Machine Learning at Scale

James G. Shanahan[1, 2]

Assistants: Liang Dai[1, 3]

[1]NativeX, [2]iSchool UC Berkeley, CA, [3]UC Santa Cruz

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Live Session #3

January 26, 2016

Important Links for Week 3

- **Instructions for Peer grading of homework HW2**
  - https://www.dropbox.com/s/97m31frthj4ac28/HOMEWORK%20GRADING%20INSTRUCTIONS%20for%20MIDS%20MLS?dl=0
- **Homework HW3 Folder Questions + Data**
  - https://www.dropbox.com/sh/uev2esasn7bvtnd/AACVeeVkw7i4jZGuOmzvewUoa?dl=0
- **Team assignments**
  - To follow by Friday
- **Please submit your homeworks (one per team) going forward via this form (and not thru the ISVC):**
  - https://docs.google.com/forms/d/1ZOr9RnIe_A06AcZDB6K1mJN4vrLeSmS2PD6Xm3eOiis/viewform?usp=send_form

Group based solutions: HW 3 and beyond

- Homework Teams
  - Groups of 3 (try individually; group up)
  - Group people geographically/cohort/group
  - Groups will be assigned by instructor
- One solution per group

- Same group each week; rotations are possible

Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
  - Office hours: Mondays at 5:30PM. Please submit Qs by Sunday at 5:30PM
  - Homework 1&2: Naive Bayes Experts (grades due on Saturday)
- **Peer Grading**
  - HW1 peer grading
  - HW2 Answer reviews
- **Week 3**
  - Hadoop useful stuff/Trivia
    - Blocks sizes
    - Total Sort
    - Understanding Hadoop via Counters
    - Hadoop File passing pattern
    - Number of mappers and reducers
  - Async lecture recap plus Q&A: Pairs/Secondary keys/Inversion pattern
  - Association rule mining via Apriori
- **Next week (Cloud: MrJob)**
- **Wrapup: Finish RECORDING (bonus points!)**

Bernouill Naïve Bayes

- **Bernouilli Naïve Bayes TODO**

GAFA Job: Logic of this class

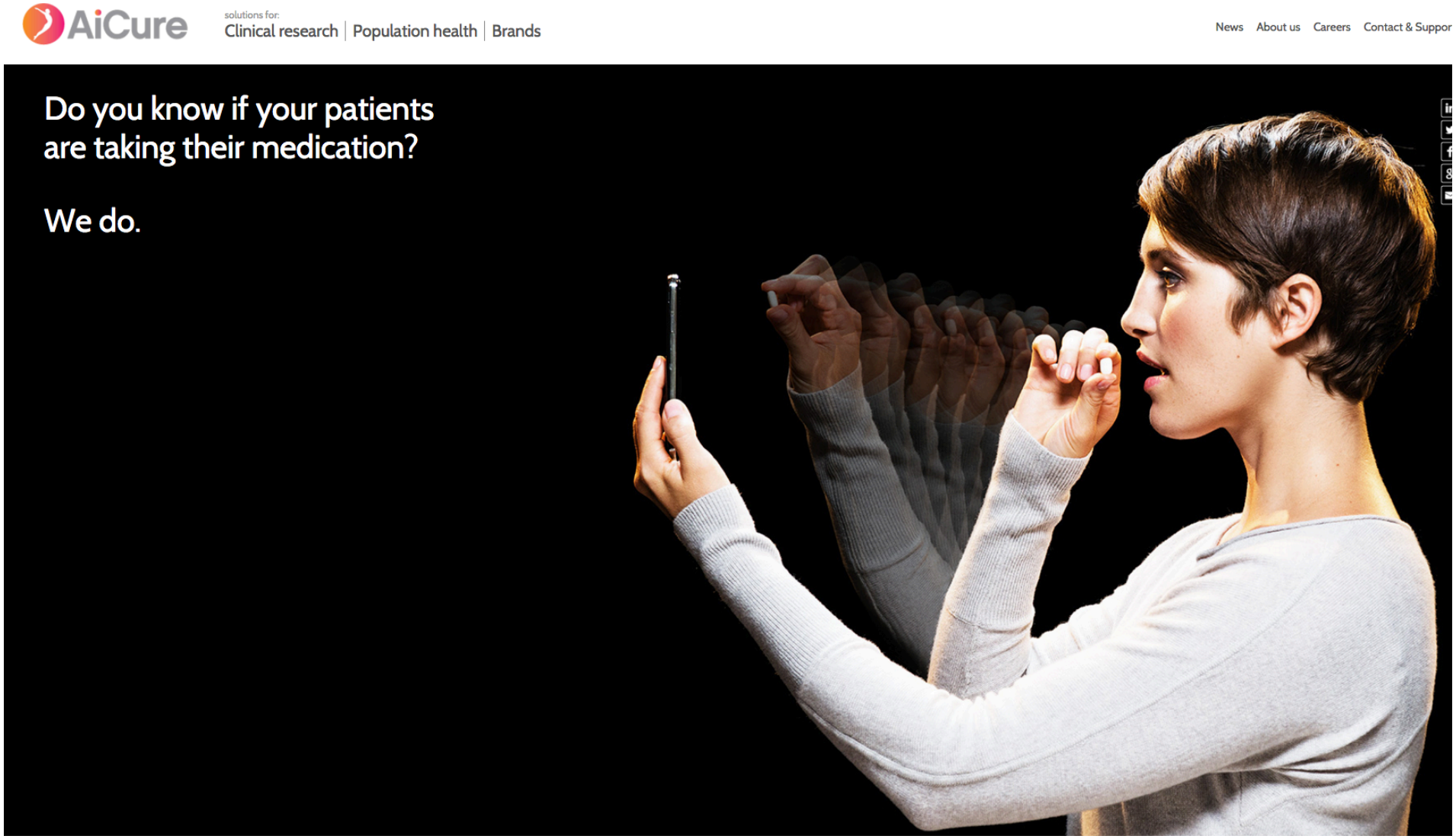
Get A Freaking Awesome (GAFA) job/PhD/career

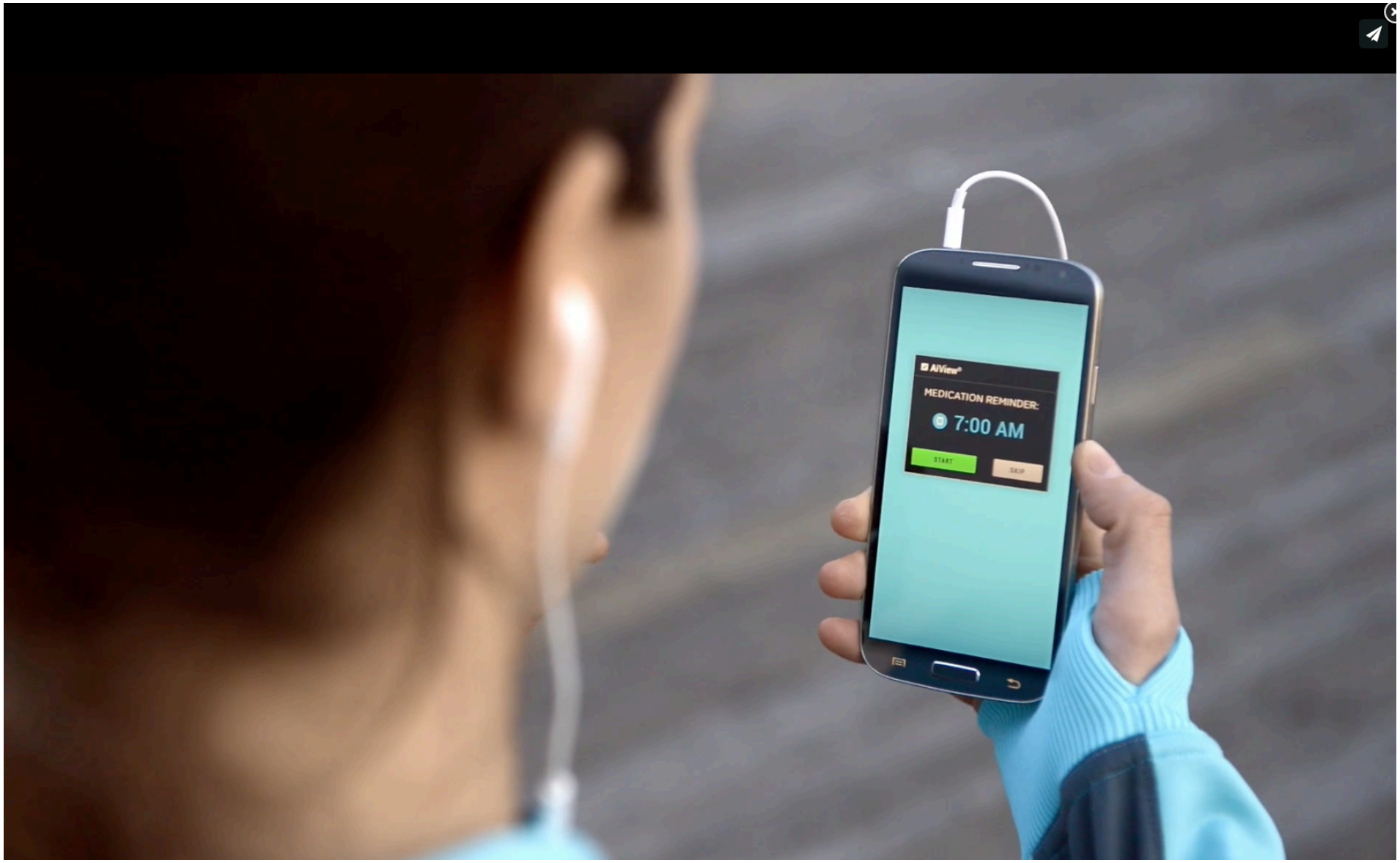
- GAFA is an acronym for Google, Apple, Facebook, and Amazon — the 4 most powerful American technology companies. Usage of the term "GAFA" is increasingly common in Europe.

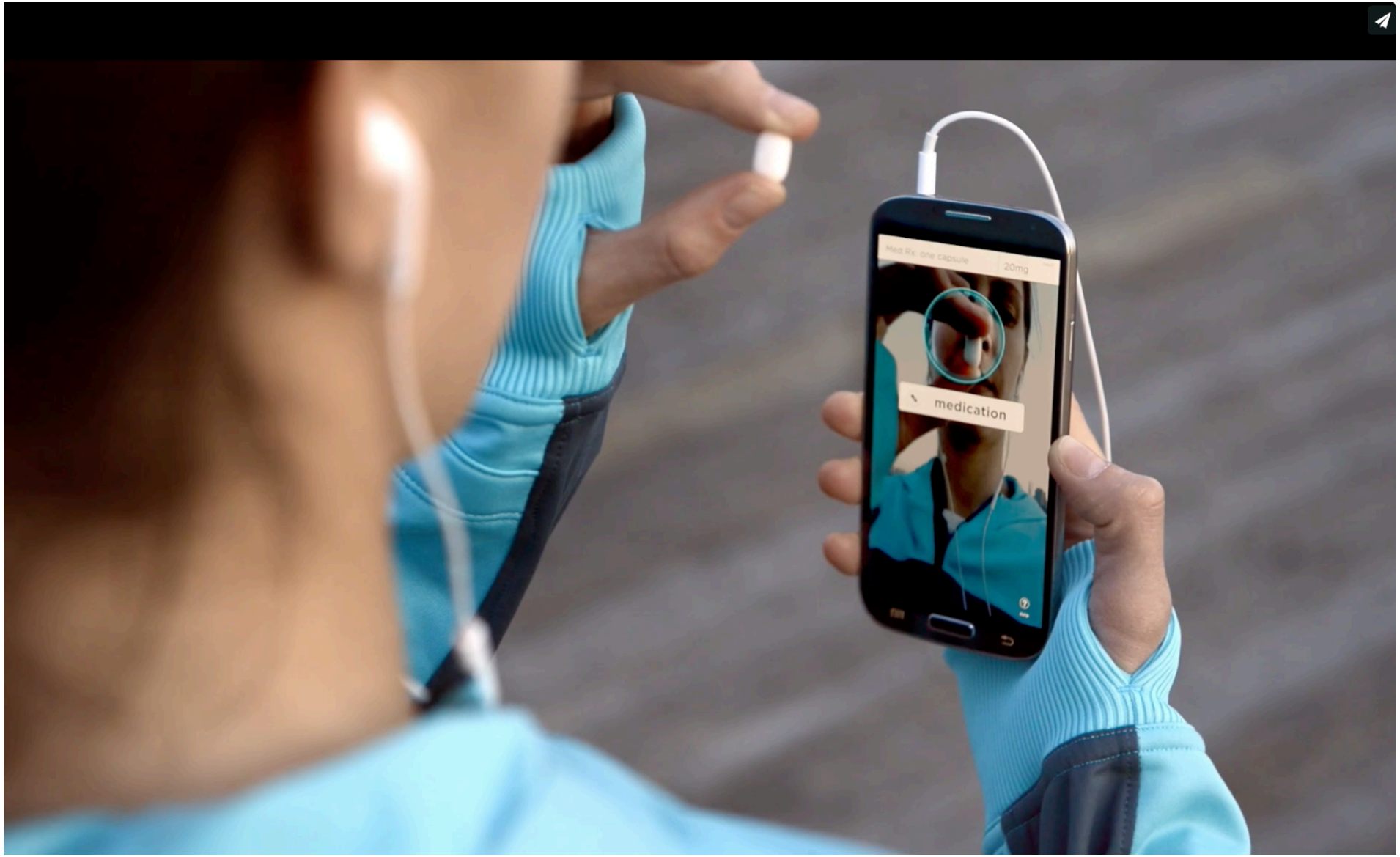
Steps towards get a job

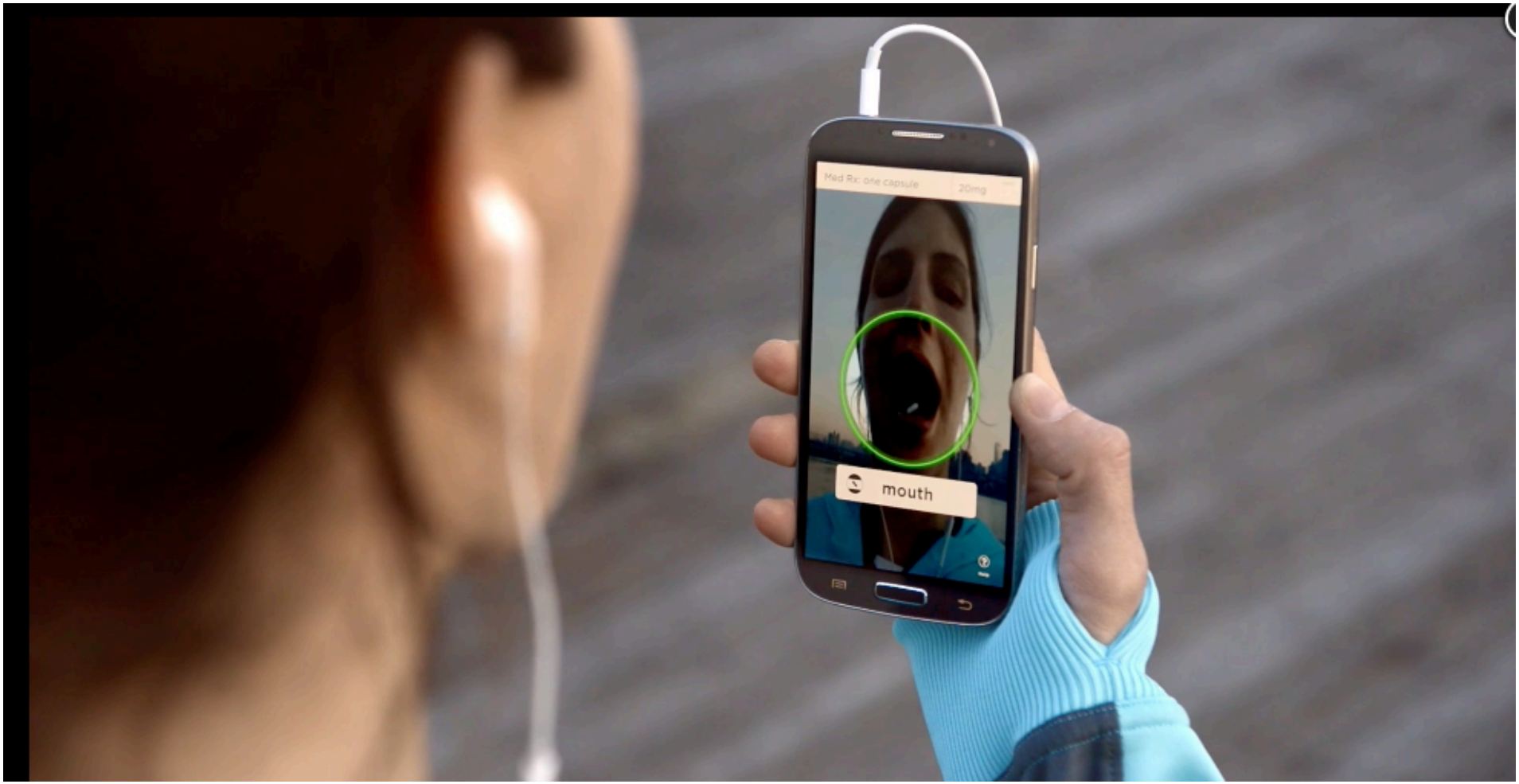
- **Interview process**
  - Estimate Bias-variance for real world problems?
  - Naïve Bayes in SQL?
- **Doing the job**
  - Chief scientist at AiCure?

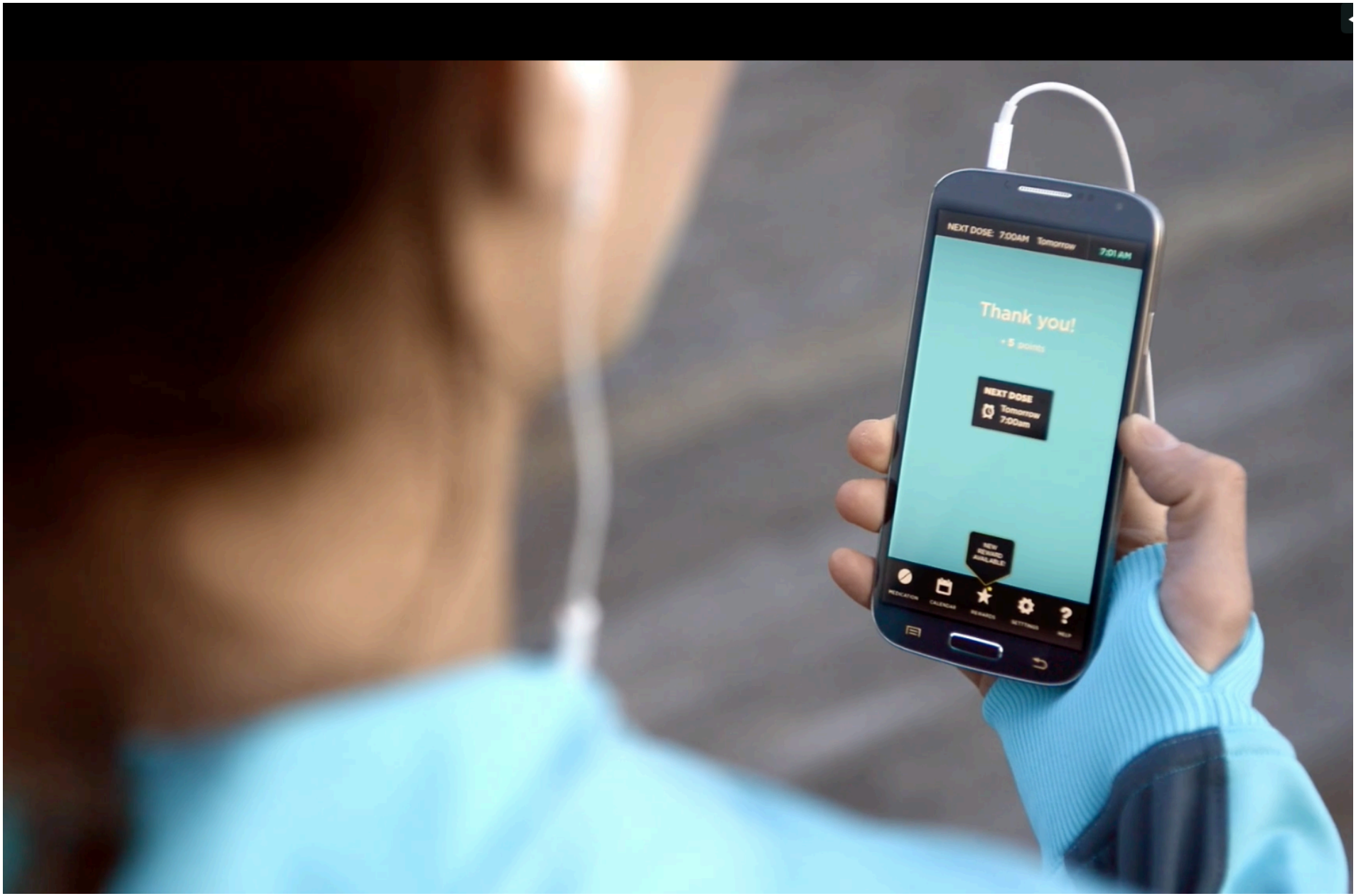
- https://www.aicure.com/

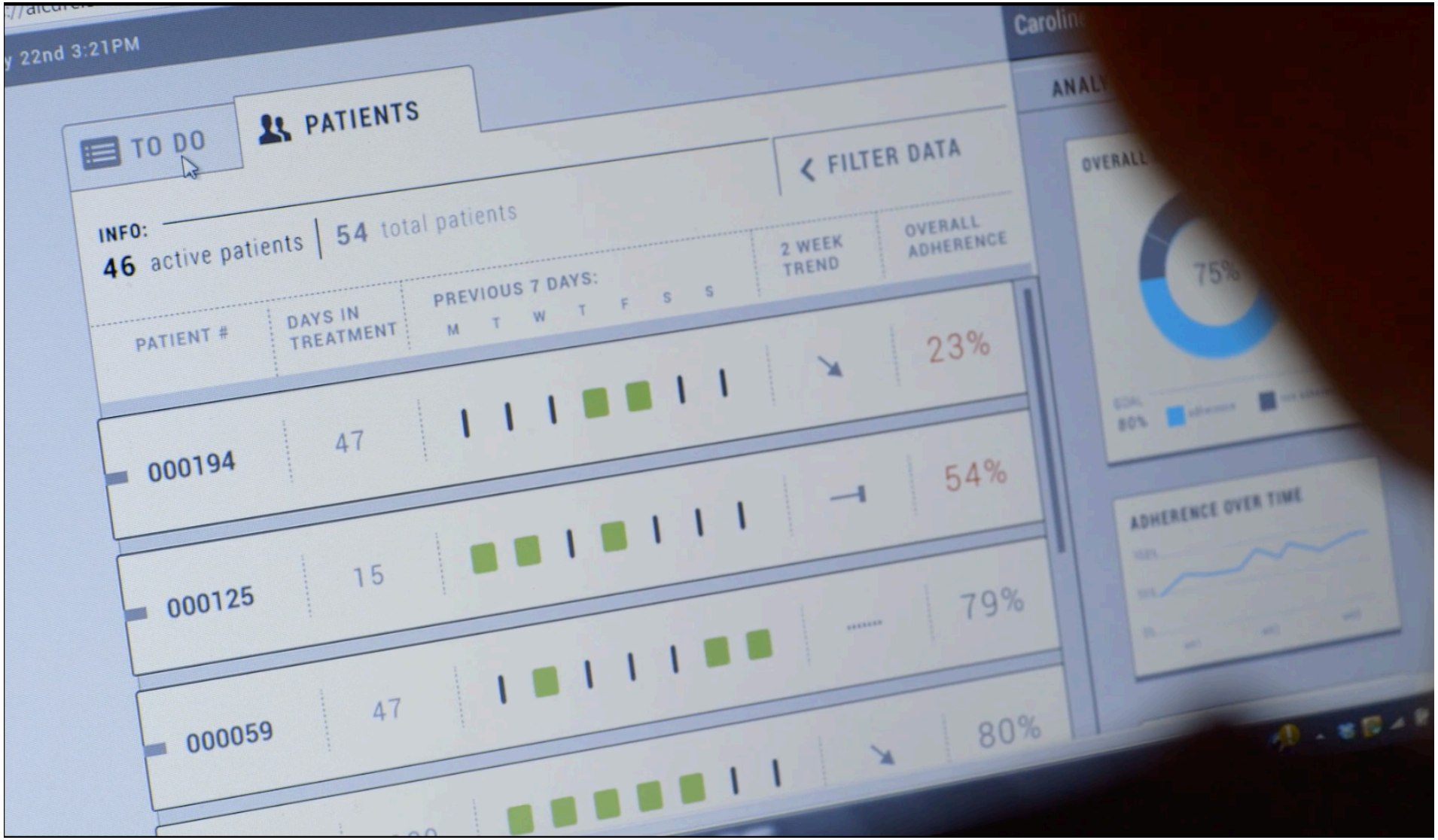

AiCure
solutions for:
Clinical research | Population health | Brands
News About us Careers Contact & Suppor
Do you know if your patients are taking their medication?
We do.

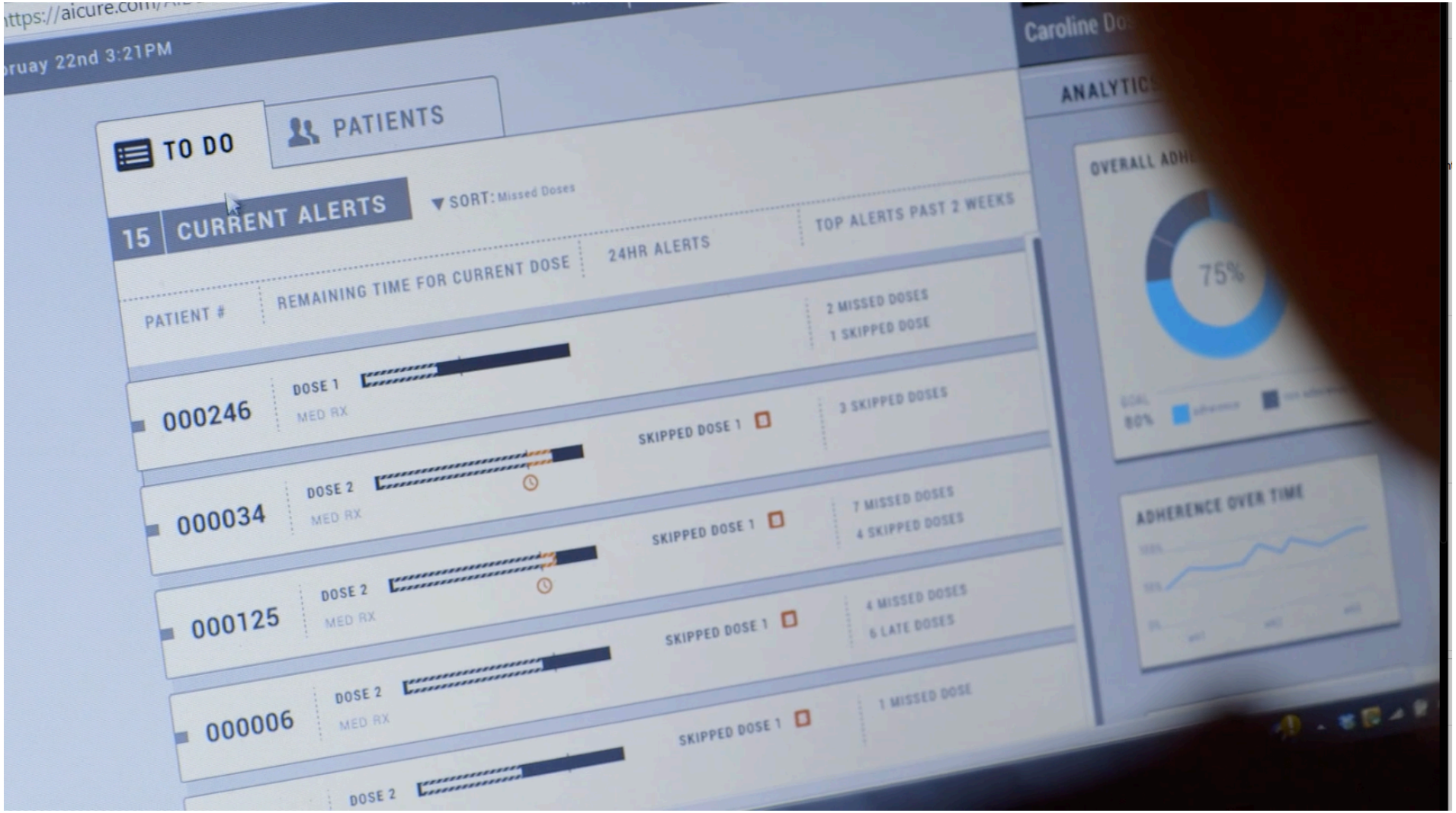

AiView
MEDICATION REMINDER:
7:00 AM
START
SKIP

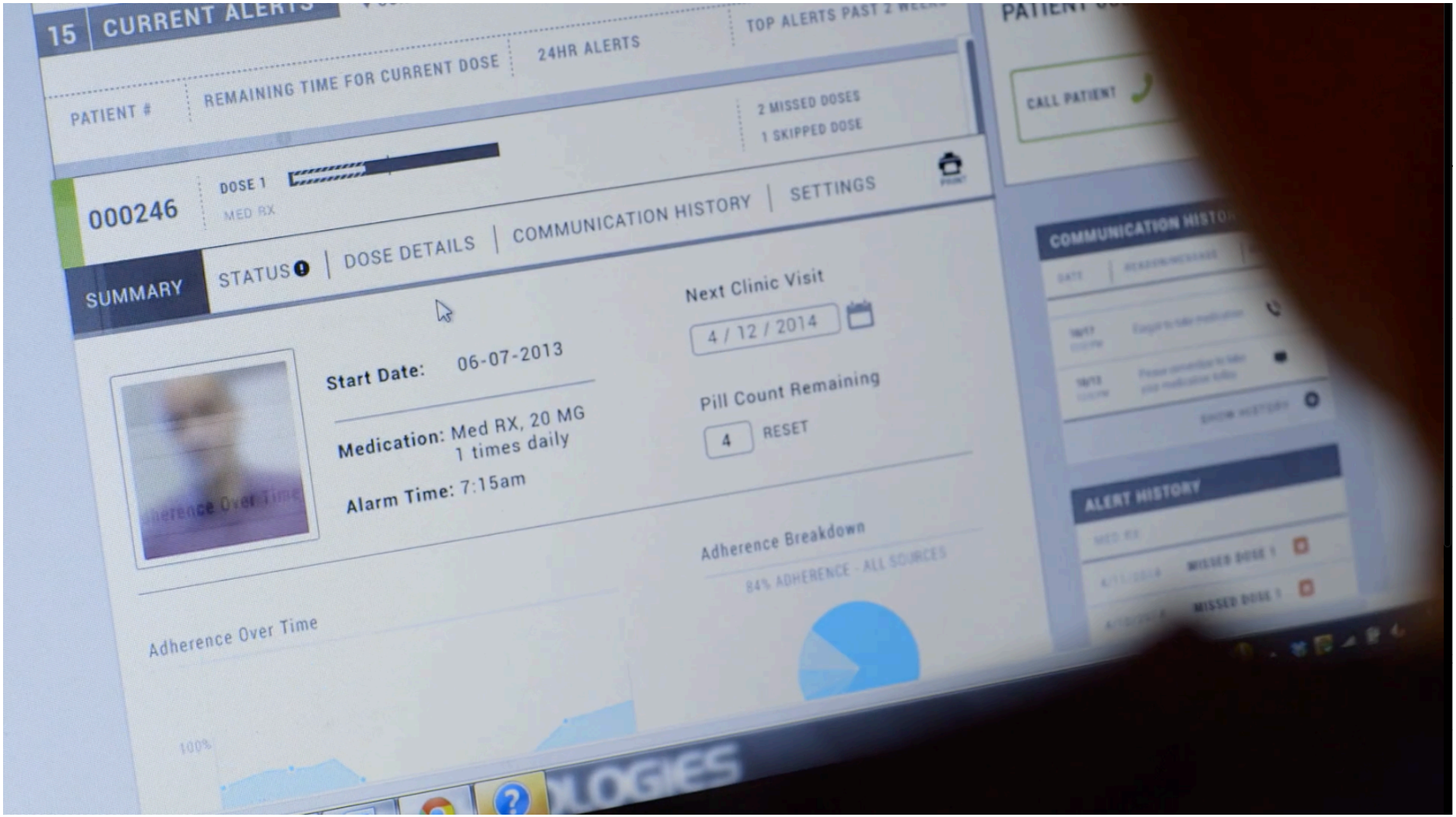

Med Rx: one capsule
20mg
medication


Med Rx: one capsule
20mg
mouth


NEXT DOSE: 7:00AM Tomorrow
7:01 AM
Thank you!
NEXT DOSE
Tomorrow
7:00am


22nd 3:21PM
TO DO
PATIENTS
FILTER DATA
INFO:
46 active patients
54 total patients
PATIENT #
DAYS IN TREATMENT
PREVIOUS 7 DAYS:
M T W T F S S
2 WEEK TREND
OVERALL ADHERENCE
000194
47
23%
000125
15
54%
000059
47
79%
80%
75%
ADHERENCE OVER TIME


TO DO
PATIENTS
15 CURRENT ALERTS
SORT: Missed Doses
PATIENT #
REMAINING TIME FOR CURRENT DOSE
24HR ALERTS
TOP ALERTS PAST 2 WEEKS
000246
DOSE 1
MED RX
2 MISSED DOSES
1 SKIPPED DOSE
000034
DOSE 2
MED RX
SKIPPED DOSE 1
3 SKIPPED DOSES
000125
DOSE 2
MED RX
SKIPPED DOSE 1
7 MISSED DOSES
4 SKIPPED DOSES
000006
DOSE 2
MED RX
SKIPPED DOSE 1
4 MISSED DOSES
6 LATE DOSES
DOSE 2
SKIPPED DOSE 1
1 MISSED DOSE
ANALYTICS
75%
ADHERENCE OVER TIME


15 CURRENT ALERTS
PATIENT #
REMAINING TIME FOR CURRENT DOSE
24HR ALERTS
TOP ALERTS PAST 2
2 MISSED DOSES
1 SKIPPED DOSE
CALL PATIENT
000246
DOSE 1
MED RX
SUMMARY
STATUS
DOSE DETAILS
COMMUNICATION HISTORY
SETTINGS
Start Date: 06-07-2013
Medication: Med RX, 20 MG
1 times daily
Alarm Time: 7:15am
Next Clinic Visit
4 / 12 / 2014
Pill Count Remaining
4
RESET
Adherence Breakdown
84% ADHERENCE - ALL SOURCES
Adherence Over Time
100%
COMMUNICATION HISTO
ALERT HISTORY
MISSED DOSE 1
MISSED DOSE 1

As a chief scientist

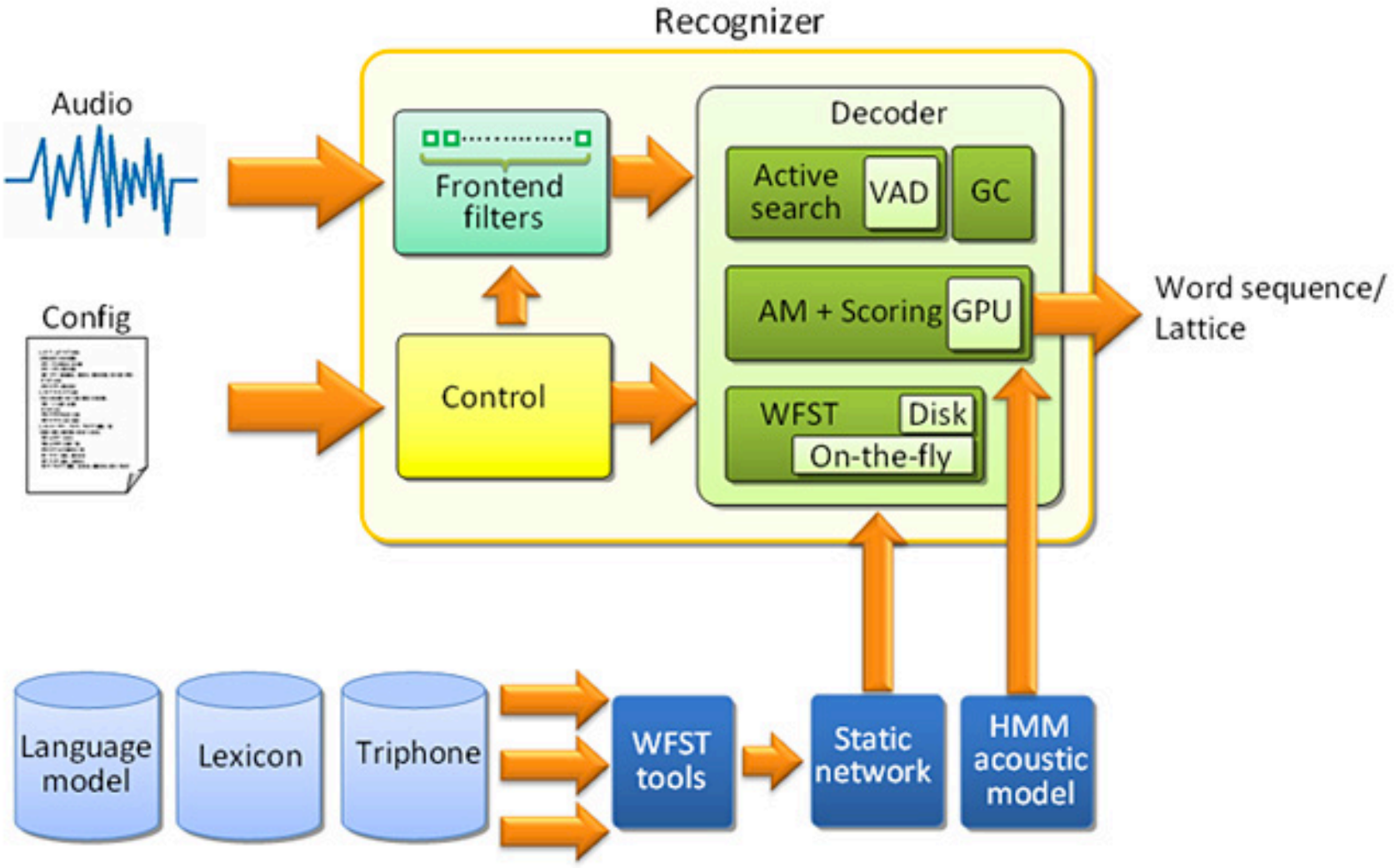
- What would you do?
- Understand what data is collected? Identify anomalies
- Infrastructure
- Classifier for patients who take and don't take
  - Explain using the independent variables
- Instrument and collect right data
- Join other thirdparty data sources (fitbit)
- Proactive communication: Predictive classifiers: detect and notify with reminders (phone call intervention)
- Tracking right person: finger print;
  - Use camera to identify and person
- EDA: 3 times versus 4 times. Side-effects
- Different types of medication: inhaler versus pills (recommendation system); suggest different medical plans
- Data export: ownership
- Measures to secure data
- AB Testing
- Wearables strategy (intelligent pill box)

Speech Recognition Breakthrough for the Spoken, Translated Word

- **Published on Nov 8, 2012**
- **Chief Research Officer Rick Rashid demonstrates a speech recognition breakthrough via machine translation that converts his spoken English words into computer-generated Chinese language. The breakthrough is patterned after deep neural networks and significantly reduces errors in spoken as well as written translation.**
- **For more information on Speech Recognition and Translation, visit**
  - http://www.microsoft.com/translator/skype.aspx
- **Excellent Video (please watch all this video!)**
  - https://www.youtube.com/watch?v=Nu-nlQqFCKg (Minute 7:11)
  - English text (ASR) → Chinese Text → Text to speech system (sound like english speaker)

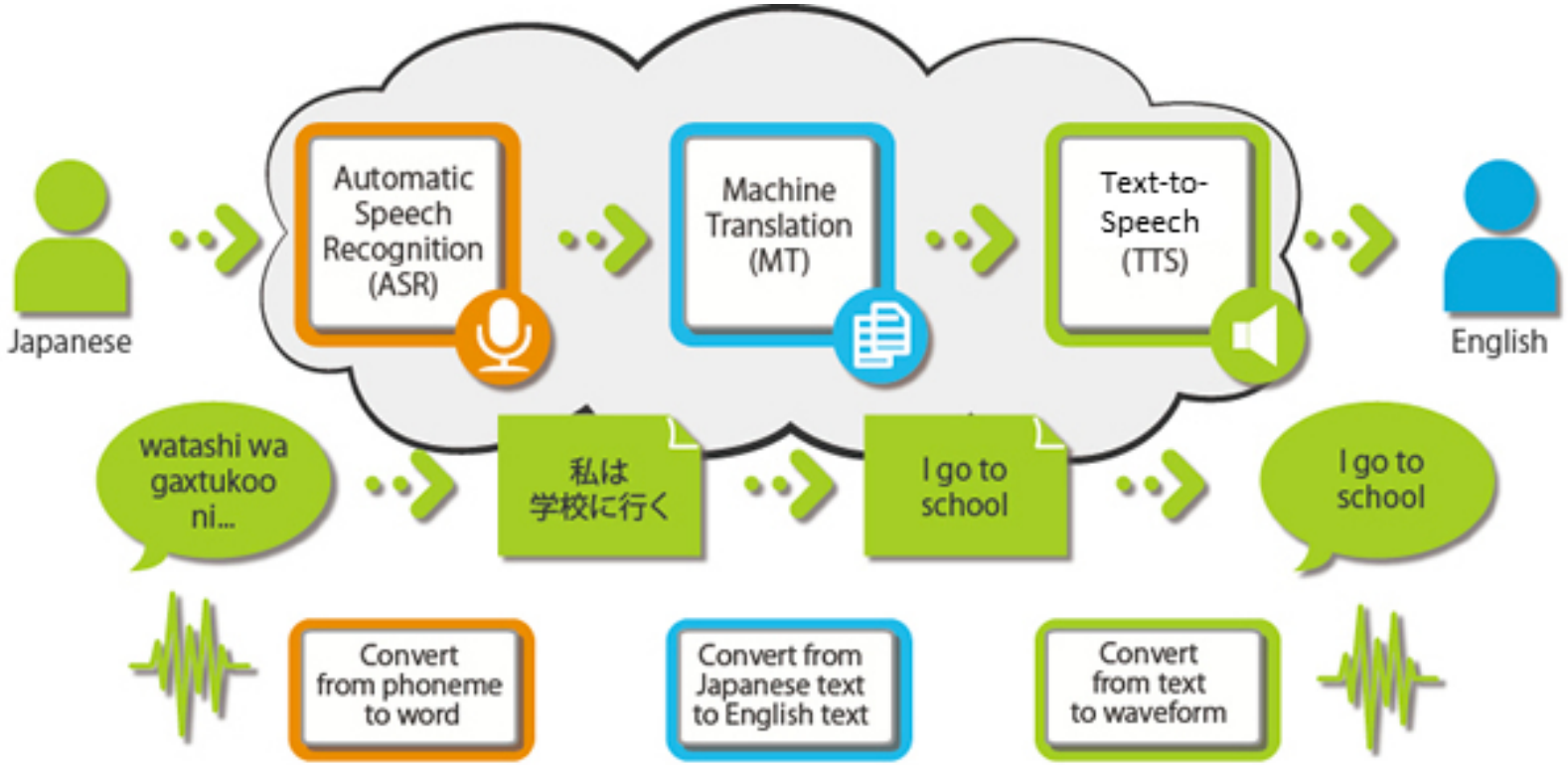

https://www.youtube.com/watch?v=Nu-nlQqFCKg
(4) MIDS-MLS-2015
nbviewer.ipython.org
Stanford Machine Le
Getting Started
Statistical Analysis H
eBay/Google 2013:
Tube Red
English text (ASR) → Chinese Text → Text to speech system (sound like english speaker)
I'm speaking in english and hopefully you'll hear me speaking chinese in my own voice.
Again, the results are not perfect.
They are in fact quite a few
我是说英语，希望你能听我说中国人自己的声音。
再次，结果并不完美。
计算

ASR



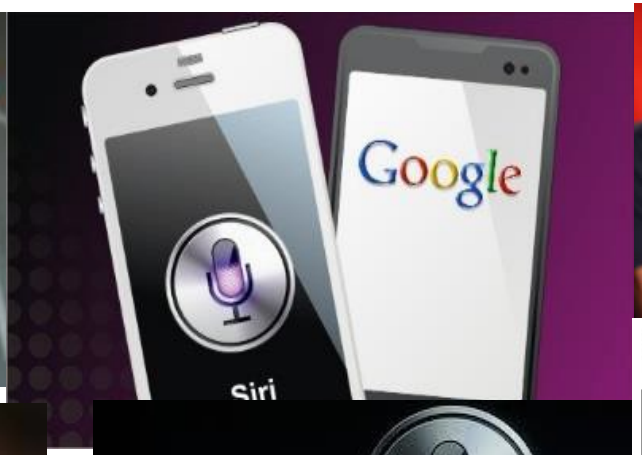
HMM, Deep Learning, Language models

Japanese to English



http://www.ustar-consortium.com/research.html

Impact of deep learning in speech technology



Skype to get 'real-time' translator

Finding Friends

Linking other things such as groups

- **Growing body of research captures dynamics of social network graphs**

  [Latanzi, Sivakumar '08] [Zheleva, Sharara , Getoor '09] [Kumar, Novak, Tomkins '06] [Kossinets, Watts '06] [L., Kleinberg, Faloutsos '05]

- **What links will occur next?** [LibenNowell, Kleinberg '03]
  - **Networks + many other features:**

    Location, School, Job, Hobbies, Interests, etc.

facebook | Search | Home | Profile | Account

Find friends from different parts of your life

Use the checkboxes below to discover people you know from your hometown, school, employer and more.

Hometown
- [ ] Indianapolis, Indiana

Enter another city

Current City
- [ ] Indianapolis, Indiana

Enter another city

High School
- [ ] North Central High School

Enter another high school

Mutual Friend

Enter a name

College or University
- [ ] Martin University

Enter another college

Employer
- [ ] ARIES GRAPHIC DESIGN

Enter another employer


Judy Pyles
36 mutual friends
Add Friend


Rocky Campbell
41 mutual friends
Add Friend

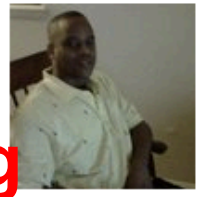

Laura White
12 mutual friends
Add Friend

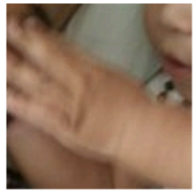

King Ro Conley
59 mutual friends
Add Friend


Dillon Rhodes
43 mutual friends
Add Friend

Rhonda Landrum
54 mutual friends
Add Friend


David Corbitt
90 mutual friends
Add Friend


Eric Bettis
15 mutual friends
Add Friend


Eric Hughes
110 mutual friends
Add Friend


Marki Ann
26 mutual friends
Add Friend


Michael Pugh
21 mutual friends
Add Friend


Lisa Williams
22 mutual friends
Add Friend

Growing


LouieBaur Digg
39 mutual friends
Add Friend


LaTonya Mayberry Bynum
51 mutual friends
Add Friend


Durece Johnson
2 mutual friends
Add Friend


Kendale Adams
64 mutual friends
Add Friend


Bruce T. Caldwell
143 mutual friends
Add Friend


Angela Blackwell Miller
61 mutual friends
Add Friend


Landon Montel

Kevin Brown

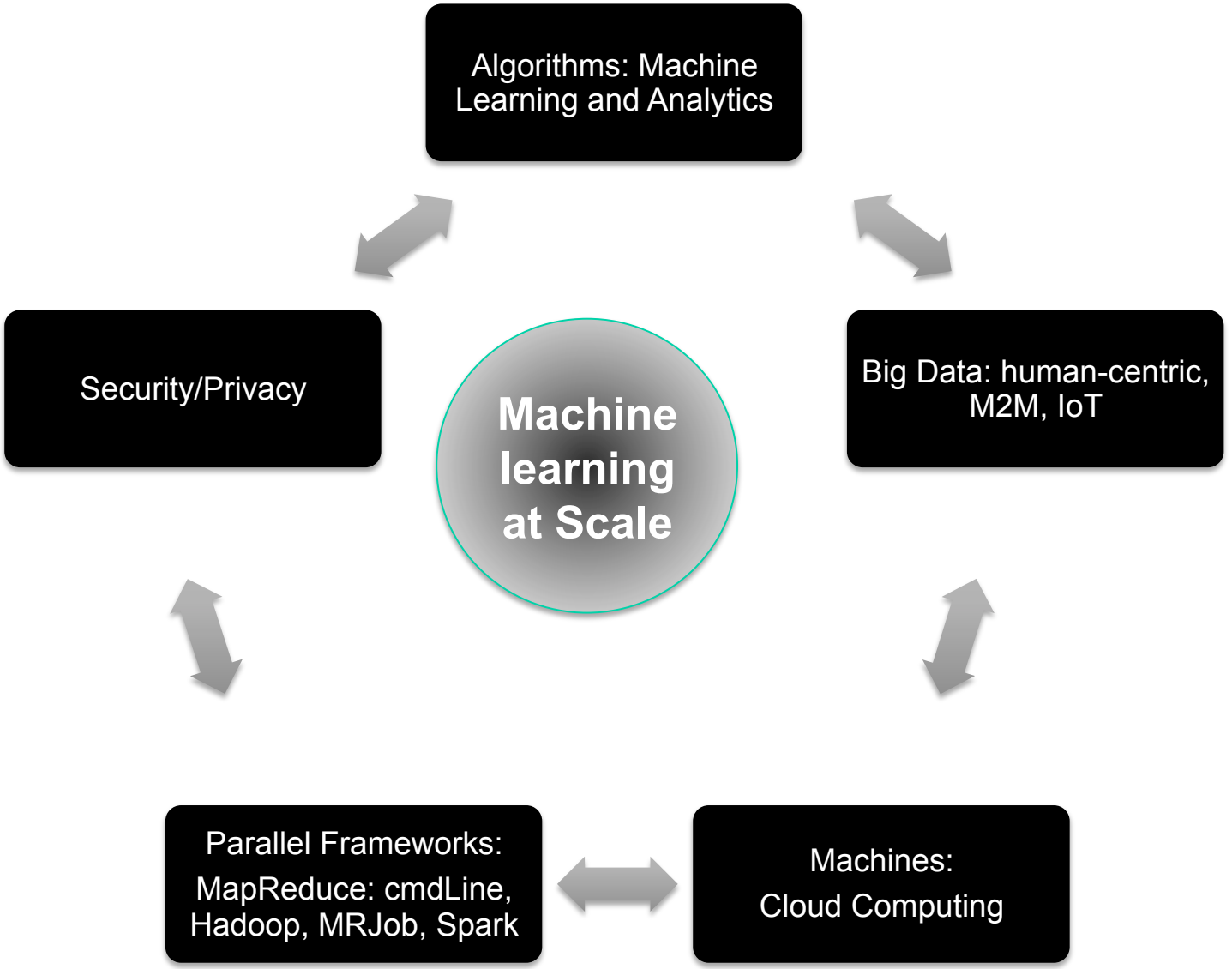

Stanley F. Henry


Saundria Mccrackin


Ebonye X-Endsley


Anita Hawkins

Machine learning at Scale



Core: 3 hours + 7 hours

- **3-4 hours (lectures and live session)**
- **6-7 hours of homework (doing+grading+ learning)**

- **Stretch goals: 10-20 hours**

- **Due dates are important:**
  - Train leaves the station

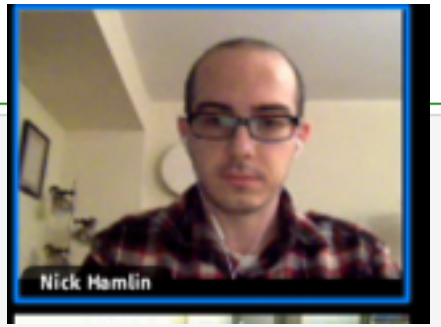
Group based solutions: HW 3 and beyond

- Groups of 3 (try individually; group up)
- One solution per group
- Group people geographically/cohort/group
- Groups will be assigned by instructor

- Same group each; rotations are possible

Weekly Acknowledgement

HW1 Solution, DATSCI W261, Master Solution




```
FirstName LastName
Email address
Time of Initial Submission: 9:21 PM EST, Monday, January 18, 2016
Time of **Resubmission**: 8:38 AM EST, Friday, January 22, 2016
W261-3, Spring 2016
Week 1 Homework

**Acknowledgements**: Jake Ryland Williams, Liang Dai, Nick Hamlin, James G. Shanahan, Patrick Ng, amongst others
```

Useful References:

- Original Assignment Instructions
- Wikipedia explaination of Naive Bayes document classification
- Original paper describing the background of the Enron email corpus
- Documentation for Scikit-Learn implementation of Naive Bayes
- Stanford NLP Group's explaination of Naive Bayes algorithm

#/Users/jshanahan/anaconda/bin/ipython notebook&

HW1.0.0: Define big data. Provide an example of a big data problem in your domain of expertise.

Data too large to fit on or be processed by a single machine or tradional techniques is big. Big data is also well characterized by the four V's: Volume, Variety, Velocity, and Veracity. In short there is no one definition or definiing characteristic of big data, which is exemplified by the range of domains from which one can draw examples. Other possible talking points for this question: PROCESSING: Think of your laptop that gets overwhelmed with 3-4 gig of data (disk space

Humor

- **“I should be stripping”, Glenn Dunmire**
- **Backend engineer**

Acknowledgement

- **HW1 : Notebooks**
  - Jake Ryland Williams, Liang Dai, Nick Hamlin, James G. Shanahan, Patrick Ng, amongst others

- **Formal request**
  - Use your HW submissions as a learning tool
  - Plan to cut and paste good solutions, interesting solutions
    - Use for class discussions
    - Use for master solutions
  - Blanket ask: as for forgiveness more than permission?


Group 4
Professor Shanahan
Samuel Kahn
Marjorie Sayer
Filip Krunic
Glenn Dunmire
Jason Anastasopoulos
Hussein Danish
Thomas Atkins
Tuhin Mahmud
Evyatar Ram
Ron Cordell
Group 2
Professor Shanahan
Kuan Lin
Hetal Chandaria
Patrick Ng
Natarajan Krishnaswami
Michael Kennedy
Vincent Chio
Sharmila Velamur
Lei Yang
Ricardo Barrera
Brandon Shurick
Alan Wang
Benjamin Spooner
Minhchau Dang
Alejandro Rojas
Amin Venjara
Miki Seltzer
Group 3
Professor Shanahan
Tigi Thomas
Angela Gunn
Nick Hamlin
Karthik Chepudira
Linfang Yang
Jing Xu
Happiness Munedzimwe
Safyre Anderson
Konniam Chan
Jaime Villalpando
Howard Wen
Amazing question and fantastic discussion and answers!

Peer Grading

- **First iteration of peer grading for this class**

- **Eureka moments: Deep learning opportunities**
  - Deeper understanding; learn from each other
  - E.g., Bias-Variance
  - Naïve Bayes
- **Visibility into grades**
  - Not just a number.
- **Potential improvements**
  - Can make it a double blind process
    - Graders wont know who they are grading and
    - Gradees wont know who grade them

Feedback on Peer Grading

Homework : HW2 Review

Style Grade [0-100]	HW1.0 Grade [0-100]	HW1.3 Grade [0-100]	General comments or fee	General comments or fee	Feedback on the process
35	65	40	-The link submitted went t -The raw notebook file is -The explanation 1.0.0 dc -1.0.1 has no explanation -Comments are missing tl -1.3 doesn't perform any	For reference, the link provided in the Google sheet was: https://github.com/yanglinfang/W261/blob/master/week1_assignment/week1assignment.ipynb This does not contain the assignment for HW1. In the associated Github repo, we found the following PDF, which https://github.com/yanglinfang/W261/blob/master/week1_assignment/MIDS-W261-2015-HWK-Week01-yanglinfan	
100	95	100	In 1.0.1, the parts on the	I don't feel like we're really qualified to grade 1.0.1.	
100	95	80	HW1.0.1 doesn't address bootstrapping and the lack of true function f(x). In HW1.3, the prediction logic seems to be incorrect.		
100	100	100			
100	100	100	Nicely done, well documented, easy to read. A plea		Very very short on time so the amount of review is very limited.
95	100	90	The outputs of the classifiers and computed error ra		Very time strapped so not much of an in depth review but across the 3 of us it may be ok
50	80	90	Had to take off major style points for completely no output shown, no error rates and no comments. Had to download the notebook and run it		
95	95	88	Style issues: lines are slig HW1.0: Formulas for bias HW1.3: I can see the pric	Jing Yu did a great job!	The process appears great, it is a learning experience for the grader as well

HW 1 Shared learning thru peer grading

- Eureka moments? Please share

- Bias-Variance
- Naïve Bayes

Bias, Variance, and Noise

- **Using a test data set with 20 data points**
- **For each data point x* in the test data set compute variance over the variance predictions (50 models give 50 predictions for each data point x*).**
- **For each data point x* calculate**
  - **Variance**: $E[(h(x^*) - \underline{h(x^*)})^2]$ ## $\Sigma(h(x^*) - \underline{h(x^*)})^2/50$
    - Describes how much h(x*) varies from one training set S to another
  - **Bias**: $[\underline{h(x^*)} - f(x^*)]$
    - Describes the average error of h(x*).
  - **Noise**: $E[(y^* - f(x^*))^2] = E[\varepsilon^2] = \sigma^2$
    - Describes how much y* varies from f(x*)

Bias-Variance written more formally for a single test point x*, using say 20 models

Using a test data set with 20 data points sum $(h(x^*) - y^*)^2$ over each of the 20 points and take the average

$E_D[(h(x^*) - y^*)^2] =$

Expected MSE wrt different models that are learned from different datasets D. E.g., 20 models yield 20 predictions $h_D(x^*)$

$= E_D[(h_D(x^*) - \underline{h(x^*)})^2]$ + *Variance* +

$(\underline{h(x^*)} - f(x^*))^2$ + *$Bias^2$* +

$E[(y^* - f(x^*))^2]$ *$Noise^2$*

$= Var(h(x^*)) + Bias(h(x^*))^2 + E[\varepsilon^2]$

$= Var(h(x^*)) + Bias(h(x^*))^2 + \sigma^2$

Expected prediction error = Variance + $Bias^2$ + $Noise^2$

Estimating Bias and Variance (continued)

- For each data point **x**, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$.
- Compute the average prediction h.
- Estimate **bias** as $(h - y)$
- Estimate **variance** as $\Sigma_k (y_k - h)^2/(K - 1)$
- Assume noise is 0

$h_D(x^*)$ model prediction (assume 20 training datesets)
h(x*) Average model prediction
f(x*) TRUE (Actual function value)
Y* Observed target data (noisy)

http://www-scf.usc.edu/~csci567/17-18-bias-variance.pdf
Excellent Slides from Sofus

Bias-variance trade-off

- Consider fitting a logistic regression LTU to a data set vs. fitting a large neural net.
- Which one do you expect to have higher bias? Higher variance?
- Typically, *bias* comes from not having good hypotheses in the considered class
- *Variance* results from the hypothesis class containing too many hypotheses
- Hence, we are faced with a *trade-off:* choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias.

Source of bias

- Inability to represent certain decision boundaries
  - E.g., linear threshold units, naïve Bayes, decision trees
- Incorrect assumptions
  - E.g, failure of independence assumption in naïve Bayes
- Classifiers that are "too global" (or, sometimes, too smooth)
  - E.g., a single linear separator, a small decision tree.

If the bias is high, the model is *underfitting* the data.

Source of variance

- Statistical sources
  - Classifiers that are "too local" and can easily fit the data
    - E.g., nearest neighbor, large decision trees
- Computational sources
  - Making decision based on small subsets of the data
    - E.g., decision tree splits near the leaves
  - Randomization in the learning algorithm
    - E.g., neural nets with random initial weights
  - Learning algorithms that make sharp decisions can be unstable (e.g. the decision boundary can change if one training example changes)

If the variance is high, the model is *overfitting* the data

Estimating Bias Variance: 2 Cases

- **Case 1: Simulated world**
  - Estimating Bias and Variance in a simulated world where we know the target function
- **Case 2: Real world**
  - Estimating Bias and Variance in the real world where we do NOT know the target function

Case 1: Simulated world

One (1)Test Set
Many training sets

TEST Dataset		prediction for X given the model generated from sample i				
X	**y_true**	**h_star1**	**…**	**h_star_n**	**H_bar**	**Variance**
1.2						
3.4						

Estimating Bias and Variance in a simulated world where

```
# CASE 1 in a simulated world where we know the ground truth
#HW1.0.1 Pseudocode
# Given
# A training data samples of the form (X, yObserved) that is generated from a noisy function fNoisy
# and a test set has the form X, y_true in our artificial world that is generated from
# Running our simulation generates a new column of predictions for each row of the form
#     X, y_true, h_star1, ...h_star_n,
#
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        generate a sample of data from our noisy function
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    #y_true is the vector of true classes in the test_data
    bias=h_bar-y_true
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go through each iteration to compute this
    noise=mean((y_true-h_star)^2) #As with variance, this needs to be calculated across all iterations

choose model that minimizes (bias^2+variance)
```

Measuring Bias and Variance in Practice

- **Measuring Bias and Variance in Practice**
- **There is no true function available so improvise and estimate bias and variance as follows:**

- **Bootstrapping**
  - Get multiple

- **Alternative to boostrapping**
  - If multiple values of y for the same X value
  - Or bucket X values

Measuring Bias and Variance

- In practice (unlike in theory), we have only ONE training set S.
- We can simulate multiple training sets by <u>bootstrap replicates</u>
  - $S' = \{\mathbf{x} \mid \mathbf{x}$ is drawn at random with replacement from $S\}$ and $|S'| = |S|$.

Procedure for Measuring Bias and Variance

- Construct B bootstrap replicates of S (e.g., $B = 200$): $S_1, \ldots, S_B$
- Apply learning algorithm to each replicate S_b to obtain hypothesis h_b
- Let $T_b = S \setminus S_b$ be the data points that do not appear in S_b (out of bag points)
- Compute predicted value $h_b(\mathbf{x})$ for $\mathbf{x} \in T_b$

K times X is in the test set T_b

Estimating Bias and Variance (continued)

- For each data point **x**, we will now have the observed corresponding value y and several predictions $y_1, \ldots, y_K$.
- Compute the average prediction h.
- Estimate **bias** as $(h - y)$
- Estimate **variance** as $\Sigma_k (y_k - h)^2/(K - 1)$
- Assume noise is 0

K times X is in the test set T_b
Assume y the observed value is
Assume Zero noise

Case 2: Bias Variance calculation in practice

TEST Dataset		prediction for X given the model generated from sample i				
X	y_true (yObserved)	h_star1	…	h_star_n	H_bar	Variance
1.2						
3.4						

```
#CASE 2 in the real world when we do NOT know the true target function

#HW1.0.1 Pseudocode
# Given training data of the form (X, yObserved)
# y_true is true value but in most real world problems we do NOT know the ground truth
# So in practice we set y_true =  yObserved and set noise =0
for model in models:
    #this is the bagging step needed to calculate variance
    #where n is some constant (like 50)
    for iteration from 1:n
        Split training data randomly into train_data and test_data
        Train model using train_data
        h_star=predict results for test_data
    h_bar=calculate average prediction across all iterations
    #y_true is the vector of true classes in the test_data
    bias=h_bar-y_true
    variance=sum((h_bar-h_star)^2)/n #in practice, one would need to go through each iteration to compute this
    #noise=mean((y_true-h_star)^2) #As with variance, this needs to be calculated across all iterations
    noise=0 #set the noise =0 since we can NOT estimate it
choose model that minimizes (bias^2+variance)
```

Yobservered, yPredictions(multiple predictions)

Expected prediction error = ***estimator variance*** + ***squared estimator bias*** + ***noise***

Thus the expected prediction error on new data can be used as a quantitative criterion for selecting the best model from a candidate set of estimators! It turns out that, given N new data points $(\mathbf{x}^*, \mathbf{y}^*)$, the expected prediction error can be easily approximated as the mean squared error over data pairs:

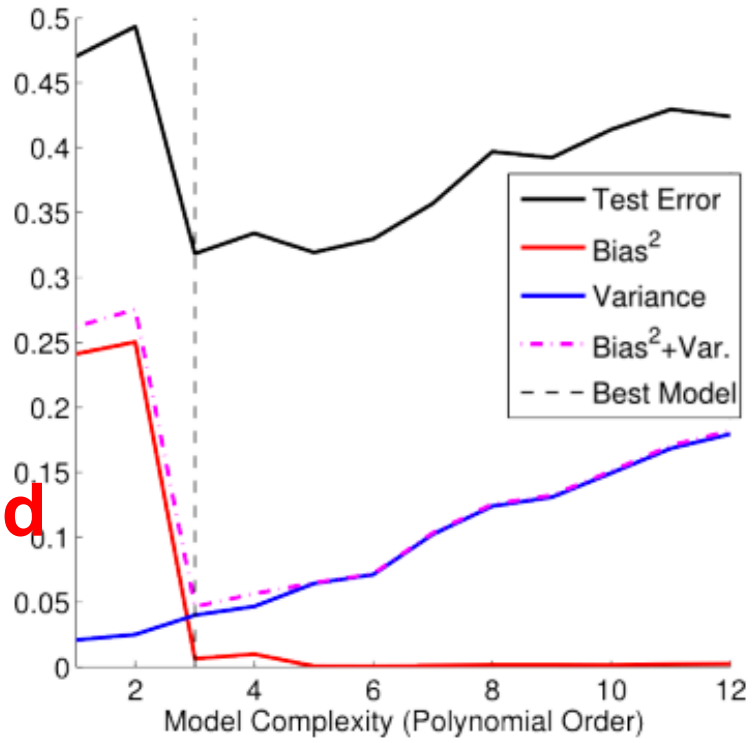
$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$

Below we demonstrate these findings with another set of simulations. We simulate 100 independent datasets, each with 25 xy pairs. We then partition each dataset into two non-overlapping sets: a training set using for fitting model parameters, and a testing set used to calculate prediction error. We then fit the parameters for estimators of varying complexity. Complexity is varied by using polynomial functions that range in model order from 1 (least complex) to 12 (most complex). We then calculate and display the squared bias, variance, and error on testing set for each of the estimators:

```
+ expand source
```

Real world

Simulated world



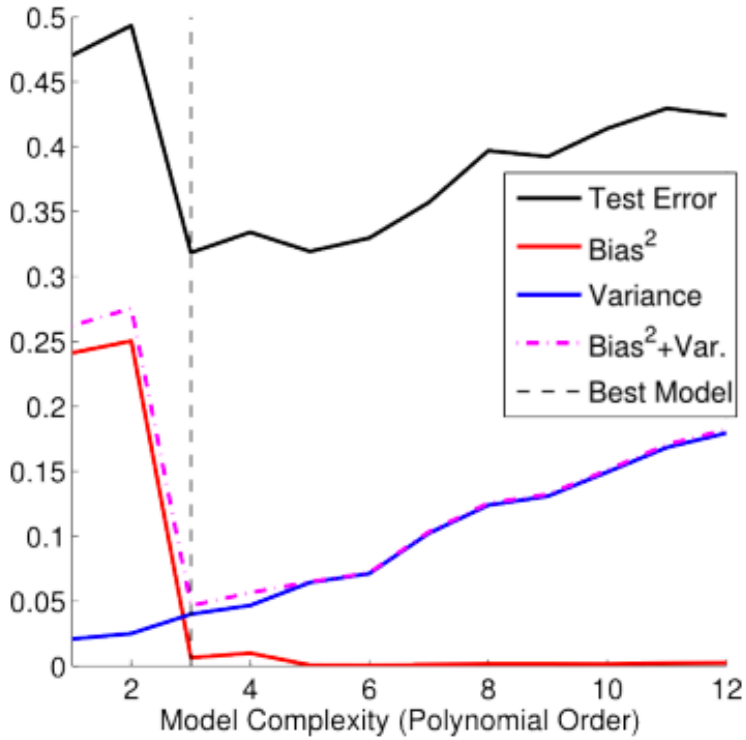
In this example, we highlight the best estimator in terms of prediction error on the testing set (black curve) with a dashed black vertical line. The best estimator corresponds to a polynomial model of order of N=3.
Notice that the vertical black line is located where function defined by the sum of the squared bias and variance (dashed magenta curve) is also at a minimum.
Notice also how the sum of the squared bias and variance also has the same shape as curve defined by the prediction error on the testing set. This exemplifies how the error on novel data can be used as a proxy for determining the best estimator from a candidate set based on squared bias and variance. The noise term in Equation (6) is also represented in the figure by the vertical displacement between the black curve and dashed magenta curve.

It is very important to point out that all of these results are based on evaluating prediction error on novel data, not used to estimate model parameters.

+ expand source

In Practice It turns out that, given N new data points, the expected prediction error E(MSE) can be easily approximated as the mean squared error over a novel test data:

$$\mathbb{E}[(g(\mathbf{x}^*) - \mathbf{y}^*)^2] \approx \frac{1}{N}\sum_{i=1}^{N}(g(x_i^*) - y_i^*)^2$$



Large-Scale Mac

Approximations in this Procedure

- Bootstrap replicates are not real data
- We ignore the noise
  - If we have multiple data points with the same **x** value, then we can estimate the noise
  - We can also estimate noise by pooling y values from nearby **x** values

The Bias-Variance Tradeoff

Given:

- the true function we want to approximate

$$f = f(\mathbf{x})$$

- the data set for training

$$D = \{(\mathbf{x}_1,t_1),(\mathbf{x}_2,t_2),\ldots,(\mathbf{x}_N,t_N)\} \text{ where } t = f + \varepsilon \text{ and } E\{\varepsilon\} = 0$$

- given D, we train an arbitrary neural network to approximate the function f by

$$y = g(\mathbf{x},\mathbf{w})$$

The mean-squared error of this networks is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

To assess the effectiveness of the network, we want to know the expectation of the MSE if we test the network on arbitrarily many test points drawn from the unknown function.

$$E\{MSE\} = E\left\{\frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2\right\} = \frac{1}{N}\sum_{i=1}^{N}E\left\{(t_i - y_i)^2\right\}$$

Let's investigate the expectation inside the sum, with a little "augmentation trick":

$$E\left\{(t_i - y_i)^2\right\} = E\left\{(t_i - f_i + f_i - y_i)^2\right\}$$
$$= E\left\{(t_i - f_i)^2\right\} + E\left\{(f_i - y_i)^2\right\} + 2E\left\{(f_i - y_i)(t_i - f_i)\right\}$$
$$= E\left\{\varepsilon^2\right\} + E\left\{(f_i - y_i)^2\right\} + 2\left(E\left\{f_i t_i\right\} - E\left\{f_i^2\right\} - E\left\{y_i t_i\right\} + E\left\{y_i f_i\right\}\right)$$

Note: $E\{f_i t_i\} = f_i^2$ since f is deterministic and $E\{t_i\} = f_i$

: $E\{f_i^2\} = f_i^2$ since f is deterministic

: $E\{y_i t_i\} = E\{y_i(f_i + \varepsilon)\} = E\{y_i f_i + y_i\varepsilon\} = E\{y_i f_i\} + 0$

: (the last term is zero because the noise in the infinite test set over which

: we take the expectation is probabilistically independent of the NN

: prediction). Thus the last term in the expectation above cancels to zero.

$$E\left\{(t_i - y_i)^2\right\} = E\left\{\varepsilon^2\right\} + E\left\{(f_i - y_i)^2\right\}$$

Bias-Variance

Thus the MSE can be decomposed in expectation into the variance of the noise and the MSE between the true function and the predicted values. This term can be further composed with the same augmentation trick as above.

$$E\left\{(f_i - y_i)^2\right\} = E\left\{\left(f_i - E\{y_i\} + E\{y_i\}_i - y_i\right)^2\right\}$$
$$= E\left\{(f_i - E\{y_i\})^2\right\} + E\left\{(E\{y_i\} - y_i)^2\right\} + 2E\left\{(E\{y_i\} - y_i)(f_i - E\{y_i\})\right\}$$
$$= bias^2 + Var\{y_i\} + 2\left(E\left\{f_i E\{y_i\}\right\} - E\left\{E\{y_i\}^2\right\} - E\{y_i f\}_i + E\left\{y_i E\{y_i\}\right\}\right)$$

Note: $E\{f_i E\{y_i\}\} = f_i E\{y_i\}$ since f is deterministic and $E\{E\{z\}\} = z$

: $E\left\{E\{y_i\}^2\right\} = E\{y_i\}^2$ since $E\{E\{z\}\} = z$

: $E\{y_i f_i\} = f_i E\{y_i\}$

: $E\left\{y_i E\{y_i\}\right\} = E\{y_i\}^2$

: Thus the last term in the expectation above cancels to zero.

$$E\left\{(f_i - y_i)^2\right\} = bias^2 + Var\{y_i\}$$

Thus the decomposition of the MSE in expectation becomes:

$$E\left\{(t_i - y_i)^2\right\} = Var\{noise\} + bias^2 + Var\{y_i\}$$

Note that the variance of the noise can not be minimized; it is independent of the neural network. Thus in order to minimize the MSE, we need to minimize both the bias and the variance. However, this is not trivial to do this. For instance, just neglecting the input data and predicting the output somehow (e.g., just a constant), would definitely minimize the variance of our predictions: they would be always the same, thus the variance would be zero—but the bias of our estimate (i.e., the amount we are off the real function) would be tremendously large. On the other hand, the neural network could perfectly interpolate the training data, i.e., it predict y=t for every data point. This will make the bias term vanish entirely, since the E(y)=f (insert this above into the squared bias term to verify this), but the variance term will become equal to the variance of the noise, which may be significant (see also Bishop Chapter 9 and the Geman et al. Paper). In general, finding an optimal bias-variance tradeoff is hard, but acceptable solutions can be found, e.g., by means of cross validation or regularization.

http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

When would you use Bias-variance?

- **To understand the power of your different learning algorithms/ features**

HW2: Optional Questions

- **Completed?**
- **Optional Question: 2.6.1, 2.7, 2.8**

Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
  - Office hours: Mondays at 5:30PM. Please submit Qs by Sunday at 5:30PM
  - Homework 1&2: Naive Bayes Experts (grades due on Saturday)
- **Peer Grading**
  - HW1 peer grading
  - HW2 Answer reviews
- **Week 3**
  - Hadoop useful stuff/Trivia
    - Blocks sizes
    - Total Sort
    - Understanding Hadoop via Counters
    - Hadoop File passing pattern
    - Number of mappers and reducers
  - Async lecture recap plus Q&A: Pairs/Secondary keys/Inversion pattern
  - Association rule mining via Apriori
- **Next week (Cloud: MrJob)**
- **Wrapup: Finish RECORDING (bonus points!)**

The Relationship Between Input Splits and HDFS Blocks

The logical records that `FileInputFormats` define usually do not fit neatly into HDFS blocks. For example, a `TextInputFormat`'s logical records are lines, which will cross HDFS boundaries more often than not. This has no bearing on the functioning of your program—lines are not missed or broken, for example—but it's worth knowing about because it does mean that data-local maps (that is, maps that are running on the same host as their input data) will perform some remote reads. The slight overhead this causes is not normally significant.

Figure 7-3 shows an example. A single file is broken into lines, and the line boundaries do not correspond with the HDFS block boundaries. Splits honor logical record boundaries, in this case lines, so we see that the first split contains line 5, even though it spans the first and second block. The second split starts at line 6.

Hadoop Trivia

Maps work on data that is stored but occasionally it will have to do remote reads to grab parts of a record that was cut in two at chunking time (BlockSize = 64M).

hadoop splits records during chunking

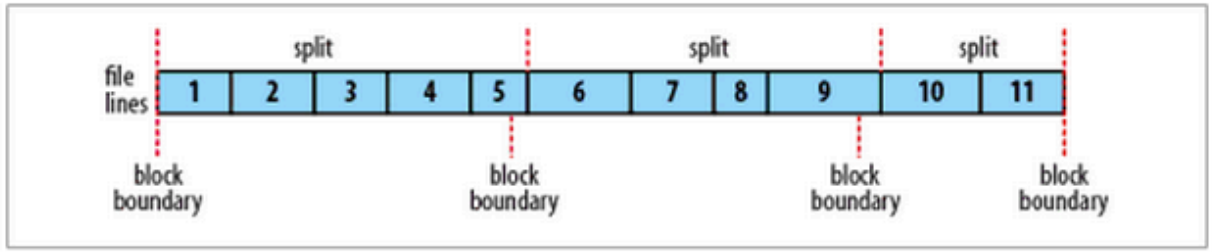
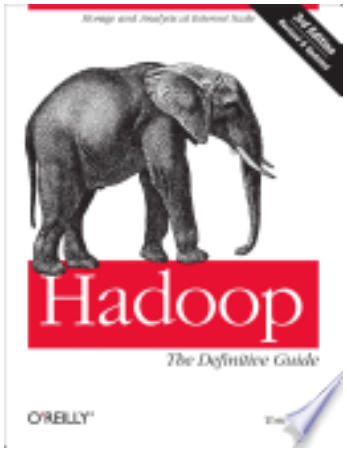


Figure 7-3. Logical records and HDFS blocks for TextInputFormat



Total Sorting in Hadoop

- How can you produce a globally sorted file using Hadoop?

Total sort: How can you produce a globally sorted file using Hadoop?

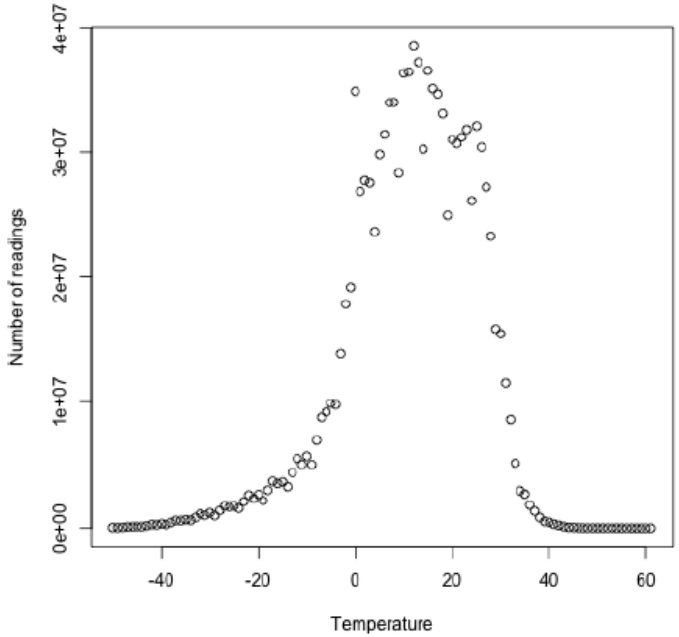
- **One Reducer: The naive answer is to use a single partition.**
  - But this is incredibly inefficient for large files because one machine has to process all of the output, so you are throwing away the benefits of the parallel architecture that MapReduce provides.
- **Multiple Reducers:**
  - Use a partitioner that respects the total order of the output
  - Instead, it is possible to produce a set of sorted files that, if concatenated, would form a globally sorted file.
    - The secret to doing this is to use a partitioner that respects the total order of the output.
    - For example, for a temperature data set, if we had four partitions, we could put keys for temperatures:
      - less than –10C in the first partition,
      - those between –10C and 0C in the second,
      - those between 0C and 10C in the third, and
      - those over 10C in the fourth.

Imbalanced data is common

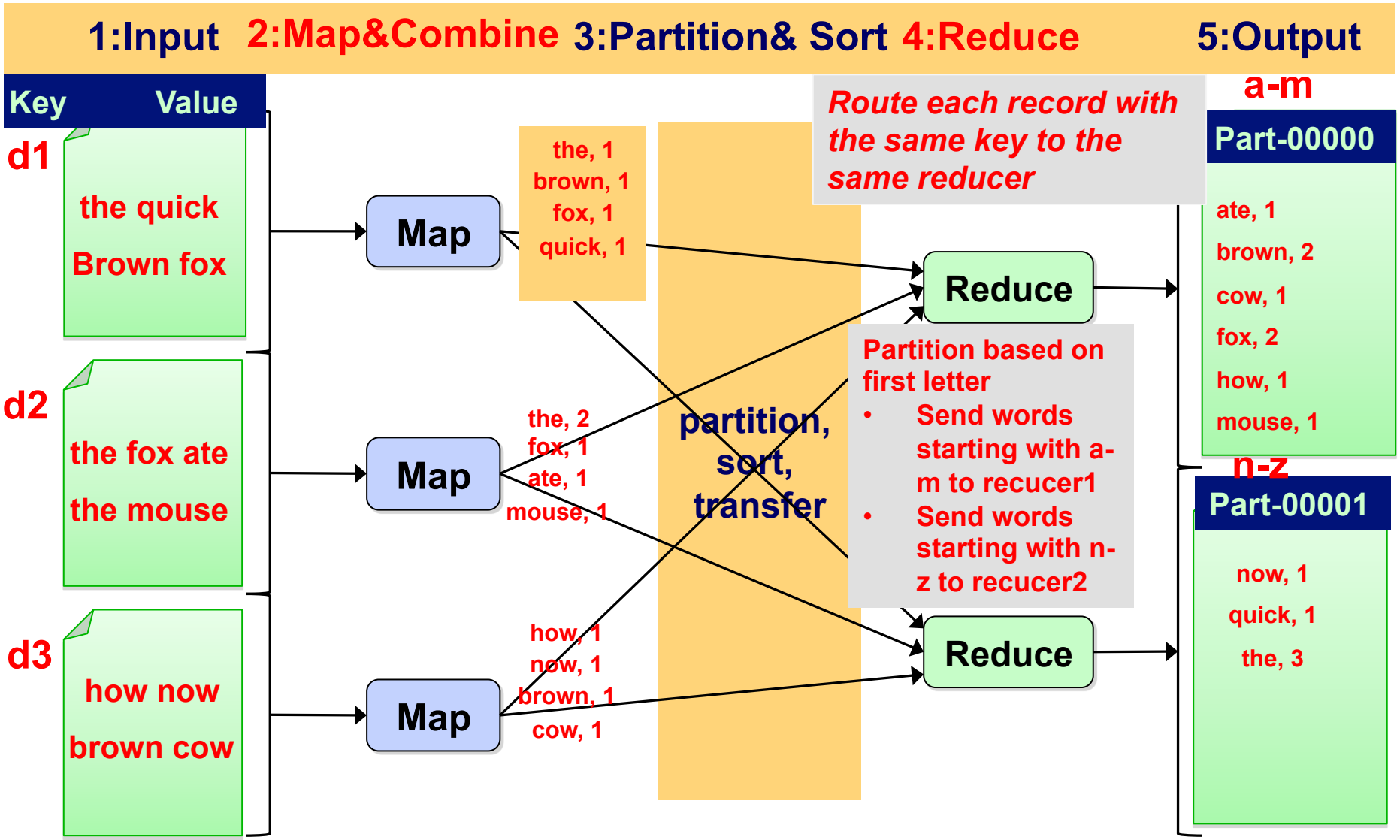
- These partitions are not very even. To construct more even partitions, we need to have a better understanding of the temperature distribution for the whole dataset

Temperature range	< –10°C	[–10°C, 0°C)	[0°C, 10°C)	>= 10°C
Proportion of records	11%	13%	17%	59%

- Temperature distribution for the weather dataset (very skewed; not uniform)



Word Count Workflow: Total Sort



Sample the key space to construct the partition

- It's possible to get a fairly even set of partitions by sampling the key space. The idea behind sampling is that you look at a small subset of the keys to approximate the key distribution, which is then used to construct partitions.
- Luckily, we don't have to write the code to do this ourselves, as Hadoop comes with a selection of samplers.

- One of the nice properties of InputSampler and TotalOrderPartitioner is that you are free to choose the number of partitions; that is, the number of reducers.
- However, TotalOrderPartitioner will work only if the partition boundaries are distinct.
- One problem with choosing a high number is that you may get collisions if you have a small key space.

4.2.2 Total order sorting

You'll find a number of situations where you'll want to have your job output in total sort order. For example, if you want to extract the most popular URLs from a web graph you'll have to order your graph by some measure of popularity, such as PageRank. Or if you want to display a table in your portal of the most active users on your site, you need the ability to sort them based on some criteria such as the number of articles they wrote.

TECHNIQUE 22 **Sorting keys across multiple reducers**

We all know that the MapReduce framework sorts map output keys prior to feeding them to reducers. This sorting is only guaranteed within each reducer, and unless you specify a partitioner for your job, you will be using the default MapReduce partitioner, `HashPartitioner`, which partitions using a hash of the map output keys. This ensures that all records with the same map output key go to the same reducer, but the `HashPartitioner` doesn't perform total sorting of the map output keys across all the reducers. Knowing this, you may be wondering how you could use MapReduce to sort keys across multiple reducers so that you can easily extract the top and bottom *N* records from your data.

Problem

You want a total ordering of keys in your job outputs, but without the overhead of having to run a single reducer.

Solution

This technique covers use of the `TotalOrderPartitioner` class, a partitioner that is bundled with Hadoop, to assist in sorting output across all reducers. The partitioner ensures that output sent to the reducers is totally ordered, so as long as the reducer emits the same output key as the input key, total job output is guaranteed.

Discussion

Hadoop has a built-in partitioner called the `TotalOrderPartitioner`, which distributes keys to specific reducers based on a partition file. The partition file is a precomputed SequenceFile that contains *N*-1 keys, where *N* is the number of reducers. The keys in the partition file are ordered by the map output key comparator, and as such each key represents a logical range of keys. To determine which reducer should receive an output record, the `TotalOrderPartitioner` examines the output key, determines which range it falls into, and maps that range into a specific reducer.

Figure 4.15 shows the two parts of this technique. You need to create the partition file and then run your MapReduce job using the `TotalOrderPartitioner`.

First you'll use the `InputSampler` class, which samples the input files and creates the partition file. You can use one of two samplers, the `RandomSampler` class, which as the name

- ..

Source Code

http://www.javased.com/?source_dir=hadoop-book_1/src/main/java/com/manning/hip/ch4/sort/total/TotalSortMapReduce.java

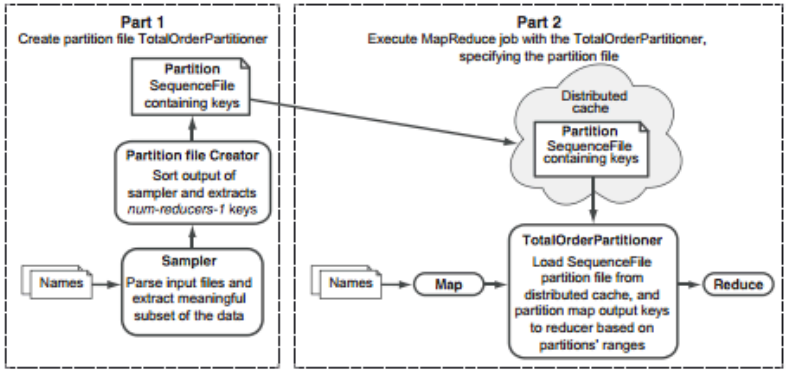


Figure 4.15 Using sampling and the `TotalOrderPartitioner` to sort output across all reducers

ggests picks random records from the input, and the `IntervalSampler` class, which for ery *R* record includes the record in the sample. Once the samples have been tracted, they're sorted and then *N*-1 keys are written to the partition file, where *N* is number of reducers. The `InputSampler` isn't a MapReduce job; it reads records from `InputFormat` and produces the partition within the process calling the code. following code shows the steps you need to execute prior to calling the `InputSampler` nction:[14]

```
int numReducers = 2;
Path input = new Path(args[0]);
Path partitionFile = new Path(args[1]);

InputSampler.Sampler<Text, Text> sampler =
    new InputSampler.RandomSampler<Text,Text>
        (0.1,
        10000,
        10);

JobConf job = new JobConf();

job.setNumReduceTasks(numReducers);

job.setInputFormat(KeyValueTextInputFormat.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

TotalOrderPartitioner.setPartitionFile(job, partitionFile);

FileInputFormat.setInputPaths(job, input);

InputSampler.writePartitionFile(job, sampler);
```

The probability that a key will be picked from the input.

The number of samples to extract from the input.

The maximum number of input splits that will be read to extract the samples.

Run the InputSampler code to sample and create the partition file. This code uses all the items set in the JobConf object to perform this task.

Set the number of reducers (which is used by the InputSampler when creating the partition file).

Set the InputFormat for the job, which the InputSampler uses to retrieve records from the input.

You also need to specify the map output key and value classes, even if the InputFormat explicitly types them.

Specify the location of the partition file.

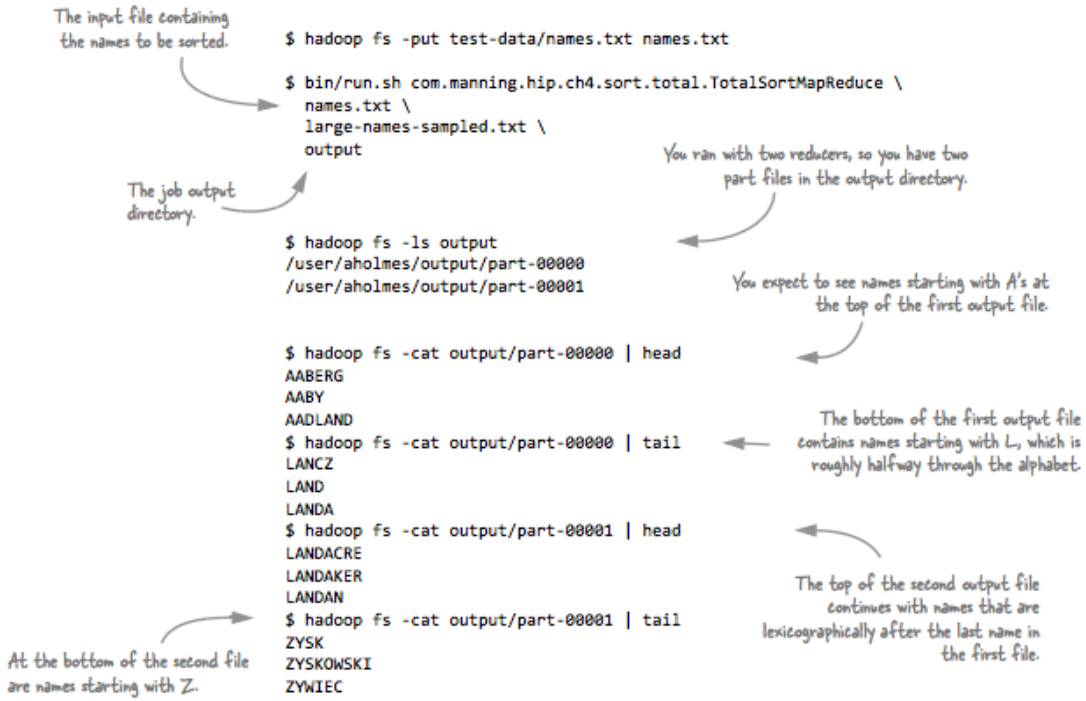
Set the job input files.

[14] **GitHub source**—https://github.com/alexholmes/hadoop-book/blob/master/src/main/java/com/manning/hip/ch4/sort/total/TotalSortMapReduce.java

Next up is specifying that you want to use the `TotalOrderPartitioner` as the partitioner for your job:

```
job.setPartitionerClass(TotalOrderPartitioner.class);
```

For this technique you don't want to do any processing in your MapReduce job, so you'll not specify the map or reduce classes. This means the identity MapReduce classes will be used, so you're ready to run the code:



You can see from the results of the MapReduce job that the map output keys are indeed sorted across all the output files.

Summary

This technique used the `InputSampler` to create the partition file, which is subsequently used by the `TotalOrderPartitioner` to partition map output keys.

You could also use MapReduce to generate the partition file. An efficient way of doing this would be to write a custom `InputFormat` class that performs the sampling, and then output the keys to a single reducer, which in turn can create the partition file. This brings us to sampling, the last section of this chapter.

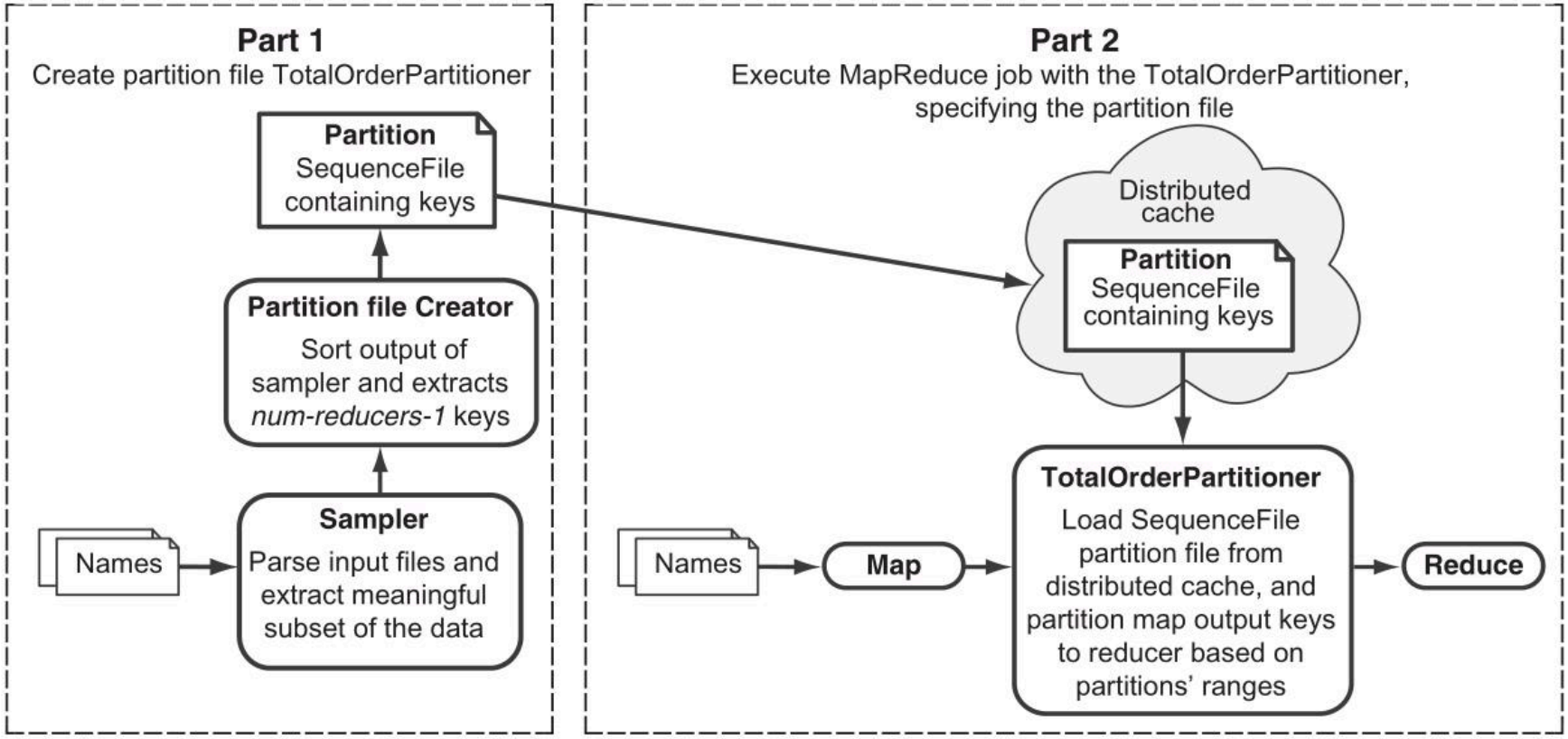


Figure 4.15 **Using sampling and the `TotalOrderPartitioner` to sort output across all reducers**

4.3 Sampling

Imagine you're working with a terabyte-scale dataset and you have a MapReduce application you want to test with that dataset. Running your MapReduce application against the dataset may take hours, and constantly iterating with code refinements and rerunning against it isn't an optimal workflow.

To solve this problem you look to sampling, which is a statistical methodology for extracting a relevant subset of a population. In the context of MapReduce, sampling provides an opportunity to work with large datasets without the overhead of having to wait for the entire dataset to be read and processed. This greatly enhances your ability to quickly iterate when developing and debugging MapReduce code.

TECHNIQUE 23 Reservoir sampling

You need to iterate over the development of a MapReduce job, and have a large dataset that you want to work with to iteratively test with. Working with the entire dataset takes a long time, and impedes your ability to rapidly work with your code.

Problem

You want to work with a small subset of a large dataset during the development of a MapReduce job.

Solution

You'll write an input format that can wrap the actual input format used to read data. The input format that you'll write can be configured with the number of samples that should be extracted from the wrapped input format.

Discussion

In this technique you'll use reservoir sampling[15] to choose samples. Reservoir sampling is a strategy that allows a single pass through a stream to randomly produce a sample. As such it's a perfect fit for MapReduce because input records are streamed from an input source. Figure 4.16 shows the algorithm for reservoir sampling.

Step 1: Fill the reservoir until it is full.

Step 2: Randomly replace a sample in the reservoir.

```
1: Samples ← ∅
2: i ← 0
3: for all record ∈ largeDataSet do
4:     if |Samples| ≠ requiredSamples then
5:         Samples[i] = record
6:     else
7:         j ← random(1, i) {random number between 1 and i inclusive}
8:         if j <= requiredSamples then
9:             Samples[j] = record
10:    i ← i + 1
```

Figure 4.16 The reservoir sampling algorithm allows one pass through a stream to randomly produce a sample.

[15] See http://en.wikipedia.org/wiki/Reservoir_sampling.

- http://stackoverflow.com/questions/23644545/using-totalorderpartitioner-in-hadoop-streaming

Counters in MrJob/Hadoop

James G. Shanahan

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Large Scale Machine Learning

MIDS, UC Berkeley

Lecture 4, April, 2015

What is counter in Hadoop MapReduce?

- **Counters are lightweight objects in Hadoop that allow you to keep track of system progress in both the map and reduce stages of processing.**

Why counter is useful?

- It provides a way to track the occurrences of global events within the map and reduce phases of jobs, e.g. how many IO operations during a MapReduce task?

- It is one of the easiest way to investigate internal behaviors of MapReduce programs.

Very useful in debugging and tracking resources!

Counters

- **See chapter 9 in "Hadoop The definitive Guide 4th edition"**

- **MrJob Notebook**
  - http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/5thl14n4pqvhzt5/Counter.ipynb

CHAPTER 9

MapReduce Features

This chapter looks at some of the more advanced features of MapReduce, including counters and sorting and joining datasets.

Counters

There are often things you would like to know about the data you are analyzing but that are peripheral to the analysis you are performing. For example, if you were counting invalid records and discovered that the proportion of invalid records in the whole dataset was very high, you might be prompted to check why so many records were being marked as invalid—perhaps there is a bug in the part of the program that detects invalid records? Or if the data were of poor quality and genuinely did have very many invalid records, after discovering this, you might decide to increase the size of the dataset so that the number of good records was large enough for meaningful analysis.

Counters are a useful channel for gathering statistics about the job: for quality control or for application-level statistics. They are also useful for problem diagnosis. If you are tempted to put a log message into your map or reduce task, it is often better to see whether you can use a counter instead to record that a particular condition occurred. In addition to counter values being much easier to retrieve than log output for large distributed jobs, you get a record of the number of times that condition occurred, which is more work to obtain from a set of logfiles.

Built-in Counters

Hadoop maintains some built-in counters for every job, and these report various metrics. For example, there are counters for the number of bytes and records processed, which allows you to confirm that the expected amount of input was consumed and the expected amount of output was produced.

User-Defined Streaming Counters

A Streaming MapReduce program can increment counters by sending a specially formatted line to the standard error stream, which is co-opted as a control channel in this case. The line must have the following format:

```
reporter:counter:group,counter,amount
```

This snippet in Python shows how to increment the "Missing" counter in the "Temperature" group by one:

```
sys.stderr.write("reporter:counter:Temperature,Missing,1\n")
```

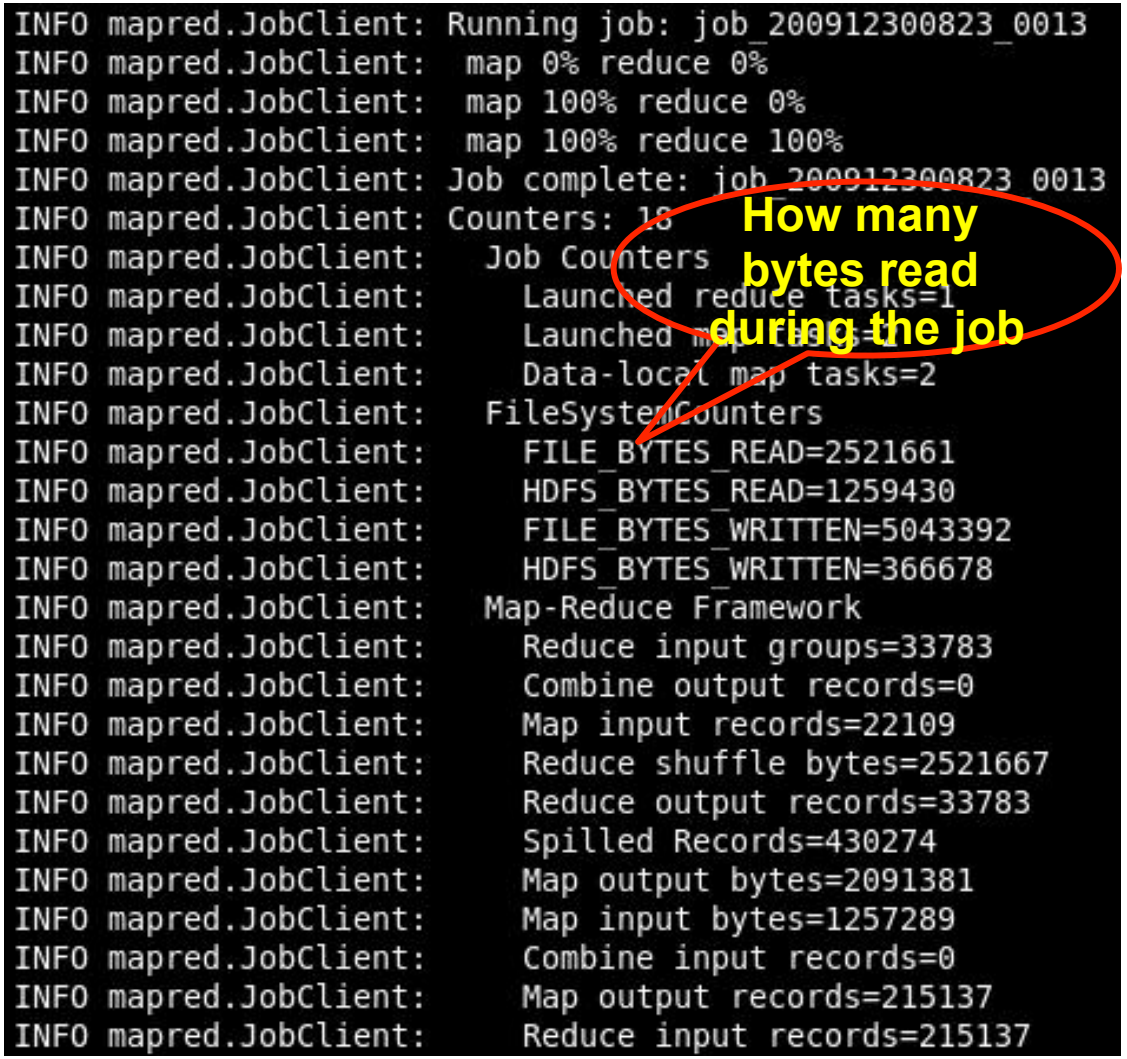
In a similar way, a status message may be sent with a line formatted like this:

```
reporter:status:message
```

Counters in hadoop

- **Internal Counters:**

Even if you have never defined any counters in Hadoop, you can see some of them each time you are running an Hadoop job.



3 Steps to customize counter in Java

- **You can also easily define your own counter in Hadoop.**

Step 1: define an enum that contains a list of the counters you'd like to have:

```
protected static enum MyCounter {
   INPUT_WORDS
};
```

Step 2: in the mapper or reducer, update the counter via the reporter object:

```
reporter.incrCounter(MyCounter.INPUT_WORDS, 1);
```

Step 3: Access the counters in the following manner

```
RunningJob job = JobClient.runJob(conf);  // blocks until job
completes
Counters c = job.getCounters();
long cnt = c.getCounter(MyCounter.INPUT_WORDS);
```

Hadoop Streaming

Reporting in Streaming

- Streaming code can increment counters and update statuses
- Write string to standard error in "streaming reporter" format
- To increment a counter:

```
reporter:counter:<counter_group>,<counter>,<increment_by>
```

Word Count in Hadoop Streaming in Python

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token: print token[0] + '\t1'
```

1. Read one line at a time from standard input

2. tokenize

3. Emit first letter, tab, then a count of 1

Mapper

```
#!/usr/bin/python

import sys

(lastKey, sum)=(None, 0)

for line in sys.stdin:
    (key, value) = line.strip().split("\t")

    if lastKey and lastKey != key:
        print lastKey + '\t' + str(sum)
        (lastKey, sum) = (key, int(value))
    else:
        (lastKey, sum) = (key, sum + int(value))

if lastKey:
    print lastKey + '\t' + str(sum)
```

Variables to manage key group boundaries

Process one line at a time by reading from standard input

If key is different emit current group and start new

Reducer

Word Count with internal diagnostics via counters

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token: print token[0] + '\t1'
```

1. Read one line at a time from standard input

2. tokenize

3. Emit first letter, tab, then a count of 1

countMap_withReporting.py

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    for token in line.strip().split(" "):
        if token:
            sys.stderr.write("reporter:counter:Tokens,Total,1\n")
            print token[0] + '\t1'
```

Print counter information in "reporter protocol" to standard error

- ..

```
yarn jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
        -D mapred.job.name="Count Job via Streaming" \
        -files $HADOOP_SAMPLES_SRC/scripts/countMap.py,\
                        $HADOOP_SAMPLES_SRC/scripts/countReduce.py \
        -input /training/data/hamlet.txt \
        -output /training/playArea/wordCount/ \
        -mapper countMap.py \
        -combiner countReduce.py \
        -reducer countReduce.py
```

Name the job

-files options makes scripts available on the cluster for MapReduce

Example of counter customization in Mrjob

- **Count how many times mapper and reducer function are called**

MrJob class for Counter

```
1  %%writefile mr_wc_counter.py
2  from mrjob.job import MRJob
3  from mrjob.step import MRJobStep
4  import re
5
6  WORD_RE = re.compile(r"[\w']+")
7
8  class MRWordFreqCount(MRJob):
9      def mapper(self, _, line):
10         self.increment_counter('group', 'Num_mapper_calls', 1)
11         for word in WORD_RE.findall(line):
12             yield word.lower(), 1
13
14     def reducer(self, word, counts):
15         self.increment_counter('group', 'Num_reducer_calls', 1)
16         yield word, sum(counts)
17
18 if __name__ == '__main__':
19     MRWordFreqCount.run()
```

Note: no explicit declaration required

Example of counter customization in Mrjob(Cont.)

- **Input:** foo foo quux labs foo bar quux
- **Driver:**

```
1 from mr_wc_counter import MRWordFreqCount
2 mr_job = MRWordFreqCount(args=['WordCount.txt'])
3 with mr_job.make_runner() as runner:
4     runner.run()
5     # stream_output: get access of the output
6     print runner.counters()
```

- **Output:**

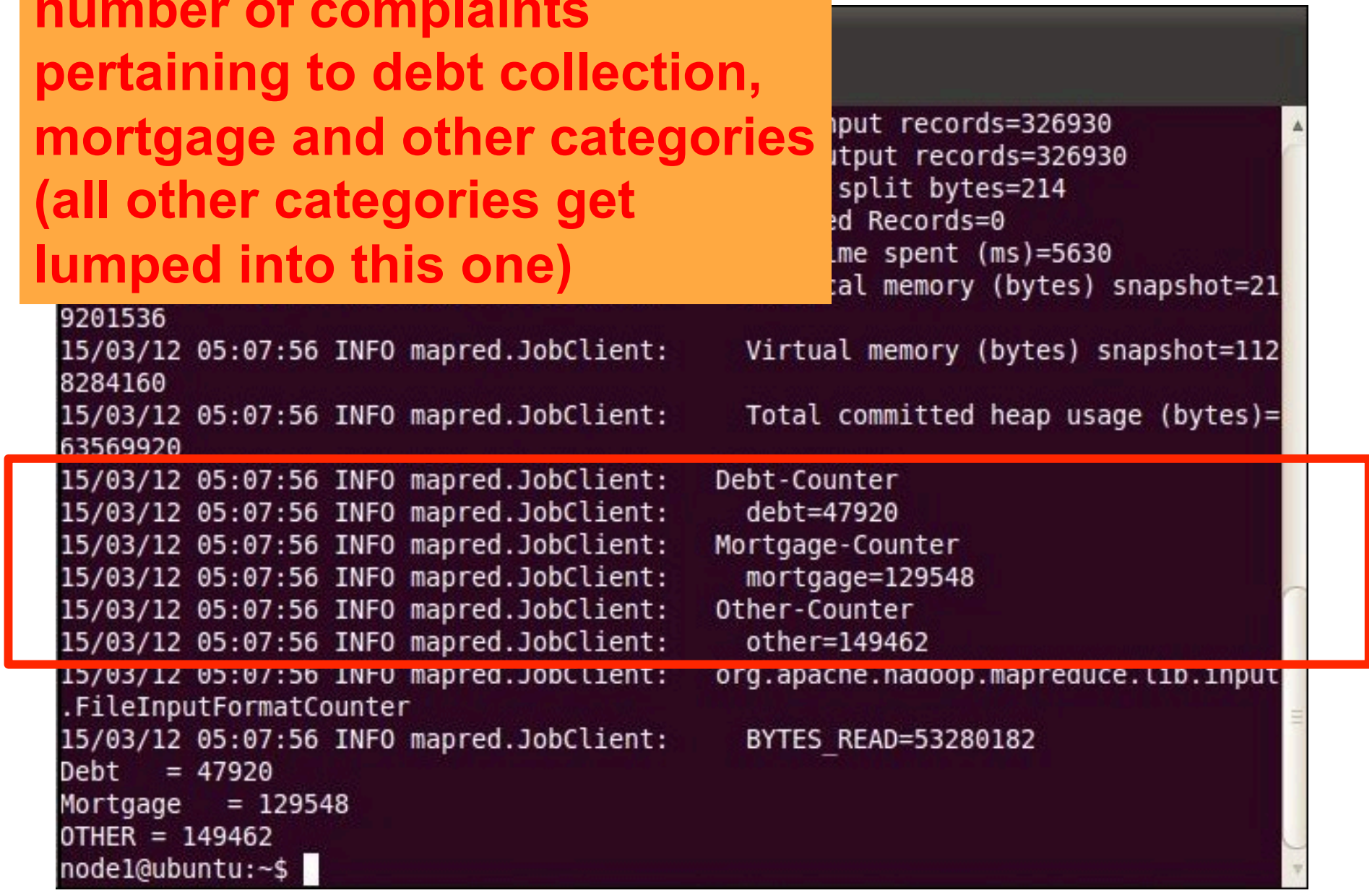
```
[{'group': {'Num_mapper_calls': 1, 'Num_reducer_calls': 4}}]
```

Consumer Complaints Data

```
Complaint ID,Product,Sub-product,Issue,Sub-issue,State,ZIP code,Submitted via,Date received,Date sent to company,Company,Company response,Timely response?,Consumer disputed?
1114245,Debt collection,Medical,Disclosure verification of debt,Not given enough info to verify debt,FL,32219,Web,11/13/2014,11/13/2014,"Choice Recovery, Inc.",Closed with explanation,Yes,
1114488,Debt collection,Medical,Disclosure verification of debt,Right to dispute notice not received,TX,75006,Web,11/13/2014,11/13/2014,"Expert Global Solutions, Inc.",In progress,Yes,
1114255,Bank account or service,Checking account,Deposits and withdrawals,,NY,11102,Web,11/13/2014,11/13/2014,"FNIS (Fidelity National Information Services, Inc.)",In progress,Yes,
1115106,Debt collection,"Other (phone, health club, etc.)",Communication tactics,Frequent or repeated calls,GA,31721,Web,11/13/2014,11/13/2014,"Expert Global Solutions, Inc.",In progress,Yes,
```

number of complaints pertaining to debt collection, mortgage and other categories (all other categories get lumped into this one)

number of complaints pertaining to debt collection, mortgage and other categories (all other categories get lumped into this one)



	Counter	Map	Reduce	Total
File System Counters	FILE: Number of bytes read	0	0	0
	FILE: Number of bytes written	314,030	0	314,030
	FILE: Number of read operations	0	0	0
	FILE: Number of large read operations	0	0	0
	FILE: Number of write operations	0	0	0
	HDFS: Number of bytes read	53,280,689	0	53,280,689
	HDFS: Number of bytes written	5,109,191	0	5,109,191
	HDFS: Number of read operations	6	0	6
	HDFS: Number of large read operations	0	0	0
	HDFS: Number of write operations	4	0	4
Job Counters	Launched map tasks	0	0	2
	Data-local map tasks	0	0	2
	Total time spent by all maps in occupied slots (ms)	0	0	44,647
	Total time spent by all reduces in occupied slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
Map-Reduce Framework	Map input records	326,930	0	326,930
	Map output records	326,930	0	326,930
	Input split bytes	214	0	214
	Spilled Records	0	0	0
	CPU time spent (ms)	5,630	0	5,630
	Physical memory (bytes) snapshot	219,201,536	0	219,201,536
	Virtual memory (bytes) snapshot	1,128,284,160	0	1,128,284,160
	Total committed heap usage (bytes)	63,569,920	0	63,569,920
Debt-Counter	debt	47,920	0	47,920
Mortgage-Counter	mortgage	129,548	0	129,548
Other-Counter	other	149,462	0	149,462
org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter	BYTES_READ	53,280,182	0	53,280,182

Hadoop offers Job Tracker, an UI tool to determine the status and statistics of all jobs. Using the job tracker UI, developers can view the Counters that have been created (refer the below screenshot for more clarity).

In [28]:
```
#usr/local/Cellar/hadoop/2.6.0/libexec/share/hadoop/tools/lib
dataDir = "/Users/jshanahan/Dropbox/lectures-uc-berkeley-ml-class-2015/Notebooks/WordCount"

!hadoop jar /usr/local/Cellar/hadoop/2.6.0/libexec/share/hadoop/tools/lib/hadoop-streaming*.jar \
   -mapper WordCount/mapper.py \
   -reducer WordCount/reducer.py \
   -input historical_tours.txt \
   -output gutenberg-output  \
   -numReduceTasks 2
   #--D mapreduce.job.reduces=2
#-input historical_tours.txt  file on Hadoop
#output directory on Hadoop
```

Multiple reducers

```
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=9
        Map-Reduce Framework
                Map input records=1941
                Map output records=13741
                Map output bytes=108364
                Map output materialized bytes=135858
                Input split bytes=109
                Combine input records=0
                Combine output records=0
                Reduce input groups=3736
                Reduce shuffle bytes=135858
                Reduce input records=13741
                Reduce output records=3736
                Spilled Records=27482
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=25
```

In [27]:
```
pwd
```

Out[27]:
```
u'/Users/jshanahan/Dropbox/lectures-uc-berkeley-ml-class-2015/Notebooks'
```

In [18]:
```
!hdfs dfs -ls
```

```
16/01/27 20:49:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
Found 2 items
drwxr-xr-x   - jshanahan supergroup          0 2016-01-27 20:49 gutenberg-output
-rw-r--r--   1 jshanahan supergroup      87483 2015-02-26 19:36 historical_tours.txt
```

In [29]:
```
!hdfs dfs -ls gutenberg-output
```

```
16/01/27 21:10:53 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
Found 3 items
-rw-r--r--   1 jshanahan supergroup          0 2016-01-27 21:10 gutenberg-output/_SUCCESS
-rw-r--r--   1 jshanahan supergroup      18505 2016-01-27 21:10 gutenberg-output/part-00000
-rw-r--r--   1 jshanahan supergroup      18043 2016-01-27 21:10 gutenberg-output/part-00001
```

Distributed Cache

James G. Shanahan[1]

[1]NativeX and iSchool, UC Berkeley, CA

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

Packaging Files With Job Submissions

You can specify any executable as the mapper and/or the reducer. The executables do not need to pre-exist on the machines in the cluster; however, if they don't, you will need to use "-file" option to tell the framework to pack your executable files as a part of job submission. For example:

```
hadoop jar hadoop-streaming-2.5.2.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myPythonScript.py \
    -reducer /usr/bin/wc \
    -file myPythonScript.py
```

The above example specifies a user defined Python executable as the mapper. The option "-file myPythonScript.py" causes the python executable shipped to the cluster machines as a part of job submission.

In addition to executable files, you can also package other auxiliary files (such as dictionaries, configuration files, etc) that may be used by the mapper and/or the reducer. For example:

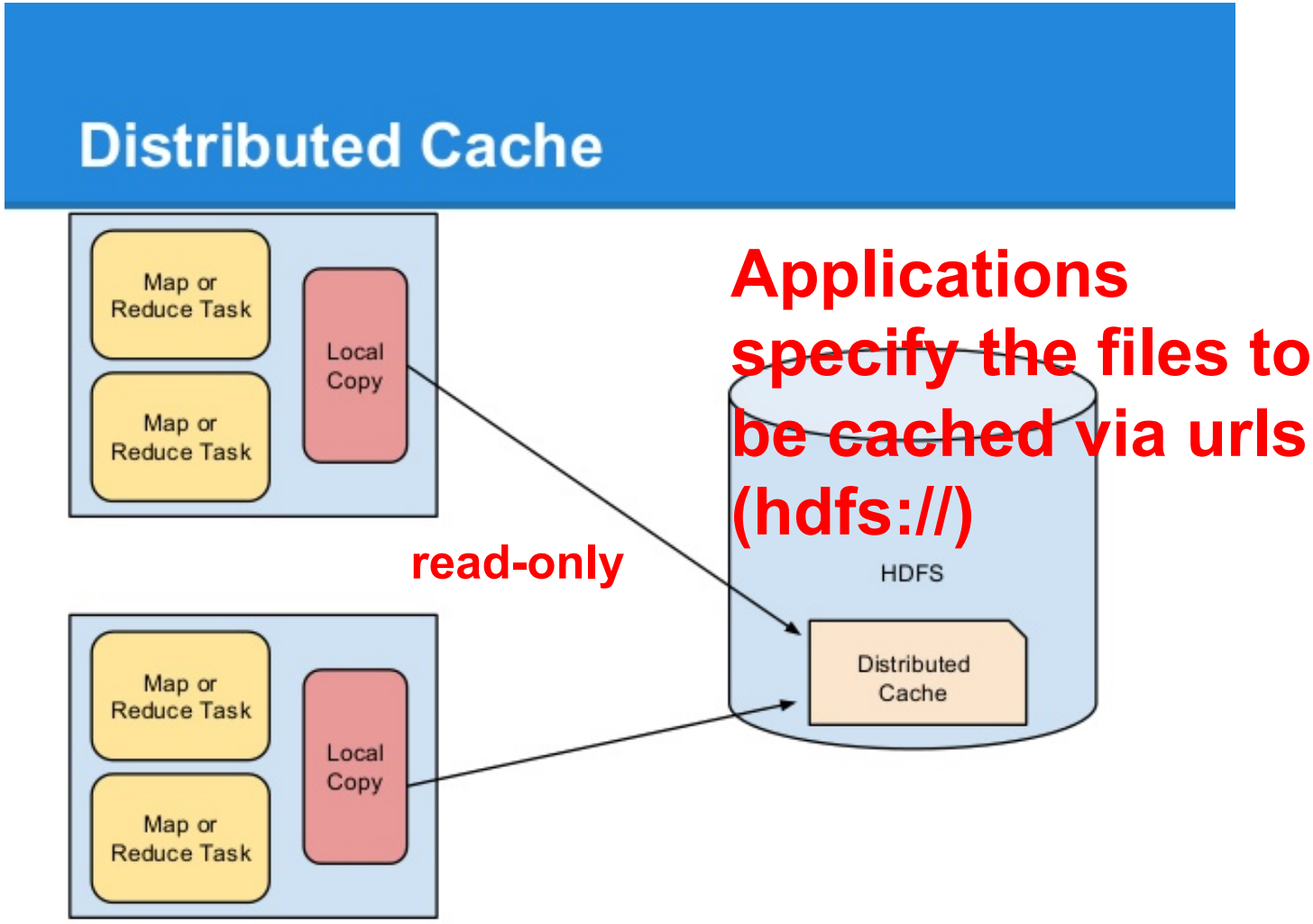
```
hadoop jar hadoop-streaming-2.5.2.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper myPythonScript.py \
    -reducer /usr/bin/wc \
    -file myPythonScript.py \
    -file myDictionary.txt
```

Distributed Cache in Hadoop

- **Distributed Cache is a facility provided by the MapReduce framework to cache files (text, archives, jars and so on) needed by applications.**

- **Distributed Cache can be used to distribute simple, read-only data/text files and more complex types such as archives and jars.**

Distributed Cache in Hadoop



Distributed Cache in Hadoop streaming

- File is distributed cached through -files

-files hdfs://host:fs_port/user/file.txt#filename

URL of the file File name in mapper and reducer

- Now you can access the file in mapper and reducer

```
open(filename,'r')
```

Example

Hadoop streaming script

```
!hadoop jar hadoop-*streaming*.jar -files 'dictionary.txt#dictionary' \
                                   -mapper mapper.py -reducer reducer.py \
                                   -input wordcount.txt \
                                   -output wordcountDictOutput
```

Mapper

```
%%writefile mapper.py
#!/usr/bin/python
import sys
# input comes from STDIN (standard input)
f = open('dictionary', 'r')
word_dict = []
for line in f:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    for word in words:
        word_dict.append(word)
```

Example: Filtered WordCount

- http://nbviewer.ipython.org/
- https://www.dropbox.com/s/zll36pds0z1bqtp/Hadoop%20Streaming%20WordCount-Distributedcache.ipynb?dl=0

Reading in Dictionary values (key-value records) in python

Starting in Python 2.6 you can use the built-in `ast.literal_eval`:

```
>>> import ast
>>> ast.literal_eval("{'muffin' : 'lolz', 'foo' : 'kitty'}")
{'muffin': 'lolz', 'foo': 'kitty'}
```

This is safer than using `eval`. As its own docs say:

```
>>> help(ast.literal_eval)
Help on function literal_eval in module ast:

literal_eval(node_or_string)
    Safely evaluate an expression node or a string containing a Python
    expression.  The string or node provided may only consist of the following
    Python literal structures: strings, numbers, tuples, lists, dicts, booleans,
    and None.
```

OR via JSON

using `json.loads`

```
>>> import json
>>> h = '{"foo":"bar", "foo2":"bar2"}'
>>> type(h)
<type 'str'>
>>> d = json.loads(h)
>>> d
{u'foo': u'bar', u'foo2': u'bar2'}
>>> type(d)
<type 'dict'>
```

Mapper

http://nbviewer.ipython.org/urls/dl.dropbox.com/s/zll36pds0z1bqtp/Hadoop%20Streaming%20WordCount-Distributedcache.ipynb

```
%%writefile mapper.py
#!/usr/bin/python
import sys
# input comes from STDIN (standard input)
f = open('dictionary', 'r')
word_dict = []
for line in f:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    for word in words:
        word_dict.append(word)

for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        if word in word_dict:
            print '%s\t%s' % (word, 1)
```

- **Number of mappers and reducers**

task 1 of 13

Partitioning your job into maps and reduces

Picking the appropriate size for the tasks for your job can radically change the performance of Hadoop. Increasing the number of tasks increases the framework overhead, but increases load balancing and lowers the cost of failures. At one extreme is the 1 map/1 reduce case where nothing is distributed. The other extreme is to have 1,000,000 maps/ 1,000,000 reduces where the framework runs out of resources for the overhead.

Number of Maps

The number of maps is usually driven by the number of DFS blocks in the input files. Although that causes people to adjust their DFS block size to adjust the number of maps. The right level of parallelism for maps seems to be around 10-100 maps/node, although we have taken it up to 300 or so for very cpu-light map tasks. Task setup takes awhile, so it is best if the maps take at least a minute to execute.

Actually controlling the number of maps is subtle. The mapred.map.tasks parameter is just a hint to the InputFormat for the number of maps. The default InputFormat behavior is to split the total number of bytes into the right number of fragments. However, in the default case the DFS block size of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via mapred.min.split.size. Thus, if you expect 10TB of input data and have 128MB DFS blocks, you'll end up with 82k maps, unless your mapred.map.tasks is even larger. Ultimately the InputFormat determines the number of maps.

The number of map tasks can also be increased manually using the JobConf's conf.setNumMapTasks(int num). This can be used to increase the number of map tasks, but will not set the number below that which Hadoop determines via splitting the input data.

Number of Reduces

http://wiki.apache.org/hadoop/HowManyMapsAndReduces

The ideal reducers should be the optimal value that gets them closest to:

* A multiple of the block size * A task time between 5 and 15 minutes * Creates the fewest files possible

Anything other than that means there is a good chance your reducers are less than great. There is a tremendous tendency for users to use a REALLY high value ("More parallelism means faster!") or a REALLY low value ("I don't want to blow my namespace quota!"). Both are equally dangerous, resulting in one or more of:

* Terrible performance on the next phase of the workflow * Terrible performance due to the shuffle * Terrible overall performance because you've overloaded the namenode with objects that are ultimately useless * Destroying disk IO for no really sane reason * Lots of network transfers due to dealing with crazy amounts of CFIF/MFIF work

Now, there are always exceptions and special cases. One particular special case is that if following that advice makes the next step in the workflow do ridiculous things, then we need to likely 'be an exception' in the above general rules of thumb.

Currently the number of reduces is limited to roughly 1000 by the buffer size for the output files (io.buffer.size * 2 * numReduces << heapSize). This will be fixed at some point, but until it is it provides a pretty firm upper bound.

The number of reduce tasks can also be increased in the same way as the map tasks, via JobConf's conf.setNumReduceTasks(int num).

How Many Mappers And Reduces?

- **Picking the appropriate size for the tasks for your job can radically change the performance of Hadoop.**
- **Increasing the number of tasks increases the framework overhead, but increases load balancing and lowers the cost of failures.**
  - At one extreme is the 1 map/1 reduce case where nothing is distributed. The other extreme is to have 1,000,000 maps/ 1,000,000 reduces where the framework runs out of resources for the overhead.
- **The number of reduce tasks can also be increased in the same way as the map tasks, via JobConf's conf.setNumMapTasks(int num).**

128MB Block size for 10TB Job →82k maps
10-100 maps per node; each map takes 1 minute
~~10^5 tasks~~

- The number of maps is usually driven by the number of DFS blocks in the input files.
- Although that causes people to adjust their DFS block size to adjust the number of maps.
- The right level of parallelism for maps seems to be around 10-100 maps/node, although we have taken it up to 300 or so for very cpu-light map tasks. Task setup takes awhile, so it is best if the maps take at least a minute to execute.
- E.g., if you expect 10TB of input data and have 128MB DFS blocks, you'll end up with 82k maps, unless your mapred.map.tasks is even larger
- The right level of parallelism for maps seems to be around 10-100 maps/node

Back of the envelope Estimates: 10TB

- 128MB Block size for 10TB Job →82k maps ($10^{5)}$
- Recommended to have 10-100 maps per node;
- 10^5 tasks * 10^{-2} = 1,000 nodes
- Each map takes 1 minute, so about 100 minutes to run the mappers

Number of Reducers

- The ideal reducers should be the optimal value that gets them closest to:

- A multiple of the block size
- A task time between 5 and 15 minutes
- Creates the fewest files possible

- The number of reduce tasks can also be increased in the same way as the map tasks, via JobConf's conf.setNumReduceTasks(int num).

How Many Maps And Reduces

- **Empirical question: best solved by taking this guidance and experimenting with some small jobs and establishing the length of map and reduce tasks at different block sizes etc.**

- **Recommend block sizes of 128MB – 1,000MB (1Gig)**

- **Partition Part-0000**
  - 1900,….
  - 1900
  - 1901
  - -----1910
- **Partition Part-0001**
  - 1911,….
  - 1911,…
  - …
  - 1933
- **Partition Part-000N**
  - 2015,….
  - ….
  - 2016,….

head -1 Part-0000

Cat Part* > myLovelyOutput.txt

Hadoop job2 –input OutputDirFo

Tail -1 Part-000N #newest and lowest t

Computing the Mean: Version 4: In-memory mapper

```
1: class Mapper
2:     method Initialize
3:         S ← new AssociativeArray
4:         C ← new AssociativeArray
5:     method Map(string t, integer r)
6:         S{t} ← S{t} + r
7:         C{t} ← C{t} + 1
8:     method Close
9:         for all term t ∈ S do
10:            Emit(term t, pair (S{t}, C{t}))
```

In-memory hash table of customer ids

Super efficient: no shuffling!

Inside the mapper, the partial sums and counts associated with each string are held in memory across input key-value pairs. Intermediate key-value pairs are emitted only after the entire input split has been processed; similar to before, the value is a pair consisting of the sum and count. The reducer is exactly the same as above

Are combiners still needed? For 10TB data problem

- **Debugging practices**

Writing Mrjob code

- **Example 1: WordCount (Cont.)**

  Brief Introduction of Yield:

Generally speaking, iterators and generators (functions that create iterators, for example with Python's `yield` statement) have the advantage that an element of a sequence is not produced until you actually need it. This can help a lot in terms of computational expensiveness or memory consumption depending on the task at hand.

```
8  def TestGenerator(l):
9    for e in l:
10    print ("before yield:" + str(e))
11    yield e
12    print ("after yield:" + str(e))
13
14 for el in TestGenerator([6,7,8,9]):
15   print (el)
```

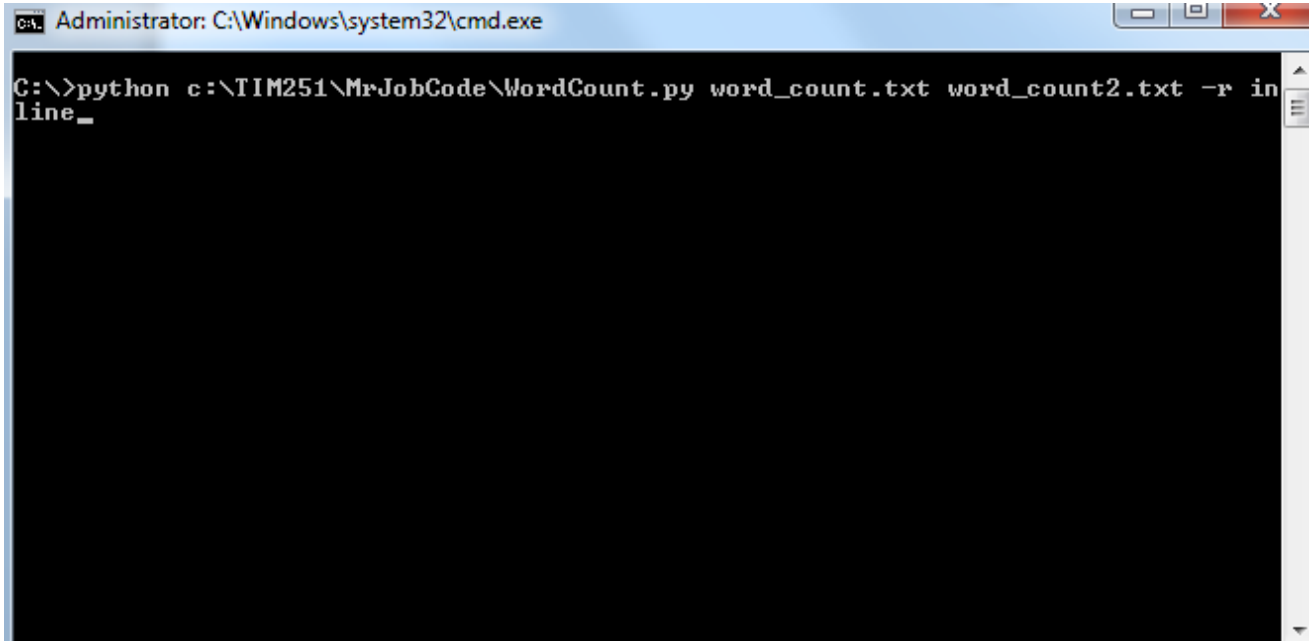
```
before yield:6
6
after yield:6
before yield:7
7
after yield:7
before yield:8
8
after yield:8
before yield:9
9
after yield:9
```

Running Mrjob code on command line

- **Example 1: WordCount (Cont.)**

  Command Line:

  $python MRWordFreqCount.py inputfile.txt



Runing Mrjob code: 2 output PART files

- **Example 1: WordCount (Cont.)**

  Result: partial results

  Notice two output partitions (two reducers are running)

```
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.055355
.422000\step-0-mapper_part-00000 -> c:\users\liang.dai\appdata\local\temp\WordCo
unt.liang.dai.20140421.055355.422000\output\part-00000
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.055355
.422000\step-0-mapper_part-00001 -> c:\users\liang.dai\appdata\local\temp\WordCo
unt.liang.dai.20140421.055355.422000\output\part-00001
Streaming final output from c:\users\liang.dai\appdata\local\temp\WordCount.lian
g.dai.20140421.055355.422000\output
"hi"    1
"hello" 1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
"hi"    1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
removing tmp directory c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai
.20140421.055355.422000
```

Writing code in Mrjob: key development steps

- **key development steps using WordCount Example**
  - Write mapper and reducer functions for word count
  - What does each step (mapper, reducer, combine) do?
  - Verify mapper, reducer, combine steps
    - Let's check it step by step with two input files.
    - By Mrjob default setting, two mappers will be assigned. It is also a good way for debugging.

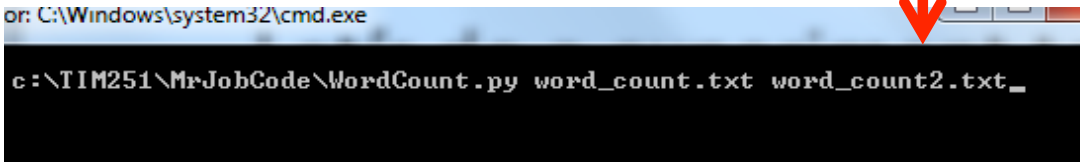
Writing Mrjob code with no Combiner: test mapper only

- **Example 1: WordCount (Cont.)**
  - Just use a Mapper
  - Debug tactic!

```
 9 from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19 '''
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     def reducer(self, word, counts):
24         yield word, sum(counts)
25 '''
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

Each mapper output word, 1

Two Inputs




```
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.d
.422000\step-0-mapper_part-00000 -> c:\users\liang.dai\appdata
unt.liang.dai.20140421.055355.422000\output\part-00000
Moving c:\users\liang.dai\appdata\local\temp\WordCount.liang.d
.422000\step-0-mapper_part-00001 -> c:\users\liang.dai\appdata
unt.liang.dai.20140421.055355.422000\output\part-00001
Streaming final output from c:\users\liang.dai\appdata\local\t
g.dai.20140421.055355.422000\output
"hi"    1
"hello" 1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
"hi"    1
"hi"    1
"hello" 1
"hello" 1
"hi"    1
removing tmp directory c:\users\liang.dai\appdata\local\temp\W
.20140421.055355.422000
```

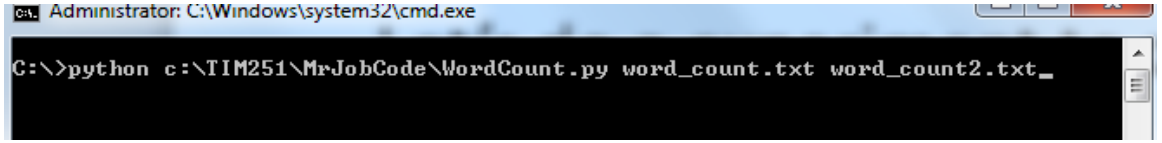
Writing Mrjob code with Combiner: Test Combiner output

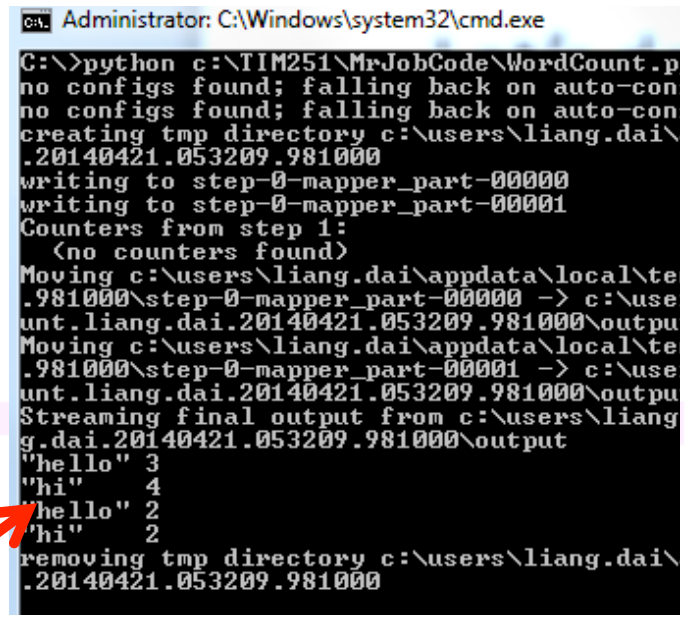
- **Example 1: WordCount (Cont.)**

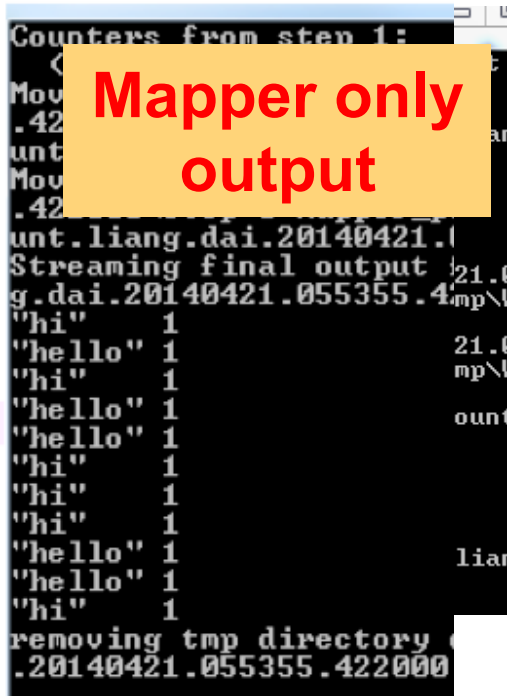
  Let's check combiner (and no reducer!)

  Good debug tactic!!

```
 9 from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     # def reducer(self, word, counts):
24     #     yield word, sum(counts)
25
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```







Mapper only output

Combiner is a mapper-side reducer (two mappers as the word "hello" occurs twice in the output stream)

Writing Mrjob code: test reducer

- **Example 1: WordCount (Cont.)**

  Let's check reducer:

2 Reducers

```
 9 from mrjob.job import MRJob
10 import re
11
12 WORD_RE = re.compile(r"[\w']+")
13
14 class MRWordFreqCount(MRJob):
15
16     def mapper(self, _, line):
17         for word in WORD_RE.findall(line):
18             yield word.lower(), 1
19
20     def combiner(self, word, counts):
21         yield word, sum(counts)
22
23     def reducer(self, word, counts):
24         yield word, sum(counts)
25
26 if __name__ == '__main__':
27     MRWordFreqCount.run()
```

```
Administrator: C:\Windows\system32\cmd.exe

C:\>python c:\TIM251\MrJobCode\WordCount.py word_count.txt word_count2.txt_
```

```
writing to step-0-mapper_part-00000
writing to step-0-mapper_part-00001
Counters from step 1:
  (no counters found)
writing to c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.05
3402.399000\step-0-mapper-sorted
> sort 'c:\users\liang.dai\appdata\local\temp\WordCount.liang.dai.20140421.05340
2.399000\step-0-mapper_part-00000' 'c:\users\liang.dai\appdata\local\temp\WordCo
unt.liang.dai.20140421.053402.399000\step-0-mapp
Piping files into sort for Windows compatibility
> sort
writing to step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving c:\users\liang.dai\appdata\local\temp\Wor
.399000\step-0-reducer_part-00000 -> c:\users\li
ount.liang.dai.20140421.053402.399000\output\par
Streaming final output from c:\users\liang.dai\a
g.dai.20140421.053402.399000\output
"hello" 5
"hi"    6
removing tmp directory c:\users\liang.dai\appdat
.20140421.053402.399000
```

Combiner only output

```
"hello" 3
"hi"    4
"hello" 2
"hi"    2
removing tmp direc
.20140421.053209.9
```

Reducer combines the results from the mappers and combiners

Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
  - Office hours: Mondays at 5:30PM. Please submit Qs by Sunday at 5:30PM
  - Homework 1&2: Naive Bayes Experts (grades due on Saturday)
- **Peer Grading**
  - HW1 peer grading
  - HW2 Answer reviews
- **Week 3**
  - Hadoop useful stuff/Trivia
    - Blocks sizes
    - Total Sort
    - Understanding Hadoop via Counters
    - Hadoop File passing pattern
    - Number of mappers and reducers
  - Async lecture recap plus Q&A: Pairs/Secondary keys/Inversion pattern
  - Association rule mining via Apriori
- **Next week (Cloud: MrJob)**
- **Wrapup: Finish RECORDING (bonus points!)**

Secondary Sort

- The MapReduce framework sorts the records by key before they reach the reducers.
- For any particular key, however, the values are not sorted.
  - The order in which the values appear is not even stable from one run to the next, because they come from different map tasks, which may finish at different times from run to run.
  - Generally speaking, most MapReduce programs are written so as not to depend on the order in which the values appear to the reduce function.
- However, it is possible to impose an order on the values by sorting and grouping the keys in a particular way.

Max temperature for each year

- To illustrate the idea, consider the MapReduce program for calculating the maximum temperature for each year.
- If we arranged for the values (temperatures) to be sorted in descending order, we wouldn't have to iterate through them to find the maximum; instead, we could take the first for each year and ignore the rest. (This approach isn't the most efficient way to solve this particular problem, but it illustrates how secondary sort works in general.)

Year increasing
Temp descreaing

Key, Value

Concatenate

Key=Year+Temp

```
1900 35°C
1900 34°C
1900 34°C
...
1901 36°C
1901 35°C
```

Max temperature for each year

- **How to do this in MApReduce?**

Key, Value

```
1900 35°C
1900 34°C
1900 34°C
...
1901 36°C
1901 35°C
```

Year+Temp

- **Key=Year+Value: what happens?**
  - Partition the output records from mapper by the key part of the record
  - Records for the same year get sent to different reducers
  - Get a special partitioner that partitons by the first part of the key

Secondary Key Receipe

- To summarize, there is a recipe here to get the effect of sorting by value:
  - Make the key a composite of the natural key and the natural value.
  - The sort comparator should order by the composite key, that is, the natural key and natural value.
  - The partitioner and grouping comparator for the composite key should consider only the natural key for partitioning and grouping.

Streaming

To do a secondary sort in Streaming, we can take advantage of a couple of library classes that Hadoop provides. Here's the driver that we can use to do a secondary sort:

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \
  -D stream.num.map.output.key.fields=2 \
  -D mapreduce.partition.keypartitioner.options=-k1,1 \
  -D mapreduce.job.output.key.comparator.class=\
org.apache.hadoop.mapred.lib.KeyFieldBasedComparator \
  -D mapreduce.partition.keycomparator.options="-k1n -k2nr" \
  -files secondary_sort_map.py,secondary_sort_reduce.py \
  -input input/ncdc/all \
  -output output-secondarysort-streaming \
  -mapper ch09-mr-features/src/main/python/secondary_sort_map.py \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner \
  -reducer ch09-mr-features/src/main/python/secondary_sort_reduce.py
```

N =numeric
Nr numeric reverse

Our map function (Example 9-7) emits records with year and temperature fields. We want to treat the combination of both of these fields as the key, so we set `stream.num.map.output.key.fields` to 2. This means that values will be empty, just like in the Java case.

Example 9-7. Map function for secondary sort in Python

```
#!/usr/bin/env python

import re
import sys

for line in sys.stdin:
  val = line.strip()
  (year, temp, q) = (val[15:19], int(val[87:92]), val[92:93])
  if temp == 9999:
    sys.stderr.write("reporter:counter:Temperature,Missing,1\n")
  elif re.match("[01459]", q):
    print "%s\t%s" % (year, temp)
```

However, we don't want to partition by the entire key, so we use the `KeyFieldBasedPartitioner` partitioner, which allows us to partition by a part of the key. The specification `mapreduce.partition.keypartitioner.options` configures the partitioner. The value `-k1,1` instructs the partitioner to use only the first field of the key, where fields are

Composite-Key: Natural key+secondary key

Very useful: gets the map-reduce framework to do all the heavy lifting!

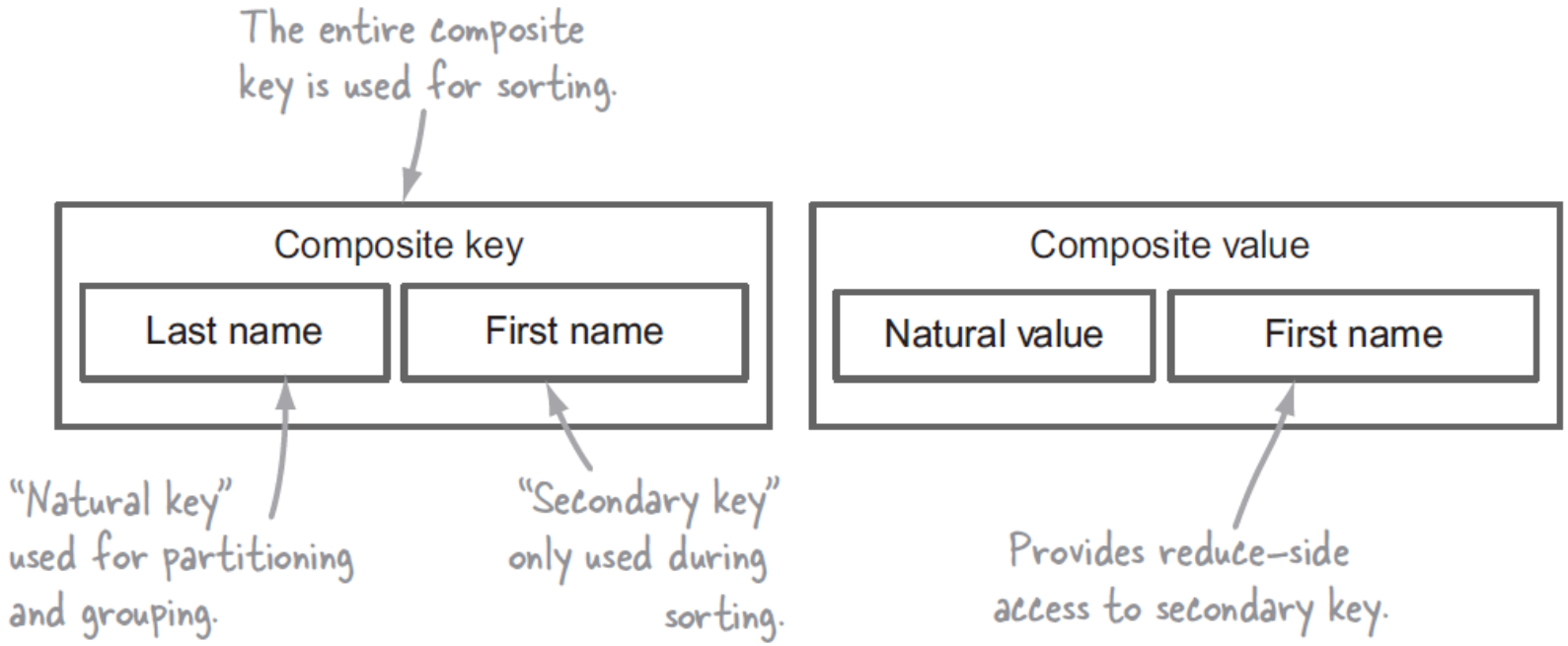
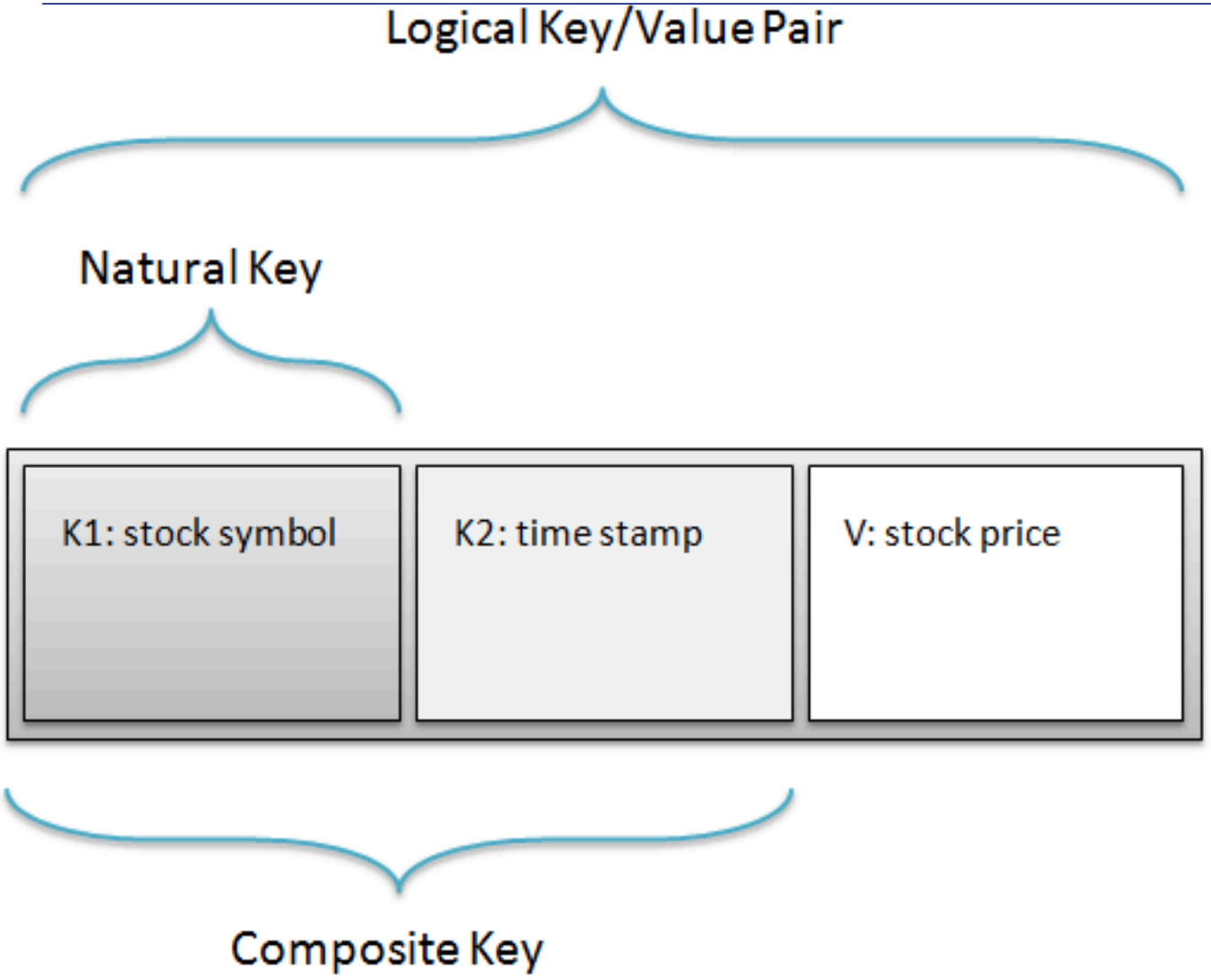


Figure 4.13 The user composite key and value

Composite Key: Partition and sort



The Composite Key gives Map-Reduce framework the needed information during the shuffle to perform a sort not only on the "stock symbol", but on the time stamp as well.

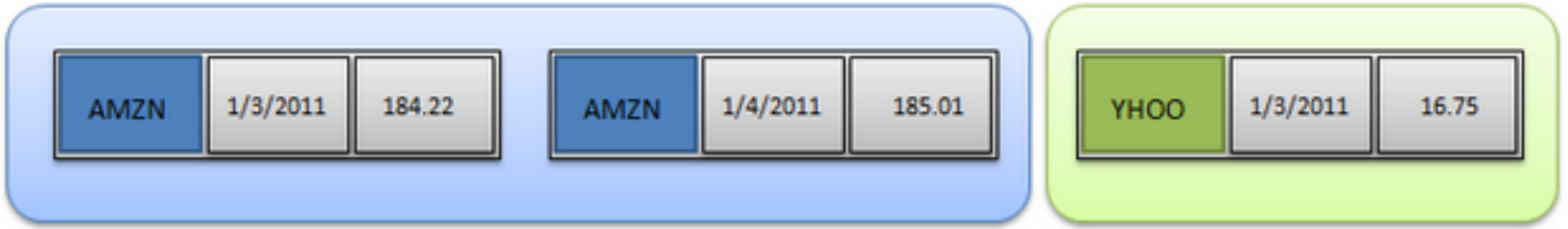
Partition on K1 Sort on K1+K2

Figure-1: *Composite Key Diagram*

Secondary sort
Composite Key:“stock symbol” + timestamp

Sorting by the Composite Key, using integers for both keys for simplicity.

The Composite Key gives Map-Reduce framework the needed information during the shuffle to perform a sort not only on the “stock symbol”, but on the time stamp as well.

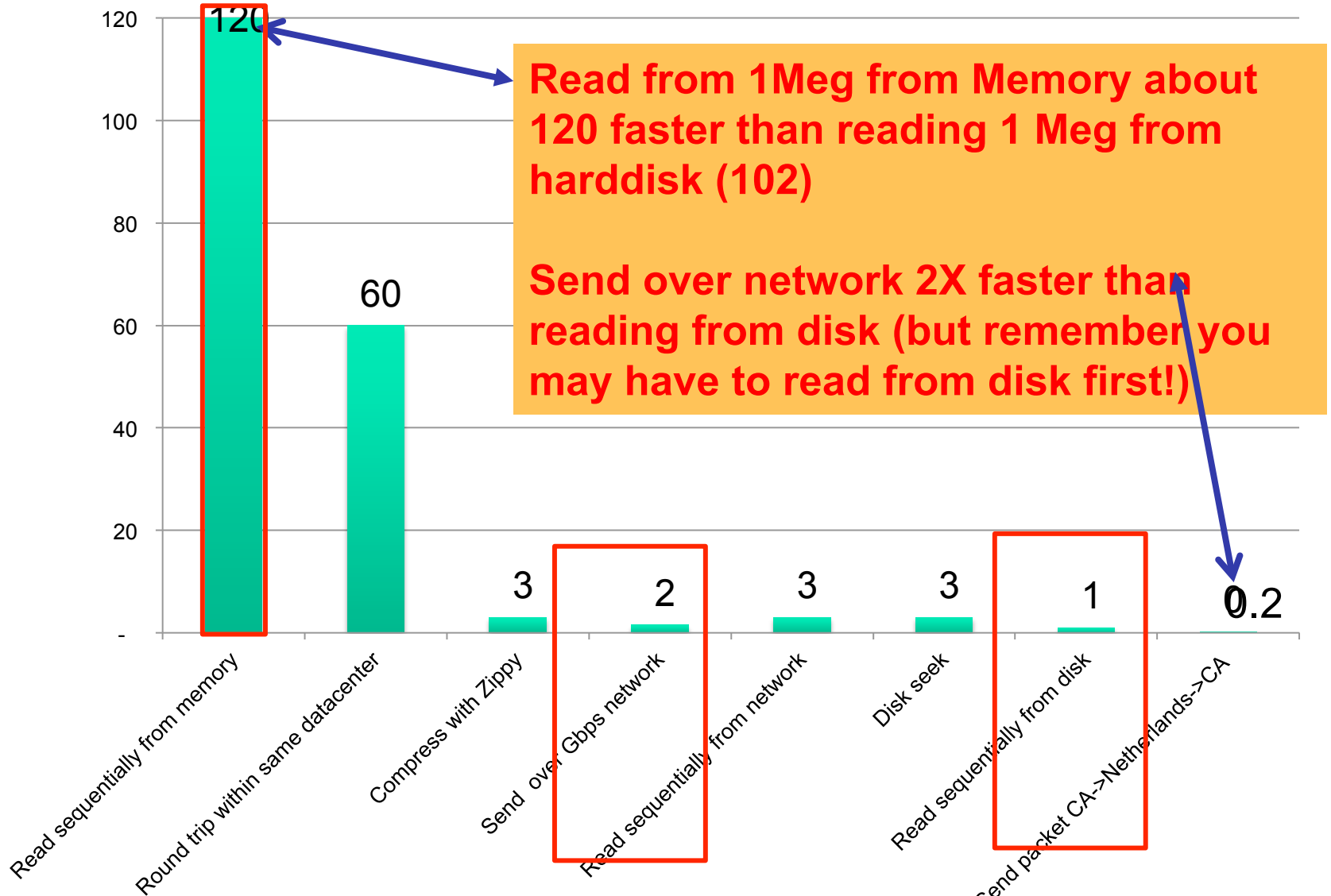


“AMZN” Partitioning by the natural key.

“YHOO” Partitioning by the natural key.

Figure-4: *Partitioning by the natural key with the NaturalKeyPartitioner.*

Task vs 1M Read from Disk
RAM vs. disk vs. bandwidth

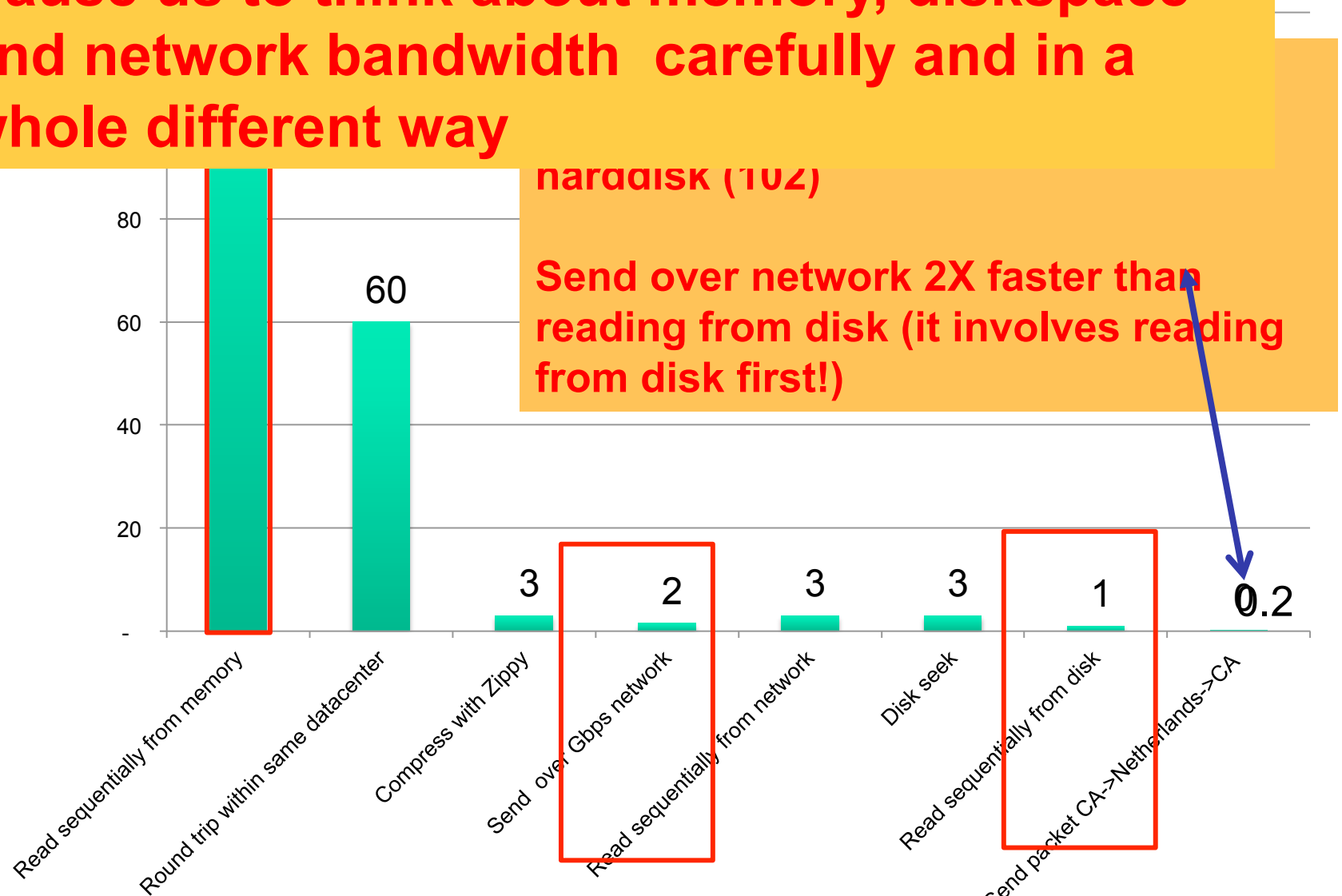


Task vs 1M Read from Disk RAM vs. disk vs. bandwidth

Cause us to think about memory, diskspace and network bandwidth carefully and in a whole different way

harddisk (102)

Send over network 2X faster than reading from disk (it involves reading from disk first!)



Hadoop Shuffle

- **When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons**
- **Each map task has a circular memory buffer**
  - Each map task has a circular memory buffer that it writes the output to.
    - The buffer is 100 MB by default, a size that can be tuned by changing the mapreduce.task.io.sort.mb property.
  - When the contents of the buffer reaches a certain threshold size (mapreduce.map.sort.spill.percent, which has the default 0.80, or 80%), a background thread will start to spill the contents to disk.
- **Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete.l**

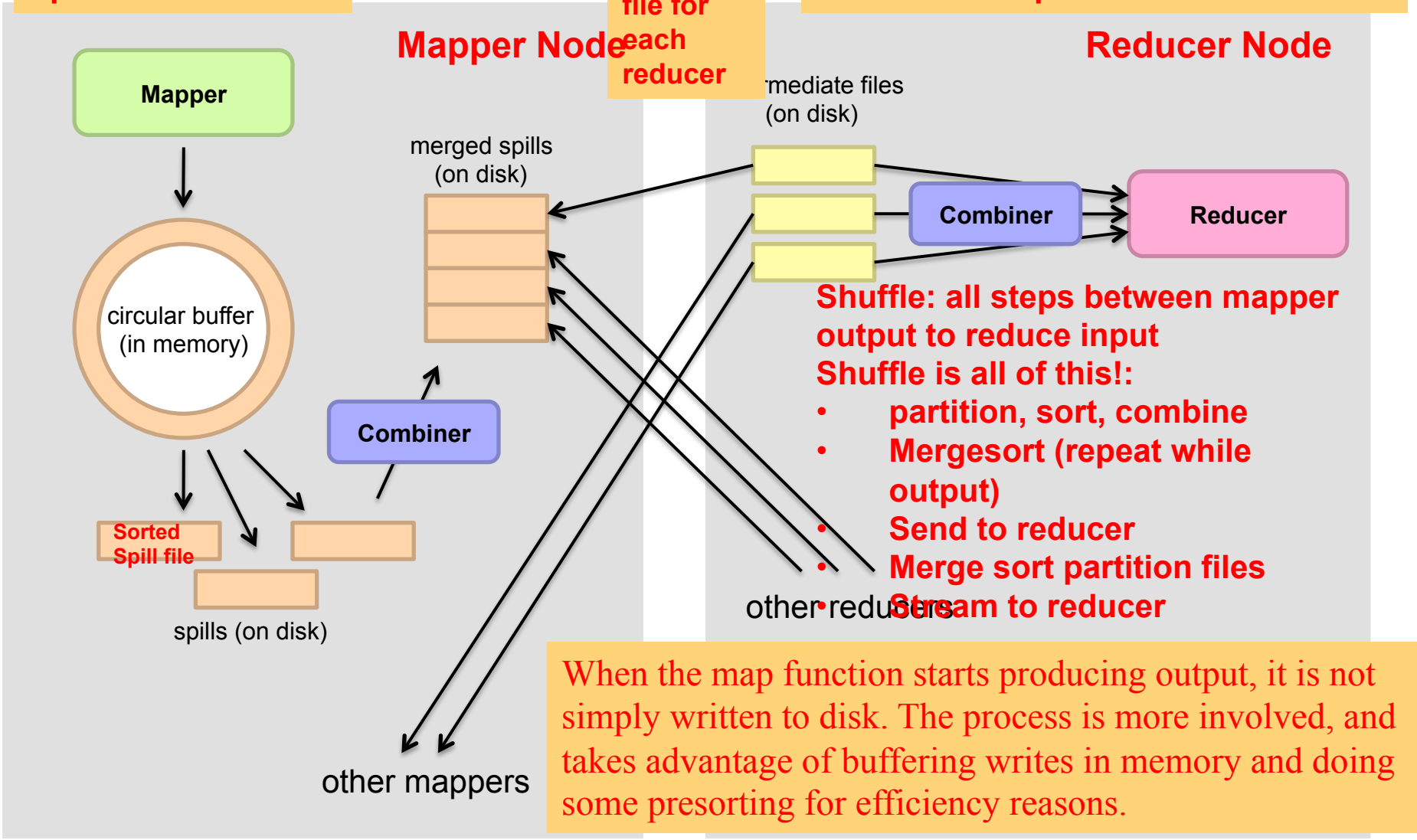
First of all shuffling is the process of transferring data from the mappers to the reducers

Summary: Hadoop Shuffle

Shuffle: partition, sort, combine Spill to disk

Ship a single partition file for each reducer

Merge the partition file from each mapper, Combine, and generate a stream of inputs for the reducerd



Shuffle: all steps between mapper output to reduce input

Shuffle is all of this!:

- **partition, sort, combine**
- **Mergesort (repeat while output)**
- **Send to reducer**
- **Merge sort partition files**
- **Stream to reducer**

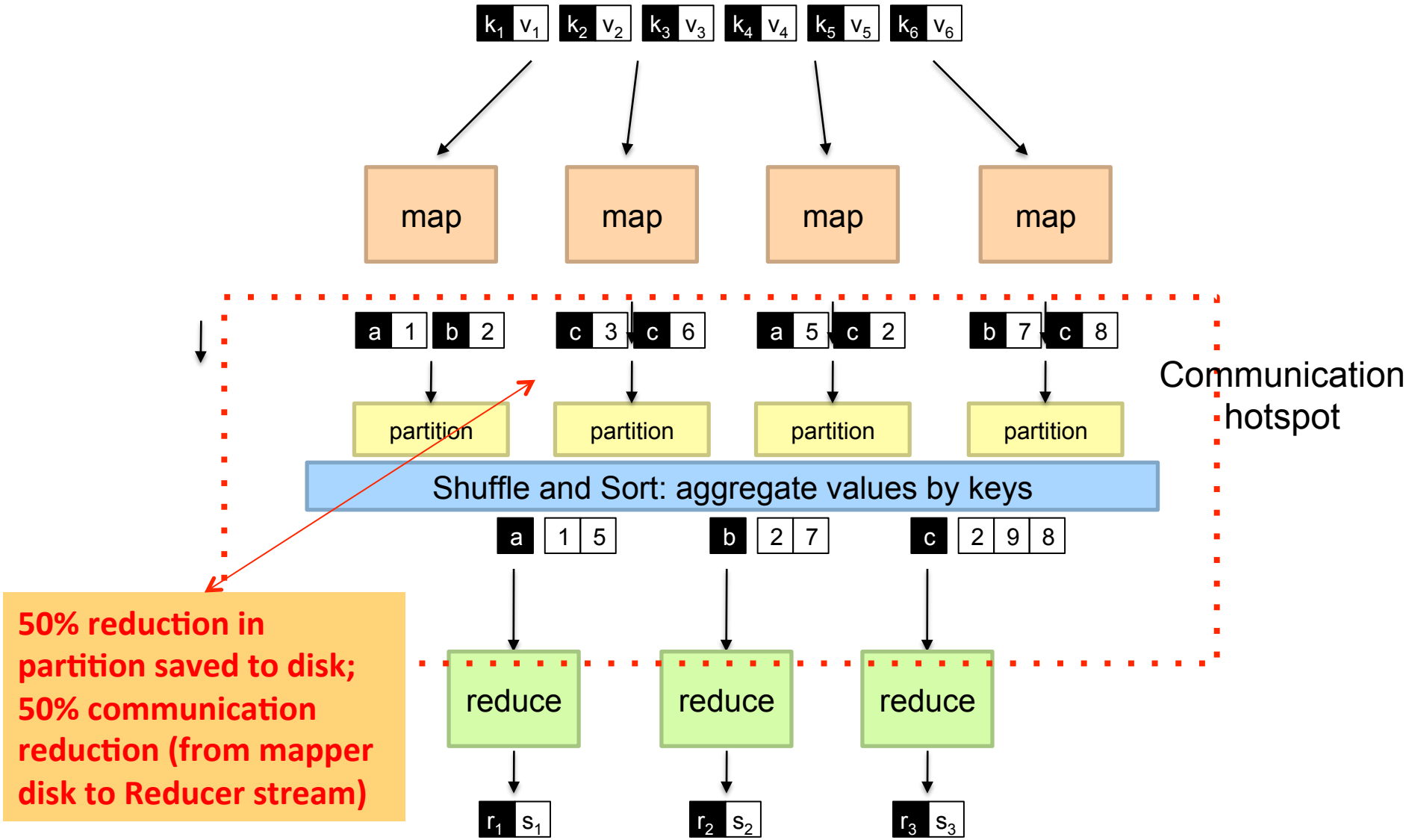
When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons.

Tools for Synchronization

- **Cleverly-constructed data structures (in memory)**
  - Bring partial results together
- **Sort order of intermediate keys**
  - Control order in which reducers process keys
- **Partitioner**
  - Control which reducer processes which keys
- **Preserving state in mappers and reducers**
  - Capture dependencies across multiple keys and values

Composite keys; secondary sorting

Communication hotspot: Transmitting Mapper results to the reducer



Word Count: Baseline

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count s)
```

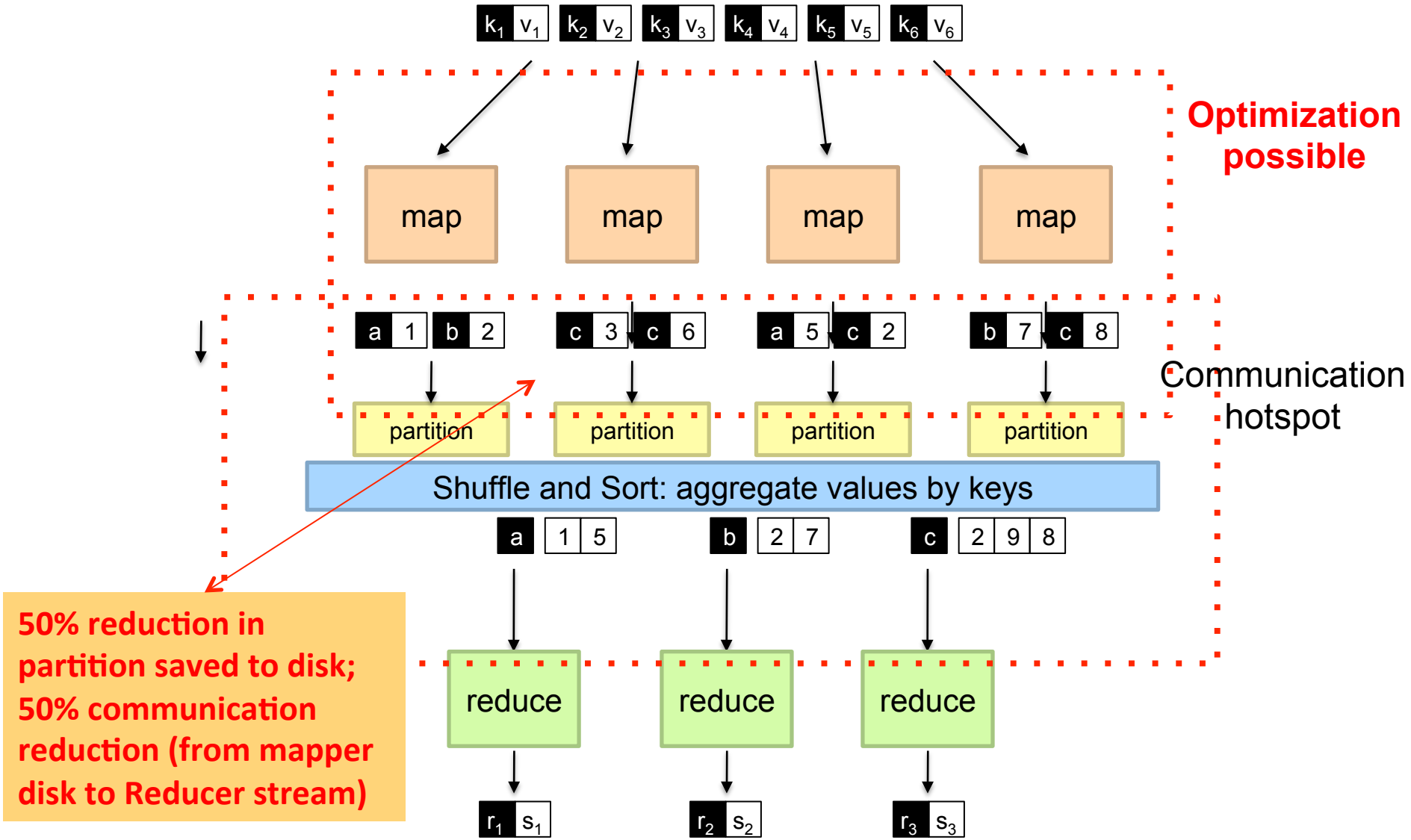
PIAT

1. In mem: Partition, sort, ~~combine~~
2. Spill to Disk
3. Merge sorted spills, ~~combine~~
4. Once mapper finishes:
   - Transfer to reducer bandwidt

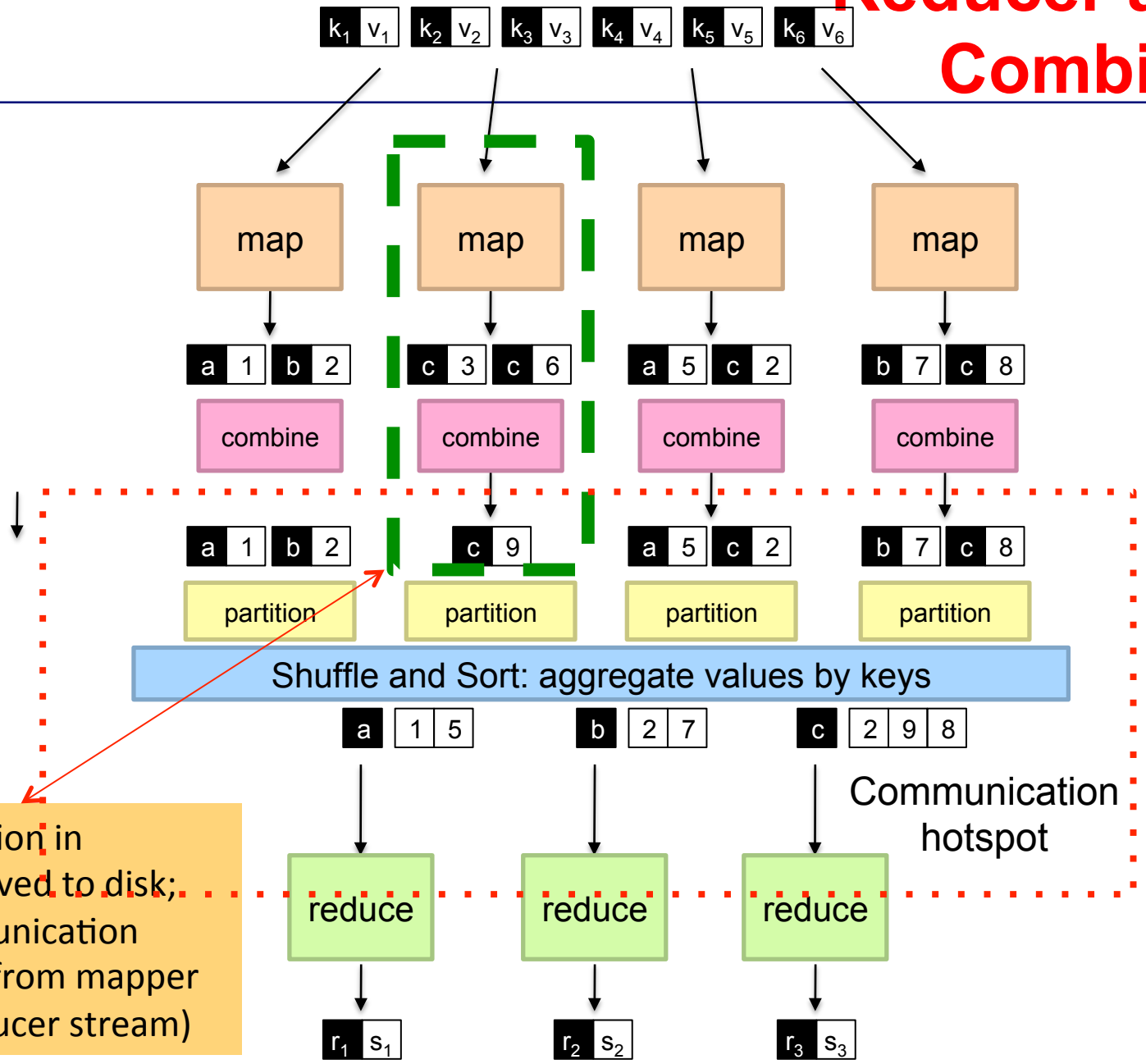
Emit each word…heavy disk storage requirements on Mapper side; big sorts; big transfers from mapper to reducer

What's the impact of combiners?

Communication hotspot: Transmitting Mapper results to the reducer



Reducer as a Combiner



50% reduction in partition saved to disk; 50% communication reduction (from mapper disk to Reducer stream)

Word Count: Baseline+mini-reducer as a combiner

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)
```

PIAT

1. In mem: Partition, sort, combine
2. Spill to Disk
3. Merge sorted spills, combine
4. Once mapper finishes:
   - Transfer to reducer bandwidth

```
1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count s)
```

Emit each word…heavy disk storage requirements on Mapper side; big sorts; big transfers from mapper to reducer

What's the impact of combiners?

Relative Frequencies of co-occurring terms

Corpus: A D C E A D F E B A C E D

Corpus

Window size: 2 (the 2 words of the either side)

Context vectors:

A co-occurs with C 3 times

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

Sparse repn

A → { B: 1, C: 3, D: 2, E:3 }

A → { B: 1/9, C: 3/9, D: 2/9, E:3/9 }

Pairs

A, D, 1
A,1
A, C, 1
A, 1
D, A, 1
D, 1

A, D, 1
A,1
A, C, 1
A, 1
A, E, 1
A,1
A, D, 1
A, 1

Denominator

How do we estimate relative frequencies from counts?

$$f(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}$$

1

1+3+2+3

Why do we want to do this?

- **How do we do this with MapReduce?**

Relative Frequencies of co-occurring terms

What is the key?

A, B 1
A*, 1
A, C 2
A*, 2
-- Reducer
How to avoid to'

Corpus: A D C E A D F E B A C E D

Corpus

Window size: 2 (the 2 words of the either side)

Context vectors:

A co-occurs with C 3 times

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

A → { B: 1, C: 3, D: 2, E:3 }

A → { B: 1/9, C: 3/9, D: 2/9, E:3/9 }

- **How do we estimate relative frequencies from counts?**

$$f(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}$$

1

1+3+2+3

- **Why do we want to do this?**
- **How do we do this with MapReduce?**

Corpus: A D C E A D F E B A C E D

Window size: 2 (the 2 words of the either side)

Context vectors:

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

Sparse repn

A → { B: 1, C: 3, D: 2, E:3 }

A → { B: 1/9, C: 3/9, D: 2/9, E:3/9 }

A, B 1
A, 1
A, C 2
A, 2
-- Reducer
A,B,1
A, 1
A, 2
A, C,2

Leads to Memory issue

Corpus: A D C E A D F E B A C E D

Window size: 2 (the 2 words of the either side)

Context vectors:

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

Sparse repn

A → { B: 1, C: 3, D: 2, E:3 }

A → { B: 1/9, C: 3/9, D: 2/9, E:3/9 }

A, B 1
A*, 1
A, C 2
A*, 2
-- Reducer
A*, 1
A*, 2
A,B,1
A, C,2

Order inversion
Leads to NO Memory issue!

- 1A, 1
- 2A, B, 1
- 1A, 2
- 2A, C, 2

Relative Frequencies of co-occurring terms

What is the key?

A, B 1
A*, 1
A, C 2
A*, 2
-- Reducer
How to avoid to'

Corpus: A D C E A D F E B A C E D

Corpus

Window size: 2 (the 2 words of the either side)

Context vectors:

A co-occurs with C 3 times

	A	B	C	D	E
A	0	1	3	2	3
B	1	0	1	0	1
C	3	1	0	2	2
D	2	0	2	0	4
E	3	1	2	4	0

A → { B: 1, C: 3, D: 2, E:3 }

A → { B: 1/9, C: 3/9, D: 2/9, E:3/9 }

- **How do we estimate relative frequencies from counts?**

$$f(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}$$

1

1+3+2+3

- **Why do we want to do this?**
- **How do we do this with MapReduce?**

Need to reconstruct the list of co-occurring terms with the term of interest

- To make this work, we must define the sort order of the pair so that keys are first sorted by the left word, and then by the right word.
- Given this ordering, we can easily detect if all pairs associated with the word we are conditioning on (**wi**) have been encountered.
- At that point we can go back through the in-memory buffer, compute the relative frequencies, and then emit those results in the final key-value pairs.

Order Inversion Pattern in the stream

- It is so named because through proper coordination, we can access the result of a computation in the reducer (for example, an aggregate statistic) before processing the data needed for that computation.
- The key insight is to convert the sequencing of computations into a sorting problem.

Get the Word counts counts out first (as they will be denominator for relative frequency)

key	values	
$(\text{dog}, *)$	[6327, 8514, ...]	compute marginal: $\sum_{w'} N(\text{dog}, w') = 42908$
$(\text{dog}, \text{aardvark})$	[2,1]	$f(\text{aardvark}\|\text{dog}) = 3/42908$
$(\text{dog}, \text{aardwolf})$	[1]	$f(\text{aardwolf}\|\text{dog}) = 1/42908$
...		
$(\text{dog}, \text{zebra})$	[2,1,1,1]	$f(\text{zebra}\|\text{dog}) = 5/42908$
$(\text{doge}, *)$	[682, ...]	compute marginal: $\sum_{w'} N(\text{doge}, w') = 1267$
...		

Figure 3.12: Example of the sequence of key-value pairs presented to the reducer in the pairs algorithm for computing relative frequencies. This illustrates the application of the order inversion design pattern.

f(B|A): "Pairs": avoids in-memory reducer problem

Get the Word counts counts out first (as they will be denominator for relative frequency

(a, *) → 3
(a, *) → 12
…..

(a, *) → 32 …..

Reducer holds this value in memory

(a, b_1) → 3
(a, b_1) → 12
(a, b_3) → 7
(a, b_4) → 1
…

(a, b_2) → 15 / 32 #3+12
(a, b_3) → 7 / 32
(a, b_4) → 1 / 32
…

- **For this to work:**
  - Must emit extra (a, *) for every co-occurrence b_n in mapper
  - Must make sure all a's get sent to same reducer (use custom partitioner)
  - Must make sure (a, *) comes first (define sort order)
  - Must hold state in reducer across different key-value pairs

AKA Secondary sort

"Order Inversion"

- **Common design pattern**
  - Computing relative frequencies requires marginal counts
  - But marginal cannot be computed until you see all counts
  - Buffering is a bad idea!
  - Trick: getting the marginal counts to arrive at the reducer before the joint counts
- **Optimizations**
  - Apply in-memory combining pattern to accumulate marginal counts
  - Should we apply combiners?

Another Try: "Stripes"

- **Idea: group together pairs into an associative array**

  (a, b) → 1
  (a, c) → 2
  (a, d) → 5
  (a, e) → 3
  (a, f) → 2

  a → { b: 1, c: 2, d: 5, e: 3, f: 2 }

- **Each mapper takes a sentence:**
  - Generate all co-occurring term pairs
  - For each term, emit a → { b: $count_b$, c: $count_c$, d: $count_d$ … }
- **Reducers perform element-wise sum of associative arrays**

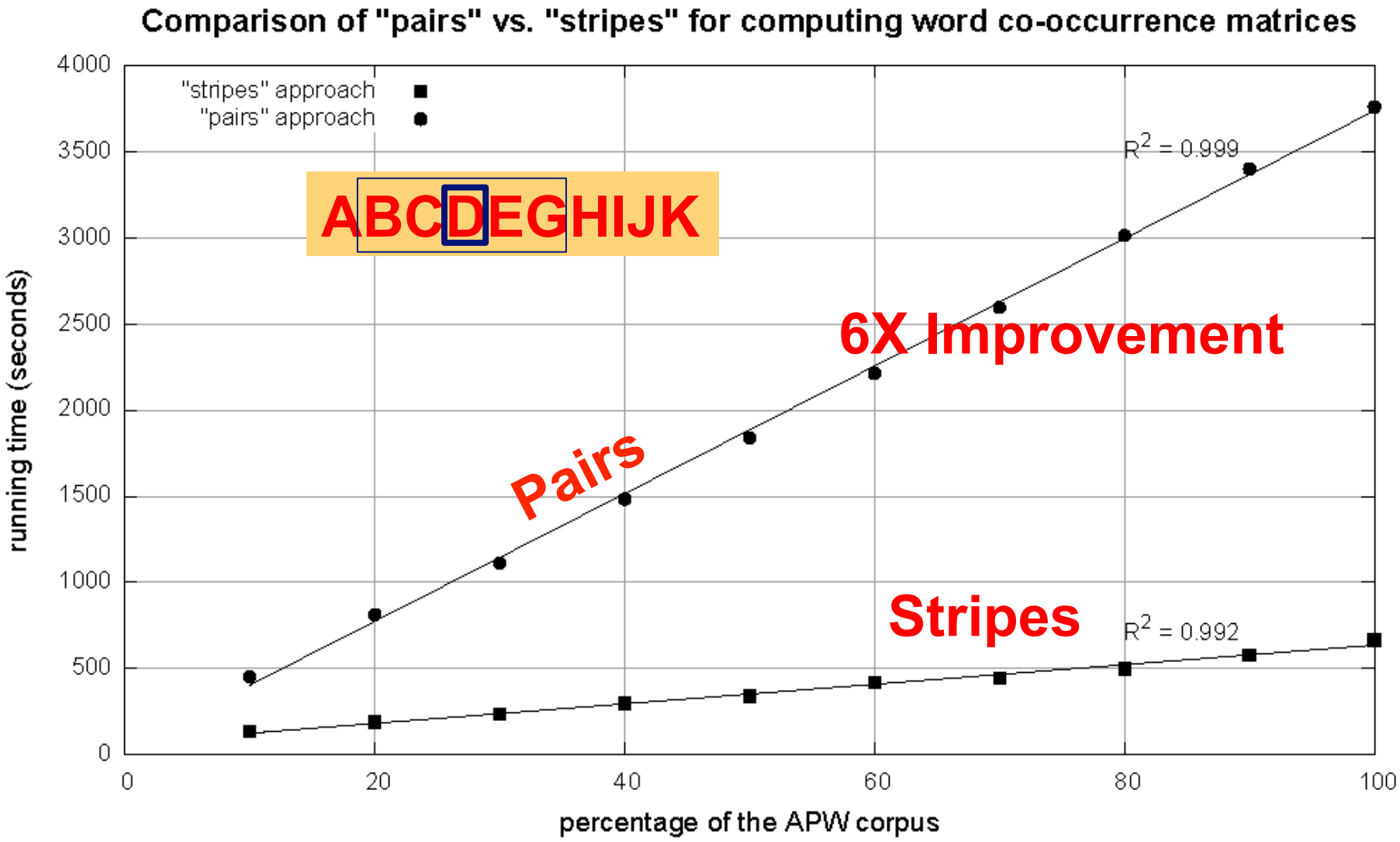
a → { b: 1, d: 5, e: 3 }
\+ a → { b: 1, c: 2, d: 2, f: 2 }
a → { b: 2, c: 2, d: 7, e: 3, f: 2 }

Key: cleverly-constructed data structure brings together partial results

- **Stripes: Million cooc of a: 10^6 co-occurring terms:**

- **Stripes A: {b1:count, ….b10: count}**
  - Transmit a once
  - Sort

- **Pairs (a, b1, count); (a, b2, count); …(a, b10, count)**
  - Sorting is 10X
  - (a, b1, count)
  - (a, b1, count)
  - (a, b1, count)
  - (a, b1, count)
- **Transmit a 10 times**



Cluster size: 38 cores (19 slave nodes with two cores)
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

6X improvement on entire corpus

Description	Pairs	Stripes
CPU time on 38 cores	62 Minutes	11 Minutes
Mappers	2.6 Billion KV pairs (32GB)	653M KV Pairs (48GB)
After Combiners	1.1B KV pairs	29M KV Pairs
Co-occurrence matrix	1.42M word pairs	1.42M word pairs

- The mappers in the pairs approach generated 2.6 billion intermediate key-value pairs totaling 31.2 GB. After the combiners, this was reduced to 1.1 billion key-value pairs, which quantifies the amount of intermediate data transferred across the network. In the end, the reducers emitted a total of 142 million final key-value pairs (the number of non-zero cells in the co-occurrence matrix).

6X improvement on entire corpus

Description	Pairs	Stripes
CPU time on 38 cores	62 Minutes	11 Minutes
Mappers	2.6 Billion KV pairs (32GB)	653M KV Pairs (48GB)
After Combiners	1.1B KV pairs	29M KV Pairs
Co-occurrence matrix	1.42M word pairs	1.42M word pairs

- The mappers in the pairs approach generated 2.6 billion intermediate key-value pairs totaling 31.2 GB. After the combiners, this was reduced to 1.1 billion key-value pairs, which quantifies the amount of intermediate data transferred acro a total of 1 -zero cells

As expected, the stripes approach provided more opportunities for combiners to aggregate intermediate results, thus greatly reducing network traffic in the shuffle and sort phase.

Summary Lecture 3

- **Plus**
- **Patterns**
  - Local Aggregation
    - Combiners and In-Mapper Combining
    - Algorithmic Correctness with Local Aggregation
    - Number of tasks and In-memory mapper
  - Pairs and Stripes
  - Computing Relative Frequencies
  - Order Inversion Pattern in the stream
  - Secondary sorts

Synchronization and communication

- PLUS
- Constructing complex keys and values that bring together data necessary for a computation. This is used in all of the above design patterns.
- Controlling the partitioning of the intermediate key space. This is used in order inversion and value-to-key conversion.
- Controlling the sort order of intermediate keys. This is used in order inversion and secondary sorting.

Conclusion

- This concludes our overview of MapReduce algorithm design.
- It should be clear by now that although the programming model forces one to express algorithms in terms of a small set of rigidly-defined components,
- there are many tools at one's disposal to shape the flow of computation.

- These patterns apply not JUST to Hadoop MapReduce but also to MRJob, and Spark as we shall see during the rest of this class

Live Session Outline

- **Housekeeping**
  - Start RECORDING (bonus points for reminding me!)
  - Office hours: Mondays at 5:30PM. Please submit Qs by Sunday at 5:30PM
  - Homework 1&2: Naive Bayes Experts (grades due on Saturday)
- **Peer Grading**
  - HW1 peer grading
  - HW2 Answer reviews
- **Week 3**
  - Hadoop useful stuff/Trivia
    - Blocks sizes
    - Total Sort
    - Understanding Hadoop via Counters
    - Hadoop File passing pattern
    - Number of mappers and reducers
  - Async lecture recap plus Q&A: Pairs/Secondary keys/Inversion pattern
  - Association rule mining via Apriori
- **Next week (Cloud: MrJob)**
- **Wrapup: Finish RECORDING (bonus points!)**

- **End of live session**

Association rule mining via Apriori

Data Mining
Association Analysis: Basic Concepts and Algorithms

Summary Points for APriori

- **Item set: co-occurring items**
- **Association rule**
  - An implication expression of the form X → Y, where X and Y are itemsets
- **Efficient way to find association rules**
- **Support and Confidence**

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{\text{Milk, Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|\text{T}|} = \frac{2}{5} = 0.8$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{4}{4} = 1.0$$

References

- **Chapter 6. Association Analysis: Basic Concepts and Algorithms (612KB)**
  - http://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf
  - https://www.dropbox.com/s/0h348q8597gci3g/Kumar-Data-Mining-Book-%20chapter-6.pdf?dl=0

Association Rule Mining

- **Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction**

Market-Basket transactions

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Implication means co-occurrence, not causality!

Definition: Frequent Itemset + Support

- **Itemset**
  - A collection of one or more items
    - Example: {Milk, Bread, Diaper}
  - k-itemset
    - An itemset that contains k items
- **Support count (σ)**
  - Frequency of occurrence of an itemset
  - E.g. σ({Milk, Bread,Diaper}) = 2
- **Support**
  - Fraction of transactions that contain an itemset
  - E.g. s({Milk, Bread, Diaper}) = 2/5
- **Frequent Itemset**
  - An itemset whose support is greater than or equal to a *minsup* threshold

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Definition: Association Rule and Metrics

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - Example:
    {Milk, Diaper} → {Beer}

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

- **Rule Evaluation Metrics**
  - Support (s)
    - Fraction of transactions that contain both X and Y
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X

Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|\text{T}|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

Minimum Confidence Threshold

- Confidence is defined as the measure of certainty or trustworthiness associated with each discovered pattern.

IF A ⇒ B

$$confidence(A \Rightarrow B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{\#\_tuples\_containing\_A}$$

Minimum Support Threshold

- The support of an association pattern refers to the percentage of task-relevant data transactions for which the pattern is true.

IF A⇒ B

$$support\ (A \Rightarrow B) = \frac{\#\_tuples\_containing\_both\_A\_and\_B}{total\_\#\_of\_tuples}$$

Association Rule Mining Task

- **Given a set of transactions T, the goal of association rule mining is to find all rules having**
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

- **Brute-force approach:**
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds

  ⇒ Computationally prohibitive!

Mining Association Rules

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:

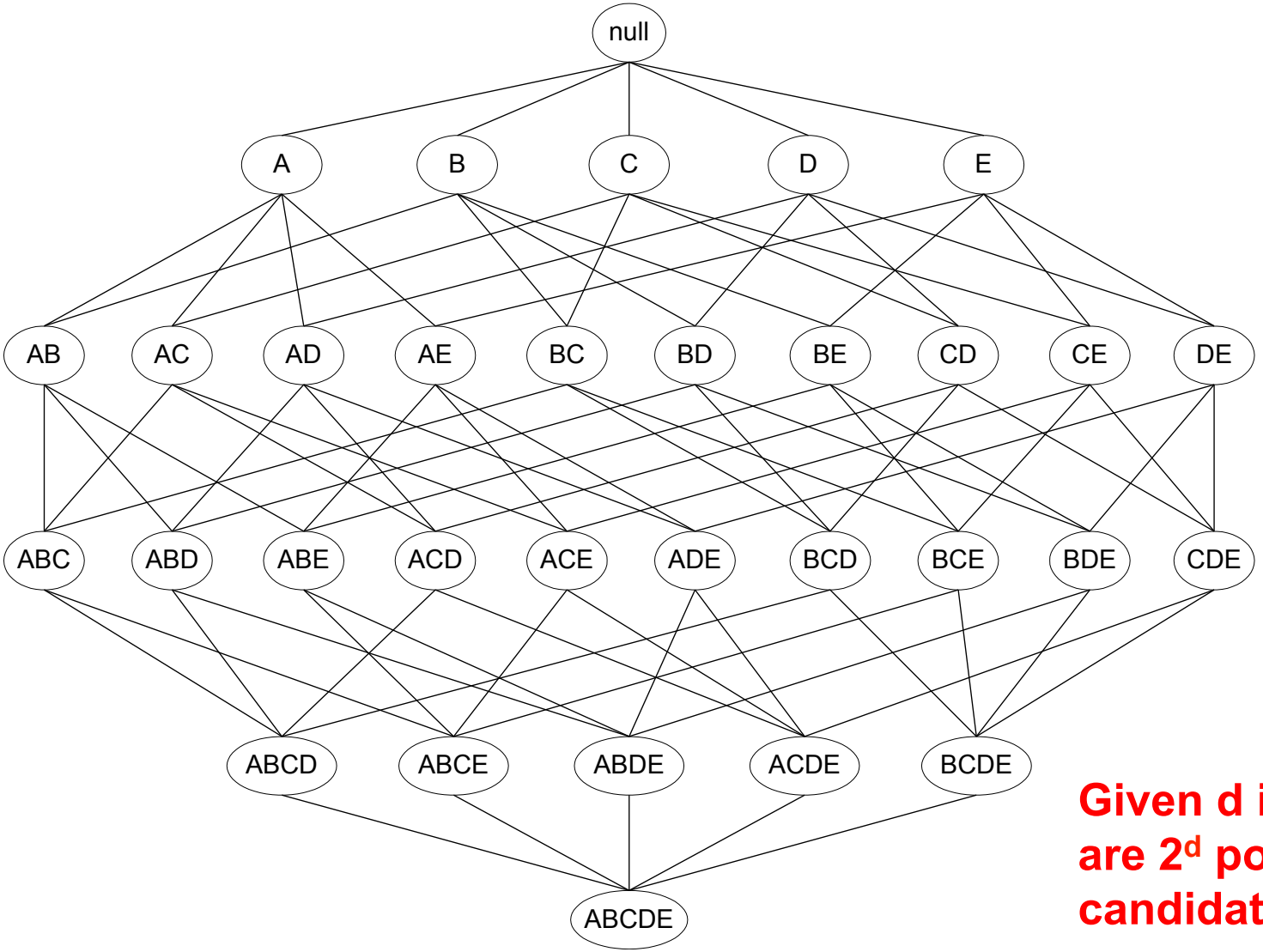
- All the above rules are binary partitions of the same itemset:
  {Milk,Diaper} → {Beer} is a binary partition of {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- **Two-step approach:**
  1. Frequent Itemset Generation
     - Generate all itemsets whose support ≥ minsup
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

- **Frequent itemset generation is still computationally expensive**

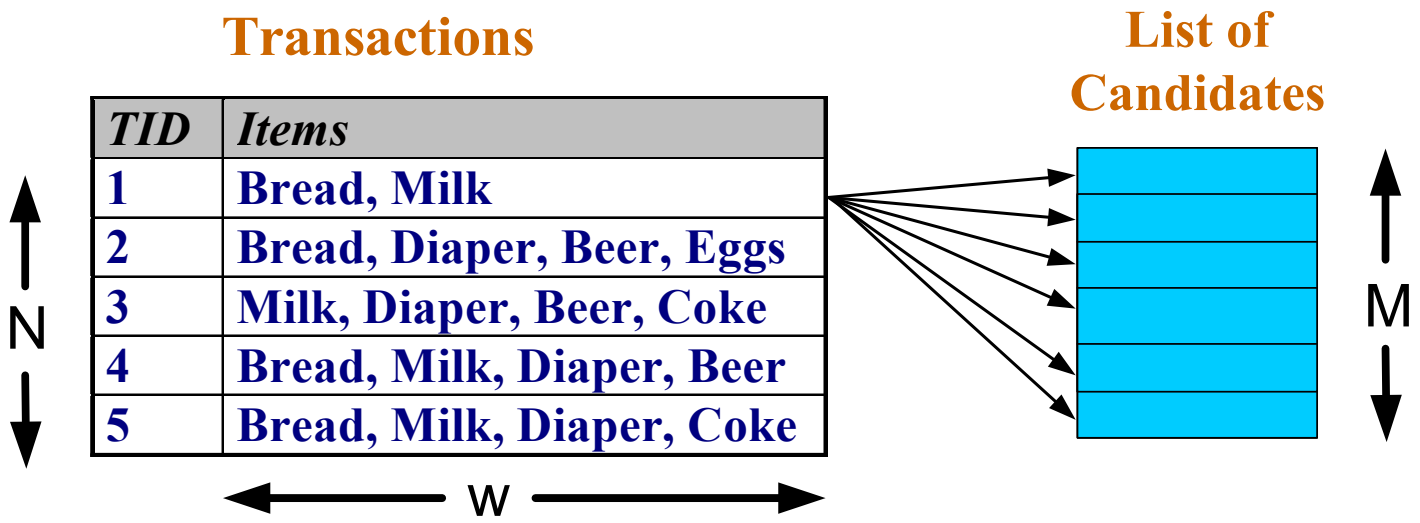
Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

Frequent Itemset Generation

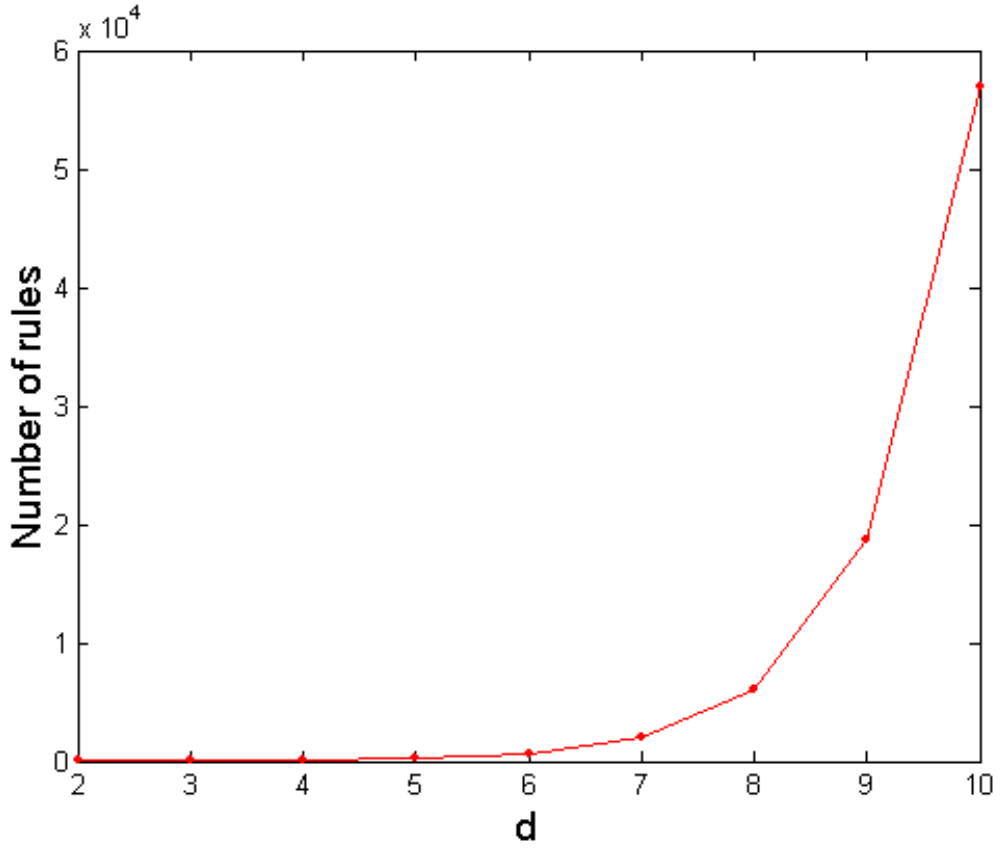
- **Brute-force approach:**
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database



  - Match each transaction against every candidate
  - Complexity ~ O(NMw) => Expensive since M = 2^d !!!

Computational Complexity: 6 items

- **Given d unique items:**
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$

$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

$6_{Choose}5$ +$5_{Choose}4$+…
$5_{Choose}4$ + $4_{Choose}3$ …
Etc..

Frequent Itemset Generation Strategies

- **Reduce the number of candidates (M)**
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- **Reduce the number of transactions (N)**
  - Reduce size of N as the size of itemset increases
  - Used by DHP and vertical-based mining algorithms
- **Reduce the number of comparisons (NM)**
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

Reducing Number of Candidates

- **A priori principle:**
  - If an itemset is frequent, then all of its subsets must also be frequent

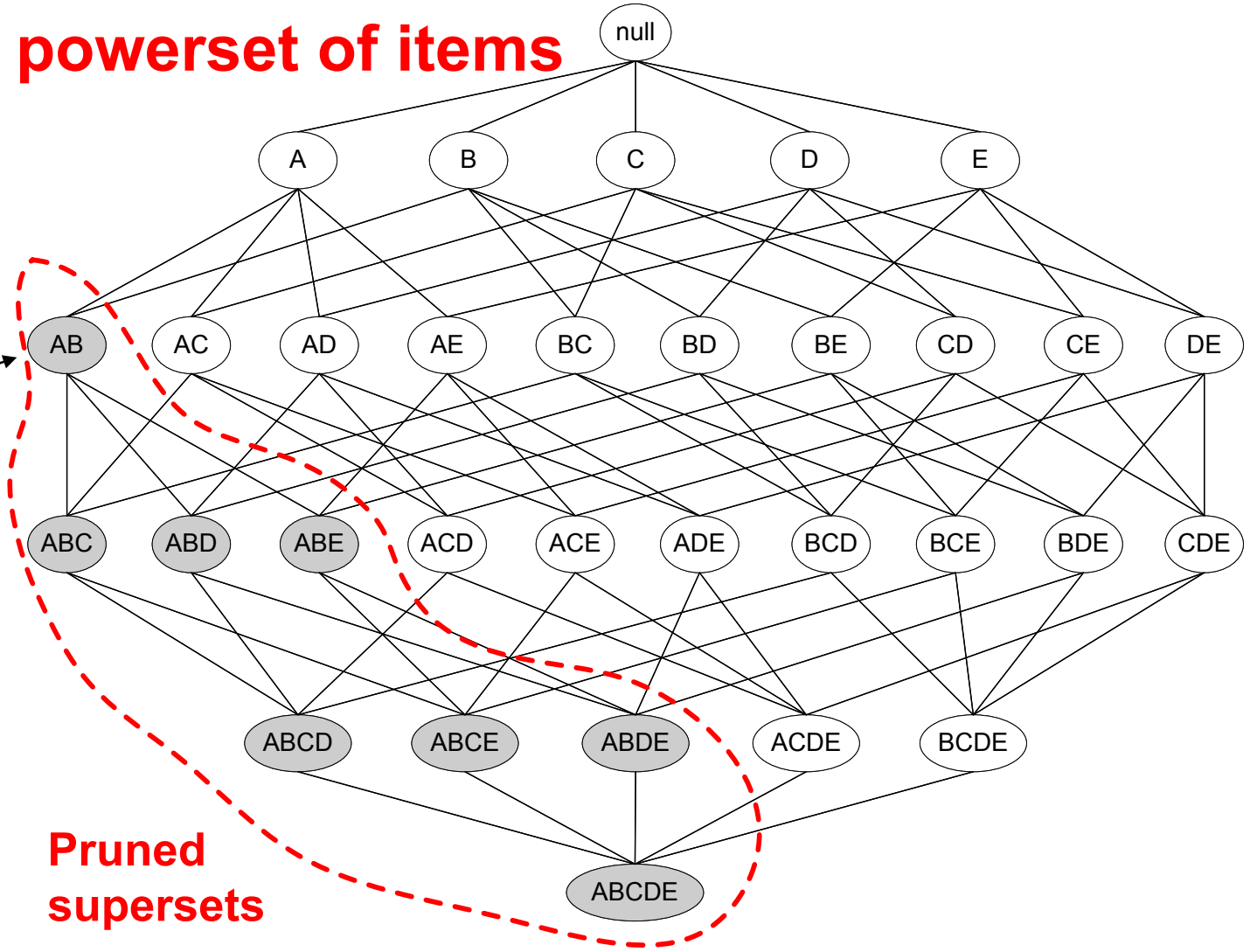
- **A priori principle holds due to the following property of the support measure:**

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Illustrating Apriori Principle

Explore the powerset of items
2^d itemsets

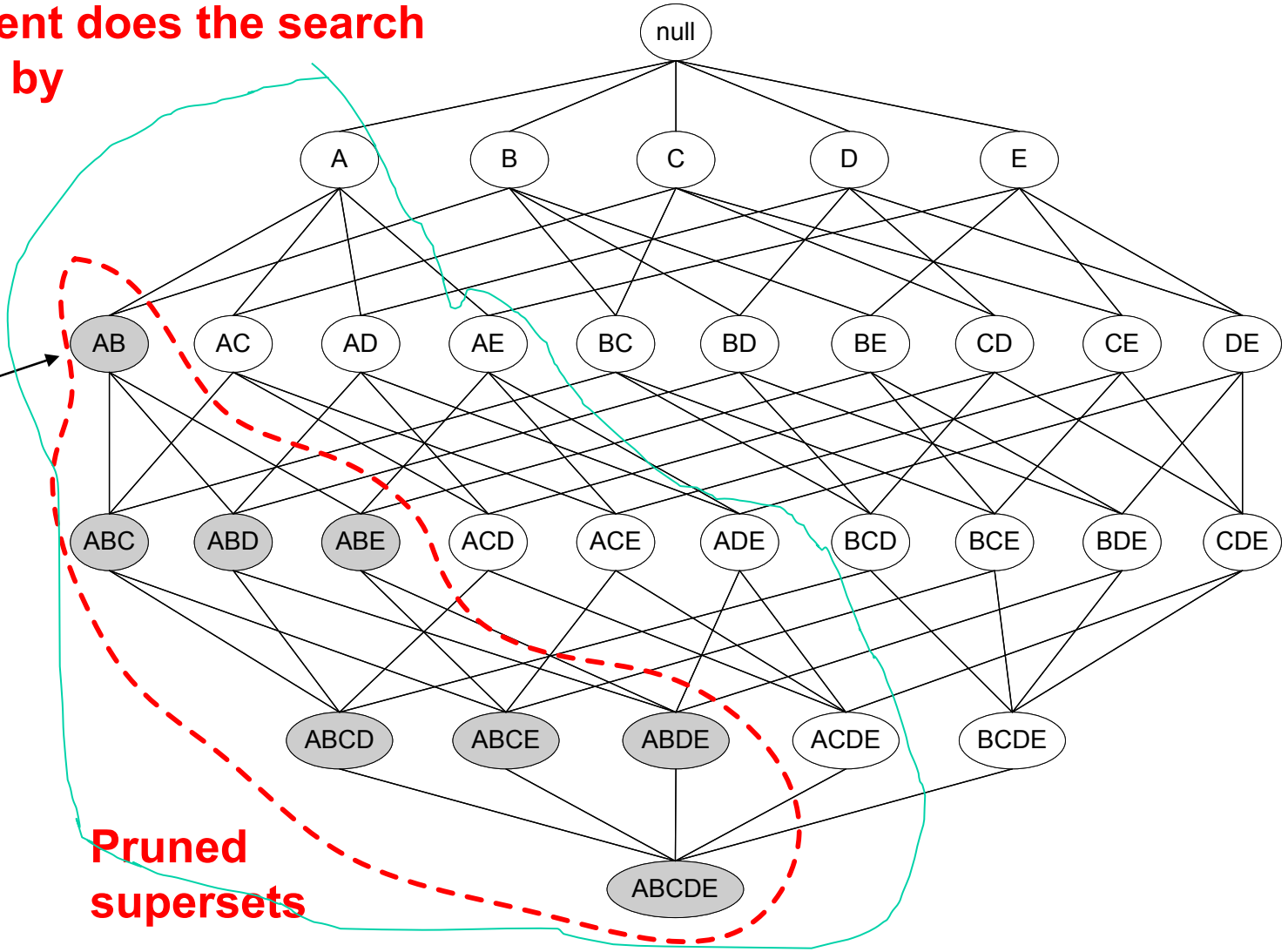


Found to be Infrequent

Pruned supersets

Illustrating Apriori Principle

If A is infrequent does the search space reduce by
(A) 20%
(B) 50%

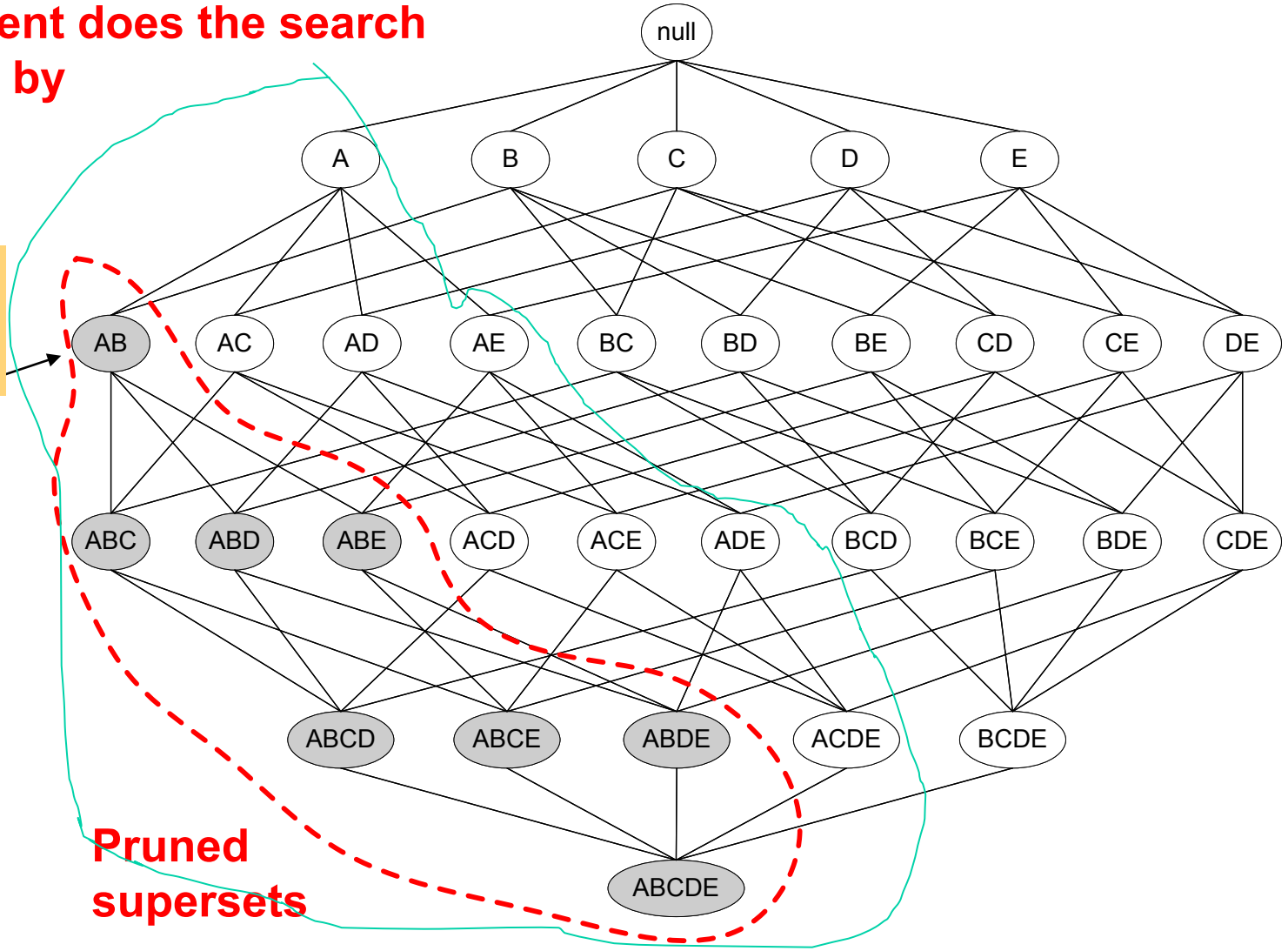


Illustrating Apriori Principle

If A is infrequent does the search space reduce by
(A) 20%
(B) 50%

50%: It goes from 2^d itemsets to 2^{d-1}

Found to be Infrequent



Apriori Algorithm: Example

Table 6.1. An example of market basket transactions.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

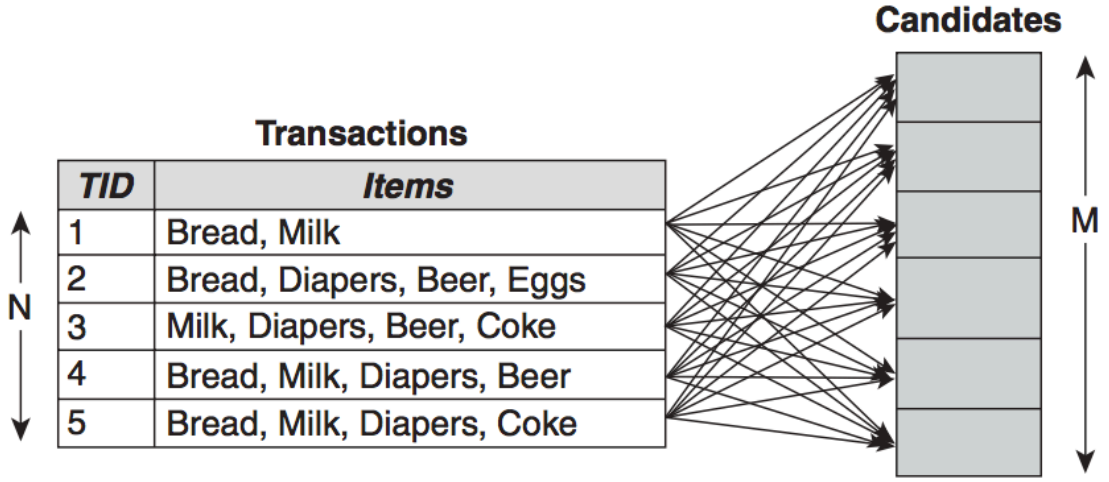


Figure 6.2. Counting the support of candidate itemsets.

https://www.dropbox.com/s/0h348q8597gci3g/Kumar-Data-Mining-Book-%20chapter-6.pdf?dl=0

Illustrating Apriori Principle: Minimum Support = 3

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	2

If every subset (Brute-force) is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$ itemsets considered
With support-based pruning (aPriori principle),
6 + 6 + 1 = 13 itemsets considered

Apriori Principle:

all 2-itemsets that makeup a 3-itemset must have min-support

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,
$${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$$
With support-based prun
$$6 + 6 + 1 = 13$$

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

Look at all 2-itemset that makeup a 3-itemset
E.g., (Bread, Milk, Diaper) make sure all of the following 2-itemsets have a support of 3 or more:

- **Bread, Milk (3)**
- **Bread, Diaper (3)**
- **Milk, Diaper (3)**

Illustrating Apriori Principle

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

How about {Bread milk beer}?

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based prun
6 + 6 + 1 = 13

Look at all 2-itemset that makeup a 3-itemset
E.g., (Bread, Milk, Diaper) make sure all of the following 2-itemsets have a support of 3 or more:

- **Bread, Milk (3)**
- **Bread, Diaper (3)**
- **Milk, Diaper (3)**

Illustrating Apriori Principle: Exercise

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Items (1-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

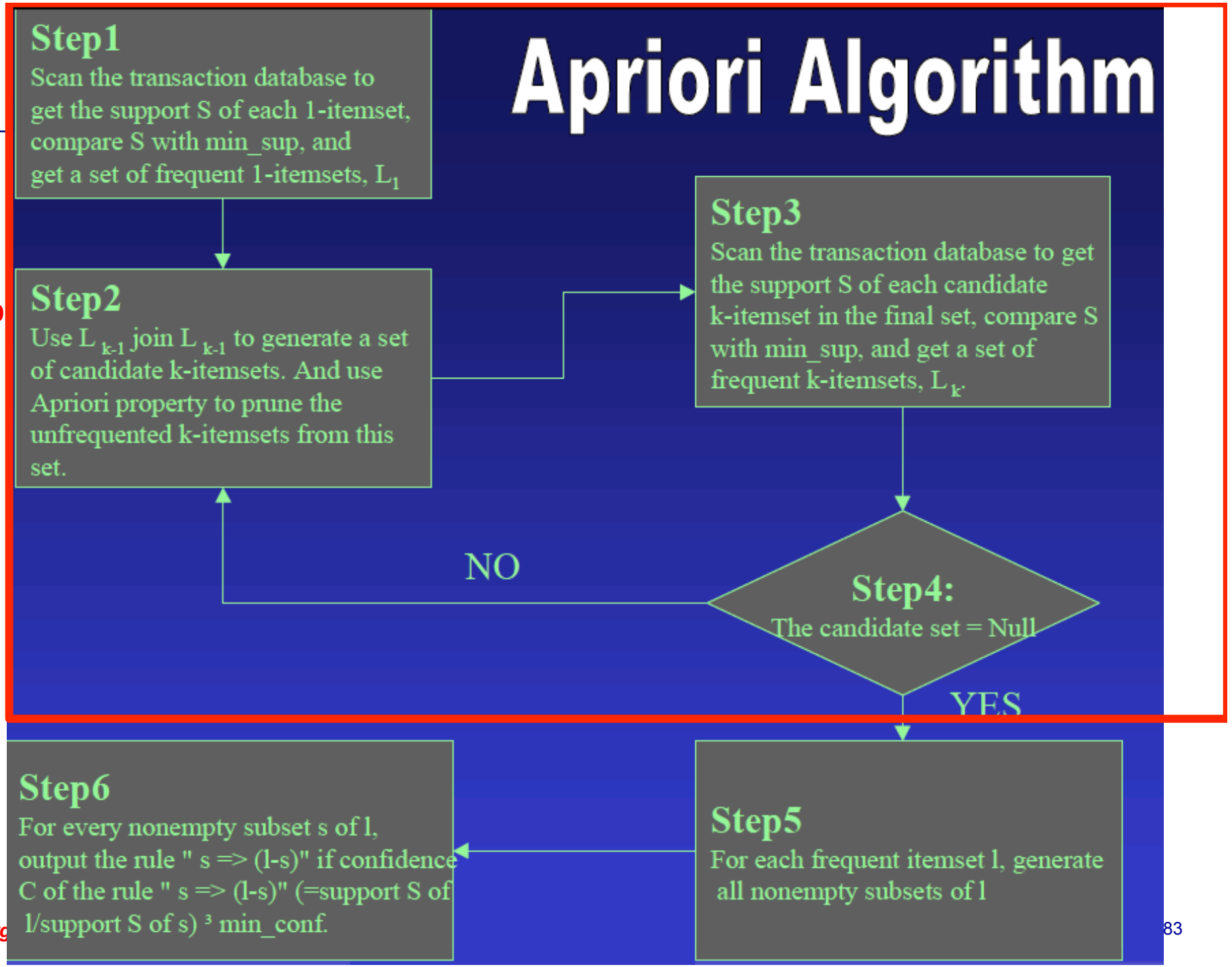
Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
6 + 6 + 1 = 13

Find all other 3 itemsets given this data

Apriori Algorithm

- **Phase 1: generate all itemsets > Min Support**
  - Let k=1
  - Generate frequent item sets of length 1
  - Repeat until no new frequent item sets are identified
    - Generate length (k+1) candidate itemsets from length k frequent item sets
    - Prune candidate item sets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent
- **Phase 2: generate all rules from discovered itemsets that have a confidence score > minimum confidence**
  - Each frequent itemset can generate $2^K - 2$ possible rules
  - can not leverage aPriori principle here; but clever heuristics exist
  - see page 249 in Kumar book for details of rule generation;


Apriori Algorithm
Phase 1: Itemsets generation
Phase 2: Rules
Step1
Scan the transaction database to get the support S of each 1-itemset, compare S with min_sup, and get a set of frequent 1-itemsets, L1
Step2
Use Lk-1 join Lk-1 to generate a set of candidate k-itemsets. And use Apriori property to prune the unfrequented k-itemsets from this set.
Step3
Scan the transaction database to get the support S of each candidate k-itemset in the final set, compare S with min_sup, and get a set of frequent k-itemsets, Lk.
Step4:
The candidate set = Null
NO
YES
Step5
For each frequent itemset l, generate all nonempty subsets of l
Step6
For every nonempty subset s of l, output the rule " s => (l-s)" if confidence C of the rule " s => (l-s)" (=support S of l/support S of s) ³ min_conf.

Factors Affecting Complexity

- **Choice of minimum support threshold**
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent item sets
- **Dimensionality (number of items) of the data set**
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- **Size of database**
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- **Average transaction width**
  - transaction width increases with denser data sets
  - This may increase max length of frequent item sets and traversals of hash tree (number of subsets in a transaction increases with its width)

Example 1

Apriori Algorithm

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

1-Itemsets	Sup-count
1	6
2	7
3	6
4	2
5	2

Example 1

Apriori Algorithm

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

1-Itemsets	Sup-count
1	6
2	7
3	6
4	2
5	2

Join

2-Itemsets	Sup-count
1, 2	4
1, 3	4
1, 4	1
1, 5	2
2, 3	4
2, 4	2
2, 5	2
3, 4	0
3, 5	1
4, 5	0

Min support = 2

Prune

Frequent 2-Itemsets	Sup-count
1, 2	4
1, 3	4
1, 5	2
2, 3	4
2, 4	2
2, 5	2

Join

Frequent 3-Itemsets	Sup-count
1, 2, 3	2
1, 2, 5	2

Example 2

Problem data

An example with a transactional data D contents a list of 5 transactions in a supermarket .

TID	List of items (item_Ids)
1	Beer(I1), Diaper(I2), Baby Powder(I3), Bread(I4), Umbrella(I5)
2	Diaper(I2), Baby Powder(I3)
3	Beer(I1), Diaper(I2), Milk(I6)
4	Diaper(I2), Beer(I1), Detergent(I7)
5	Beer(I1), Milk(I6), Coca Cola (I8)

The University of Iowa Intelligent Systems Laboratory

Solution Procedure

Step1 *min_sup* = 40% (2/5)

C1 → L1

Item_ID	Item	Support
I1	Beer	4/5
I2	Diaper	4/5
I3	Baby powder	2/5
~~I4~~	~~Bread~~	~~1/5~~
~~I5~~	~~Umbrella~~	~~1/5~~
I6	Milk	2/5
~~I7~~	~~Detergent~~	~~1/5~~
~~I8~~	~~Coca cola~~	~~1/5~~

Item_ID	Item	Support
I1	Beer	4/5
I2	Diaper	4/5
I3	Baby	2/5
I6	Milk	2/5

The University of Iowa Intelligent Systems Laboratory

Solution Procedure

Step2

C2

Item_ID	Item	Support
{I1, I2}	Beer, Diaper	3/5
~~{I1, I3}~~	~~Beer, Baby powder~~	~~1/5~~
{I1, I6}	Beer, Milk	2/5
{I2, I3}	Diaper, Baby powder	2/5
~~{I2, I6}~~	~~Diaper, Milk~~	~~1/5~~
~~{I3, I6}~~	~~Baby powder, Milk~~	~~0~~

Step 3

L2

Item_ID	Item	Support
{I1, I2}	Beer, Diaper	3/5
{I1, I6}	Beer, Milk	2/5
{I2, I3}	Diaper, Baby powder	2/5

Solution Procedure

Step4: L2 is not Null, so repeat Step2

Item_ID	Item
{I1, I2, I3}	Beer, Diaper, Baby powder
{I1, I2, I6}	Beer, Diaper, Milk
{I1, I3, I6}	Beer, Baby powder, Milk
{I2, I3, I6}	Diaper, Baby powder, Milk

C3 =Null

Solution Procedure

Step 5

***min_sup*=40% *min_conf*=70%**

Item_ID	Item	support(A B)	support A	Confidence
I1 I2	Beer Diaper	60%	80%	75%
~~I1 I6~~	~~Beer Milk~~	~~40%~~	~~80%~~	~~50%~~
~~I2 I3~~	~~Diaper Baby powder~~	~~40%~~	~~80%~~	~~50%~~
I2 I1	Diaper Beer	60%	80%	75%
I6 I1	Milk Beer	40%	40%	100%
I3 I2	Baby powder Diaper	40%	40%	100%

Solution Procedure

Step 6

***min_sup*=40% *min_conf*=70%**

Strong ruls		Support	Confidence
I1=> I2	Beer=> Diaper	60%	75%
I2=> I1	Diaper=> Beer	60%	75%
I6 => I1	Milk=> Beer	40%	100%
I3 => I2	Baby powder=> Diaper	40%	100%

Combinations in Python

- **https://docs.python.org/2/library/itertools.html#itertools.combinations**

- **End of live session #3**