TDT4305 Big Data-arkitektur
Våren 2022

Dagens tema

- Praktisk informasjon
- Big Data
- System/infrastruktur for Big Data: Intro til Hadoop-økosystemet

Hvem er vi?

- Kjetil Nørvåg

- Hassan Abedi (prosjekt)

TDT4305 Big Data-arkitektur

- **Mål med faget:**
  Gi studentene en innføring i hovedaspekt ved Big Data og Big-Data-system. Emnet tar for seg hovedaspekt ved Big Data. Sentrale tema er rammeverk for Big Data-prosessering (MapReduce, Spark, Storm, etc.), datagruvedrift av Big Data, datastrømmer og analyse av tidsserier, anbefalingssystemer, og analyse av sosiale nettverk.
- **Viktige tema innenfor Big Data *ikke* dekket i detalj av dette faget:**
  Datagruvedrift generelt (TDT4300)
  Maskinlæring/prediksjon (TDT4300 og TDT4173)

Forkunnskapskrav

- Anbefalt forkunnskapskrav
  - TDT4145 Datamodellering og databasesystemer, eller tilsvarende (Som forutsetter TDT4100 Objektorientert programmering og TDT4120 Algoritmer og datastrukturer eller tilsvarende)
- Har med hensikt valgt å ikke forutsette at studentene har hatt TDT4300 Datavarehus/datagruvedrift

Undervisningsopplegg

For å kunne ha eit opplegg som er mest mogleg robust mht. endringar i smittesituasjon og campus-nærvær, vil opplegget til dels vere basert på førehandsinnspelte videoar, med spørsmåls-sesjonar relatert til desse. Forfattarane av læreboka har lagt ut svært gode, profesjonelt innspelte, videoar som dekker pensum i boka, så vi kjem til å basere oss på desse, pluss eigne for tema frå artiklar/tilleggsmateriale. De finn videoane (og læreboka) her: http://www.mmds.org/ Her finn de også foilar laga av lærebokforfattarane, vi kjem i tillegg til å legge ut dei vi sjølve har brukt i tidlegare år.
Forventningane for spørsmåls-sesjonane er at deltakarar på førehand har sett videoar, lese pensum, og sett på foilane. Med tanke på at sesjonane er tenkt å vere interaktive vil det ikkje verte gjort opptak av desse.

Forelesningsplan
(vil vert oppdatert i faginfo gjennom semesteret)

Veke	Tema
2	Faginfo, generell intro til Big Data (ingen videoar til denne forelesninga) Forelesning: Kjetil Nørvåg [ElmasriNavathe] [MMDS, 1]
3	Map-Reduce and the New Software Stack Forelesning: Kjetil Nørvåg MMDS: Video 1-6 [HDFS], [DeanGhemawat], [MMDS, 2], [HDG]
4	Spark Forelesning: Kjetil Nørvåg [Zaharia], [LearningSpark]
5	Gjesteforelesning frå Sikt/Uninett Geir Solskinnsbakk og Emil Henry Flakk Tittel: Defensive Security and Big Data
6	Finding similar items Q&A: Kjetil Nørvåg MMDS: Video 1-6 [MMDS, 3]
7	Mining Data Streams Q&A: Kjetil Nørvåg Forelesning: Heri Ramampiaro (opptak) [MMDS, 4]
…	…

Pensum (vil bli oppdatert)

- Hovedsaklig artikler (ingen egnede bøker som passer fagets "visjon" ☹), ligg i zip-fil på Blackboard:
  - *Fundamentals of Database Systems, 7th ed,* Pearson, s. 911-916 (Big Data), Elmasri & Navathe, 2016.
  - *The Hadoop Distributed File System*, Schvachko et al., Proc. of MSST, 2010.
  - *MapReduce Simplified Data Processing on Large Clusters*, (Unntatt kap. 5 og appendix A), Dean and Ghemawat, Proc. of OSDI, 2004.
  - *Hadoop: The Definitive Guide*, s. 19-37 (MapReduce), White, O'Reilly, 2015.
  - *Hadoop: The Definitive Guide*, s. 185-201 (How MapReduce Works), White, O'Reilly, 2015.
  - *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*, Zaharia et al., Proc. of NSDI, 2012. (Unntatt sek. 3.2, 5.2, 6-8)
  - *Learning Spark*, s. 23-60, Karau et al., O'Reilly, 2015
  - *AsterixDB: A Scalable, Open Source BDMS (unntatt Section 5 og 6),* Alsubaiee et al., PVLDB, 2014.
  - *Data Ingestion in AsterixDB,* Grove and Carey, Proc. of EDBT, 2015.
  - *Storm @Twitter*, Toshniwal et al., Proc. of SIGMOD, 2014.
  - *Mining of Massive Datasets*, Leskovec et al., utvalde tema, sjå faginfo for oppdatert liste.
- "NTNU-produserte" foiler og øvinger er også pensum
- NB! Listen over foreløpig, oppdatert info vil dere til enhver tid finne i faginfo på Blackboard

Prosjekt/øvingsopplegg

- Prosjekt som tel 25% av totalkarakter i faget, meir info neste foil ☺
- Frivillige øvingar:
  - Frivillige øvingar som dekker «teoridelen», m/løysingsskisse, er lagt ut på Blackboard
  - Basert på erfaringar frå 2021 (lav deltaking) prioriterer vi ikkje å bruke ressursar på organisert aktivitet relatert til desse i år, men spørsmål kan stillast på Piazza
- Test-spørsmål til kap. 4/9/10, laga av Heri: Meint som hjelp medan de les pensum, svara finn de i pensum
- Oppfordrar til bruk av forumet på Piazza (og dei fleste spørsmål finn ein svar på ved å Google litt...☺)
  - Skriv på engelsk om det er spørsmål relatert til prosjektet

Prosjektdelen av TDT4305

- Fokus på ”hands-on” erfaring fra del 1 av faget (”systemdelen”)
- Grupper på *max* to personer
- To-delt:
  - (Del 0: Installering av Spark på egen maskin og gjennomgang av «quick tour»)
  - Del 1: Bli kjent med Spark, bruke Spark til enkel analyse av data
  - Del 2: Utvikle en applikasjon på BigData-rammeverk (Spark)
- Programmeringsspråk: Java, Scala eller Python
- Planlegg publisering av prosjektet ca. veke 6, første innlevering veke 9 (analyse av datasett), og andre innlevering veke 12 (applikasjon og rapport)
- Årets prosjekt blir tilsvarende tidligere prosjekter, eksempel:

2016:

1) Analyse av Foursquare datasett
2) Sentiment-analyse på tweets i Twitter-datasett (positiv/negativ/nøytral, og mer avansert analyse som f.eks. aggregert over byer i USA), med fokus på ytelse og skalerbarhet

2017:

1) Exploration and analysis of AirBnB dataset
2) Sentiment analysis of AirBnB review, important terms, finding similar AirBnB listings

Eksamen

- Eksamen: Sannsynlegvis heimeeksamen, 2 timar, med bokstavkarakter, tel 75% på totalkarakter NTNU vil publisere oppdatert informasjon om dette 27. januar
- Gamle eksamensoppgåver er lagt ut på Blackboard.
- NB!
  - Eksamen for 2020 var utan bokstavkarakter og ikkje nødvendigvis representativ for korleis eksamen i 2022 vert (og av same grunn vart det ikkje laga utfyllande løysingsforslag for publisering)
  - Eksamen for 2021 var 4-tiimars heime-eksamen med alle hjelpemiddel tillatne
  - Eldre eksamenar er representative tema-messig, men ein kan forvente litt endring i type spørsmål mht. at det også i 2022 sannsynlegvis vert heime-eksamen med alle tilgjengelege hjelpemiddel

BIG
DATA
INFORMATION
TECHNOLOGIES
MANAGEMENT
PETABYTES
ANALYSIS
RESEARCH
SEARCH
INTERNET
LEARNING
WORLD
BASED
STORAGE
LARGE
SOFTWARE
SANNAS
VOLUME
PROCESS
GOVERNMENT
SETS
MILLION
DEVELOPMENT
TECHNOLOGY
SCIENCE
FUTURE
NEW
BILLION
PARALLEL
INITIATIVE
ECONOMIC
SIZE
EXABYTES
DATABASES
PROCESSING
ANALYTICS
SIMULATIONS
VISUALIZATION
FRAMEWORK
BUSINESS
EFFECTIVE
BETWEEN
DEFINITION
STATISTICS
USING
MASSIVE
FACTOR
SYSTEMS
ONLINE
DAY
MANAGE
WORLDWIDE
MAKING
INTEGRATION
RESULTS
YEARS
SOCIAL
DRIVE
REPLICATION
HANDLE
SOURCES
LIMITS
CHALLENGES
COMPLEX
ANNOUNCED
PAST
CAPACITY
HAND
ONE
OUT
INCLUDE
NETWORKS
HYPOTHESIS
END
MOST
NEARLY
MAPREDUCE
CRITIQUES
STATE
CRITIQUE
PEOPLE
TRAFFIC
USE
CURRENTLY
TERABYTES
RELATED
MARKET
TIME
SENSOR
VARIETY
WOULD
PROJECT
ANNUAL
NOW
DISTRIBUTED
RELEVANT
USED
PER
PRIVACY
INTERNATIONAL
IMPLEMENTATION
BECOMES
SECOND
NATIONAL
FLOW
INTELLIGENCE
APPROACH
MANY
APPROACHES
PROCESSES
PARADIGM
RECORDS
COST
FUNDING
INSIGHT
TOOLS
CENTER
ARCHITECTURE
SET
DEPARTMENT
QUERIES
HIGH
USERS
DETERMINE
WELL
HIGHER
ALGORITHMS
COMPANIES
AMOUNT
REAL
STRUCTURE
WORK
RATE
APPLICATIONS
INFRASTRUCTURE
TERMS
YEAR
PRIVATE
LESS
HUGE
EVERY
VARIOUS
TIMES
SHARED
CAPABILITIES
COMPARED
SUCCESSFUL
NEEDED
USES

Terminologi

- *Strukturerte data*: veldefinerte felt, t.d. representert i tabellar
- *Ustrukturerte data*: typisk ”av menneske, for menneske”, t.d. tekst-meldingar
- *Semi-strukturerte data*: typisk sjølv-skildrande (med taggar), t.d. XML og JSON
- *Batch-orientert*: å køyre ein serie med dataprogram utan menneskelig inngripen (dvs. motsett av ”interaktiv”)
- *Nær-sanntid* (near-realtime): kort forseinking mellom når data vert tilgjengeleg og dei vert behandla
- *Sanntid* (realtime): garantert tid for svar frå data vert tilgjengeleg til dei vert behandla
- *Straum-data* (streaming data): kan sjåast på som ”ordna sekvens av instansar”, t.d. sensor-data eller Twitter-meldingar, med høg innfrekvens, og der ein prosesserer data umiddelbart og utan å ha heile sekvensen tilgjengeleg (heller ikkje det som har kome tidlegare)

Big Data: Definisjonar/karakteristikk

"**Big data** is a broad term for data sets so **large** or **complex** that traditional data processing applications (i.e., DBMS) are inadequate for capturing/storing/managing/analyzing. " – *McKinsey*

"***Big data*** *is high-****volume****, high-****velocity*** *and high-****variety*** *information assets that demand* ***cost-effective, innovative*** *forms of information processing for* ***enhanced insight and decision making****." – Gartner*

Dei 3 (og 5 :) V'ane

- *Volume*: enorme mengder data kontinuerlig generert, både brukargenererte og maskingenererte (t.d. sensorar)
  Ikkje i seg sjølv nok til at det er "Big Data"!
- *Velocity*: hastigheit på datagenerering og prosessering, ofte i sanntid, t.d. umiddelbar deteksjon av trendar på Twitter, ofte straum-data
- *Variety*: variasjon i typar data, kombinasjon av strukturerte og ustrukturerte
- *Veracity*: "sannferdigheit", dvs. kvalitet og nøyaktigheit, stort volum kan kompensere for dette
- *Value*: (potensiale for) verdi av data

Viktige teknikkar (fagområde) i Big Data

- Massiv parallellitet
- Datalagring
- Nettverk
- Databasar/spørjingar
- Tungrekning
- Datagruvedrift
- Maskinlæring
- Informasjonsgjenfinning
- Visualisering
- (Personvern)
- …

”Gamle” teknikkar, så kvifor *Big Data* no?

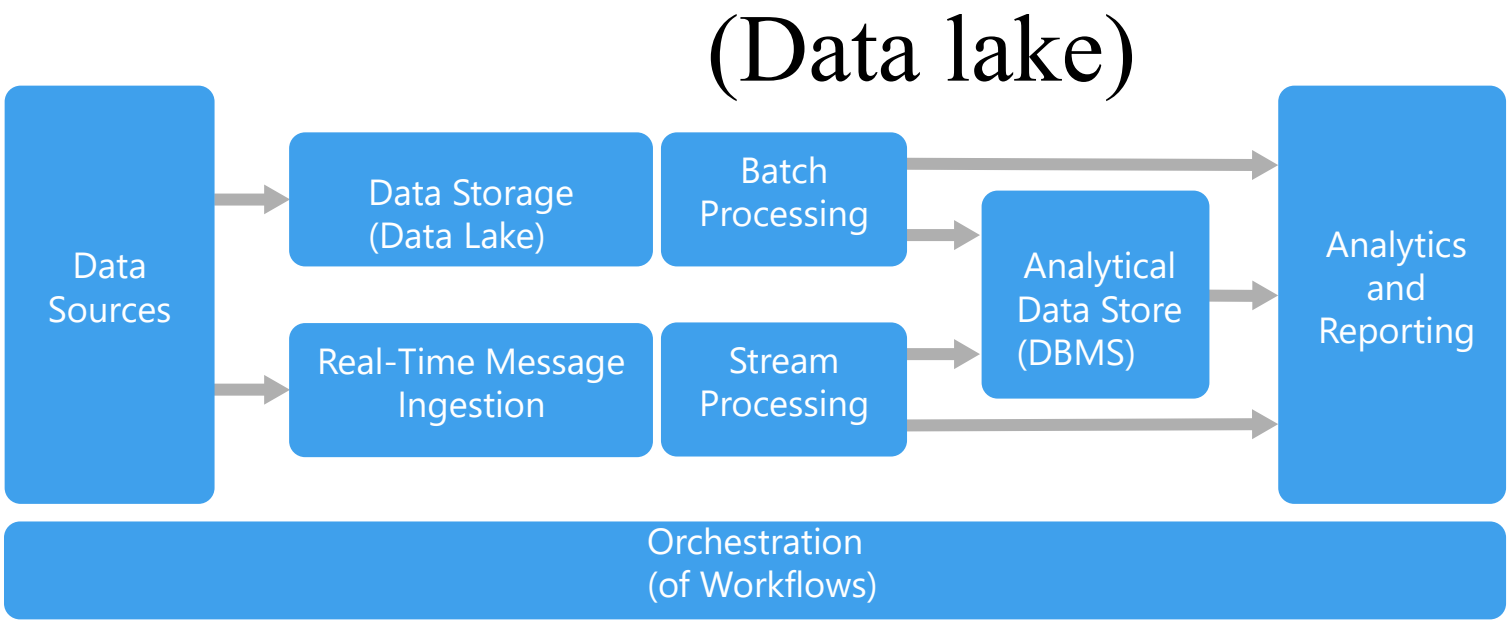
Kvifor Big Data no?

- Meir data innsamla (og lagra):
  - Web (inkl. web-aviser)
  - Sosiale mediar, brukargenererte data
    - Facebook: 1 mrd. kommentarar og 60 mrd Messenger-meldingar pr. dag (april 2016)
    - Twitter: 500 mill tweets per dag (2022), dvs. 6000/minutt!
    - I tillegg: alle data om brukar-aksessar/interaksjon!
  - Transaksjonsdata, t.d. frå telekom
  - Internet of Things (sensorar, RFID, etc.)
- Åpen kjeldekode
  - GNU, Linux, Hadoop, Spark...
- Standard maskin- og programvare (hyllevare)

Eksempel på Big Data-applikasjonsområde (av mange)

- Anbefalingssystem
- Finans-analyse
- Monitorering av sosiale media
- Personaliserte søk
- Trafikk-navigasjon
- Helse (monitorering etc.)

Big Data: Generell arkitektur

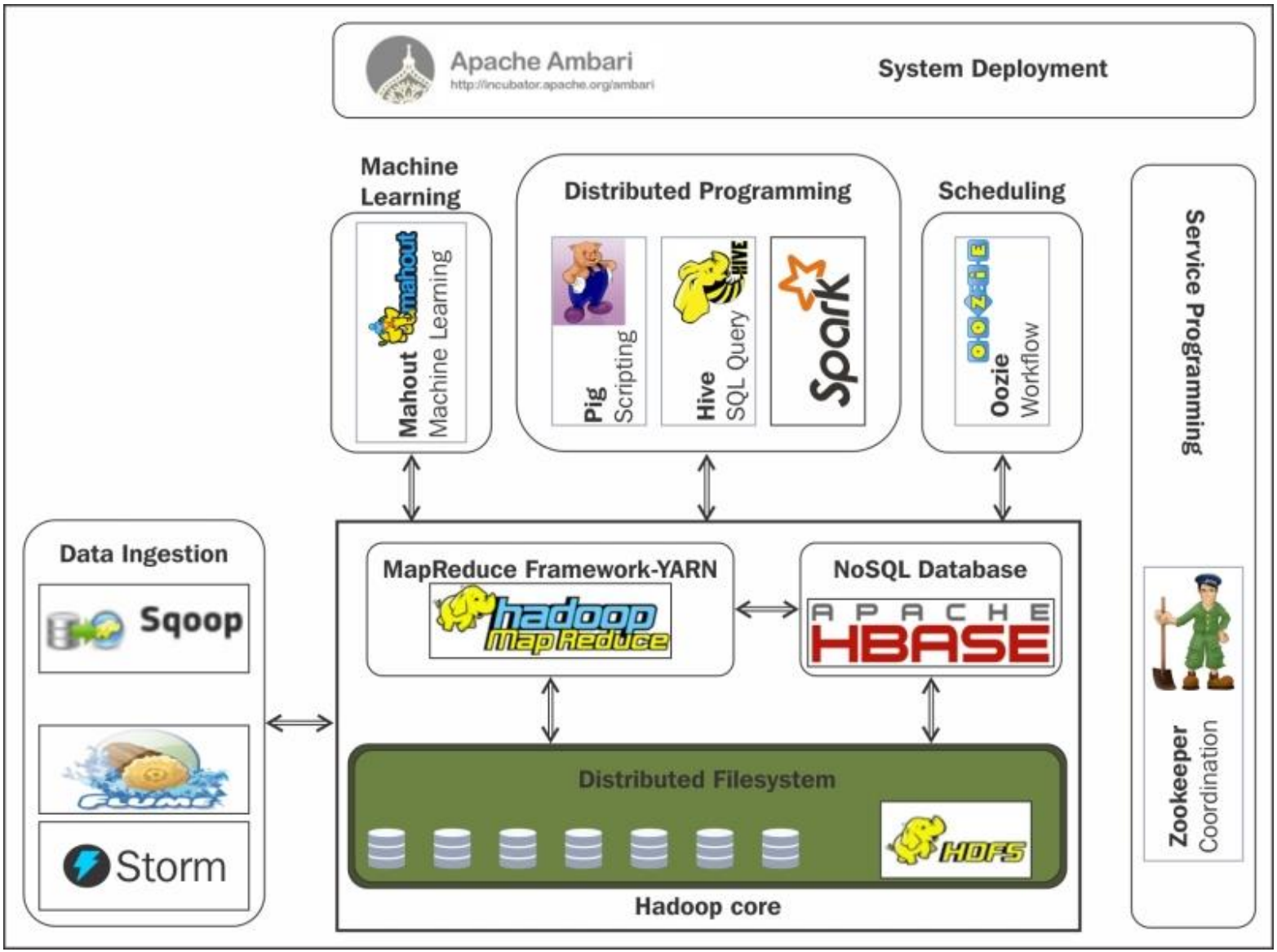


(https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data)

Big Data: Infrastruktur og verktøy

- Hadoop starta i 2006 av Doug Cutting (Yahoo!)
- Verktøy i Apache Hadoop-økosystemet mykje brukt
  - Også som del av kommersielle produkt (t.d. IBM)
  - Hadoop: Distribuert filsystem, ressurs-handsaming (Yarn), prosessering (MapReduce)
  - Mange relaterte Apache-prosjekt som utgjer økosystemet rundt Hadoop
- Apache Spark
- Open-source, men levert i forskjellige ”distribusjonar” (med dyr support ☺) frå kommersielle firma
  - Cloudera, Databricks, pluss «gamle» firma som IBM og HP
- Kan også køyrast i sky-tenestar som t.d. Amazon AWS og Microsoft Azure

Hadoop-økosystemet



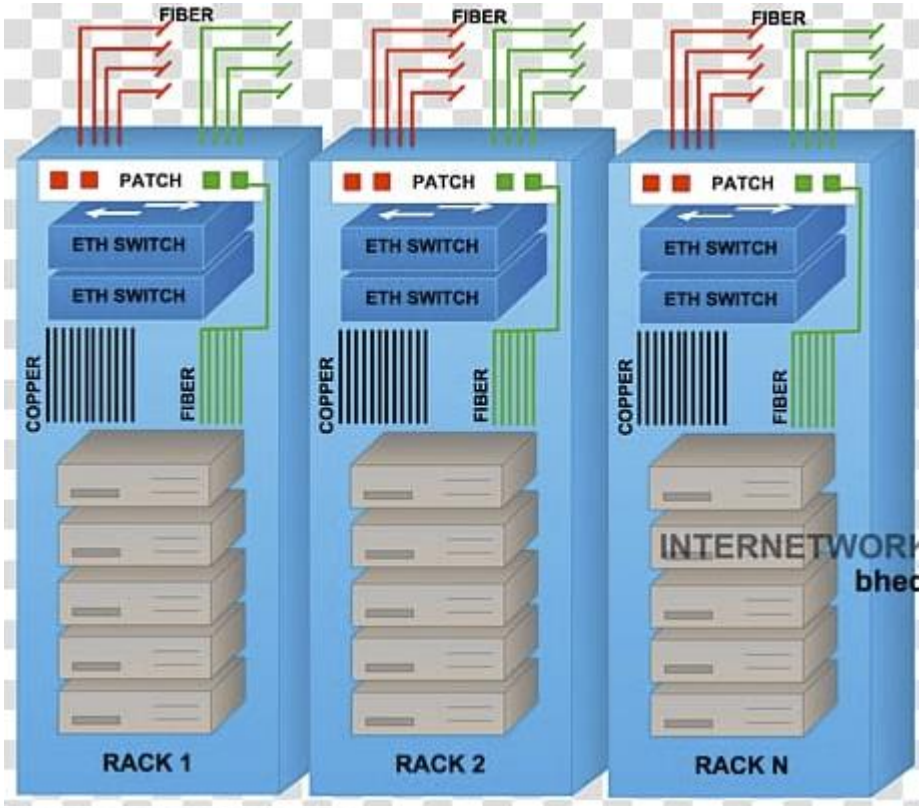
(Frå *Hadoop Essentials*)

Interessante datasett fritt tilgjengeleg

- Wikipedia pageviews
- Fullstendig Wikipedia historie
- Sample av tweets vha. Twitter Firehose
  - 1% sample av alle tweets
- Web Science-metode: ”Skrap” nett-stadar sjølv, t.d. allrecipes.com, tripadvisor.com, booking.com etc.

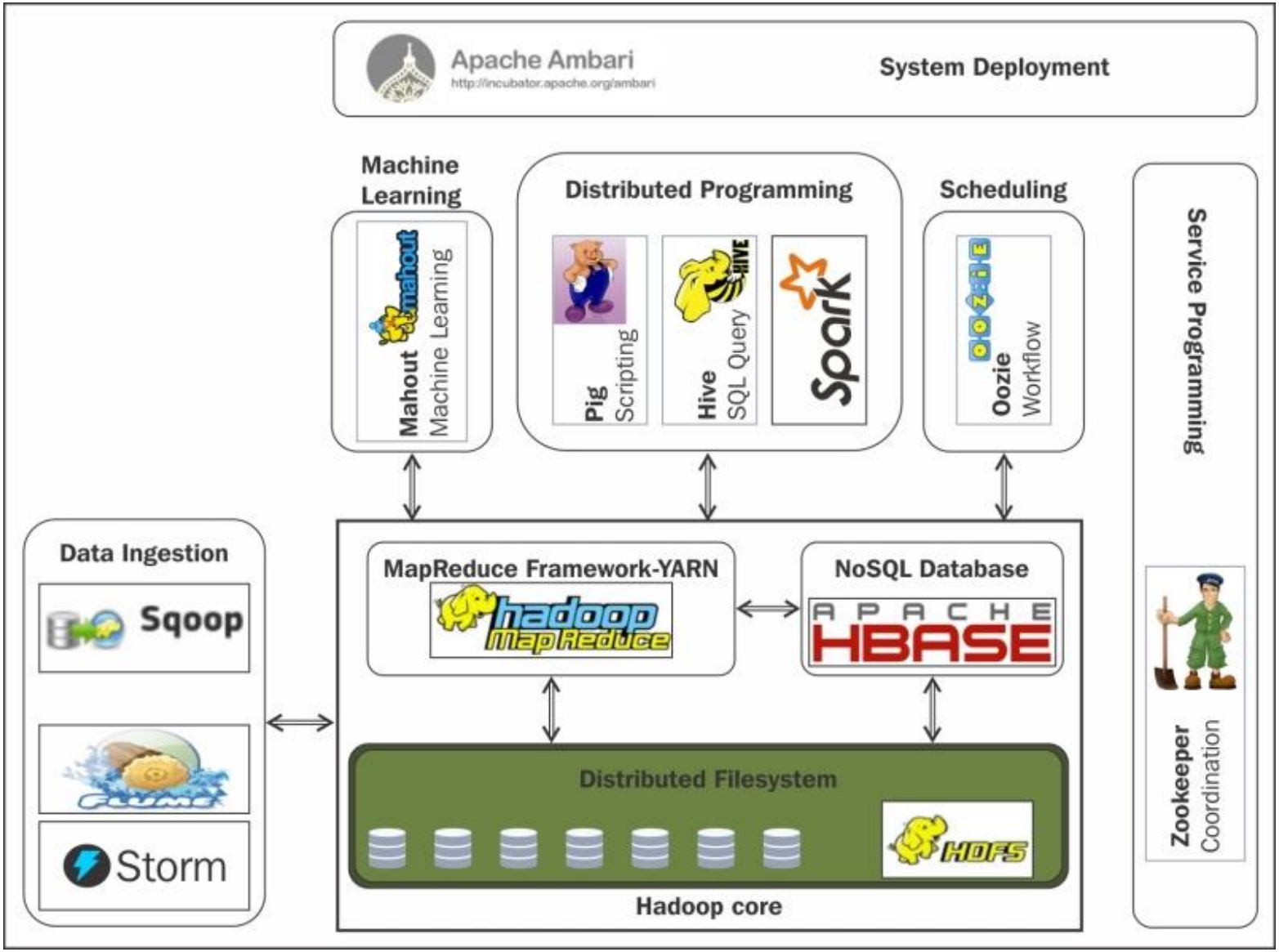
Hadoop File System

Typisk maskinvare-kontekst for Big Data-rammeverk



«In house» eller skybasert

Hadoop-økosystemet

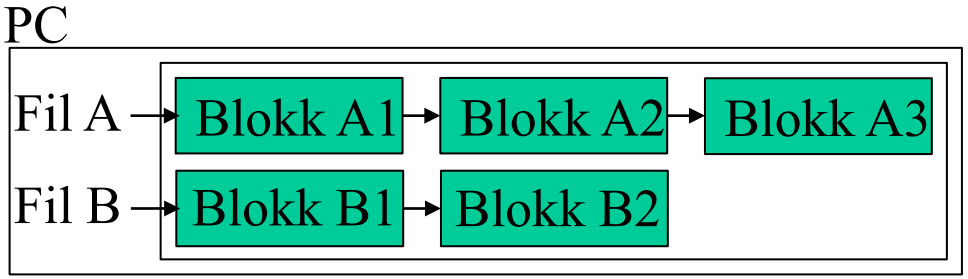


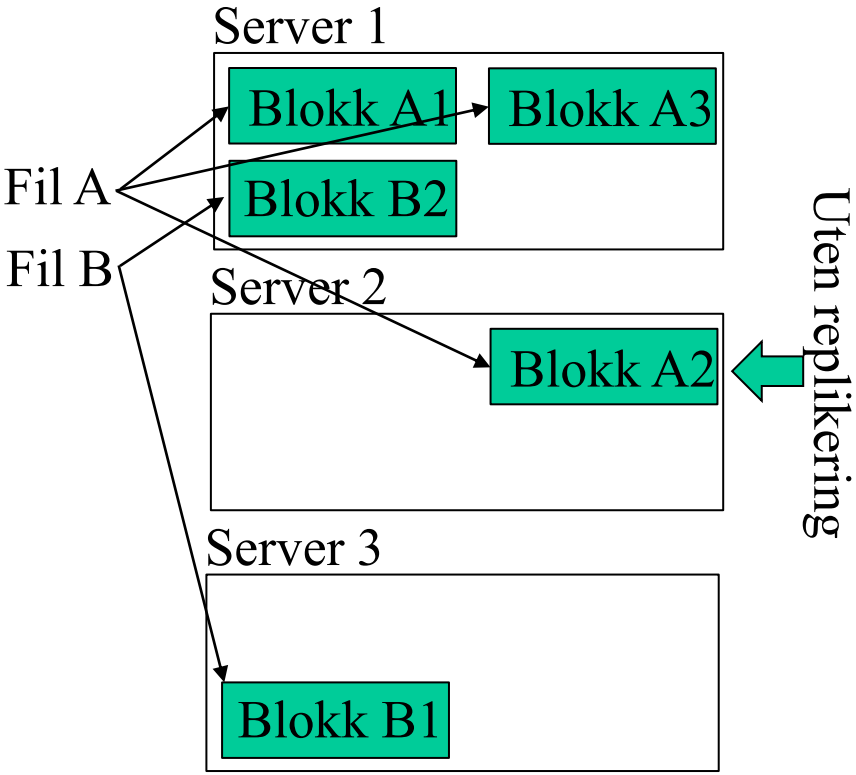
(Frå *Hadoop Essentials*)

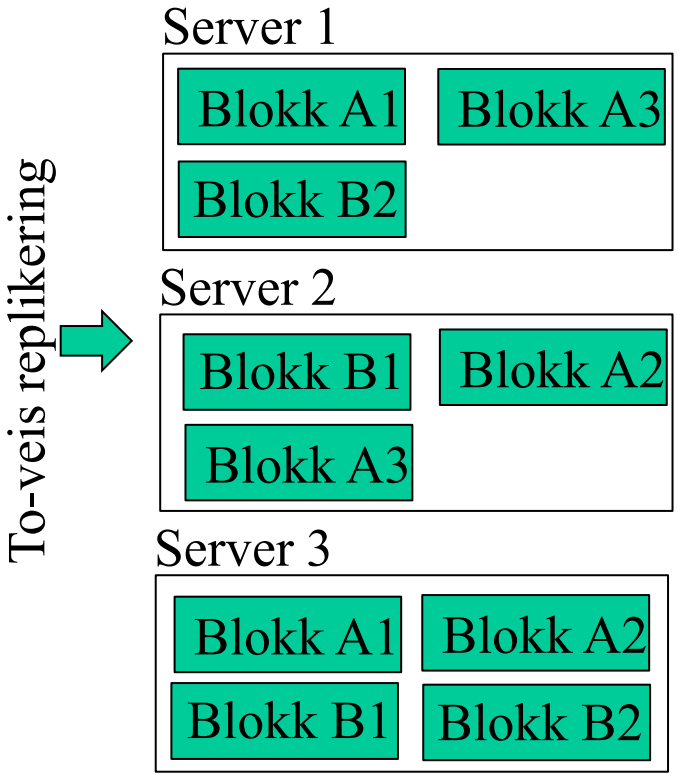
Hadoop File system (HDFS)

- Sterkt inspirert av Google File System (GFS)
- Standard filsystem i Hadoop
- ”User-level” filsystem
  - På topp av lokalt FS, alle data lagra i «vanlege» filer
- Mål:
  - (Veldig) store filer
  - Datastrøm-aksess
  - Hyllevare (maskiner)
- Ikkje eigna til:
  - Data-aksess med krav til liten forseinking (*low-latency data access)*
  - Mange små filer
  - Fleire samtidige skrivarar (dvs. er *single-writer*)
  - Vilkårlege oppdateringar i filer (dvs. er *append-only*)

Lokalt vs. distribuert filesystem (*veldig* forenkla)







Blokker

- Filer og metadata i HDFS lagra på ”lokalt” filsystem
- Filer delt opp i *blokker* med default størrelse typisk 64MB eller 128MB
  - *Jfr: disk-sektor på 512B og (typisk) 4 eller 8 KB blokkstørrelse i lokalt filsystem*
- Store blokker reduserer aksess-tid (seek)
- Kvar blokk lagra som ei ”lokal” fil
- Feiltoleranse vha. replikering av blokker på fleire nodar
- NB! Filer som er mindre enn 64/128MB bruker kun så mykje disk-plass som reell størrelse på fila
- Blokker (fast størrelse):
  - Mogleg med filer som er større enn det som kan lagrast lokalt på ei maskin
  - Gjer replikering enklare (samanlikna med fil som granularitet)

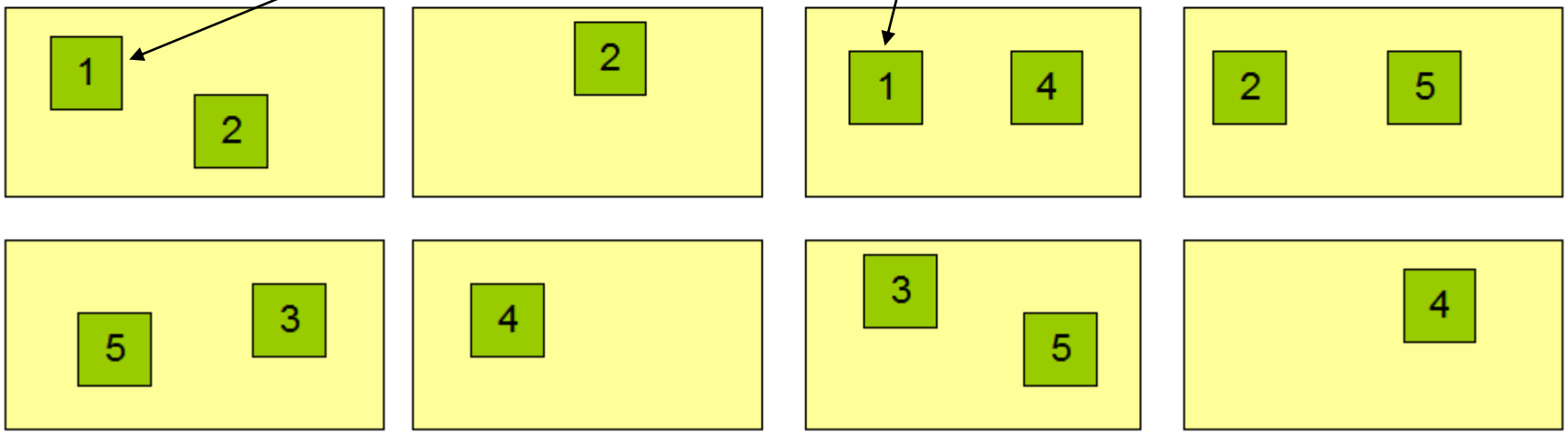
NTNU

replikeringsnivå

Block Replication

Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …

Datanodes



Node-typar

- *Master-worker*: *NameNode* og *DataNode*
- *NameNode:*
  - Handsamar namnerom til filsystem (filer, katalogar, og metadata for desse)
  - All informasjon kontinuerleg i hovudlager pga. effektivitet, disk kun for persistens
  - Informasjon lagra persistent på lokal disk som to filer: *namespace image* og *edit log*
  - Alle fil-metadata-operasjonar skrivne til log før operasjon stadfesta (write ahead logging)
  - Også oversikt for kvar fil kvar blokker er lagra
    - Ikkje nødvendig å lagre dette persistent (kan rekonstruerast frå datanodar ved restart)
  - *Secondary NameNode* (CheckpointNode): lagar nytt namespace image ved å kombinere noverande namespace image med operasjonane i edit log, kan så hentast av NameNode

Viktige datastrukturar og filer

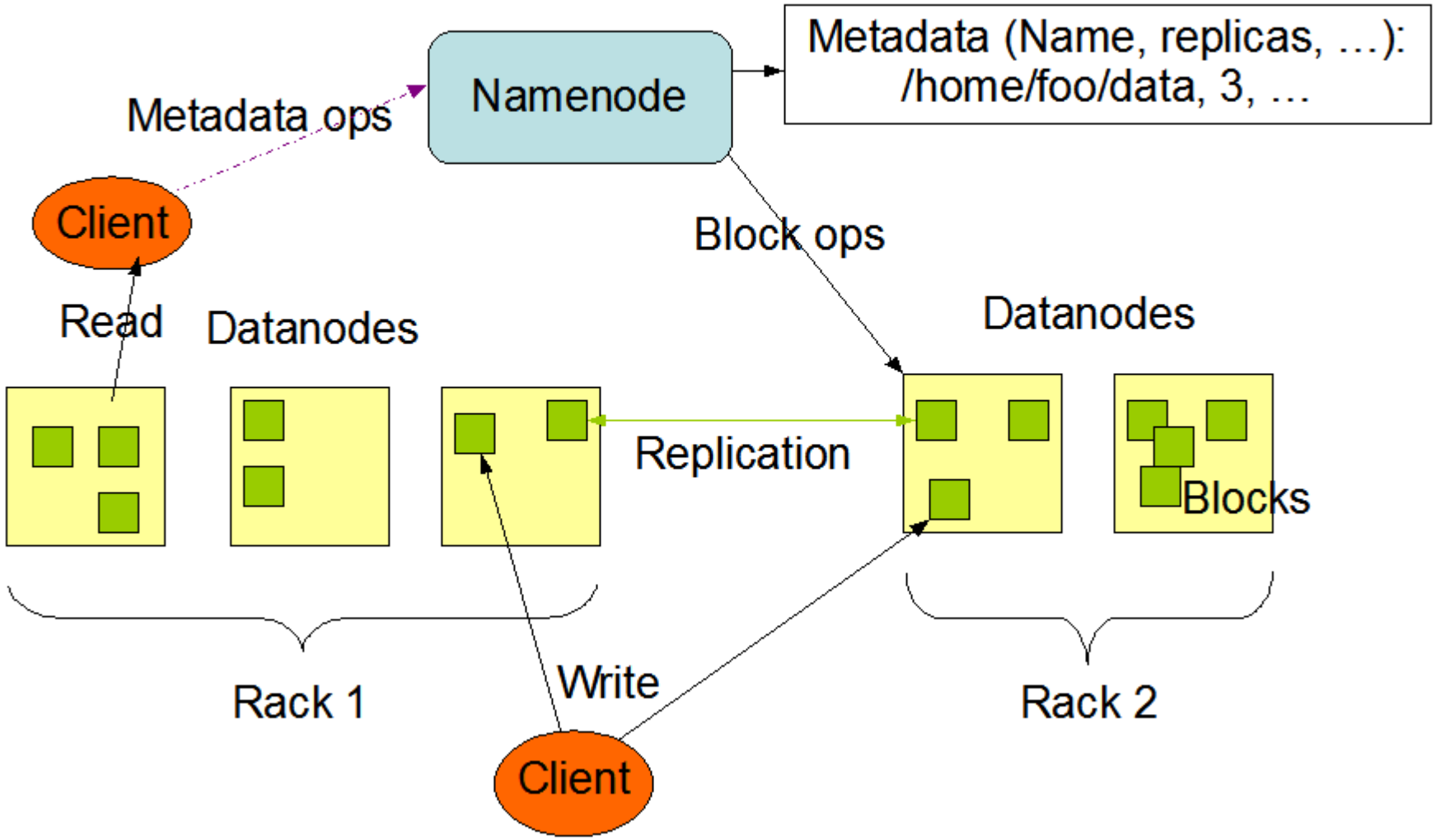
- NameNode*:*
  - Persistent i fil: *Namespace image*:
    For kvar fil/katalog (litt forenkla, i praksis organisert som inoder etc.): Path, Replication, ModificationTime, AccessTime, PreferredBlockSize, BlocksCount, FileSize, Permission, UserName, GroupName, [block identifiers]
  - I minne: Også kva datanodar blokkene ligg lagra på
  - I loggfiler: fil-metadata-operasjonar
- DataNode:
  - Ei fil for kvar datablokk, blokk-identifikator som del av filnamn
  - Separat metadata-fil for kvar datablokk, som m.a. inneheld sjekksummar

Klientar

- *Klient* aksesserer filsystemet på vegne av brukar/program gjennom kommunikasjon med NameNode og datanodar
- Java API
- Kommandolinjenivå (eksempel):

```
hadoop fs -ls
hadoop fs -mkdir testdir
hadoop fs -rmdir testdir
hadoop fs  -copyFromLocal test.txt
hadoop fs  -copyToLocal test.txt
sudo -u hdfs hdfs dfsadmin –report
```

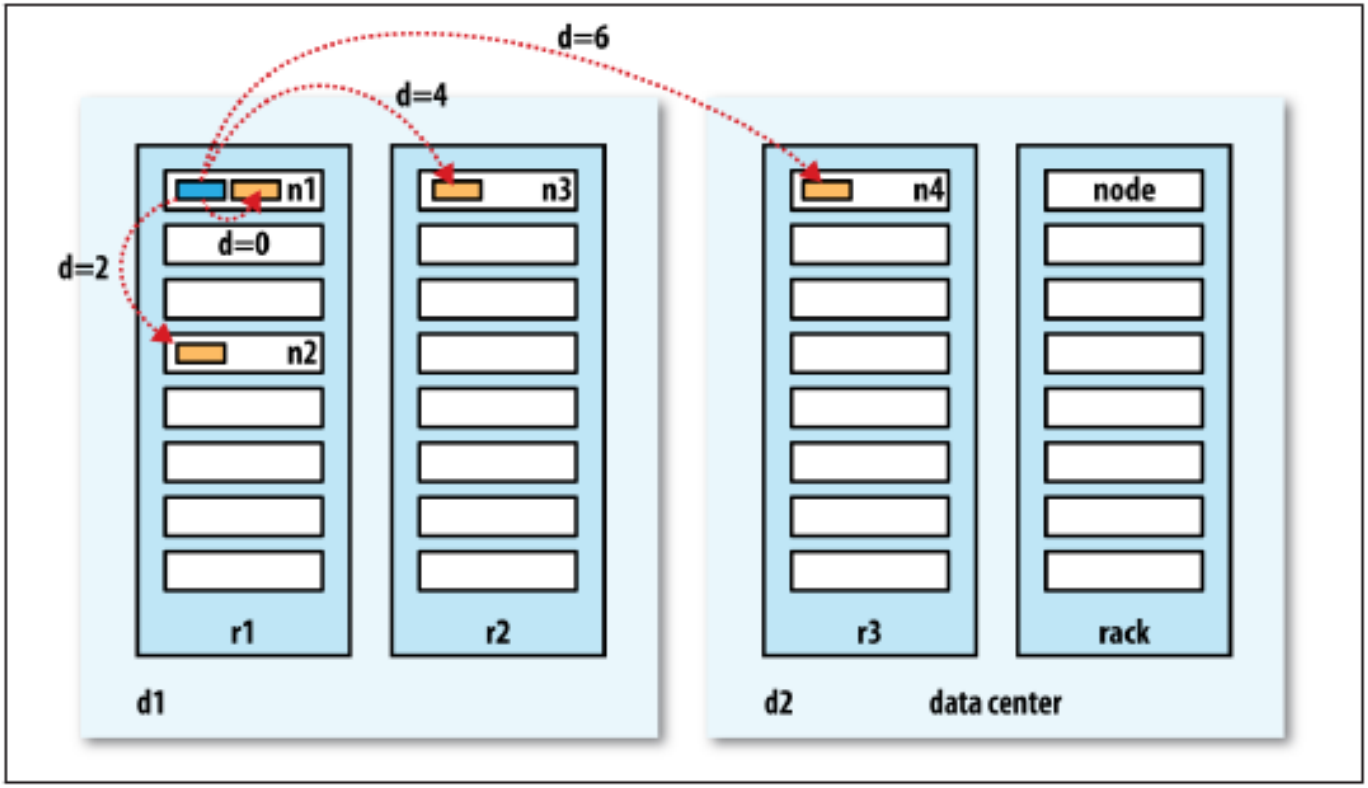
HDFS Architecture



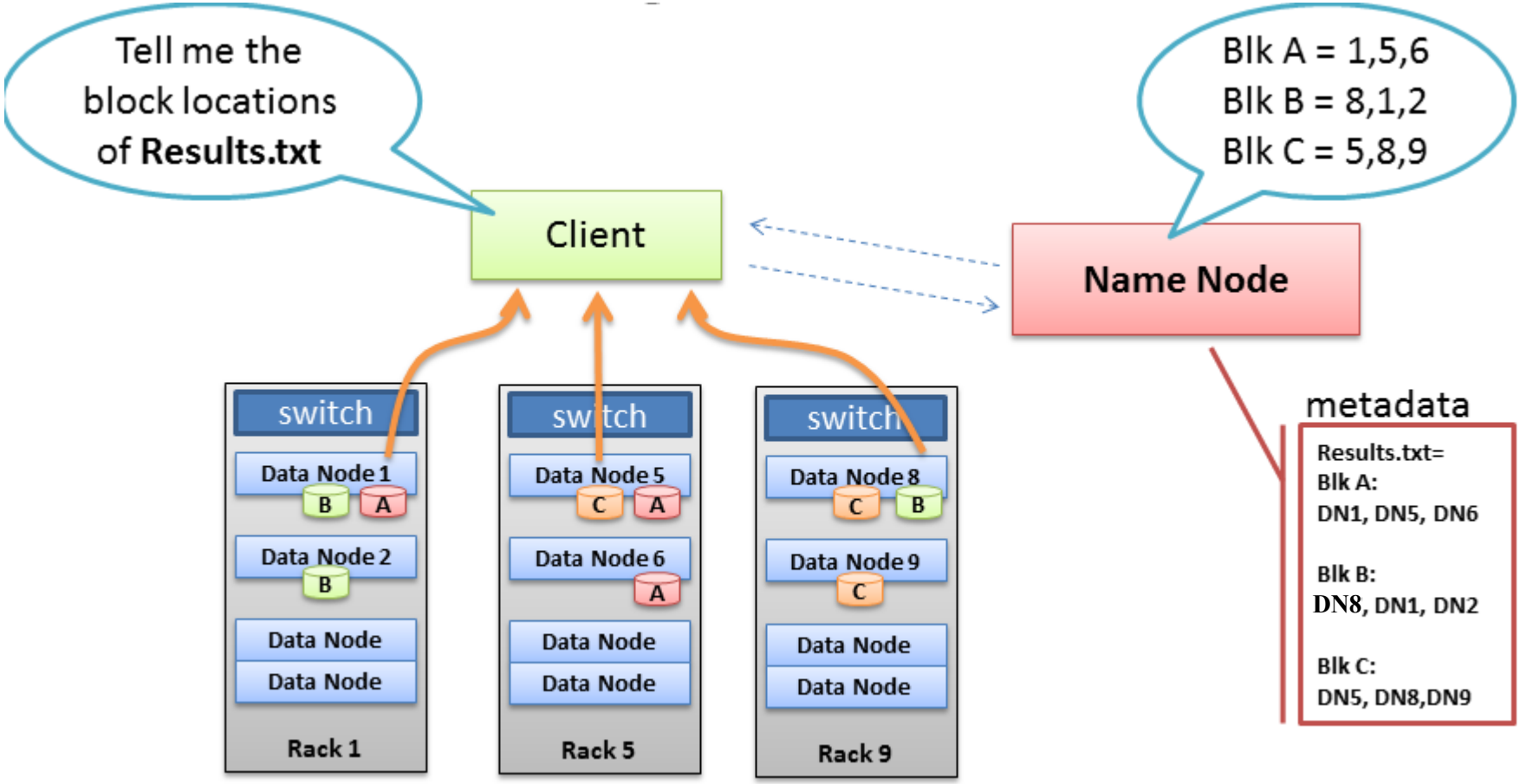
Feiltoleranse

- NameNode feiltoleranse vha:
  - Lagring av persistent tilstand på meir enn ein disk
  - Lagring av persistent tilstand på ikkje-lokal disk (andre nodar), evt. NFS-filsystem
  - Bruk av namespace image på Secondary NameNode ved gjenoppretting (recovery)
  - Hadoop 2: aktiv standby-NameNode for redusert nede-tid
- DataNode feiltoleranse:
  - Ikkje nødvendig pga. replikering av blokker
  - Bruk av sjekksum på kvar blokk for å detektere diskfeil

Nettverks-distanse i Hadoop



Lesing av fil



1) Finn blokk-lokasjon(ar)

2) Les fil (blokkar) sekvensielt, velg blokk som er "nærmast" (jfr. nettverks-distanse)

Skriving til fil

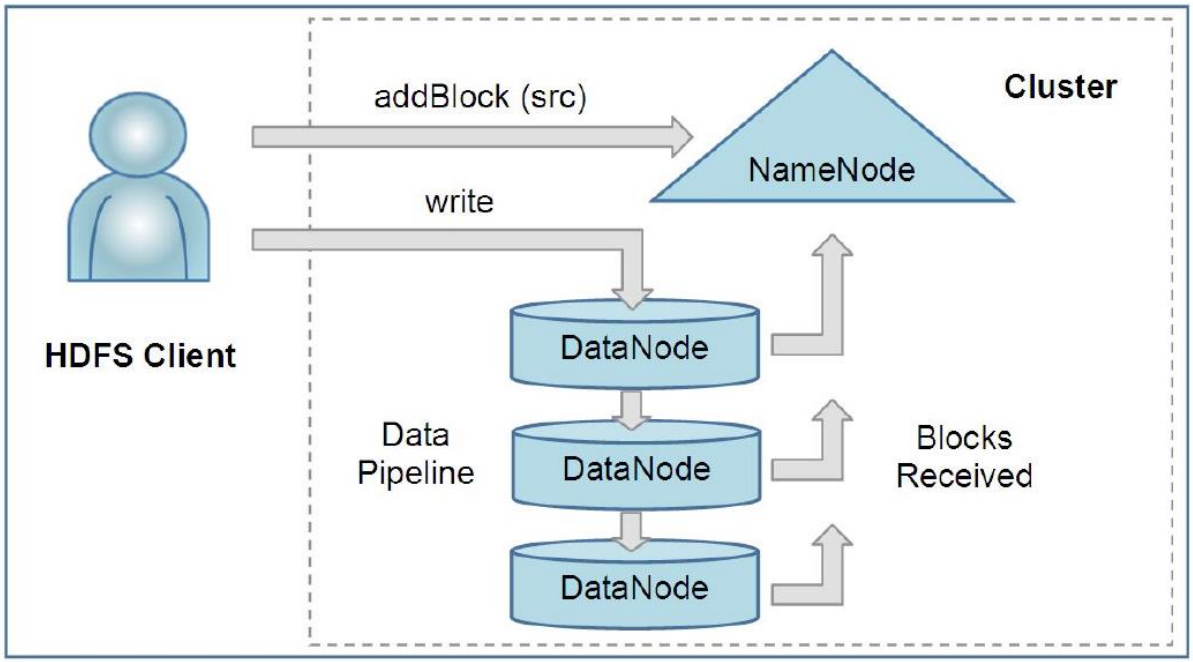
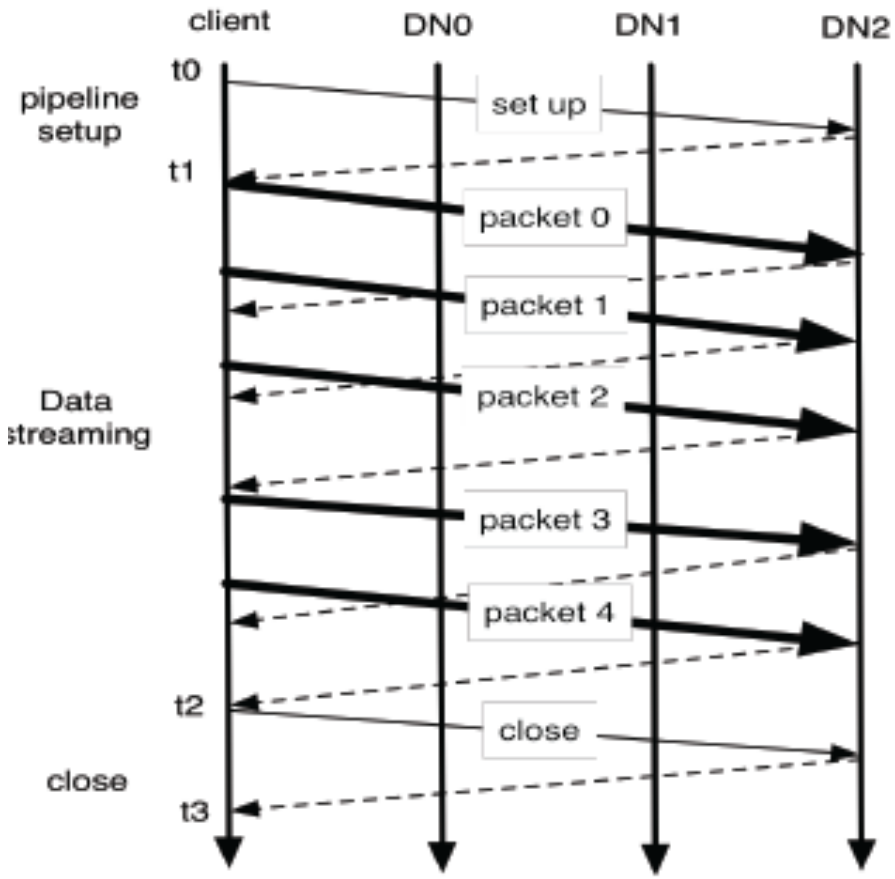


Figure 1. An HDFS client creates a new file by giving its path to the NameNode. For each block of the file, the NameNode returns a list of DataNodes to host its replicas. The client then pipelines data to the chosen DataNodes, which eventually confirm the creation of the block replicas to the NameNode.

Data-samleband ved skriving av blokk

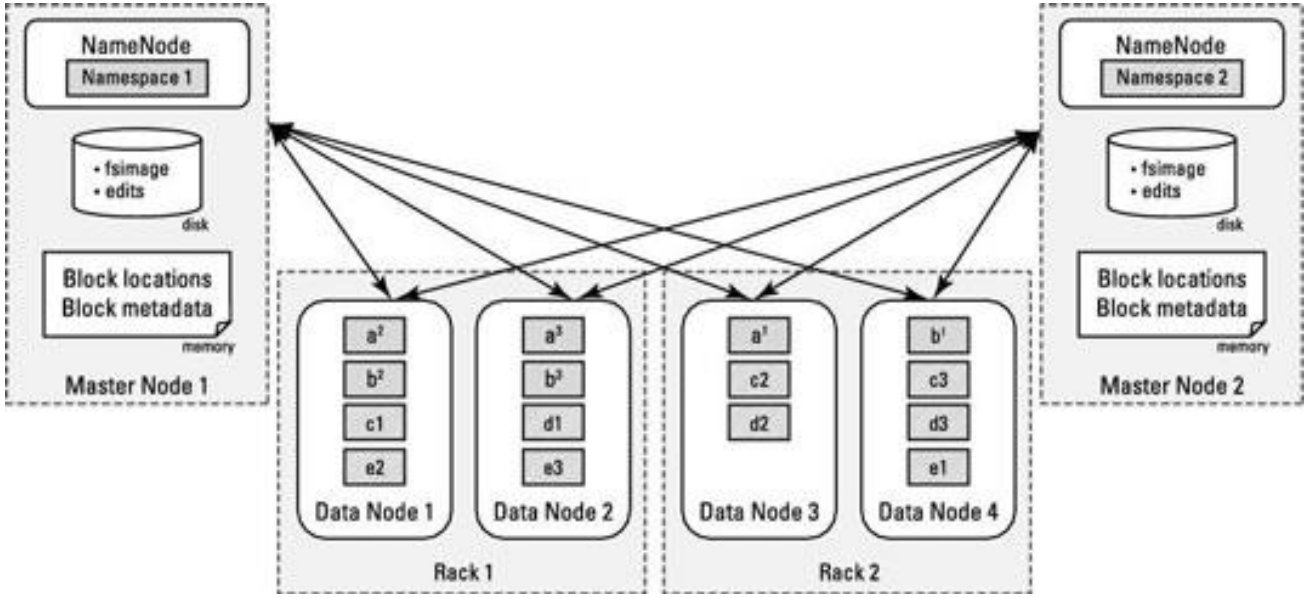


Vedlikehald

- Handsaming av replika
  - Over/under-replikering detektert vha. periodisk info frå datanodar
- Rebalansering
- *Block scanner:* verifisering av sjekksum på datablokker
  - Om feil: merka korrupt og nytt replikat vert etter kvart generert
- Dekommisjonering: kontrollert fjerning av datanodar
  - Først generer nye replikat av blokker som har vore lagra på noda, deretter kan noda fjernast

Nytt i HDFS/Hadoop 2.0 (1)

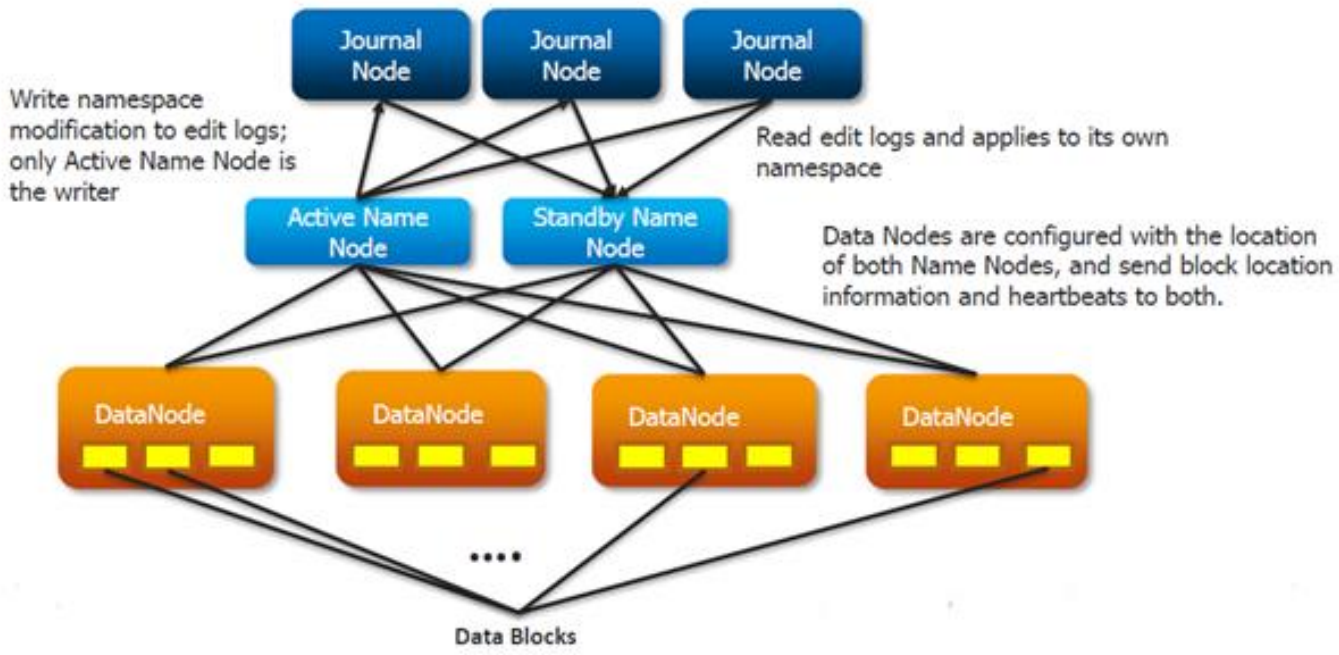
- Federering: distribuert/partisjonert namnerom (uavhengige NameNode'er)



- Namnerom: t.d. /usr og /home

Nytt i HDFS/Hadoop 2.0 (2)

- Betre støtte for høg-tilgjengelegheit:



- Betre støtte for sikkerheit (autentisering av klientar etc.)
- Kontinuerleg forbetring av yting
- Hadoop 3: Multiple namnenoder

Erfaringar

- Svært påliteleg
- Lite ressursar for drift, ca. ein operatør pr. 3K nodar
- Skalerbar: > 100 PB, > 4500 nodar i ei klynge

For dei som vil prøve MapReduce, sjå ”Quickstart to installing tools used in TDT4305“ som er lagt ut på Blackboard

MapReduce

- Programmerings-rammeverk for *parallell og skalerbar prosessering av (veldig) store datasett*
- Mål:
  - Prosessere store filer/datasett (mange PB)
  - Mange noder (mange tusen)
  - Lineær skalering (t.d. 10 x nodar ➔ 10 x yting)
  - Skjule kompleksitet i distribusjon, parallellisering og feiltoleranse for applikasjons-utviklarar
  - Skilje mellom kode for distribuert utføring og applikasjonskode ➔ endring i applikasjonar/algoritmar enkelt
  - Hyllevare (maskiner)

MapReduce

- *Jobb* i MapReduce skildra (og utført) som to separate faser:
  - *Map*-funksjon (også kalla ”mapper”): prosesserer key/value-par og genererer *liste med key/value-par*
    (k1, v1) → List(k2, v2)
  - *Reduce*-funksjon (også kalla ”reducer”): prosesserer alle verdiar med same nøkkel
    (k2, List(v2)) → List(k3, v3)
- Ut-data frå *Map*-fase:
  - Partisjonert basert på k2 slik at alle tuplar med same verdi for k2 ender opp på same Reduce-node (shuffle)
  - Sortert slik at ein effektivt kan utføre Reduce på alle tuplar med same k2

Eksempel: Telje førekomst av termar i dokumentsamling

d1= Deer Bear River
d2= Car Car River
d3= Deer Car Bear

Algoritme (konseptuelt):

Map

1. for kvart dokument *d i D*
2. for kvar term *t* i D
3. skriv ut (*t*, 1) til fil TEMP

Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1

4. sorter TEMP på attributt #1 (term)

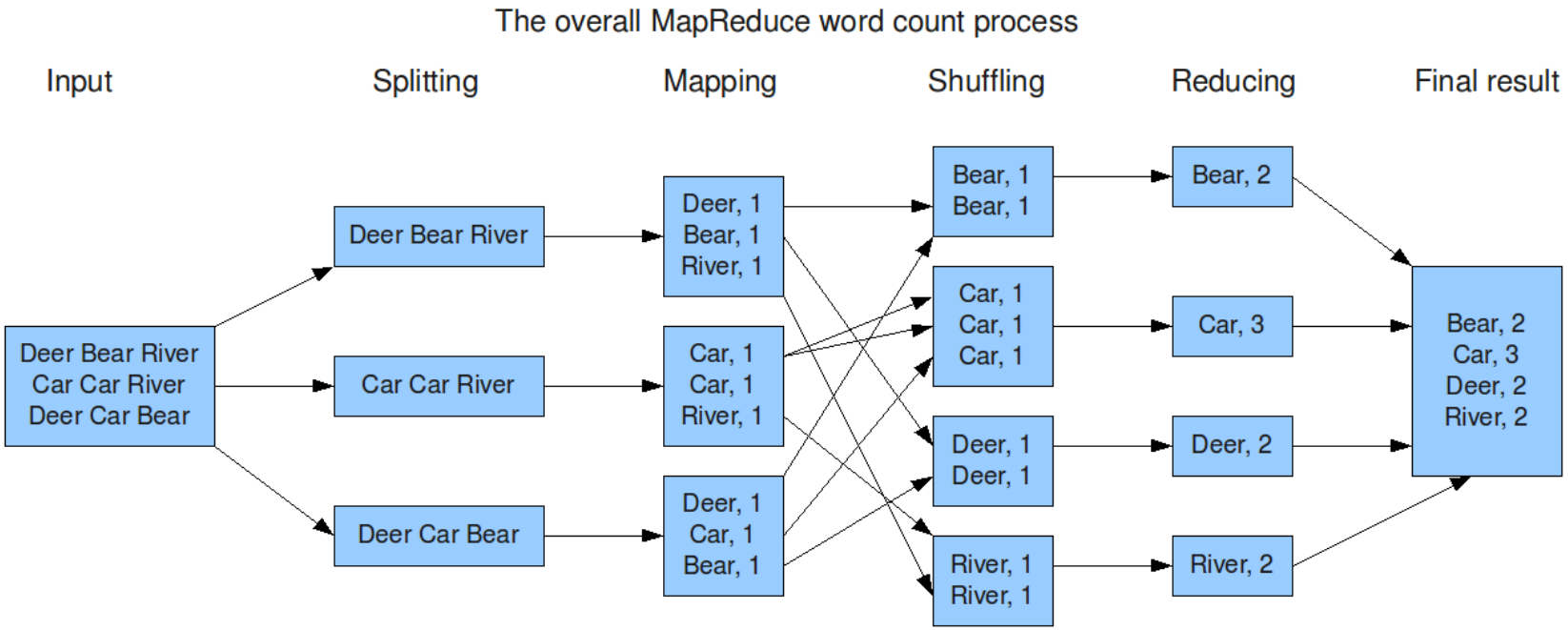
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer,1
River,1
River,1

Reduce

5. for kvar distinkt *t*
6. summér verdiar i attributt #2 (tal på førekomstar) i *s*
7. skriv (t, s)

Bear, 2
Car, 3
Deer, 2
River, 2

Eksempel: telje førekomst av termar i MapReduce



Implementasjon (høgnivå)

```
mapper (filename, file-contents):
        for each word in file-contents:
                output (word, 1)

reducer (word, values):
        sum = 0
        for each value in values:
                sum = sum + value
        output(word, sum)
```

Eksempel (frå Hadoop-boka)

- Utgangspunkt:
  - Datasett med ver-observasjonar
  - Ei fil for kvar dag for kvar stasjon (> 10K stasjonar)
  - Kvar linje i fil har informasjon om stasjon, data, tid, temperatur, vindstyrke, etc.:

```
0067011990999991950051507004...9999999N9+00001+99999999999...
0043011990999991950051512004...9999999N9+00221+99999999999...
0043011990999991950051518004...9999999N9-00111+99999999999...
0043012650999991949032412004...0500001N9+01111+99999999999...
0043012650999991949032418004...0500001N9+00781+99999999999...
```

År

°C*10

  - Key-value-par som input til Map-funksjon:

key er posisjon i fila

```
(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)
(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)
(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)
(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)
(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)
```

  - Ønskjer å finne max-temperatur for kvart år

Dataflyt

- Kun år og temperatur av interesse og ut frå Map som dette:

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

- Input til Reduce som dette:

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

- Resultat frå Reduce (max per år):

```
(1949, 111)
(1950, 22)
```

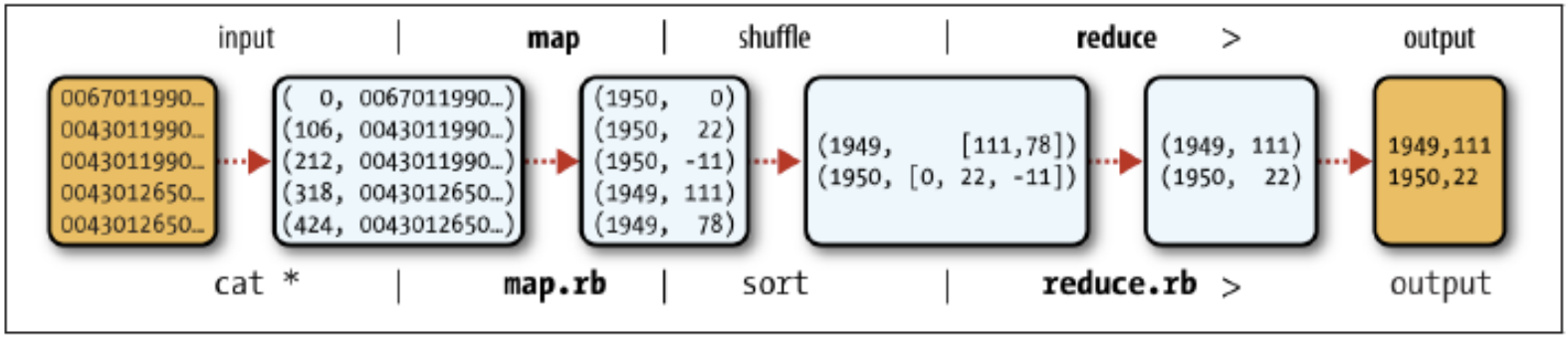


Figure 2-1. MapReduce logical data flow

Mapper i Java

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

"Serialiserte" datatypar:

Reducer i Java

```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

Applikasjon i Java

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

NB! Til kompilering: hadoop-mapreduce-client-core-3.0.0.jar og hadoop-common-3.0.0.jar eller tilsvarande frå $HADOOP_HOME/share/hadoop

MapReduce dataflyt

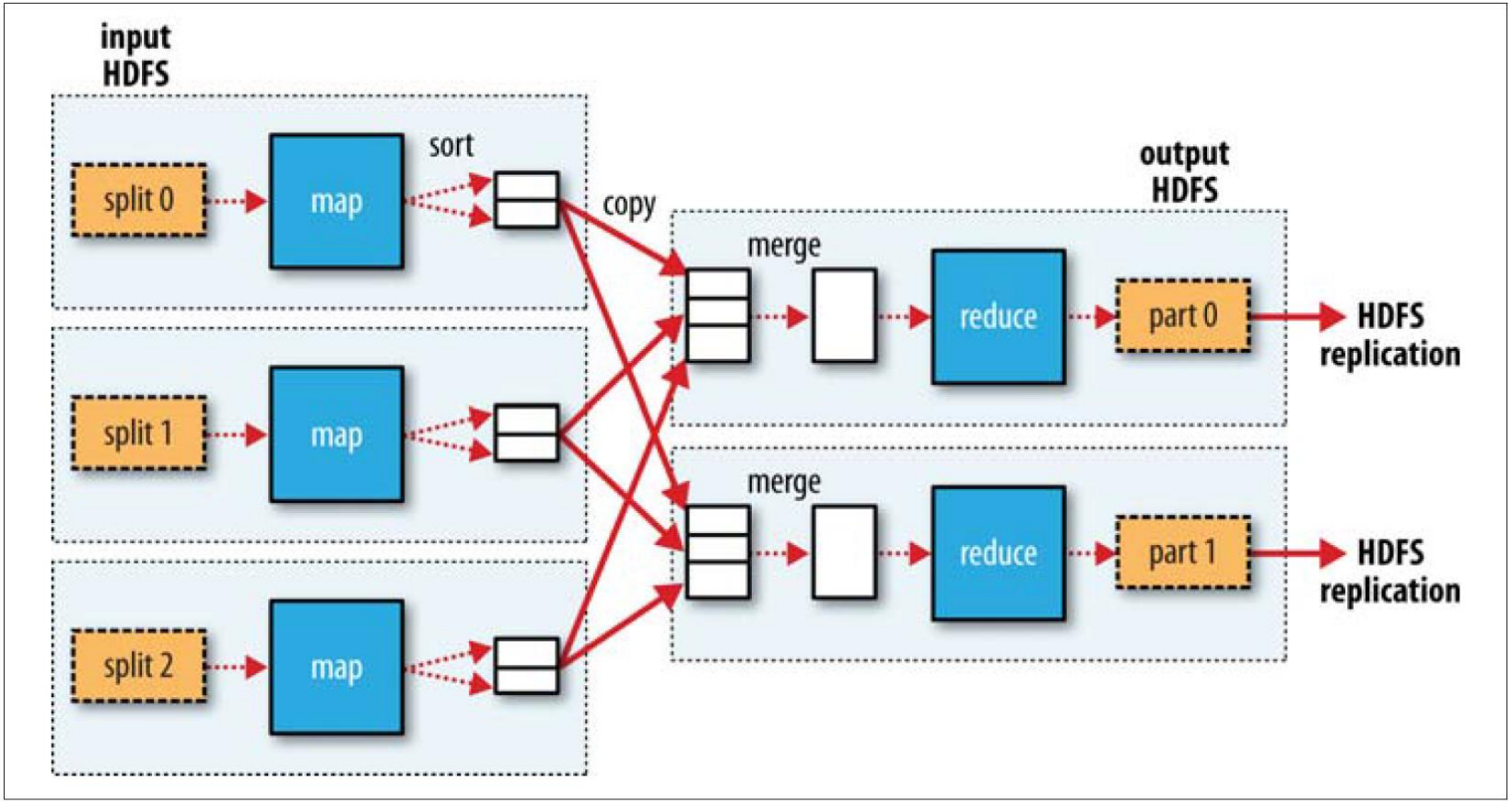
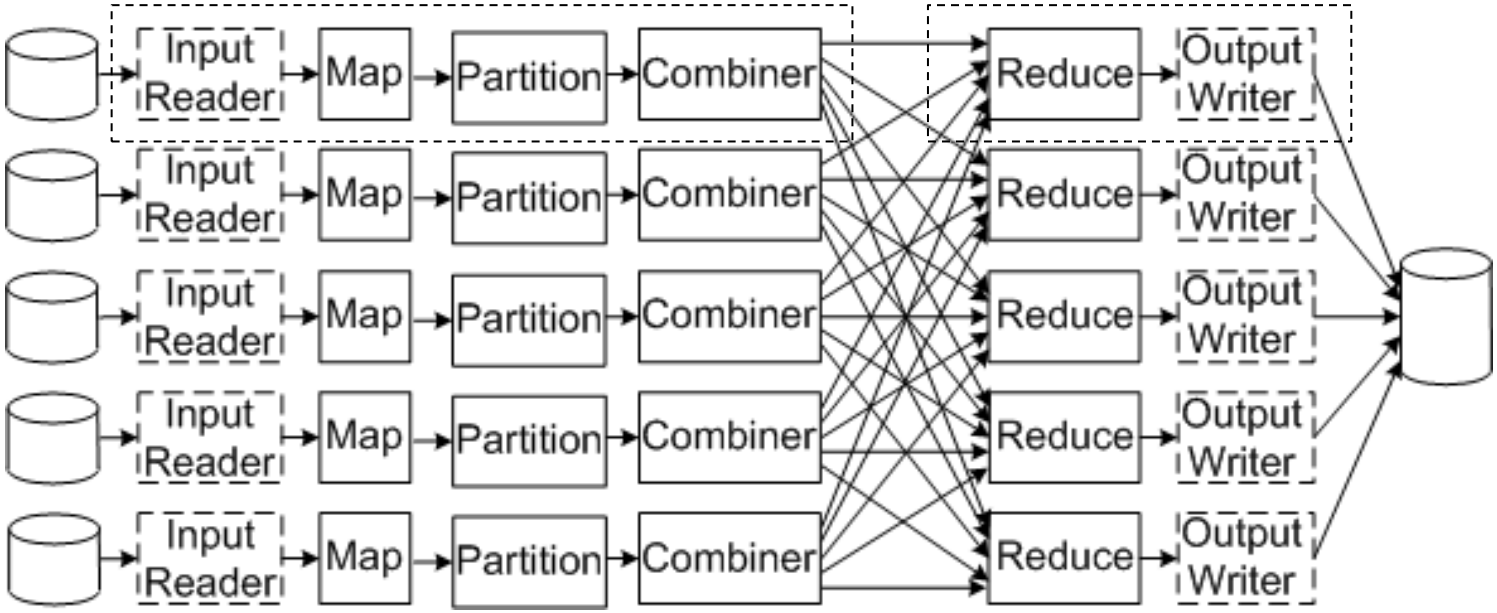
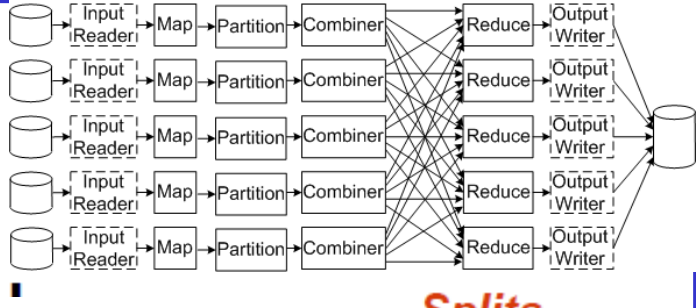


Figure 2-4. MapReduce data flow with multiple reduce tasks

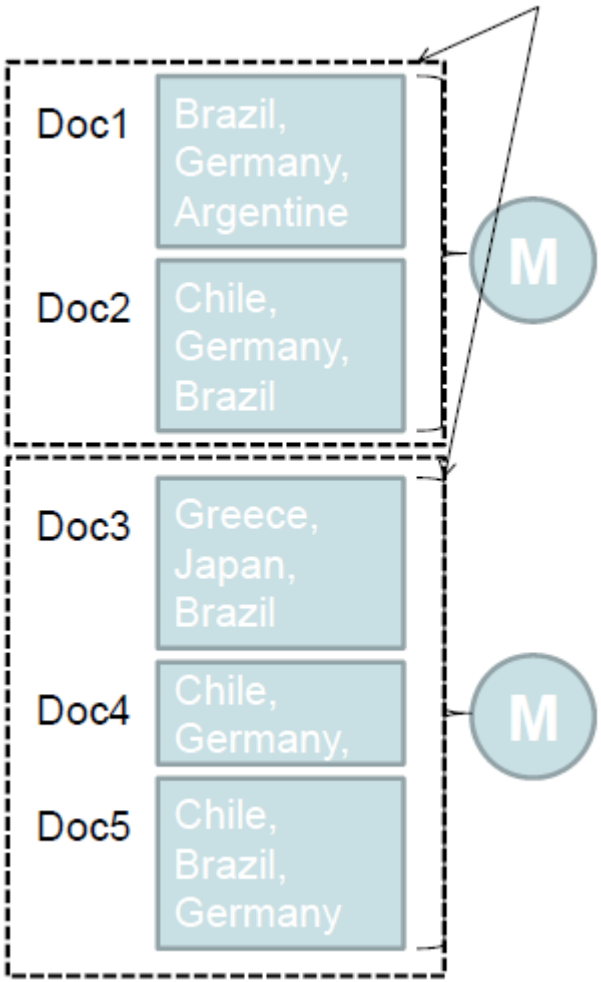
MapReduce/Hadoop: 6 steg

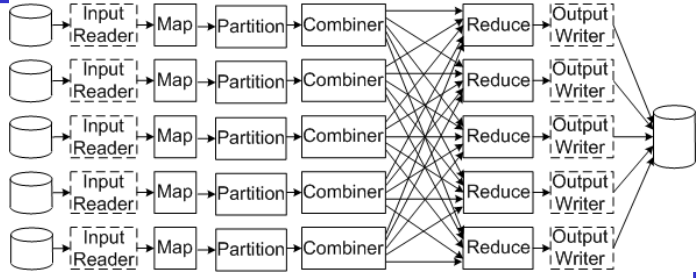


1. Input Reader



- Les data frå filer og transformerer til *key-value*-par
  - Kan også bruke data frå andre kjelder, t.d. databasar
- *Splits* vert generert frå data, *split* er granulariteten til data prosessert av ein *map task*
  - Typisk størrelse på *split* er same som blokk



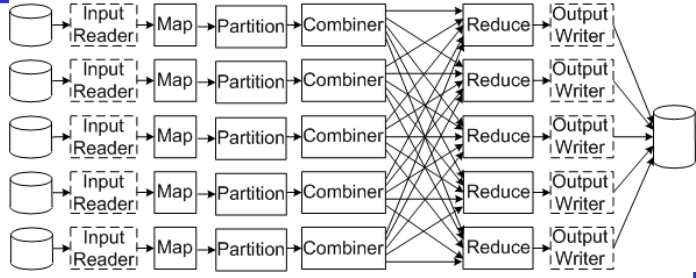


2. Map-funksjon

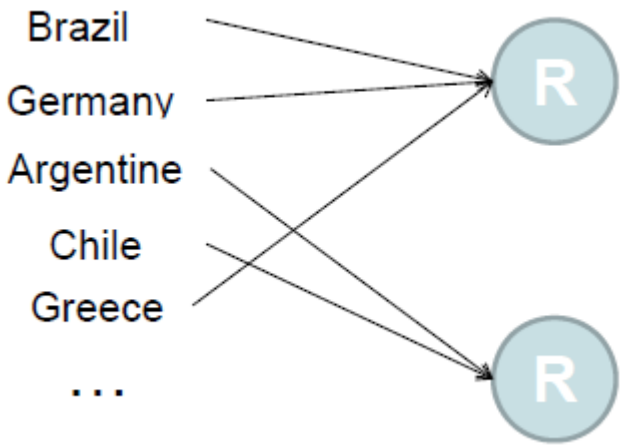
- Input er *key-value*-par frå *Input Reader*
- Køyrer koden i Map-funksjonen på paret
- Produserer nytt *key-value*-par som resultat

- Resultat frå Map-funksjon lagra i buffer i hovudlager, vert skrive til disk (*spill files*) når dette er nesten fullt
- Filene fletta til ei sortert fil

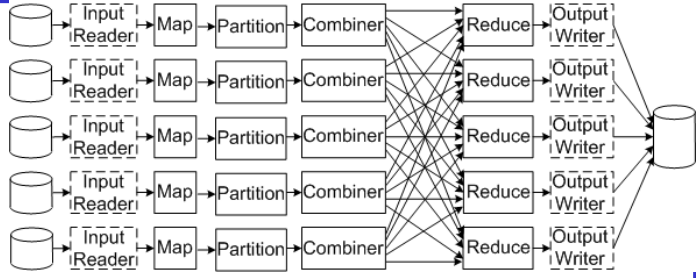
3. Partisjonerings-funksjon



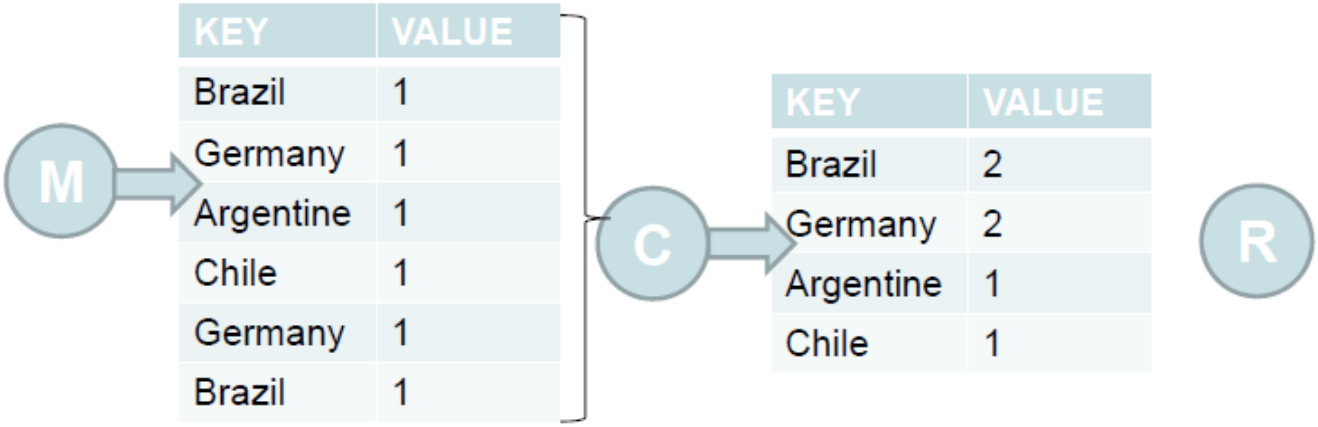
- Default: *hash-funksjon* brukt for å partisjonere resultat av Map-tasks til Reduce-tasks
  - Fungerer vanlegvis bra for *last-balansering*
- Av og til nyttig med annan partisjonerings-funksjon
  - *Brukar-definert*



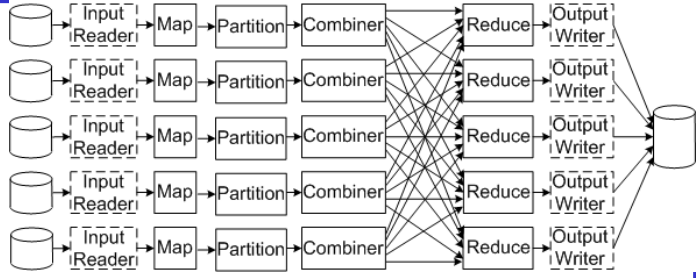
4. Combiner-funksjon



- Bruk av denne er valgfri
- Kan redusere kommunikasjonskostnad
- Bruk når
  - Map-funksjon produserer mange par med same nøkkel
  - Reduce-funksjon er kommutativ (a+b=b+a) og assosiativ (a+b)+c=a+(b+c)
- I eksempelet vil Combiner-funksjon utføre delvis samanslåing slik at *par med same nøkkel vil verte prosessert som gruppe av Reduce-task*



- Ofte vil ein bruke same funksjon til Combiner og Reducer



5. Reduce-funksjon

- Vert kalla *ein gong* for kvar *distinkt nøkkel* og anvendt *på alle verdiar som høyrer til den nøkkelen*
  - Alle par med same nøkkel prosessert som gruppe
- Input til kvar Reduce-task er sortert basert på nøkkel
- Mogleg med eigen-definert *comparator*

Shuffle/sort i meir detalj

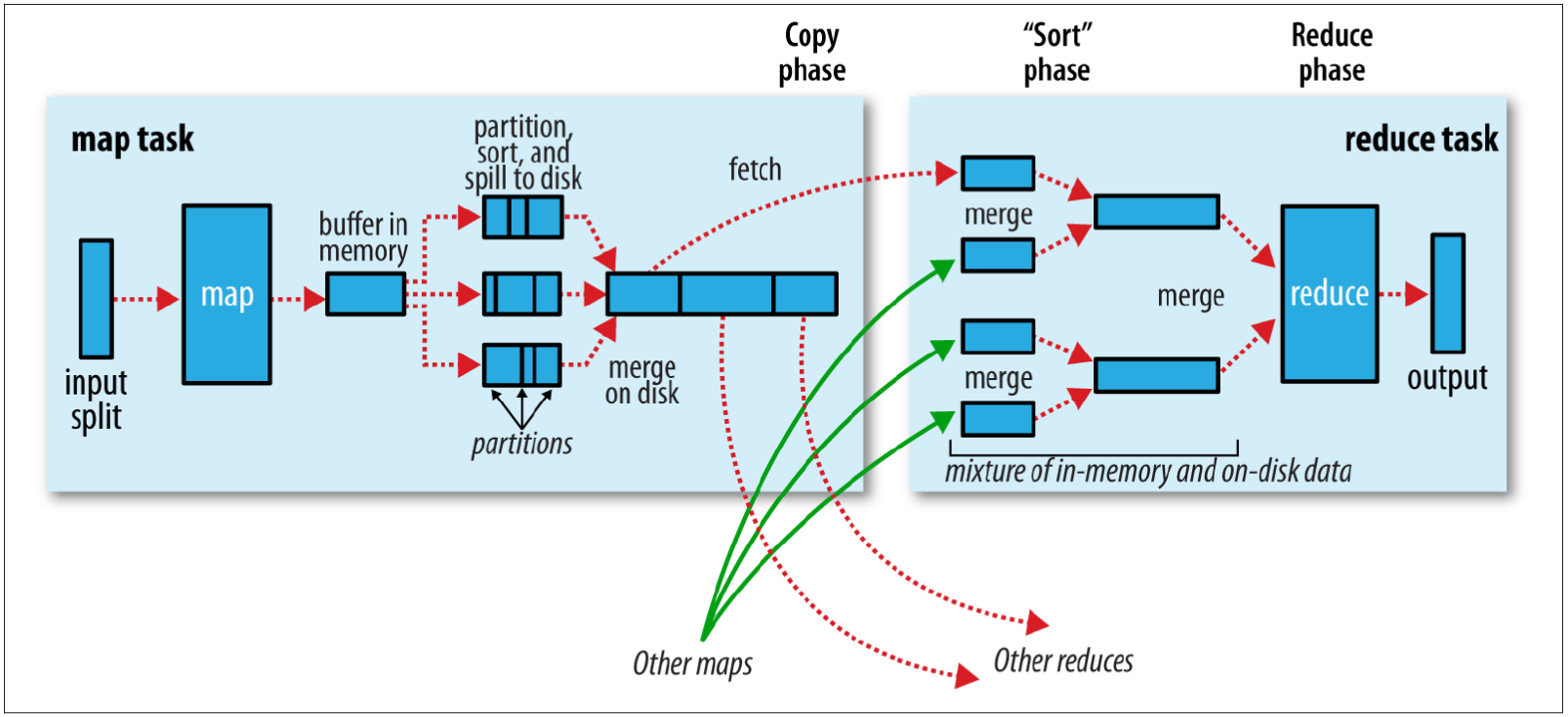


Figure 7-4. Shuffle and sort in MapReduce

- Ein *map task* for kvar *split*
- Prøver å køyre map task på node som lagrar split (HDFS-blokk)

6. Output Writer

- Ansvarleg for å skrive resultat til sekundærlager (disk)
  - Typisk ein katalog, med ei fil for kvar Reducer-task
  - Kan også modifiserast til å lagre i t.d. database

Oppgåve

- Anta at ein har ei fil PersonInfo.txt som inneheld informasjon om namn, alder og løn, dvs. format som dette:

```
Kari 45 450000
Ola 30 200000
Kate 30 500000
Pål 45 550000
```

- Vi ønskjer å finne gjennomsnittsinntekt for kvar alder, dvs. resultat som dette (treng ikkje å vere sortert):

```
45 500000
30 350000
```

- Vis med pseudokode for *mapper og reducer* korleis dette kan gjerast i MapReduce. Anta for enkelheits skyld at value til map er ein post med felta age og salary, dvs. bruk følgjande som utgangspunkt:

```
public void map(key(name), value(age,salary))
public void reduce(key, Iterable values)
```

Mogleg svar

Gjennomsnittsinntekt for kvar alder i MapReduce:

```
public void map(key(name), value(age, salary))
     write(age, salary)

public void reduce(key, Iterable values)
     n = 0
     total = 0
     foreach val in values
          n++
          total = total+ val
     avg = total/n
     write(key, avg)
```

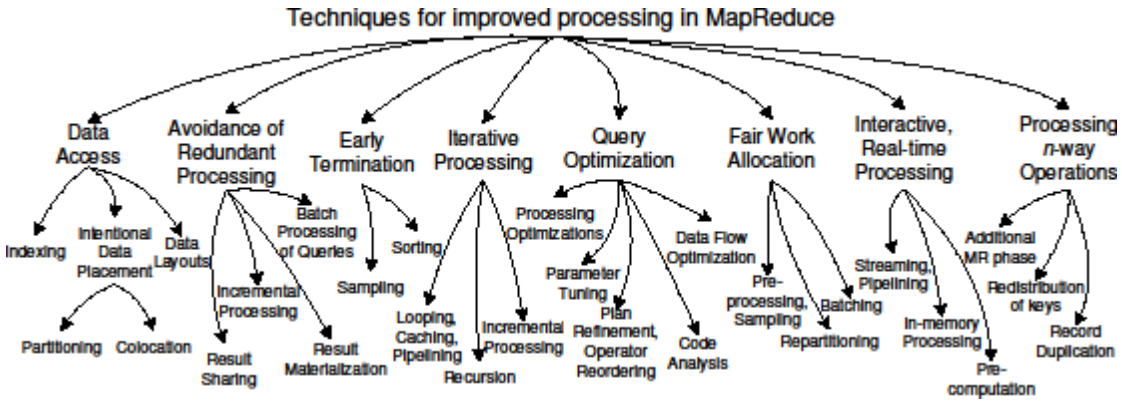
Deklarative språk på toppen av MapReduce: Pig & Hive

```
-- max_temp.pig: Finds the maximum temperature by year
records = LOAD 'input/ncdc/micro-tab/sample.txt'
  AS (year:chararray, temperature:int, quality:int);
filtered_records = FILTER records BY temperature != 9999 AND
  quality IN (0, 1, 4, 5, 9);
grouped_records = GROUP filtered_records BY year;
max_temp = FOREACH grouped_records GENERATE group,
  MAX(filtered_records.temperature);
DUMP max_temp;
```

```
CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999 AND quality IN (0, 1, 4, 5, 9)
    > GROUP BY year;
1949    111
1950    22
```

Generelle svakheiter/begrensningar med MapReduce

- Høg kommunikasjonskostnad
- Støttar dårleg t.d.:
  - Iterasjon
  - ”Early termination” (eks.: top-*k*)
  - Sanntidsprosessering
  - Multi-vegs operatorar (inkl. join)
  - …
- Har gjeve rom til mange forslag til utvidingar, og til nye system som t.d. Spark

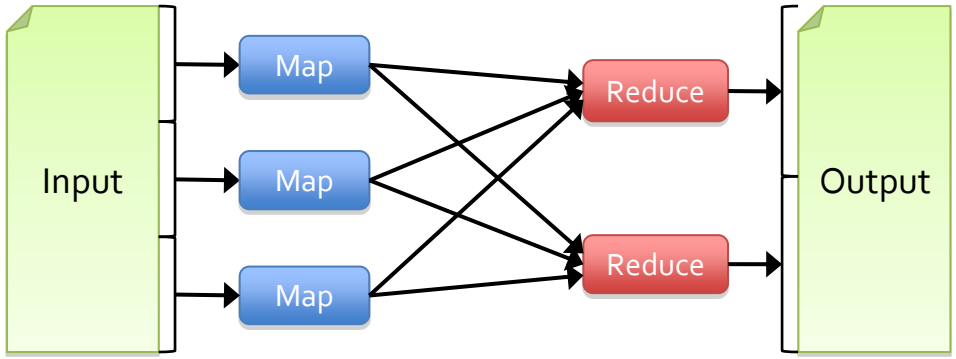


NTNU

Spark

Bakgrunn og motivasjon

- Observasjon: MapReduce gjorde analyse av Big Data svært mykje enklare ved å skjule aspekt rundt skalering og feilhandsaming for brukarar/applikasjonsutviklarar
- Ønske om tilsvarande abstraksjonar for breiare klasser av applikasjonar
- Motivasjon: Eksisterande programmeringsmodellar som MapReduce basert på asyklisk datastraum *frå* stabilt lager *til* stabilt lager

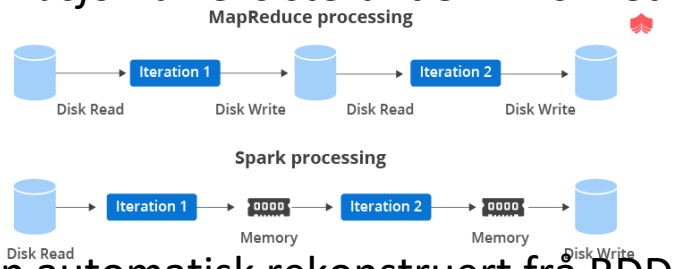


Mål

- Støtte applikasjonar der datasett vert gjenbrukt av (parallelle) operasjonar
  - Iterative jobbar (vanleg i maskinlæring)
  - Interaktiv dataanalyse og datagruvedrift
- Behalde (implisitt) feiltoleranse og skalerbarheit som i MapReduce
- Eksperimentere med programmerbarheit
  - Integrere i programmeringsspråket Scala
  - Støtte interaktiv bruk i Scala-interpreter

Programmingmodell: RDD

- Resilient distributed datasets (RDDs)
  - Uforanderlige (immutable), partitisjonerte samlingar med objekt, konstruert frå:
    -Eksternlager (t.d. HDFS-filer eller database)
    -"Parallelliserte" arrays
    -Resultat av *transformasjon* av eksisterande RDDs med map, filter, groupBy,…
  - Kan verte buffra for effektiv gjenbruk



  - *Resilient*: Tapt partisjon automatisk rekonstruert frå RDDs den er basert på

- Handlingar (actions) på RDDs
  - Count, reduce, collect, save, …
    ➔ resultat til "drivar"-program (applikasjon) eller t.d. fil

- Lat ("lazy") evaluering: Utføring startar først ved "action"

- Programmering:
  - spark-shell (Scala interpreter)
  - pyspark (Python interpreter)
  - spark-submit (jar-fil, dvs. Scala eller Java-applikasjon, eller Python-applikasjon)
  - Spark-sql
  - sparkR

Scala
(Akkurat nok til å forstå eksempla ☺)

- Fokusert på funksjonell programmering (sjølv om ein i praksis kan skrive ”Java i Scala”)
- Funksjon som verdi:

```
scala> val inc = (x : Int) => x + 1
inc: Int => Int = <function1>
scala> inc(1)
res0: Int = 2
```

- ```
  scala> List(1, 2, 3).map((x: Int) => x + 1)
  res1: List[Int] = List(2, 3, 4)
  ```
- ```
  scala> List(1, 2, 3).map(_ + 1)
  res2: List[Int] = List(2, 3, 4)
  ```
- ```
  scala> List(1, 2, 3).map((x: Int) => inc(x))
  res3: List[Int] = List(2, 3, 4)
  ```
- ```
  scala> List(1, 2, 3)(2)
  res4: Int = 3
  ```
- På ”minieksempelnivå” ganske likt Python

Eksempel: Logg-gruvedrift

Laste feilmeldingar frå logg inn i hovudlager, interaktivt søk for forskjellige mønster

```
// Base RDD
lines = sc.textFile("filename")
// Transformert RDD
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
// Actions
errors.count()
errors.filter(_.contains("MySQL")).count()
. . .
```

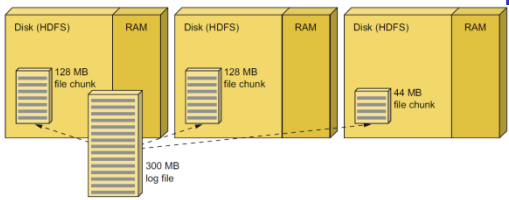


Figure 1.3 Storing a 300 MB log file in a three-node Hadoop cluster

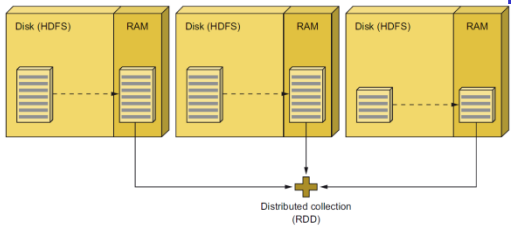


Figure 1.4 Loading a text file from HDFS

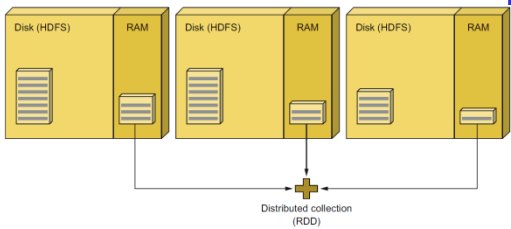
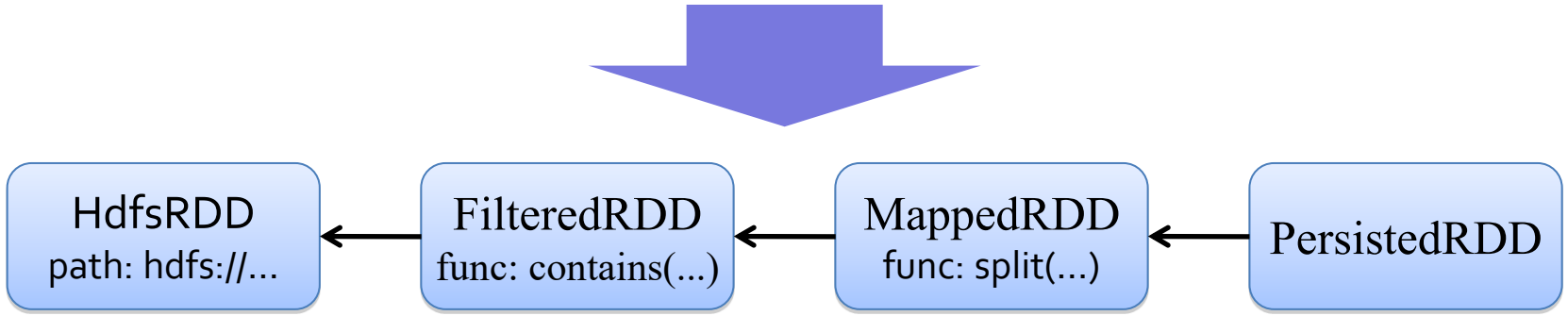


Figure 1.5 Filtering the collection to contain only lines containing the `OutOfMemoryError` string

RDD-feiltoleranse

- RDDs har avstammings-informasjon (*lineage)* som kan brukast til å rekonstruere tapte partisjonar
- Eksempel:

```
cachedMsgs = sc.textFile(...).filter(_.contains("error"))
                              .map(_.split('\t')(2))
                              .persist()
```



Viktige transformasjonar: map og filter

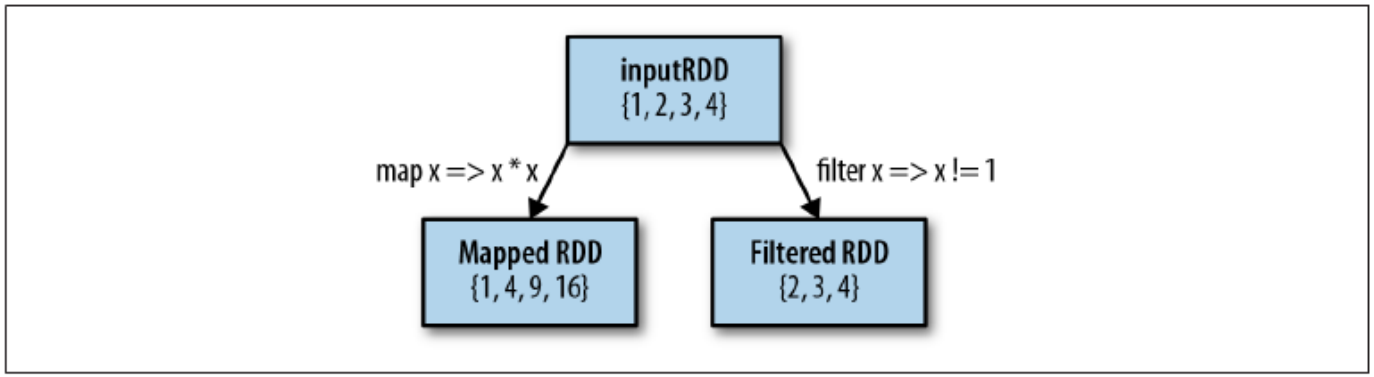


Figure 3-2. Mapped and filtered RDD from an input RDD

Example 3-26. Python squaring the values in an RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
    print "%i " % (num)
```

Example 3-27. Scala squaring the values in an RDD

```
val input = sc.parallelize(List(1, 2, 3, 4))
val result = input.map(x => x * x)
println(result.collect().mkString(","))
```

map vs. flatmap

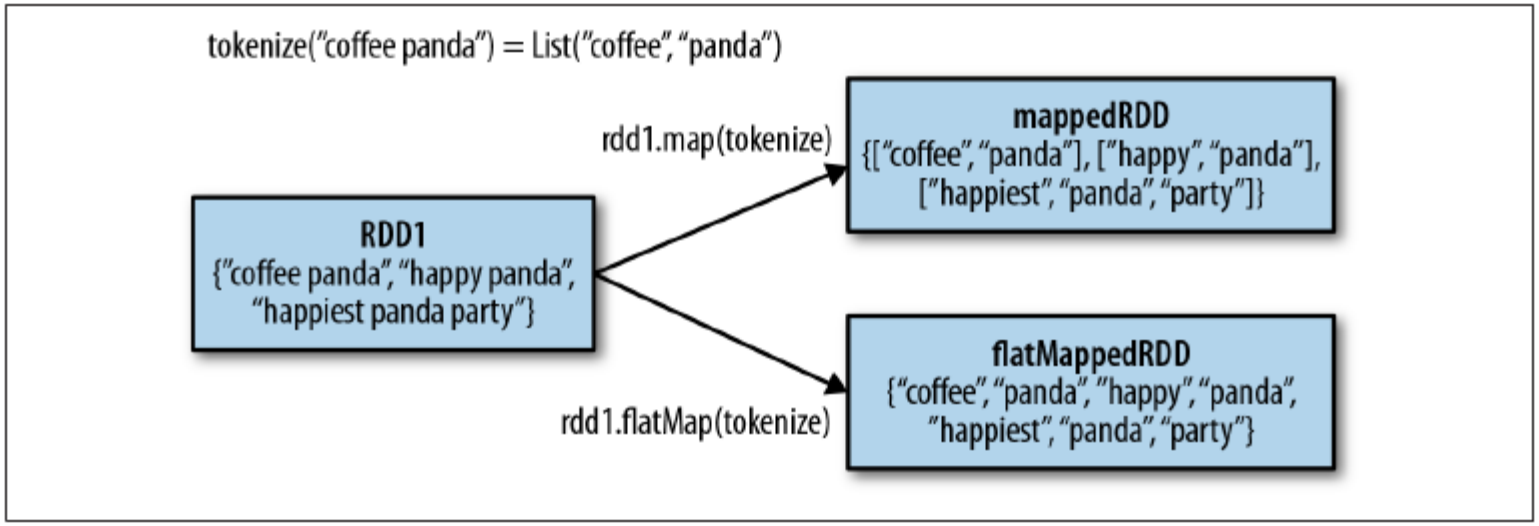


Figure 3-3. Difference between flatMap() and map() on an RDD

Example 3-29. flatMap() in Python, splitting lines into words

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first()  # returns "hello"
```

Example 3-30. flatMap() in Scala, splitting lines into multiple words

```
val lines = sc.parallelize(List("hello world", "hi"))
val words = lines.flatMap(line => line.split(" "))
words.first()  // returns "hello"
```

Ofte brukt *action*: **reduce**(*func)*

- Aggregerer elementa i datasettet ved å bruke funksjon *funk* (som tek to argument og returnerer eitt)

Example 3-32. reduce() in Python

```
sum = rdd.reduce(lambda x, y: x + y)
```

Example 3-33. reduce() in Scala

```
val sum = rdd.reduce((x, y) => x + y)
```

Eksempel m/Spark-shell (Scala)

1:login.idi.ntnu.no - [login] - F-Secure SSH Client

File Edit View Window Help

```
scala> val data = (1 to 10).toArray  // Create array with the given 10 values
data: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> val distData = sc.parallelize(data) // Distribute array over partitions
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[4] at parallelize at <console>:26

scala> distData.reduce((a, b) => a + b) // Add up elements in array and output result
res6: Int = 55

scala> val distData5 = distData.map(a => a + 5) // Add 5 to all elements
distData5: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[5] at map at <console>:28

scala> distData5.collect().foreach(println(_))
6
7
8
9
10
11
12
13
14
15

scala> distData5.reduce((a, b) => a + b) // Add up elements in array and output result
res8: Int = 105

scala>
```

Connected to login.idi.ntnu.no | SSH2 - 3des-cbc - hmac-md5 - none | 108x29 | 8, 29 | 194:57:55

Kunne brukt cache() før denne setninga for å unngå dobbel generering av distData5

RDD transformasjonar & Actions

Oversikt over desse på foilane er berre med for å vise at det er pensum, forventa at de lærer meir om desse (og tilsvarande for pair-RDDS) ved å gå gjennom ”quick tour” lagt ut på Blackboard, og også gjennom prosjektet

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	4
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return `num` elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top `num` elements the RDD.	`rdd.top(2)`	`{3, 3}`
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9

Key/Value-par

- Lage par, eks:
  - Spesielle inputformat frå fil
    - T.d. komma- eller tab-separerte filer
  - Vha. map:

```
// Lage par-RDD ved å bruke første ord som nøkkel
// Dvs. linje => (først ord på linje, linje)
val pairs = lines.map(x => (x.split(" ")(0), x))
```

Eksempel

- Oppgåve: Finne max-temperatur
- Data i tab-separert fil (TSV)

```
1950  0    1
1950  22   1
1950  -11  1
1949  111  1
1949  78   1
```

```
val lines = sc.textFile("input/ncdc/micro-tab/sample.txt")
// Frå String til Array of Strings
val records = lines.map(_.split("\t"))
// Filtrer ut postar med feil
val filtered = records.filter(rec => (rec(1) != "9999" && rec(2).matches("[01459]")))
// Til Key/Value-par
val tuples = filtered.map(rec => (rec(0).toInt, rec(1).toInt)) //
// For kvar nøkkel (år), finn max
val maxTemps = tuples.reduceByKey((a, b) => Math.max(a, b))
maxTemps.collect().foreach(println(_))
```

```
(1950,22)
(1949,111)
```

Eksempel: Teljing av termar

Deer Bear River
Car Car River
Deer Car Bear

Deer
Bear
River
Car
Car
River
Deer
Car
Bear

(Deer,1)
(Car,1)
(Bear,1)
(Deer,1)
(Bear,1)
(River,1)
(Car,1)
(Car,1)
(River,1)

```
val file = sc.textFile("t.txt")
val counts = file.flatMap(line => line.split(" ")).
            map(word => (word,1)).
            reduceByKey(_ + _).
            sortByKey()

counts.saveAsTextFile("outdir")
```

(Deer,2)
(Bear,2)
(Car,3)
(River,2)

(Bear,2)
(Car,3)
(Deer,2)
(River,2)

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey(x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`
`sortByKey()`	Return an RDD sorted by the key.	`rdd.sortByKey()`	`{(1, 2), (3, 4), (3, 6)}`
`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`

Table 4-2. Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)

Function name	Purpose	Example	Result
`subtractByKey`	Remove elements with a key present in the other RDD.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`

Par-RDD-”actions”

- I tillegg til dei nemnt tidlegare:

Table 4-3. Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

Persistens-nivå: RDD.persist(level)

Table 3-6. Persistence levels from org.apache.spark.storage.StorageLevel and pyspark.StorageLevel; if desired we can replicate the data on two machines by adding _2 to the end of the storage level

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	High	Low	Y	N	
MEMORY_ONLY_SER	Low	High	Y	N	
MEMORY_AND_DISK	High	Medium	Some	Some	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.
DISK_ONLY	Low	High	N	Y	

(SER: Serialiserte Java-objekt, dvs. eit byte-array i staden for mange objekt)

RDD.cache() ekvivalent med RDD.persist(MEMORY_ONLY)

Representasjon av RDDs

- Viktige metadata:
  - Partisjonar (atomiske delar av datasettet) og lokasjonar for desse
    - T.d. RDD som representerer HDFS-fil har ein partisjon for kvar blokk og veit lokasjon for desse
  - Partisjonering: Om (og korleis) RDD er hash/range-partisjonert
  - Avhengigheiter (foreldre-RDDs): Kva RDD(s) ein RDD er basert på
    - Narrow dependency: Kvar partisjon i forelder-RDD brukt av max ein partisjon i barn-RDD
    - Wide dependency (shuffle): Fleire barn-partisjonar kan vere avhengig av ein forelder-RDD

Narrow vs. wide dependency

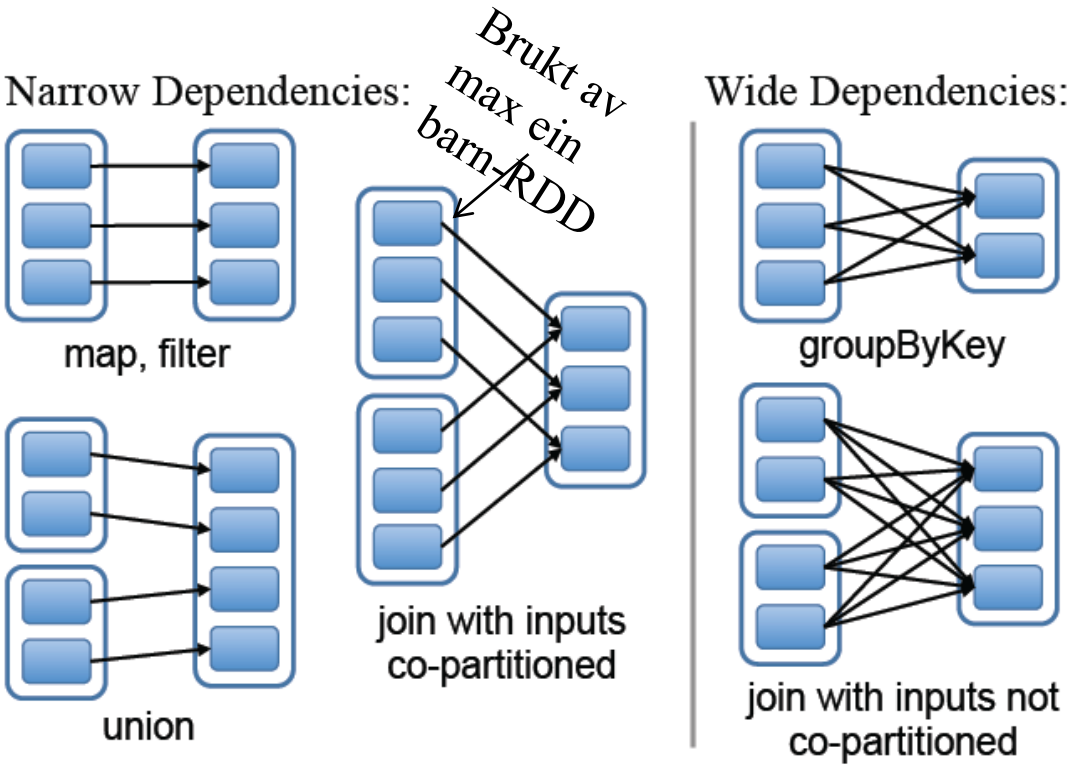


Figure 4: Examples of narrow and wide dependencies. Each box is an RDD, with partitions shown as shaded rectangles.

- Skilnad mellom narrow og wide viktig:
  - Narrow: samlebands-utrekning på ei node, recovery enklare/billegare
  - Wide: shuffle nødvendig

Eksempel på utføring i Spark

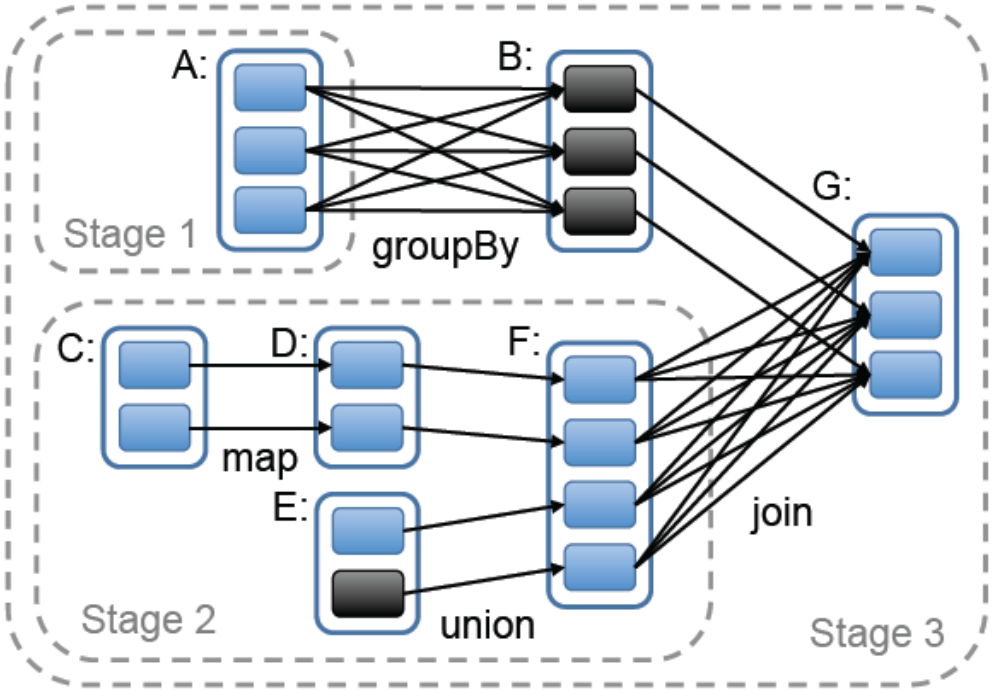


Figure 5: Example of how Spark computes job stages. Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory. To run an action on RDD G, we build build stages at wide dependencies and pipeline narrow transformations inside each stage. In this case, stage 1's output RDD is already in RAM, so we run stage 2 and then 3.

Application
Jobs
Stages
Tasks

Dataframes og Datasets

- DataFrames:
  - Distribuert samling med *rad-objekt*
  - Namngjevne kolonner
  - Tilsvarande *tabell* i relasjonsdatabasesystem
  - Kan spørjast med SQL
  - Logisk plan og optimalisering ved utføring
  - Rask/effektiv intern representasjon
- Datasets:
  - Hovudsakleg som DataFrames, men *"type-sikker"*
- Eksempel:

// If you want a copy of file: https://folk.idi.ntnu.no/noervaag/TDT4305/

```
val tweetsDF = spark.read.json("tweets.json.gz")
tweetsDF.printSchema()
tweetsDF.createOrReplaceTempView("tweets")
val numPopularUsers =
    spark.sql("SELECT user.screen_name FROM tweets
    WHERE user.followers_count > 5000").count()
```

Eksempel på tweet i json-format

{"in_reply_to_status_id_str":null,"in_reply_to_status_id":null,"created_at":"Sat Dec 26 14:53:24 +0000 2015","in_reply_to_user_id_str":null,"source":"<a href=\"http://twitter.com\" rel=\"nofollow\">Twitter Web Client<\/a>","retweet_count":0,"retweeted":false,"geo":null,"filter_level":"low","in_reply_to_screen_name":null,"is_quote_status":false,"id_str":"680763346614890496","in_reply_to_user_id":null,"favorite_count":0,"id":680763346614890496,"text":"Our poster \"Top-k Dominating Queries, in Parallel, in Memory\" w/Orestis Gkorgkas and Sean Chester accepted at #EDBT2016!","place":{"country_code":"NO","country":"Norge","full_name":"Norway","bounding_box":{"coordinates":[[[-9.083133,57.956207],[-9.083133,80.824562],[33.63209,80.824562],[33.63209,57.956207]]],"type":"Polygon"},"place_type":"country","name":"Norway","attributes":{},"id":"0ce8b9a7b2742f7e","url":"https://api.twitter.com/1.1/geo/id/0ce8b9a7b2742f7e.json"},"lang":"en","favorited":false,"coordinates":null,"truncated":false,"timestamp_ms":"1451141604461","entities":{"urls":[],"hashtags":[{"indices":[110,119],"text":"EDBT2016"}],"user_mentions":[],"symbols":[]},"contributors":null,"user":{"utc_offset":3600,"friends_count":243,"profile_image_url_https":"https://pbs.twimg.com/profile_images/470768125/noervaag_2000px_normal.jpg","listed_count":9,"profile_background_image_url":"http://abs.twimg.com/images/themes/theme1/bg.png","default_profile_image":false,"favourites_count":84,"description":"Professor at NTNU, Trondheim, Norway.","created_at":"Wed Oct 14 11:30:14 +0000 2009","is_translator":false,"profile_background_image_url_https":"https://abs.twimg.com/images/themes/theme1/bg.png","protected":false,"screen_name":"noervaag","id_str":"82337208","profile_link_color":"0084B4","id":82337208,"geo_enabled":true,"profile_background_color":"C0DEED","lang":"en","profile_sidebar_border_color":"C0DEED","profile_text_color":"333333","verified":false,"profile_image_url":"http://pbs.twimg.com/profile_images/470768125/noervaag_2000px_normal.jpg","time_zone":"Copenhagen","url":"http://www.idi.ntnu.no/~noervaag/","contributors_enabled":false,"profile_background_tile":false,"statuses_count":295,"follow_request_sent":null,"followers_count":240,"profile_use_background_image":true,"default_profile":true,"following":null,"name":"Kjetil NÃ¸rvÃ¥g","location":"Trondheim, Norge","profile_sidebar_fill_color":"DDEEF6","notifications":null}}

```
scala> val tweetsDF = spark.read.json("tweets.json.gz")
20/01/27 15:01:06 WARN util.Utils: Truncated the string representation of a plan since
it was too large. This behavior can be adjusted by setting
'spark.debug.maxToStringFields' in SparkEnv.conf.
tweetsDF: org.apache.spark.sql.DataFrame = [_corrupt_record: string, contributors:
string ... 31 more fields]
scala> tweetsDF.printSchema()
root
 |-- coordinates: struct (nullable = true)
 |-- created_at: string (nullable = true)
 |-- id_str: string (nullable = true)
 |-- in_reply_to_screen_name: string (nullable = true)
....
 |-- retweet_count: long (nullable = true)
 |-- text: string (nullable = true)
 |-- timestamp_ms: string (nullable = true)
 |-- user: struct (nullable = true)
 |    |-- followers_count: long (nullable = true)
 |    |-- following: string (nullable = true)
...and many many more
scala> tweetsDF.createOrReplaceTempView("tweets")
scala> val numPopularUsers =     spark.sql("SELECT user.screen_name FROM tweets
WHERE user.followers_count > 5000").count()
numPopularUsers: Long = 654
```

Prøv Spark sjølve! ☺

På Blackboard:

1) Quickstart to installing tools used in TDT4305
2) Quick Tour of Spark
3) Eksempel-datafiler

For dei som vil vite meir...

- http://spark.apache.org/docs/latest/
- http://spark.apache.org/docs/latest/rdd-programming-guide.html

Sikt

Sikt

Cyber Defense and Big Data

Emil Henry Flakk, systems engineer
Geir Solskinsbakk, data scientist (PhD)
SIKT @ NTNU, 2022-02-03

Sikt

How modern defensive security works and applies big data techniques to achieve its goals.

Sikt

Om **Sikt** – Kunnskapssektorens tjenesteleverandør

- ~80 tjenester
- 370 ansatte
- 700 kunder
- Hovedkontor i Trondheim
- Mellom 1,3 og 1,5 millioner aktive brukere
- Kontorsted i Oslo, Bergen og på hjemmekontorene

Sikt

Samfunnsoppdrag

- **Sikt skal bidra til at virksomheter** (universiteter, forskningsinstitutter, grunnopplæringen osv.) **og brukere** (studenter, forelesere, elever, forskere etc) **i kunnskapssektoren når sine mål**

- **Sikt skal utvikle, forvalte og drifte digitale fellestjenester og infrastruktur**. Blant annet Feide, vitnemålsportalen, samordna opptak, Zoom og forskningsnettet.

- **Sikt skal bidra til sikker digitalisering i kunnskapssektoren** blant annet med cybersikkerhetssenteret og personverntjenester, **og legge til rette for innovasjon.**

Sikt

Noen **få** av tjenestene våre

- Feide
- Eduroam
- **Cybersikkerhetssenter**
- Personvern (NSD)
- Zoom
- Studentweb

Sikt

Cybersikkerhetssenter for forskning og utdanning

- Sektorvist responsmiljø (SRM)
- Varsler som netteier, informasjon fra samarbeidspartnere
- Sikkerhetstjenester

Virksomheten

Eget team for å håndtere hendelser (Incident Response Team, IRT).

Stort sett ansvar for egen drift og sikkerhet.

Sikt

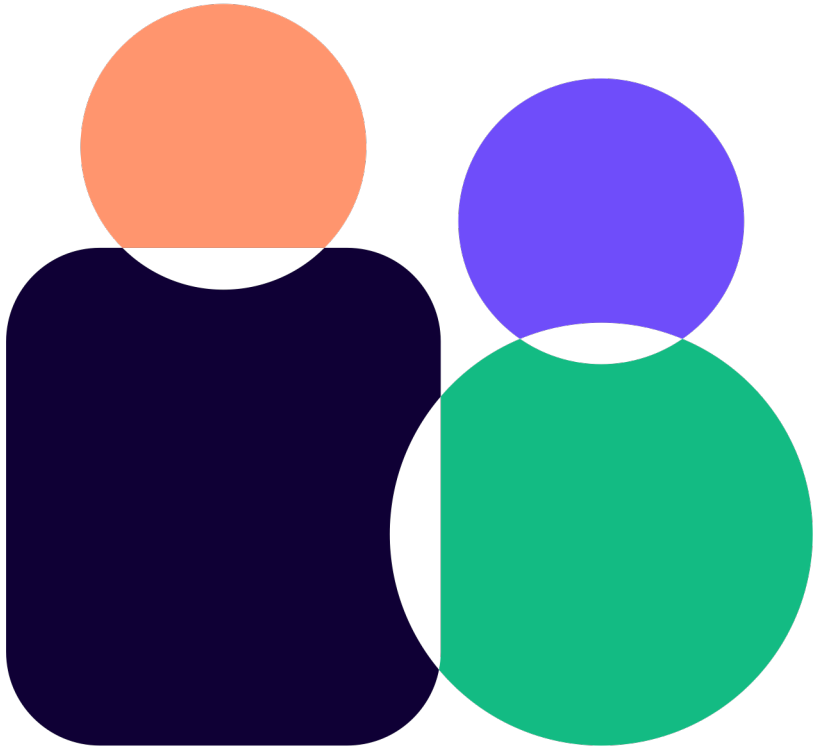
Basic terminology

Incident response (IR). Prioritizing, controlling and resolving (mitigating) incidents.

Forensics. Investigating and determining when, how and why an incident occured.

Threat intelligence. Semi-structured information about threats. Might contain **indicators of compromise** (IoCs), e.g. IPs.

Sensor. Anything that lets us observe something, e.g. application logs, traffic measurements.



Business expectations

Protect the mission

Lessen the impact or likelihood of attacks affecting goals

Enable decisions

Provide feedback and suggestions for solutions to security concerns

Build confidence

"Guard rails" prevent FUD when making agile solutions

Sikt

It's kinda hard to win

> 400 Gbps optic fibers

> **1.5M** daily users of our services

> 500,000 *daily* security alerts, ~100 fields/details

~52M unique IP addresses (14d count)

More SaaS services. More data in general.

Encryption makes networks opaque

Attacker needs only succeed once ☺

Customers (e.g. universities) run the endpoints,

Sikt

Other engineering considerations

Threat detection: Soft realtime

Forensics: Long-term retention. When and how data was processed (provenance)

At-least-once delivery. High availability.

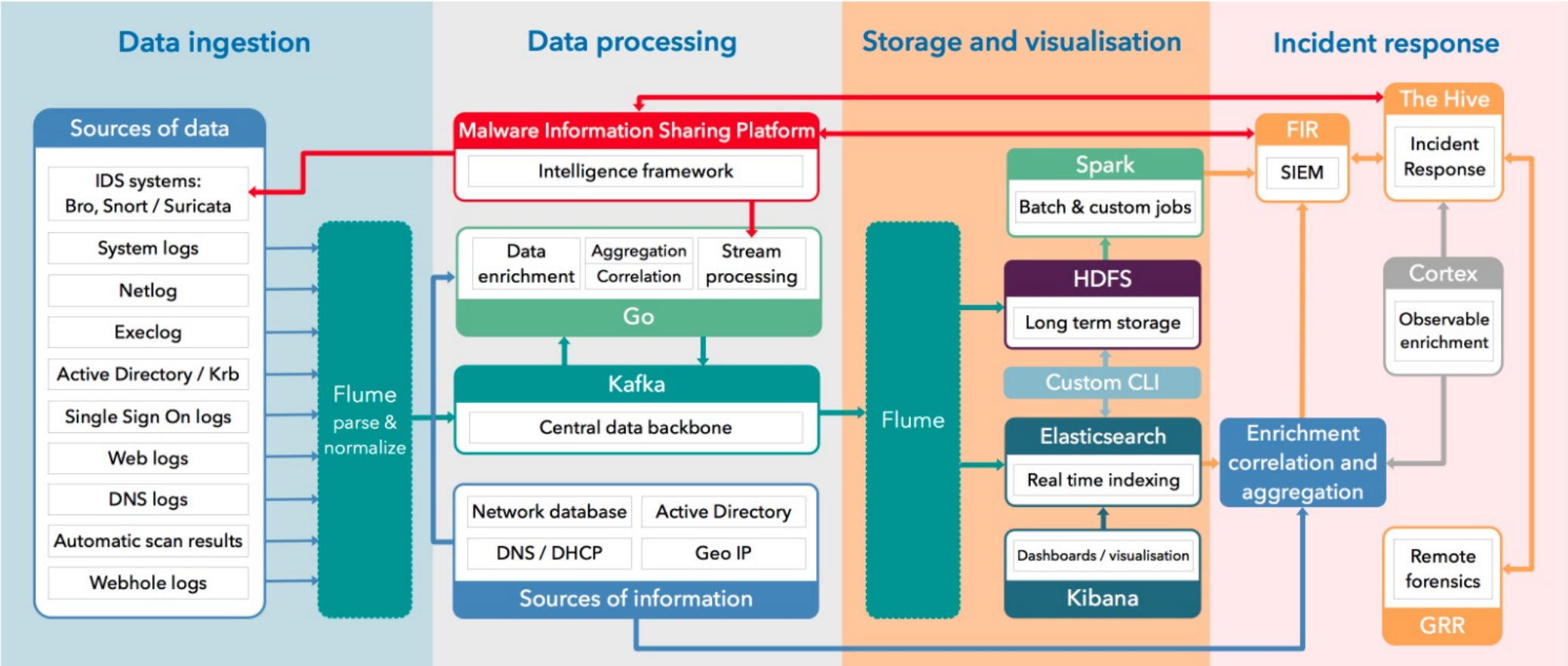
Ad-hoc queries to aid hypothesis-based investigations and innovation.

Needs big fat pipes, usually some kind of data bus (e.g. Kafka) and dataflow mgmt

Security analysis is big data

- 3V: Velocity, volume, variety.
- #1: How to store and process the data, quickly?
  - Stream analysis – mostly for detection
  - OLAP/data warehouses for investigations and producing more intelligence (insights)
- #2: Not enough *human* time to analyse this much data.
  - Bayesian base rate fallacy (too many FPs)
  - The future is machine learning

Example: Modern security infrastructures (CERN)



Sikt

Example of interesting problems: Logs

Find keyword in 1 PB of Apache logs in < 60 seconds

Full-text search is costly and slow. Consider mixed approach:

- #1: Logs are time series. Index the timestamp: "Show me data from this week" (inverted, sorted index)
- #2: CPU is cheap and fast, storage is expensive and slow. Compress the data in order to read and store fewer bytes.
- #3: Bruteforce search on modern hyperscalar processors is actually fast. Use bloom filters to skip chunks of data.
- #4: Store columns separately. Avoids reading the whole row when need a few columns. Improves compresses too.

```
8.211.223.229 - - [03/Feb/2022:14:09:57 +0100] "GET /wp-content HTTP/1.0" 200 5
033 "http://www.murphy.com/wp-content/main.htm" "Mozilla/5.0 (Macintosh; PPC Ma
c OS X 10_10_4; rv:1.9.3.20) Gecko/2016-07-15 00:50:53 Firefox/3.8"
137.153.146.14 - - [03/Feb/2022:14:12:15 +0100] "DELETE /apps/cart.jsp?appID=69
54 HTTP/1.0" 200 4980 "http://shields-hancock.net/faq.html" "Mozilla/5.0 (Macin
tosh; Intel Mac OS X 10_10_9; rv:1.9.2.20) Gecko/2014-06-11 22:34:38 Firefox/3.
8"
119.240.226.82 - - [03/Feb/2022:14:12:45 +0100] "GET /list HTTP/1.0" 404 4892 "
https://powell-stanton.org/explore/login/" "Mozilla/5.0 (iPhone; CPU iPhone OS
14_2_1 like Mac OS X) AppleWebKit/531.2 (KHTML, like Gecko) CriOS/16.0.868.0 Mo
bile/99M735 Safari/531.2"
212.146.162.104 - - [03/Feb/2022:14:15:49 +0100] "DELETE /app/main/posts HTTP/1
.0" 200 5010 "http://jacobs.com/search/" "Mozilla/5.0 (X11; Linux i686) AppleWe
bKit/535.1 (KHTML, like Gecko) Chrome/21.0.838.0 Safari/535.1"
34.246.109.60 - - [03/Feb/2022:14:18:19 +0100] "POST /wp-content HTTP/1.0" 200
5007 "https://www.king-george.com/tags/app/explore/register.php" "Mozilla/5.0 (
X11; Linux x86_64) AppleWebKit/533.0 (KHTML, like Gecko) Chrome/23.0.880.0 Safa
ri/533.0"
91.218.12.226 - - [03/Feb/2022:14:21:22 +0100] "GET /posts/posts/explore HTTP/1
.0" 200 4967 "https://www.hall.net/categories/categories/search/home.html" "Moz
illa/5.0 (iPad; CPU iPad OS 3_1_3 like Mac OS X) AppleWebKit/531.0 (KHTML, like
 Gecko) FxiOS/15.0o1623.0 Mobile/47H311 Safari/531.0"
```

Example of search optimalization

- Initially: **1 PB** of raw logs. Looking for **'contoso.com'**
- Compress them (10:1). **100 TB**.
- Only grab this week's logs (10:1). **10 TB**.
- Bloom filter: only chunks with 'contoso.com' (10:1). **1 TB**.
- Only read the columns we need to (10:1). **100 GB**.
- NVMe PCI 4.0 speed (on a good day): ~5 GB/s.
- 100 GB / 5 GB/s => **20 s**. Can go faster with multiple disks, preferably multiple storage buses

Sikt

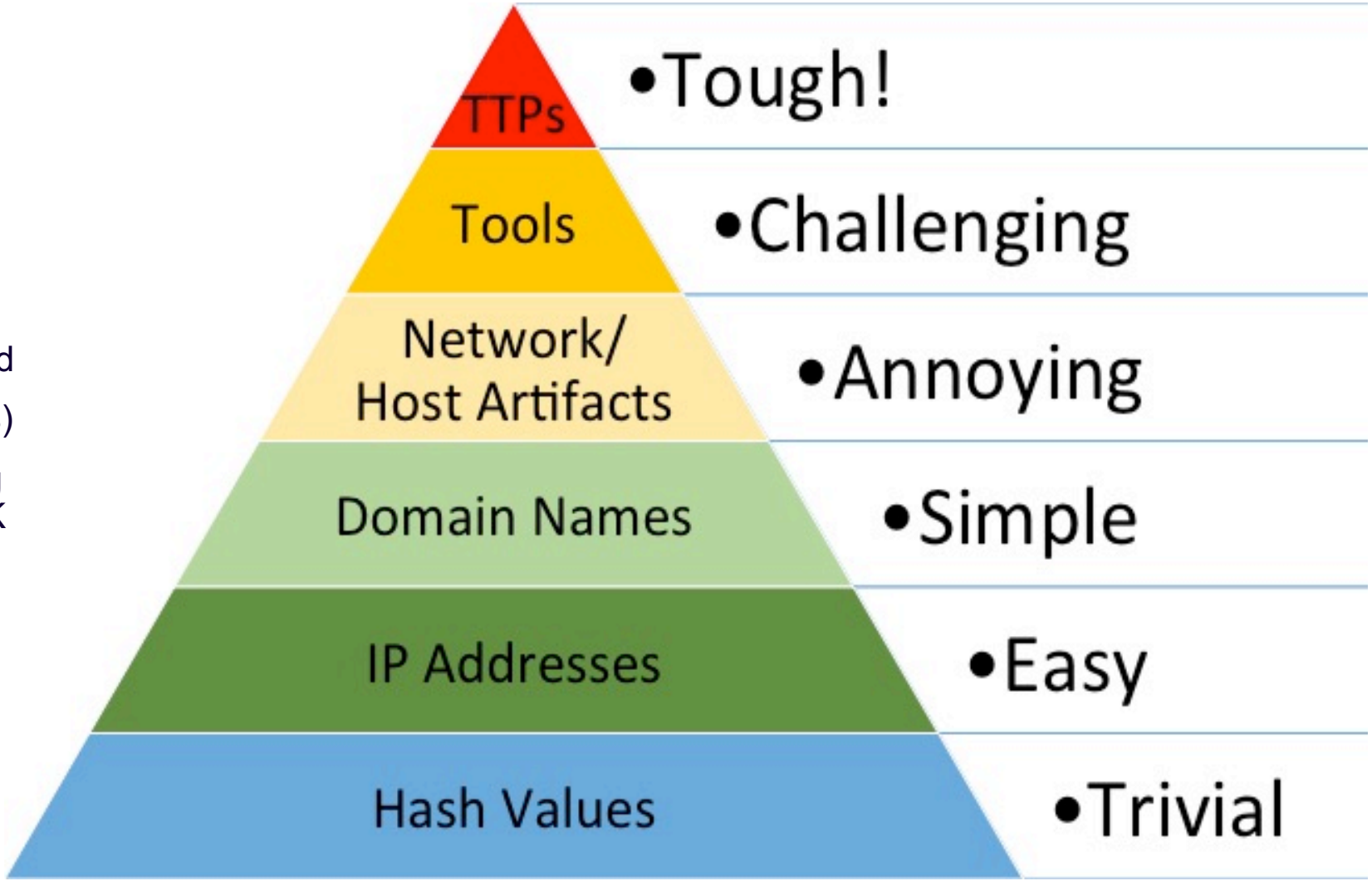
Overcoming adversity

Instead of chasing alerts, remove *opportunities* for attackers to succeed

Tools, Tactics and Procedures (TTPs)

Threat intelligence tracks this using structured data, e.g. MITRE ATT&CK

Build tools to observe and prevent TTPs relevant to our environment



Sikt

Fake login pages

Very convincing clones of service pages across multiple customers

Valid TLS certificates

Hundreds of domains at their disposal. Switches to new domains once detected

Goal: Steal credentials.

Tactic

Common patterns in the naming convention used for certificates

All LetsEncrypt certs are registered in a public log known as **Certificate Transparency (CT)**

Outcome

New streaming detections:

Alert on patterns in newly issued certificates via CT

Alert on DNS lookups to detect compromised users before they're abused.

Could reliably detect and block their attempts 30 minutes in advance

Sikt

Economic fraud

Very easy to start new fraud campaigns. Just change the template and pay Amazon some $$$ to send emails.

Valid TLS certificates

Change domain if detected. Hundreds of them available

Goal: Steal credentials.

Tactic

You need some kind of payment infrastructure, often reliant on *real* humans

Common theme: Financial fraud victims fooled into receiving and sending $$$

Examples of data: SWIFT/IBAN numbers, bitcoin wallet IDs

Outcome

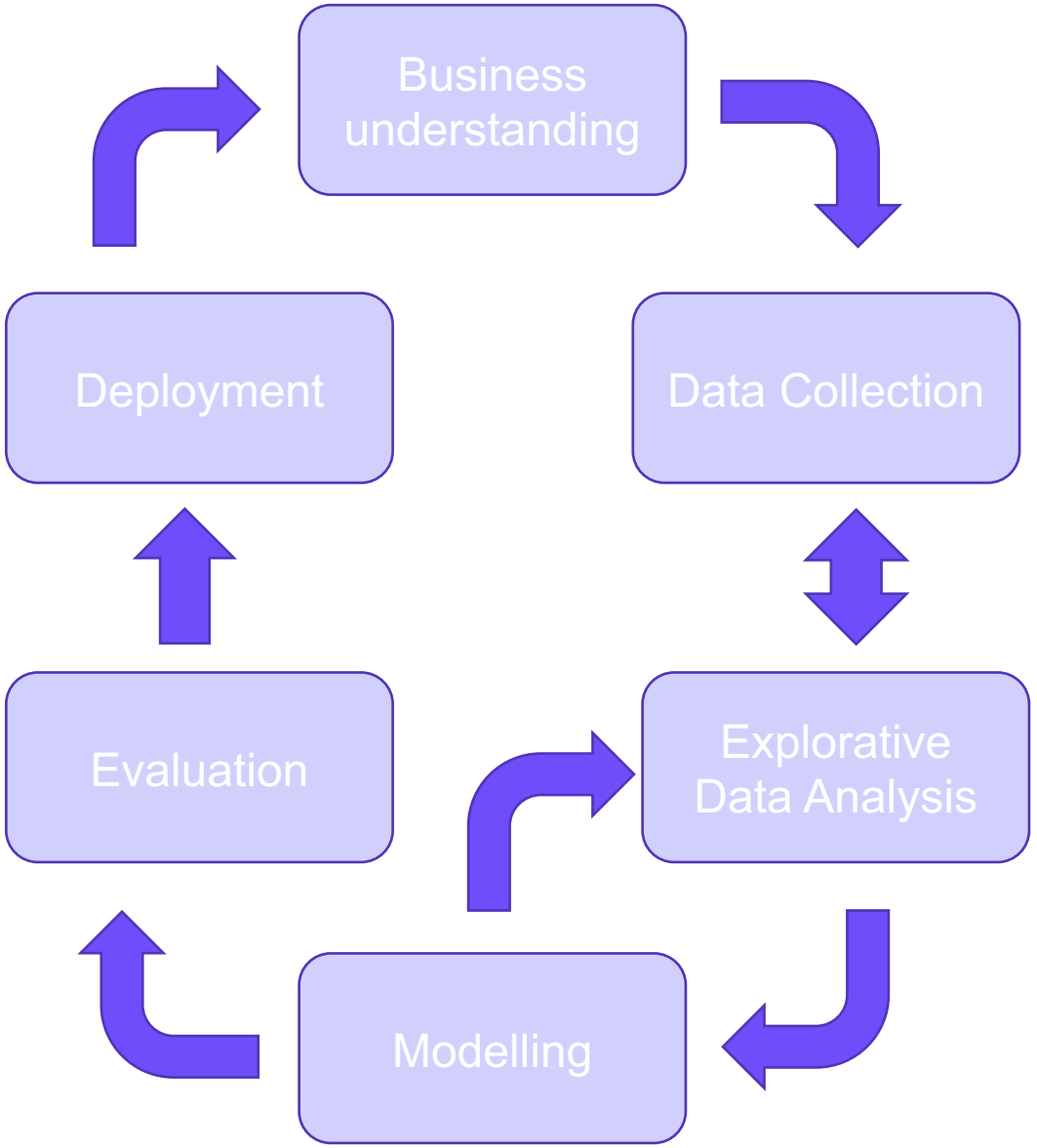
Track and report accounts. Share them with other organizations that might be targeted as well. (Streaming, OLAP analysis, even graph-based techniques)

Attackers run out of viable options to move money and switch to new careers

Sikt

Data Science – what and why?

- Extract knowledge from data
- Increasingly complex threats
  - Manual analysis increasingly complex
  - Machine learning can interpret signals that we as humans can't
- Automate analysis – humans have limited capacity
  - Continuous monitoring of multiple scenarios



Goals for analytics

- Finding the needle in the haystack
  - A diverse set of logs – may need to combine many sources to get the big picture
- Extract insights / alerts / events that are interesting for further analysis
  - Answering one question opens up another
- Start with manual analysis
- Automation of analysis
  - Important factors: scale and response time
- False positives vs. False negatives

OLAP example: Passive DNS intelligence

- DNS queries are usually not encrypted, so let's log them. Drop client IP for privacy reasons.
- Aggregate on (RRname, RRtype, answer), e.g. (vg.no, *A*, 195.88.54.16) + first/last seen.
- "Wow, this domain stores malware instructions in its TXT record"
- "When did someone first visit totallynotavirus[.]com?"
- "Has anyone queried Russian domains containing the name of our company?"

Sikt

Adding context to existing data

- Fact tables: Knowing what the network is used for makes triaging easier (e.g. GeoIP, ASN)
  - VPN traffic in the server DMZ doesn't look good
- What is the entropy of a domain?
  - English has skewed letter distribution and hence will be less random (less entropy)
  - More random domains are likely sampled from uniform distribution and hence machine generated, for use by malware etc

E	13%
T	9.1%
A	8.2%
O	7.5%
I	7%
N	6.7%
S	6.3%
H	6.1%
R	6%
D	4.3%
L	4%
C	2.8%
U	2.8%
M	2.5%
W	2.4%
F	2.2%
G	2%
Y	2%
P	1.9%
B	1.5%
V	0.98%
K	0.77%
J	0.15%
X	0.15%
Q	0.095%
Z	0.074%

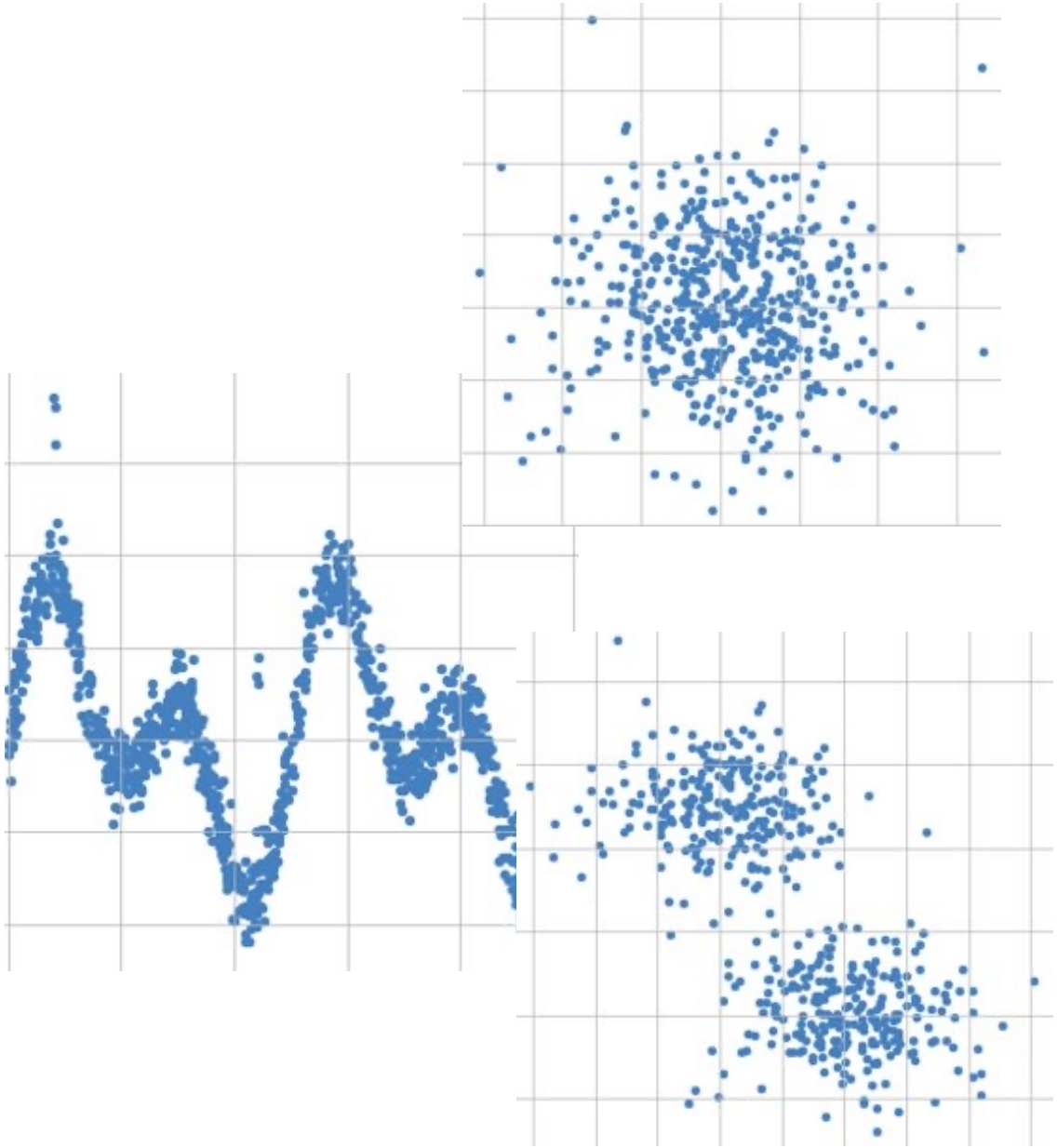
Table of letter frequencies in English language from: https://en.wikipedia.org/wiki/Letter_frequency

Machine Learning

- Supervised
  - Labeled data sets
    - Good labeled data sets are hard to get / costly to make
    - Outdated?
      - A dog will not stop being a dog
      - New attack vectors emerge
  - Classification / regression / ++
- Unsupervised
  - Unlabeled data
  - Discover structures in data
  - Clustering / anomaly detection / ++

Anomaly detection

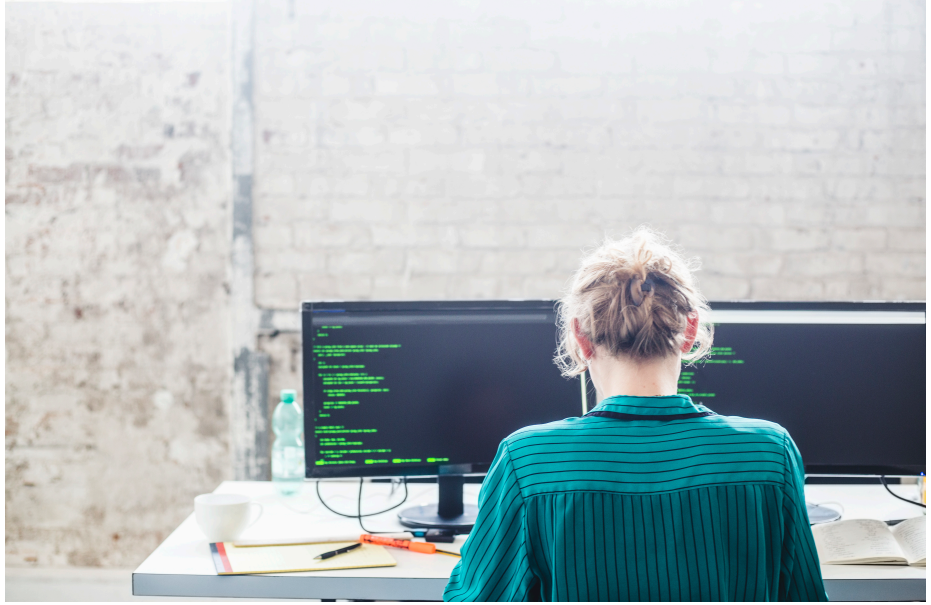
- What is an anomaly?
  - One or more data points that come from a distribution different from the regular data / data generated from a different process
- Measured vs. existing (baseline) data
  - Do we know for certain that the existing data is clean?
- Anomlies can point at interesting parts of the data
  - Pivot points
  - Combine with other data to make more sense



Sikt

Anomaly detection – malicious logins

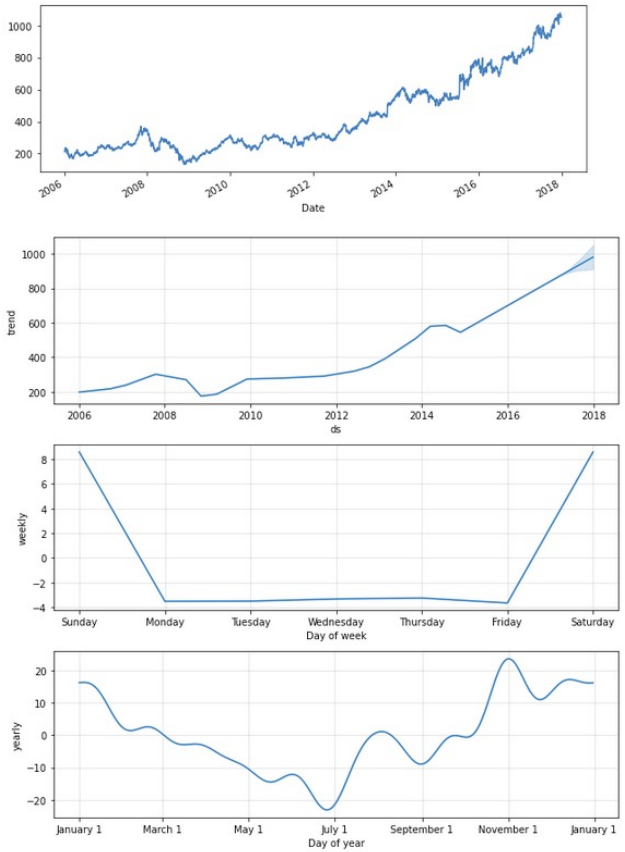
- Goal: Find logins that seem suspicious
  - E.g., password leaked / user hacked / ++
- "login heuristics"
  - Working hours?
  - Location?
  - Behaviour after login?
  - Tor / VPN?
- A login from a usual location in the middle of the night?
- Login from a different continent at an unusual time?



Sikt

Anomaly detection – time series

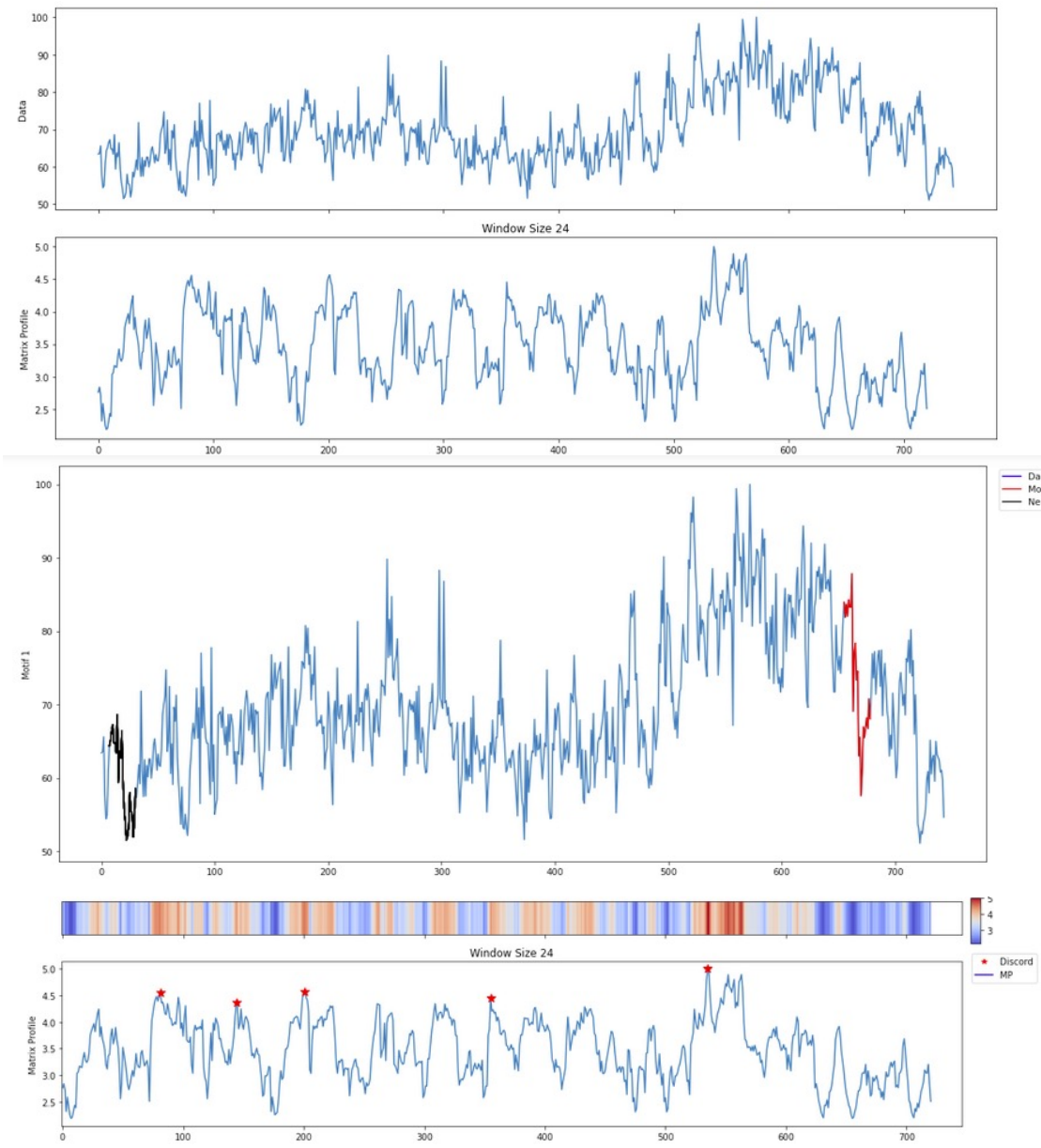
- Forecasting time series
  - Confidence intervals
  - Use this to find outliers / anomalies
    - Trends + uncertainty
  - E.g. prophet from Facebook
  - Alert on spikes in failed logins vs. Expected number of failed logins



Sikt

Anomaly detection – time series II

- Matrix profile
  - Time windows
  - For each time window store the similarity and location of the most similar time window
    - Z-normalized window
- Recognize repeating patterns / anomalous patterns
  - E.g. search for traffic into/out from a server that fits malicious profile
  - Find patterns that don't fit with the "baseline"



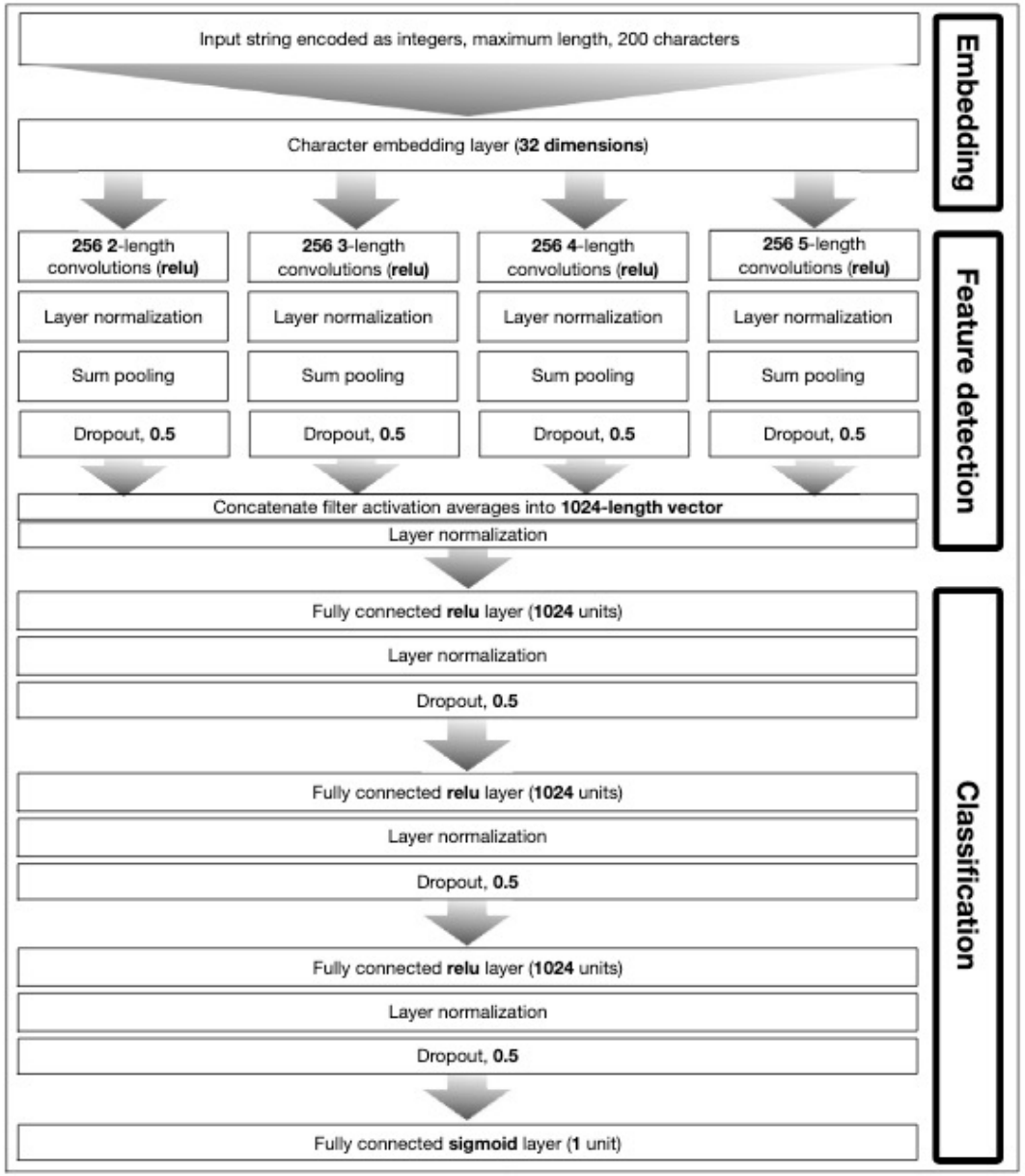
Traffic analysis

- Packet capture analysis
  - Indicators: ips, payloads, signatures ++
- With more encrypted traffic these indicators are to a larger extent hidden from view
- Encrypted traffic analysis
  - Is the conversation benign or malignant?
  - Can't inspect the packages
  - Inspect for (anomolous) patterns in the traffic
    - Timing
    - Sizes
    - Contact points
    - Ports
    - ++

Sikt

Classification – malicious URLs (supervised)

- Goal: recognize malicious URLs
- Dataset
  - Large number of benign URLs
  - Large number of malicious URLs
- Deep learning
  - Learns features – don't need to do a lot of feature engineering



Techinque and figure: Saxe & Berlin: **eXpose: A Character-Level ConvolutionalNeural Network with Embeddings For DetectingMalicious URLs, File Paths and Registry Keys** (https://arxiv.org/pdf/1702.08568.pdf)

Analytics - Summary

- No silver bullets
  - Large number of scenarios that require different techniques (perhaps in combination)
    - Adding up evidence
  - Big variation in data sources
  - Understanding domain and exploration of the data space is key
- Need to keep the false positives at an acceptable level
  - Alert fatigue
- Scaling from manual analysis to automated production environment

DIY

Want to play around with security analytics?

Our recommendation:

- Free data from https://opendata.rapid7.com/
- Shodan.io has a generous free tier for students ☺ Just email their support.

Common software for analytics

- ClickHouse (probably our favorite)
- Spark
- Jupyter notebooks / pandas / scikit-learn / ++
- Feel free to email us for further suggestions ☺

Sikt

Questions?

Shingles, MinHash, LSH

Tomasz Kuśmierczyk: tomaszku@ntnu.no
2017-02-24

What’s obligatory?

Slides marked with star (*) are for people that want a deeper understanding of the topic, and are not curriculum for the exam.

Additional materials:

Book ch. 3 (only some parts apply)

http://infolab.stanford.edu/~ullman/mmds/ch3.pdf

Slides from the book's authors:

http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

Context

- Many real-world applications require **finding “near duplicate”/”similar” pairs:**
  - Pages with similar words
  - Customers who purchased similar products
  - Products with similar customer sets
  - Images with similar features
  - Users who visited similar websites

Today’s Setting

- Goal: Given a large number (in the millions or billions) of documents, **find “near duplicate” pairs**

http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

Today’s Setting

- Goal: Given a large number (in the millions or billions) of documents, find “near duplicate” pairs
- **Applications**:
  - Mirror websites, or approximate mirrors
    - Don’t want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by “same story”

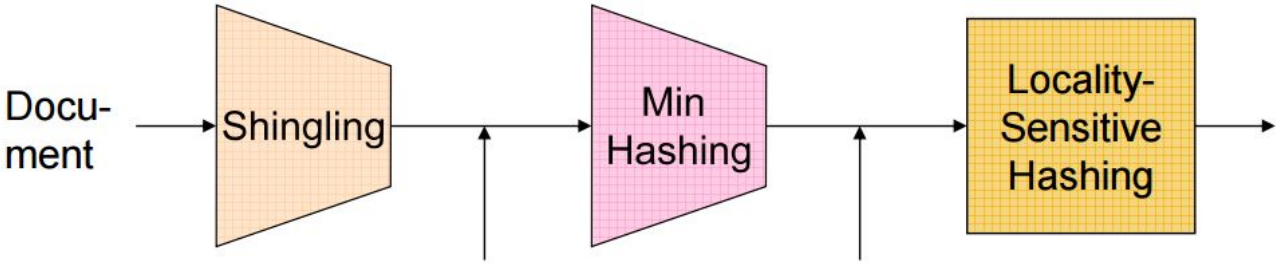
http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

Today's Setting

- Goal: Given a large number (in the millions or billions) of documents, find "near duplicate" pairs
- Applications:
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by "same story"
- **Problems**:
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

What are we going to see today?



Candidate pairs: those pairs of signatures that we need to test for similarity

The set of strings of length ***k*** that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity

Representations

Shingles: docs -> sets conversion

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc
- Tokens can be characters, words or something else, depending on the application

Example: bigrams

- k=2

- document D1 = knowledge is important
- bigrams: {knowledge is, is important}

- document D2 = nowadays imagination is important
- bigrams: {nowadays imagination, imagination is, is important}

Example: chars

- k=2

- document D1 = abcab
- 2-shingles: {ab, bc, ca}

- document D2 = caab
- 2-shingles: {?, ?}

Example: chars

- k=2

- document D1 = abcab
- 2-shingles: {ab, bc, ca}

- document D2 = caab
- 2-shingles: {ab, aa, ca}

What k?

- k should be chosen that:
  - Probability of two documents having the same shingle by chance is low
  - If two documents are similarity they contain at least one matching shingle

What k?

- k should be chosen that:
  - Probability of two documents having the same shingle by chance is low
  - If two documents are similarity they contain at least one matching shingle

How many 10-shingles are there?

- k=10, ASCII chars (27)
  - 27^10 ~ 2 * 10^14 - pretty many
- But most (99.9%) of them are unlikely
- Let’s compress:
  - represent a document by the set of hash values of its k-shingles:
  - Abcab -> {ab, bc, ca} -> {1, 5, 7} (2^32 numbers)

Jaccard

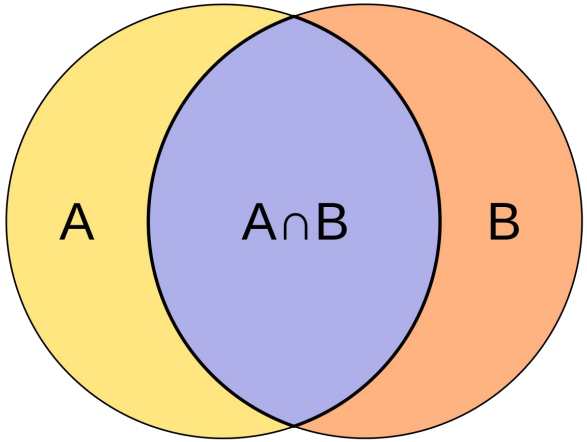
Similarity Measures: Jaccard Similarity

- we have docs as sets (for example of n-grams)

but

- **what “near duplicated” / “similar” means?**

Jaccard Similarity



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

coffee

Jaccard Similarity example

knowledge is very very very important

imagination is more important than knowledge

abs is more important than imagination

potato is the most important

Jaccard Similarity example

knowledge is very very very important

S_1 = *{knowledge, is, very, important}*

imagination is more important than knowledge

S_2 = *{imagination, is, more, important, than, knowledge}*

abs is more important than imagination

S_3 = *{abs, is, more, important, than imagination}*

potato is the most important

S_4 = *{potato, is, the, most, important}*

Jaccard Similarity example

knowledge more most potato than the

music is important very and

cabbage imagination

S_1 = {*knowledge, is, very, important*}

S_2 = {*imagination, is, more, important, than, knowledge*}

Preprocessing is important [Assignment]

potato

knowledge

music

important

cabbage

imagination

S_1 = {*knowledge*, *important*}

S_2 = {*imagination*, *important*, *knowledge*}

Preprocessing is important

potato

knowledge

music

important

cabbage

imagination

S_1 = {*knowledge*, *important*}

S_4 = {*potato*, *important*}

MinHashing

Setting

- Goal: Find similar documents (sets)
  - Documents are already converted to sets
- Similarity between documents = J
- **Documents are large**
  - Example: 10000 characters per doc, 9-shingles
  - How much additional space for representations?
- Every document is compared many times

How do we solve it?

- Comparing is too slow!
- We need: smaller representations of documents (=sets) that can be compared even faster

How do we solve it?

- Comparing is too slow!
- We need: smaller representations of documents (=sets) that can be compared even faster
  - Like: MinHash signatures!

Sets can be encoded as bit (column) vectors [Assignment]

For now all possible elements (words) = $\{a,g,i,k,p\}$

Sets (documents) = $\{S_1, S_2, S_3, S_4\}$

$S_1 = \{k, i\} \rightarrow S_4 = (0\ 0\ 1\ 1\ 0)^T$

$S_2 = \{g, i, k\} \rightarrow S_4 = (0\ 1\ 1\ 1\ 0)^T$

$S_3 = \{a, i, g\} \rightarrow S_4 = (1\ 1\ 1\ 0\ 0)^T$

$S_4 = \{p, i\} \rightarrow S_4 = (???)^T$

Sets can be encoded as bit (column) vectors

For now all possible elements (words) = $\{a,g,i,k,p\}$

Sets (documents) = $\{S_1, S_2, S_3, S_4\}$

$S_1 = \{k, i\} \rightarrow S_4 = (0\ 0\ 1\ 1\ 0)^T$

$S_2 = \{g, i, k\} \rightarrow S_4 = (0\ 1\ 1\ 1\ 0)^T$

$S_3 = \{a, i, g\} \rightarrow S_4 = (1\ 1\ 1\ 0\ 0)^T$

$S_4 = \{p, i\} \rightarrow S_4 = (0\ 0\ 1\ 0\ 1)^T$

Matrix representation

Ixs <-> Elements

Ix		S1	S2	S3	S4
0	a	0	0	1	0
1	g	0	1	1	0
2	i	1	1	1	1
3	k	1	1	0	0
4	p	0	0	0	1

Minhash Signature: random permutations + min

Ix		S1	S2	S3	S4
0	a	0	0	1	0
1	g	0	1	1	0
2	i	1	1	1	1
3	k	1	1	0	0
4	p	0	0	0	1

Permutate rows

Ix		S1	S2	S3	S4	Ix2
4	p	0	0	0	1	0
0	a	0	0	1	0	1
1	g	0	1	1	0	2
2	i	1	1	1	1	3
3	k	1	1	0	0	4

Min(Ix2) containing element

	S1	S2	S3	S4
h1	3	2	1	0

First row of a signature

Minhash Signature: random permutations + min [Assignment]

Ix		S1	S2	S3	S4
0	a	0	0	1	0
1	g	0	1	1	0
2	i	1	1	1	1
3	k	1	1	0	0
4	p	0	0	0	1

Permutate rows

Ix		S1	S2	S3	S4	Ix2
3	k	1	1	0	0	0
0	a	0	0	1	0	1
2	i	1	1	1	1	2
4	p	0	0	0	1	3
1	g	0	1	1	0	4

Min(Ix2) containing element

	S1	S2	S3	S4
h2	?	?	?	?

Second row of a signature

Minhash Signature: random permutations + min

Ix		S1	S2	S3	S4
0	a	0	0	1	0
1	g	0	1	1	0
2	i	1	1	1	1
3	k	1	1	0	0
4	p	0	0	0	1

Permutate rows

Ix		S1	S2	S3	S4	Ix2
3	k	1	1	0	0	0
0	a	0	0	1	0	1
2	i	1	1	1	1	2
4	p	0	0	0	1	3
1	g	0	1	1	0	4

Min(Ix2) containing element

	S1	S2	S3	S4
h2	0	0	1	2

Second row of a signature

Minhash Signature

	S1	S2	S3	S4
h1	3	2	1	0

	S1	S2	S3	S4
h2	0	0	1	2

	S1	S2	S3	S4
h1	3	2	1	0
h2	0	0	1	2

Minhash Signature

Ix		S1	S2	S3	S4
0	a	0	0	1	0
1	g	0	1	1	0
2	i	1	1	1	1
3	k	1	1	0	0
4	p	0	0	0	1

	S1	S2	S3	S4
h1	3	2	1	0
h2	0	0	1	2

Great! We have only smaller fixed (here =2) number of signature rows (= new elements) to deal with. Not-even binary…

How to compare them to get similarity?

Derivation: probability? *

Ix		S1	S2	Ix2
4	k	1	1	0
0	a	0	0	1
1	i	1	1	2
2	p	0	0	3
3	g	0	1	4

Let's calc the **probability** of:

h(S1) = h(S2)

for some random permutation

Derivation: Row types *

Ix		S1	S2	Ix2
4	k	1	1	0
0	a	0	0	1
1	i	1	1	2
2	p	0	0	3
3	g	0	1	4

We have three row types:

- x rows with 1 in both cols
- y rows with 1 in one col
- z rows with 0 in both cols

Derivation: observation *

Ix		S1	S2	Ix2
4	k	1	1	0
0	a	0	0	1
1	i	1	1	2
2	p	0	0	3
3	g	0	1	4

Let's calc the probability of:
h(S1) = h(S2)
for some random permutation

Observation 1: gray rows can change h(S1) or h(S2) but can not change if h(S1) = h(S2)

Derivation: green and red left *

Observation 1: gray rows can change h(S1) or h(S2) but can not change if h(S1) = h(S2)

We can ignore them for now!

Ix		S1	S2	Ix2
4	k	1	1	0
1	i	1	1	2
3	g	0	1	4

Now:

h(S1) = h(S2) if the first row is green

h(S1) != h(S2) if the first row is green

Derivation: what's the solution? *

Now:

h(S1) = h(S2) if the first row is red

h(S1) != h(S2) if the first row is green

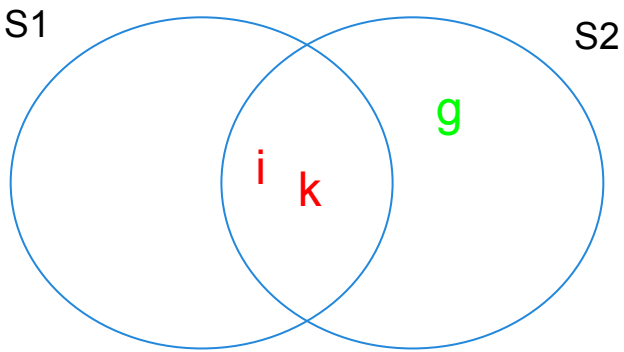
Ix		S1	S2	Ix2
4	k	1	1	0
1	i	1	1	2
3	g	0	1	4

There are x red rows and y green rows

For how many permutations red is on the top?

Derivation: what's the solution? *

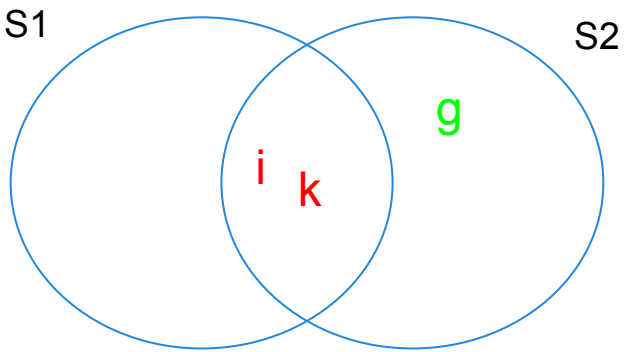
P[h(S1) = h(S2)] = x / (x+y) = ?



Ix		S1	S2	Ix2
4	k	1	1	0
1	i	1	1	2
3	g	0	1	4

Derivation: what's the solution?

P[h(S1) = h(S2)] = x / (x+y) = J(S1,S2)



Ix		S1	S2	Ix2
4	k	1	1	0
1	i	1	1	2
3	g	0	1	4

Problem?

Nice! We know that there is some probability s:

$$s = P[\ h(S1) = h(S2)\ ] = J(S1,S2)$$

But we do not observe probabilities!
We observe events:
if h(S1) = h(S2) for some h or not

Problem?

We need to estimate somehow this probability s:

- Permute i = 1..H times
- Check how many times: $h_i(S1) = h_i(S2)$
- Then the estimator of s:

$$s_{MLE} = \# \, h_i(S1) = h_i(S2) \, / \, H$$

Back to our example: results vs true

	S1	S2	S3	S4
h1	3	2	1	0
h2	0	0	1	2

Reality

- Matrix representation would be huge
- Materializing a permutation takes a lot of space
- What with a new document?

Solutions:

- We do not store the data as a matrix
- We do not permute -> we simulate permutating with hash functions
- Signatures are computed iteratively

Simulating permutations: general formula

$$\mathbf{ix2(ix) = ((a \cdot ix + b) \bmod p)}$$

For and a,b and prime p

Simulating permutations: *example*

$$ix2(ix) = ((3 \cdot ix+1) \bmod 5)$$

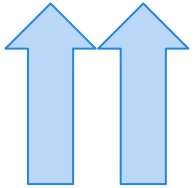
Ix	Ix2
0	1
1	4
2	2
3	0
4	3

Algorithm to calc a signatures of all docs (from the book)

1. Initialize signatures with inf
2. For every element indexed with ix:
    a. For every perumutation:
        i. For every document:
            1. If element not present in a doc skip, otherwise:
            2. signature = min(signature, ix2(ix))

Algorithm example *(blackboard)*

Ix		S1	S2	S3	S4	Ix2	Ix2
0	a	0	0	1	0	1	1
1	g	0	1	1	0	2	4
2	i	1	1	1	1	3	2
3	k	1	1	0	0	4	0
4	p	0	0	0	1	0	3



h1 h2

1. Initialize signatures with inf
2. For every element indexed with ix:
   a. For every perumutation:
      i. For every document:
         1. If element not present in a doc skip, otherwise:
         2. signature = min(signature, ix2(ix))

Algorithm sample

Ix		S1	S2	S3	S4	Ix2	Ix2
0	a	0	0	1	0	1	1
1	g	0	1	1	0	2	4
2	i	1	1	1	1	3	2
3	k	1	1	0	0	4	0
4	p	0	0	0	1	0	3

h1 h2

	S1	S2	S3	S4
h1	inf	inf	inf	inf
h2	inf	inf	inf	inf

	S1	S2	S3	S4
h1	inf	inf	1	inf
h2	inf	inf	1	inf

	S1	S2	S3	S4
h1	inf	2	1	inf
h2	inf	4	1	inf

	S1	S2	S3	S4
h1	3	2	1	3
h2	2	2	1	2

	S1	S2	S3	S4
h1	3	2	1	3
h2	0	0	1	2

	S1	S2	S3	S4
h1	3	2	1	0
h2	0	0	1	2

LSH

Number of pairs problem

- We know how to compare 2 documents in a short (for H fixed = O(1)) time

But still:

- with 10M documents we have ~10M^2 pairs = few days of computation
- Solution: LSH

LSH: General idea

- Use a function f(S1,S2) that tells whether S1 and S2 is a candidate pair
- candidate pair = a pair of elements whose similarity must be evaluated

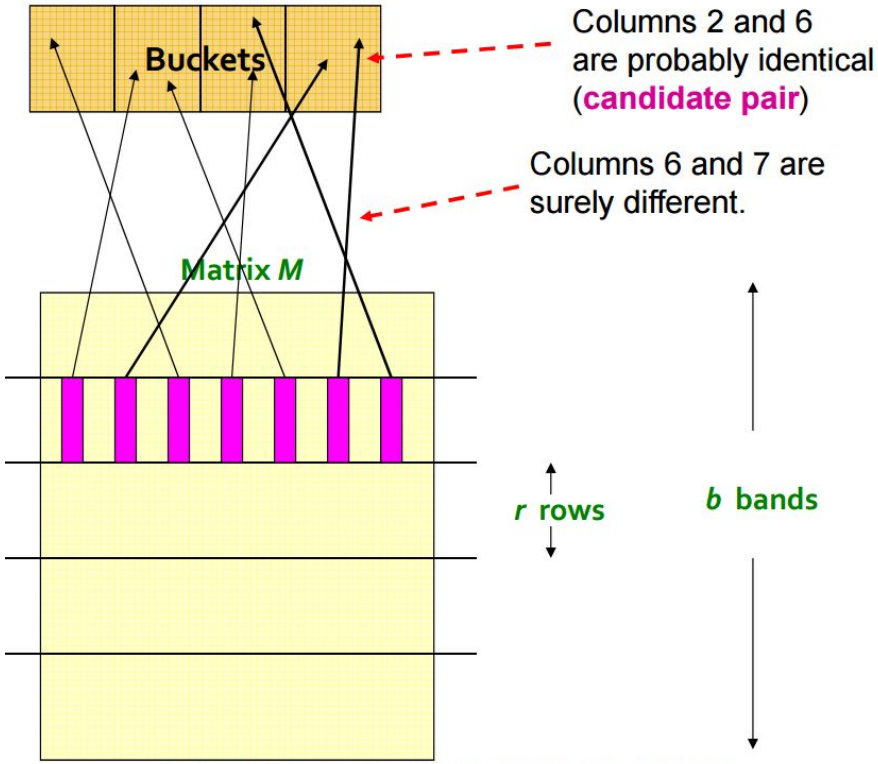
  (to check if they are really similar)

  <u>Number of candidate pairs << number of all pairs</u>
- For some **similarity threshold t** we want
  - Almost all pairs S1, S2 with s(S1,S2) >= t
  - Almost none pairs with s(S1,S2) < t

LSH: algorithm

- Input: Signature Matrix (Docs x Signatures)
- Output: list of candidate pairs S1, S2 to be later evaluated with s(S1,S2)

Overview & algorithm



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

- Split M into b bands with r rows each
- For each doc (column):
  - For each band:
    - Hash band signature part into one of buckets
- Go over all buckets for all bands:
  - If there are two docs in one bucket
    -> candidate pair

Analysis *

Probability that signatures between S1,S2 **agree in one row: s** (like in MinHash)

What's the **probability** that S1,S2 will become **a candidate pair**?

Analysis *

At least one band identical

No bands identical

$$1 - (1 - s^r)^b$$

Some row of a band unequal

All rows of a band are equal

Analysis *

Probability that signatures between S1,S2 **agree in one row: s**

What's the **probability** that S1,S2 will become **a candidate pair**?

$$P(\text{candidate}) = 1 - (1-s^r)^b$$

Example: calc one curve, r=5, b=10 *

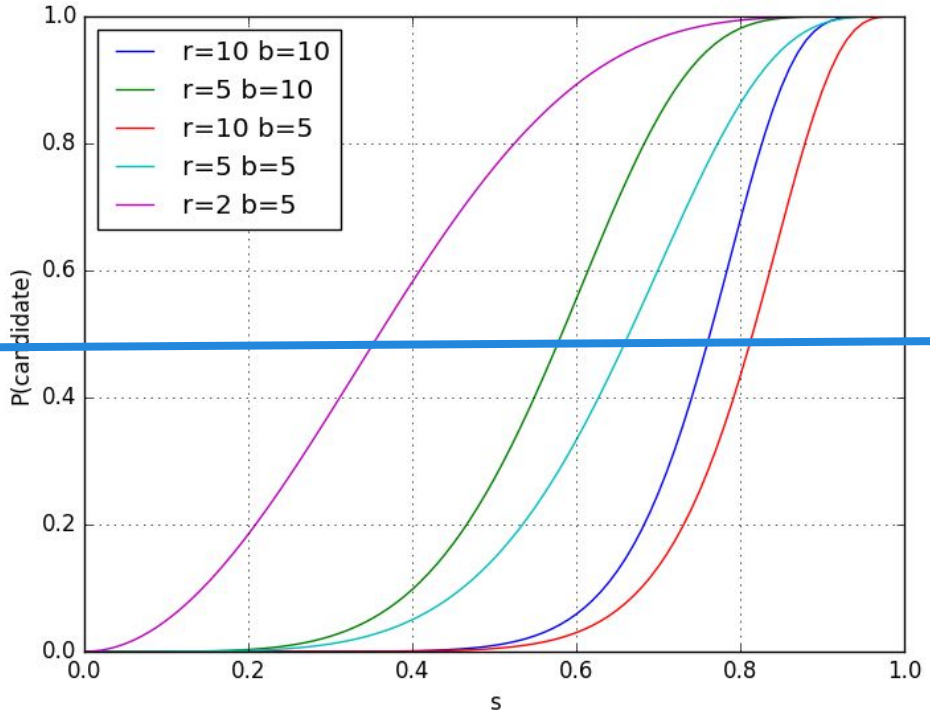
```
import numpy as np


s = np.array(range(0,5))/10
pc = lambda r,b: 1-(1-s**r)**b

print(list(s))
print(list(pc(5,10)))
```

Analysis *

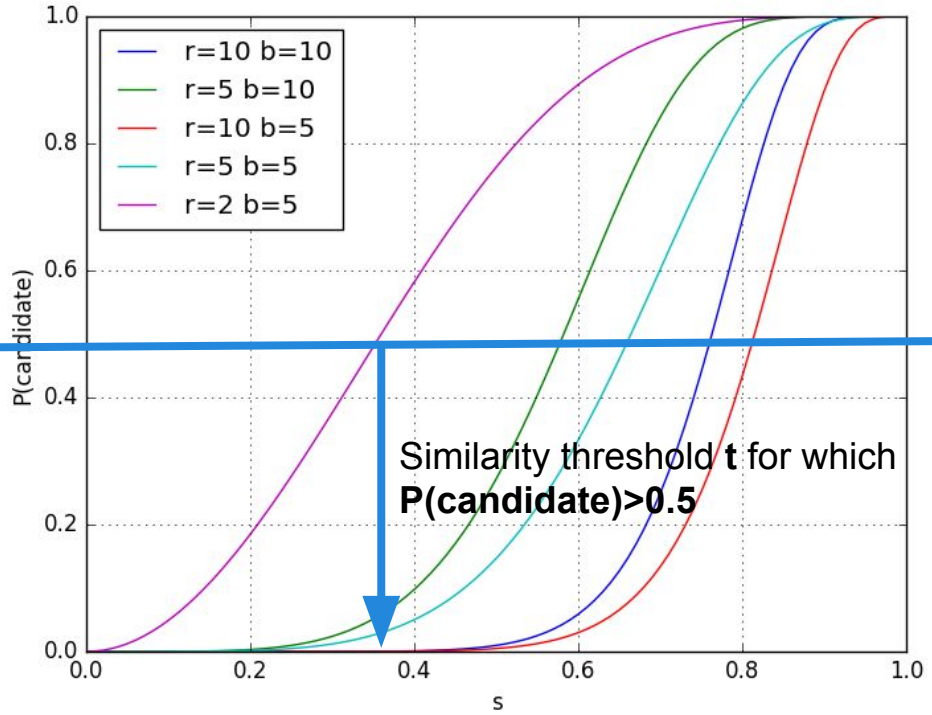
Pairs more likely to be found



Pairs that most likely will not be find

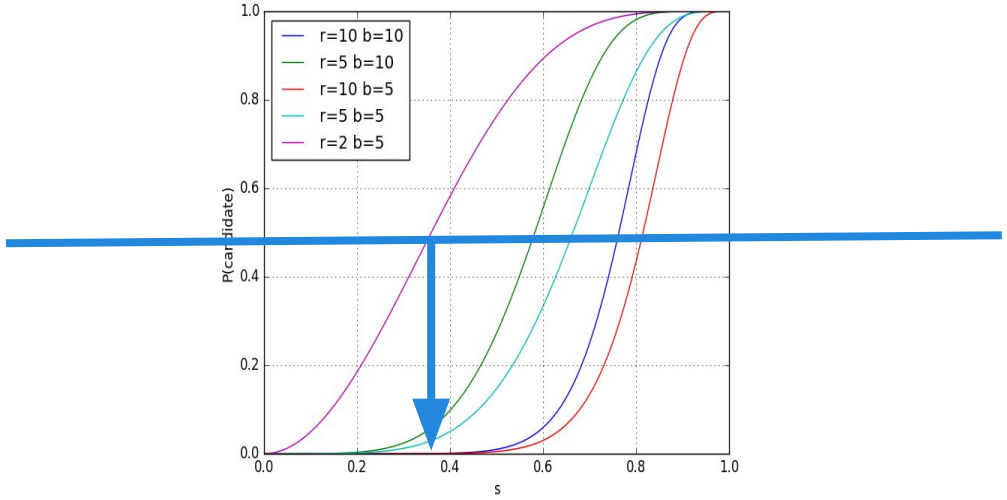
Analysis *

Pairs more likely to be found



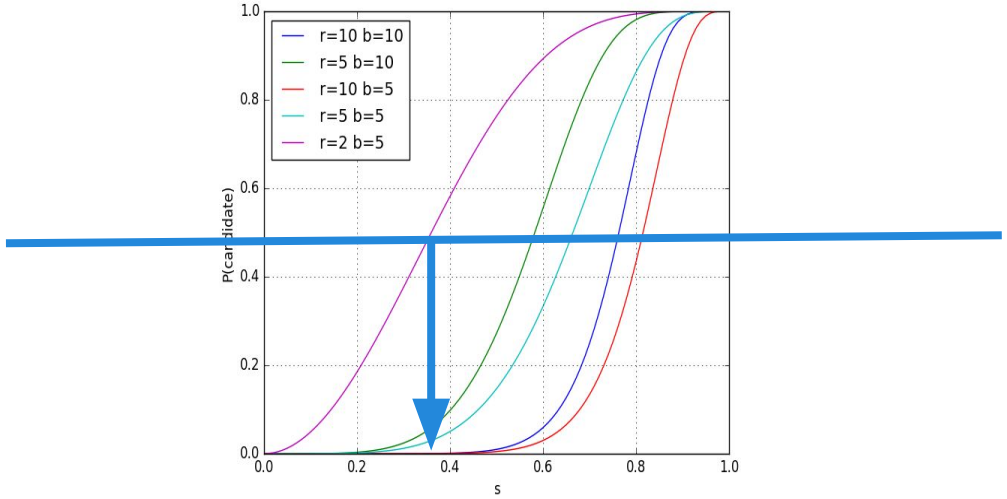
Pairs that most likely will not be find

Analysis: How to find t? *



(*blackboard*)

Analysis: How to find t? *

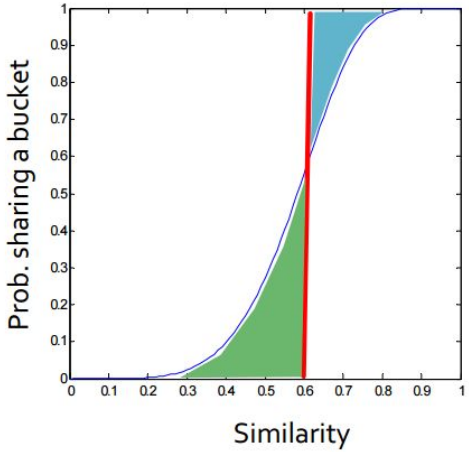


$$t \sim (1/b)^{1/r}$$

Fix t and b/r
-> get the rest
(blackboard example)

False positives, false negatives

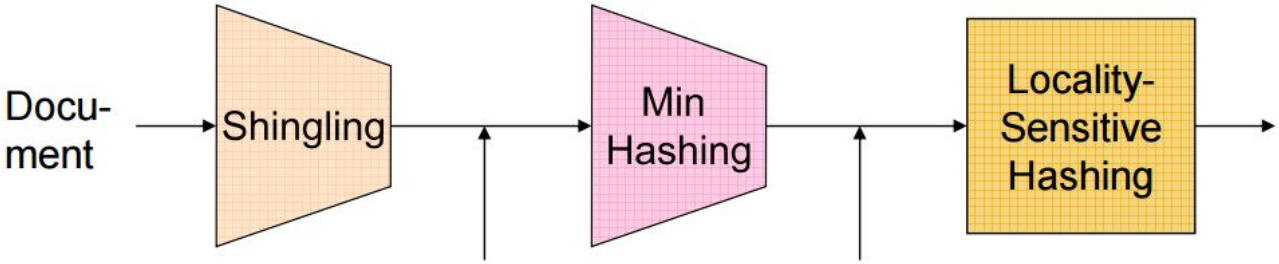
- **Picking *r* and *b* to get the best S-curve**
  - 50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Green area: False Positive rate

http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

Final pipeline



Candidate pairs: those pairs of signatures that we need to test for similarity

The set of strings of length ***k*** that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity

Assignment

Given the following documents:
D1 = be or not to be
D2 = to be two bees
D3 = not to bees

1) Represent them using bigrams.
2) Calculate Jaccard Similarity between them
3) Represent as bit-vectors <u>(vector positions should be ordered alphabetically for example: “be to” before “to be” as long as ‘b’<’t’, “be to” before “be too” as long as ‘’<’o etc.)</u>
4) Store as a matrix
5) calculate MinHash signatures for two permutations:
   a) Identity permutation ix2(ix) = ix
   b) ix2(ix) = (3*ix+1) mod 7
6) estimate Jaccard Similarity from MinHash signatures and compare to true values

Solution 1

Given the following documents:
D1 = be or not to be
D2 = to be two bees
D3 = not to bees

1) Represent them as using bigrams.
   S1 = {be_or, or_not, not_to, to_be}
   S2 = {to_be, be_two, two_bees}
   S3 = {not_to, to_bees}
2) Calculate Jaccard Similarity between them
   J(S1,S2) = 1 / 6
   J(S1,S3) = 1 / 5
   J(S2,S3) = 0

Solution 2

3) Represent as bit-vectors:

Elements ordered <u>alphabetically</u> along with vector positions:

be_or	be_two	not_to	or_not	to_be	to_bees	two_bees
0	1	2	3	4	5	6

Pos: 0 1 2 3 4 5 6

$S1 = (1\ 0\ 1\ 1\ 1\ 0\ 0)^T$

$S2 = (0\ 1\ 0\ 0\ 1\ 0\ 1)^T$

$S3 = (0\ 0\ 1\ 0\ 0\ 1\ 0)^T$

Solution 3

4) Store as matrix

Pos: 0 1 2 3 4 5 6

$S1 = (1\ 0\ 1\ 1\ 1\ 0\ 0)^T$

$S2 = (0\ 1\ 0\ 0\ 1\ 0\ 1)^T$

$S3 = (0\ 0\ 1\ 0\ 0\ 1\ 0)^T$

Ix	S1	S2	S3
0	1	0	0
1	0	1	0
2	1	0	1
3	1	0	0
4	1	1	0
5	0	0	1
6	0	1	0

Solution 4

5) calculate MinHash signatures for two permutations:

a) Identity permutation ix2(ix) = ix

Ix	S1	S2	S3	Ix2
0	1	0	0	0
1	0	1	0	1
2	1	0	1	2
3	1	0	0	3
4	1	1	0	4
5	0	0	1	5
6	0	1	0	6

	S1	S2	S3
h1	0	1	2

Solution 4

5) calculate MinHash signatures for two permutations:

b) ix2(ix) = (3*ix+1) mod 7

Ix	S1	S2	S3	Ix2
0	1	0	0	1
1	0	1	0	4
2	1	0	1	0
3	1	0	0	3
4	1	1	0	6
5	0	0	1	2
6	0	1	0	5

Ix	S1	S2	S3	Ix2
2	1	0	1	0
0	1	0	0	1
5	0	0	1	2
3	1	0	0	3
1	0	1	0	4
6	0	1	0	5
4	1	1	0	6

	S1	S2	S3
h2	0	4	0

Solution 5

	S1	S2	S3
h1	0	1	2

	S1	S2	S3
h2	0	4	0

	S1	S2	S3
h1	0	1	2
h2	0	4	0

6) estimate Jaccard Similarity from MinHash signatures and compare to true values:

J'(S1,S2) = 0 < J(S1,S2) = 1 / 6

J'(S1,S3) = ½ > J(S1,S3) = 1 / 5

J'(S2,S3) = 0 = J(S2,S3) = 0

LSH for MinHash

2017-03-20

Number of pairs problem

- We know how to compare 2 documents in a short (for H fixed = O(1)) time

But still:

- with 10M documents we have ~10M^2 pairs = few days of computation
- Solution: approximate NNS with LSH

LSH: Candidate pairs

- Get set of candidate pairs

  <u>Number of candidate pairs << number of all pairs</u>

- candidate pair = a pair of elements whose similarity must be evaluated

  (to check if they are really similar)

- For some **similarity threshold t** we want
  - Almost all pairs S1, S2 with s(S1,S2) >= t
  - Almost none pairs with s(S1,S2) < t

One document = one vector

1	9	5
4	0	7
2	0	4
6	3	1
7	1	7
1	4	1
2	6	2
3	7	3
4	9	2
6	1	5
1	2	5
9	4	0

b bands of r rows

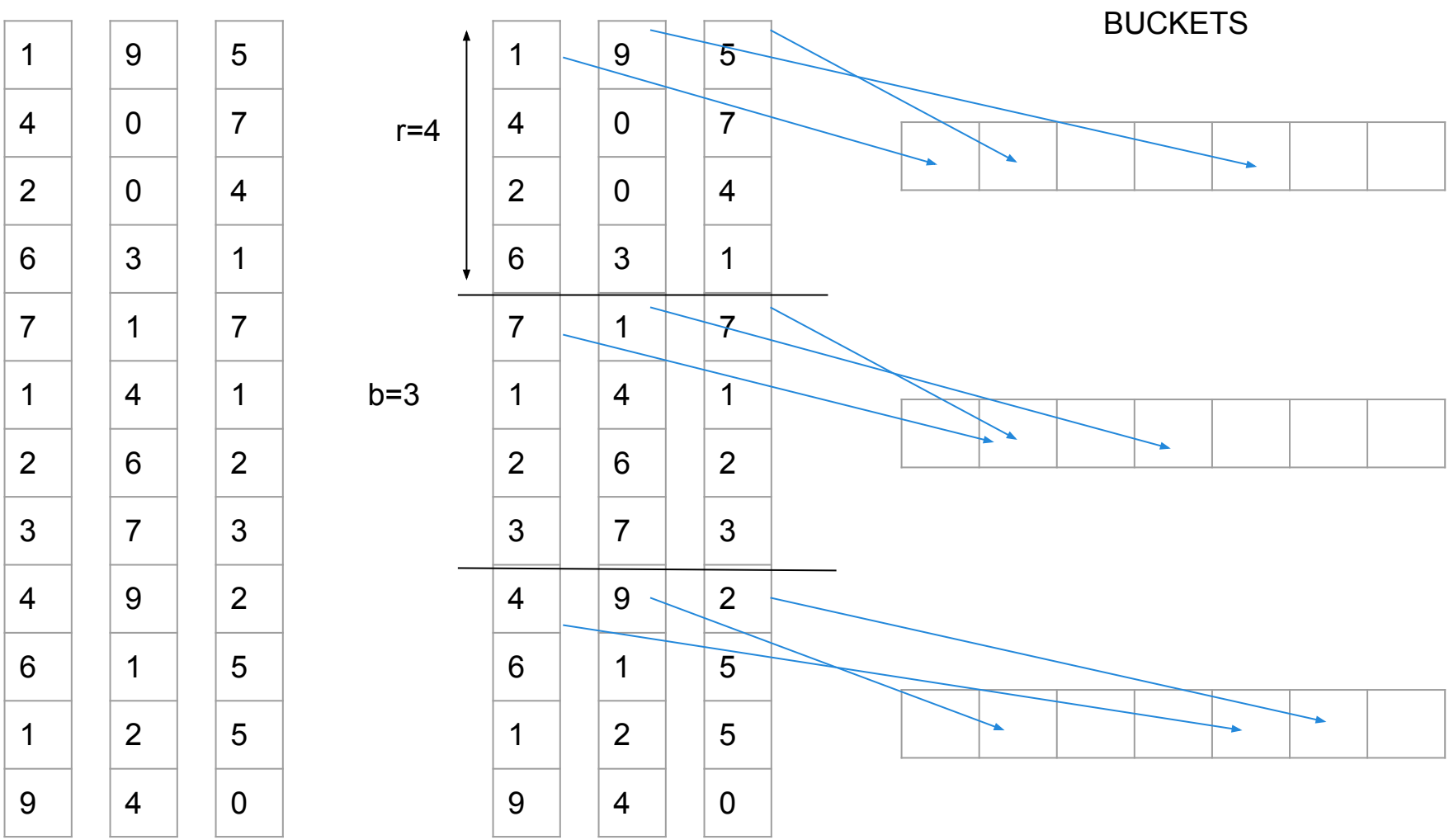
1	9	5
4	0	7
2	0	4
6	3	1
7	1	7
1	4	1
2	6	2
3	7	3
4	9	2
6	1	5
1	2	5
9	4	0

r=4

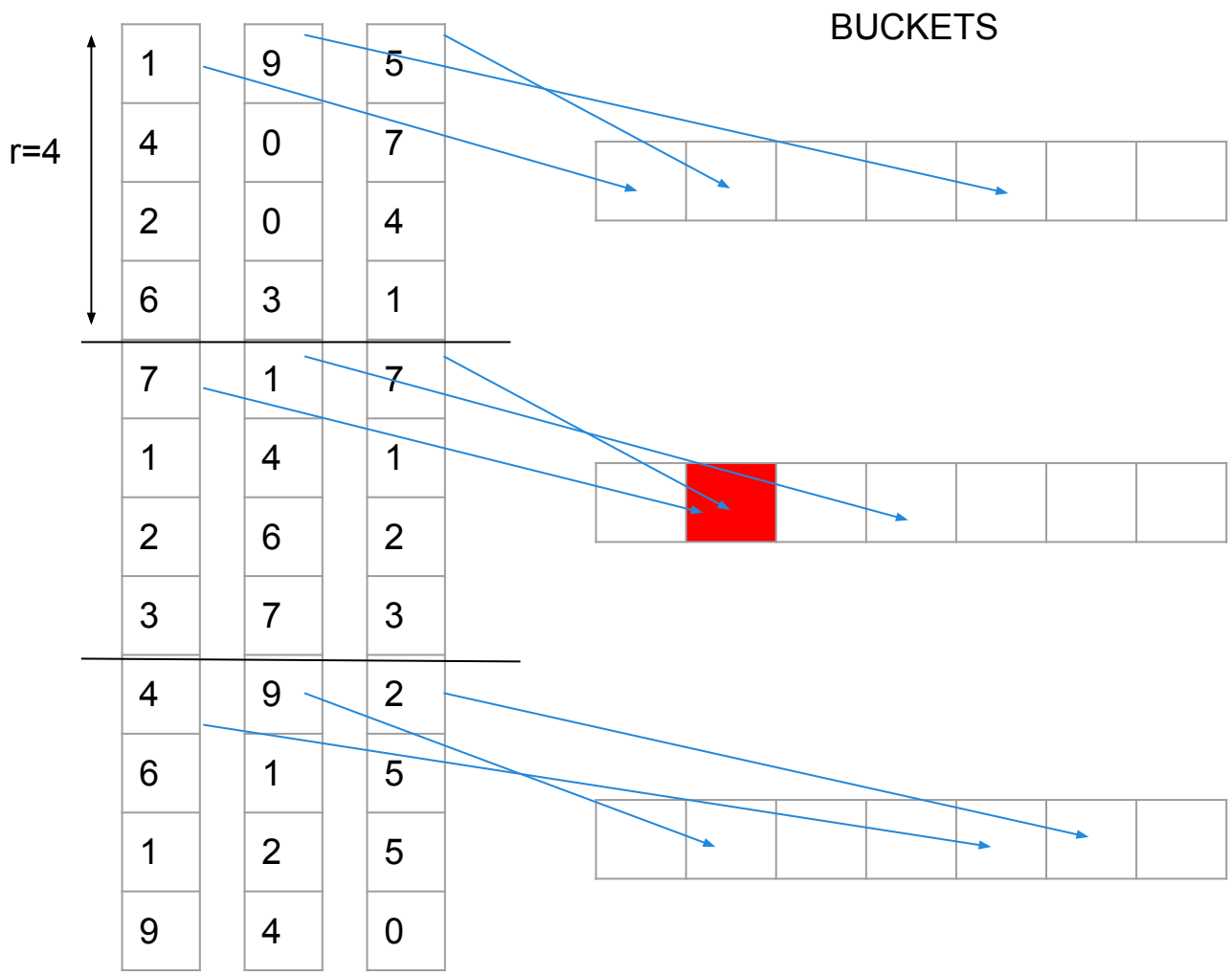
b=3

1	9	5
4	0	7
2	0	4
6	3	1
7	1	7
1	4	1
2	6	2
3	7	3
4	9	2
6	1	5
1	2	5
9	4	0

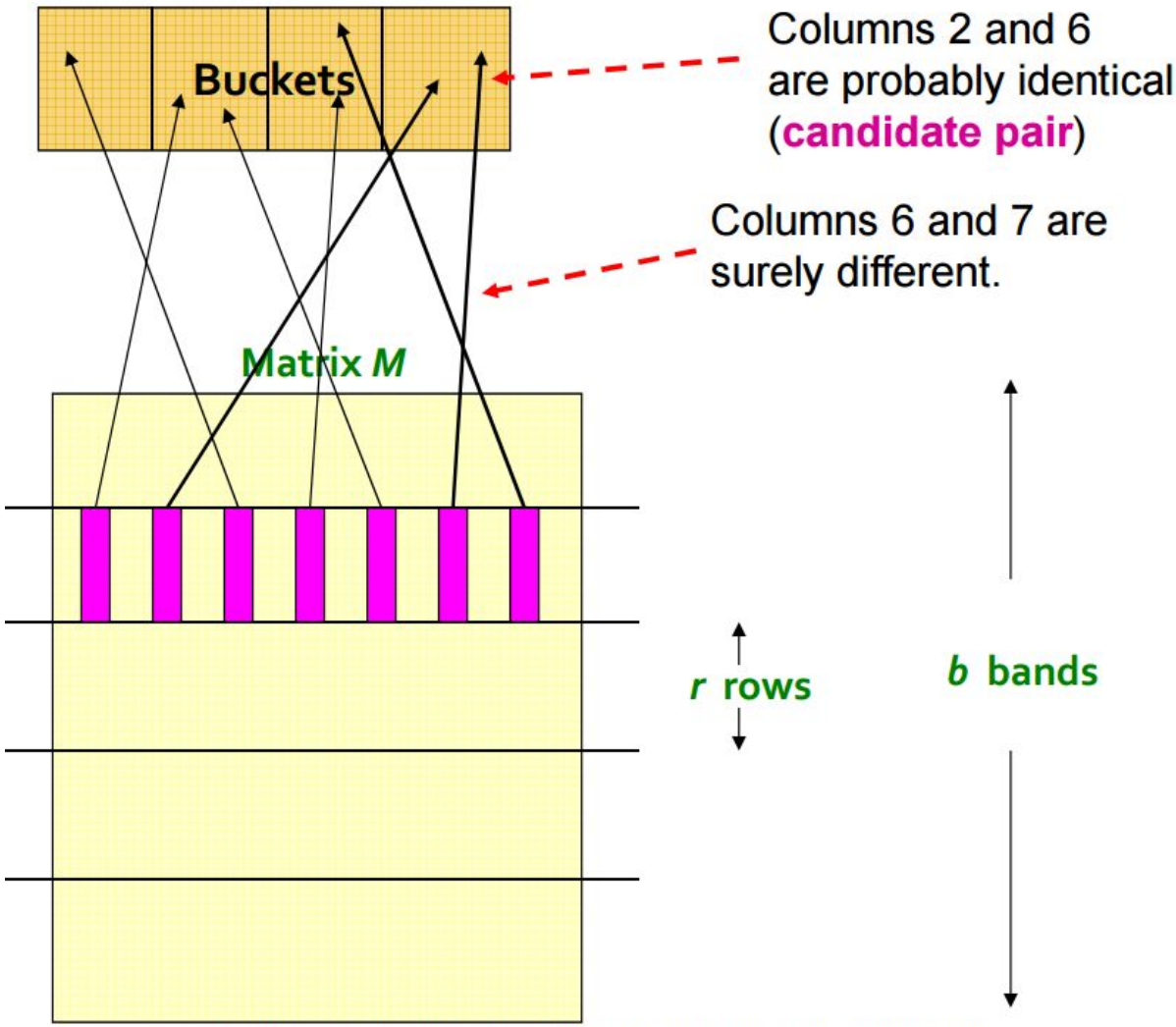
Hashing bands



Hashing bands

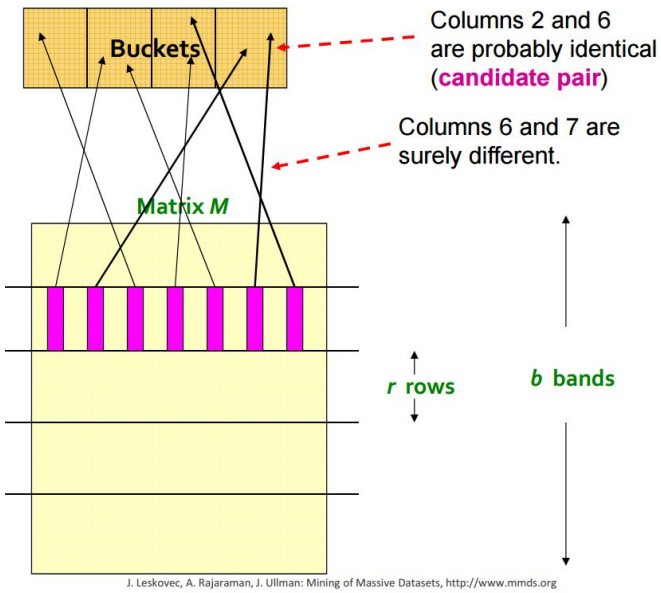


Another example



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Overview & algorithm



- Split M into b bands with r rows each
- For each doc (column):
  - For each band:
    - Hash band signature part into one of buckets
- Go over all buckets for all bands:
  - If there are two docs in one bucket
    -> candidate pair

Mining Streaming Data

Goal

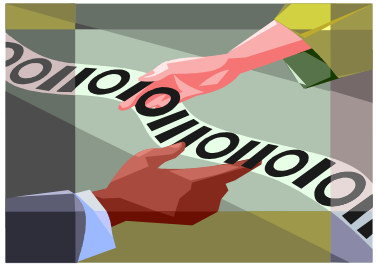
To understand the **importance** of mining methods for streaming data, and learn about existing **methods handling** streaming data

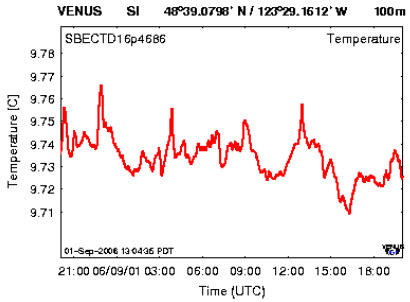
Characteristics/Description

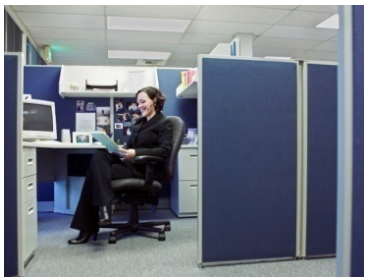
- Stream data sets are…
  - Continuous
  - Massive
  - Unbounded
  - Possibly infinite
- Fast changing and requires fast, real-time response
  - Examples:

    Life threatening: collision avoidance

    Lost revenue/transactions: hung-up networks







Sizing the challenge

- WalMart Records >20 Million Transactions
- Google Handles >100 Million Searches
- AT&T produces >275 million call records
- Earth sensing satellite produces GBs of data
- Twitter handles millions of tweets in a second

This just in a day!

The Stream Model

- Input tuples enter at a rapid rate, at one or more input ports.
- Impractical to store the whole streaming data
- How do you make critical calculations about the stream using a limited amount of (primary or secondary) memory?
- Simple calculation per data due to time and space constraints

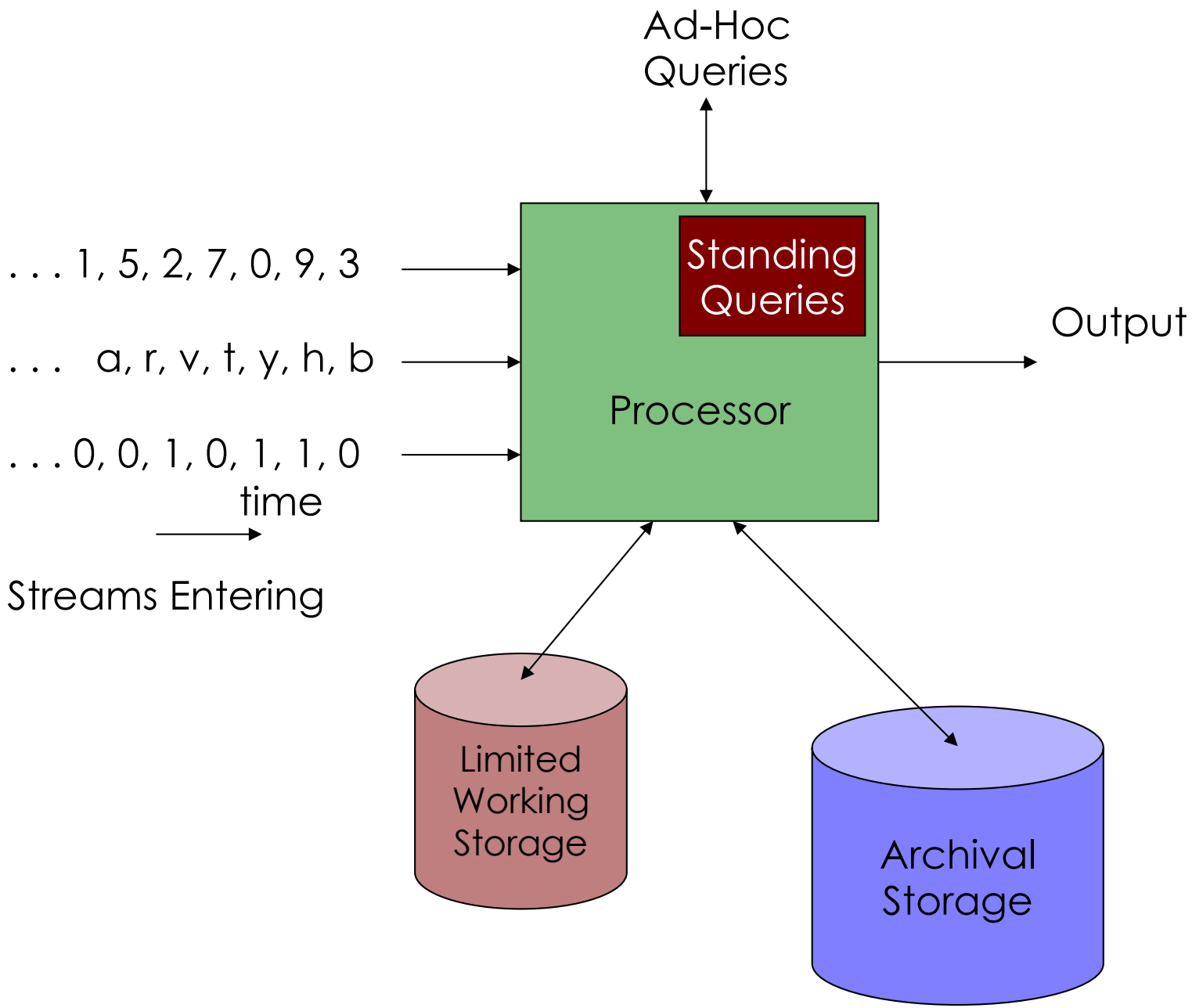
Time/Space constrained

- Not enough memory
- Can't afford storing/revisiting the data
  - Single pass computation
- External memory algorithms for handling data sets larger than main memory cannot be used.
  - Do not support continuous queries
  - Too slow real-time response

Two Forms of Query

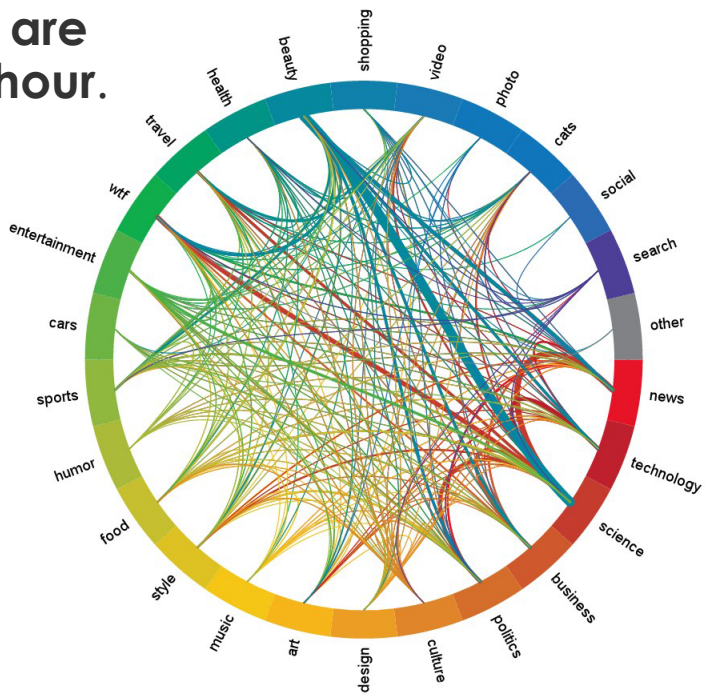
1. ***Ad-hoc queries***: Normal queries asked one time about streams.
   - Example: What is the maximum value seen so far in stream *S*?

2. ***Standing queries***: Queries that are, in principle, asked about the stream at all times.
   - Example: Report *each new* maximum value ever seen in stream *S*.


Ad-Hoc
Queries
. . . 1, 5, 2, 7, 0, 9, 3
. . . a, r, v, t, y, h, b
. . . 0, 0, 1, 0, 1, 1, 0
time
Streams Entering
Standing
Queries
Processor
Output
Limited
Working
Storage
Archival
Storage

Example Applications

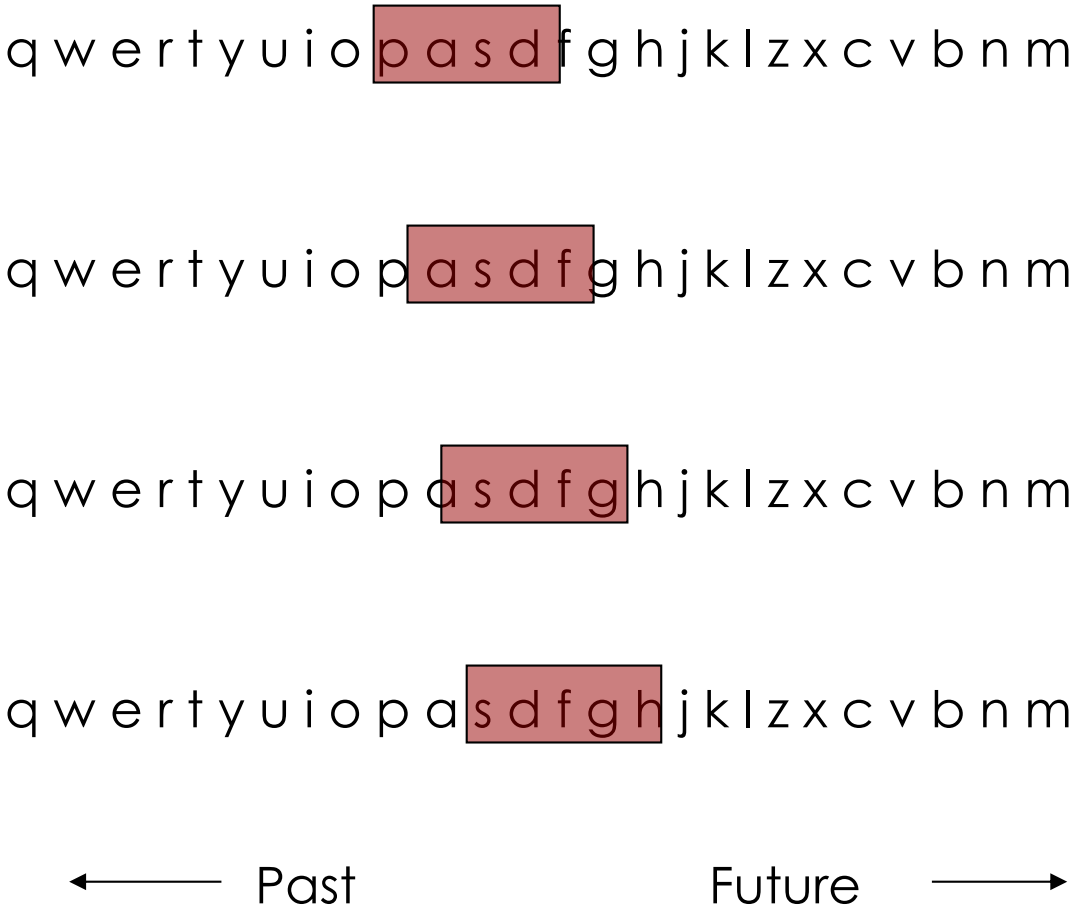
- Mining query streams.
  - E.g.: Google wants to know what queries are **more frequent today than yesterday**.
- Mining click streams.
  - E.g.: Yahoo wants to know **which of its pages are getting an unusual number of hits in the past hour**.
    - Often caused by annoyed users clicking on a broken page.
- IP packets can be monitored at a switch.
  - E.g.:
    - Gather information for optimal routing.
    - Detect denial-of-service attacks.



Sliding Windows

- A useful model of stream processing
  - queries are about a **window** of length *N* – the *N* most recent elements received.
  - **Alternative**: elements received within a time interval *T*.
- **Interesting case**:
  - ***N too large*** to be stored in main memory
  - Or, ***too many streams*** that windows for all do not fit in main memory.

Sliding Windows



Example: Averages

- Stream of integers, window of size *N*.
- **Standing query**: what is the average of the integers in the window?
- For the first *N* inputs, sum and count to get the average.

Question:

- What do we do now for the next input *i*?

Example: Averages

Answer:

- When a new input *i* arrives:
  - change the average by adding *(i - j)/N*, where *j* is the oldest integer in the window before *i* arrived.

- **Good**: O(1) time per input.
- **Bad**: Requires the entire window in main memory.

Counting 1's

Approximating Counts
Exponentially Growing Blocks
DGIM Algorithm

Approximate Counting

- Can show that an exact sum or count of the elements in a window:
  - Cannot use less space than the window itself.
- If willing to accept an ***approximation***:
  - Can use much less space.
- Consider the simple case of counting bits
- Sums are a fairly straightforward extension.

Counting Bits

- **Problem:** given a stream of 0's and 1's, be prepared to answer queries of the form "how many 1's in the most recent k bits?" where $k \leq N$.
- **Obvious solution**: store the **most recent *N* bits**
- But answering the query will take $O(k)$ time
  - Very possibly too much time.
- And the space requirements can be too great.
  - Especially if many streams to be managed in main memory at once, or N is huge.

Example: Bit Counting

- Count recent hits on URL's belonging to a site.
- Stream is a sequence of URL's.
- Window size N = 1 billion.
- Think of the data as many streams – one for each URL.
- Bit on the stream for URL x is 0 unless the actual stream has x.

DGIM Method

- Name refers to the inventors:
  - **D**atar, **G**ionis, **I**ndyk, and **M**otwani.
- Store only $O(\log_2 N)$ bits per stream.
  - N = window size.
- Gives approximate answer, never off by more than 50%.
  - Error factor can be reduced to any $\varepsilon > 0$, with more complicated algorithm and proportionally more stored bits.

Timestamps

- Each bit in the stream has a *timestamp*, starting 0, 1, …
- Record timestamps modulo N (the window size), so we can represent any *relevant* timestamp in $O(\log_2 N)$ bits.

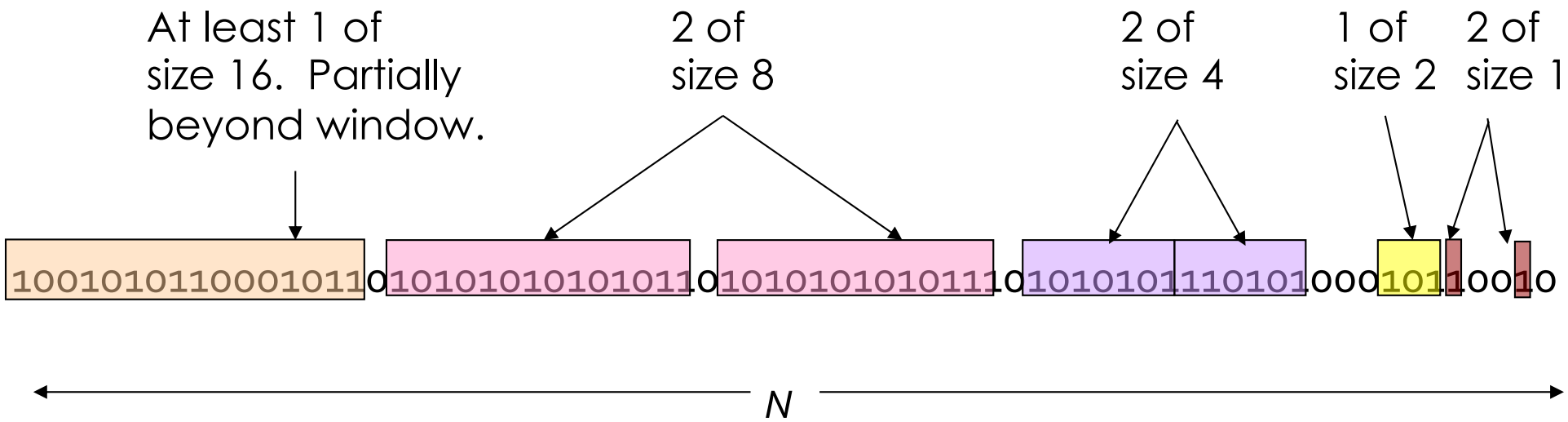
Buckets

- A segment of the window, represented by a record consisting of:
  - The **timestamp** of its end [O(log N) bits].
  - The **number of 1's between its beginning and end**.
    - Number of 1's = size of the bucket.
- Constraint on bucket sizes: number of **1's must be a power of 2**.
  - Thus, only *O(log log N)* bits are required for this count.

Representing a Stream by Buckets

- Either one or two buckets with the same power-of-2 number of 1's.
- Buckets do not overlap.
- Buckets sorted by size.
  - Older buckets not smaller than newer buckets.
- Buckets disappear when their end-time is > N time units in the past.

Example: Bucketized Stream



Updating Buckets

- New bit in, drop the last (oldest) bucket if its end-time is prior to *N* time units before the current time.
- If the current bit is 0, no other changes are needed.

Updating Buckets: Input = 1

- If the current bit is 1:
  - Create a new bucket of size 1, for just this bit.
    - End timestamp = current time.
  - If there are now three buckets of size 1, combine the oldest two into a bucket of size 2.
  - If there are now three buckets of size 2, combine the oldest two into a bucket of size 4.
  - And so on …

Example: Managing Buckets

Initial

1001010110001011010101010101011010101010101110101010111010100001011100 10

1 arrives; makes third block of size 1.

0010101100010110101010101010110101010101011101010111010100001011001 01

Combine oldest two 1's into a 2.

0010101100010110101010101010110101010101011101010111010100001011001 01

Later, 1, 0, 1 arrive. Now we have 3 1's again.

0101100010110101010101010110101010101011101010111010100001011001011 01

Combine two 1's into a 2.

0101100010110101010101010110101010101011101010111010100001011001011 01

The effect ripples all the way to a 16.

0101100010110101010101010110101010101011101010111010100001011001011 01

Querying

- To estimate the number of 1's in the most recent $k < N$ bits:
  - Restrict your attention to only those buckets whose end time stamp is at most k bits in the past.
  - Sum the sizes of all these buckets but the oldest.
  - Add half the size of the oldest bucket.
- Remember:
  - we don't know how many 1's of the last bucket are still within the window.

Error Bound

- Suppose the oldest bucket within range has size $2i$.
- Then by assuming $2i - 1$ of its 1's are still within the window, we make an error of at most $2i - 1$.
- Since there is at least one bucket of each of the sizes less than 2i, and at least 1 from the oldest bucket, the true sum is no less than 2i.
- Thus, error at most 50%.

Space Requirements

- Can represent one bucket in *O(log N)* bits.
  - It's just a timestamp needing log N bits and a size, needing log log N bits.
- No bucket can be of size greater than N.
- There are at most two buckets of each size 1, 2, 4, 8,…
- That's at most *log N* different sizes, and at most 2 of each size, so at most *2log N* buckets.

- Bloom Filters
- Sampling Streams
- Counting Distinct Items

Filtering Stream Content

- To motivate the **Bloom-filter** idea, consider a web crawler.
- It keeps, centrally, a list of all the URL's it has found so far.
- It assigns these URL's to any of a number of parallel tasks
  - these tasks stream back the URL's they find in the links they discover on a page.
- It needs to filter out those URL's it has seen before.

Role of the Bloom Filter

- A Bloom filter placed on the stream of URL's will declare that **certain URL's have been seen before**.
- Others will be declared new, and will be added to the list of URL's that need to be crawled.
- Unfortunately, the Bloom filter can have false positives.
  - It can declare a **URL seen before when it hasn't**.
- But if it says "never seen," then it is truly new.
- **Solution: restart the filter periodically**.

Example: Filtering Chunks

- Suppose we have a database relation stored in a DFS, spread over many chunks.
- We want to find a particular value *v*, looking at as few chunks as possible.
- A Bloom filter on each chunk will tell us certain values are there, and others aren't.
  - As before, **false positives possible**
- But now things are exactly right: if the filter says *v* is not at the chunk, it surely isn't.
  - Occasionally, we retrieve a chunk we don't need, but can't miss an occurrence of value *v*.

How a Bloom Filter Works

- A *Bloom filter* is an array of bits, together with a number of hash functions.
- The argument of each hash function is a stream element, and it returns a position in the array.
- Initially, all bits are 0.
- When input x arrives, we **set to 1 the bits *h(x)*, for each hash function *h***.

Example: Bloom Filter

- Use $N = 11$ bits for our filter.
- Stream elements = integers.
- Use two hash functions:
  - *h1(x)* =
    - Take **odd-numbered bits from the right** in the binary representation of x.
    - Treat it as an integer i.
    - Result is i modulo 11.
  - *h2(x)* = same, but **take even-numbered bits**.

Example – Continued

Stream element	h_1	h_2	Filter contents
			00000000000
25 = 11001	5	2	00100100000
159 = 10011111	7	0	10100101000
585 = 1001001001	9	7	10100101010

Note: bit 7 was already 1.

Bloom Filter Lookup

- Suppose element *y* appears in the stream, and we want to know if we have seen y before.
- Compute *h(y)* for each hash function *y*.
- If all the resulting bit positions are 1, say we have seen *y* before.
  - We could be wrong.
    - Different inputs could have set each of these bits.
- If at least one of these positions is 0, say we have not seen *y* before.
  - We are certainly right.

Example: Lookup

- Suppose we have the same Bloom filter as before, and we have set the filter to 10100101010.
- Lookup element y = 118 = 1110110 (binary).
- $h_1(y)$ = 14 modulo 11 = 3.
- $h_2(y)$ = 5 modulo 11 = 5.
- Bit 5 is 1, but bit 3 is 0, so we are sure y is not in the set.

Performance of Bloom Filters

- Probability of a false positive depends on the density of 1's in the array and the number of hash functions.
  - = (fraction of 1's)$^{\text{\# of hash functions}}$.
- The number of 1's is approximately the number of elements inserted times the number of hash functions.
  - But collisions lower that number slightly.

Sampling a Stream

When Sampling Doesn't Work

- Suppose Google would like to examine its stream of **search queries** for the past month to find out what fraction of them were **unique** – asked only once.
- But to save time, we are only going to sample 1/10th of the stream.
- The fraction of unique queries in the sample != the fraction for the stream as a whole.
  - In fact, we can't even adjust the sample's fraction to give the correct answer.

Example: Unique Search Queries

- The length of the sample is 10% of the length of the whole stream.
- Suppose a query is unique.
  - It has a 10% chance of being in the sample.
- Suppose a query occurs exactly twice in the stream.
  - It has an 18% chance of appearing exactly once in the sample.
- And so on … The fraction of unique queries in the stream is unpredictably large.

Sampling by Value

- My mistake: I sampled based on the position in the stream, rather than the value of the stream element.
- The right way: hash search queries to 10 buckets 0, 1,…, 9.
- Sample = all search queries that **hash to bucket 0**.
  - All or none of the instances of a query are selected.
  - Therefore the fraction of unique queries in the sample is the same as for the stream as a whole.

Controlling the Sample Size

- Problem: What if the total sample size is limited?
- Solution: Hash to a large number of buckets.
- Adjust the set of buckets accepted for the sample, so your sample size stays within bounds.

Example: Fixed Sample Size

- Suppose we start our search-query sample at 10%, but we want to limit the size.
- Hash to (say) 100 buckets, 0, 1,…, 99.
  - Take for the sample those elements hashing to buckets 0 through 9.
- If the sample gets too big, get rid of bucket 9.
- Still too big, get rid of 8, and so on.

Sampling Key-Value Pairs

- This technique generalizes to any form of data that we can see as tuples (K, V), where K is the "key" and V is a "value."
- Distinction: We want our sample to be based on picking some set of keys only, not pairs.
  - In the search-query example, the data was "all key."
- Hash keys to some number of buckets.
- Sample consists of all key-value pairs with a key that goes into one of the selected buckets.

Example: Salary Ranges

- Data = tuples of the form (EmpID, Dept, Salary).
- Query: What is the average range of salaries within departments?
- Key = Dept.
- Value = (EmpID, Salary).
- Sample picks some departments, has salaries for all employees of that department, including its min and max salaries.
- Result will be an unbiased estimate of the average salary range.

Counting Distinct Elements

- Problem: a data stream consists of elements chosen from a set of size *n*. Maintain a count of the number of distinct elements seen so far.
- Obvious approach: maintain the set of elements seen.

Application Examples

- How many **different words** are found among the Web pages being crawled at a site?
  - Unusually low or high numbers could indicate artificial pages (spam?).
- How many **unique users** visited Facebook this month?
- How many **different pages** link to each of the pages we have crawled.
  - Useful for estimating the PageRank of these pages.
    - Which in turn can tell a crawler which pages are most worth visiting.

Estimating Counts

- Real Problem: what if we do not have space to store the complete set?
  - Or we are trying to count lots of sets at the same time.
- Estimate the count in an unbiased way.
- Accept that the count may be in error, but limit the probability that the error is large.

Flajolet-Martin Approach

- A probabilistic counting algorithm
- Used to estimate number of **distinct** elements in a large file originally
- Use little memory
- Single pass only
- Based on statistical observation made on bits of hashed values

Flajolet-Martin Approach (2)

Estimating the cardinality of a multiset *M*:

for *i* :=0 **to** *N* - 1 **do** *BITMAP[i] :=0;*

for all x in *M* do

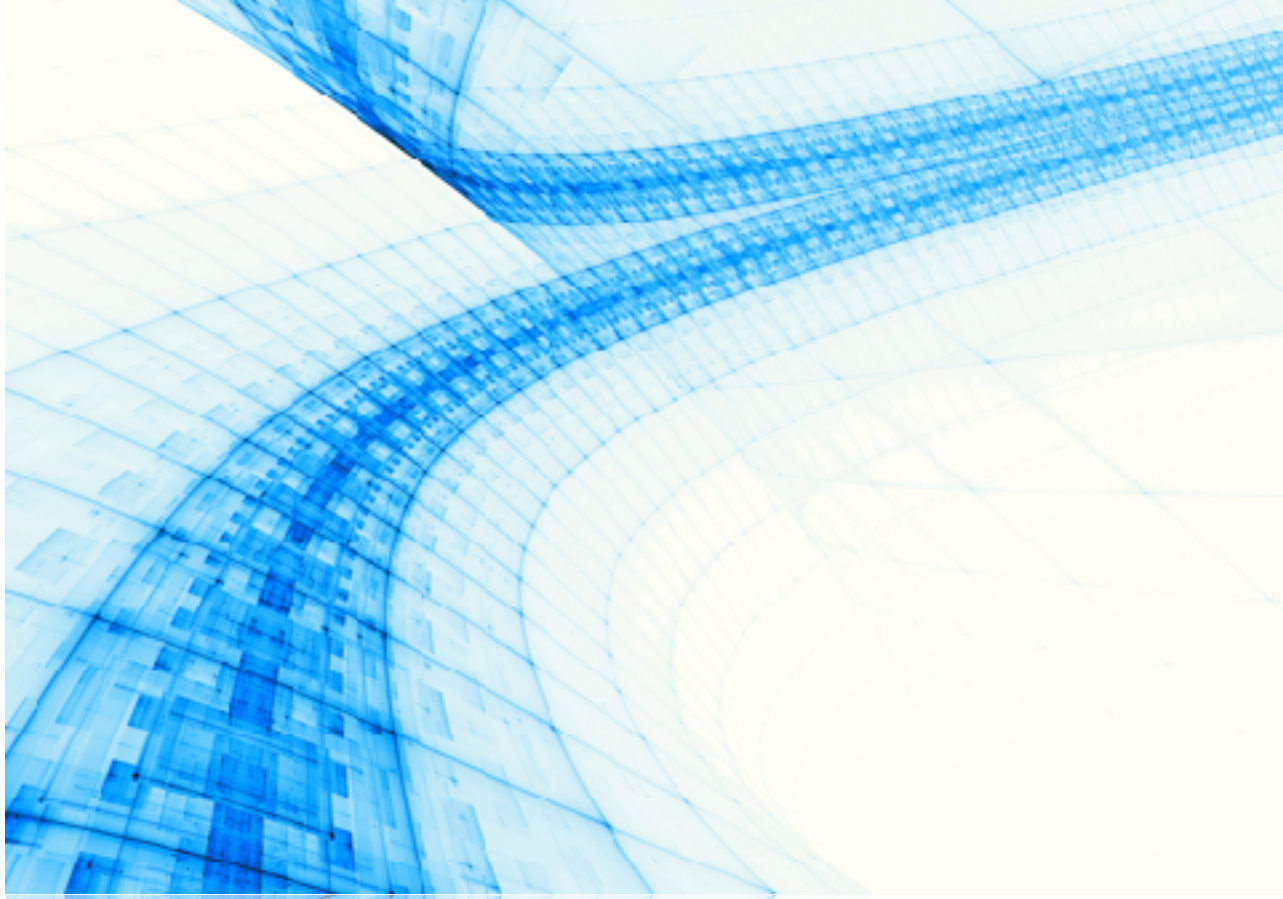
```
begin
    index := p(hash(x));
    if BITMAP[index] = 0 then BITMAP[index] := 1;
end;
```

R := the largest *index* in *BITMAP* whose value equals to 1

***Estimate* := 2^R**

Flajolet-Martin Approach (3)

- If the final BITMAP looks like this: 0000,0000,1100,1111,1111,1111
- The left most 1 at position 15
- Thus, have around 2^{15} distinct elements in the stream



Systems for data stream

AsterixDB and Storm

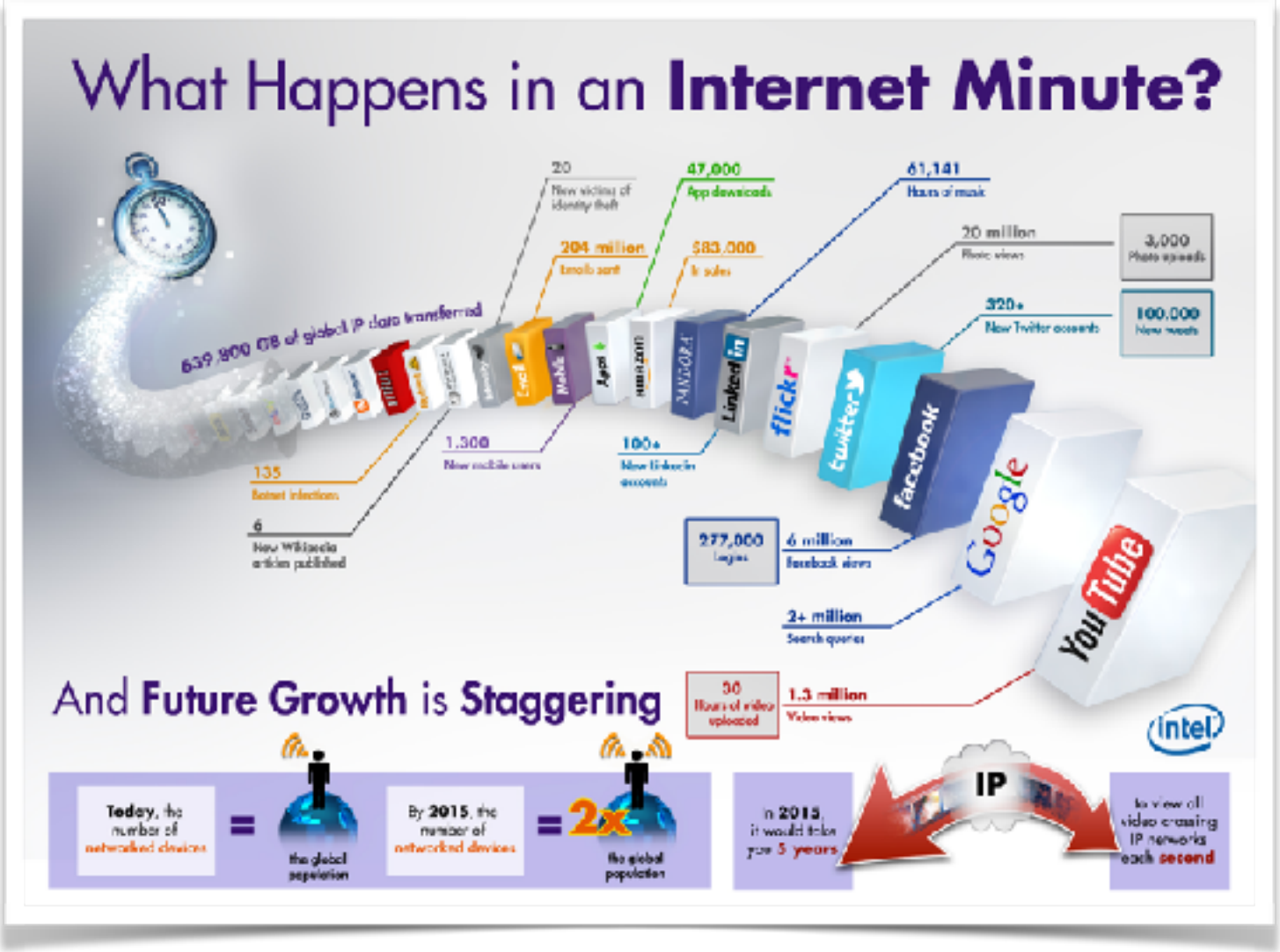
Aims

- To know how streaming data are handled in real-world systems: AsterixDB and Storm

Content

- Motivation
- Introduction to Storm
- Comparison between Storm and Spark
- Introduction to AsterixDB & Feeds

Streaming Data generation



Dealing with Streaming Data

- Continuous processing, aggregation and analyzing data **when created**
- System made like **directed acyclic graph**
- Data reduced before being centralized
- Processing messages one at time
- Collect structured, semi-structured and unstructured data from different sources

Streaming Data

- Actions in real time:
  - Monitoring trends
  - Communicate
  - Recommendation
  - Searching
- Expected response from 500 ms to one minute
- Data message flow through a chain of processors until the result reaches the final destination.
- To guarantee reliable data processing, necessary to restart processing in case of failures

Streaming Data (2)

- Delivery Semantics (Message Guarantees):
  - **At most once:**
    messages may be lost but never redelivered.
  - **At least once:**
    messages will never be lost but may be redelivered.
  - **Exactly once:**
    messages are never lost and never redelivered, perfect message delivery.

Fault-Tolerance

- **Soft Failure:** Inability to pre-process or ingest a received record either due to unexpected (null) value, duplicate record or due to inherent bug(s) in user provided code (UDF).
- **Hard Failure:** Inability to support dataflow due to loss of one of more participant nodes

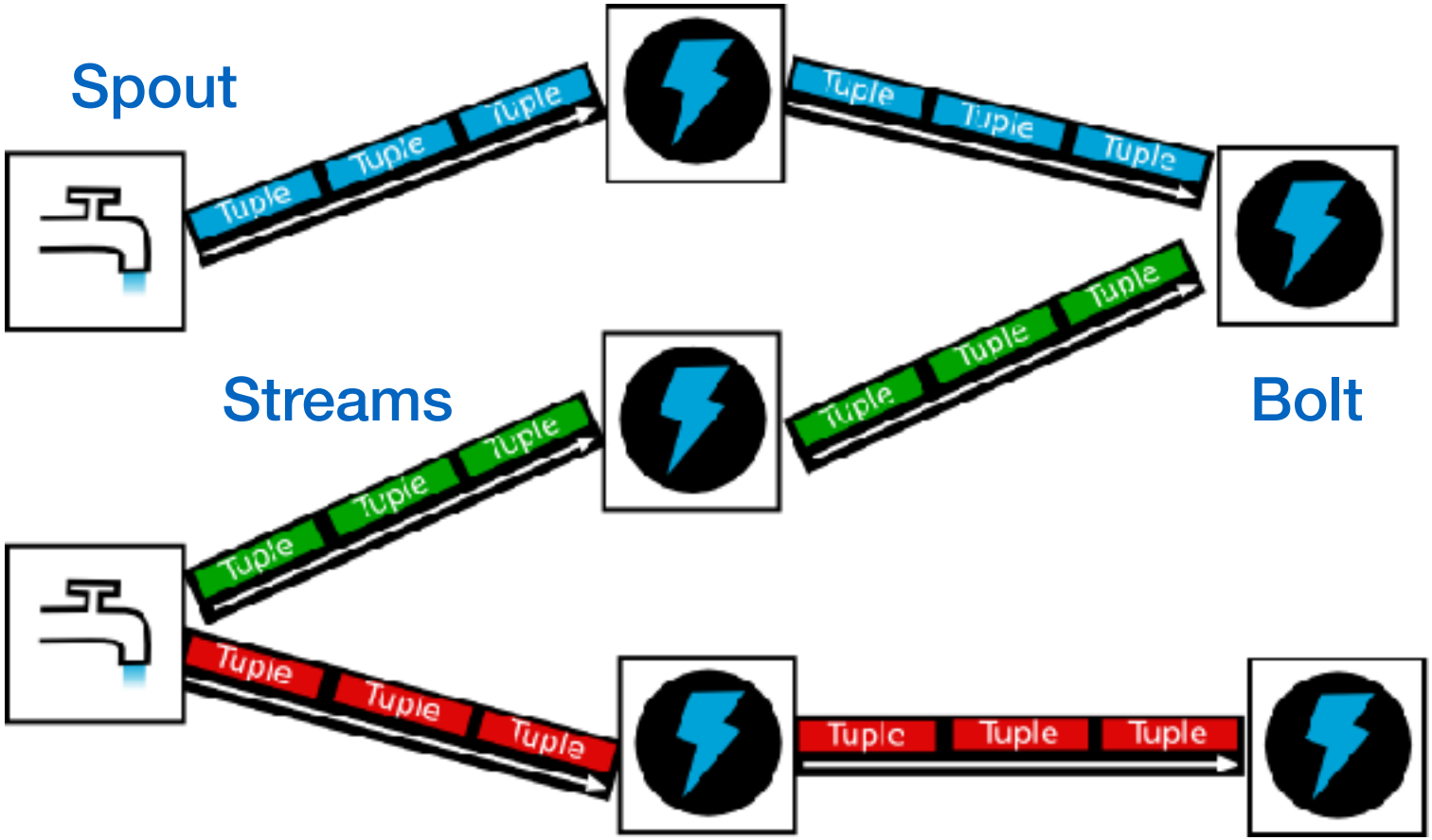
STORM

An open source, scalable, fault-tolerant, **distributed real-time computation system**.

Storm Concept

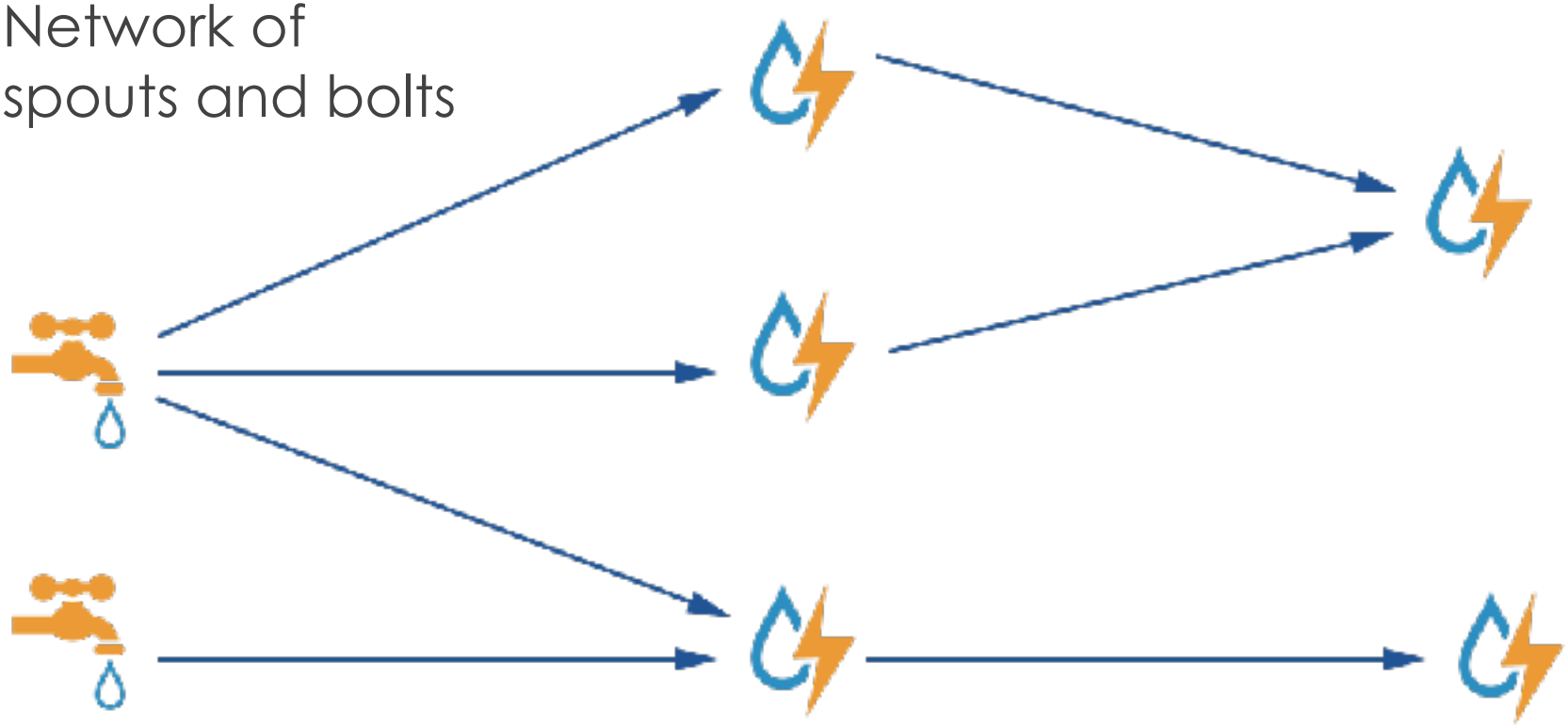
- **Streams**
  - Unbounded sequence of tuples
- **Spout**
  - Source of Stream
  - E.g. Read from Twitter streaming API
- **Bolts**
  - Processes input streams and produces new streams
  - E.g. Functions, Filters, Aggregation, Joins

Storm Concept



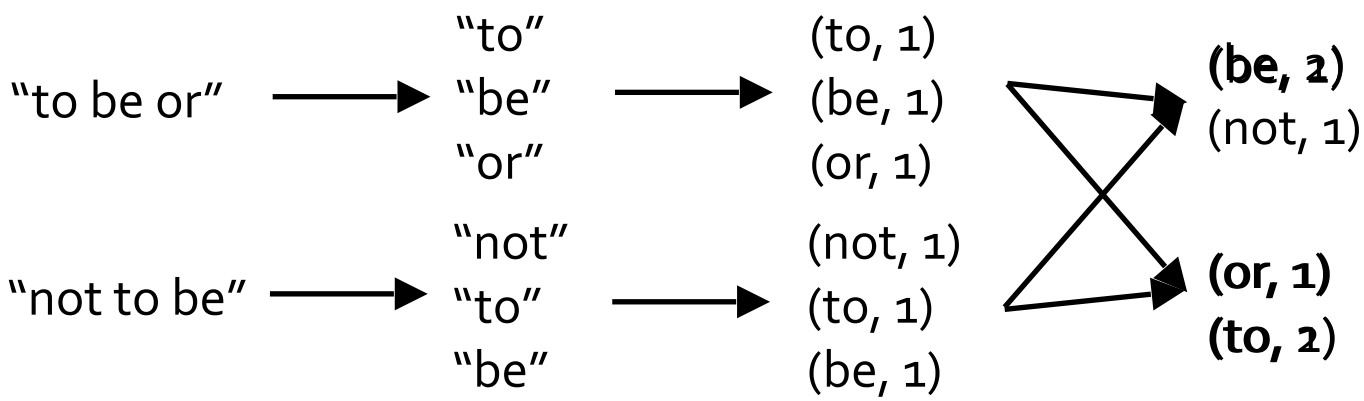
Storm Concept

- **Topologies**
  - Network of spouts and bolts

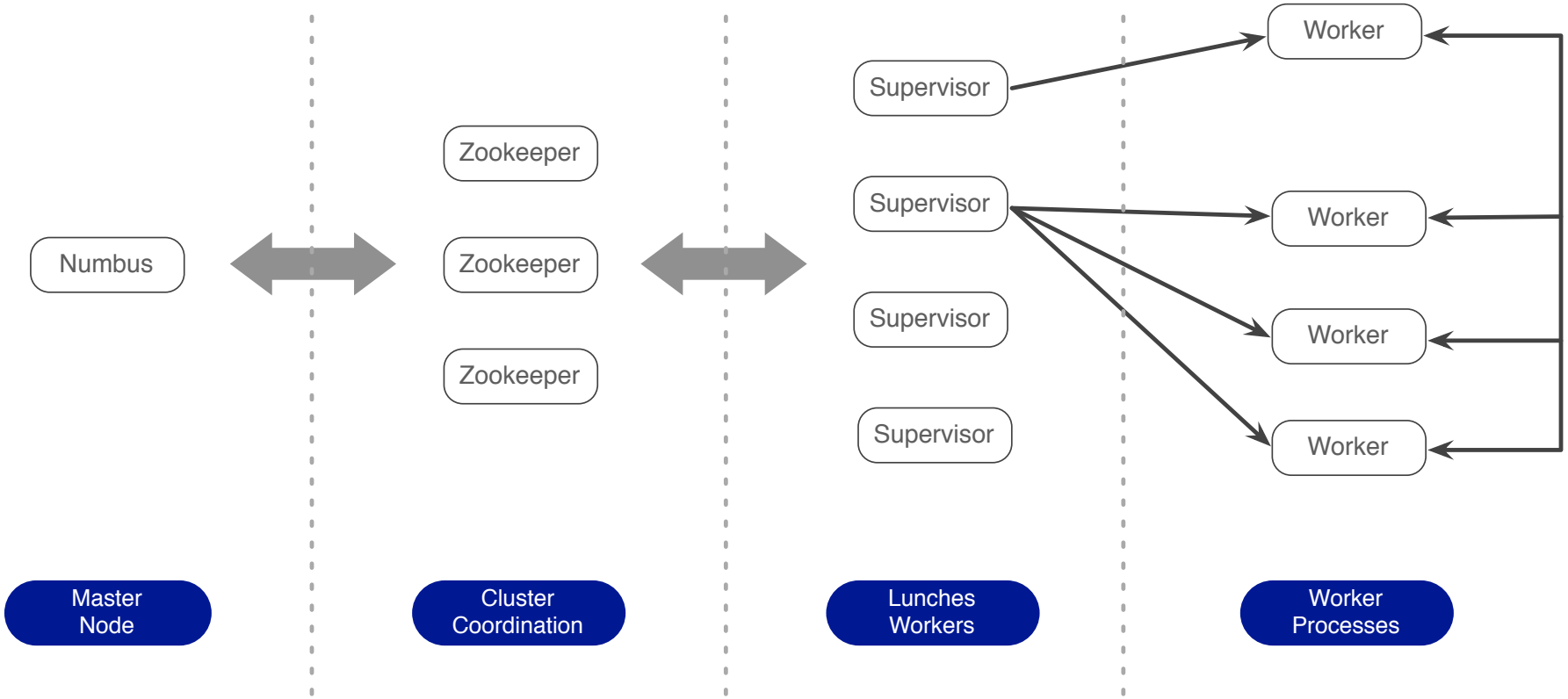


Word counting w/ Storm

```
TridentTopology topology = new TridentTopology();
TridentState wordCounts = topology.newStream("spout1", spout)
        .each(new Fields("sentence"), new Split(), new Fields("word"))
        .groupBy(new Fields("word"))
        .persistentAggregate(new MemoryMapState.Factory(), new Count(),
                new Fields("count")).parallelismHint(6);
```



Storm Architecture



Storm vs. Spark (2)

- Scale
- Latency
- Iterative Processing
  - Are there suitable non-iterative alternatives?
- Use What You Know
- Code Reuse
- Maturity

Storm Advantages

- **Fault tolerance:**
  - if worker threads die, or a node goes down, the workers are automatically restarted;
- **Scalability:**
  - throughput rates of even one million 100 byte messages per second per node can be achieved
- **Ease of use** in deploying and operating the system

STORM VS. Spark

	Storm	Spark
Processing Model	Event-Streaming	Micro-Batching / Batch (Spark Core)
Delivery Guarantees	At most once / At least once	Exactly Once
Latency	Sub-second	Seconds
Language Options	Java, Clojure, Scala, Python, Ruby	Java, Scala, Python
Development	Use other tool for batch	Batching and streaming are very similar

Spark Recommended

- Iterative Batch Processing (most Machine Learning)
  - Restricted alternatives
  - Some scale issues
- ETL - Extract, Transform and Load
- Shark/Interactive Queries

Spark Recommended (2)

- < 1 TB (or memory size of your cluster)
- Tuning to run well can be a pain
- Data Bricks and others are working on scaling.
- Streaming all μ-batch:
  - latency at least 1 sec
- Still single points of failure in streaming
- All streaming inputs replicated in memory

Storm recommended

- Latency < 1 second (single event at a time)
  - There is little else (especially not open source)
- "Real Time" …
  - Analytics
  - Budgeting
  - ML
  - Anything
- Lower Level API than Spark
- No built-in concept of look back aggregations
- Takes more effort to combine batch with streaming

STORM VS. Spark

Storm:

- Tends to be driven by creating classes and implementing interfaces
- Advantage of broader language support (code written in R or any other language not natively supported by Spark)
- DAG's as natural processing model
  Tuple natural interface for the data passed between nodes
- Processing excels at computing transformations as data are ingested with sub-second latencies

STORM VS. Spark

Spark:

- Has more of a "functional" flavor, where working with the API is driven more by chaining successive method calls to invoke primitive operations
- Tuples can feel awkward in Java but with this is going benefit of compile-time
- Use existing Hadoop or Mesos cluster
- Micro-batching trivially gives stateful computation, making windowing an easy task.

Conclusion: Neither approach is better or worse

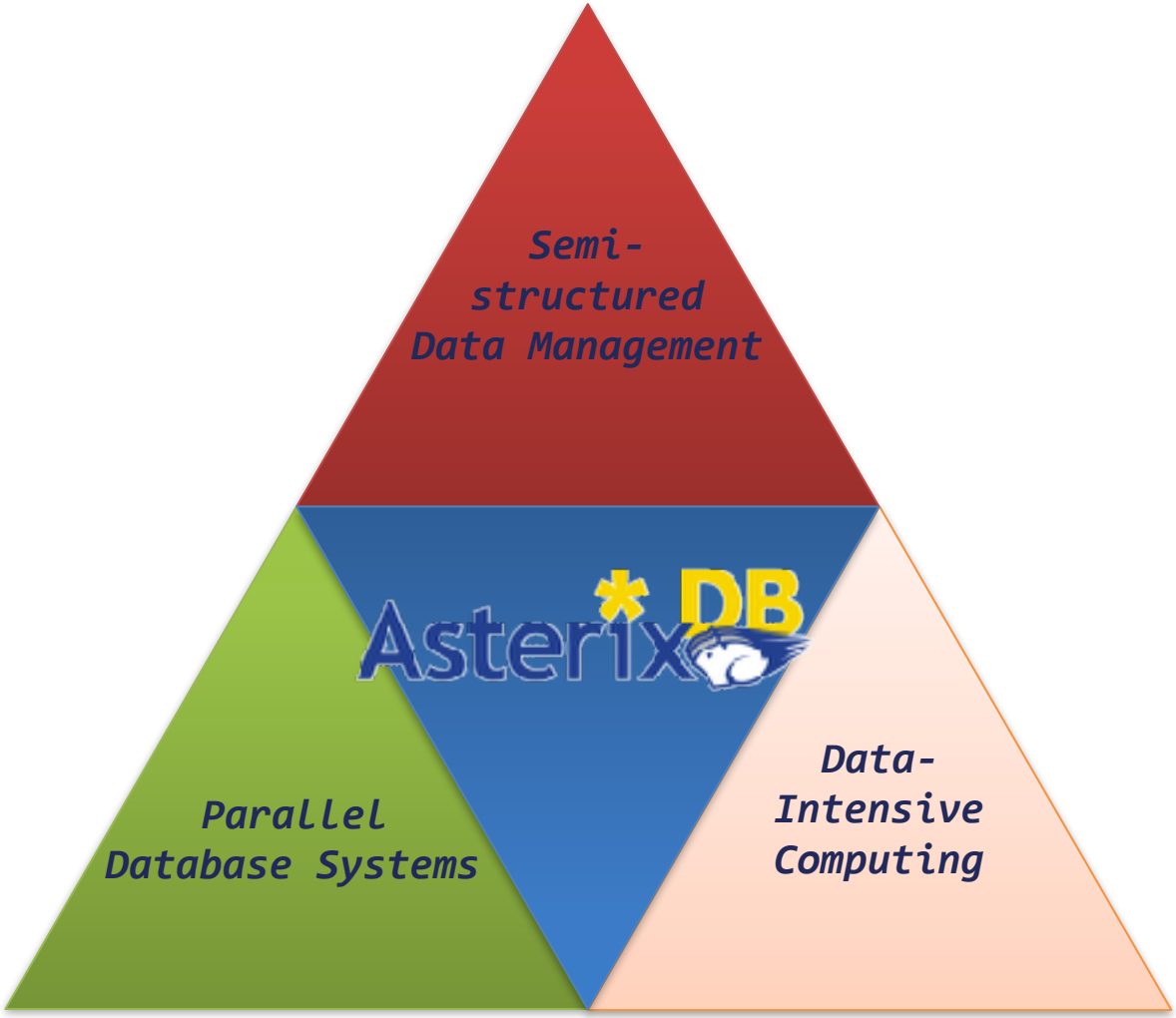
AsterixDB

An Open source BigData Management System

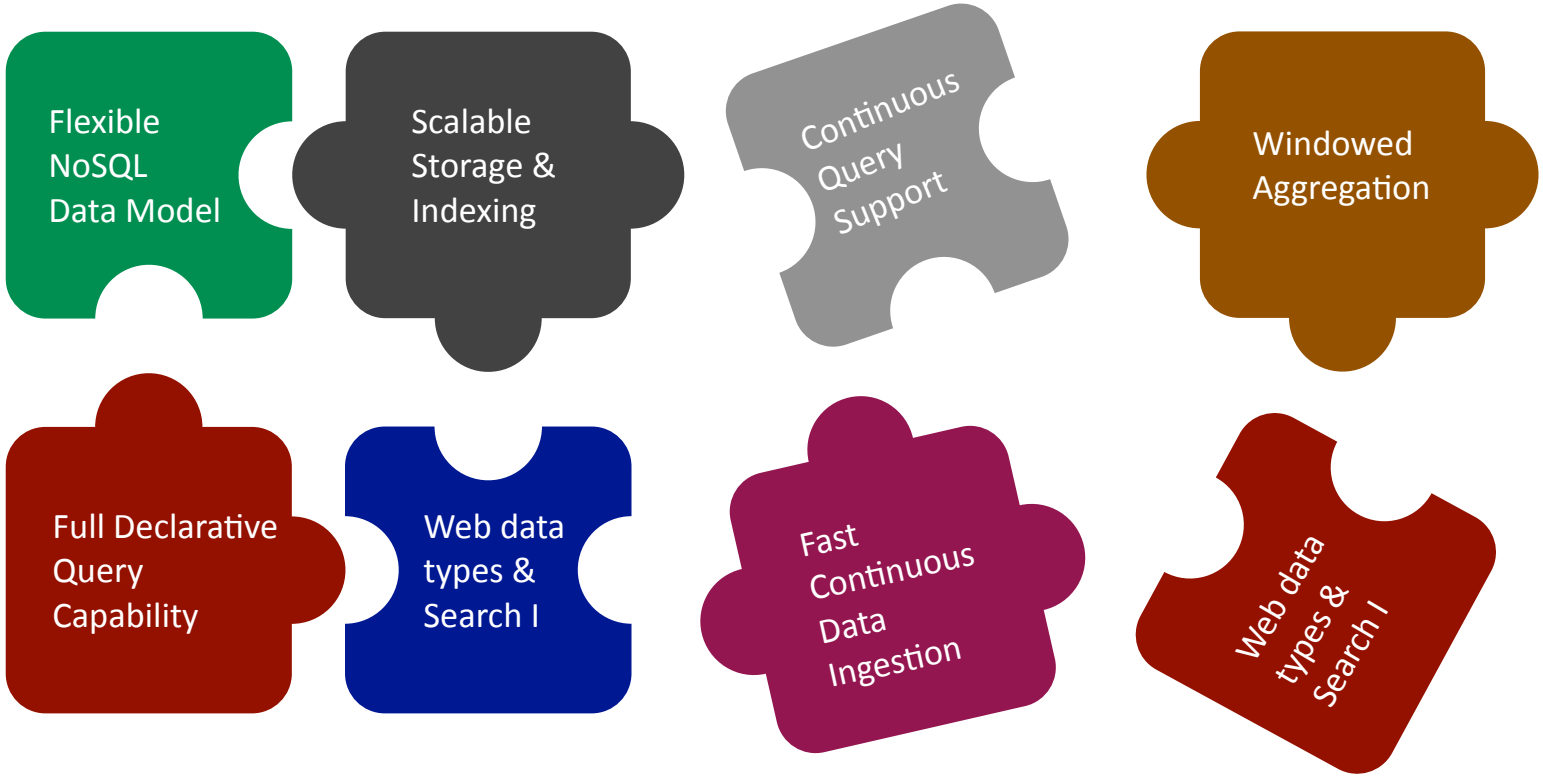
THE APACHE SOFTWARE FOUNDATION

Existing Solution(s)

AsterixDB: "One Size Fits a Bunch"



AsterixDB


Flexible NoSQL Data Model
Scalable Storage & Indexing
Continuous Query Support
Windowed Aggregation
Full Declarative Query Capability
Web data types & Search I
Fast Continuous Data Ingestion
Web data types & Search I

AsterixDB Features

Data feeds

Process incoming live data, and store it seamlessly.

Scalable runtime

AsterixDB is built on Hyracks, a parallel dataflow engine scale tested on hundreds of nodes.

External data

Query and Index your data in HDFS alongside data stored in AsterixDB.

Semi-structured data model

Give your data as much or as little schema as you want, in one system.

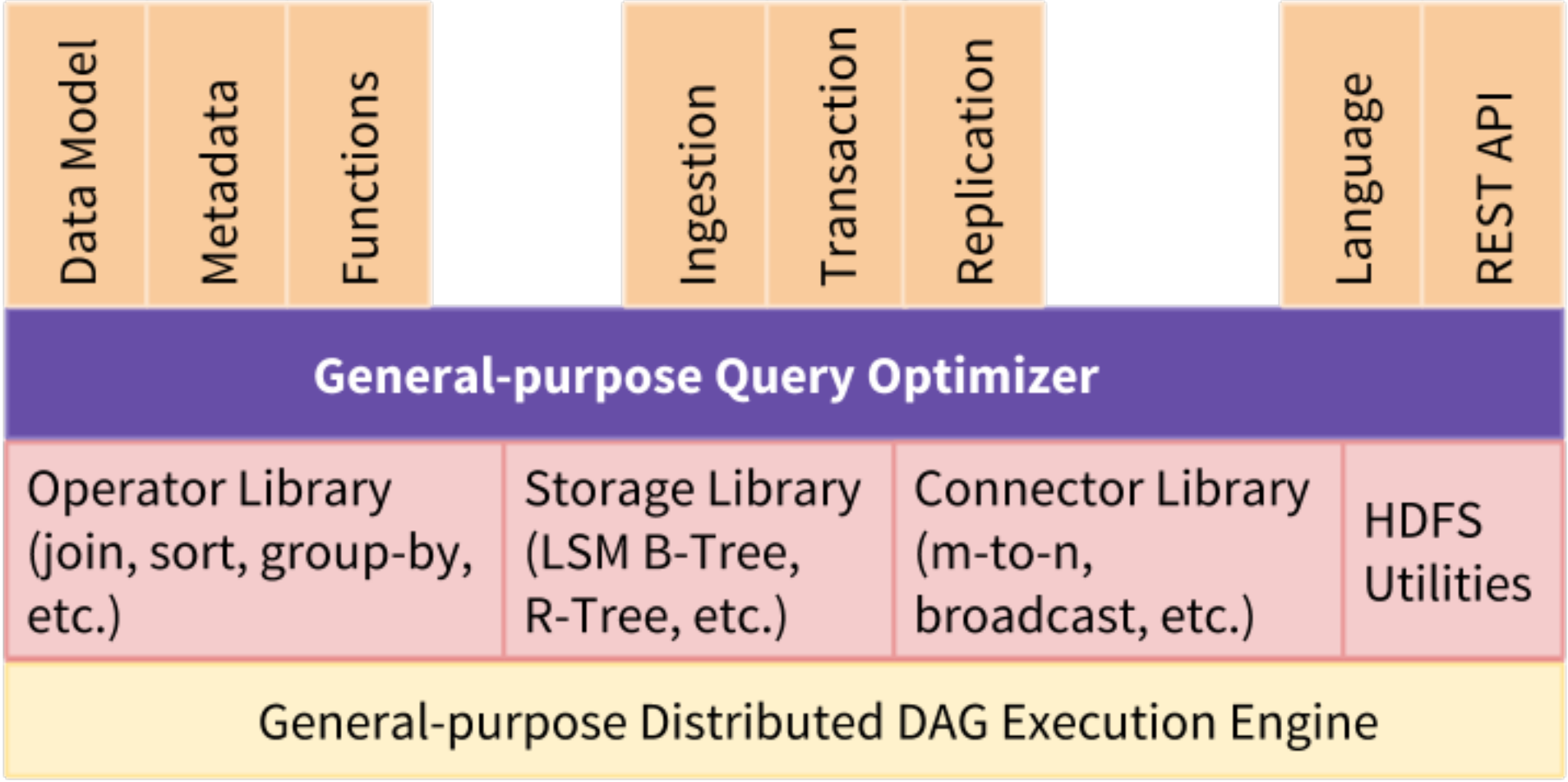
Log-structured storage

AsterixDB uses state of the art indexing techniques to support high speed data ingestion.

Rich datatypes

AsterixDB extends JSON datatypes with support for spatial and temporal data.

AsterixDB Architecuture



AsterixDB Data Model

```
create type TwitterUser as open {
screen-name: string,
   lang: string,
   friends_count: int32,
   statuses_count: int32,
   name: string,
   followers_count: int32
};
```

```
create type ProcessedTweet {
tweetid: string,
   userid: string,
   send-time: string,
   message-text: string,
   language: string,
   location: point?,
   referred-topics: {{string}}
};
```

```
create type RawTweet as open
{ tweetid: string,
   user: TwitterUserType,
   latitude: double?,
   longitude: double?,
   send-time: string,
   message-text: string,
   language: string
};
```

```
create dataset RawTweets(RawTweet)
primary key tweetid;

create dataset
ProcessedTweets(ProcessedTweet)
primary key tweetid;

create index locationIdx on
ProcessedTweets(location) type rtree;
```

Scalable Storage and Indexing

- Data is hash-partitioned on the basis of primary key
- Primary Index partition on each node with co-located secondary index partitions.
- **L**og **S**tructured **M**erge (LSM)-based Index
  A deferred-update, append-only data structure optimized for insert-intensive workloads [P. O'Neil et al, Acta Informatica, 1996]
  - Adopted in popular NoSQL systems
- "Storage Management in AsterixDB", VLDB, 2015

Full Declarative Query Support

Popularity of tourist spots as measured by the number of tweets mentioning a tourist spot:

```
for $tweet in dataset ProcessedTweets
for $spot in dataset TouristSpots
where some $topic in $tweet.referred-topics
satisfies contains($topic, $spot.name)
group by $spot.name with $tweet
return {
    ”spot": $spot.name,
    "count": count($tweet)
}
```

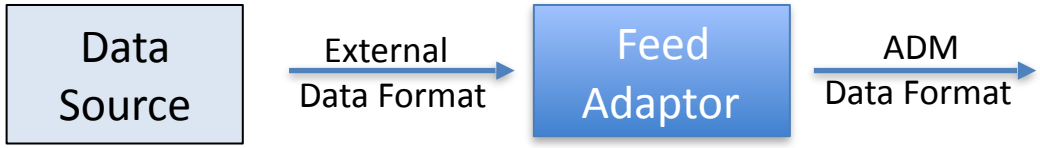
Focus: Data Feeds

Data Feed: Flow of data from an external source into persistent (indexed) storage inside a **B**ig **D**ata **M**anagement **S**ystem (BDMS).

Data Feed Management: Task of maintaining the continuous flow of data.

Data Ingestion Basics

- AsterixDB Query Language (AQL) with built-in support for data ingestion.



- Functionality of establishing connection with a data source and receiving, parsing and translating data into ADM records -> Feed Adaptor

- **Pull/Push-based ingestion**

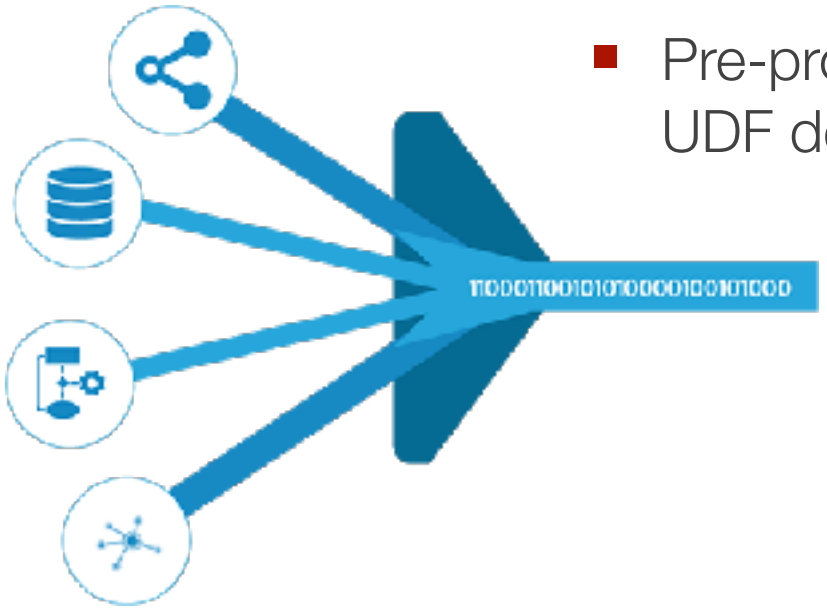
```
create feed TwitterFeed using TwitterAdaptor
("interval"=60);

create feed TwitterFeed using TwitterPushAdaptor;

create feed CNNFeed using CNNAdaptor
("topics"="politics, sports");
```

Pre-processing Collected Data

- Pre-processing arriving data;
  - E.g., adding attributes, sentiment analysis, feature extraction, etc.
- Pre-processing expressed as a UDF defined in **AQL** or **Java**.



Pre-processing Collected Data

```
create type RawTweet as open
{   tweetid: string,
    user: TwitterUserType,
    latitude: double?,
    longitude: double?,
    send-time: string,
    message-text: string,
    language: string
};
```

UDF

```
create type ProcessedTweet as open {
    tweetid: string,
    userid: string,
    send-time: string,
    message-text: string,
    language: string,
    location: point?,
    referred-topics: {{string}}
};
```

```
create feed TwitterFeed using
TwitterAdaptor
("query"="Obama", "interval"=60)
apply function addFeatures;
```

Feed Adaptor/UDF act as pluggable components

Challenge solved: Genericity/Extensibility

Fetch Once, Compute Many!

- Multiple applications may wish to consume the data ingested from a feed
  - **Primary Feed:** A feed that gets data from an external source.
  - **Secondary Feed:** A feed that derives its data from another feed.
- Pre-processing required by each application might overlap and could be done in an incremental fashion.

Fetch Once, Compute Many! (2)

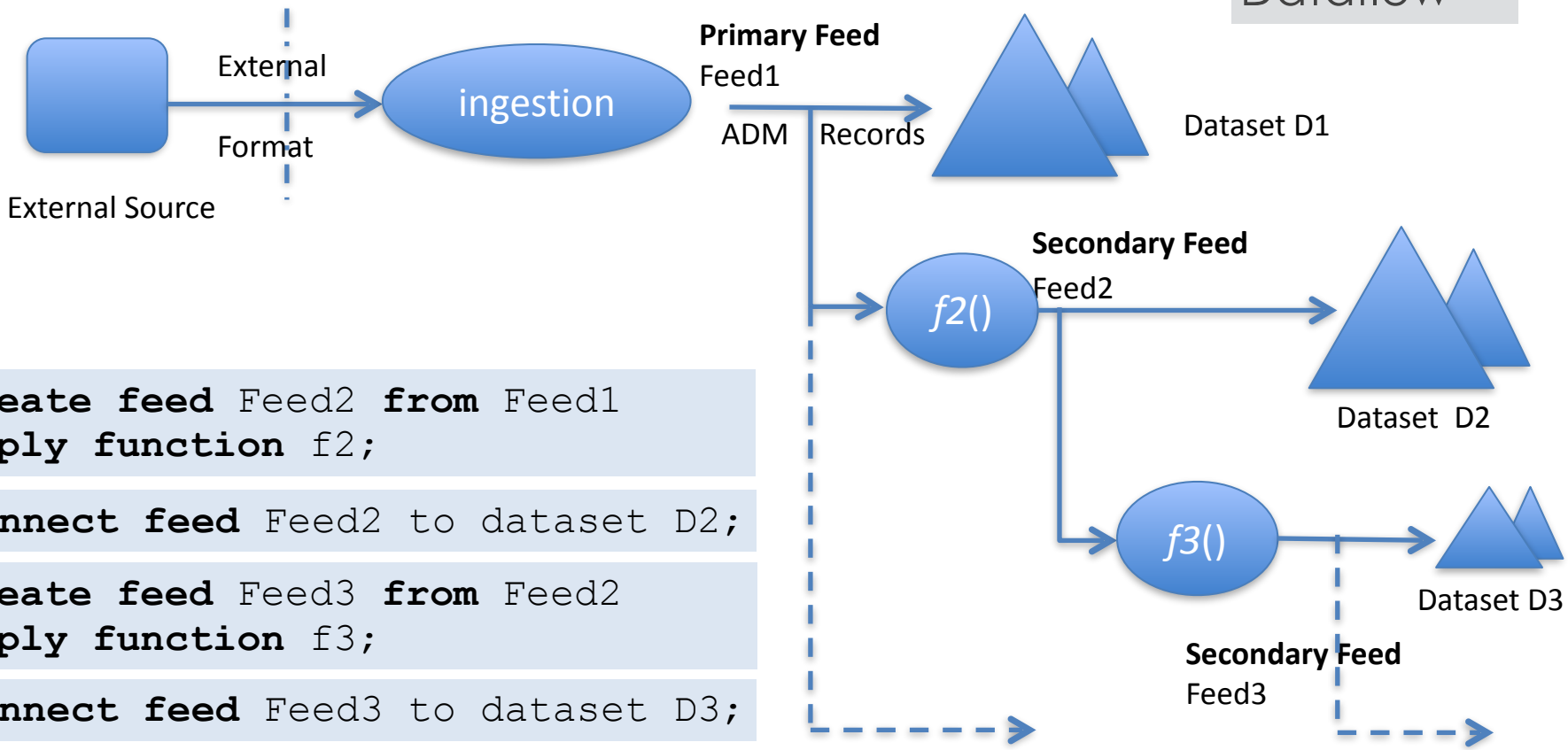
- **Secondary feed**
  - can be persisted to a target dataset
  - can be used to derive other secondary feeds to form a Cascade Network

Challenge solved: Fetch Once, Compute Many

Building a Cascade Network of Feeds


```
create feed Feed1 using myAdaptor (…);
connect feed Feed1 to dataset D1;
```

Logical Dataflow




```
create feed Feed2 from Feed1
apply function f2;
```

```
connect feed Feed2 to dataset D2;
```

```
create feed Feed3 from Feed2
apply function f3;
```

```
connect feed Feed3 to dataset D3;
```

Lifecycle of a Feed

- Subsequent to a connect feed statement, a feed is said to be in a connected state.
- Multiple feeds can simultaneously be connected to a single dataset.
- Connecting a secondary feed does not require the parent (or any ancestor) feed to be in a connected state.
- Additional feeds can be added to an existing hierarchy and connected to a dataset at any time.

```
disconnect feed TwitterFeed
from dataset Tweets;
```

Policies for Data Ingestion

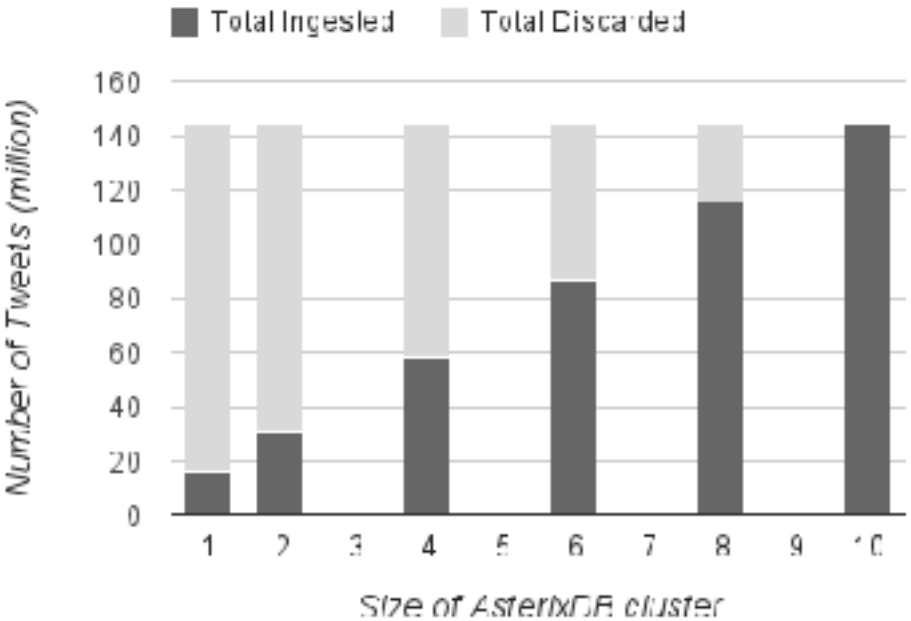
- Concurrent feeds (and queries) compete for resources (CPU cycles, network bandwidth, disk IO).
- Each feed is different
  - Rate of flow of data
  - Complexity involved in pre-processing
  - Constraints imposed by consuming application

Policies for Data Ingestion (2)

- A strategy for allocation of resources or handling of failures may work for one feed but may be a misfit or an over-fit for another!
- AsterixDB allows each data feed to have its own associated "**Ingestion Policy**"

Scalability

- 10 node cluster; each node 8gb of RAM
- Custom Tweet Generator (TweetGen): Output JSON tweets at a configurable rate (twps) and a pattern.
- Custom feed adaptor – TweetGenAdaptor
- UDF with 3 ms execution time.
- 6 instances of TweetGen for a duration of 20 minutes, each generating at 20k twps



Performance Metric: **Data Loss**

Use '**Discard**' policy to evaluate ability to successfully ingest data as a function of available

Data "Indigestion"

- Data arriving from external source may be subjected to expensive pre-processing (UDF) prior to persistence.
- Data arrival rate may exceed the rate at which it can be pre-processed!
- End user may form a custom policy; e.g., spill excess records to disk and throttle if the spillage crosses a configured threshold.

Data "Indigestion" (2)

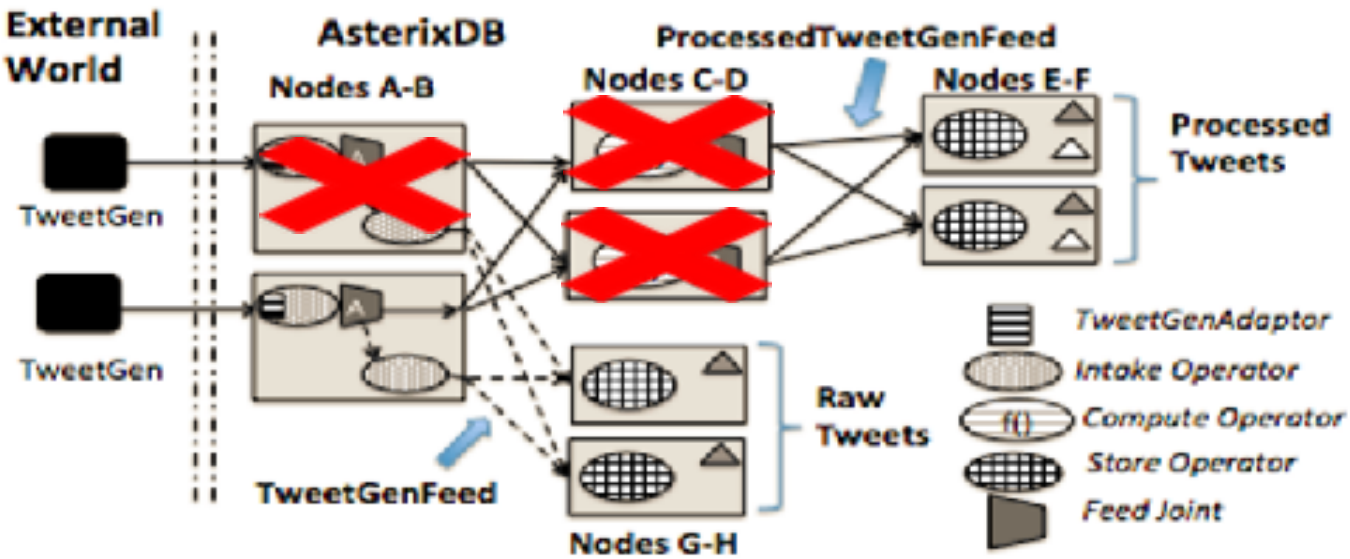
Policy	Approach to handling excess records
Basic	Buffer excess records in memory for deferred processing
Spill	Spill excess records to disk for deferred processing
Discard	Discard excess records altogether
Throttle	Randomly filter out records to regulate the rate of arrival
Elastic	Scale out/in to adapt to the rate of arrival

```
connect feed TwitterFeed to dataset Tweets
using policy Elastic;
```

Fault Tolerance

- System can recover from **soft and hard failures**
- Recovery from kinds of failures of a data ingestion data flow dictated by associated **ingestion policy**

Fault-Tolerance (contd.)



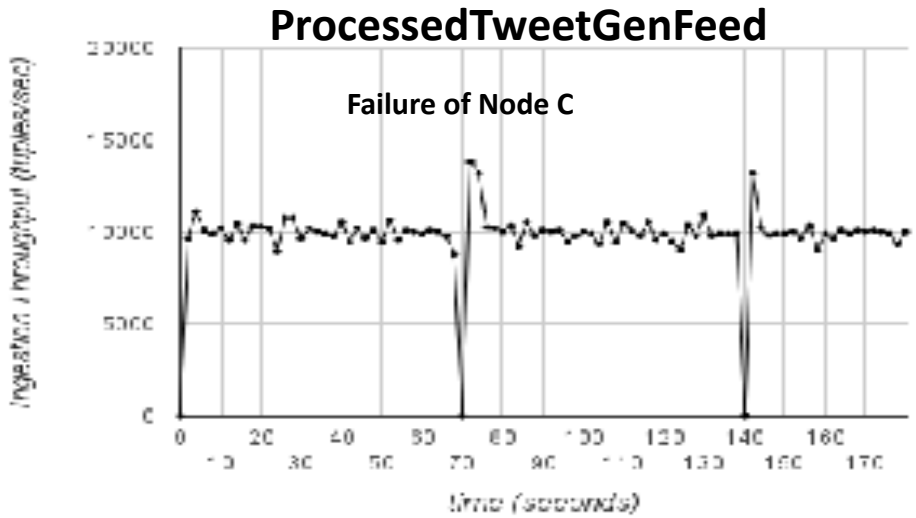
At t=70 seconds

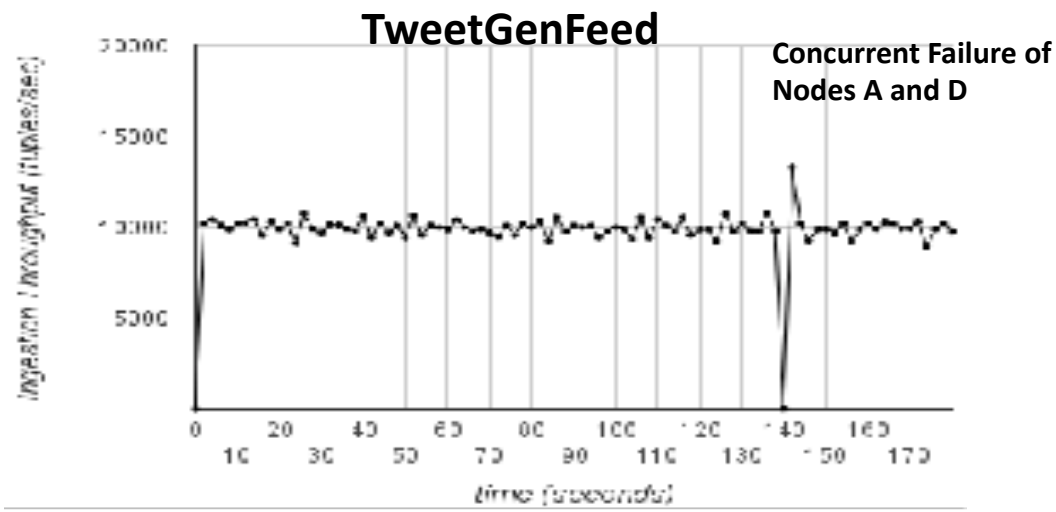
At t=140 seconds

To make things interesting…

```
connect feed ProcessedTweetGenFeed to dataset
ProcessedTweets
using policy FaultTolerant;
connect feed TweetGenFeed to dataset RawTweets
using policy FaultTolerant;
```

Fault-Tolerance (contd.)





- **Fault Isolation**
- **Recovery Time**

Summary

- Scalable, Fault-Tolerant, and Elastic data ingestion facility
- Plug-and-Play model to cater to wide variety of data sources.
- Custom built solution – Storm + MongoDB did not compare – user experience or performance characteristics (see paper).
- We can reduce the number of moving parts needed to handle Big Data.
- AsterixDB now an **Apache Incubator Project**.

Links

- Storm:
  - http://storm.apache.org/index.html
  - Tutorial: http://storm.apache.org/tutorial.html
- AsterixDB
  - https://asterixdb.apache.org
  - Documentation: https://ci.apache.org/projects/asterixdb/index.html

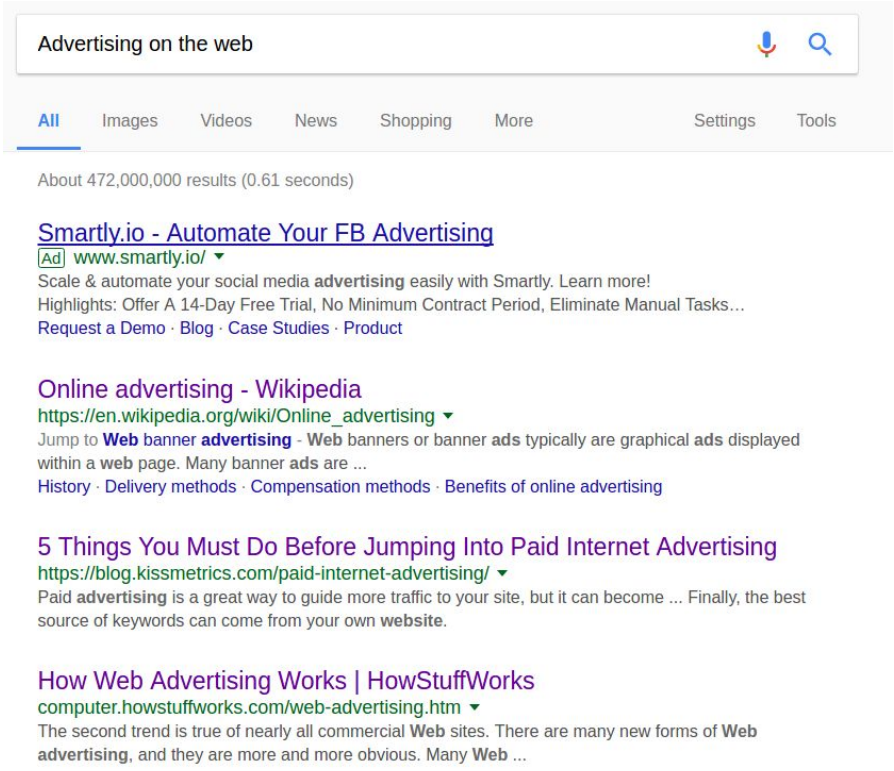
Advertising on the Web

2017-03-18

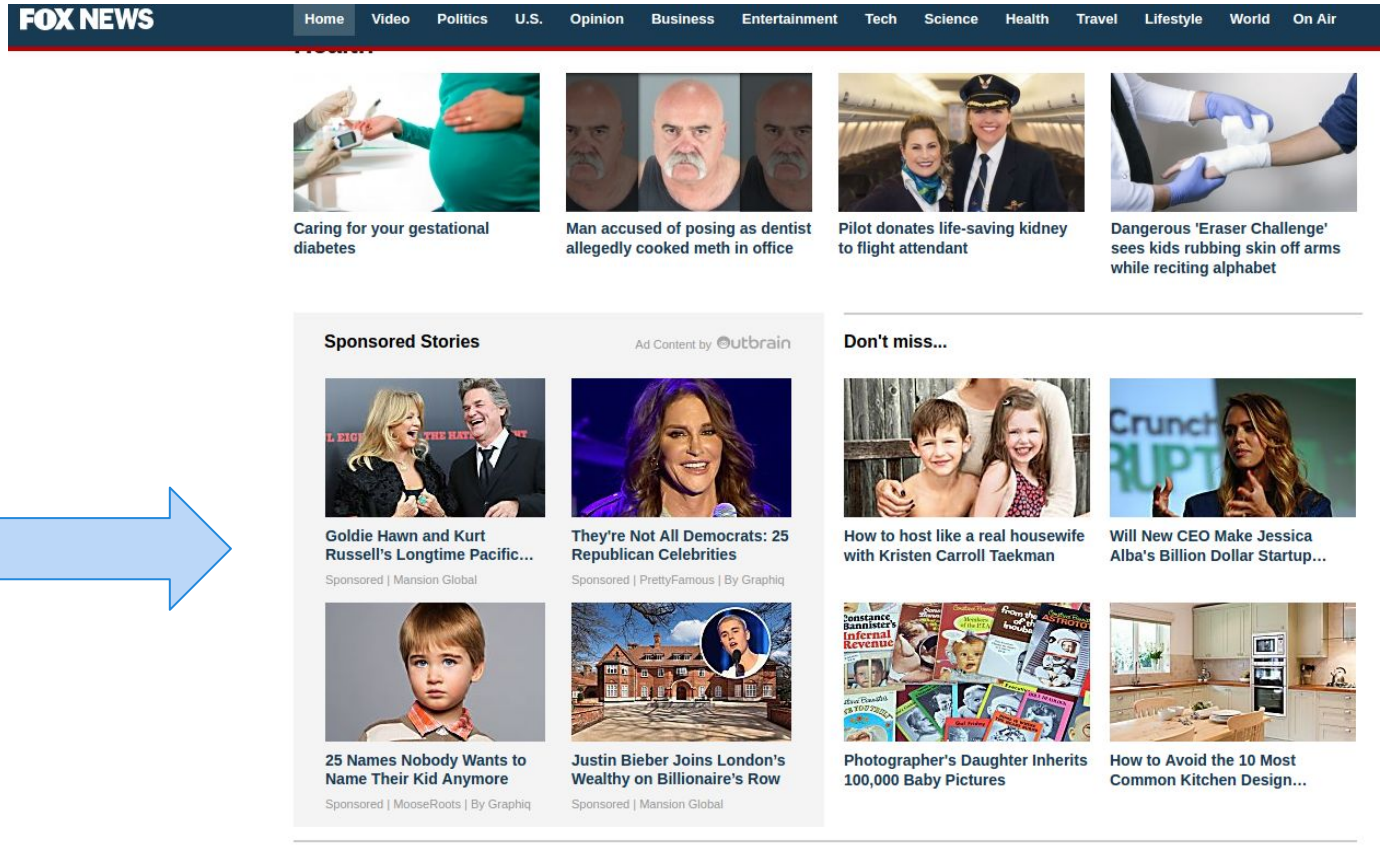
Advertisement

“a notice or announcement in a public medium promoting a product, service, or event or publicizing a job vacancy.”

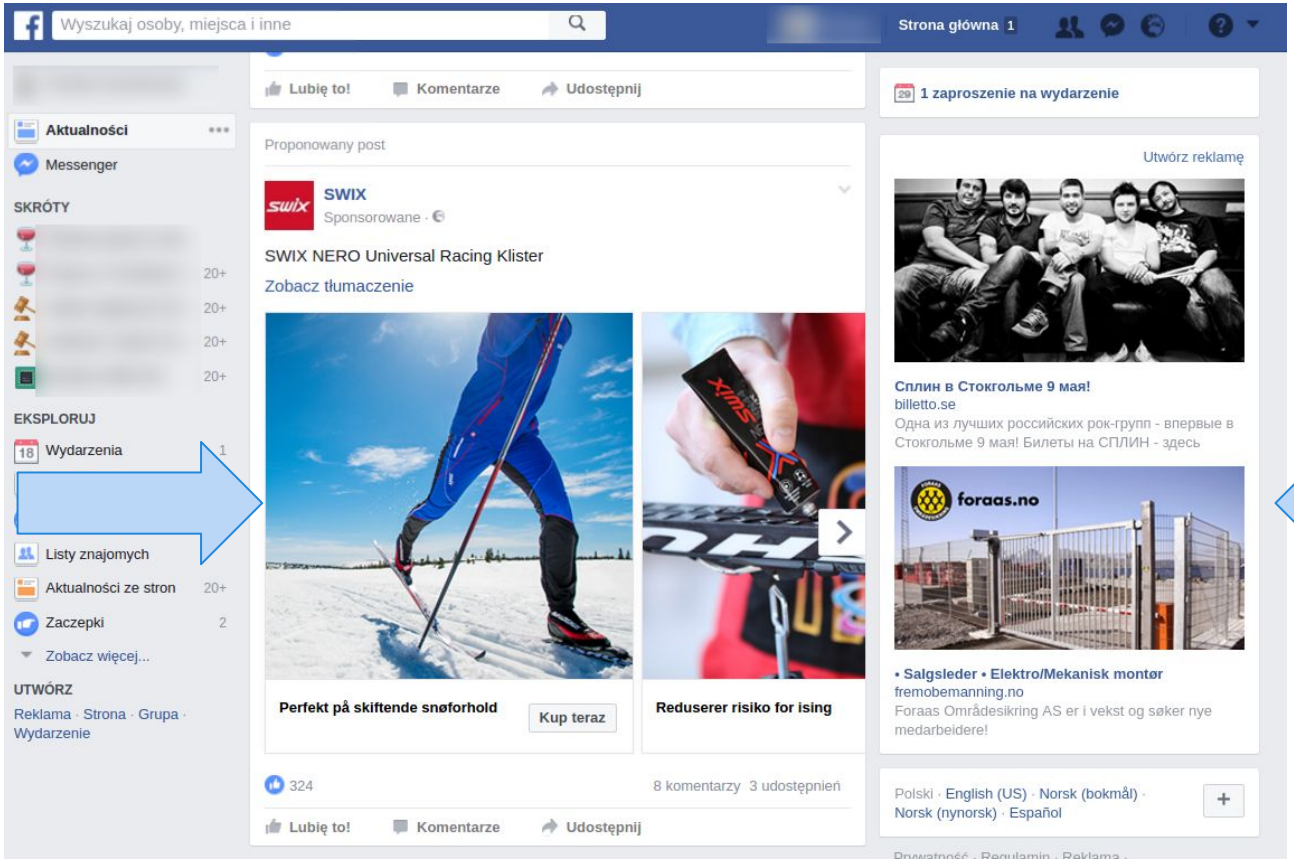
Where can I see it?



Where can I see it?



Where can I see it?



Where can I see it?

- does not matter what do you do
- in what language
- on which site

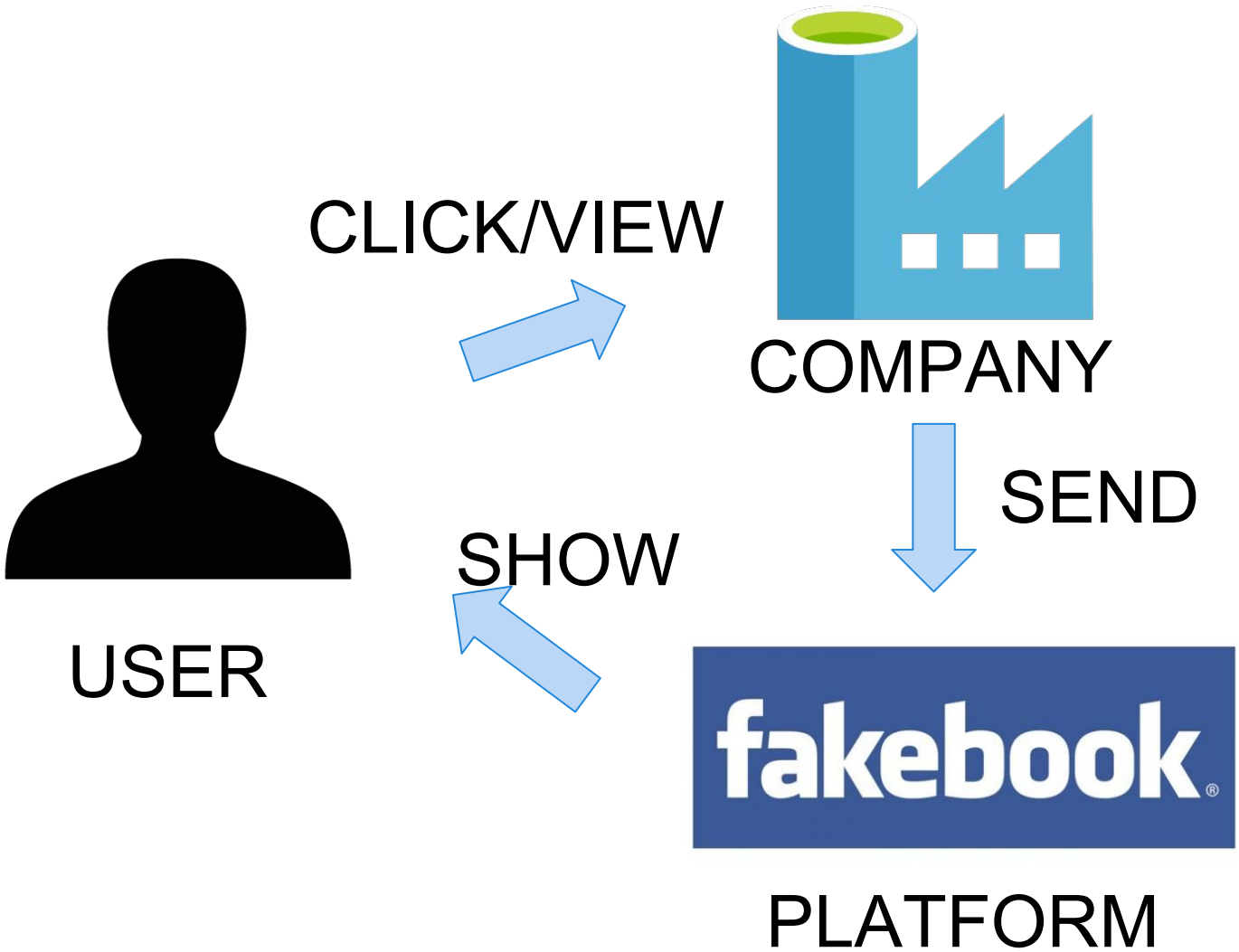
ads are

EVERYWHERE

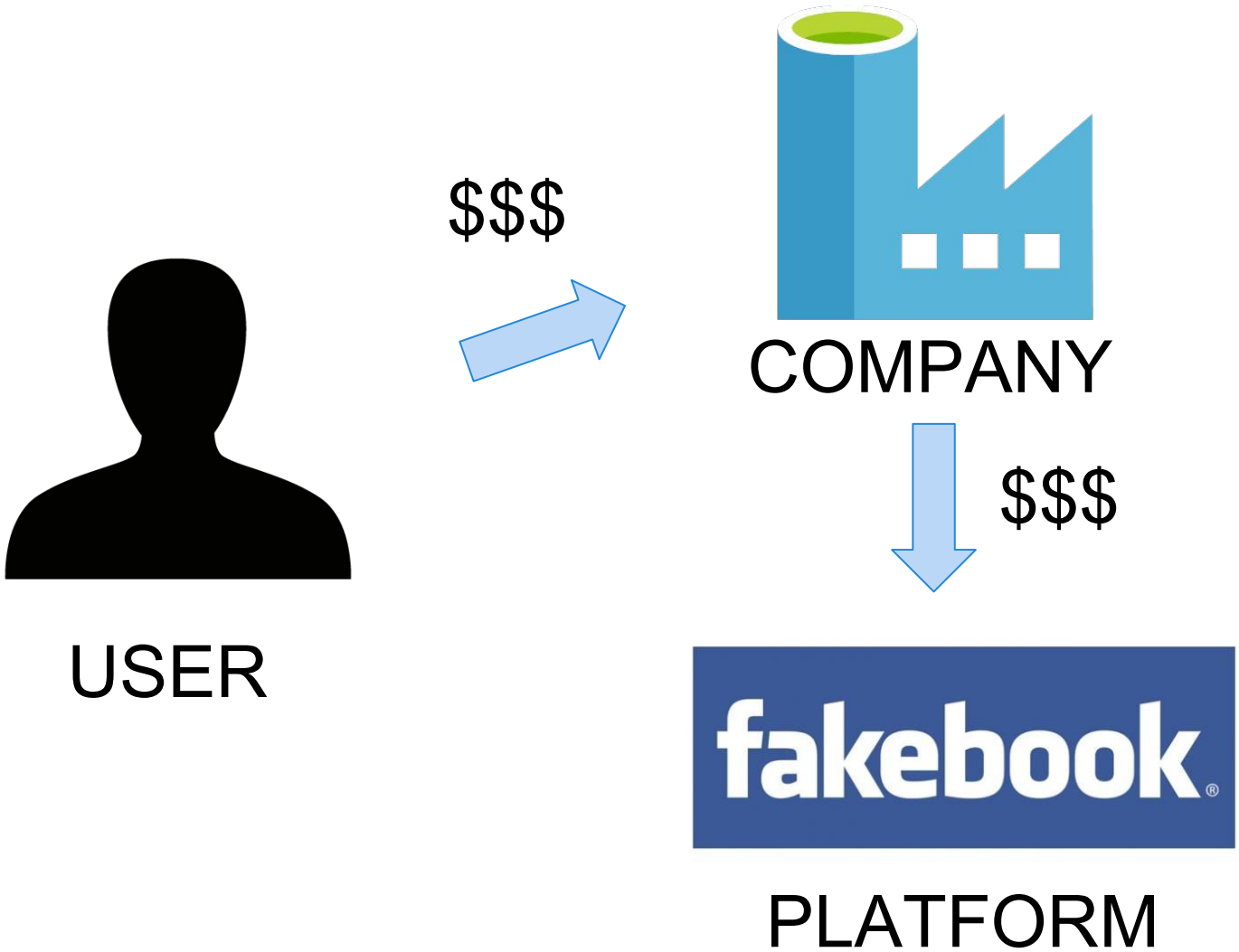
Why?

Google's total advertising revenues of <u>USD $43.7 billion</u> in 2012

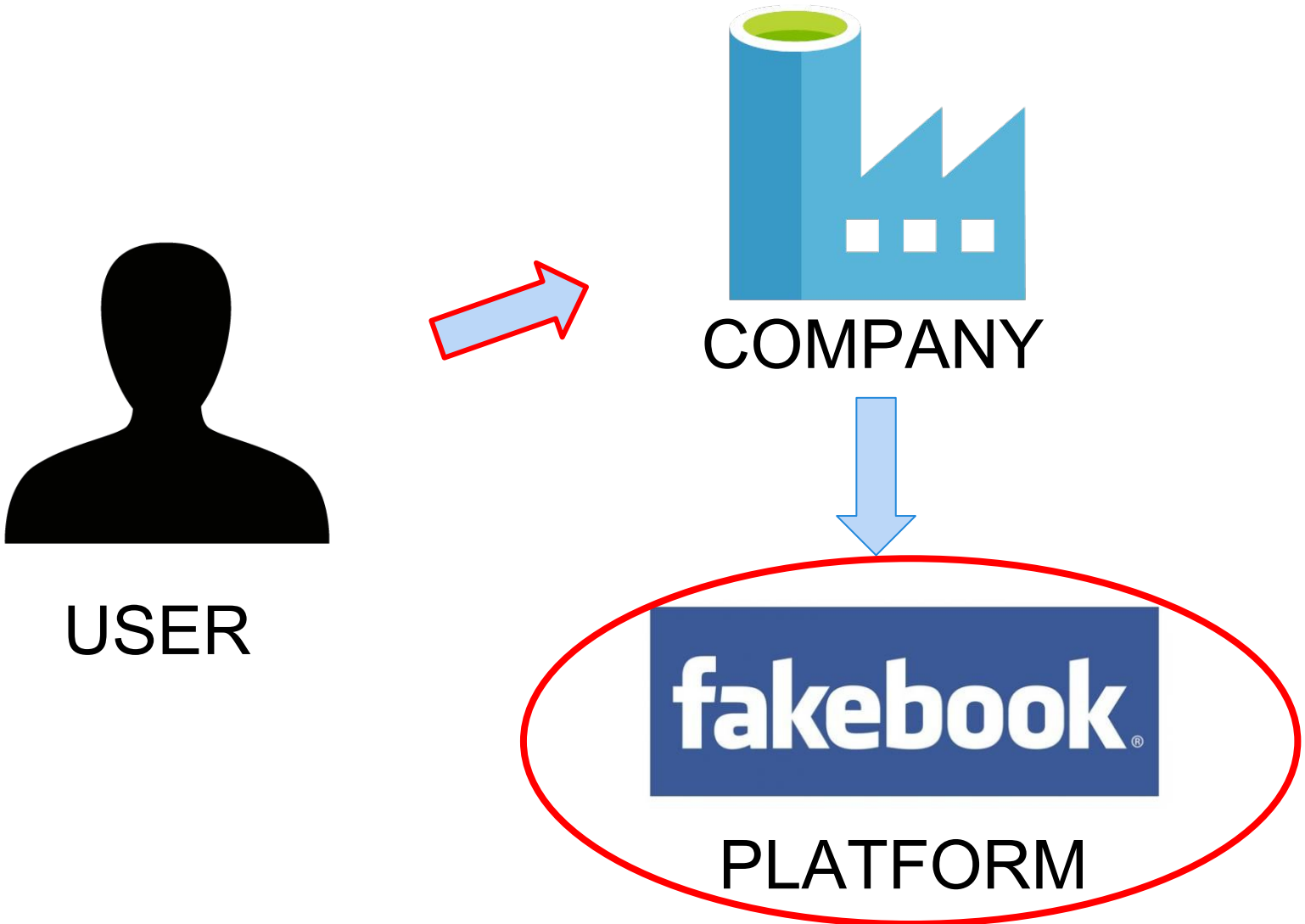
Setting: three sides - advert flow



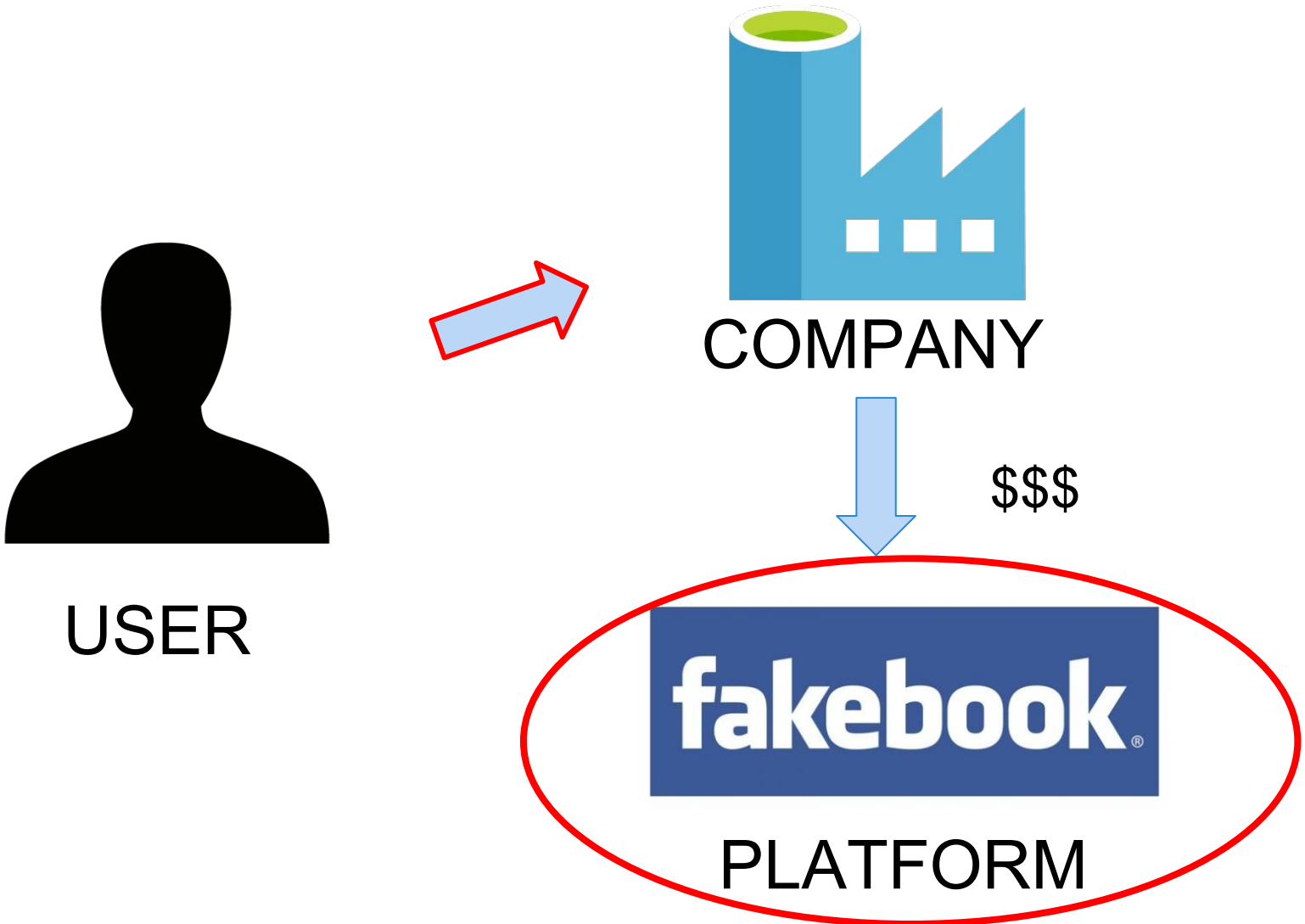
Setting: three sides - money flow



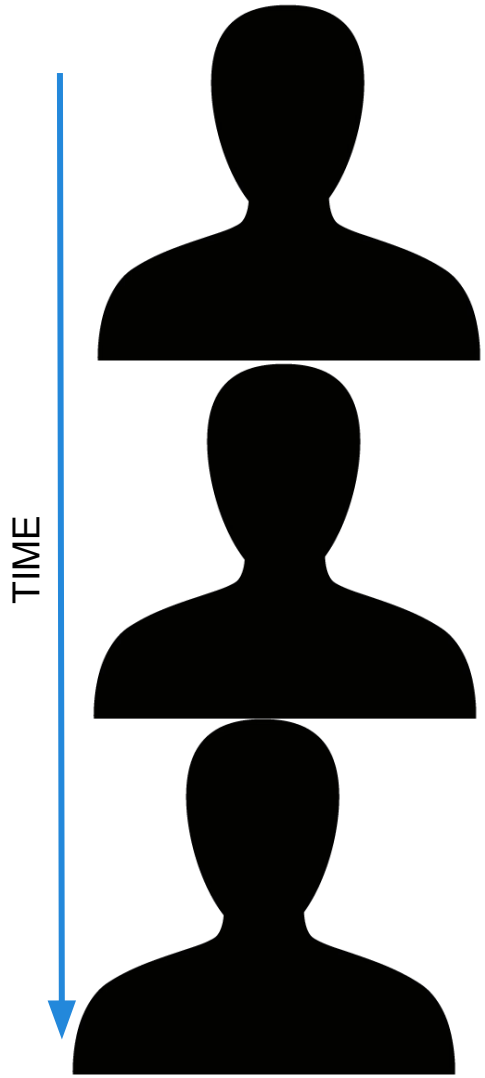
Setting: Where are we?

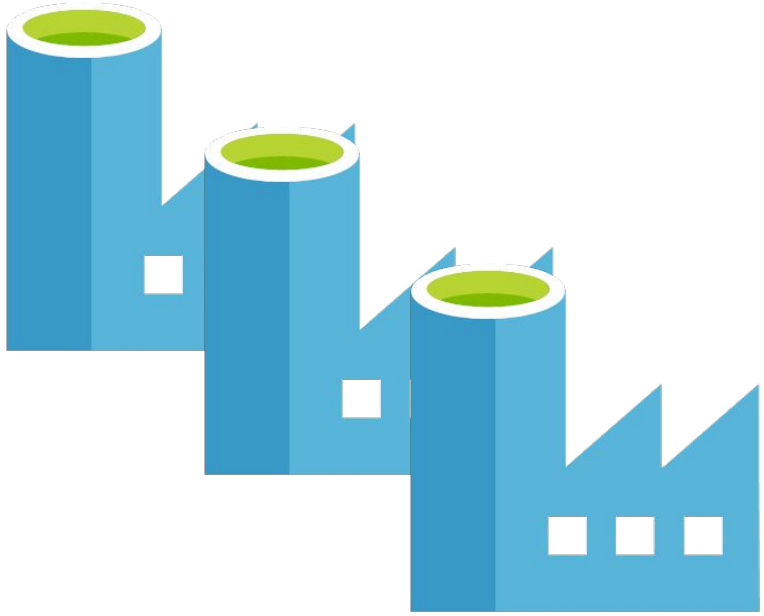


Setting: What do we want?



Setting: We forgot about TIME





fakebook

PLATFORM

Revenue

How much do we (as a platform) earn from our matching?:

- **per impression**
- **per click**

Adwords model

- **Given:**
  - **1.** A set of bids by advertisers for search queries
  - **2.** A click-through rate for each advertiser-query pair
  - **3.** A budget for each advertiser (say for 1 month)
  - **4.** A limit on the number of ads to be displayed with each search query
- **Respond to each search query with a set of advertisers such that:**
  - **1.** The size of the set is no larger than the limit on the number of ads per query
  - **2.** Each advertiser has bid on the search query
  - **3.** Each advertiser has enough budget left to pay for the ad if it is clicked upon

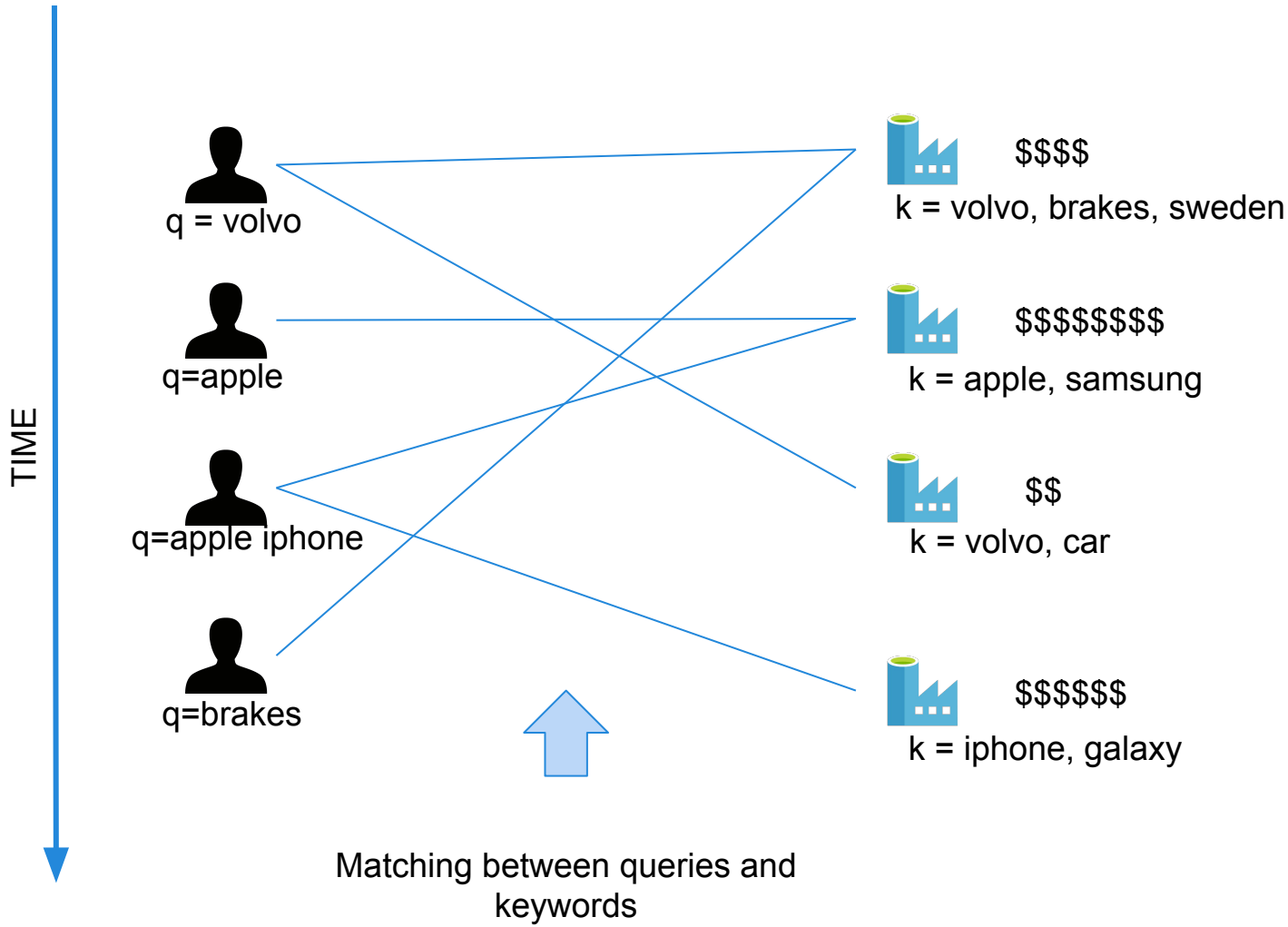
Only ~1 impression per 100 is clicked: CTR

Advertiser	Bid	CTR	Bid * CTR
A	$1.00	1%	1 cent
B	$0.75	2%	1.5 cents
C	$0.50	2.5%	1.125 cents
		Click through rate	Expected revenue

Simplifications

- Each user has one advert slot
  - If he has more we can just repeat the ad selection for each of them sequentially
- Every advert has the same expected revenue

Adwords model



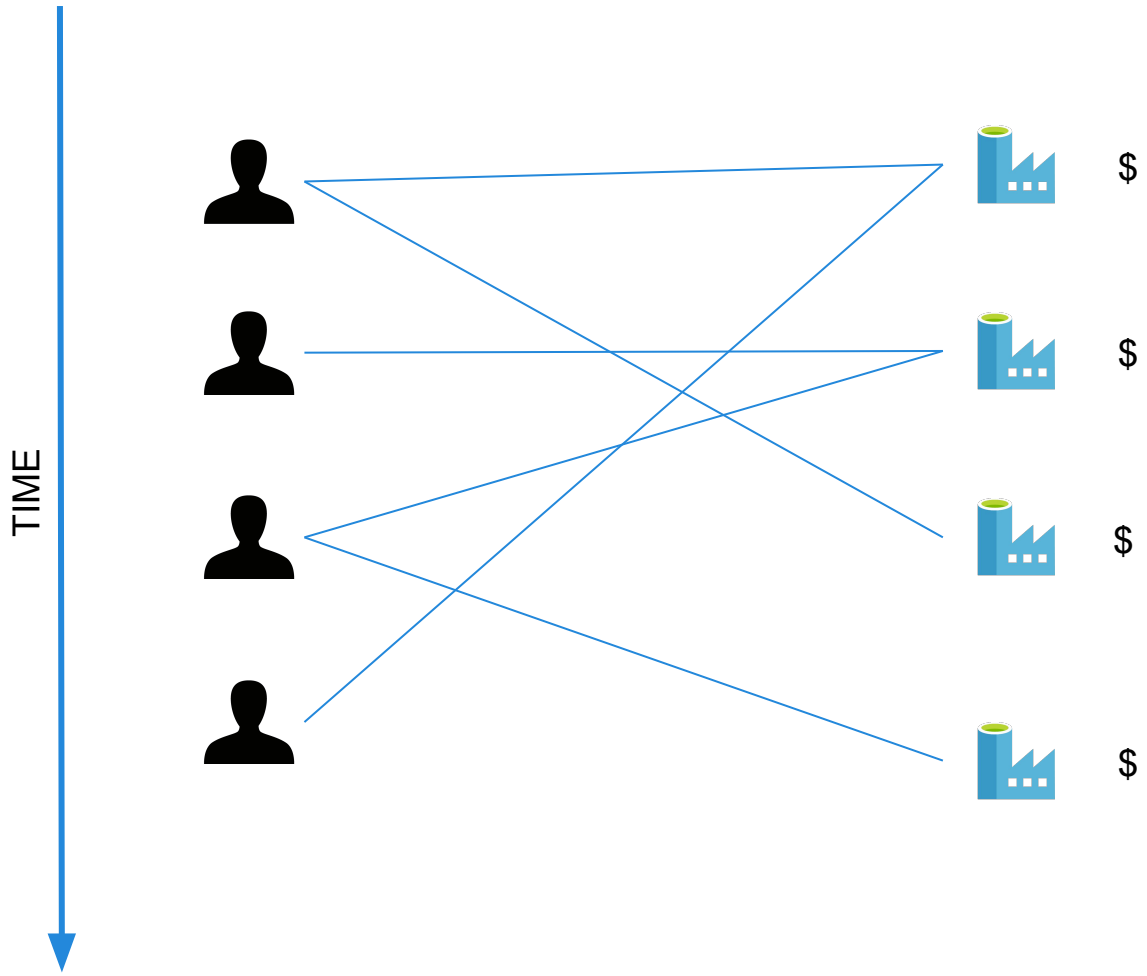
Greedy algorithm

For a query pick any advertiser who has bid for that query.

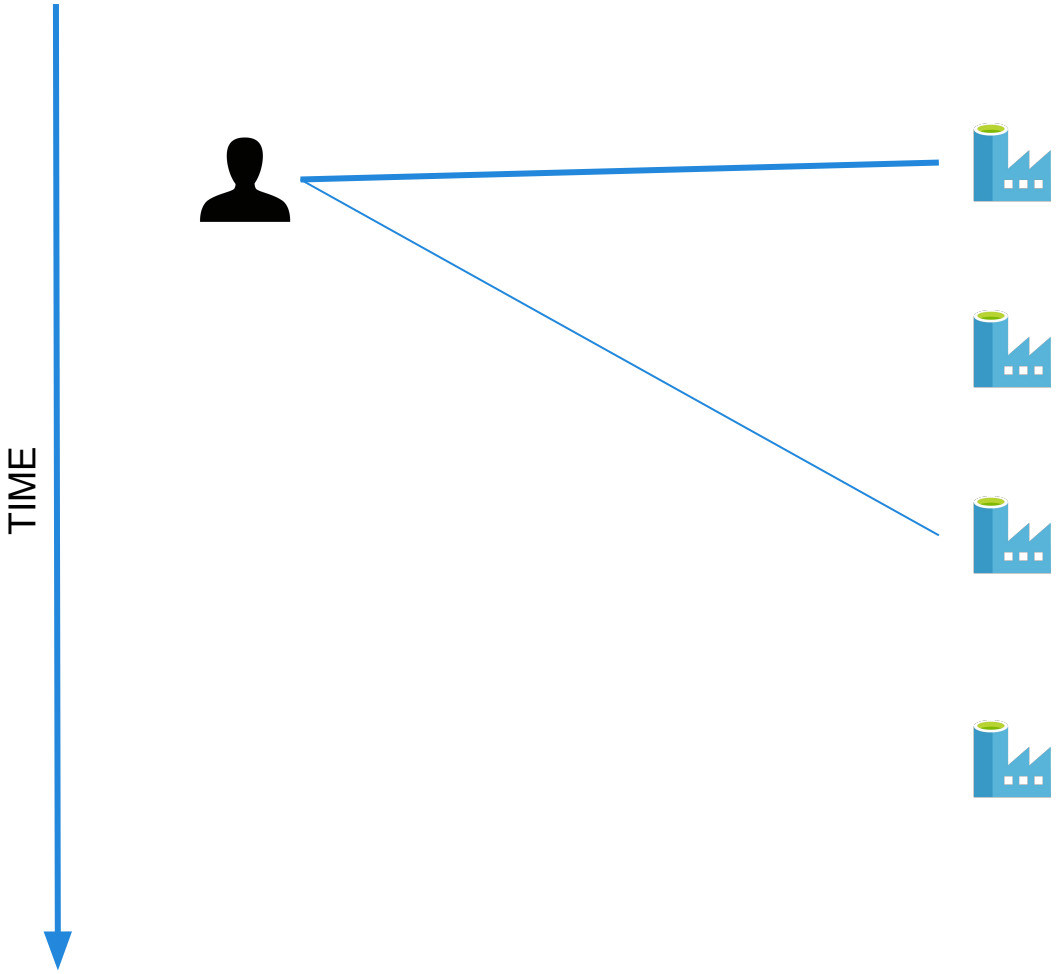
(Few possibilities in when there are several bidders.)

Simple Greedy with everybody having 1$

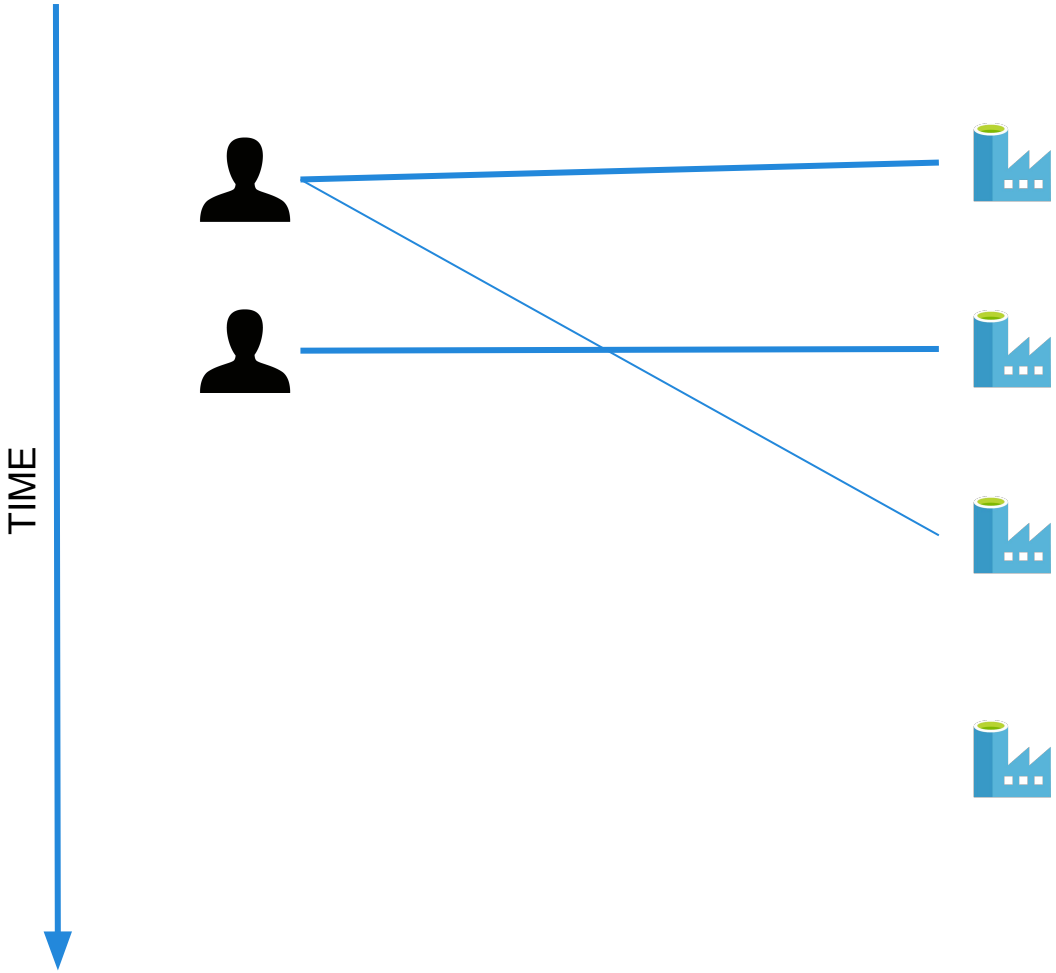
Matching 1: everybody has 1$



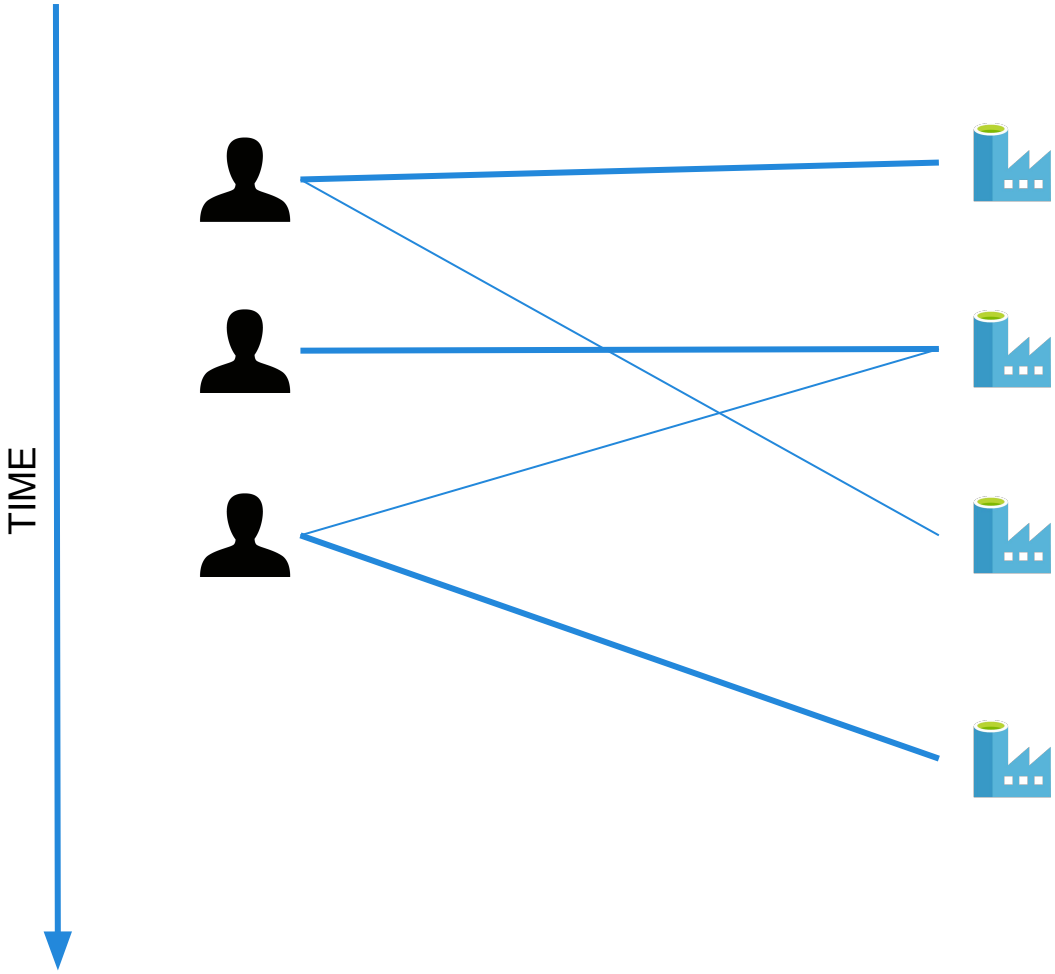
Online Greedy Matching



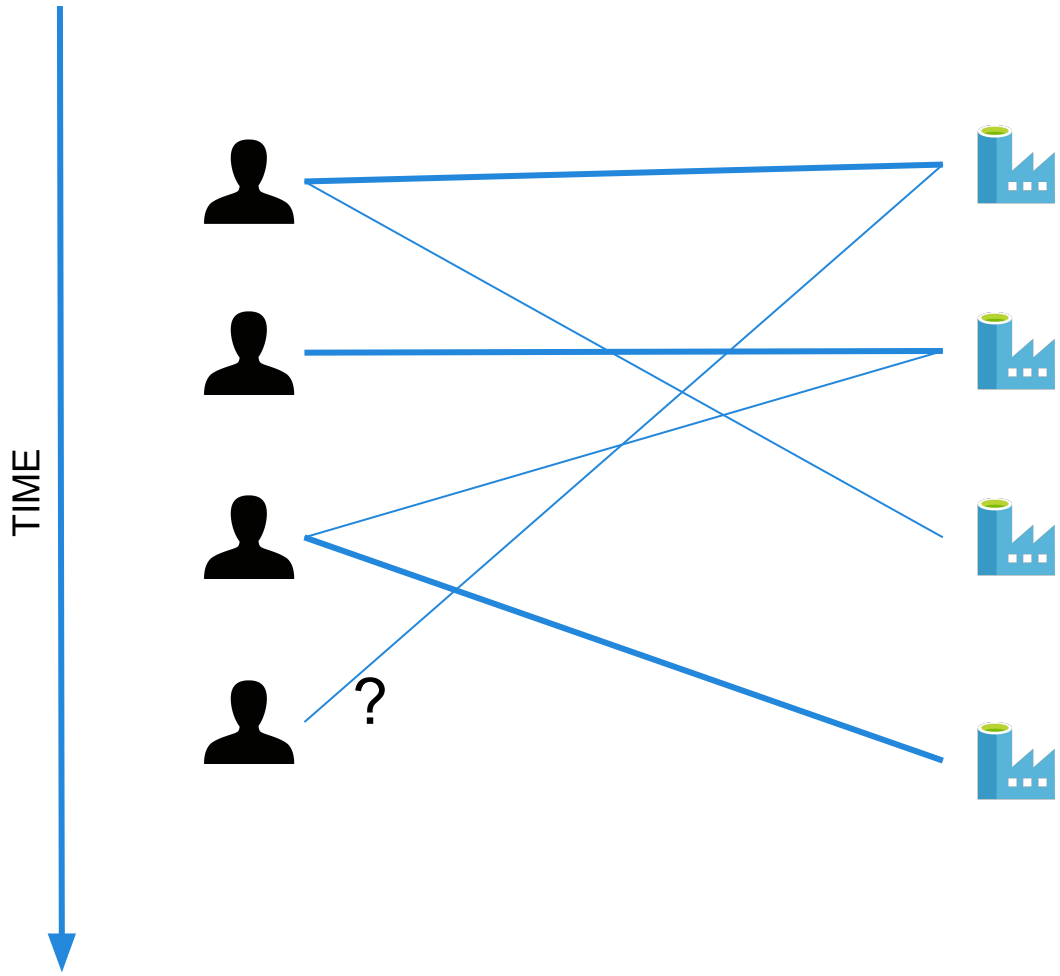
Online Greedy Matching



Online Greedy Matching

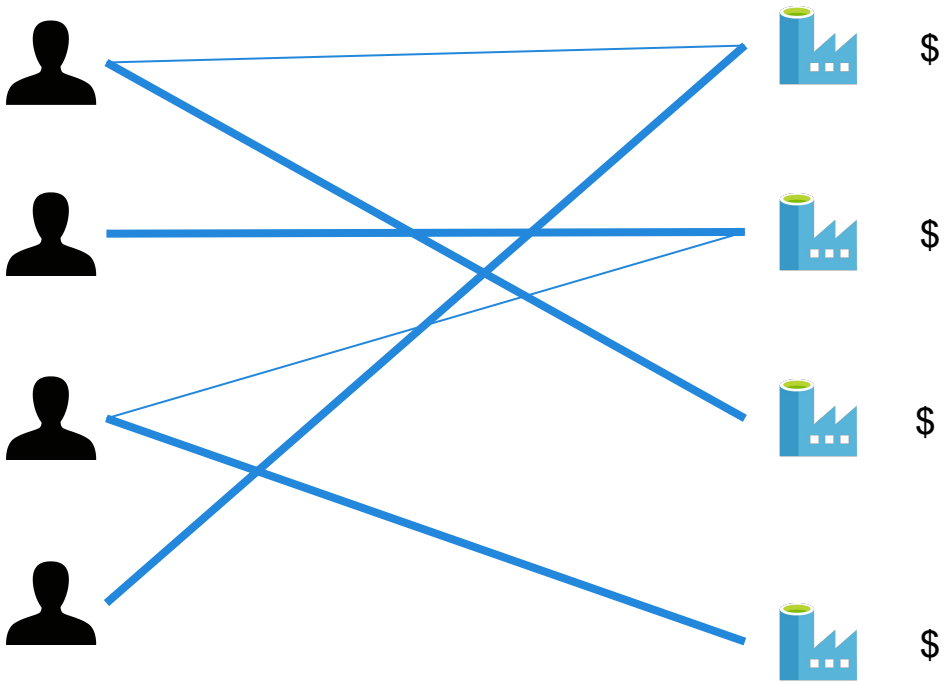


Online Greedy Matching

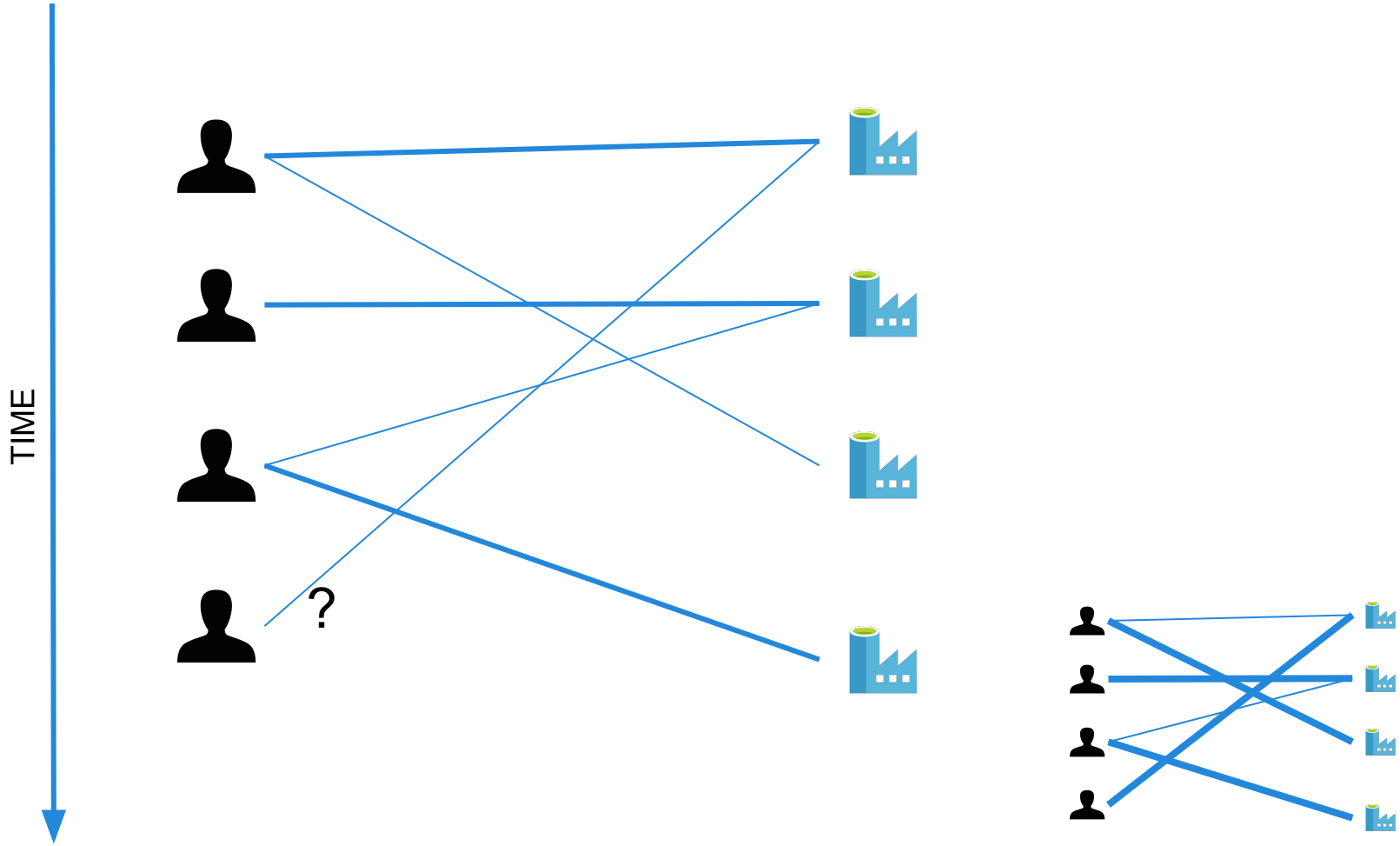


What’s the perfect matching?

Perfect Matching



Online Greedy Matching



Greedy with 2 companies (bidders) with 3$

Remark: on plots we skip lines representing possible matchings (that can be inferred from keywords overlapping)

Adwords (simplified: k=1) model

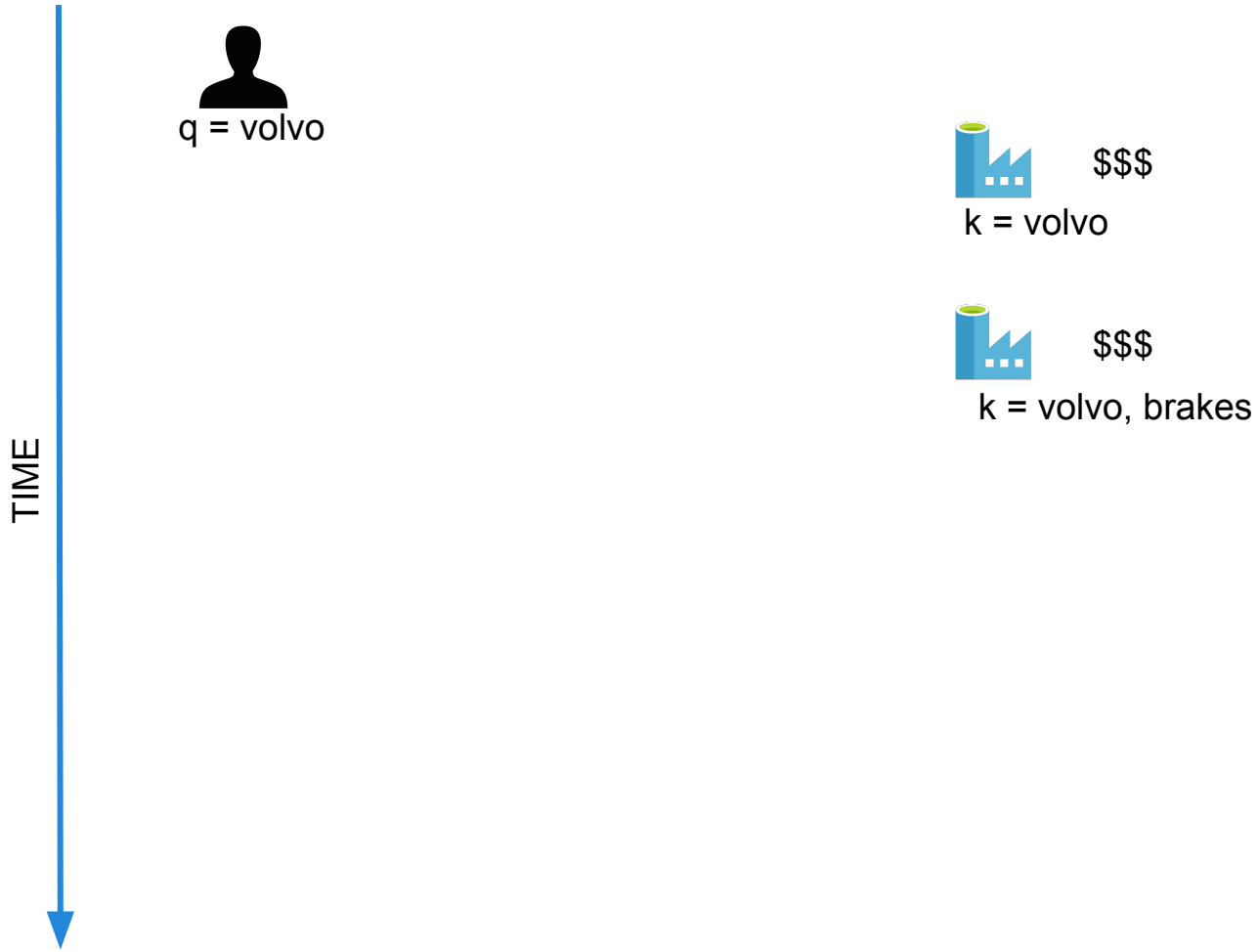
$$$

k = volvo

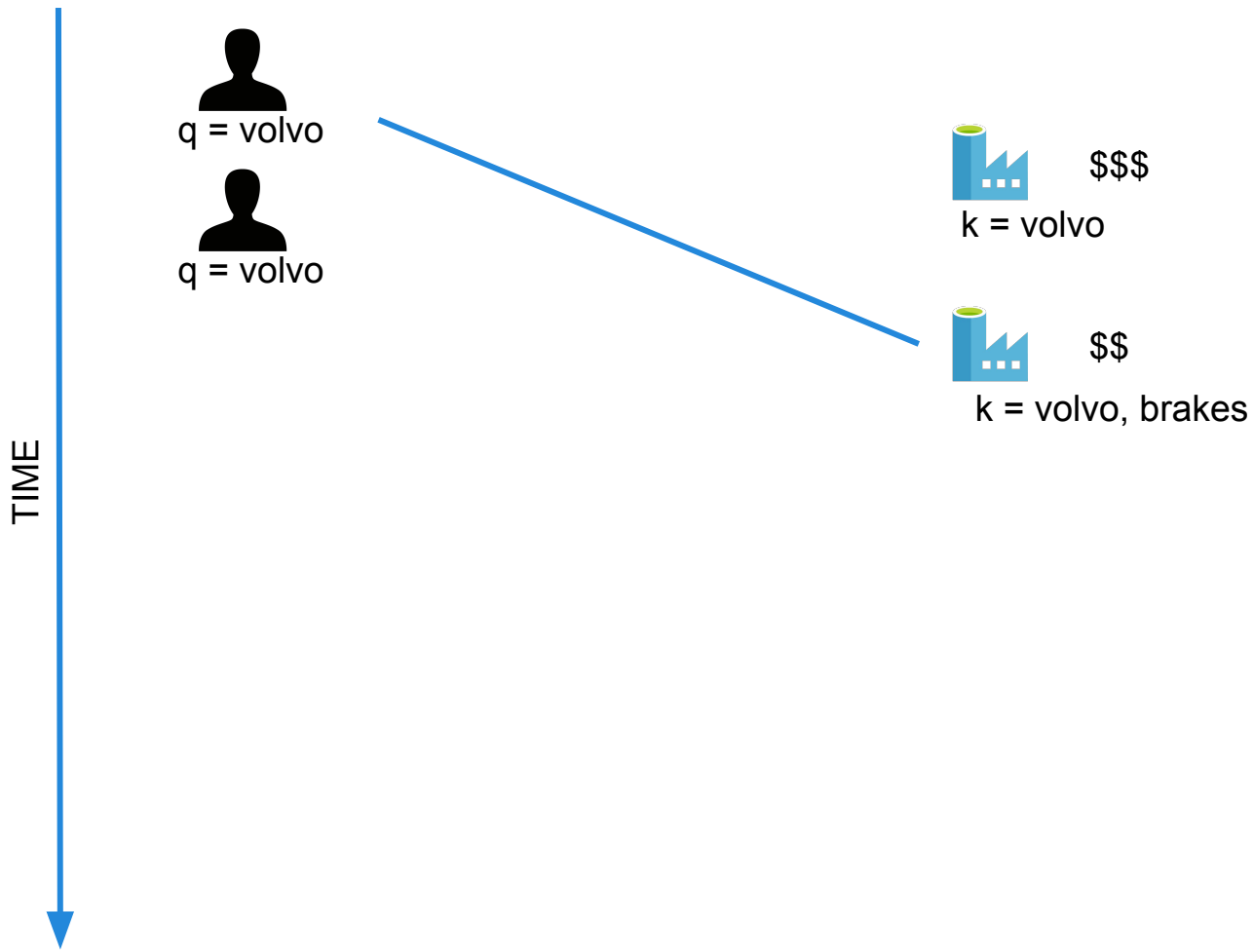
$$$

k = volvo, brakes

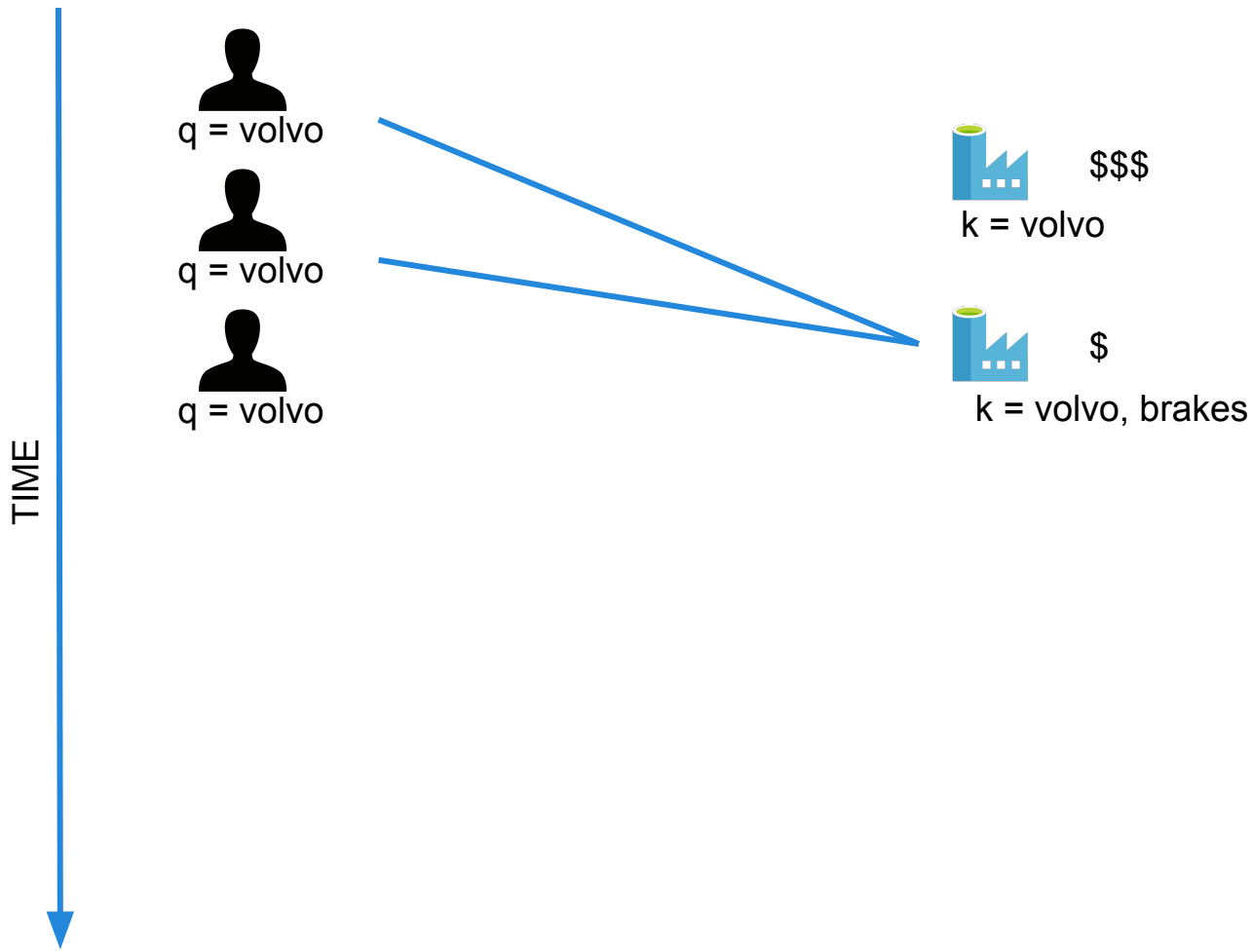
Adwords (simplified: k=1) model



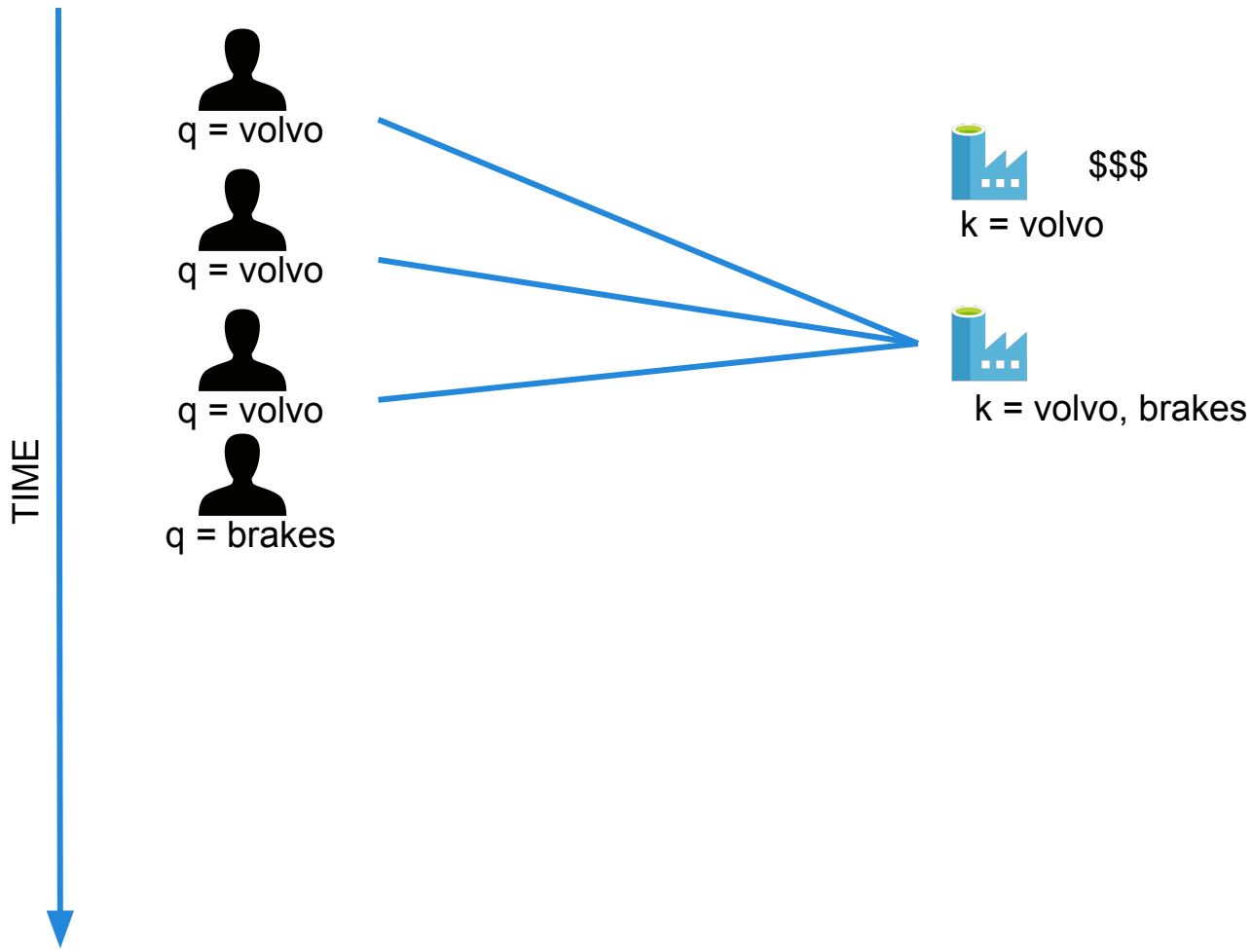
Adwords (simplified: k=1) model



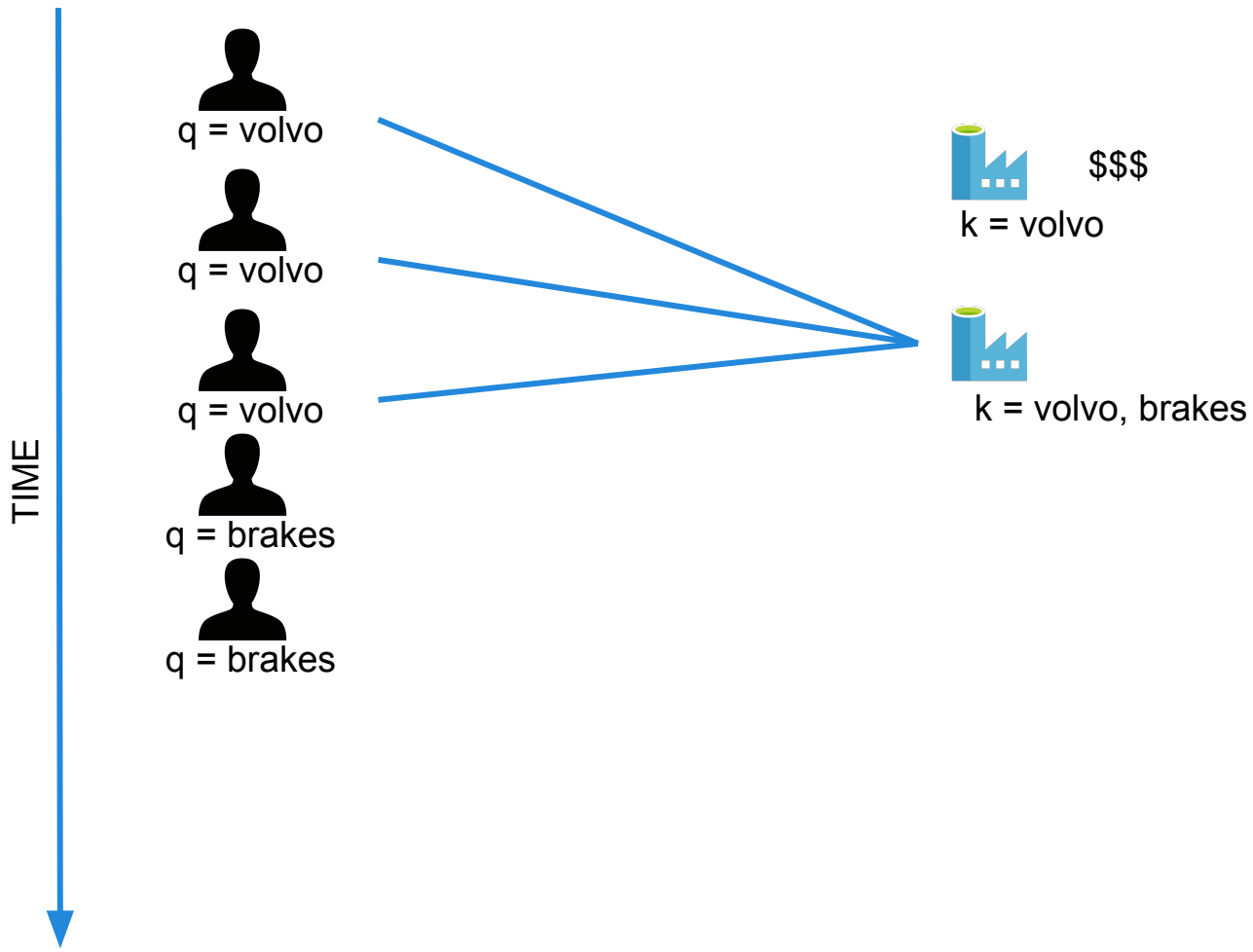
Adwords (simplified: k=1) model



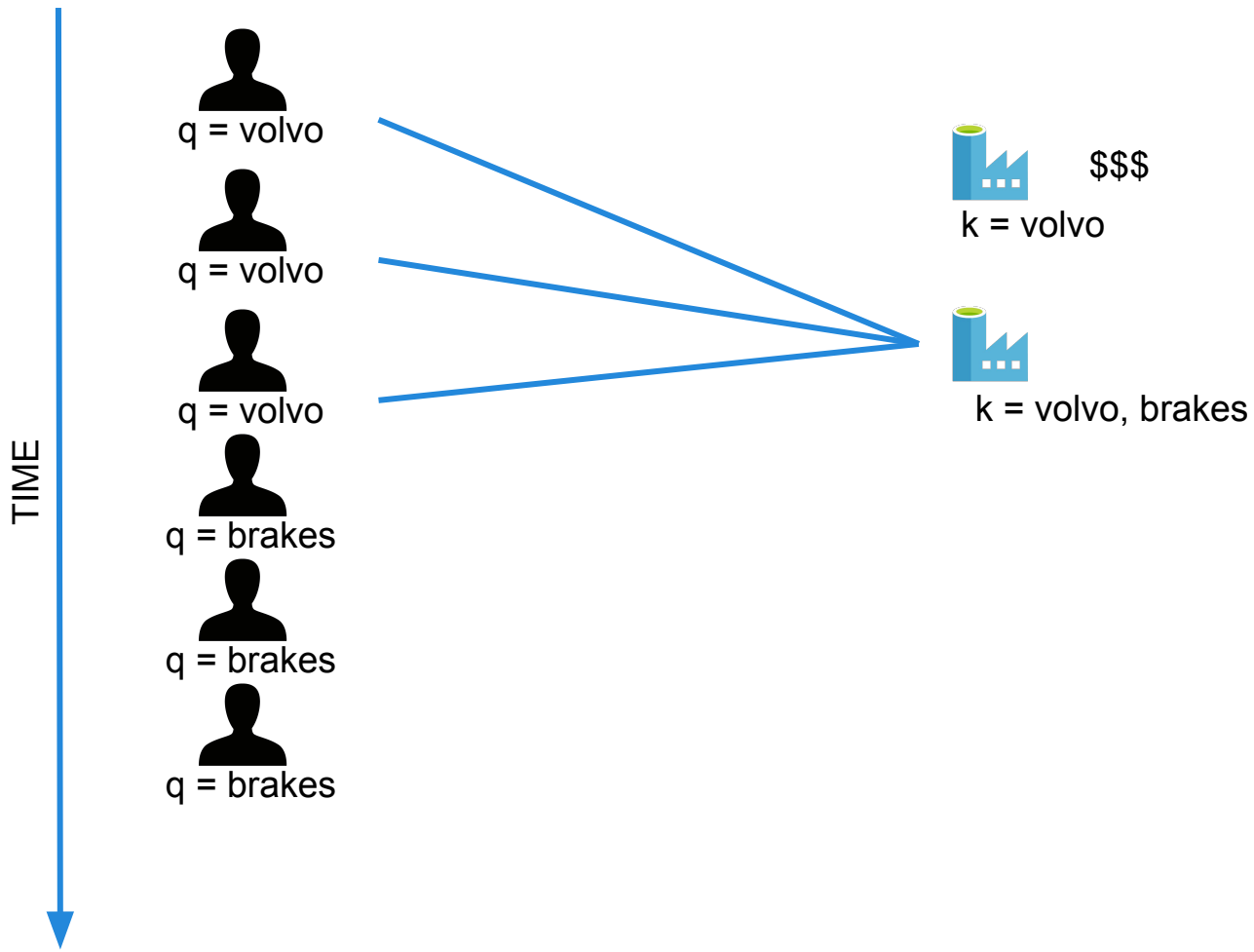
Adwords (simplified: k=1) model



Adwords (simplified: k=1) model

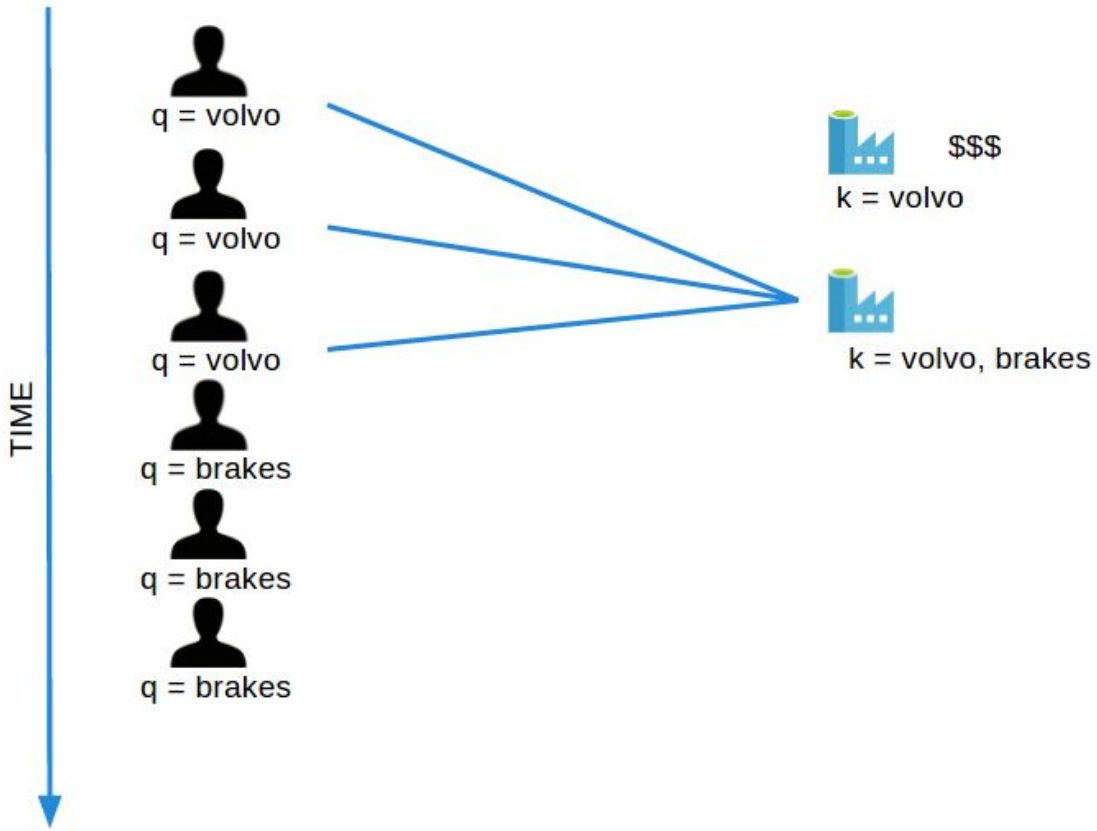


Adwords (simplified: k=1) model



Revenue

We could earn 6$ but we earned only 3$



Revenue

Better algorithm *M* ->
More matches ->
Higher Revenue

Competitiveness

- For input I, suppose greedy produces matching M_{greedy} while an optimal matching is M_{opt}

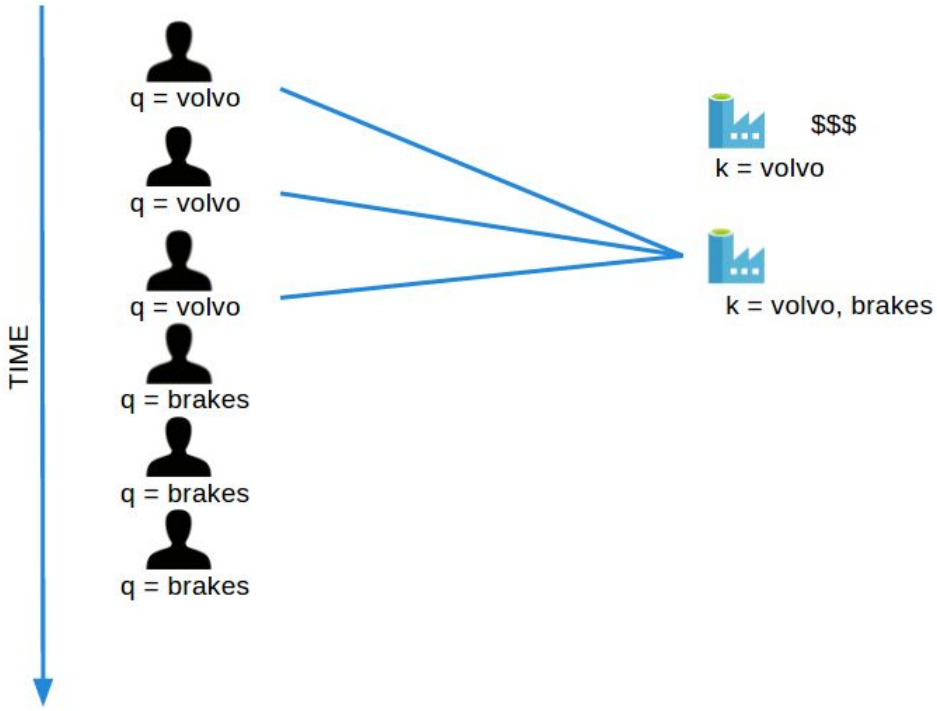
Competitive ratio =

$$min_{\text{all possible inputs } I} (|M_{greedy}|/|M_{opt}|)$$

(what is greedy's <u>worst</u> performance <u>over all possible</u> inputs I)

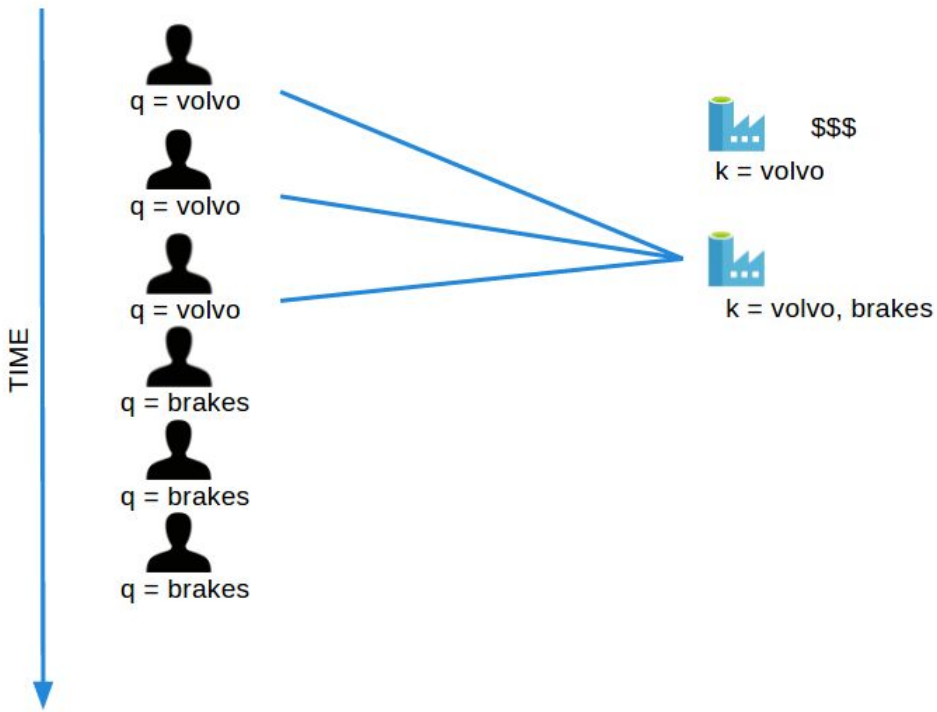
Competitiveness

For this data we matched 3/6 = 1/2

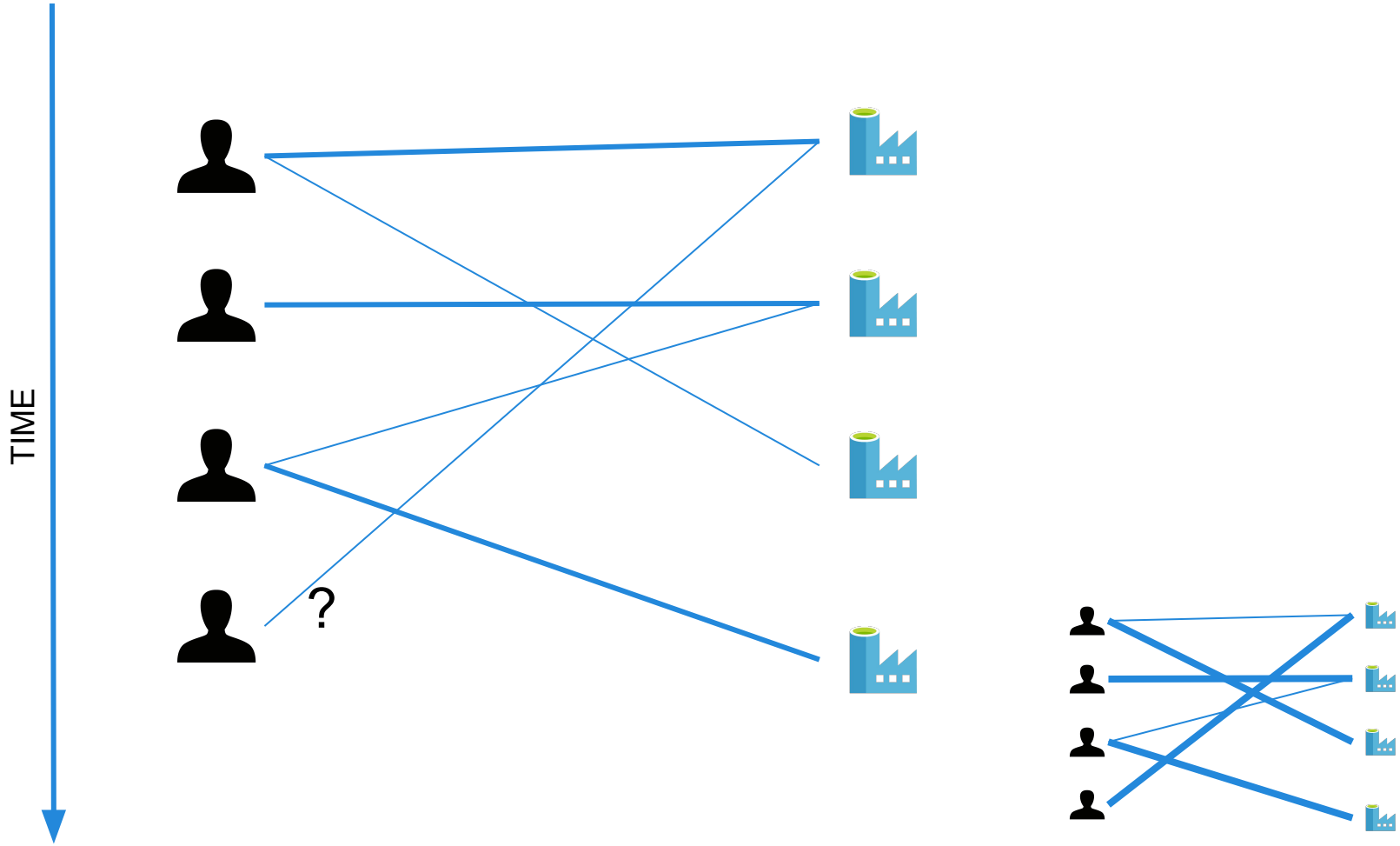


Competitiveness

Actually it's the worst scenario: (under our simplifications) competitive ratio for the greedy algorithm(s) is 1/2 (in general, not only here)



Online Greedy Matching

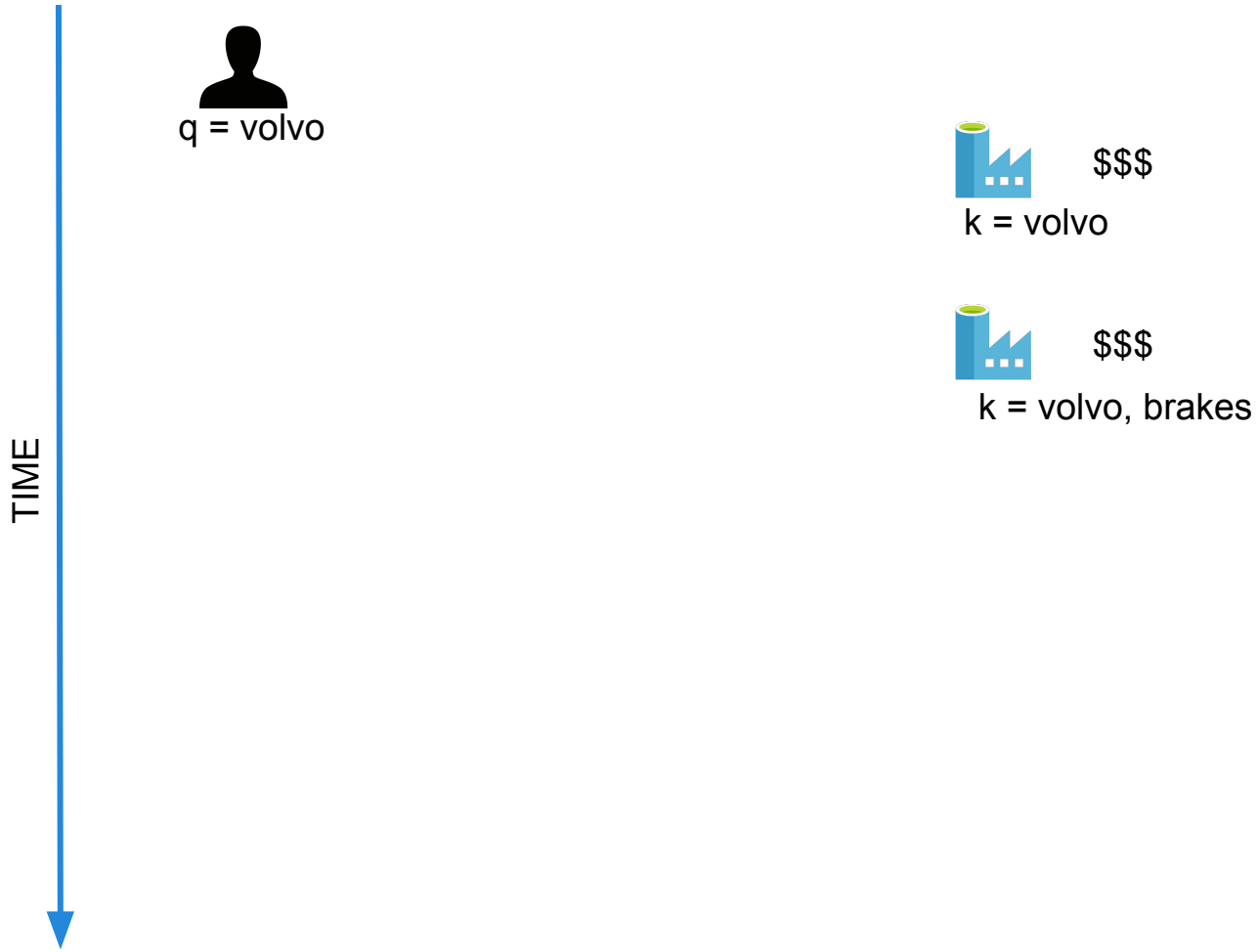


Balance algorithm

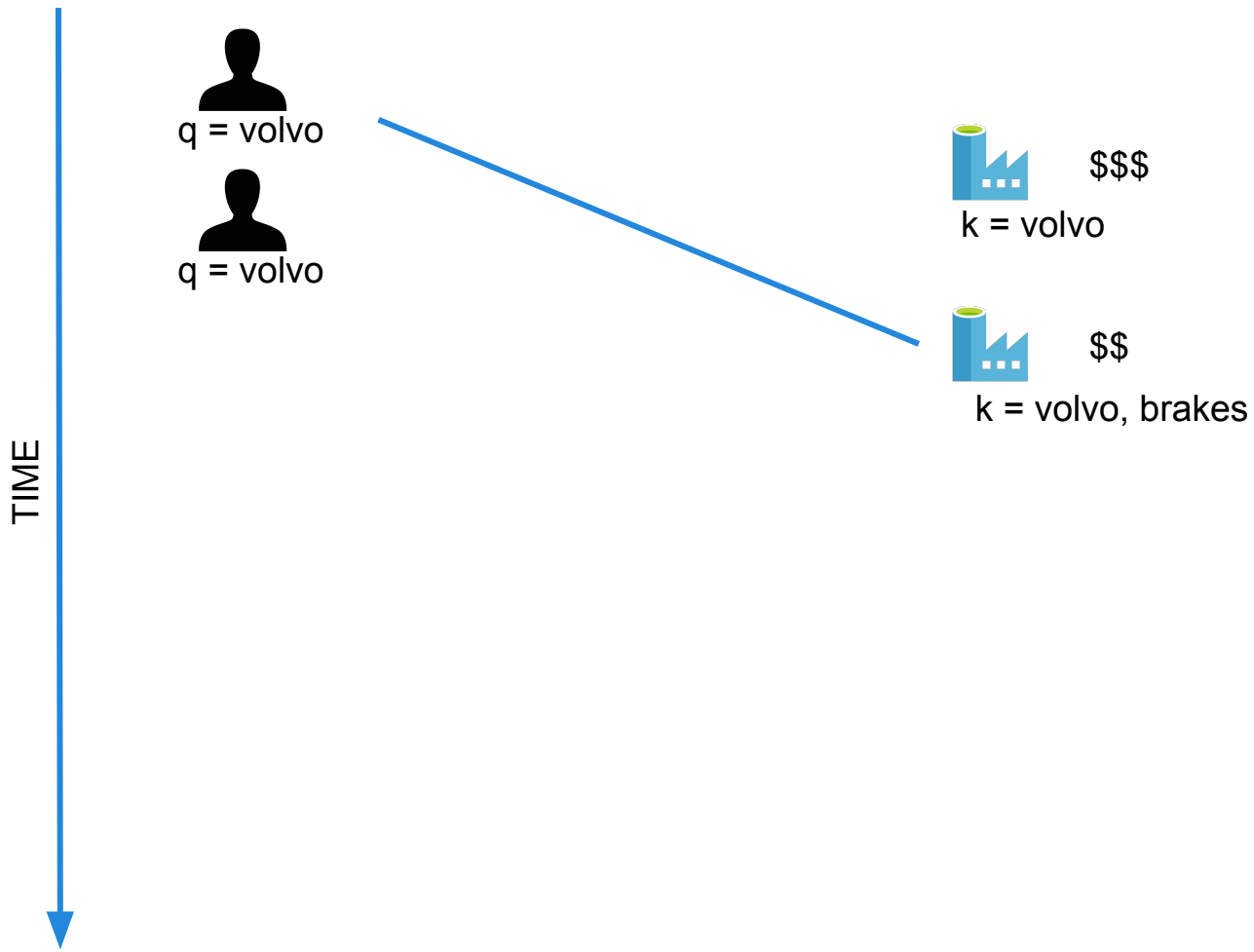
Balance algorithm

For each query, pick the advertiser with the largest unspent budget.

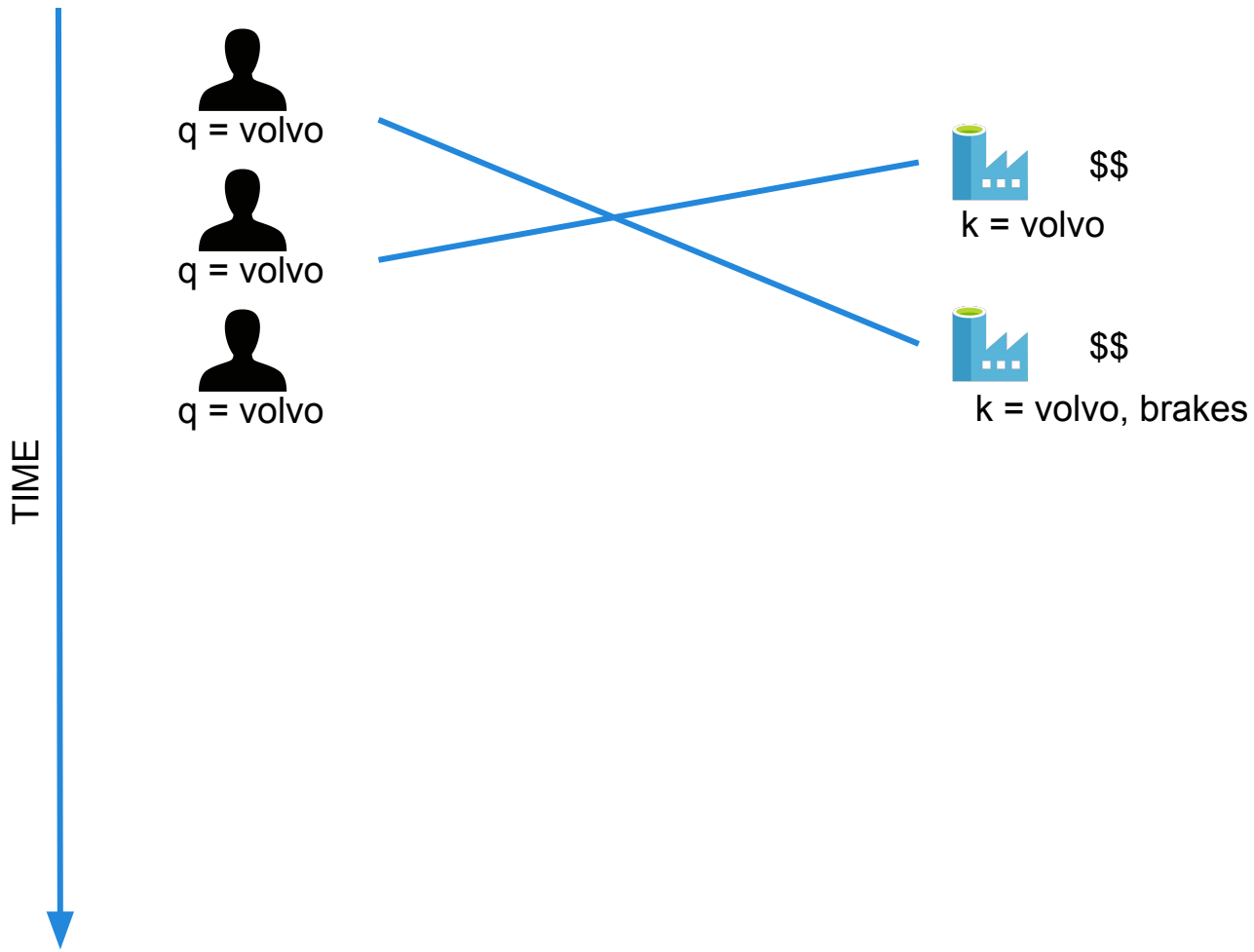
Adwords (simplified: k=1) model



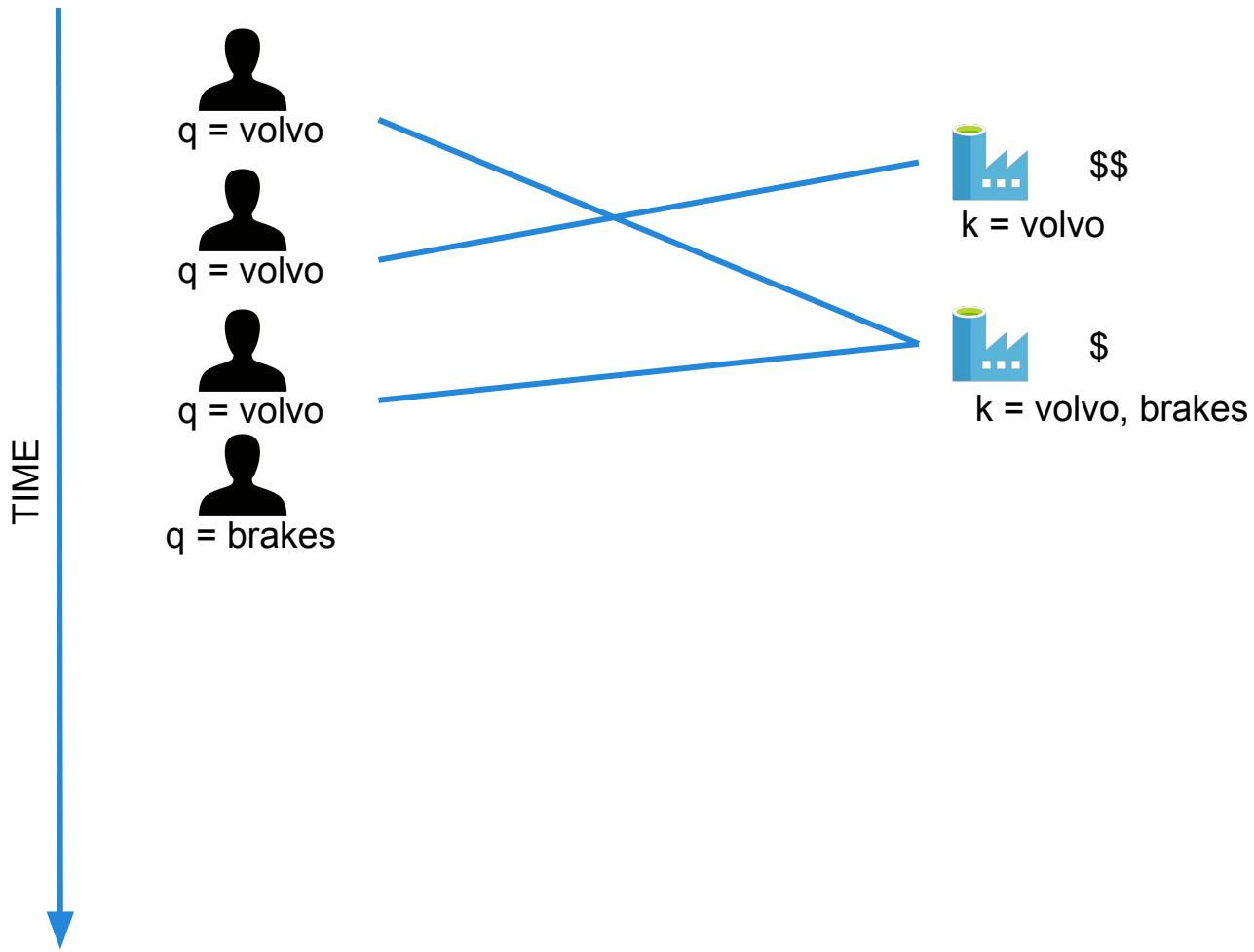
Adwords (simplified: k=1) model



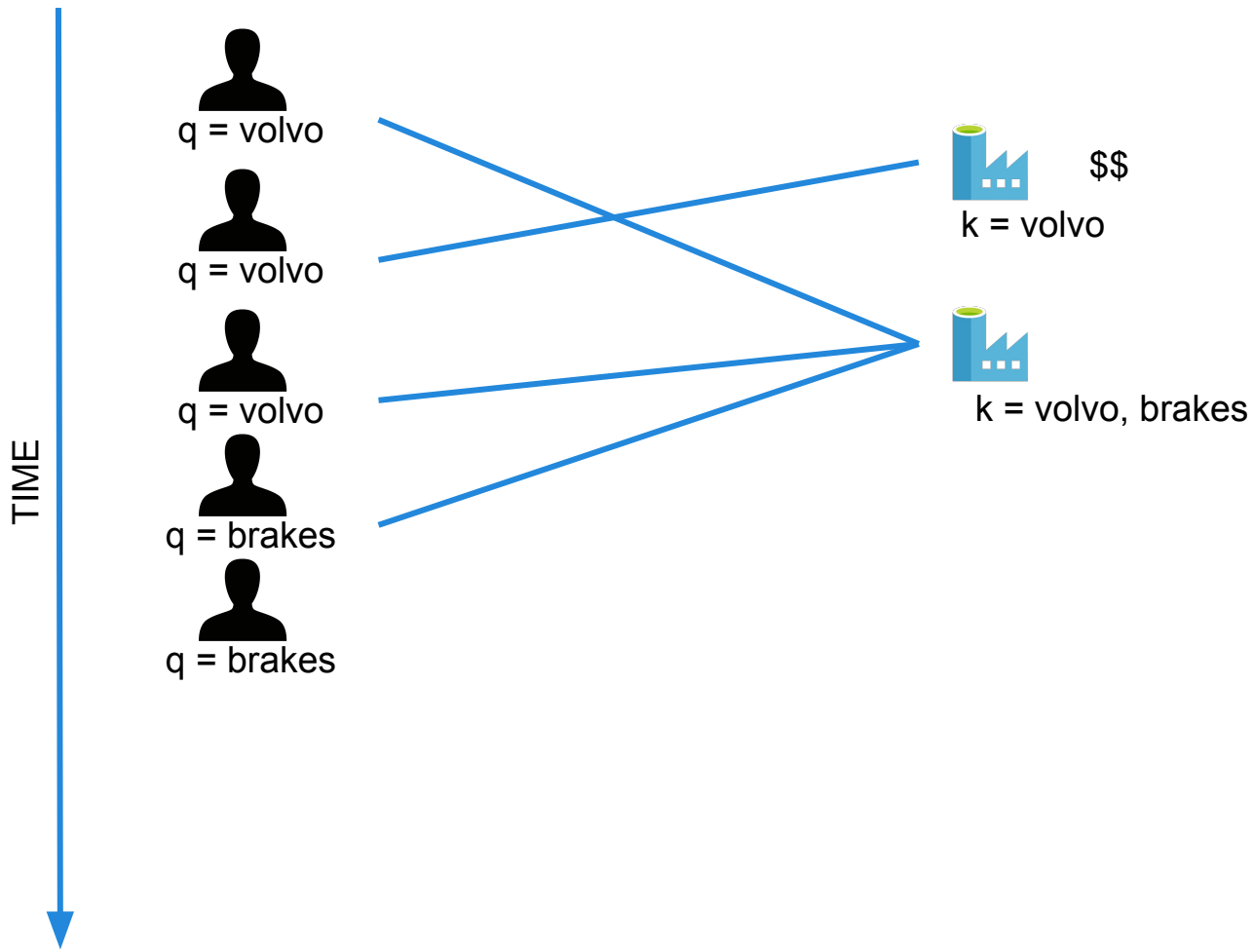
Adwords (simplified: k=1) model



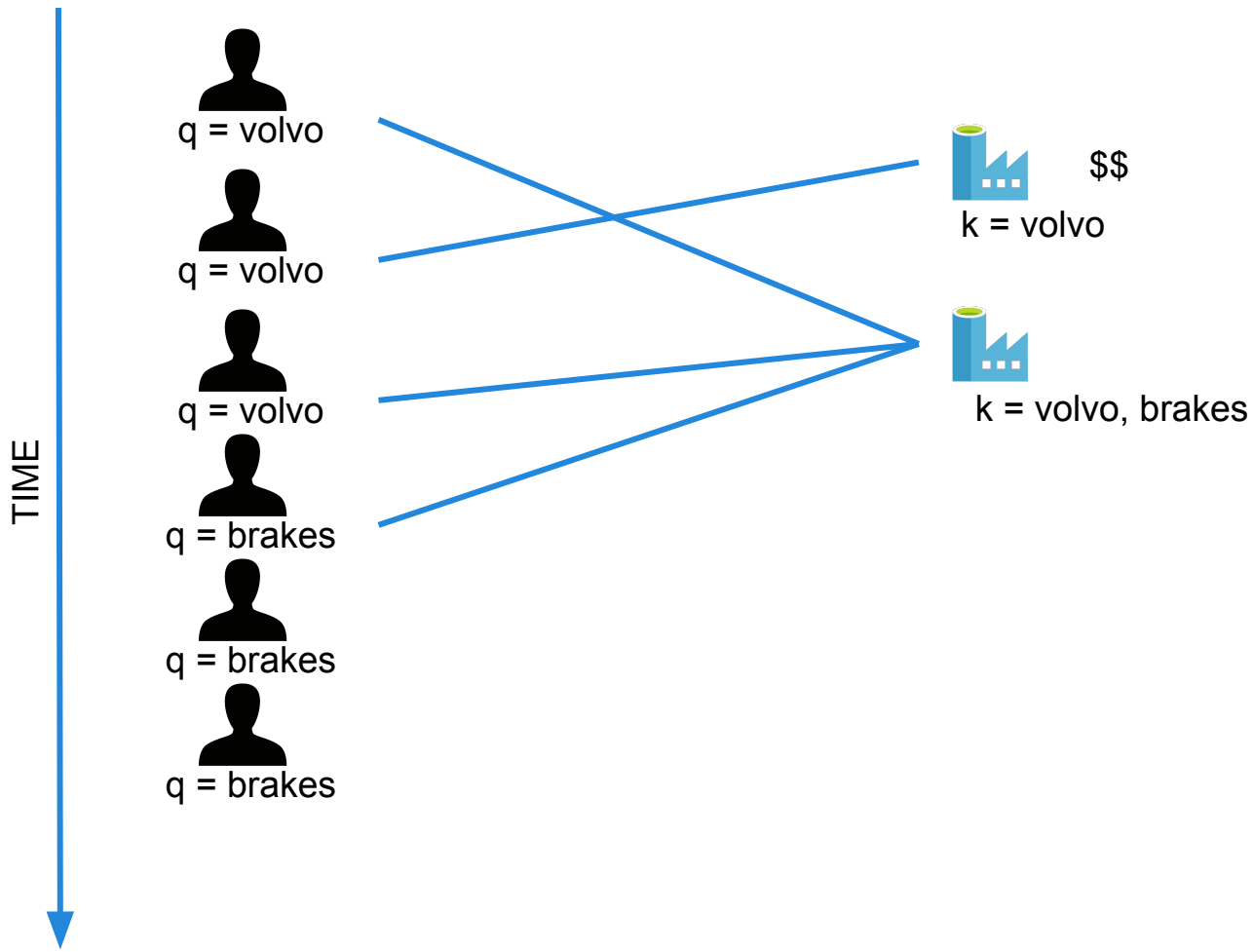
Adwords (simplified: k=1) model



Adwords (simplified: k=1) model

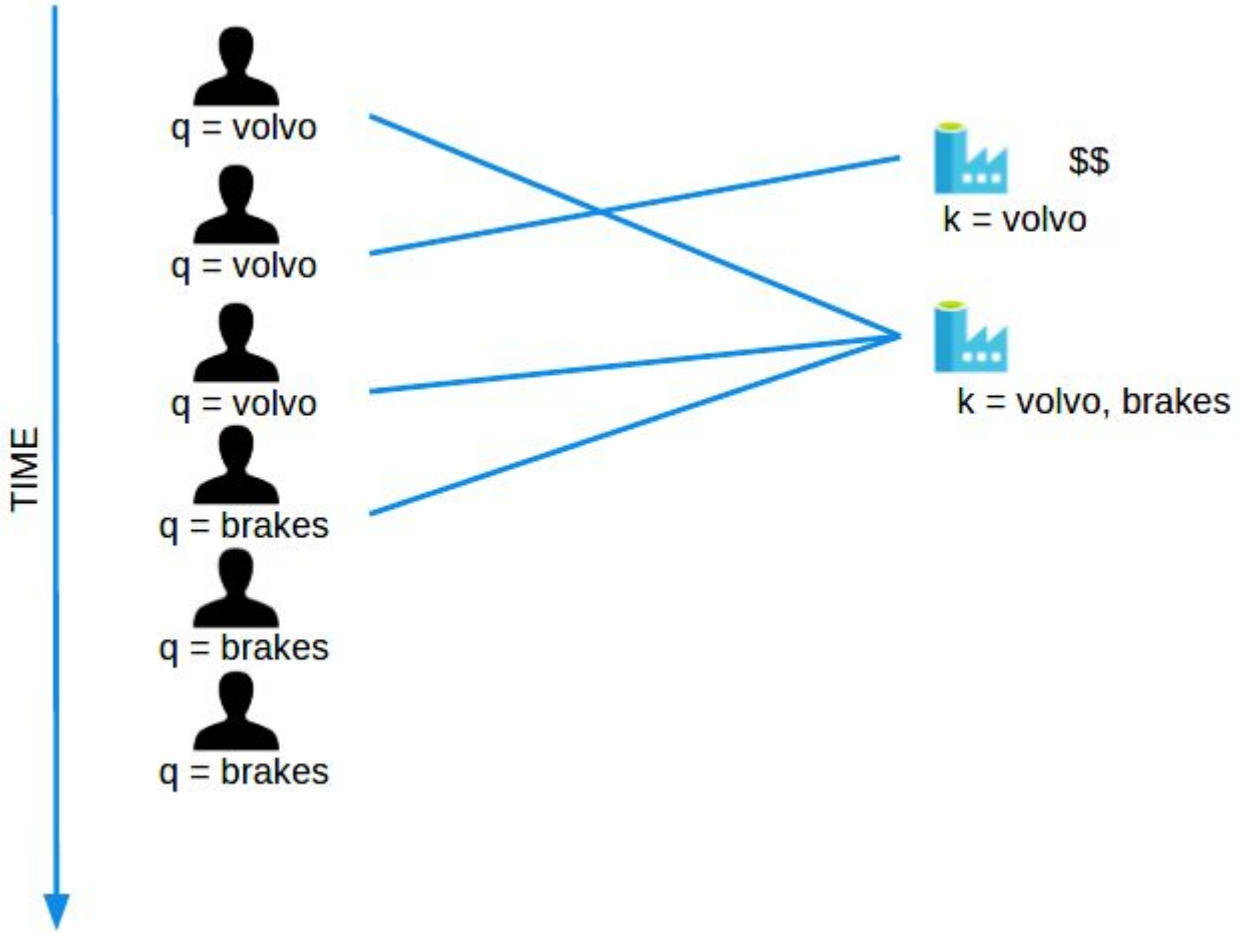


Adwords (simplified: k=1) model



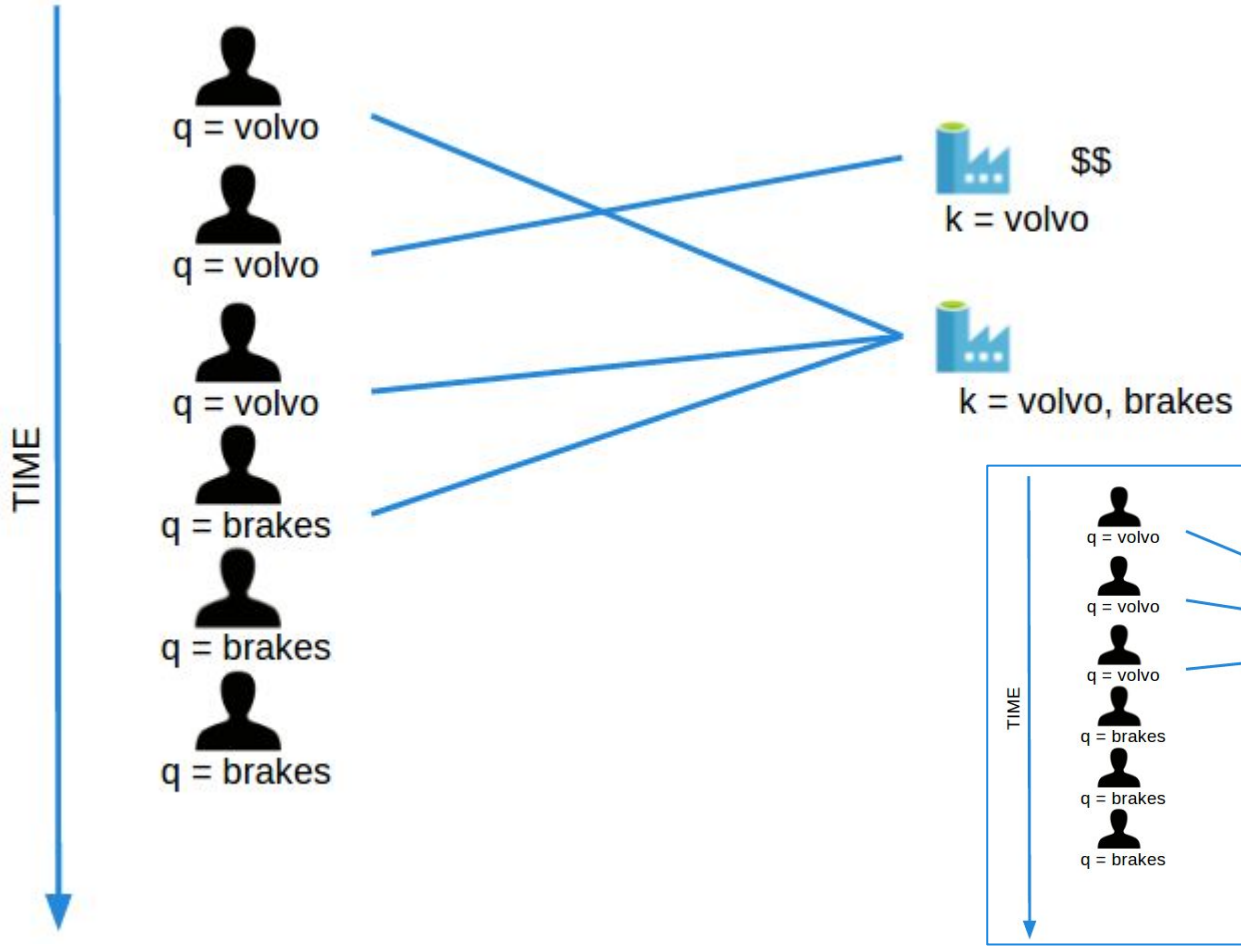
Competitiveness

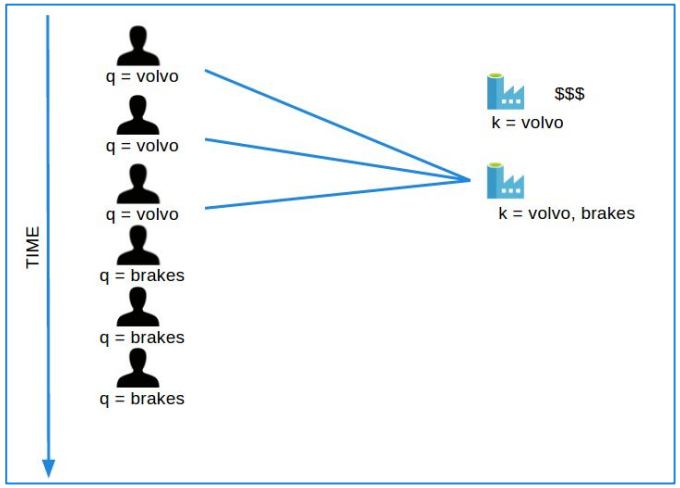
For this data we matched 4/6 = 2/3



Revenue

4$ instead of optimal 6$ but 1$ more than for greedy





Balance competitiveness

In the general case competitive ratio of BALANCE is **1–1/e** = approx. 0.63

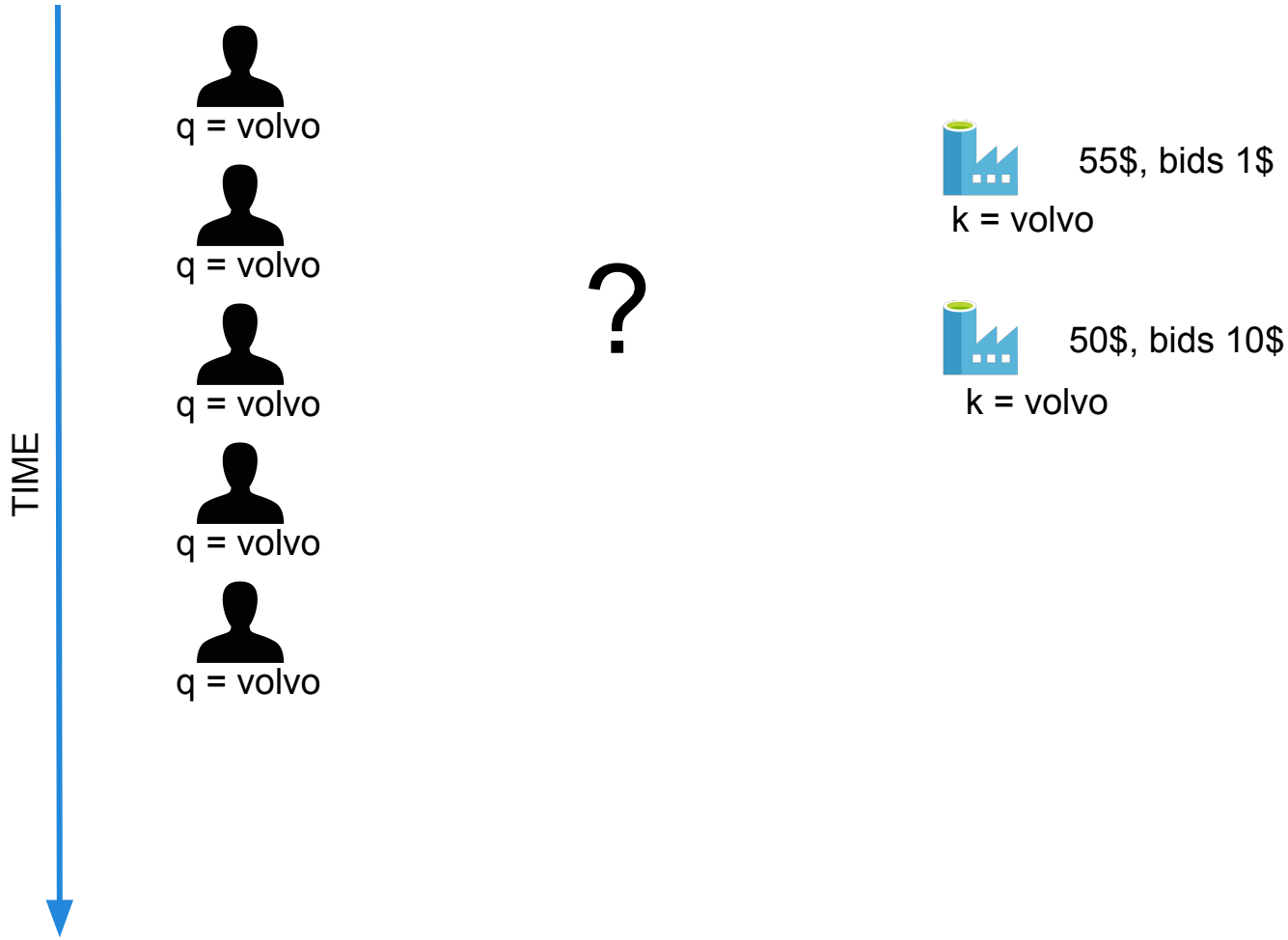
Interestingly, no online algorithm has a better competitive ratio!

Balance competitiveness

OK, how about the more general situation?:

bids can be arbitrary

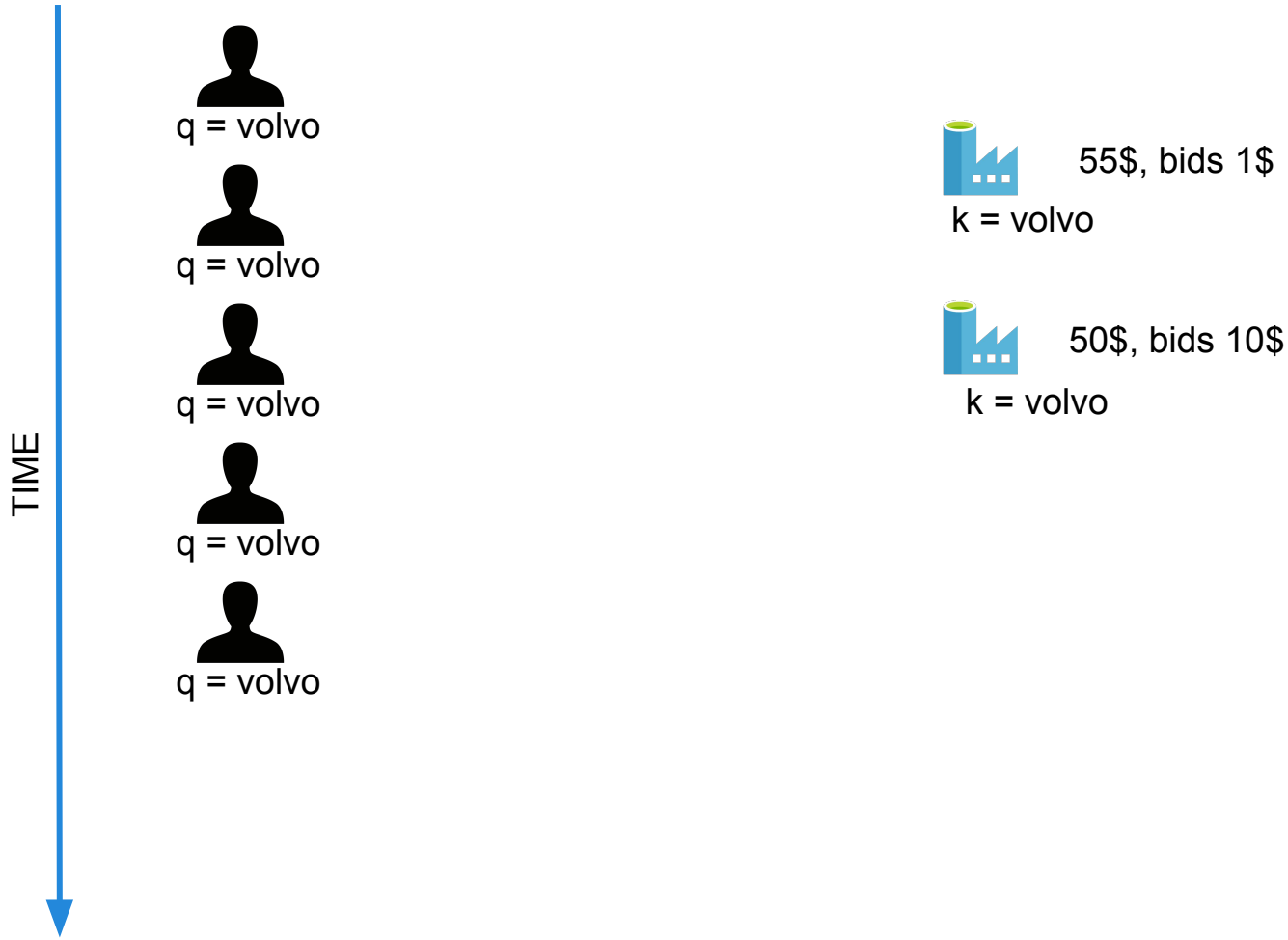
Balance Failure Example



Generalized Balance

- **Arbitrary bids:** consider query q, bidder i
  - Bid = x_i
  - Budget = b_i
  - Amount spent so far = m_i
  - Fraction of budget left over $f_i = 1-m_i/b_i$
  - Define $\psi_i(q) = x_i(1-e^{-f_i})$
- Allocate query q to bidder i with largest value of $\psi_i(q)$
- **Same competitive ratio (1-1/e)**

Generalized Balance (whiteboard)



Generalized Balance (whiteboard)

```
1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=0,    fi=6.321206
selecting 2

1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=10,   fi=5.506710
selecting 2

1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=20,   fi=4.511884
selecting 2

1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=30,   fi=3.296800
selecting 2

1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=40,   fi=1.812692
selecting 2

1. a=55 x=1,     m=0,    fi=0.632121
2. a=50 x=10,    m=50,   fi=0.000000
selecting 1

1. a=55 x=1,     m=1,    fi=0.625371
2. a=50 x=10,    m=50,   fi=0.000000
selecting 1
```

Real life

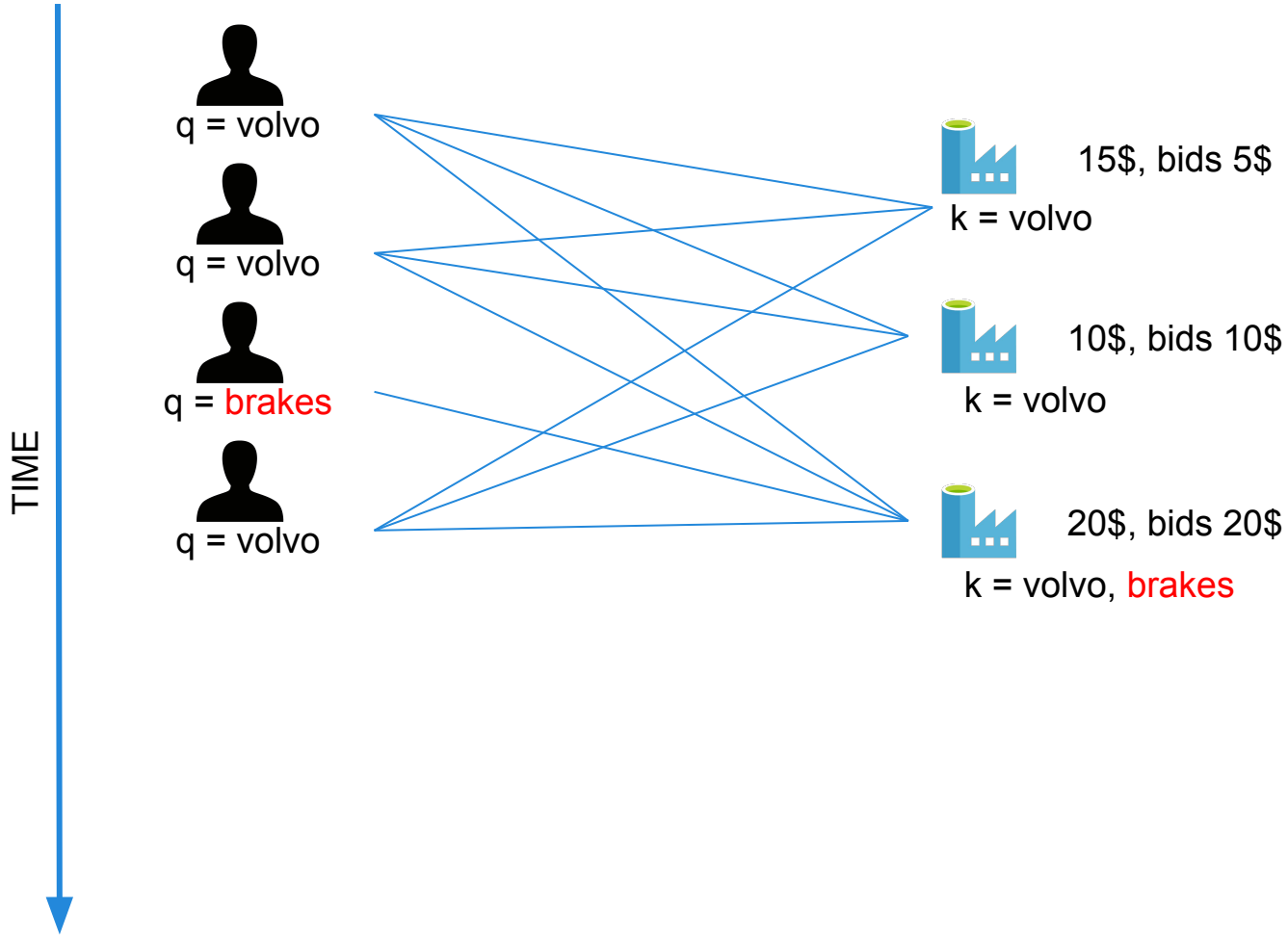
- Budgets differ
- Bids differ
- Bidders (companies) coming and leaving
- Matching is more complicated than keywords

Assignment

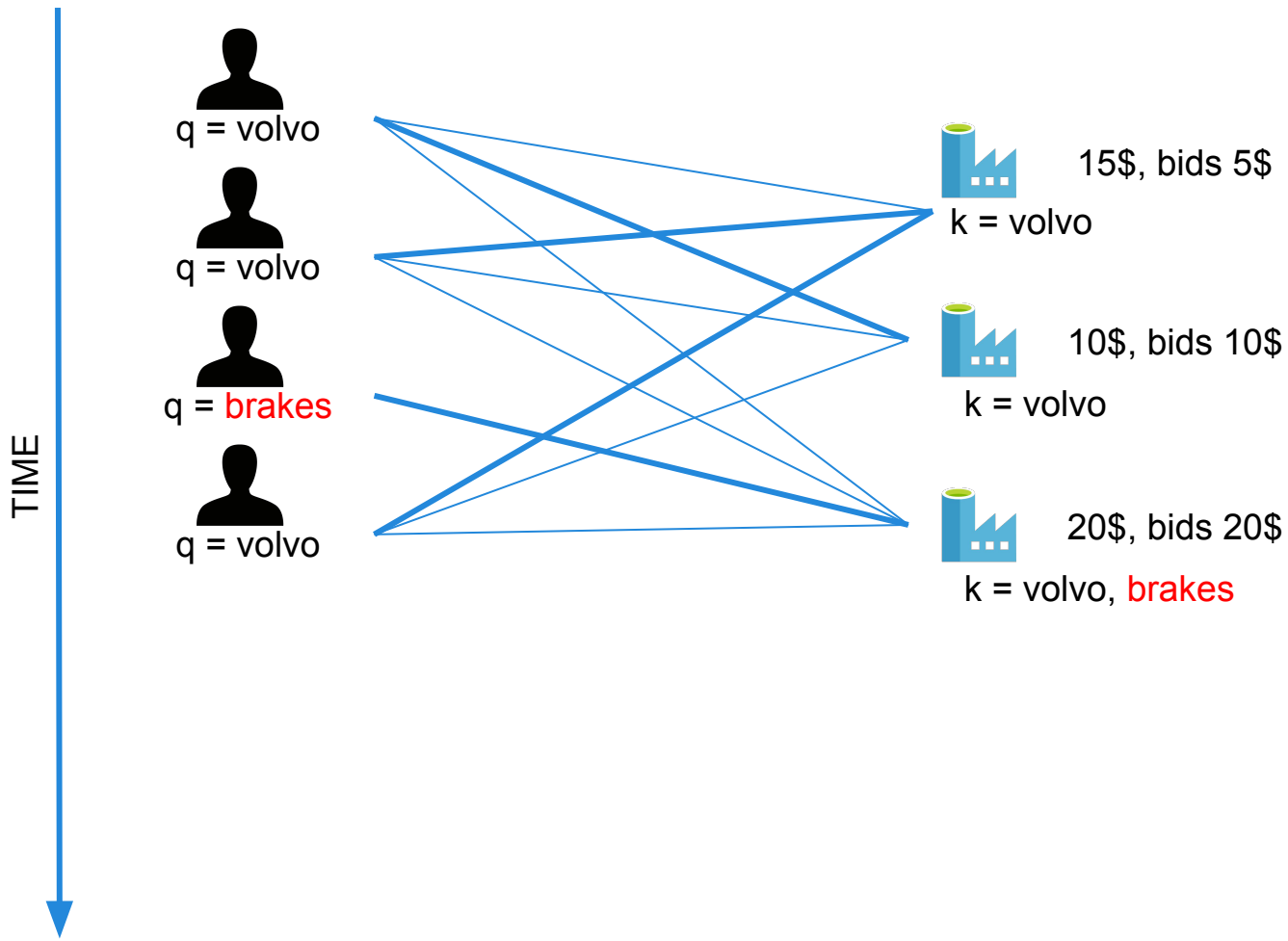
For the given below assignment problem:

- Find the optimal assignment (with max revenue)
- Find the Generalized Balance Algorithm solution.
- Compare matchings and revenues.

Problem


TIME
q = volvo
q = volvo
q = brakes
q = volvo
15$, bids 5$
k = volvo
10$, bids 10$
k = volvo
20$, bids 20$
k = volvo, brakes

Solution: sample perfect matching

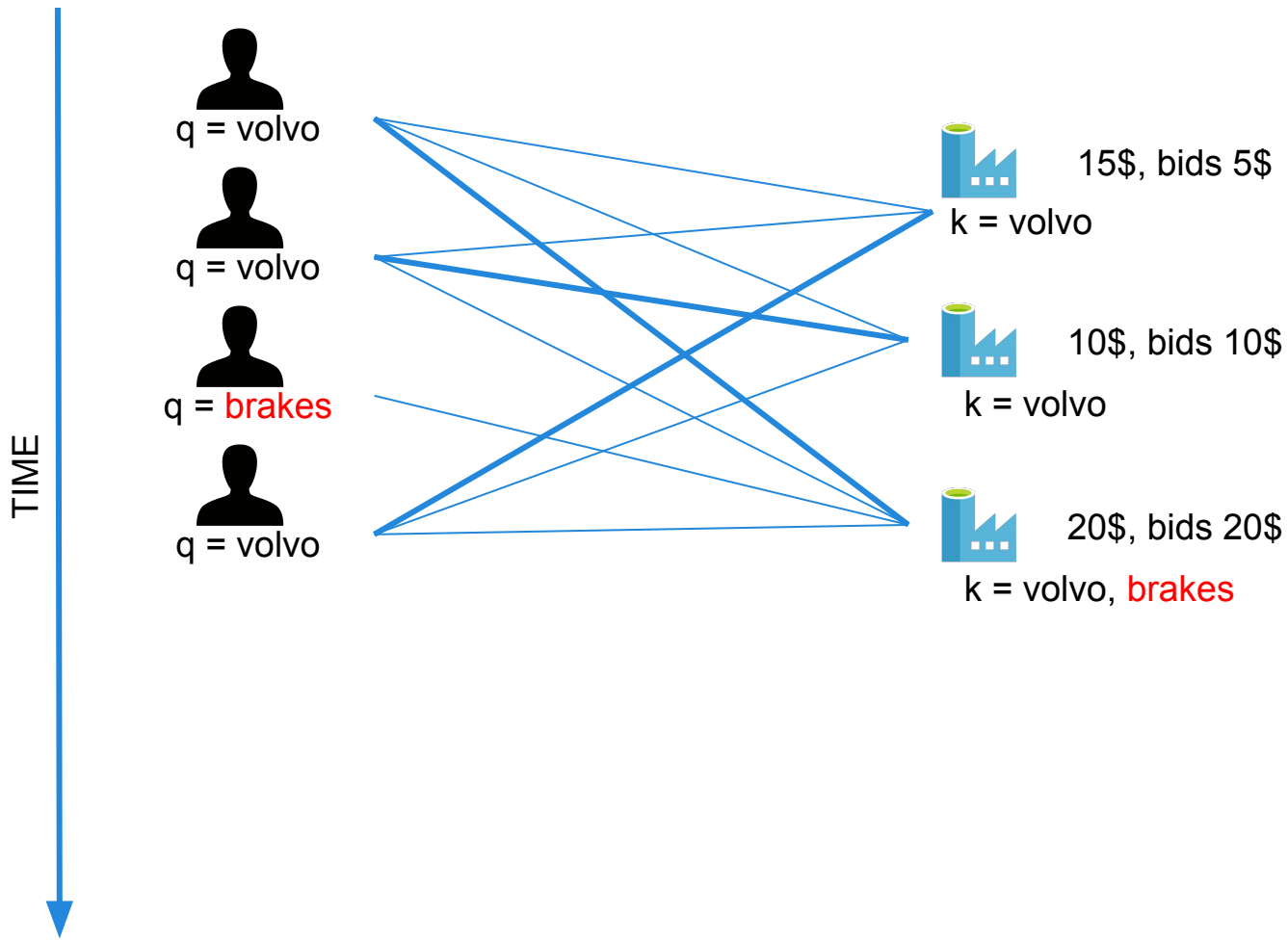


Solution: Generalized balance

i	m1	fi1	m2	fi2	m3	f3	Selected
1	0	3.16	0	6.32	0	12.64	3
2	0	3.16	0	6.32	20	0	2
3	-	-	-	-	20	0	NONE
4	0	3.16	10	0	20	0	1

User 3 has matching keywords only with the bidder no 3, but when he arrives the bidder has no more money to spend

Solution: Generalized balance



Solution

Revenues:

- Generalized balance: 35$
- Perfect matching: 40$

Matchings:

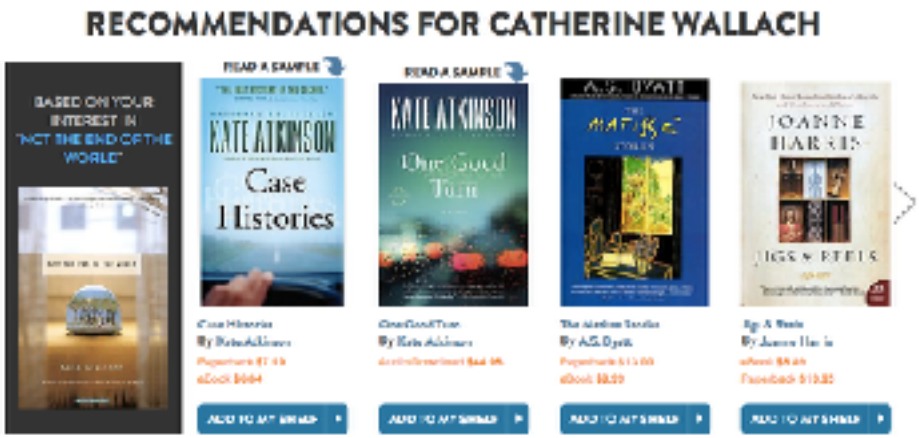
- Generalized balance: 3
- Perfect matching: 4

Recommender systems

Goal

- To know the motivation behind recommender systems (RS)
- To understand how RS works and know the principle behind RS methods/algorithms

Motivation



Problem: How to **recommend items to users** to make user, content partner, websites happy!

Recommendations

Search

Recommendations

Items

Products, web sites, blogs, news items

movielens
helping you find the right movies

LinkedIn

clicktorch
Intelligent Recommendation

NETFLIX

facebook

YouTube

amazon

P

Sugestio

last.fm

Spotify

Google news

ebay

From Scarcity to Abundance

- Shelf space scarce commodity for traditional retailers
  - Also: TV networks, movie theatres,…
- Web enables near-zero-cost dissemination of information about products
  - From scarcity to abundance
- More choice necessitates better filters
  - Recommendation engines
  - How Into Thin Air made Touching the Void a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

The Long Tail Phenomenon

Types of Recommendations

- Editorial and hand curated
  - List of favorites
  - Lists of “essential” items
- Simple aggregates
  - Top 10, Most Popular, Recent Uploads
- **Tailored to individual users**
  - Amazon, Netflix, Apple Music, Spotify, Facebook Ads…

Formal Model

- ***X*** = Set of **Customers**
- ***S*** = Set of **Items**

- **Utility function u: *X*** x ***S* –> *R***
  - ***R*** = Set of **ratings**
  - ***R*** totally ordered set
  - E.g., 0 - 5 stars. real number in [0, 1]

Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0,2	
Bob		0,5		0,3
Carol	0,2		1	
David				0,4

Key Problems

1. **Gathering "known" ratings for matrix**
   - How to collect the data in the utility matrix
2. **Extrapolate unknown ratings from the known ones**
   - Mainly interested in high unknown ratings
   - Not interested in knowing what you don't like but what you like
3. **Evaluating extrapolation methods**
   - How to measure success/performance of recommendation methods

(1) Gathering Ratings

- **Explicit**
  - Ask people to rate items
    - Doesn’t work well in practice – people can’t be bothered
  - Crowdsourcing: Pay people to label items
- **Implicit**
  - Learn ratings from user actions

    E.g., purchase implies high rating
  - What about low ratings?

(2) Extrapolating Utilities

- **Key problem**:
  - **Sparsity** of utility matrix U
    - Most people have not rated most items
  - **Cold start**:
    - New items - no ratings
    - New users - no history
- **Three approaches to recommender systems:**

  1) Content-based
  2) Collaborative
  3) Latent factor based

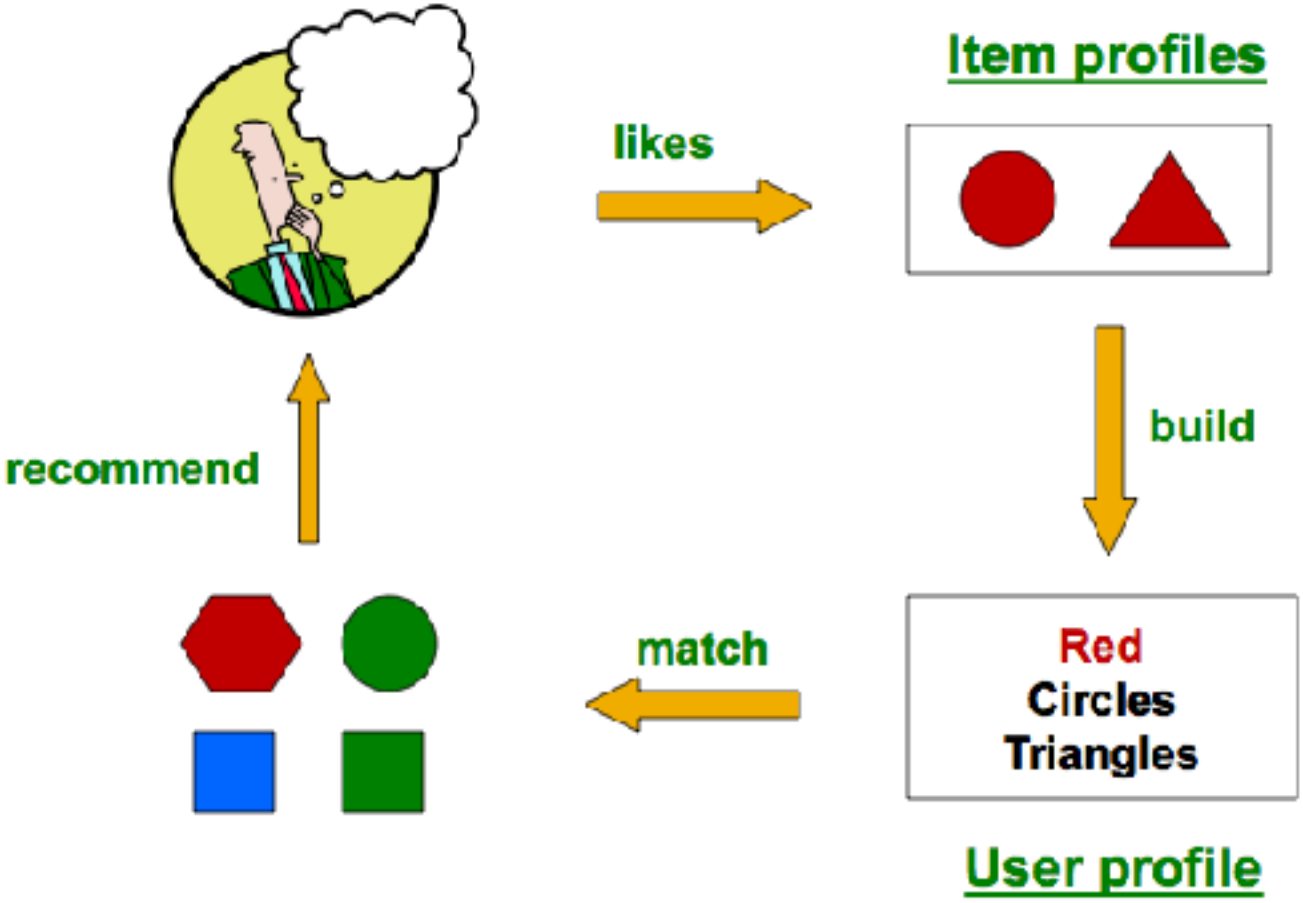
Content-based Recommendations

- **Main idea**:
  Recommend items to customer *X* similar to previous items rated highly by *X*

Example:

- **Movie recommendations**
  - Recommend movies with same actor(s), director, genre, ...
- **Websites, blogs, news**
  - Recommend other sites with “similar” content

Plan of Action



Item Profiles

- For each item, create an item profile
- Profile: a set (vector) of features
  - **Movies:** author, title, actor, director,...
  - **Text:** Set of “important” words in document
- How to pick important features?
  - Usual heuristic from text mining is TF-IDF (Term frequency * Inverse Doc Frequency)
  - Term ... Feature
  - Document ... Item

Item Features

- Description of items in terms of attributes
  - E.g.: Type, Director, Actors
- Description via keywords
- Possibility to look at content itself, like the text

TF-IDF

f_{ij} = frequency of term (feature) i in doc (item) j

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" documents

n_i = number of docs that mention term i

N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

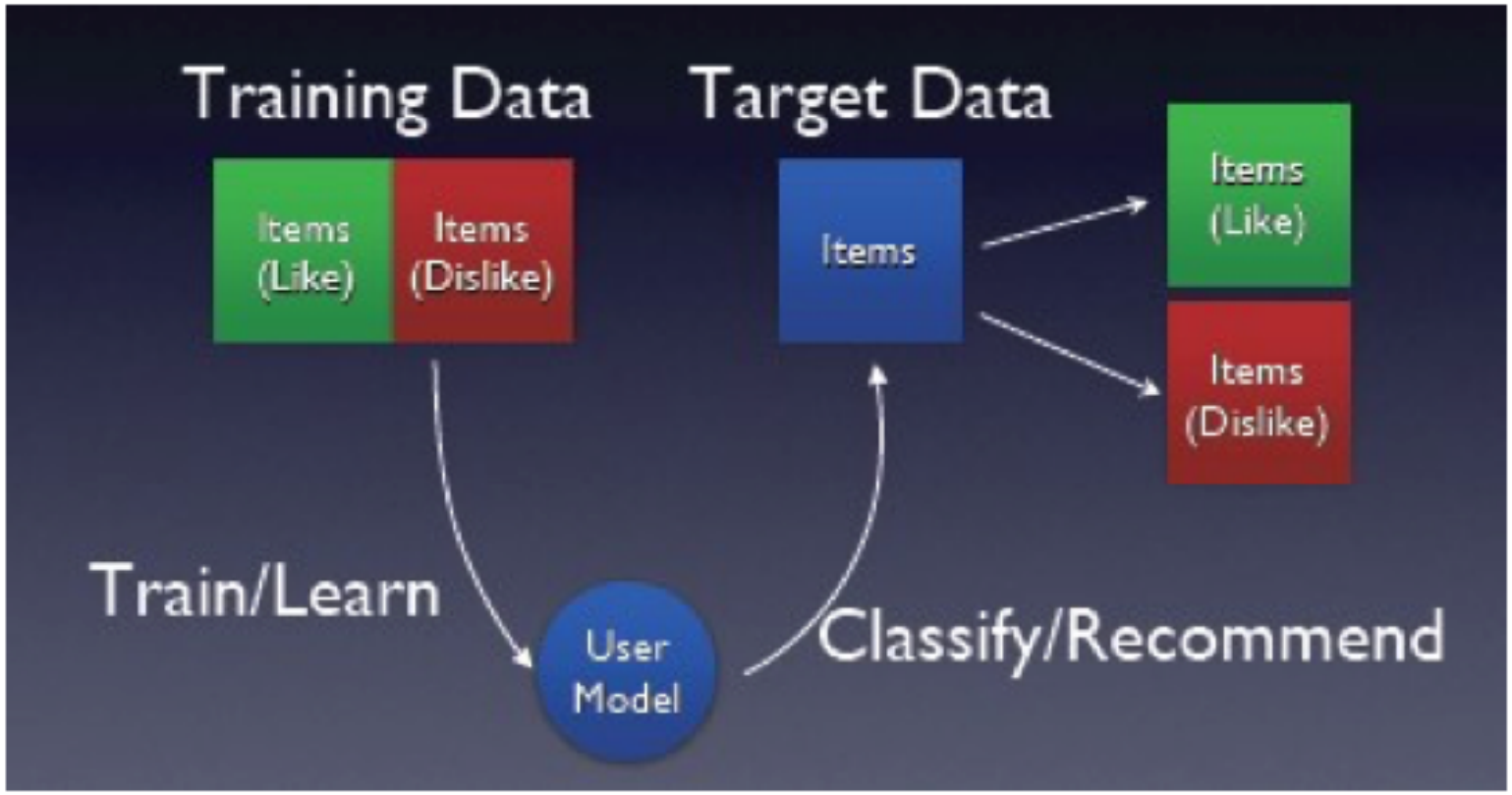
Doc profile = set of words with highest **TF-IDF** scores, together with their scores

User Profiles and Prediction

- User profile possibilities:
  - Weighted average of rated item profiles
  - Variation: weight by difference from average rating for item
  - ...
- Prediction heuristic
  - Given user profile *x* and item profile *i*, estimate

$$u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$$

Learning a User Model



Pros: Content-based approach

- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-bases Approach

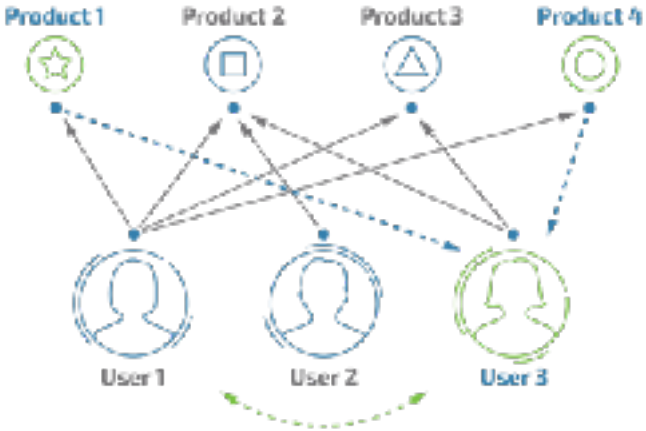
- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Recommendations for new users**
  - How to build a user profile?
- **–: Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users**

Collaborative Filtering

How can we exploit quality judgments of other users to provide relevant recommendation?

Collaborative Filtering

- Maintain a database of many **users' ratings of a variety of items**.
- For a given user, **find other similar users whose ratings strongly correlate** with the current user.
- **Recommend items rated highly by these similar users**, but not rated by the current user.
- Almost all existing commercial recommenders use this approach (e.g. Amazon).



Collaborative Filtering

- **Consider user *x***
- Find set *N* of other users whose ratings are "similar" to *x*'s ratings
- Estimate *x*'s ratings based on ratings of users in *N*

Find "similar users"

r_x = [*, _, _, *, ***]
r_y = [*, _, **, **, _]

- Let $\boldsymbol{r_x}$ be the vector of user *x*'s ratings
- **Jaccard similarity measure**
  - **Problem**: Ignores the value of the rating

r_x, r_y as sets:
r_x = {1, 4, 5}
r_y = {1, 3, 4}

- **Cosine similarity measure**

$$sim(x,y) = cos(r_x, r_y) = \frac{r_x \cdot r_y}{\|r_x\| \cdot \|r_y\|}$$

r_x, r_y as points:
r_x = {1, 0, 0, 1, 3}
r_y = {1, 0, 2, 2, 0}

  - **Problem:** Treats missing ratings as "negative"
- **Pearson correlation coefficient**
  - S_{xy} = Set of items rated by both users *x* and *y*

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

$\mathbf{r_x}$, $\mathbf{r_y}$ … avg. rating of **x**, **y**

Similarity Metric

$$sim(x,y) = \frac{\sum_i r_{xi} \cdot r_{yi}}{\sqrt{\sum_i r_{xi}^2} \cdot \sqrt{\sum_i r_{yi}^2}}$$

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- Intuitively we want: sim(A, B) > sim(A, C)
- **Jaccard similarity**: 1/5 < 2/4
- **Cosine similarity**: 0.380 > 0.322
  - Considers missing ratings as "negative"
  - Solution: subtract the (row) mean

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	7/3		
B	1/3	1/3	−2/3				
C				−5/3	1/3	4/3	
D		0					0

sim A,B vs. A,C:
0.092 **>** -0.559

Notice cosine sim. is correlation when data is centered at 0

Rating Predictions

- **From similarity metric to recommendations:**
  - Let r_x: vector of user x's ratings
  - Let N : set of k users most similar to x who have rated item i
- Prediction for item s of user x:
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$
  - Other options?
- Many other tricks possible…

Shorthand:
$\boldsymbol{s_{xy} = sim(x, y)}$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Item-Item Collaborative Filtering

- So far: **User-user collaborative filtering**
- Another view: **Item-item**
  - For item *i*, find other similar items
  - Estimate rating for item *i* based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij}… similarity of items ***i*** and ***j***
r_{xj}…rating of user ***u*** on item ***j***
N(i;x)… set items rated by ***x*** similar to ***i***

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- unknown rating

- rating between 1 to 5

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie **1** by user **5**

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to movie **1**, **rated by user 5**

Here we use Pearson correlation as similarity:

1) Subtract mean rating m_i from each movie i

$m_1 = (1+3+5+5+4)/5 = 3.6$

row 1: *[-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]*

2) Compute cosine similarities between rows

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights:

$s_{1,3}=0.41$, $s_{1,6}=0.59$

Item-Item CF (|N|=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3		?	2			4	

Predict by taking weighted average:

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

$r_{1.5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

CF: Common Practice

Before:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

- Define similarity s_{ij} of items i and j
- Select k nearest neighbors $N(i; x)$
  - Items most similar to i, that were rated by x
- Estimate rating r_{xi} as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$\boldsymbol{b_{xi} = \mu + b_x + b_i}$$

μ = overall mean movie rating

b_x = rating deviation of user x

= (avg. rating of user x) – μ

b_i = rating deviation of movie i

Item-Item vs. User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- In practice: item-item often works better than user-user
- Why? Items are simpler, users have multiple tastes

Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items

Pros/Cons of Collaborative Filtering (2)

- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Cold Start

- **New item problem**:
  - small number of users that rated an item, accurate prediction for this item cannot be generated.
- **New user problem**:
  - small number of items rated by a user, it is unlikely that there could be an overlap of items rated by this user and active users. User− to−user similarity cannot be reliably computed.
- **New community problem**:
  - Without sufficient ratings, it's hard to differentiate value by personalized CF recommendations.
- Clear **reward systems** necessary to convince users to vote or rate items.

Possible solution to cold start

- **As the solution for new user problem:**
  - Displaying non−personalized recommendation until the user has rated enough
  - Asking the user to describe their taste in aggregate
  - Asking the user for demographic information and using ratings of other users with similar demographics as recommendations
- **As the solution for new item problem:**
  - Recommending items through non−CF techniques content analysis or metadata
  - Randomly selecting items with few or no ratings and asking user to rate those items.
- **As the solution for new community problem:**
  - Provide ratings incentives to a small “bootstrap” subset of the community, before inviting the entire community.

Data Sparsity & Ratings scarcity

- The ratings matrix is sparse and only a small fraction of all possible user item entries is known.
- Many CF algorithms designed specifically for data sets with many more users than items (e.g., the MovieLens data set has 65,000 users and 5,000 movies).
- CF may be inappropriate in a domain where there are many more items than users.
- Implicit vs. explicit ratings

Evaluation of Collaborative Filtering

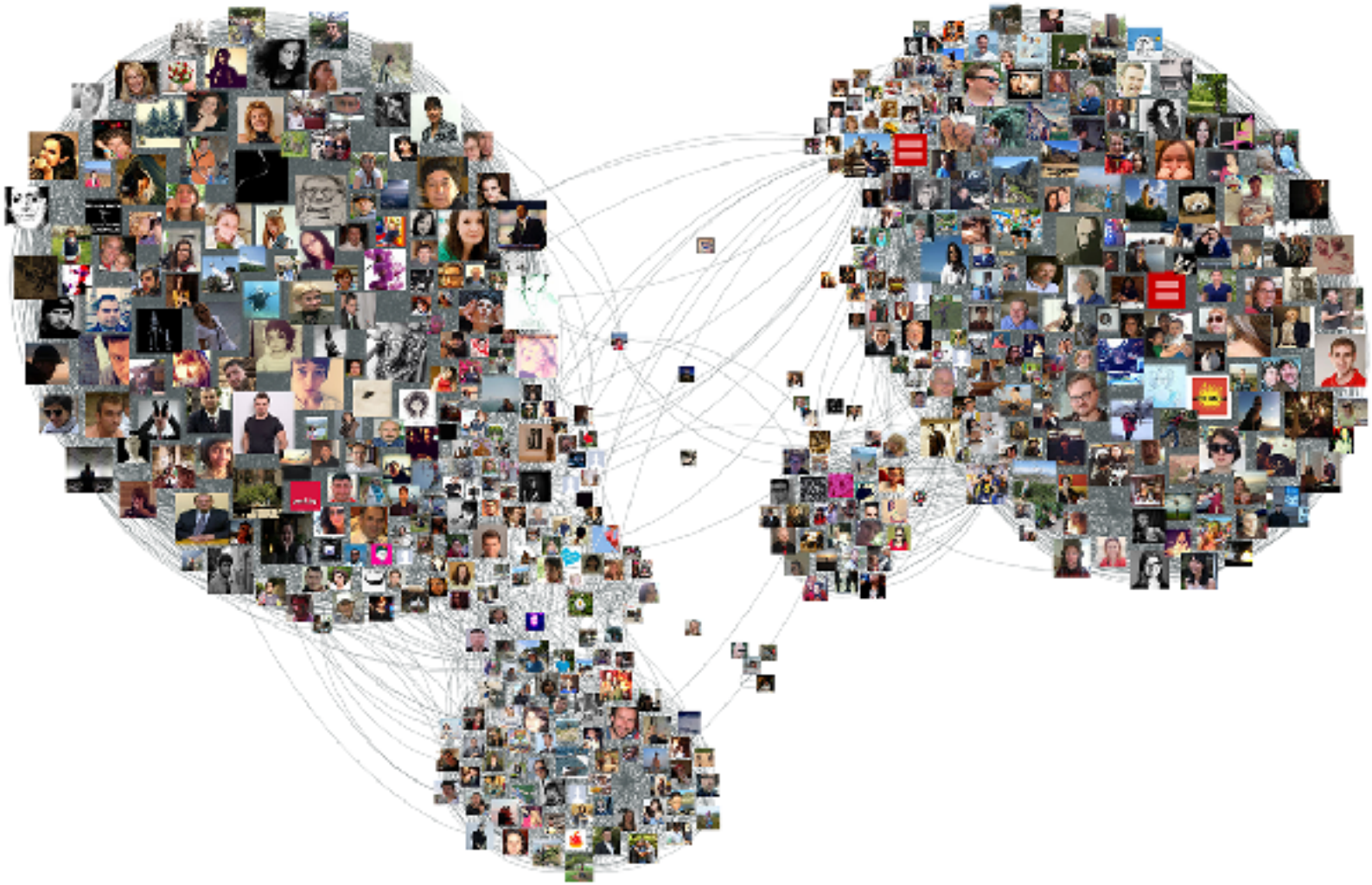
- To determine the quality of the predictions and recommendations
  - **Accuracy**
    - Rating accuracy : error between the predicted ratings and the true ratings. **Mean Absolute Error (MAE)** = average absolute difference between the predicted ratings and the actual rating given by a user
  - **Precision**
  - **Rank accuracy**: half−life utility.
  - **Novelty / Serendipity** (Karypis, 2001)
  - **Coverage** (Sarwar, et. al., 2000)
  - **Learning Rate** (Schein, et. al., 2001)
  - **Confidence** (Herlocker,2000)
  - **User Satisfaction** (Swearingen & Sinha, 2001; Dahlen, B. J., 1998)
  - Site Performance

Hybrid Methods

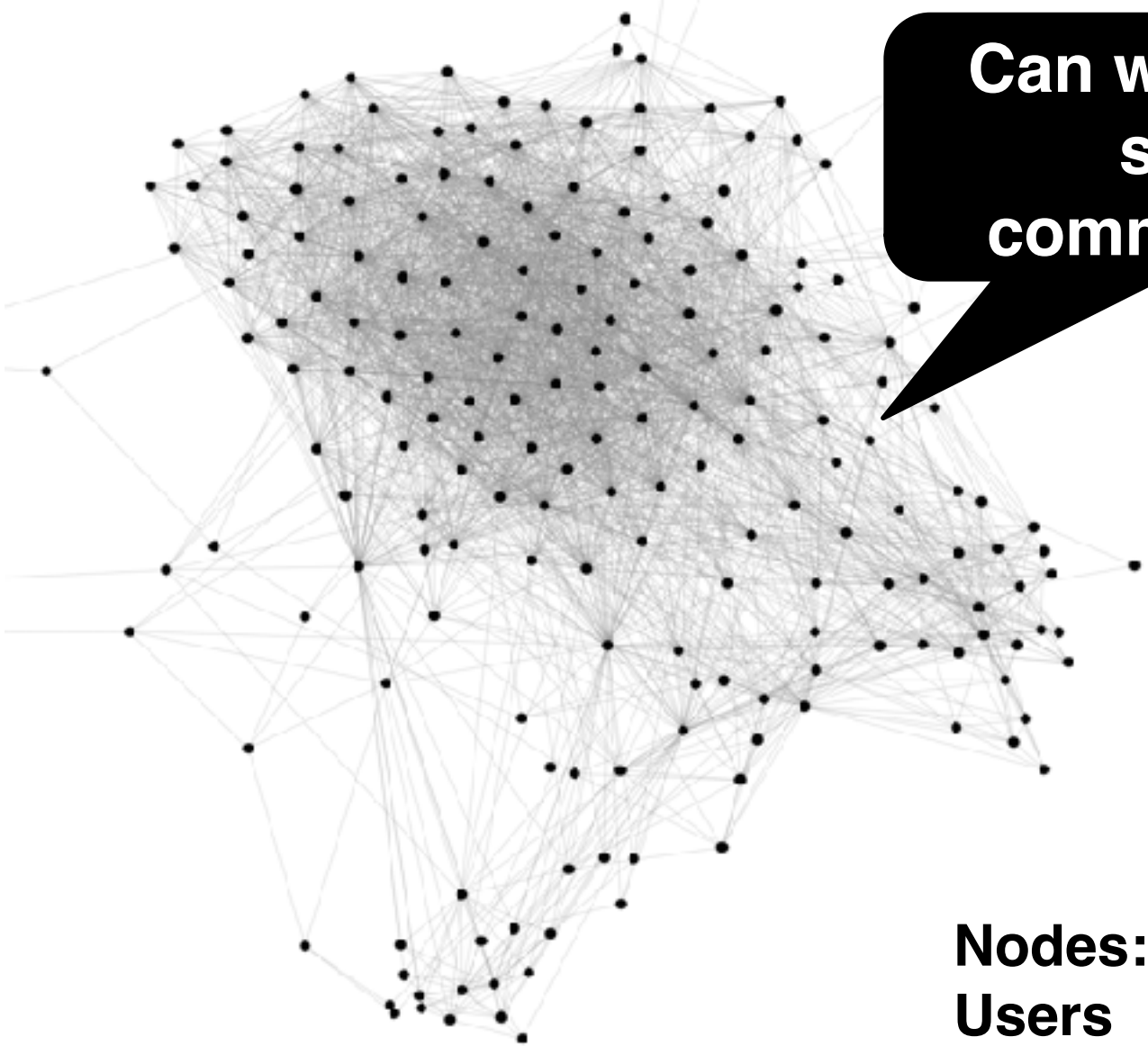
- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model
- **Add content-based methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Social Network Analysis

Social Network Graph



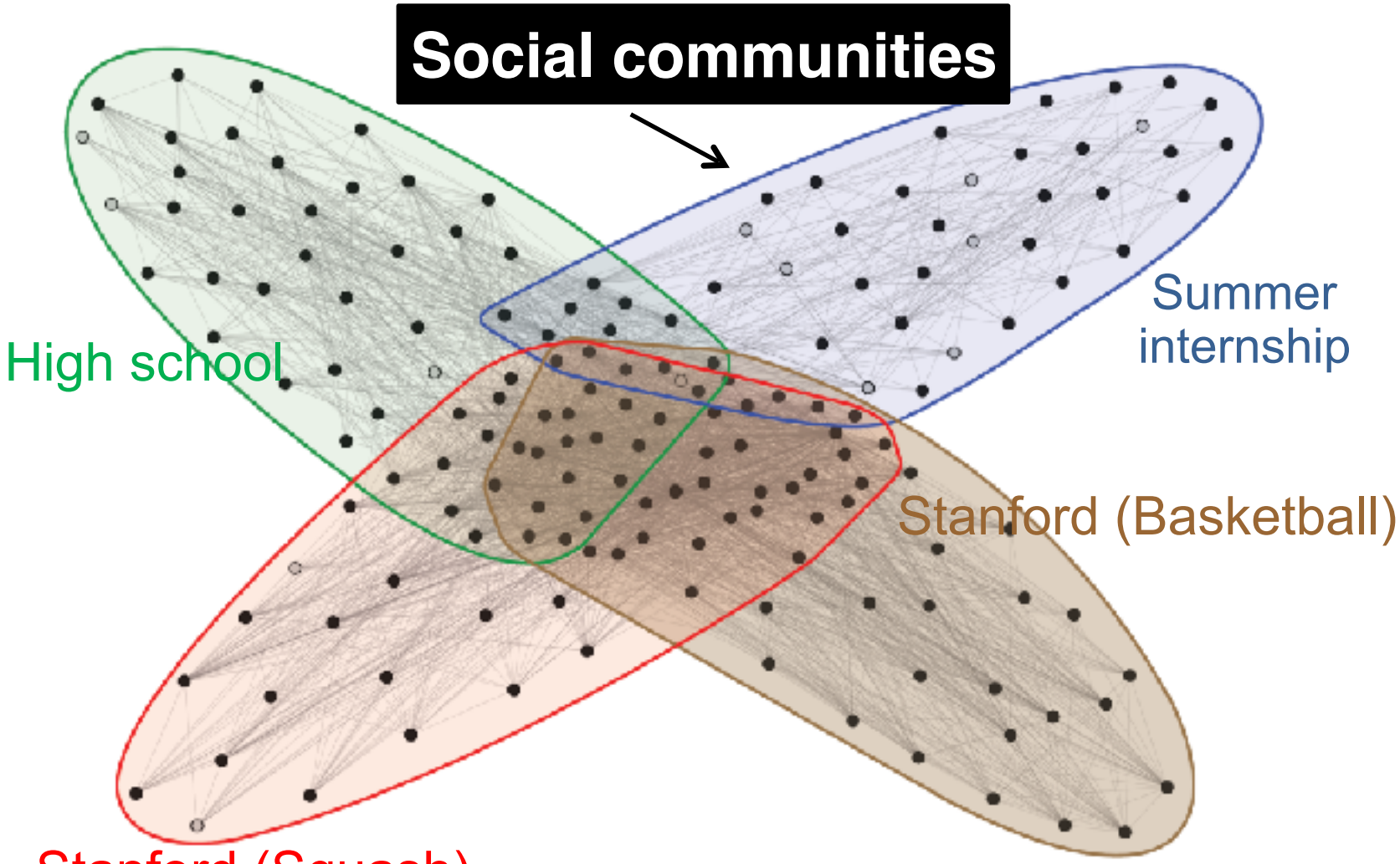
Facebook Network



Can we identify social communities?

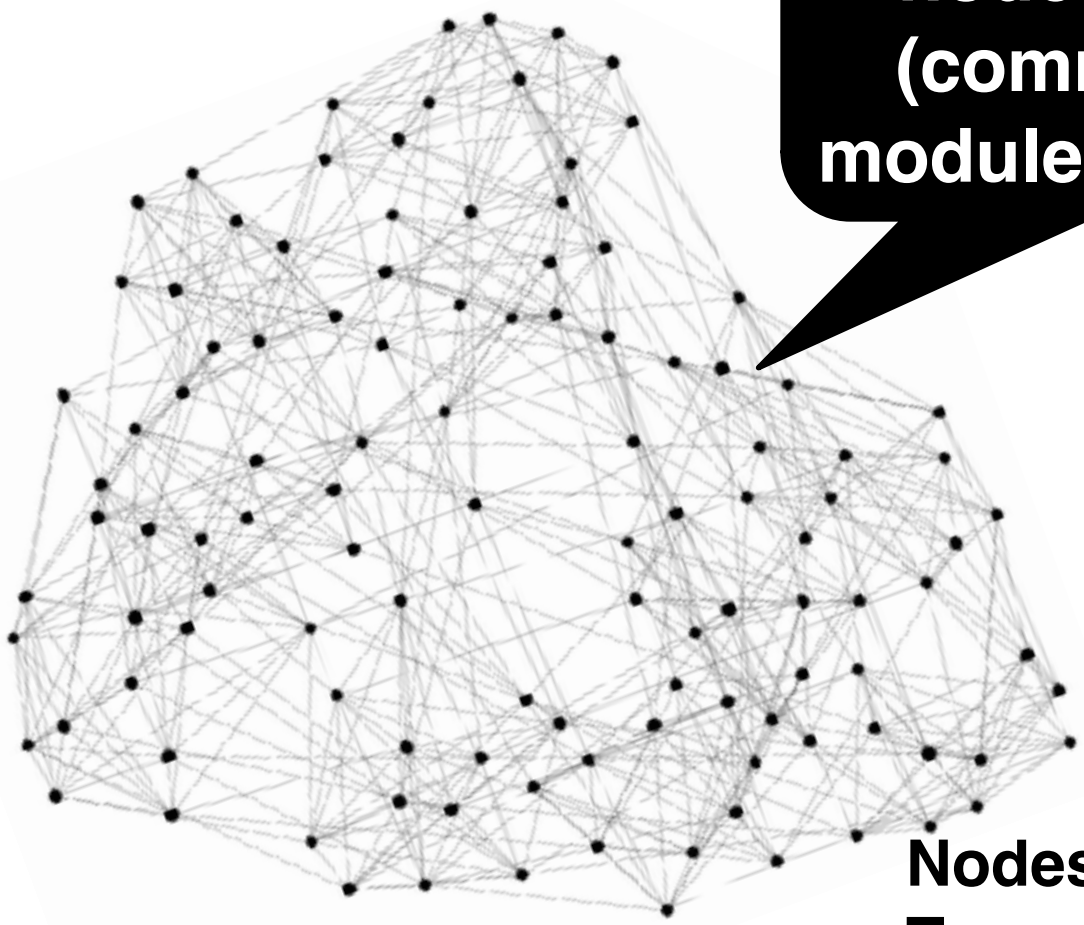
Nodes: Facebook Users

Facebook Network



Nodes: Facebook Users

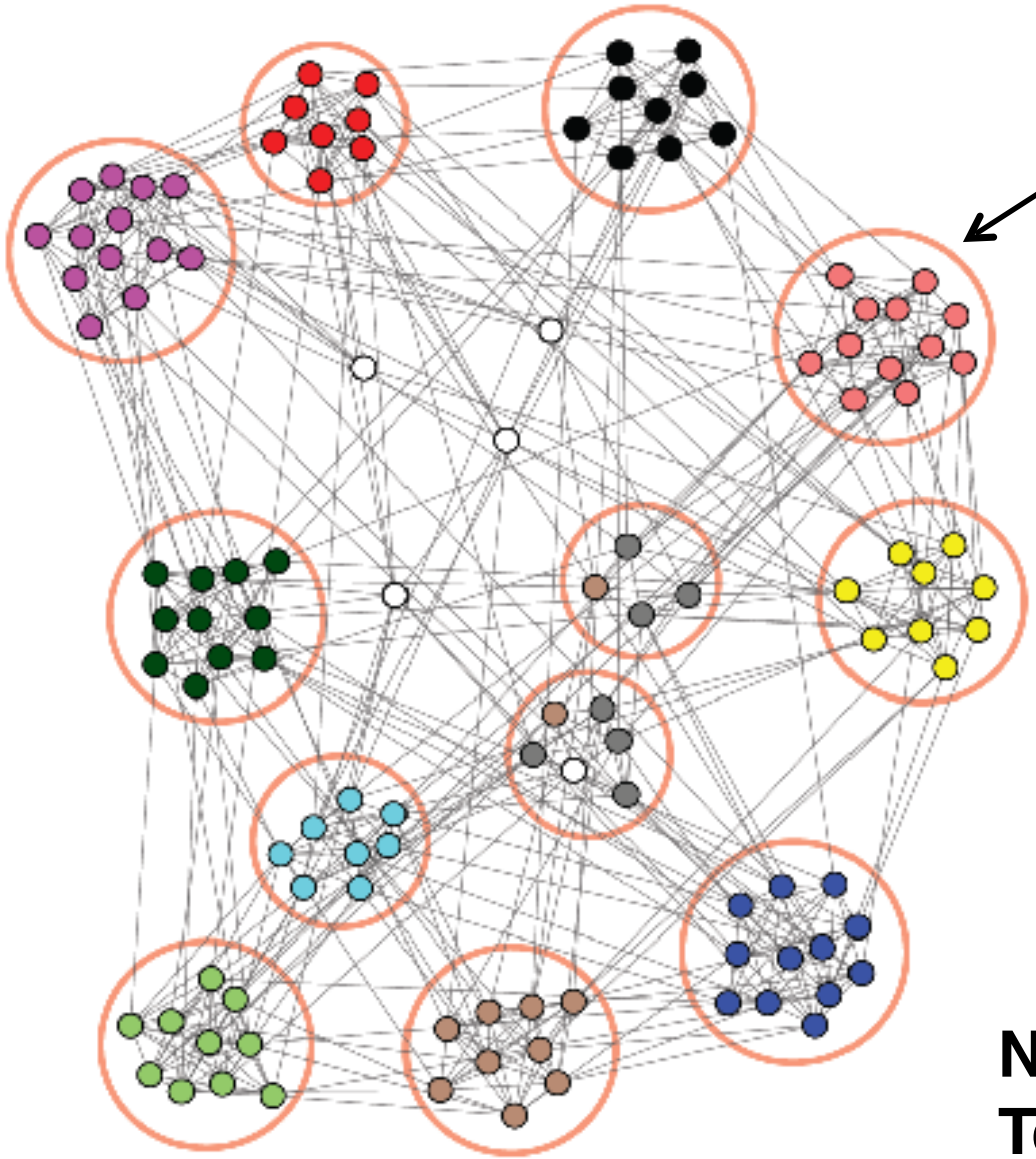
Identifying Communities



Can we identify node groups? (communities, modules, clusters)

Nodes: Football Teams

NCAA Football Network

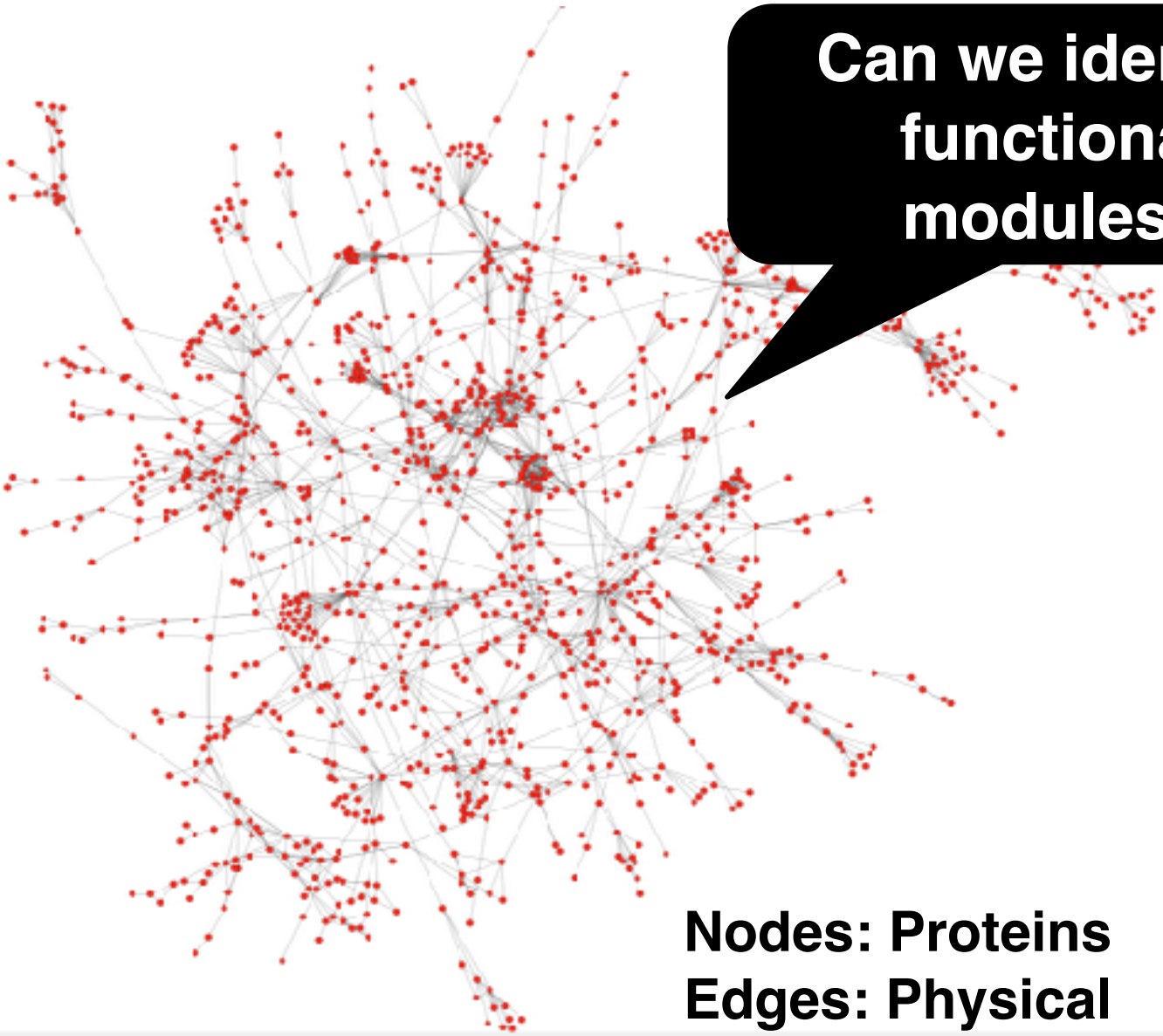


NCAA conferences

- Mid American
- Big East
- Atlantic Coast
- SEC
- Conference USA
- Big 12
- Western Athletic
- Pacific 10
- Mountain West
- Big 10
- Sun Belt
- Independents

Nodes: Football Teams

Other: Protein-Protein Interactions

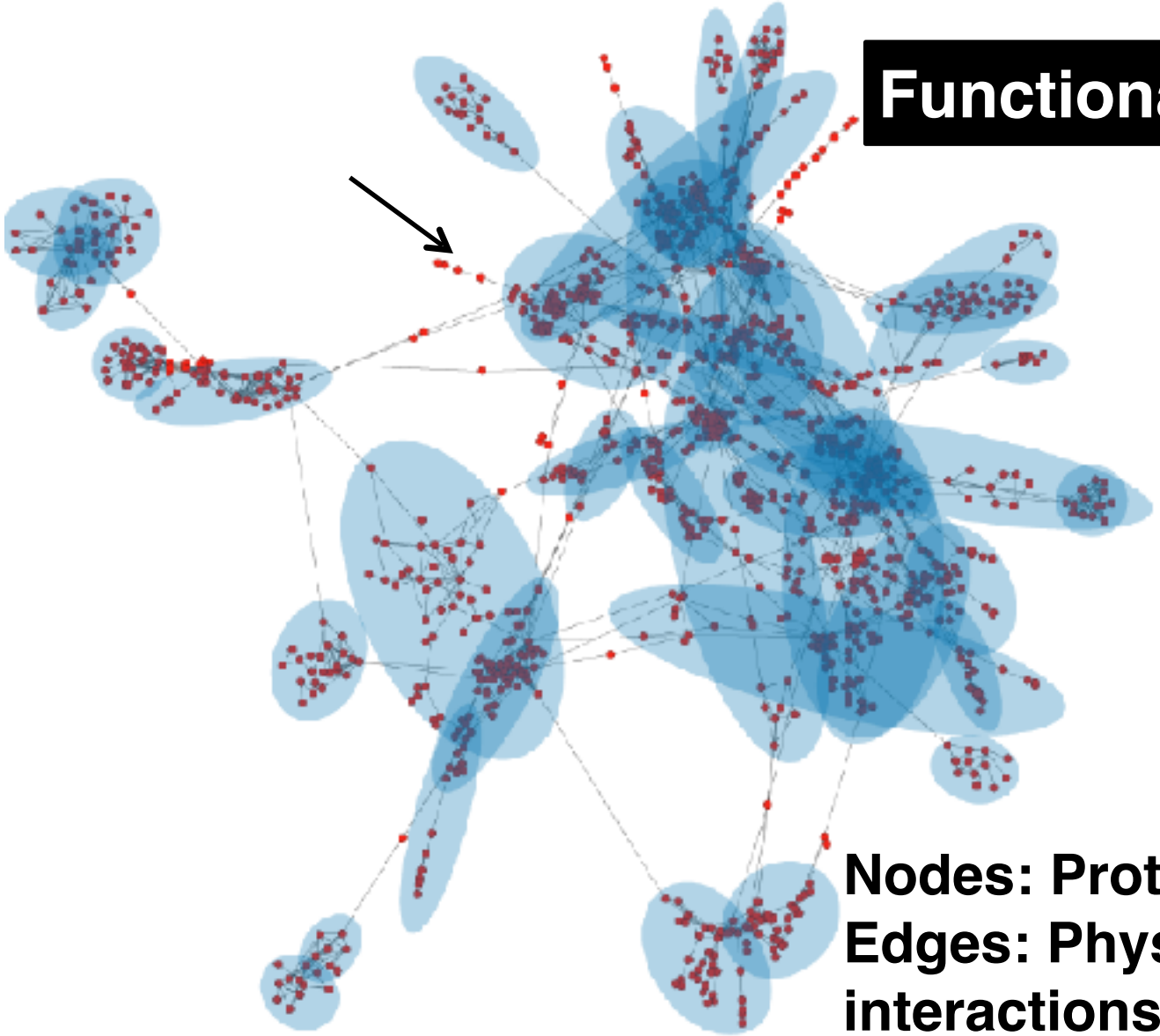


Can we identify functional modules?

Nodes: Proteins
Edges: Physical

7

Other: Protein-Protein Interactions

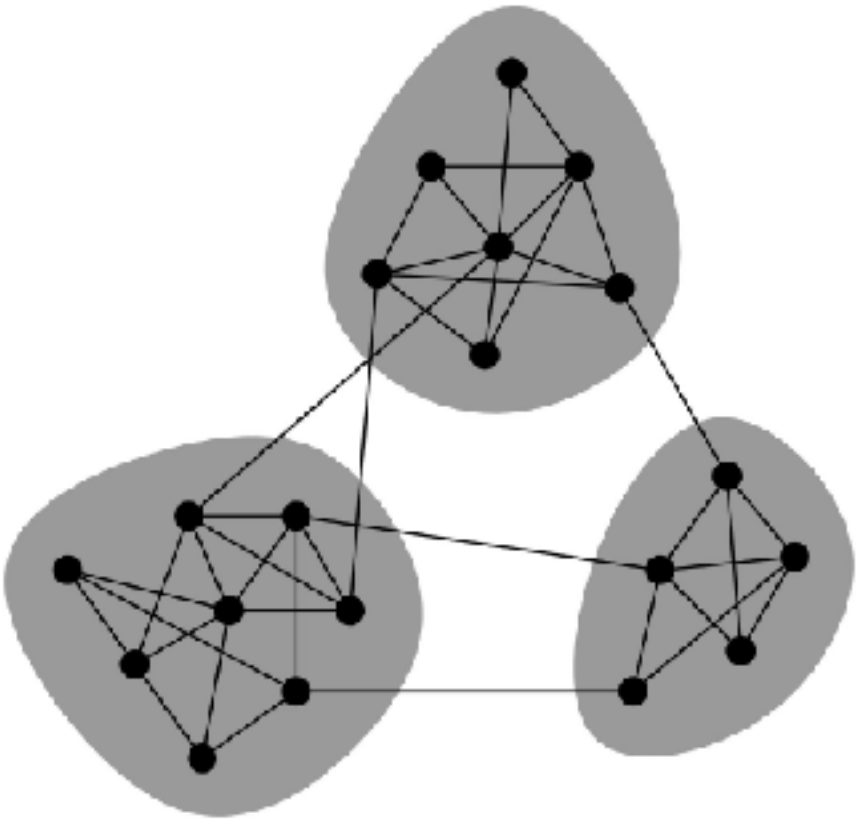


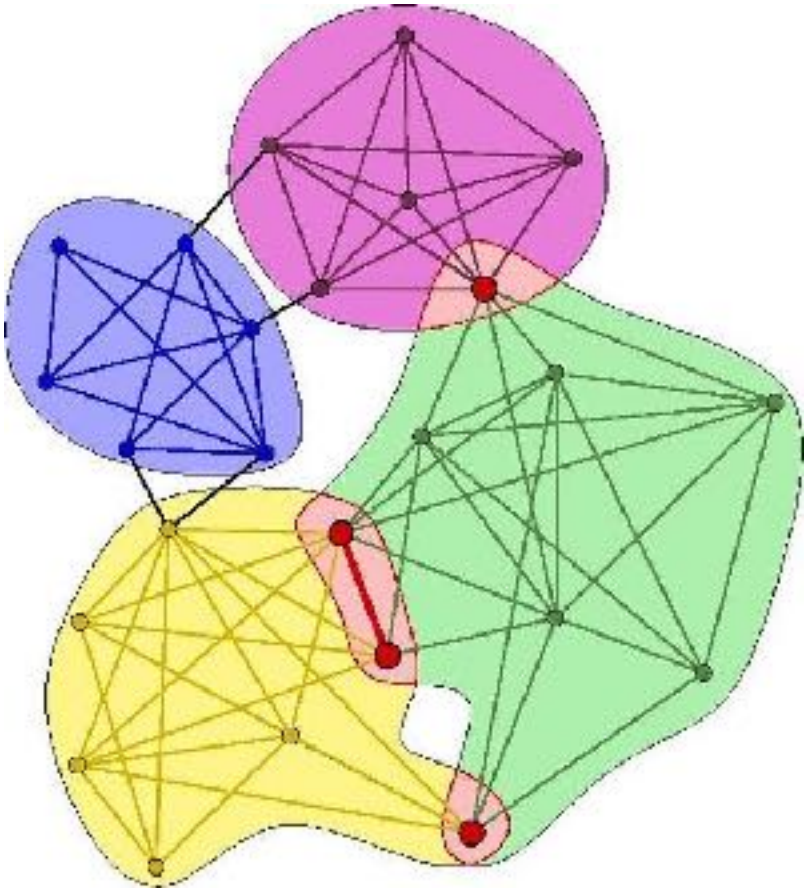
Functional modules

Nodes: Proteins
Edges: Physical interactions

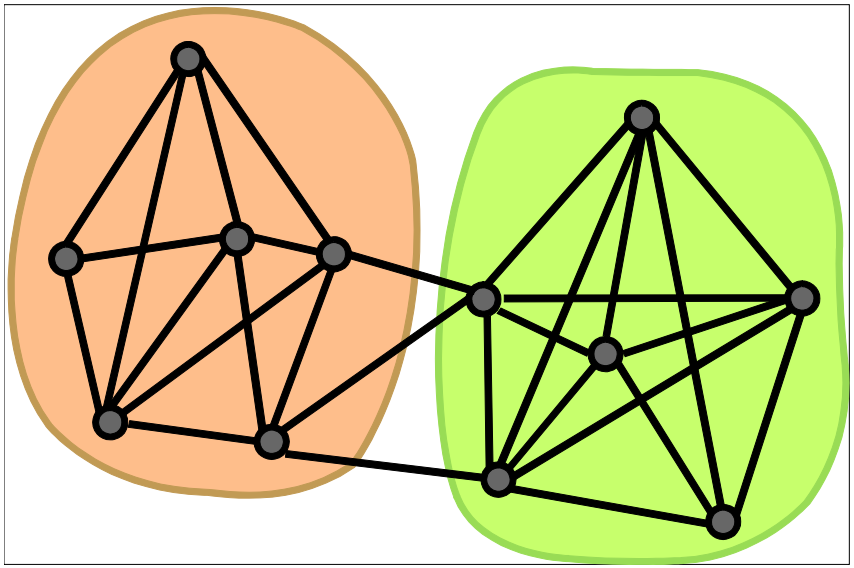
Overlapping Communities

- **Non-overlapping** vs. **overlapping communities**

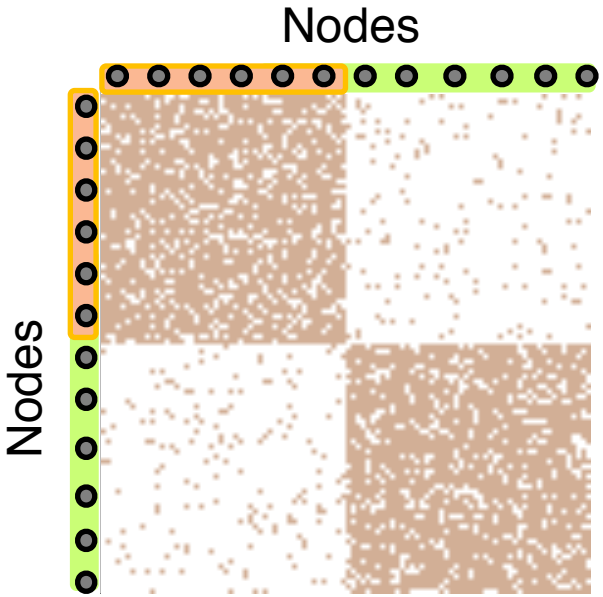




Non-overlapping Communities



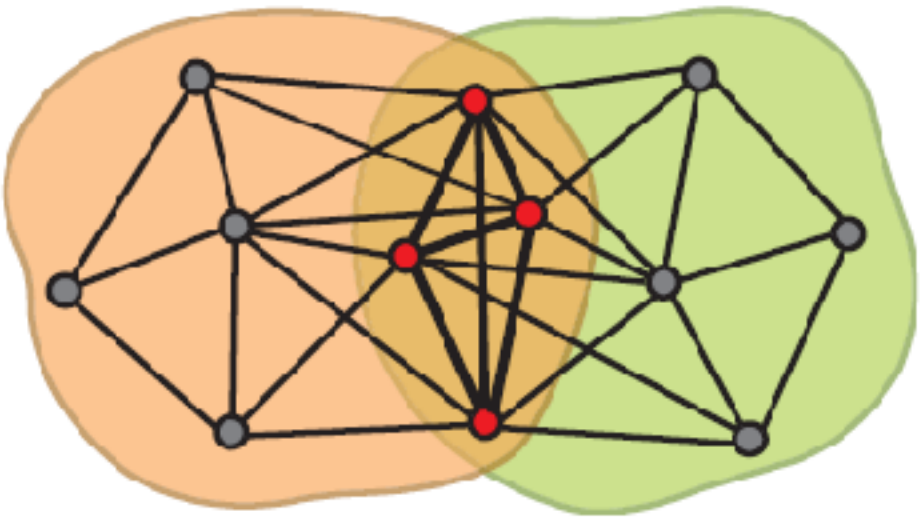
Network

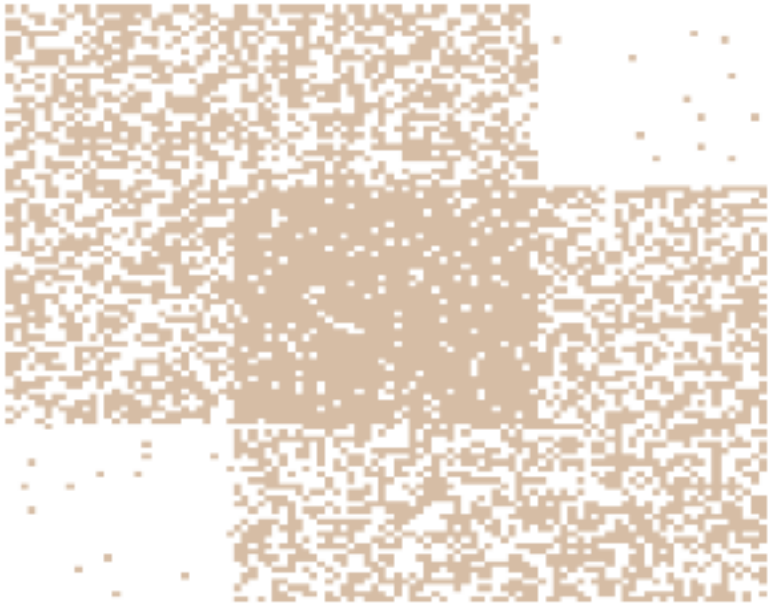


Adjacency matrix

Communities as Tiles!

- What is the structure of community overlaps: Edge density in the overlaps is higher!

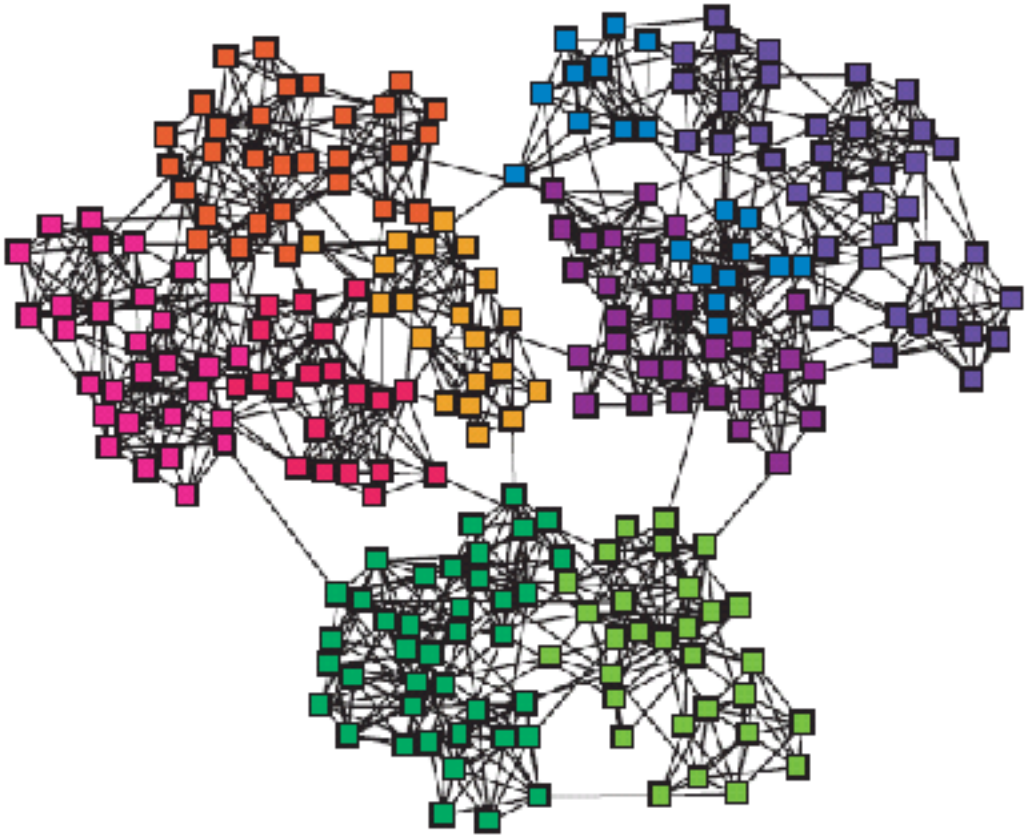




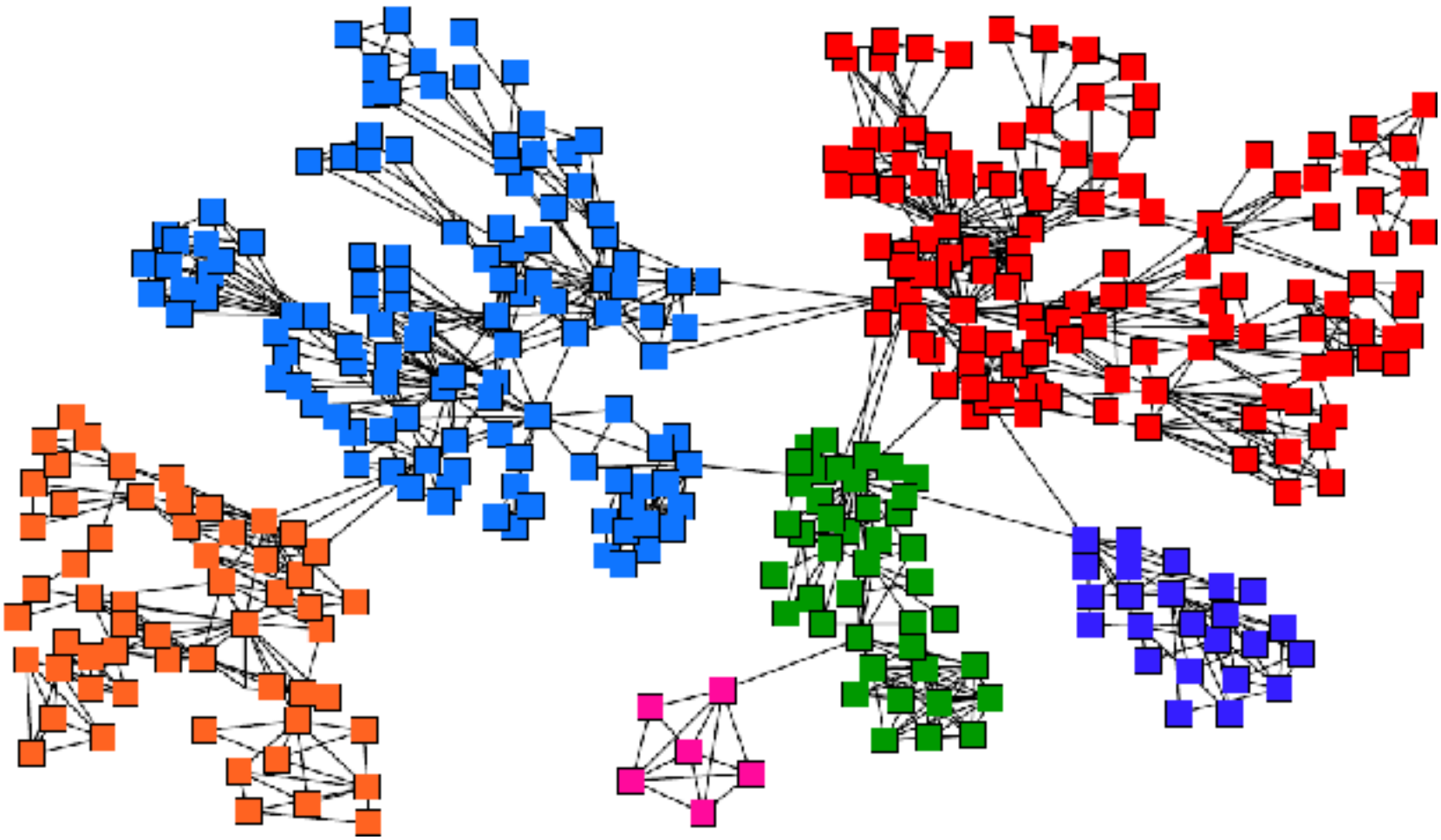
Communities as “tiles”

Networks & Communities

- **Often think networks organized into modules, cluster, communities:**

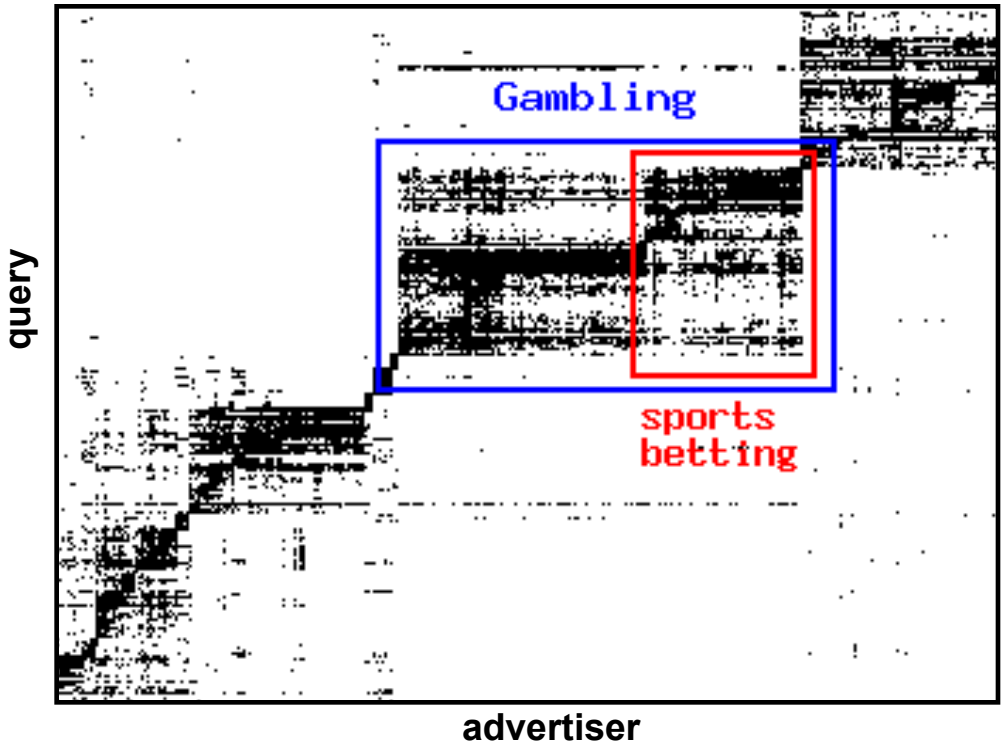


Goal: Find Densely Linked Clusters



Micro-Markets in Sponsored Search

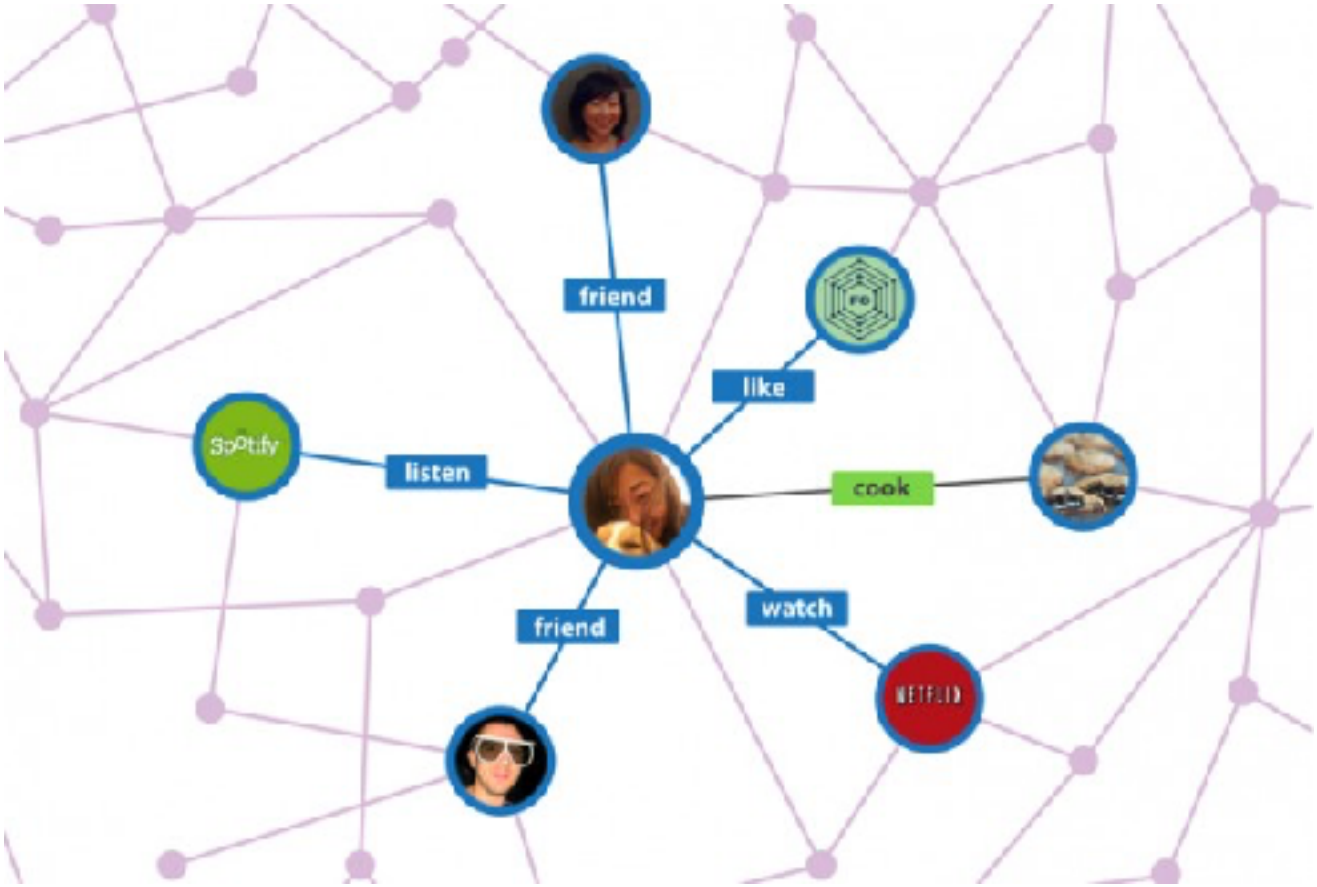
- Find micro-markets by partitioning the query-to-advertiser graph:



[Andersen, Lang: Communities from seed sets, 2006]

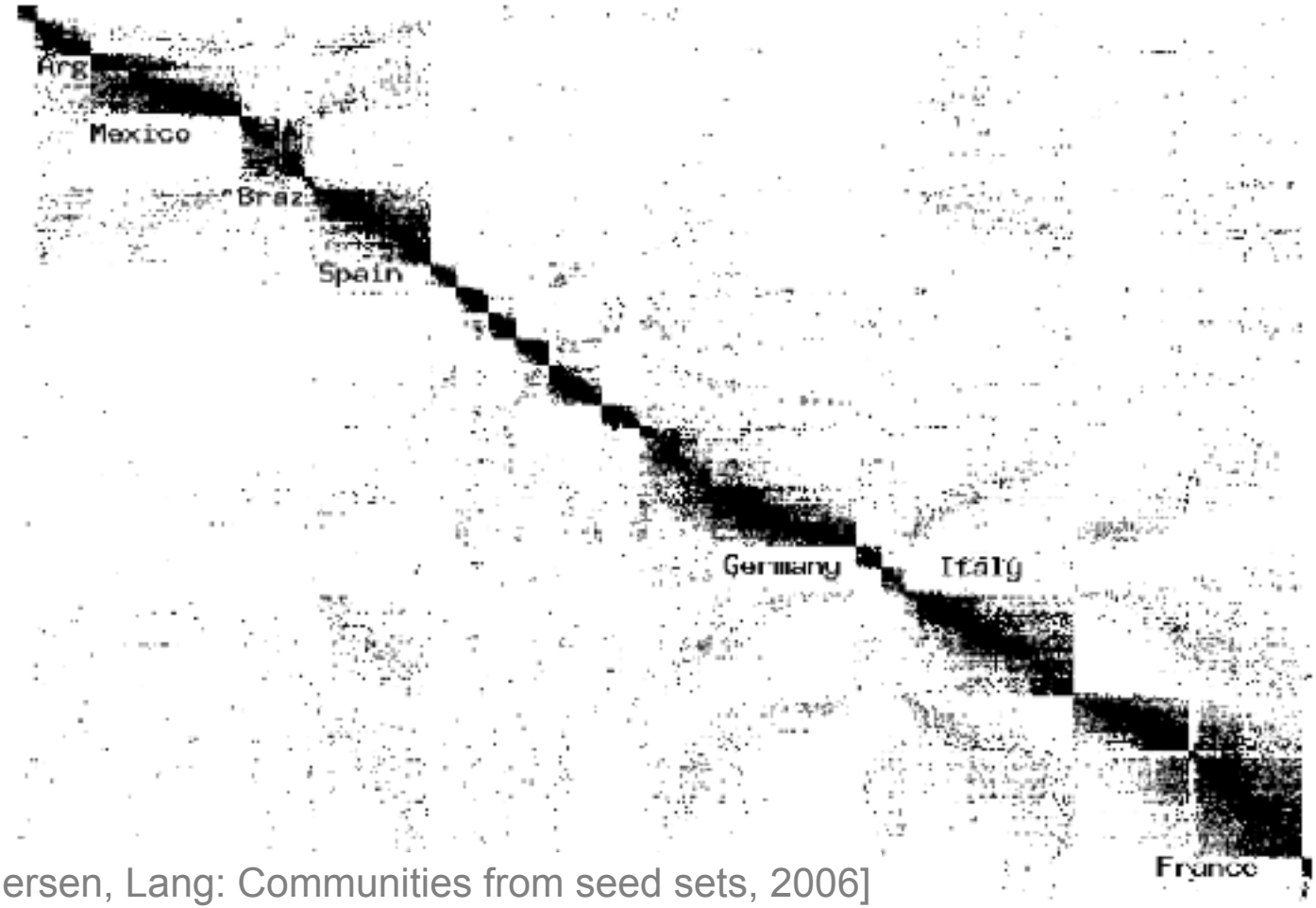
For Recommender Systems

- Find what a person likes



Movies and Actors

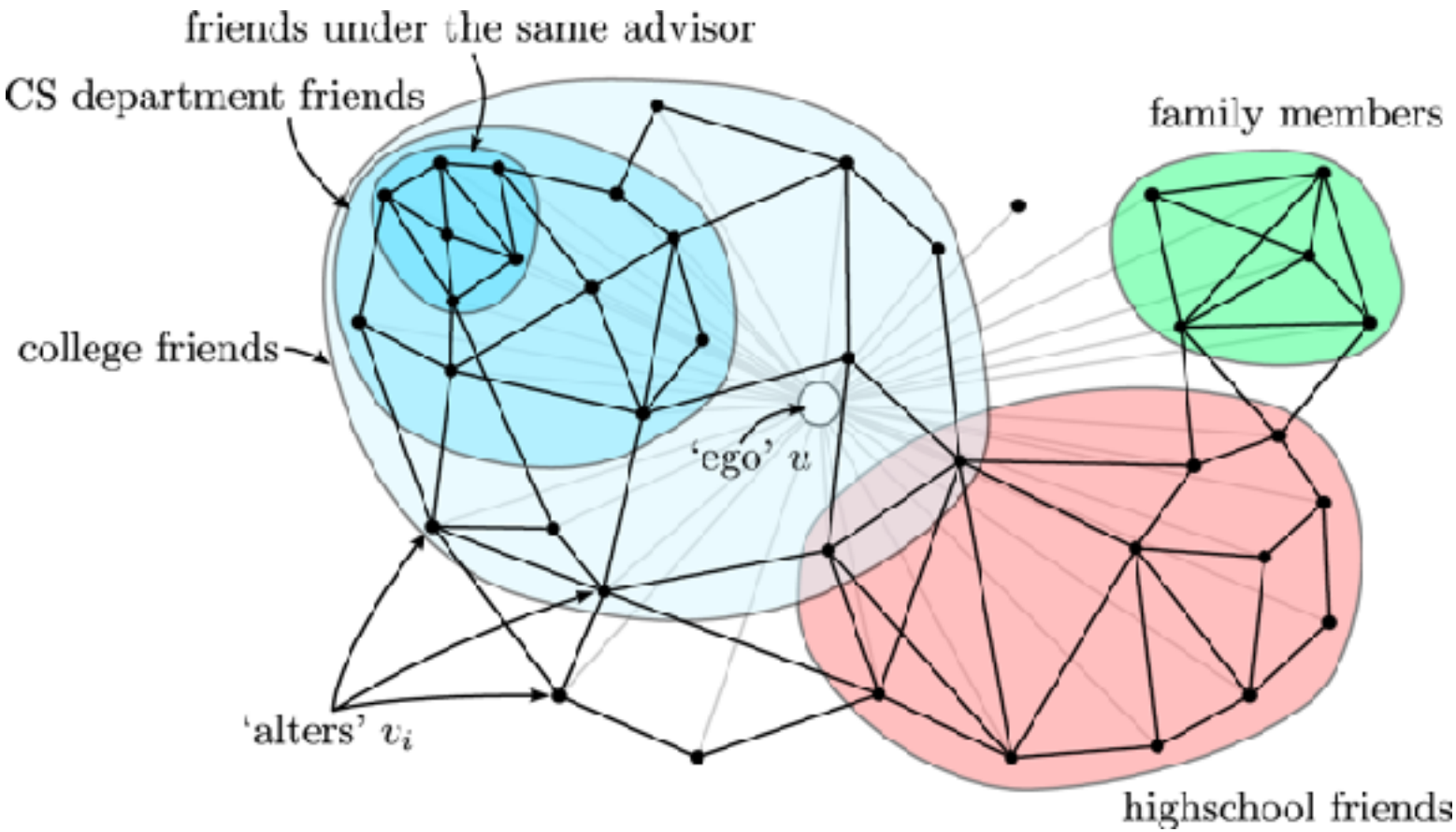
- Clusters in Movies-to-Actors graph:



[Andersen, Lang: Communities from seed sets, 2006]

Twitter & Facebook

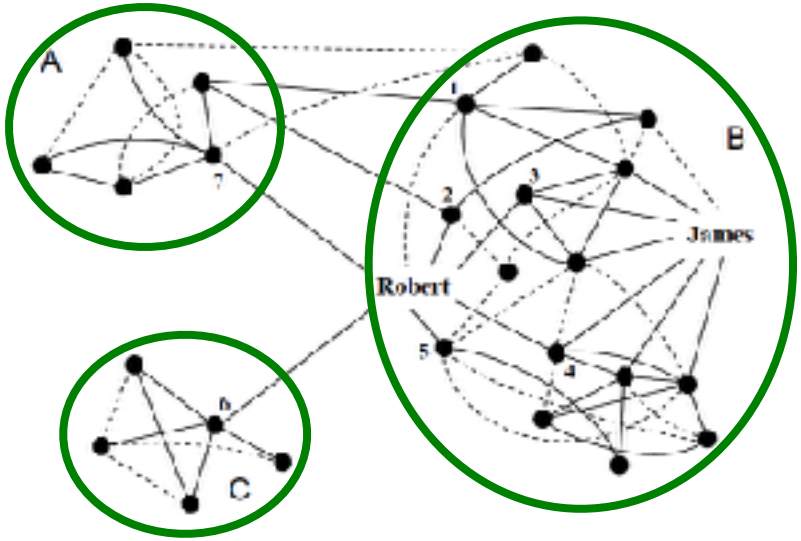
- Discovering social circles, circles of trust:

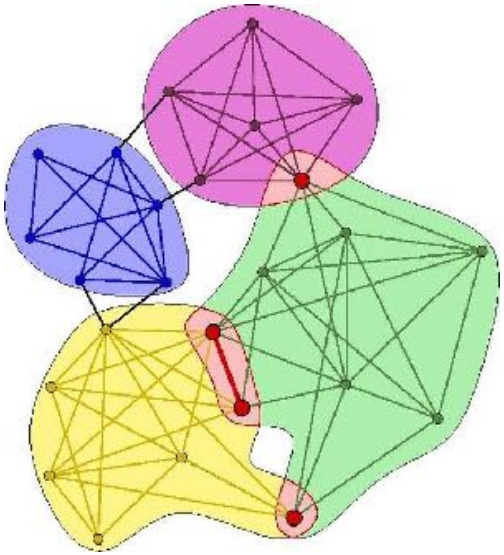


[McAuley, Leskovec: Discovering social circles in ego networks, 2012]

COMMUNITY DETECTION

How to find communities?

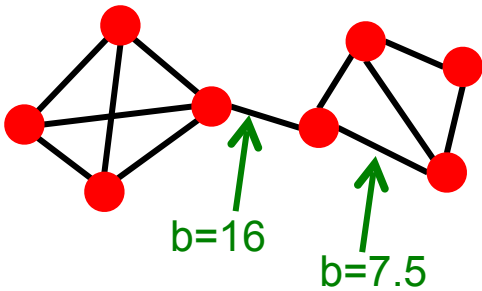


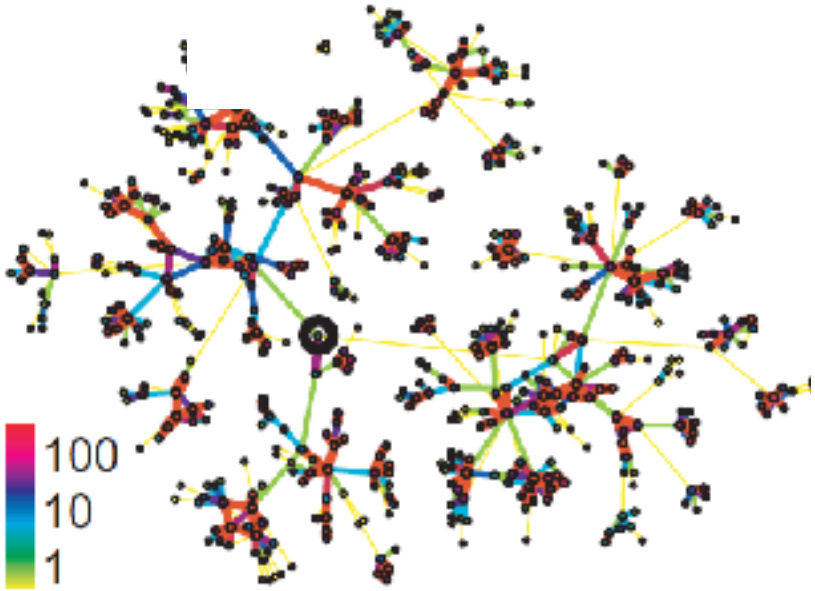


We will work with **undirected** (unweighted) networks

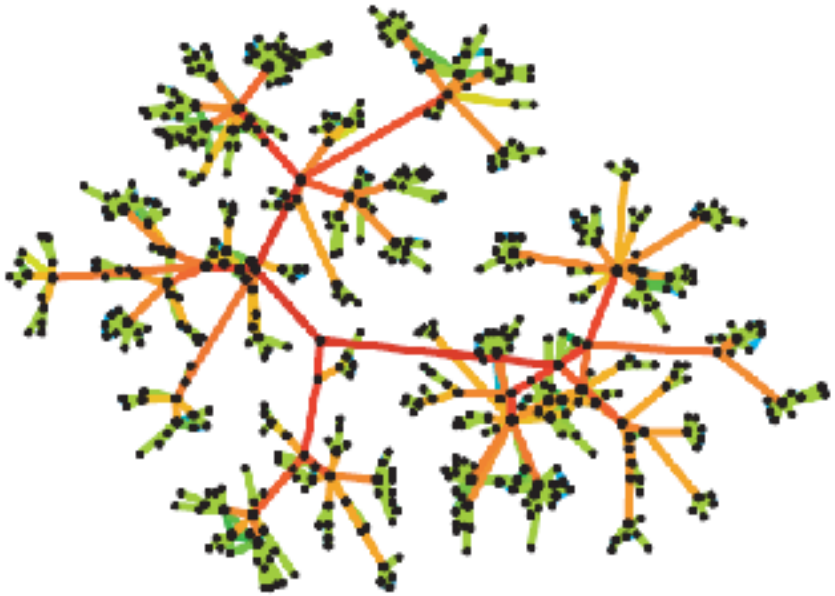
Method 1: Strength of Weak Ties

- **Edge betweenness:** Number of shortest paths passing over the edge
- Intuition:





Edge strengths (call volume) in a real network



Edge betweenness in a real network

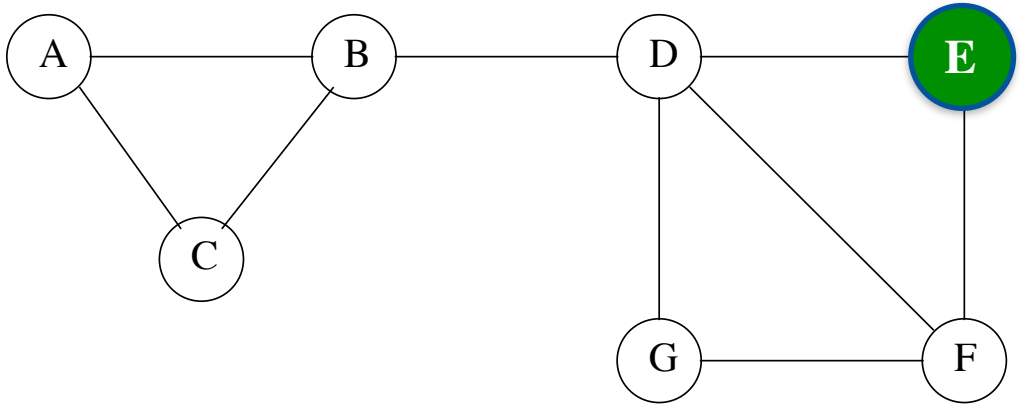
Method 1: Girvan-Newman

- Divisive hierarchical clustering based on the notion of **edge betweenness**:
  - Number of shortest paths passing through the edge
- **Girvan-Newman Algorithm**:

  Undirected unweighted networks

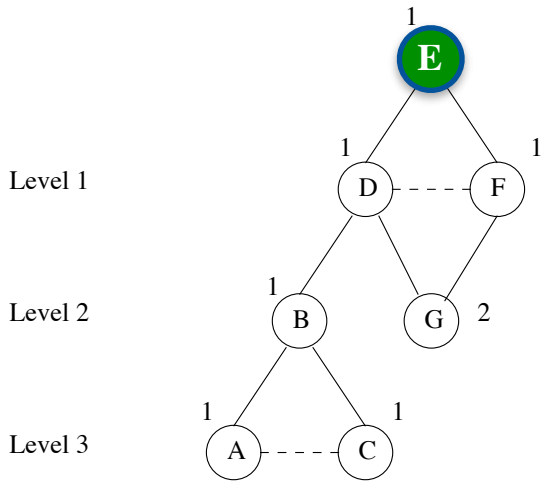
  - Repeat until no edges are left:
    - Calculate betweenness of edges
    - Remove edges with highest betweenness
  - Connected components are communities
  - Gives a hierarchical decomposition of the network

Girvan-Newman: Example

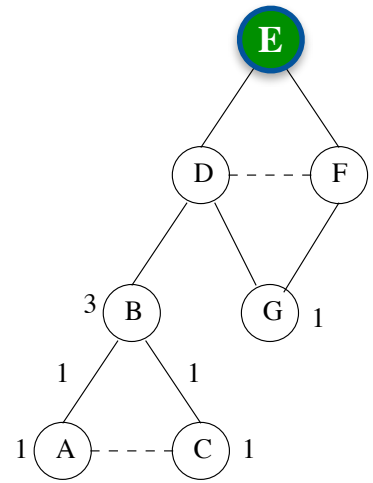


- Repeat until no edges are left:
  - **Calculate betweenness of edges**
  - Remove edges with highest betweenness
- Connected components are communities
- Gives a hierarchical decomposition of the network

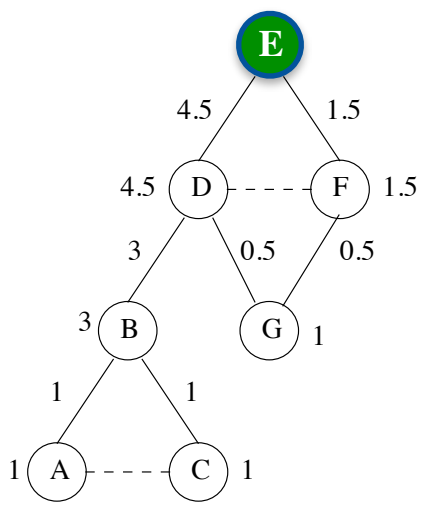
Girvan-Newman: Example



Step 1



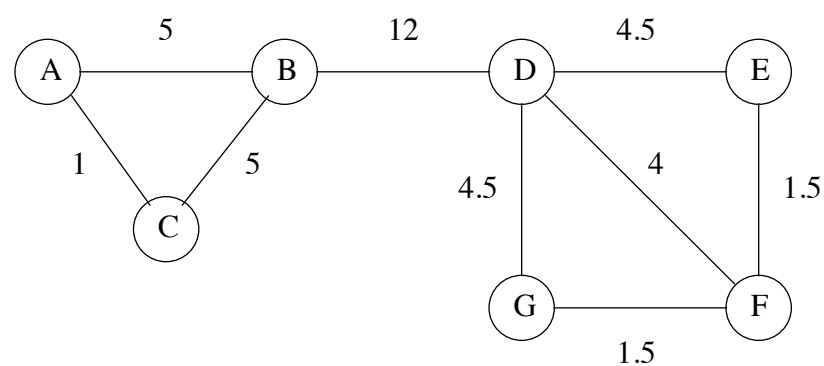
Step 2



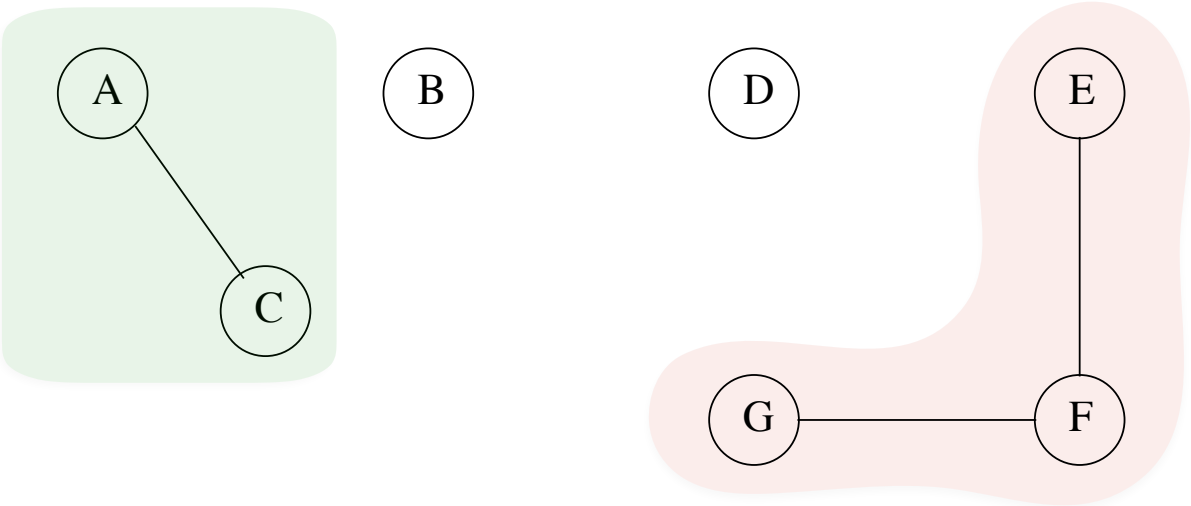
Step 3

...

Re-compute betweenness at every step

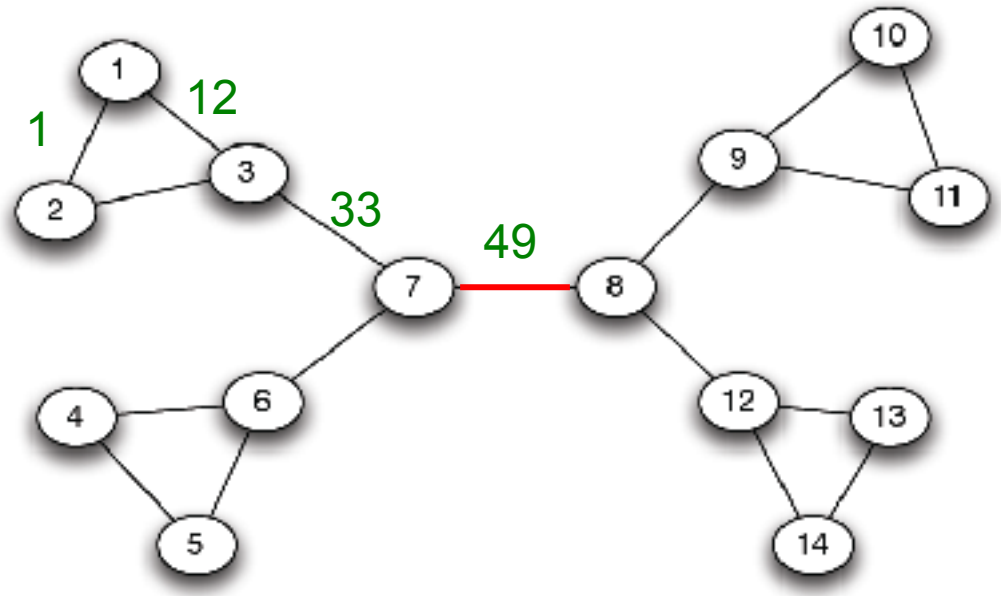


Girvan-Newman: Example



- Repeat until no edges are left:
  - Calculate betweenness of edges
  - **Remove edges with highest betweenness**
- **Connected components are communities**
- Gives a hierarchical decomposition of the network

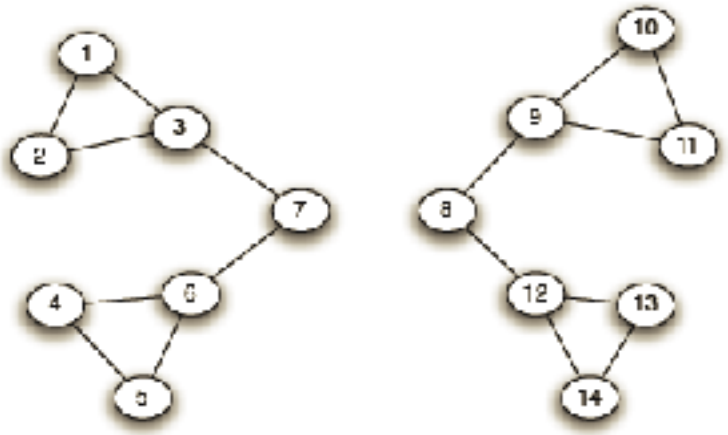
Girvan-Newman: Example



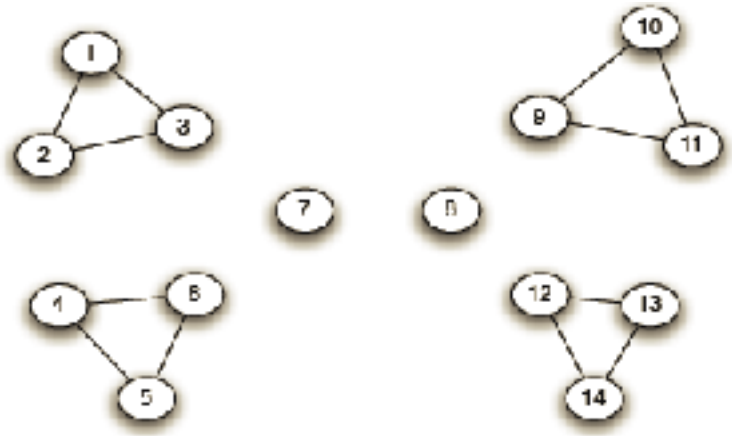
Need to re-compute betweenness at every step

Girvan-Newman: Example

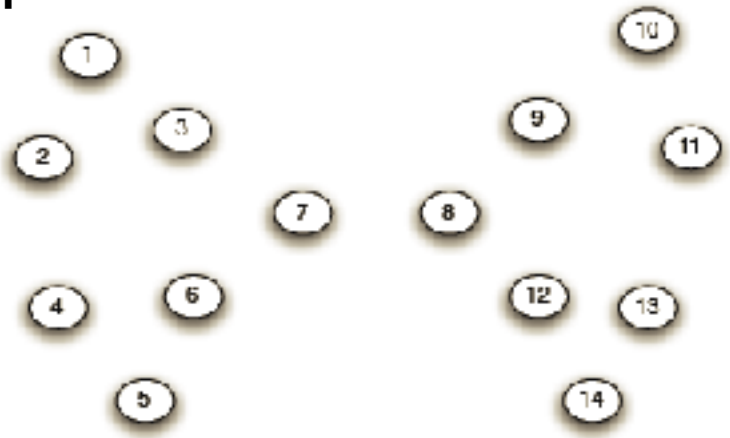
Step 1:



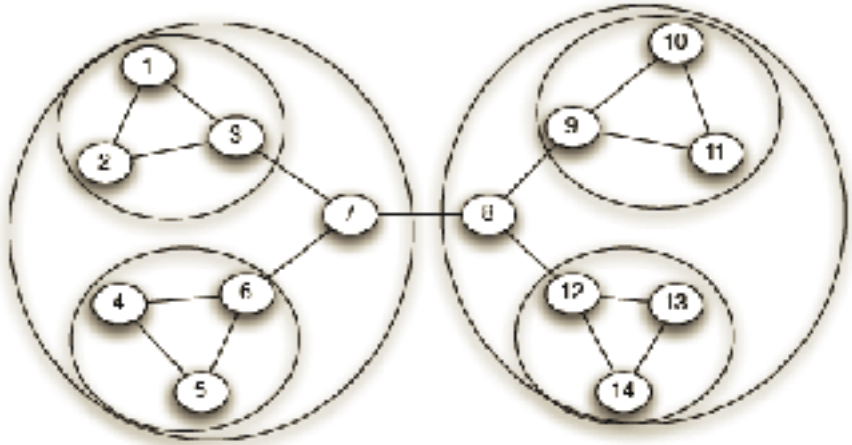
Step 2:



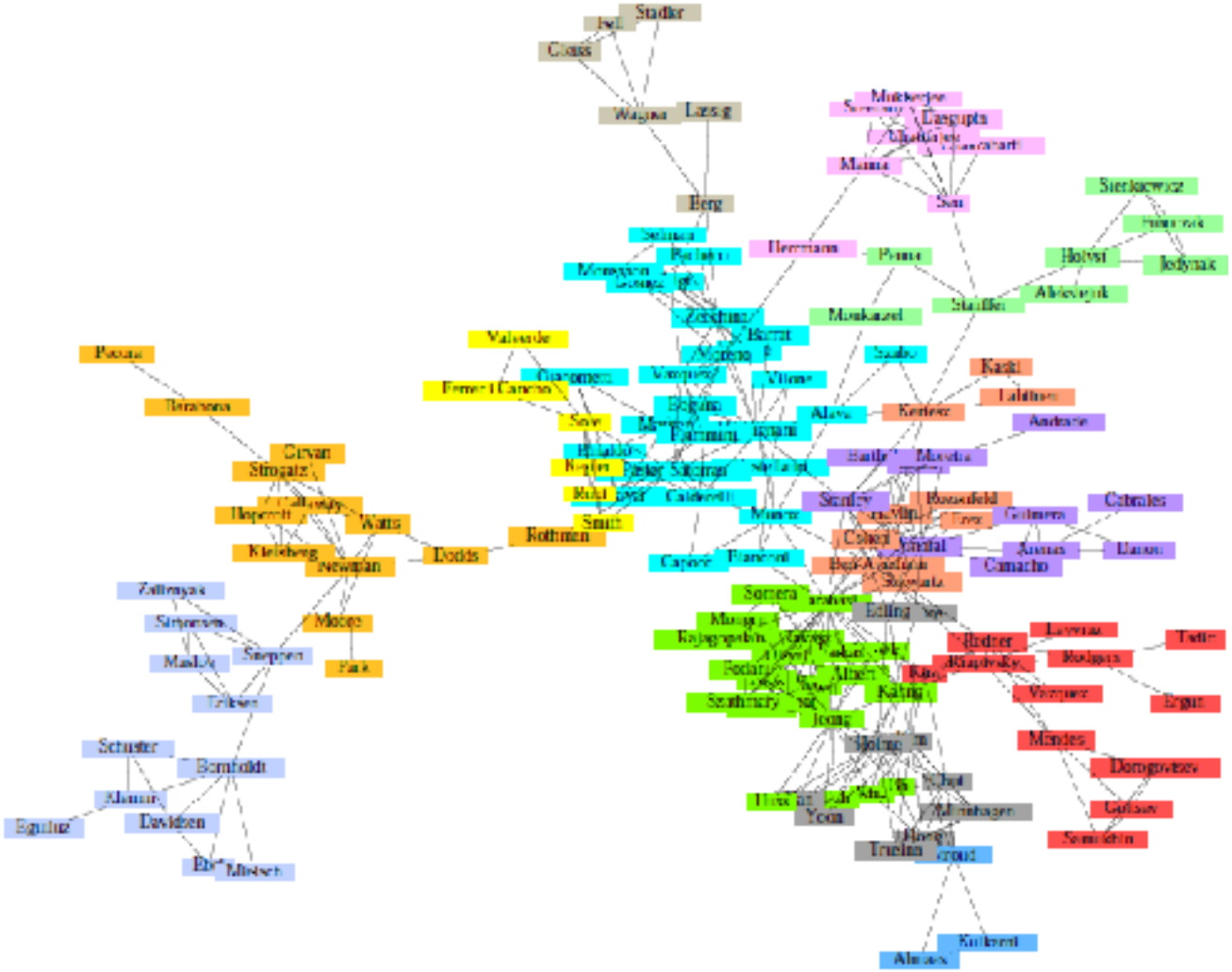
Step 3:



Hierarchical network decomposition:

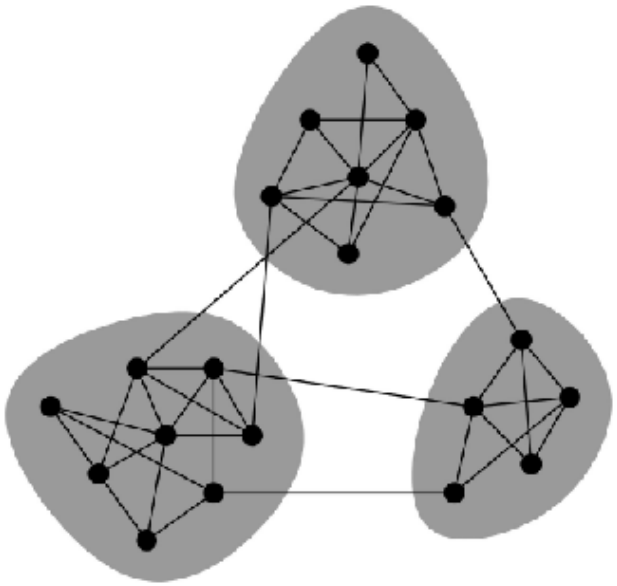


Girvan-Newman: Results



Communities in physics collaborations

Network Communities



- **Communities:** sets of **tightly connected nodes**
- Define: **Modularity Q**
  - A measure of how well a network is partitioned into communities
  - Given a partitioning of the network into groups s S:

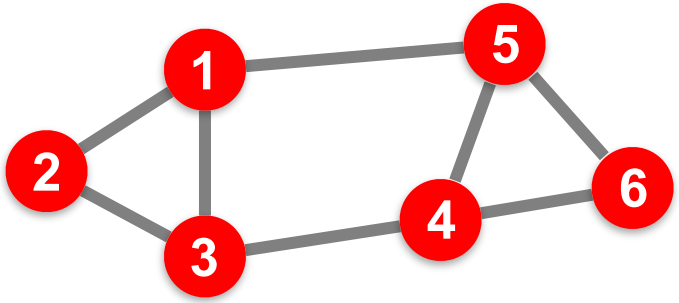
$$Q \propto \sum_{s \in S} [\,(\#\text{ edges within group } s) - (\text{expected }\#\text{ edges within group } s)\,]$$

Need a null model!

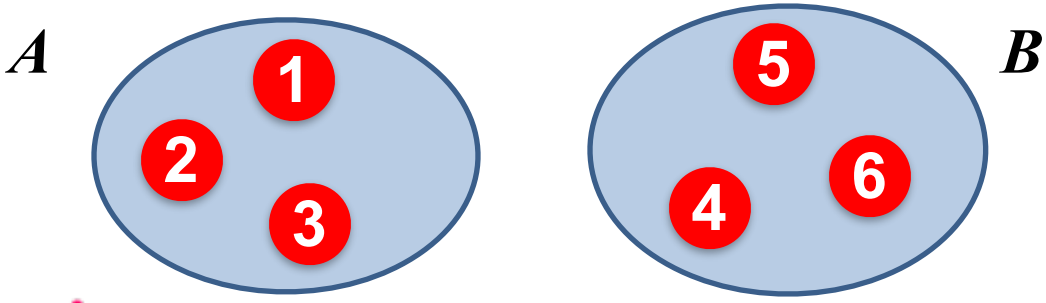
Spectral Clustering

Graph Partitioning

- Undirected graph $G(V, E)$:



- **Bi-partitioning task:**
  - Divide vertices into two disjoint groups A, B

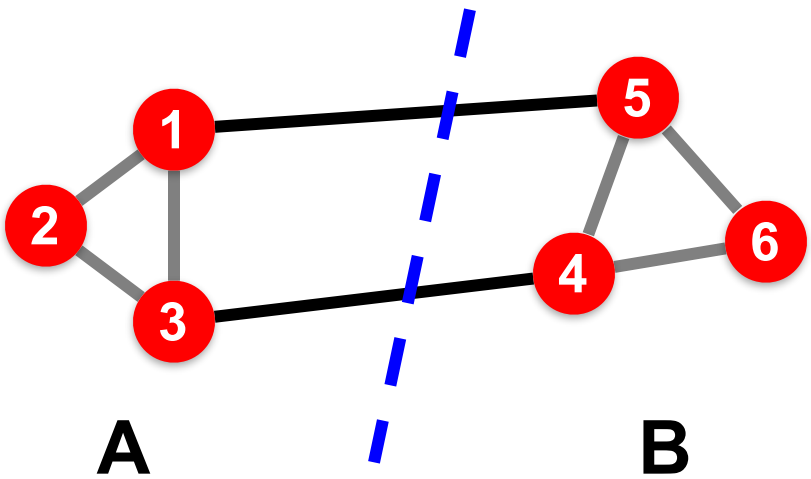


- **Questions:**
  - How can we define a "good" partition of G?
  - How can we efficiently identify such a partition?

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Graph Partitioning

- **What makes a good partition?**
  - Maximize the number of within-group connections
  - Minimize the number of between-group connections

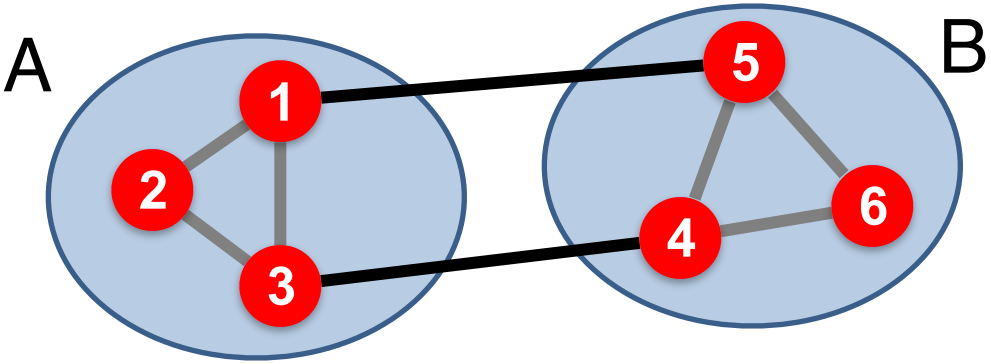


J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Graph Cuts

- Express partitioning objectives as a function of the "edge cut" of the partition
- Cut: Set of edges with only one vertex in a group:

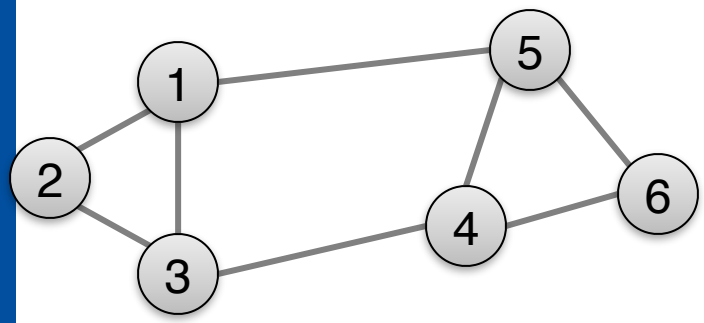
$$cut(A,B) = \sum_{i \in A,\, j \in B} w_{ij}$$



$cut(A,B) = 2$

Matrix Representations

- **Adjacency matrix (A):**
  - $n \times n$ matrix
  - $A=[a_{ij}]$, $a_{ij}=1$ if edge between node i and j



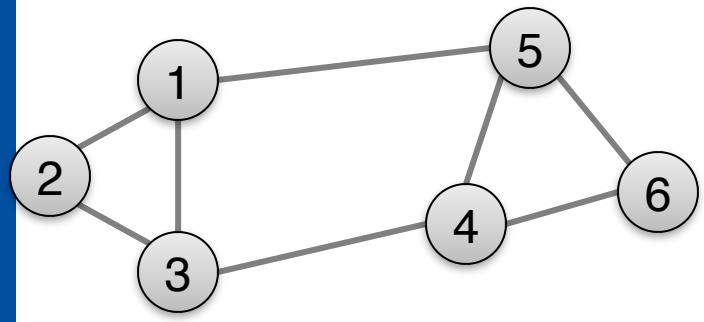
	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

- **Important properties:**
  - Symmetric matrix
  - Eigenvectors are real and orthogonal

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Matrix Representations

- **Degree matrix (D):**
  - $n \times n$ diagonal matrix
  - $D=[d_{ii}]$, d_{ii} = degree of node i

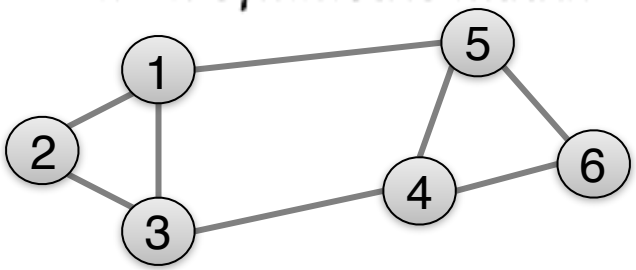


	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Matrix Representations

- **Laplacian matrix (L):**
  - $n \times n$ symmetric matrix



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- **What is trivial eigenpair?**
  - $x = (1, \ldots, 1)$ then $L \cdot x = 0$ and so $\lambda = \lambda_1 = 0$
- **Important properties:**
  - **Eigenvalues** are non-negative real numbers
  - **Eigenvectors** are real and orthogonal

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org