreundet mit David
(2012/05/16 19:11:13)

Tom ist befreundet mit Alice
(2012/05/23 22:06:16)

Fakten haben Zeitstempel

Fakten sind **identifizierbar**

PageView Faktum V1

```
struct PageView:  
    DateTime timestamp  
    String url  
    String ip_address
```

Frage: Was passiert, wenn zwei Aufrufe zur gleichen Zeit von der gleichen IP kommen?

```
struct PageView:  
    Datetime timestamp  
    String url  
    String ip_address  
    Long nonce
```

PageView Faktum V2

Fakten sind **identifizierbar**

PageView Faktum V1

```
struct PageView:  
    DateTime timestamp  
    String url  
    String ip_address
```

Frage: Was passiert, wenn zwei Aufrufe zur gleichen Zeit von der gleichen IP kommen?

```
struct PageView:  
    Datetime timestamp  
    String url  
    String ip_address  
    Long nonce
```

Ein **nonce** identifiziert ein Faktum;
Duplikate werden erkannt und
können bei der Verarbeitung
gefiltert werden

PageView Faktum V2



Zusammenfassung

Das faktenbasierte Modell...

- speichert Rohdaten als atomare **Fakten**
- hält Fakten unveränderbar und ewig wahr durch **Zeitstempel**
- macht Fakten mit **nonce** identifizierbar und Duplikaten erkennbar

Eigenschaften des faktenbasierten Modells

Die Masterdatenmenge ...

- ist für jeden **Zeitpunkt** seiner Geschichte abfragbar
 - Zeitstempel können bei der Abfrage berücksichtigt werden
- toleriert **menschliche Fehler**
 - Falsche Fakten werden (tatsächlich) gelöscht; die vorherige Version wird automatisch wirksam
- Kann mit **partiellen Informationen** umgehen
 - Fehlende Attribute erfordern keine NULL Werte, sondern es gibt einfach kein Faktum
- hat die Vorteile **normalisierter** und **denormalisierter Formen**
 - Daten sind normalisiert; Views können denormalisiert sein

Normalisierung

Datennormalisierung meint Daten in einer strukturierten Art zu speichern um Redundanz zu minimieren und Konsistenz zu fördern

Employment		
row id	name	company
1	Bill	Microsoft
2	Larry	BackRub
3	Sergey	BackRub
4	Steve	Apple
...

*Angestelltenliste
denormalisiert mit
Redundanz*

User		
user id	name	company id
1	Bill	3
2	Larry	2
3	Sergey	2
4	Steve	1
...

Company	
company id	name
1	Apple
2	BackRub
3	Microsoft
4	IBM
...	...



*Angestellten- und Arbeitgeberlisten
normalisiert ohne Redundanz*



Graph Schemas



Struktur eines faktenbasierten Datensatzes

Fakten sind entweder *Eigenschaften* von Entitäten oder *Beziehungen* zwischen Entitäten. Ein **Graph Schema** repräsentiert Beziehungen zwischen Fakten und legt ihre Typen fest.

- **Knoten**

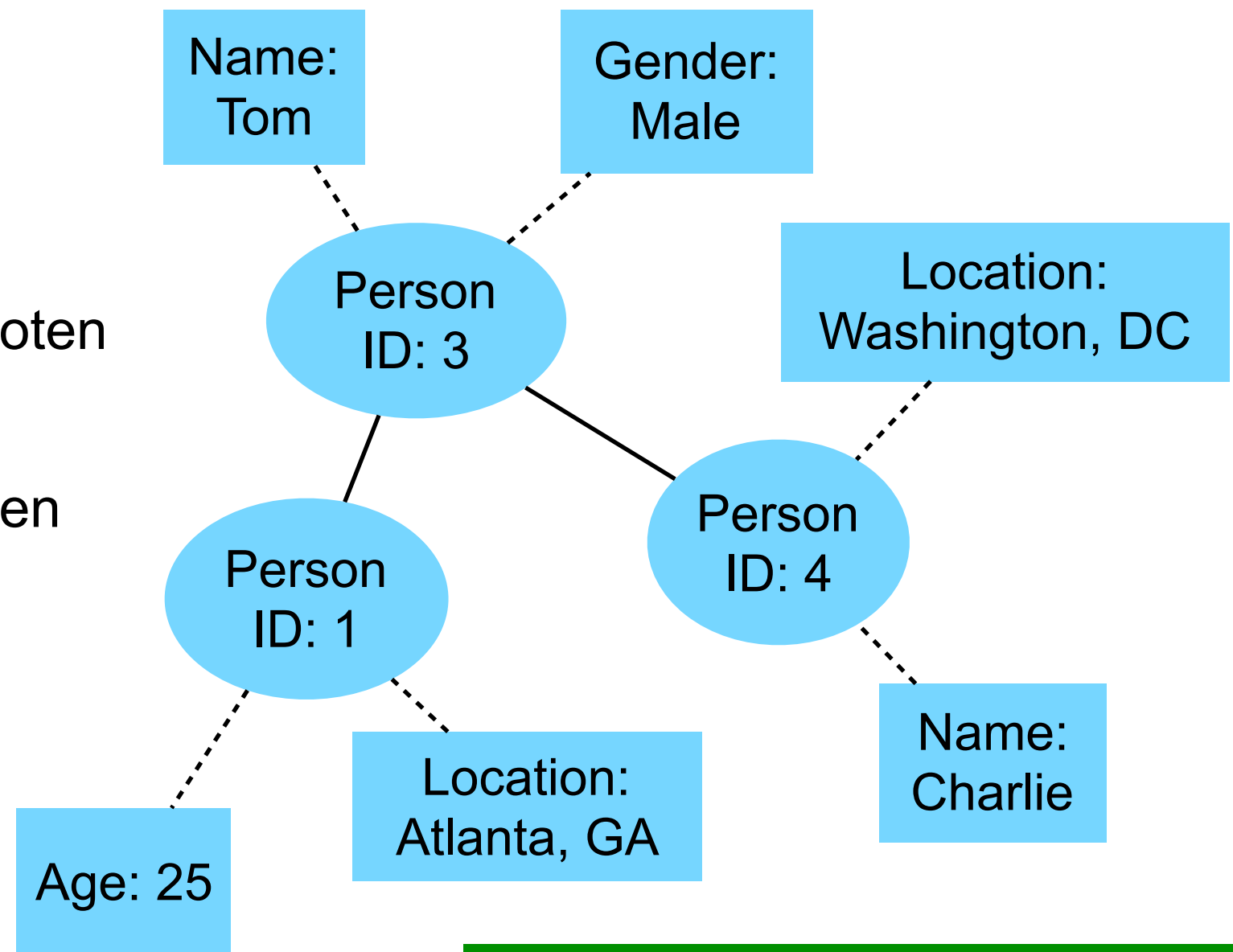
Einheiten des Systems

- **Kanten**

Beziehungen zwischen Knoten

- **Eigenschaften**

Informationen über Einheiten



Frage: Wo sind die Fakten? 

Struktur eines faktenbasierten Datensatzes

Fakten sind entweder *Eigenschaften* von Entitäten oder *Beziehungen* zwischen Entitäten. Ein **Graph Schema** repräsentiert Beziehungen zwischen Fakten und legt ihre Typen fest.

■ Knoten

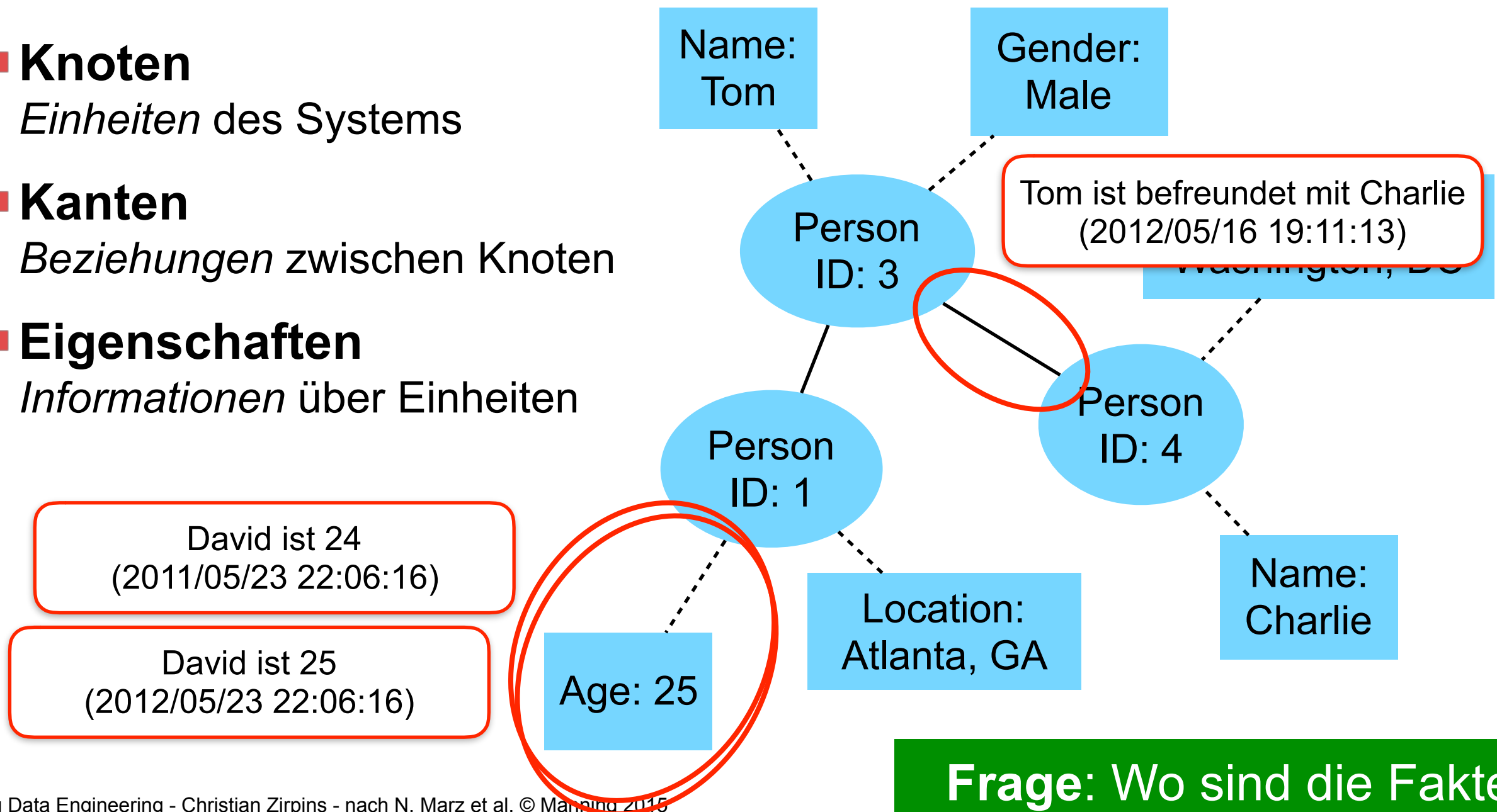
Einheiten des Systems

■ Kanten

Beziehungen zwischen Knoten

■ Eigenschaften

Informationen über Einheiten



Struktur eines faktenbasierten Datensatzes

Das Graph Schema sollte **typgesichert** sein, um die Korrektheit von Fakten in der Masterdatenmenge zu garantieren.



```
{  
  "id": 3,  
  "field": "age",  
  "value": 28,  
  "timestamp": 1333589484  
}
```

```
{  
  "id": 2,  
  "field": "age",  
  "value": 36  
}
```

Verschiedene Fakten in validem JSON

```
{  
  "name": "Alice",  
  "field": "age",  
  "value": 25,  
  "timestamp": "2012/03/29 08:12:24"  
}
```

Apache Thrift

Apache Thrift

Thrift ist ein RPC Framework

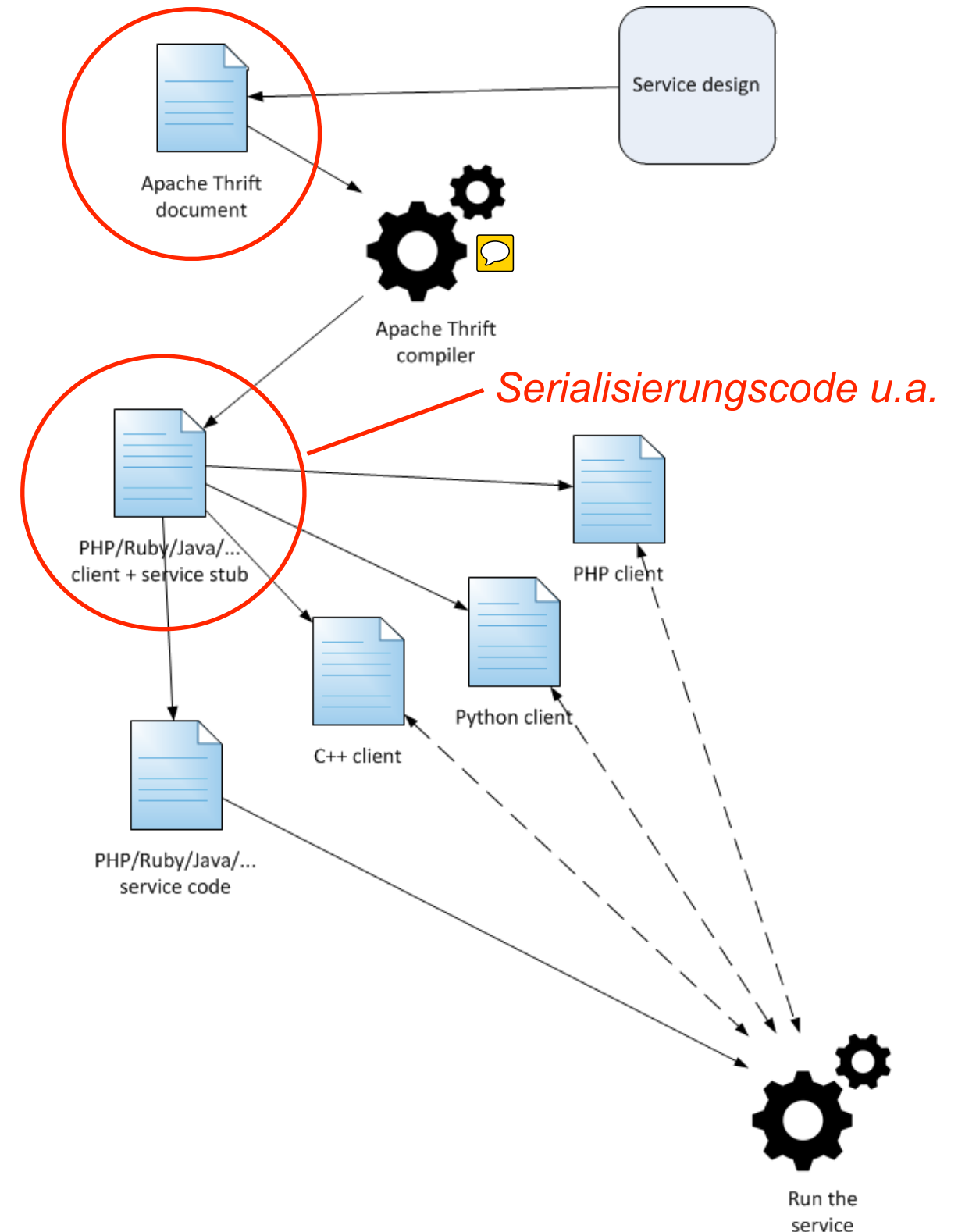
Von Facebook für "scalable cross-language services development"
nun Open Source Apache Projekt

<https://thrift.apache.org>

Thrift IDL beschreibt *Services*
und deren *Datenschema*

Thrift generiert aus IDL *Client* und *Server Stubs* in vielen Sprachen.
Diese können Daten serialisieren,
per Protokoll austauschen und
wieder deserialisieren

Datenschema in Thrift IDL



Thrift Interface Definition Language (IDL)

■ Basistypen

- **bool**: boolean value (true or false), one byte
- **byte**: signed byte
- **i16**: 16-bit signed integer
- **i32**: 32-bit signed integer
- **i64**: 64-bit signed integer
- **double**: 64-bit floating point number
- **binary**: byte array
- **string**: encoding agnostic text or binary string

■ Container

- **list<t1>**: ordered list of elements
- **set<t1>**: unordered set of unique elements
- **map<t1, t2>**: map of unique keys

Thrift Interface Definition Language (IDL)

- **enum**: erstellt einen Aufzählungstyp mit Name-Wert-Paaren
- **struct** ist Kompositions- typ; Felder sind optional oder required und mit id identifiziert
- **union**: wie struct aber mit exakt einem Feld aus mehreren Alternativen
- **typedef**: erzeugt einen alternativen Namen für einen Typ.

```
# This is a valid comment.

/*
 * This is a multi-line comment.
 */

// C++/Java style single-line comments work

enum TweetType {
    TWEET = 2,
    RETWEET
}

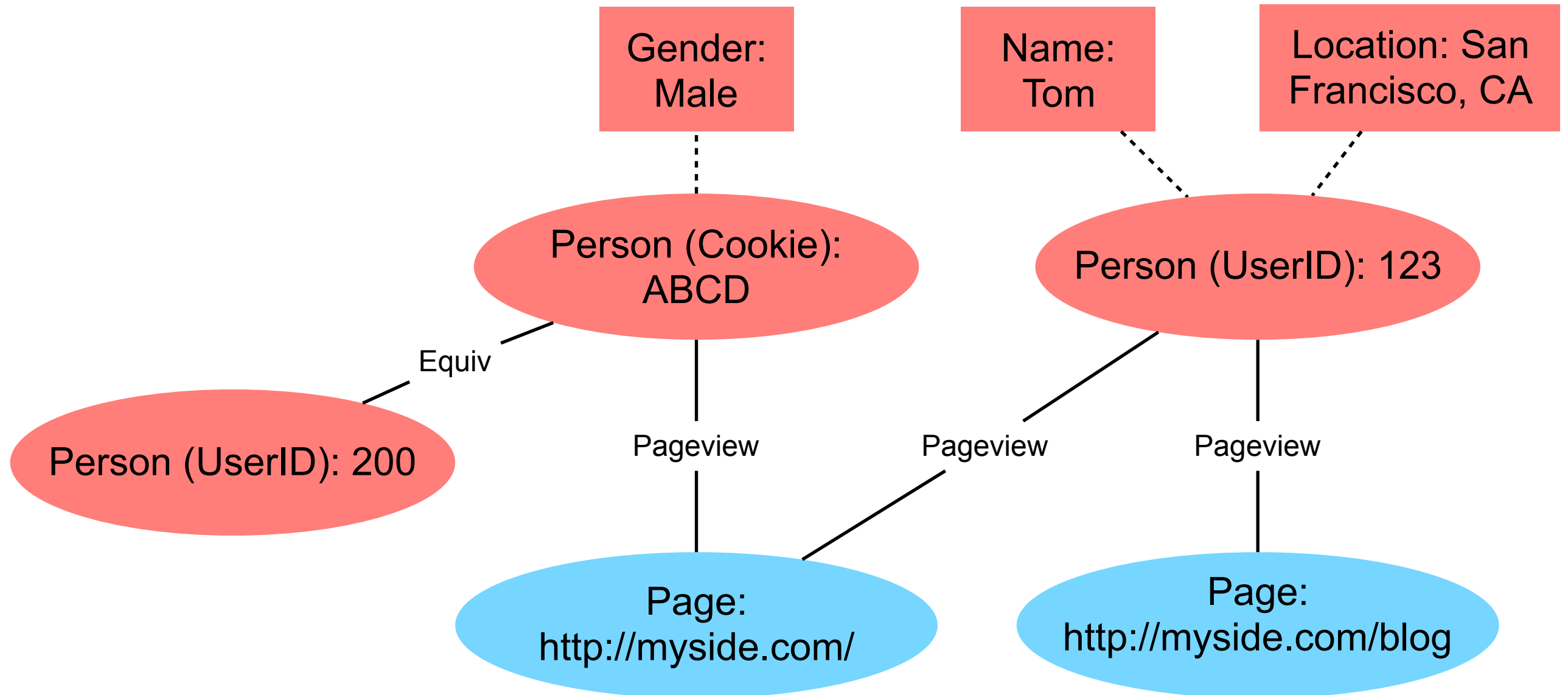
union TweetUser {
    i32 userId;
    string userName;
}

struct Tweet {
    1: required TweetUser;
    2: required string text;
    3: optional Location loc;
    4: optional TweetType tweetType = TweetType.TWEET
    16: optional string language = "english"
}

typedef Tweet ReTweet
```

Graph Schema Implementierung mit Thrift

SuperWebAnalytics Graph Schema Beispiel



Thrift Graph Schema für SuperWebAnalytics

```
union PersonID {  
  1: string cookie;  
  2: i64 user_id;  
}  
  
union PageID {  
  1: string url;  
}  
  
struct Location {  
  1: optional string city;  
  2: optional string state;  
  3: optional string country;  
}  
  
enum GenderType {  
  MALE = 1,  
  FEMALE = 2  
}  
  
union PersonPropertyValue {  
  1: string full_name;  
  2: GenderType gender;  
  3: Location location;  
}  
  
struct PersonProperty {  
  1: required PersonID id;  
  2: required PersonPropertyValue property;  
}
```

Knoten und Eigenschaften

- Personen (`PersonID`) sind durch ID oder Cookie repräsentiert
- `PersonProperty` verbindet Personen mit einer alternativen Eigenschaft (`PersonPropertyValue`)
- Ortsinformationen (`Location`) sind *kohäsiv* (zusammenhängend)

Thrift Graph Schema für SuperWebAnalytics

```
struct EquivEdge {
  1: required PersonID id1;
  2: required PersonID id2;
}

struct PageViewEdge {
  1: required PersonID person;
  2: required PageID page;
  3: required i64 nonce;
}
```

```
struct Pedigree {
  1: required i32 true_as_of_secs;
}

union DataUnit {
  1: PersonProperty person_property;
  2: EquivEdge equiv;
  3: PageViewEdge page_view;
}

struct Data {
  1: required Pedigree pedigree;
  2: required DataUnit dataunit;
}
```

Kanten

Datentypen für Fakten

- PageViewEdge Kanten sind identifizierbar (nonce)
- Metadaten (Pedigree) sind mindestens Zeitstempel (erweiterbar)
- Fakten (Data) sind Änderungen von Kanten oder Eigenschaften

Thrift Graph Schema Evolution

```
struct EquivEdge {
  1: required PersonID id1;
  2: required PersonID id2;
}

struct PageViewEdge {
  1: required PersonID person;
  2: required PageID page;
  3: required i64 nonce;
}

struct LinkEdge { ←
  1: required PageID id1;
  2: required PageID id2;
}
```

```
struct Pedigree {
  1: required i32 true_as_of_secs;
  2: optional string source; ←
}

union DataUnit {
  1: PersonProperty person_property;
  2: EquivEdge equiv;
  3: PageViewEdge page_view;
  4: LinkEdge link; ←
}

struct Data {
  1: required Pedigree pedigree;
  2: required DataUnit dataunit;
}
```

Neue Kanten

Neue Fakten und Metadaten

- *Knoten hinzufügen* ist kein Problem
- *Felder hinzufügen* ist ok; müssen aber optional sein
- *Felder löschen* auch möglich; Feld-ID dann aber nicht mehr nutzbar

Zusammenfassung und Ausblick

Heute haben wir...

- ... die **Eigenschaften von Daten** untersucht
- ... Daten **faktenbasiert** modelliert
- ... Fakten durch **Graph Schemas** strukturiert
- ... Graph Schemas mit **Thrift** implementiert

Als nächstes wollen wir...

- ... das **Hadoop Distributed File System (HDFS)** untersuchen
- ... Daten in HDFS **persistent speichern**

Literatur und Links

- Nathan Marz, James Warren, "Big Data: Principles and best practices of scalable realtime data systems", Manning, 2015, Kapitel 2+3
- Mark Slee, Aditya Agarwal and Marc Kwiatkowski, "Thrift: Scalable Cross-Language Services Implementation", Whitepaper, Facebook, 2007
<https://thrift.apache.org/static/files/thrift-20070401.pdf>
- Diwaker Gupta, "Thrift: The Missing Guide"
<http://diwakergupta.github.io/thrift-missing-guide/>