Peter Norvig

Google

Verification and Validation of AI Software


“How can we reinvent it knowing that **software** can play such an important role.”

BORDERS

BLOCKBUSTER®

Kodak

COLUMBIA

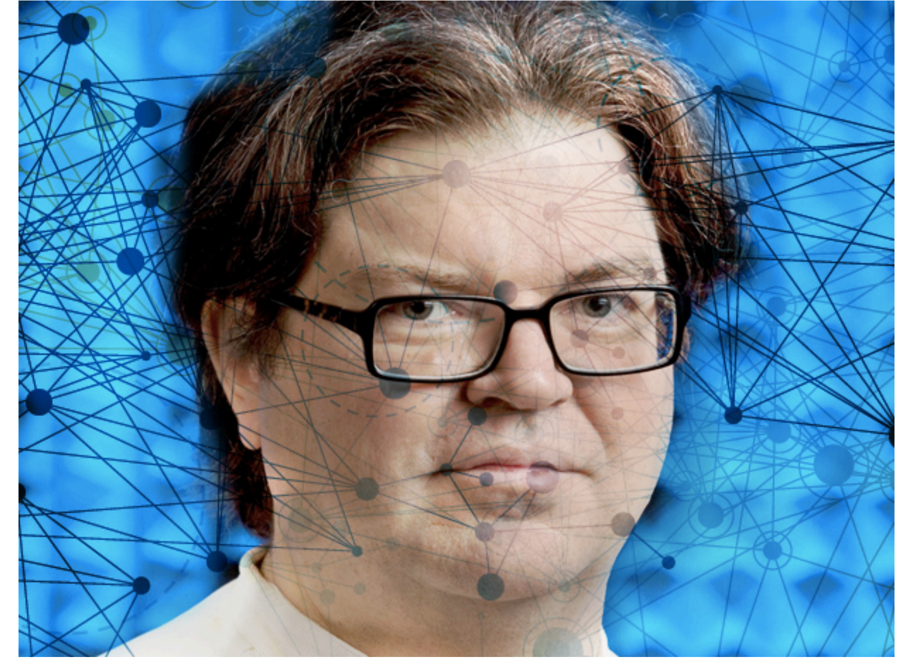
amazon

NETFLIX

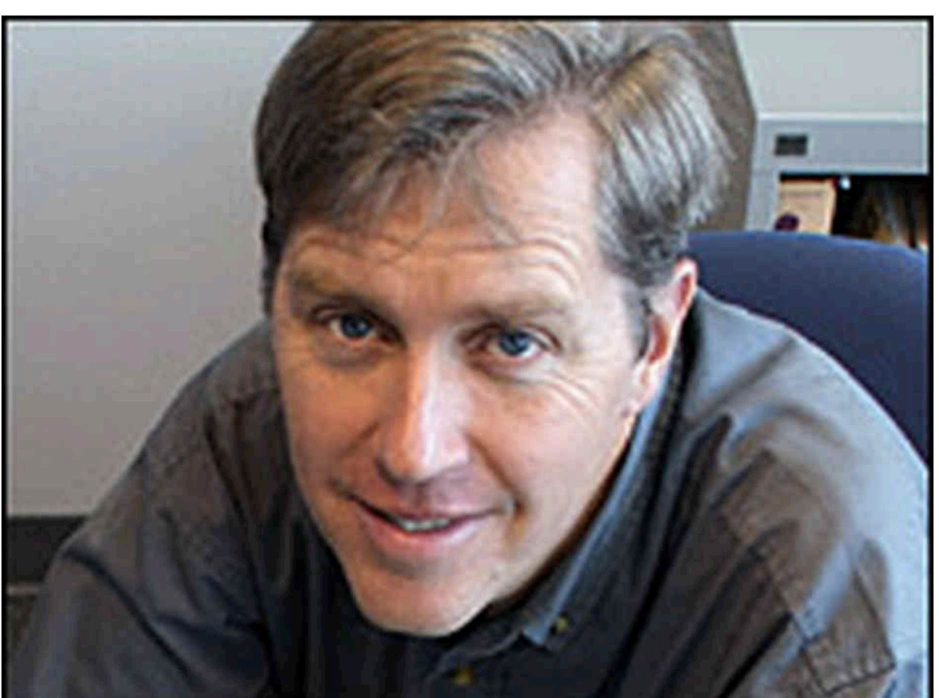
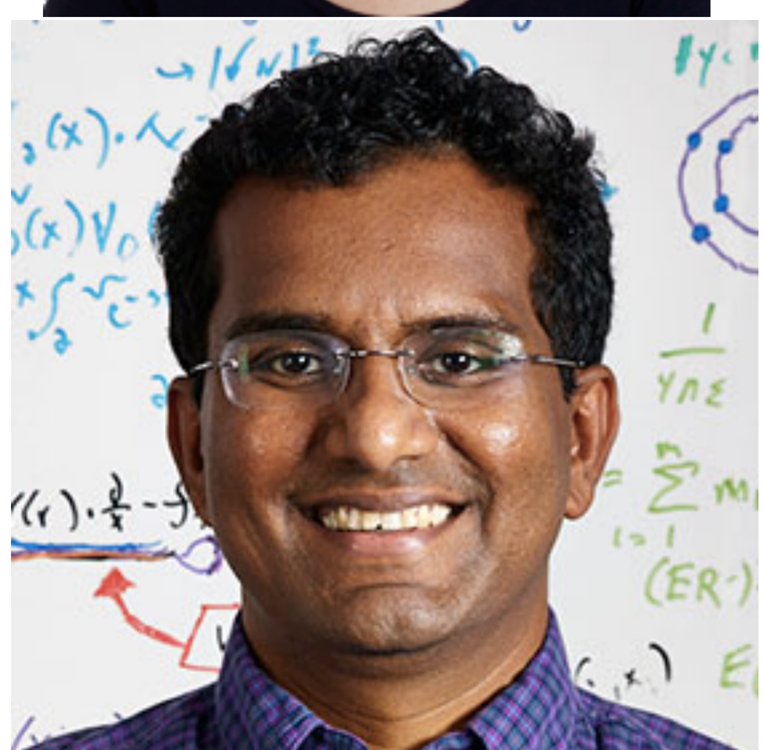
Insta

Spotify®







TechCrunch

"How can we reinvent it knowing that **artificial Intelligence** can play such an important role."

New Classes of Applications

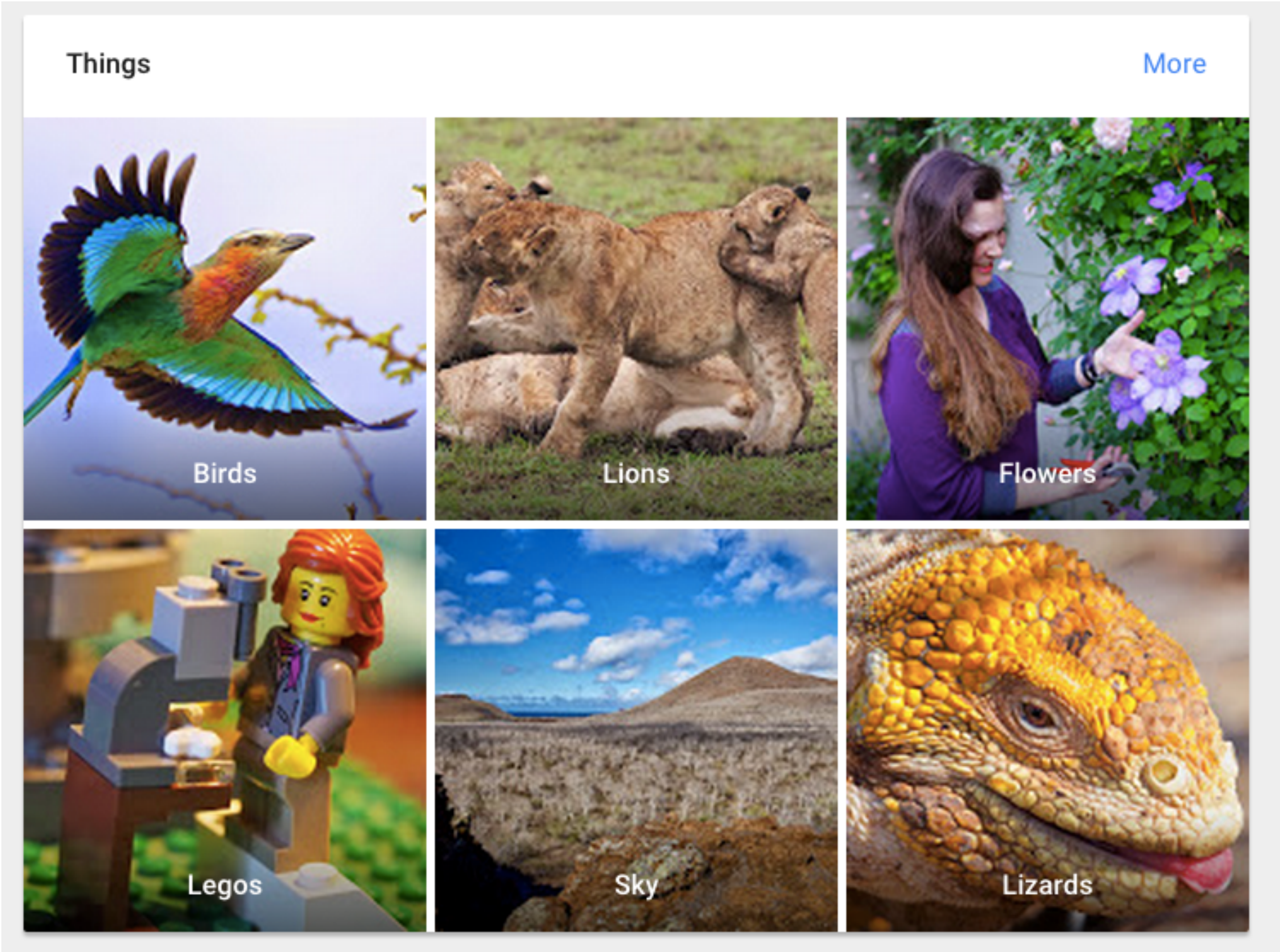
Computer Science:

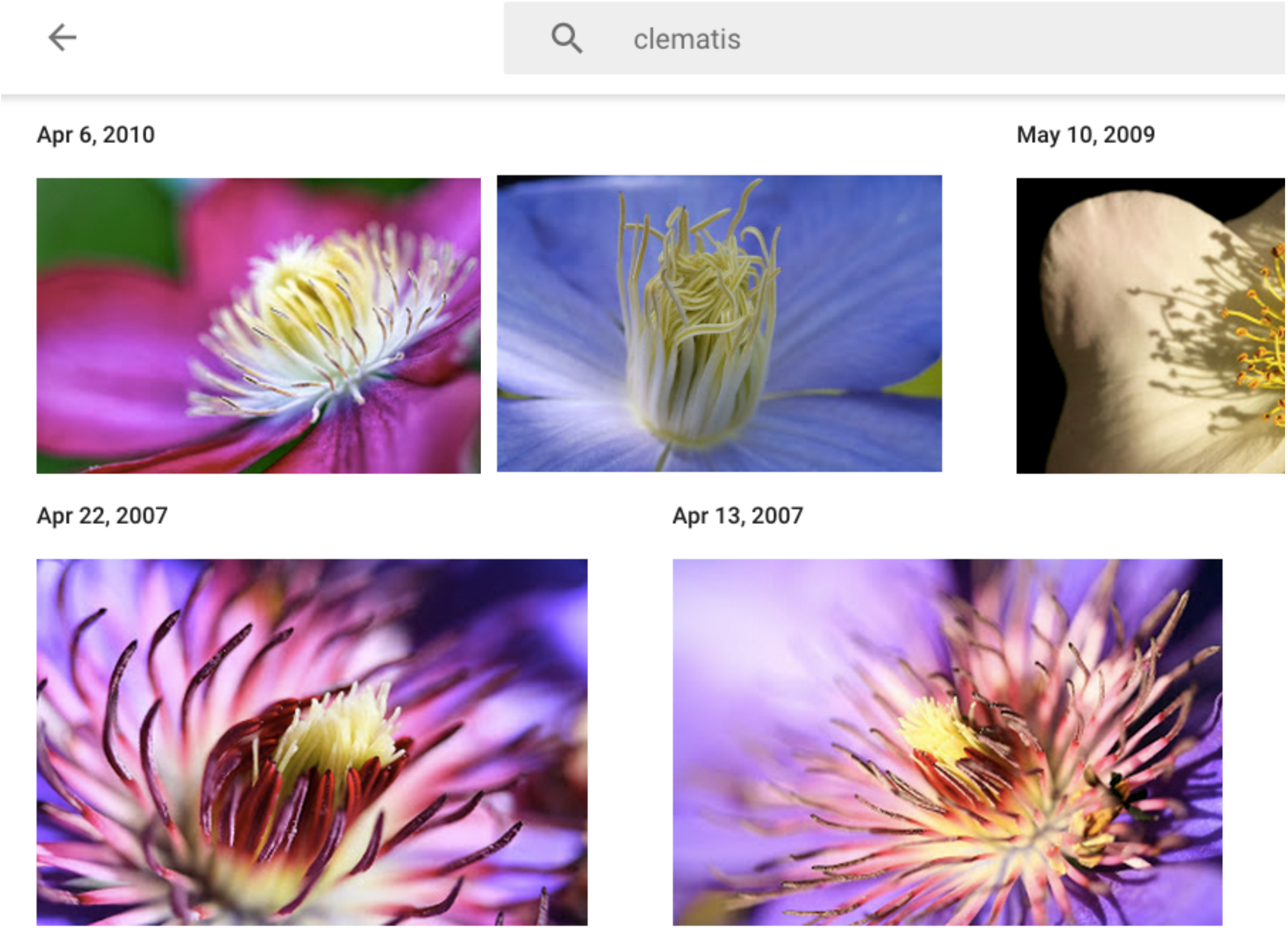
Doing the right thing, efficiently, when you can define what that means

Artificial Intelligence:

Doing the right thing, efficiently, when you don't know what to do

Visual Object Recognition


Things
More
Birds
Lions
Flowers
Legos
Sky
Lizards


clematis
Apr 6, 2010
May 10, 2009
Apr 22, 2007
Apr 13, 2007

nose

Jul 8, 2014

May 22, 2014

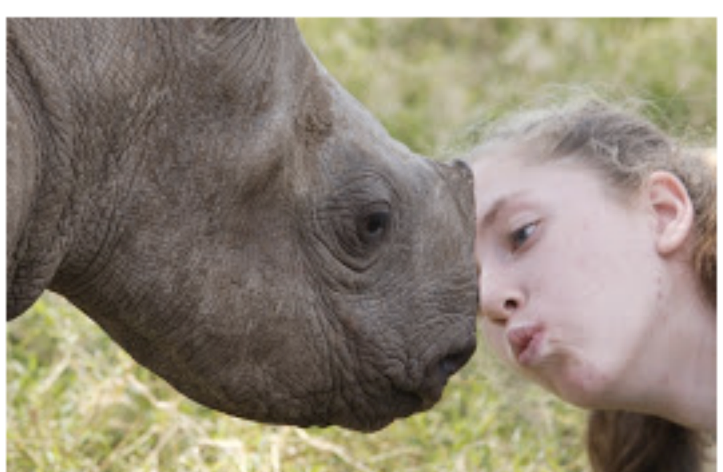
May 16, 2014

Jun 2, 2012

Sep 5, 2011

Mar 3, 2009

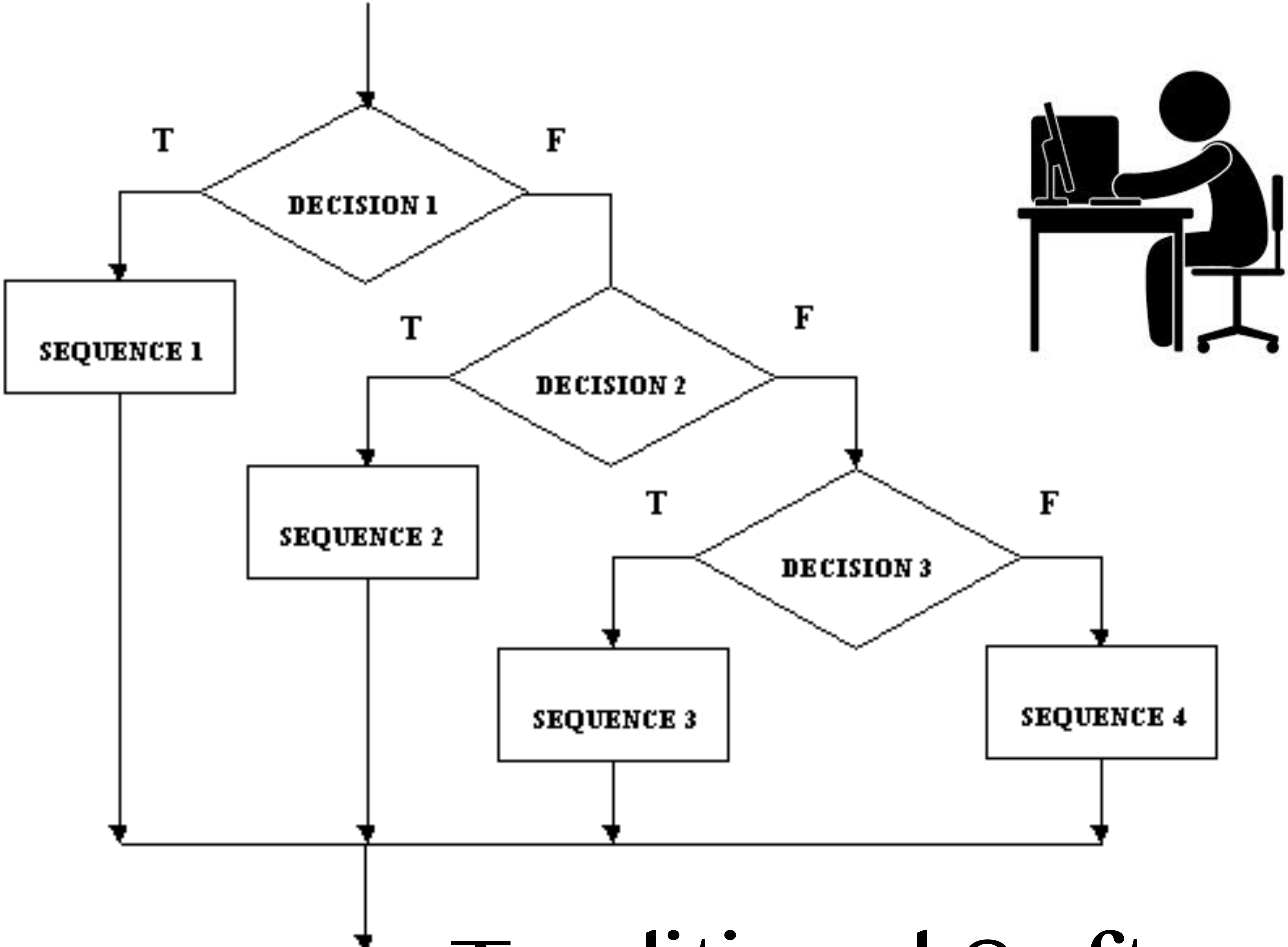
Aug 3, 2007



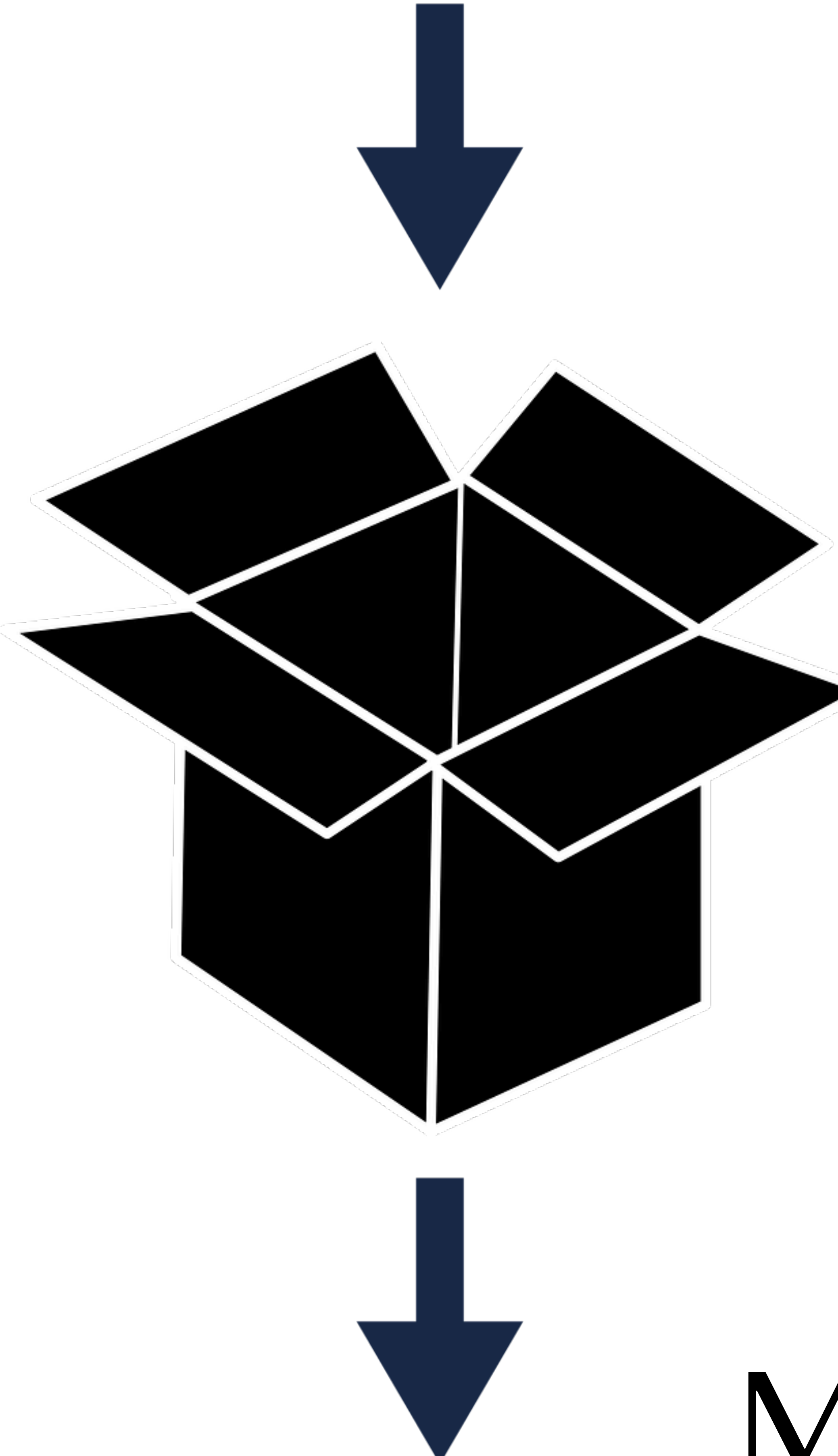
Jan 25, 2004

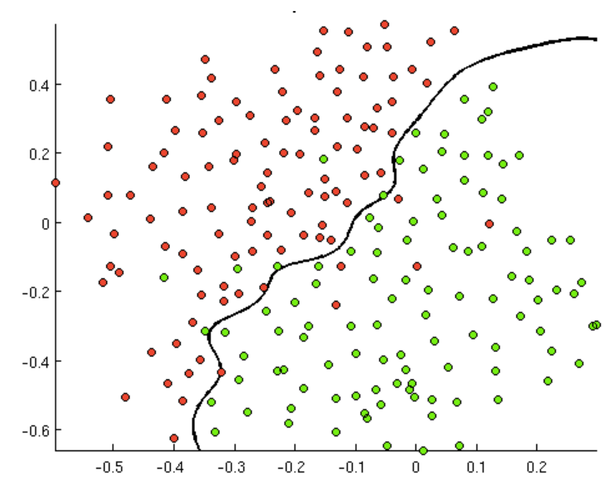
May 4, 2002

Traditional Software
vs
Machine Learning



Traditional Software



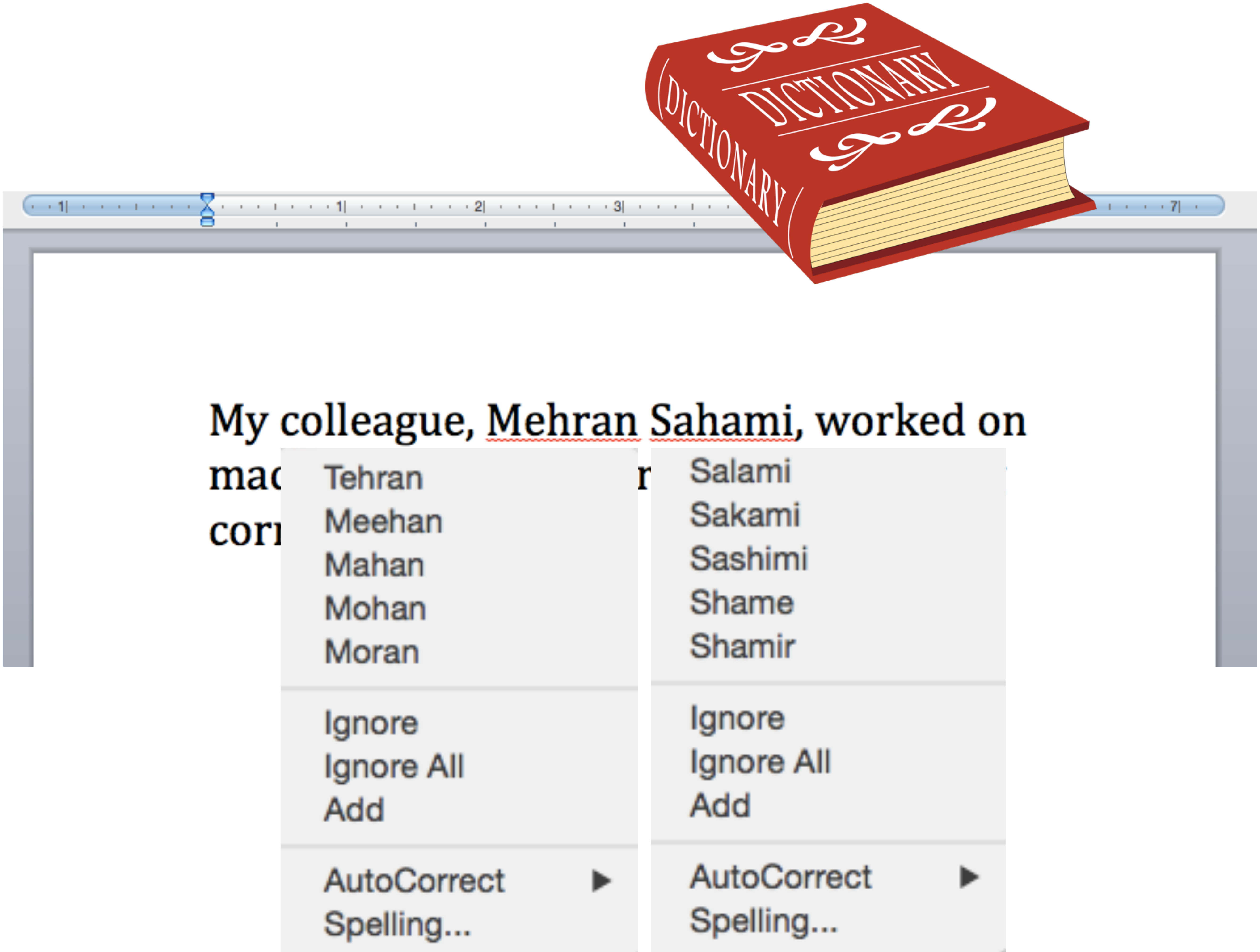


Machine Learning


2

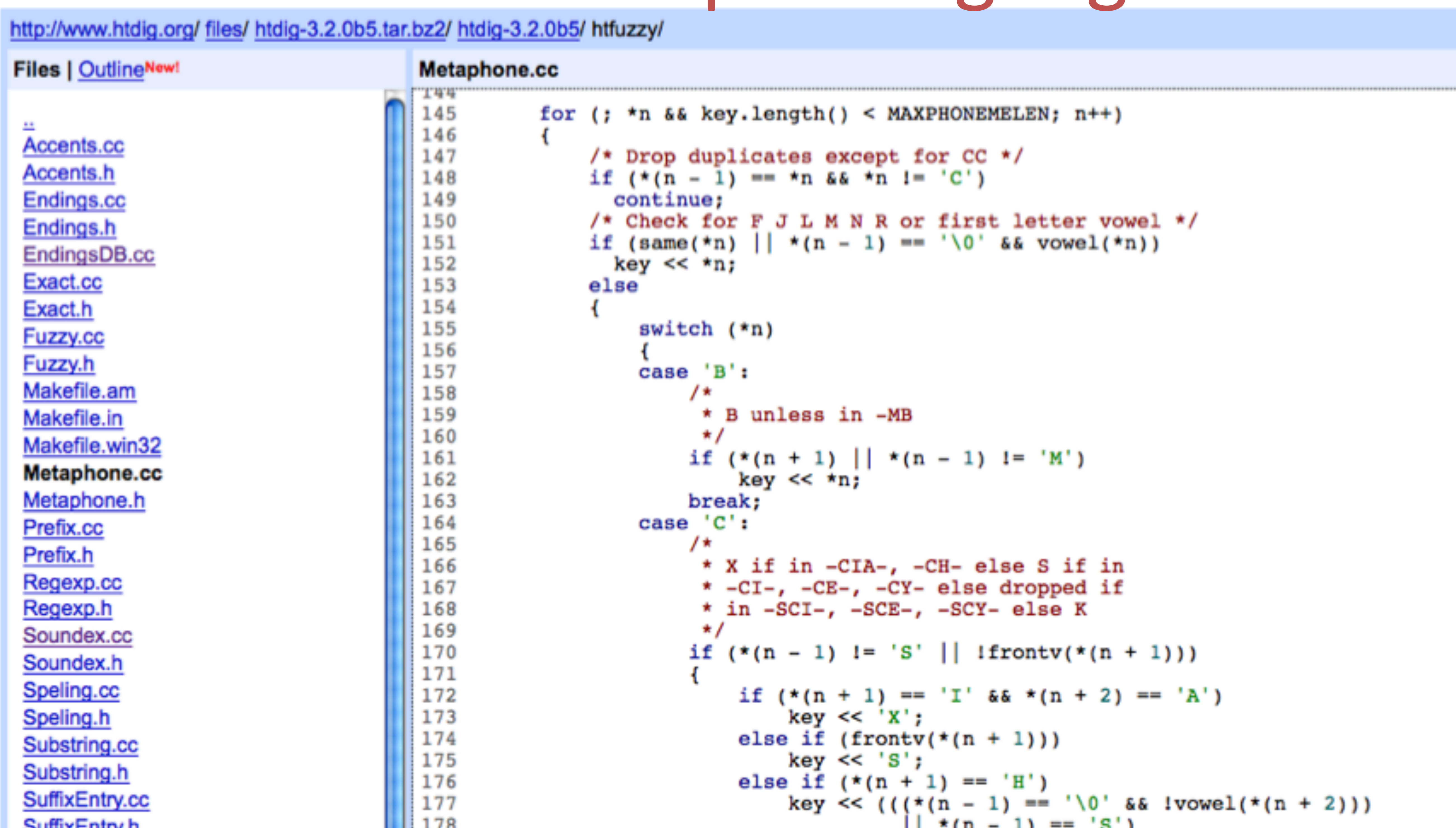
TaxSlayer
88
TaxSlayer.com
CAMARO
32
77
CAMRY
99
Just Imagine

Example: Spelling Correction


DICTIONARY
DICTIONARY
My colleague, Mehran Sahami, worked on
Tehran
Meehan
Mahan
Mohan
Moran
Ignore
Ignore All
Add
AutoCorrect
Spelling...
Salami
Sakami
Sashimi
Shame
Shamir
Ignore
Ignore All
Add
AutoCorrect
Spelling...

```
if (is_before('i', 'e') and
    not is_after('i', 'c')):
    return CORRECT
```

2000+ lines of code per language



http://www.htdig.org/ files/ htdig-3.2.0b5.tar.bz2/ htdig-3.2.0b5/ htfuzzy/

Files | Outline New!

..
Accents.cc
Accents.h
Endings.cc
Endings.h
EndingsDB.cc
Exact.cc
Exact.h
Fuzzy.cc
Fuzzy.h
Makefile.am
Makefile.in
Makefile.win32
Metaphone.cc
Metaphone.h
Prefix.cc
Prefix.h
Regexp.cc
Regexp.h
Soundex.cc
Soundex.h
Speling.cc
Speling.h
Substring.cc
Substring.h
SuffixEntry.cc
SuffixEntry.h

Metaphone.cc

```
        for (; *n && key.length() < MAXPHONEMELEN; n++)
        {
            /* Drop duplicates except for CC */
            if (*(n - 1) == *n && *n != 'C')
              continue;
            /* Check for F J L M N R or first letter vowel */
            if (same(*n) || *(n - 1) == '\0' && vowel(*n))
              key << *n;
            else
            {
                switch (*n)
                {
                case 'B':
                    /*
                     * B unless in -MB
                     */
                    if (*(n + 1) || *(n - 1) != 'M')
                        key << *n;
                    break;
                case 'C':
                    /*
                     * X if in -CIA-, -CH- else S if in
                     * -CI-, -CE-, -CY- else dropped if
                     * in -SCI-, -SCE-, -SCY- else K
                     */
                    if (*(n - 1) != 'S' || !frontv(*(n + 1)))
                    {
                        if (*(n + 1) == 'I' && *(n + 2) == 'A')
                            key << 'X';
                        else if (frontv(*(n + 1)))
                            key << 'S';
                        else if (*(n + 1) == 'H')
                            key << (((*(n - 1) == '\0' && !vowel(*(n + 2)))
```

Examples

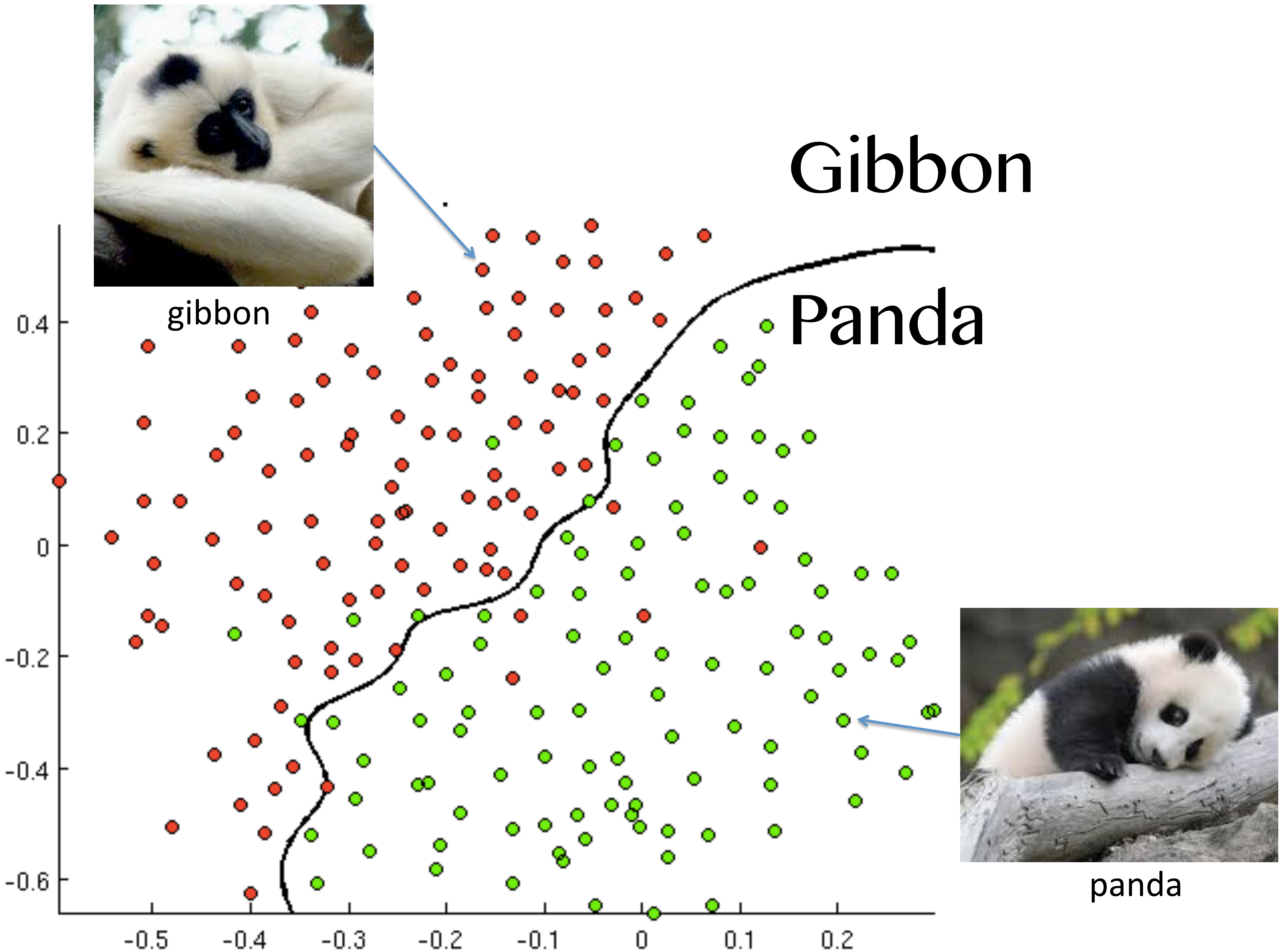

Google
Mehran Salami
mehran sahami
mehran sahami cs106a
mehran sahami java lectures
mehran sahami salary
En savoir plus
Résultats pour Mehran sahami
Essayez avec l'orthographe Mehran Salami
Images correspondant à mehran sahami
Signaler des images inappropriées
STANFORD
Plus d'images pour mehran sahami

21 lines of code total

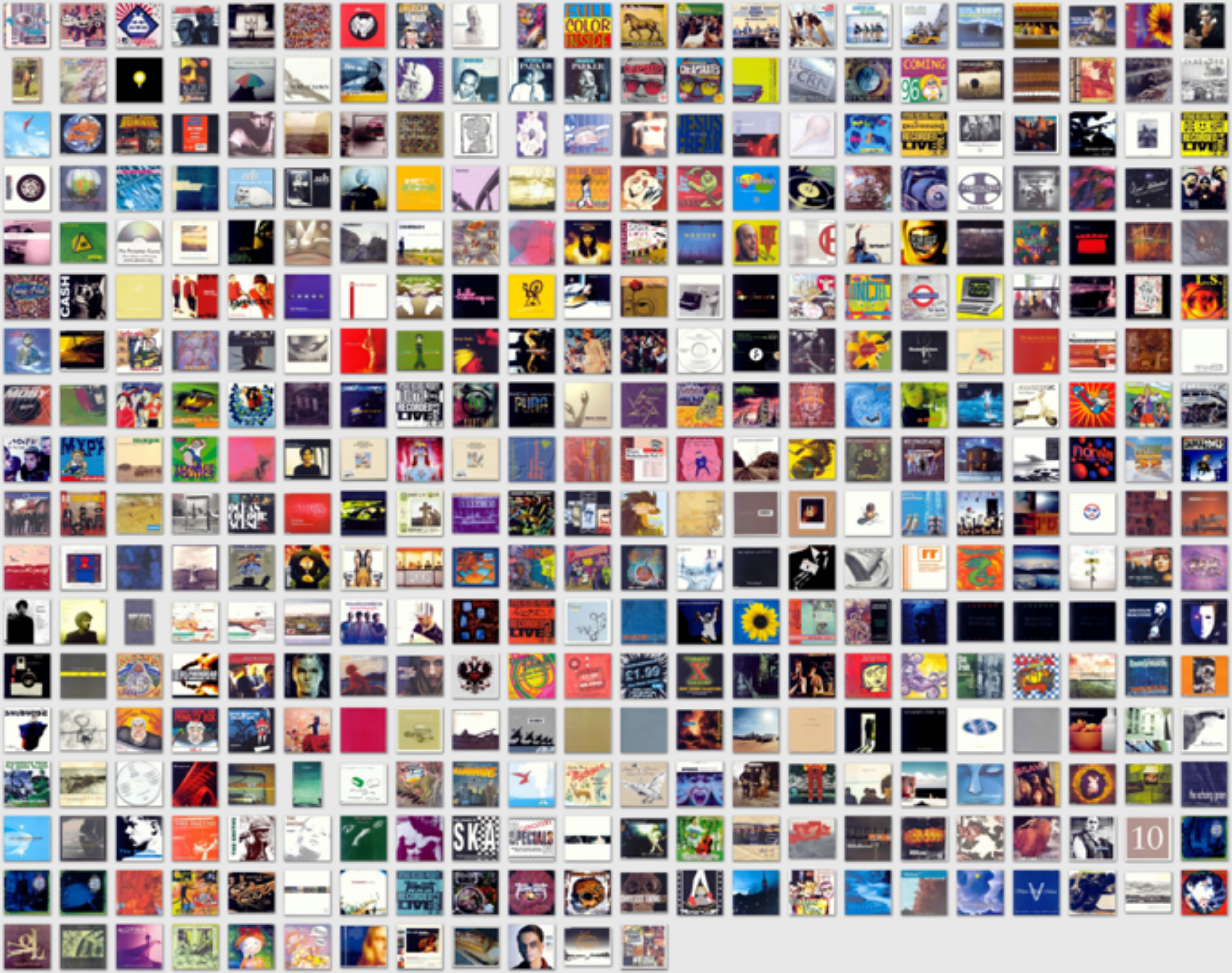
```
return argmax(P(observed|w) * P(w)
              for w in dictionary)
```


```
def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

def words(text): return re.findall('[a-z]+', text.lower())

P = train(words(file('big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in P)

def known(words): return set(w for w in words if w in P)

def correct(word):
    candidates = (known([word]) or known(edits1(word)) or
                  known_edits2(word) or [word])
    return max(candidates, key=P.get)
```

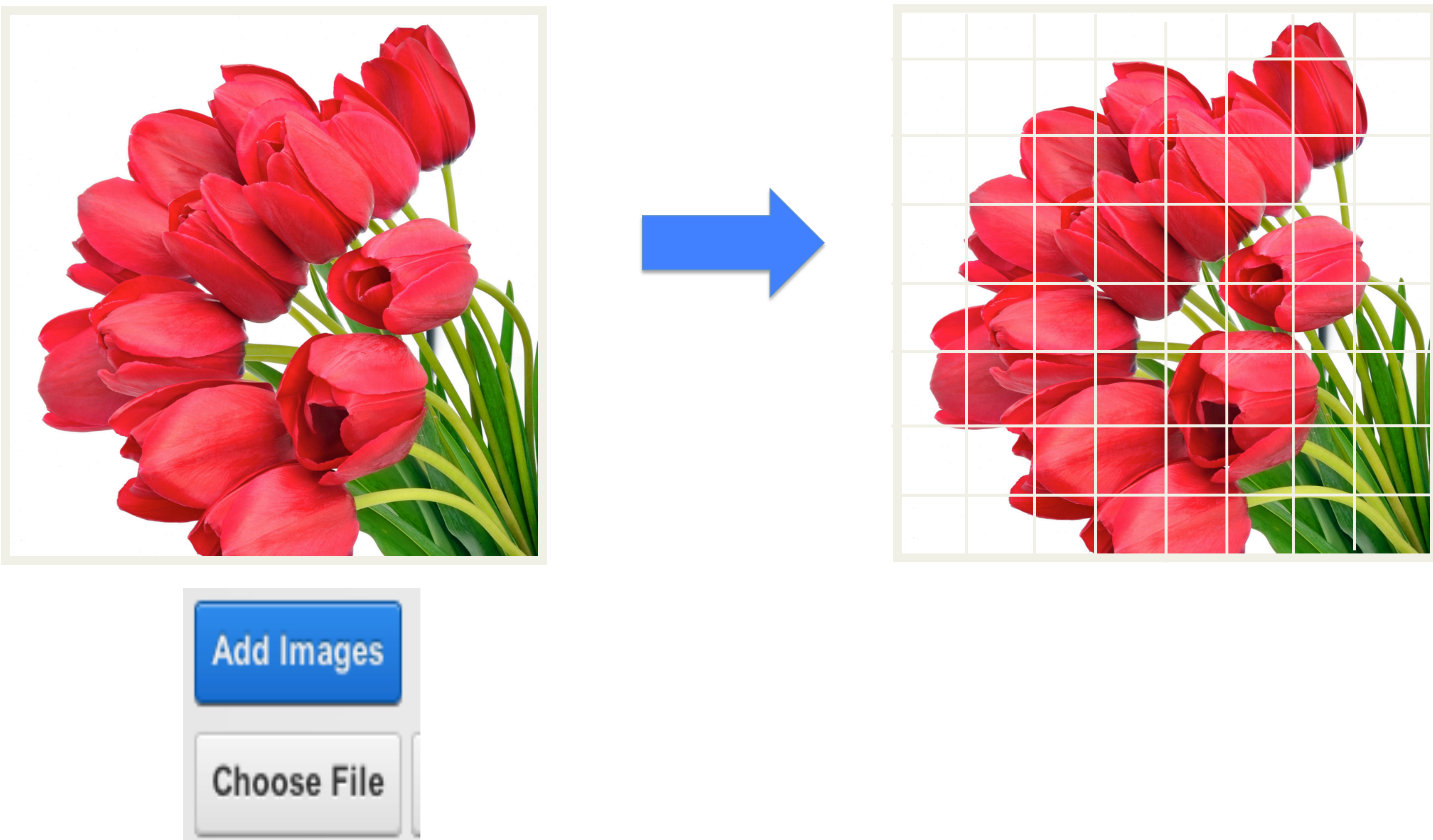
Object Recognition via Supervised Machine Learning

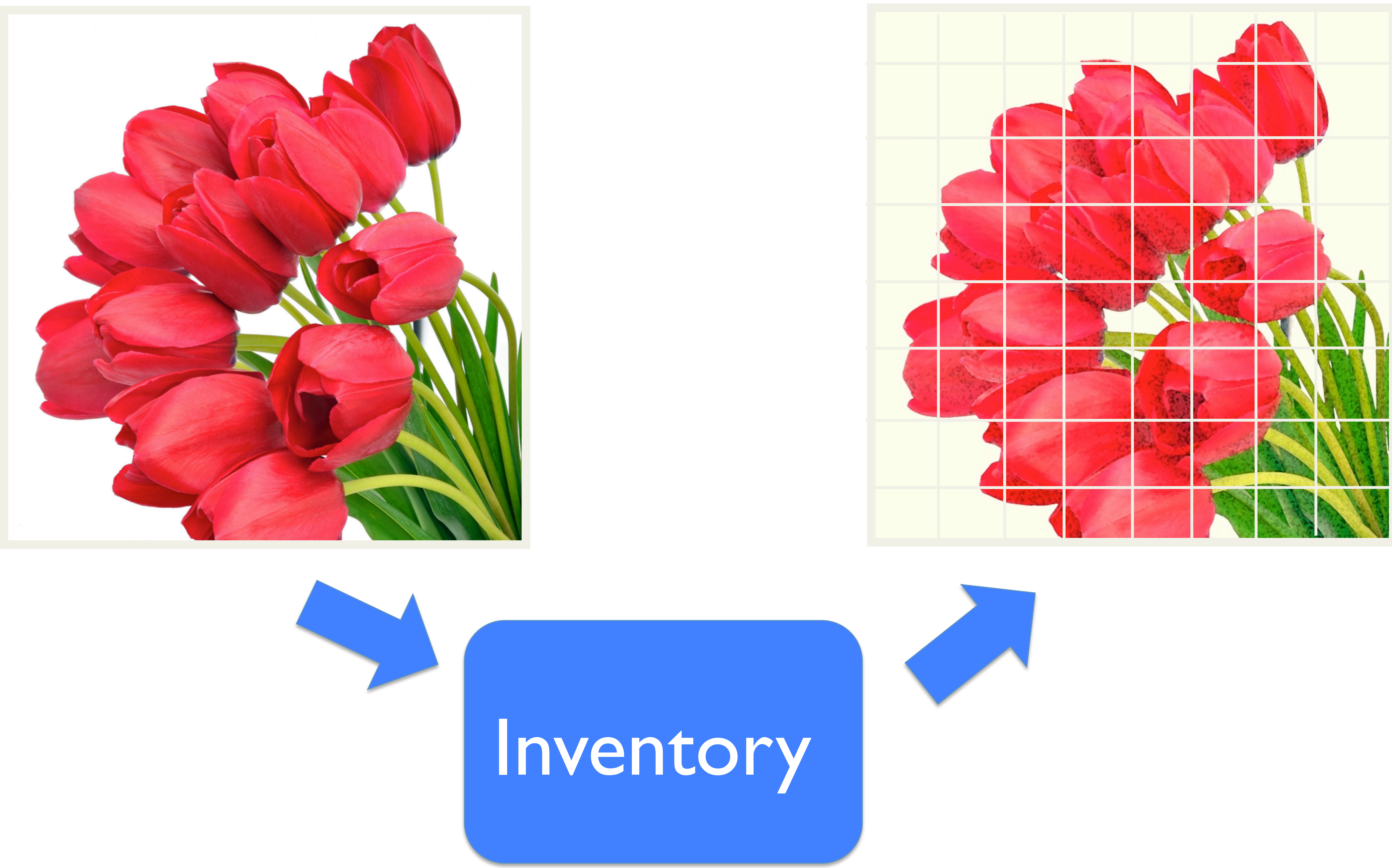

Gibbon
Panda
gibbon
panda
0.4
0.2
0
-0.2
-0.4
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2

Object Clustering via **Un**supervised Machine Learning


FULL COLOR INSIDE
PARKER
PARKER
CHEAPSKATES
CRN
COMING 96
JESUS FREAK
LIVE
CASH
L.S.
MYPX
PURA
LIVE
£1.99
X
SKA
SPECIALS
10
V
LIVE

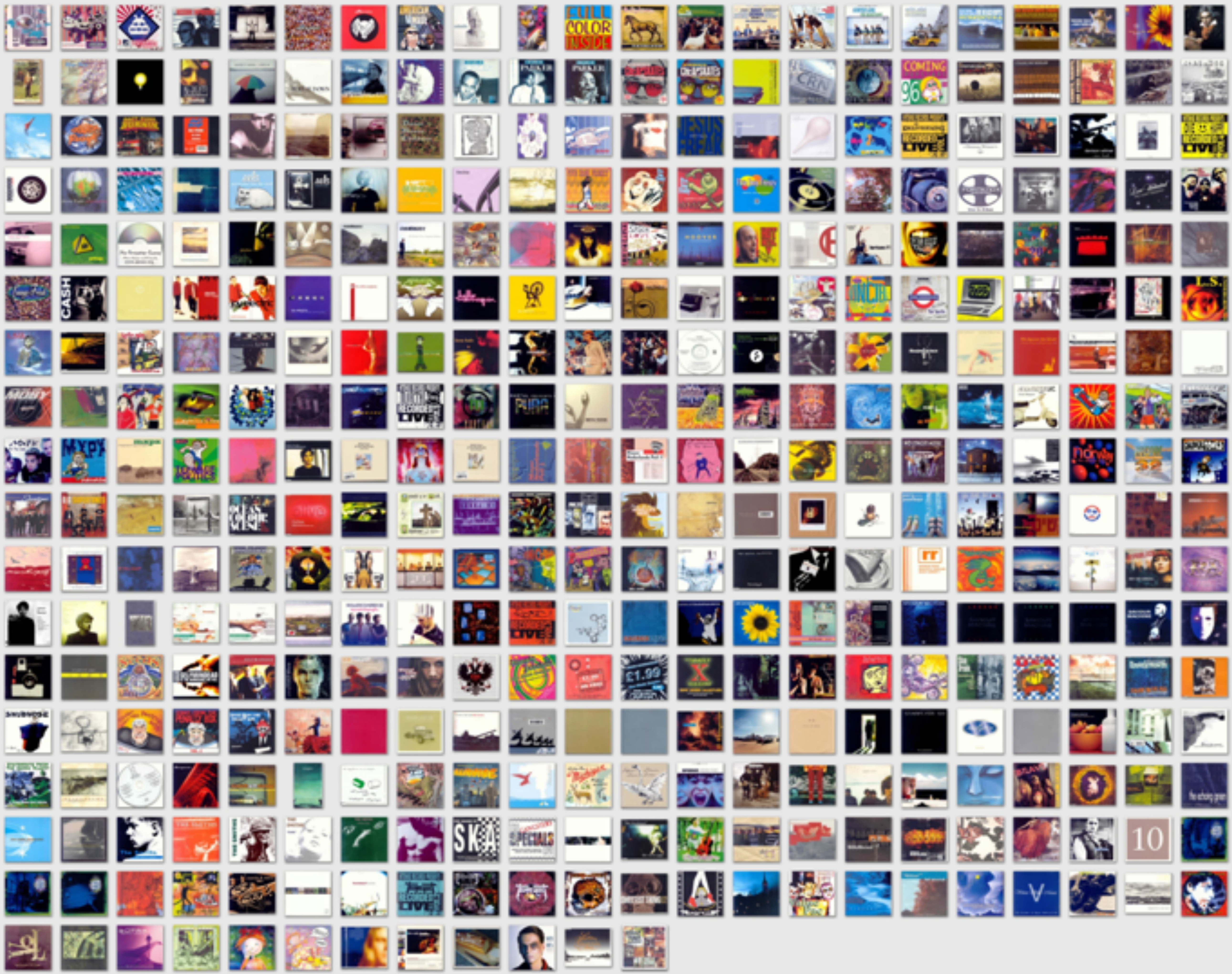
A Parable


Add Images
Choose File


Inventory

Inventory

?


FULL COLOR INSIDE
COMING
96
CASH
10

Choose a set of, say, 1000 Pieces to make near-copies of each Image minimizing difference:

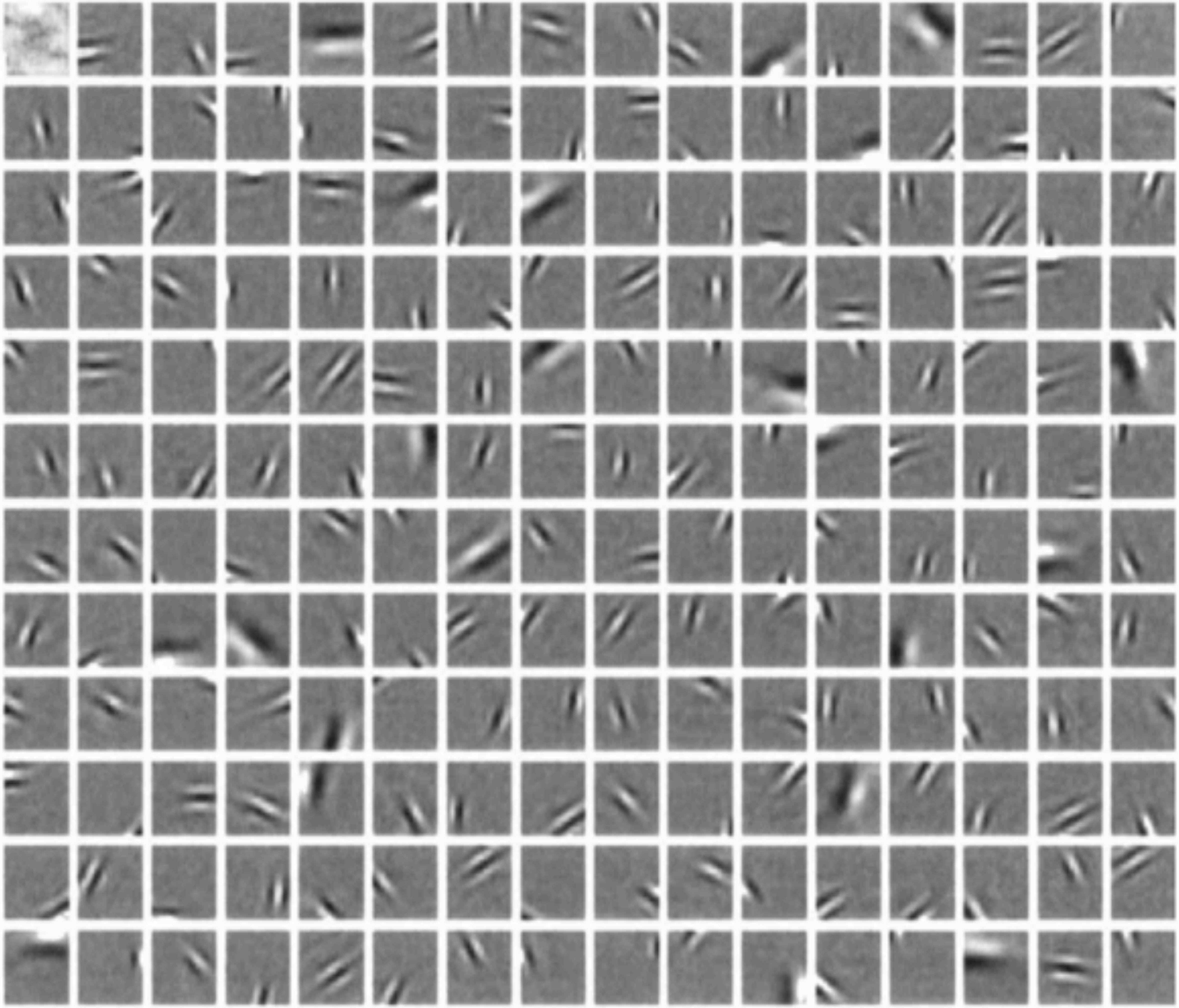
$$\Sigma\,(Copy_{x,\,y} - Image_{x,\,y})$$

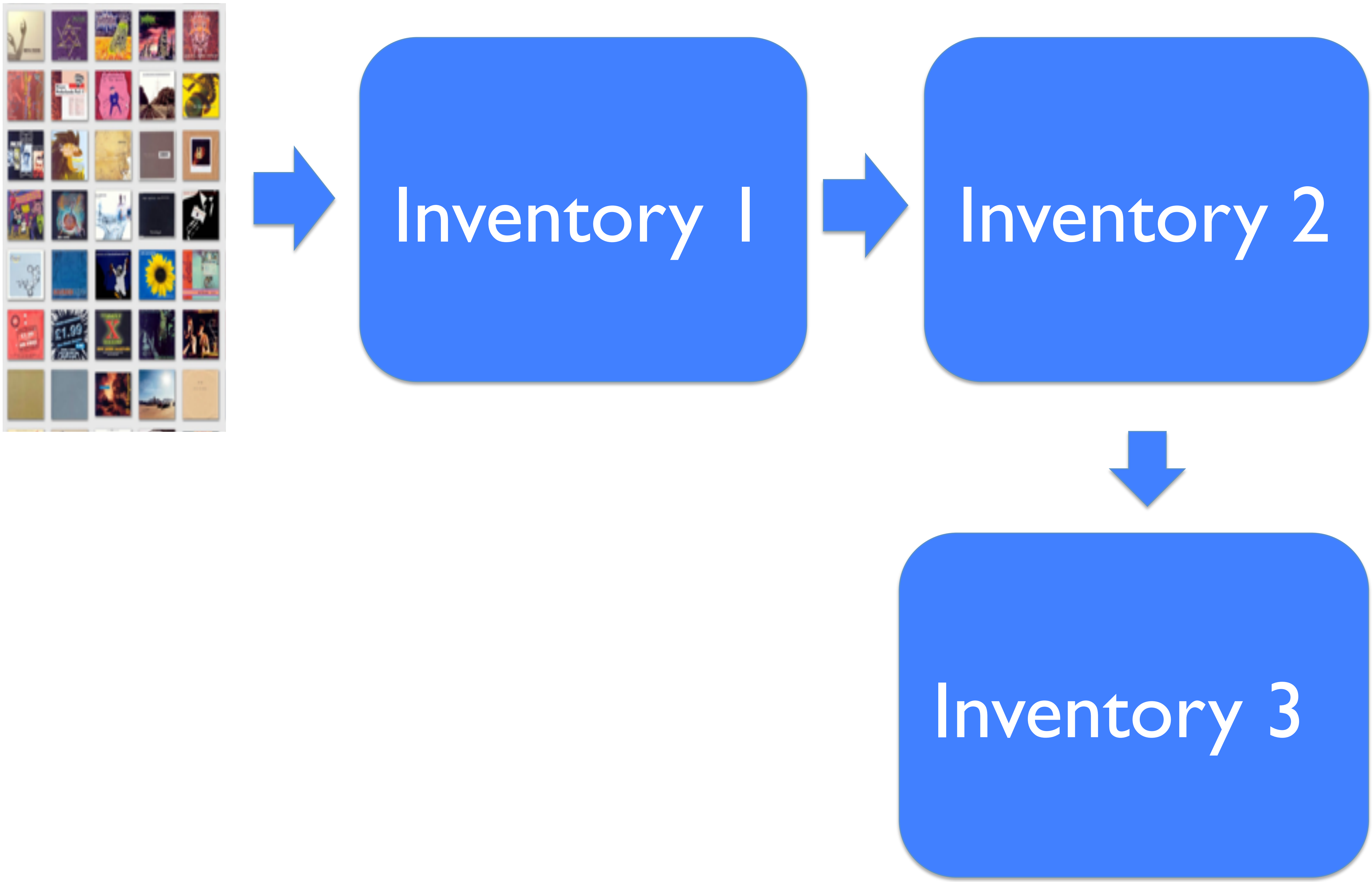
where

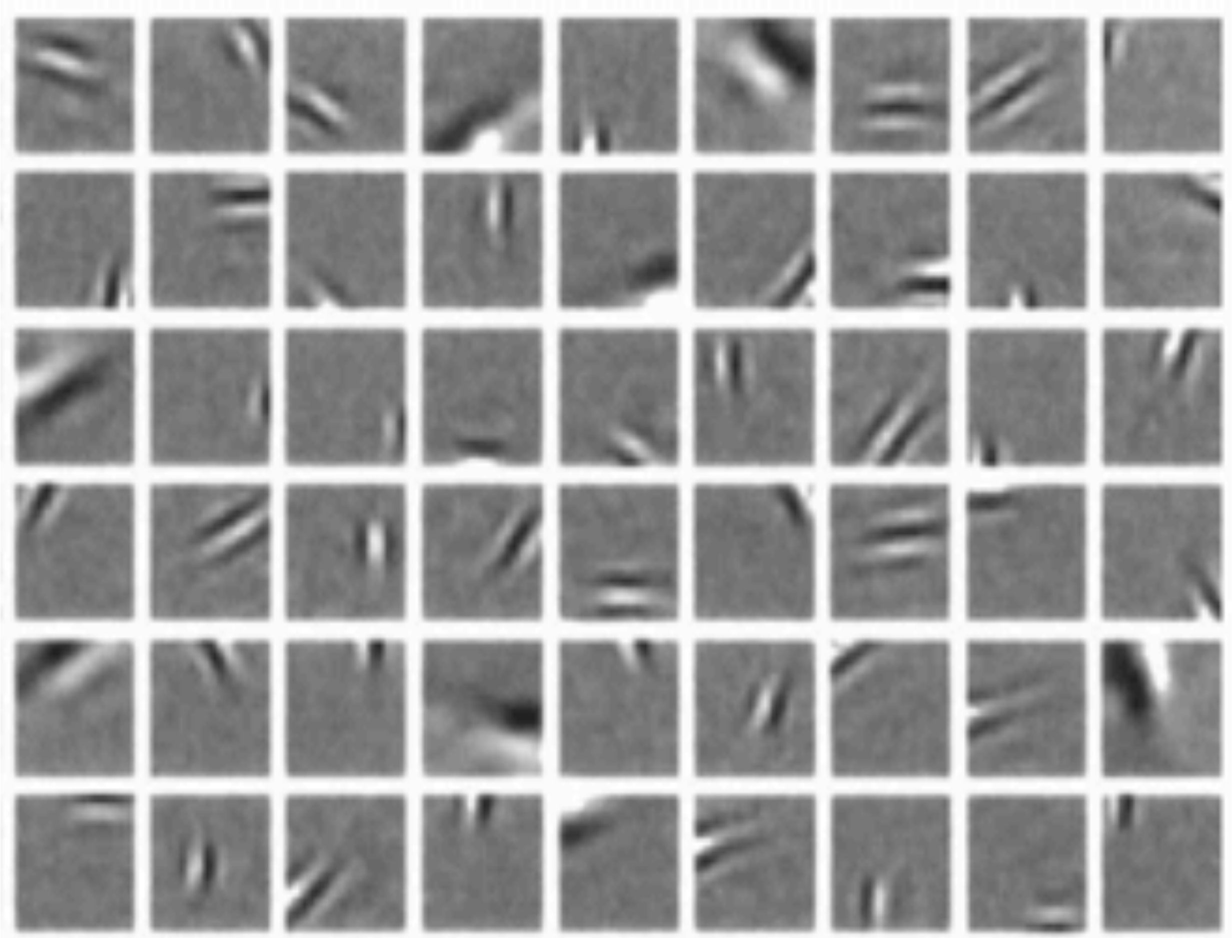
$$Copy_{x,\,y} = \Sigma\, weight_i \times Piece_i$$

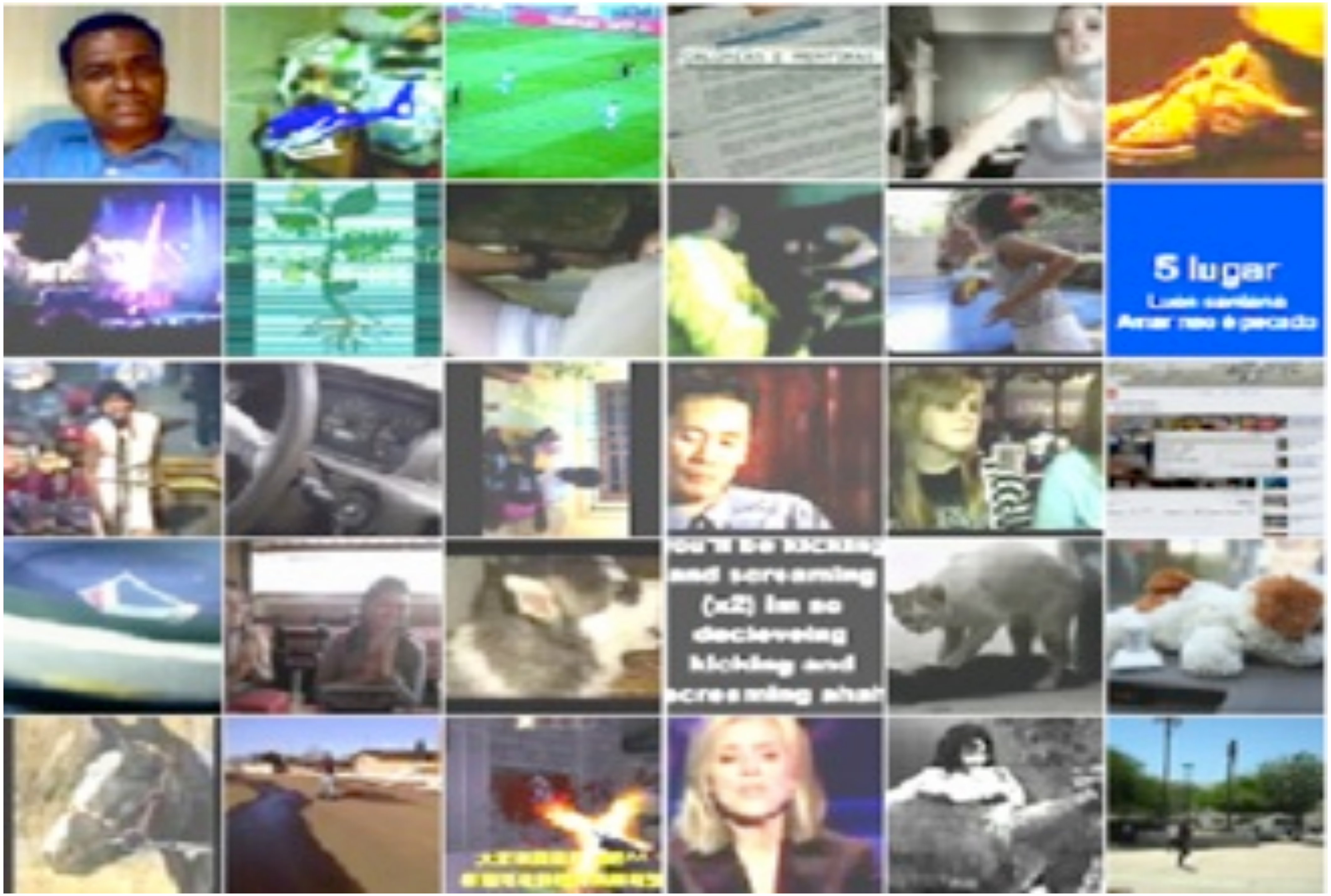
Inventory

?




Inventory 1
Inventory 2
Inventory 3





10,000,000 YouTube video frames

Synset: rust, rust fungus

Definition: any of various fungi causing rust disease in plants.

Popularity percentile:: 69%

Depth in WordNet: 6

Synset: fungus

Definition: an organism of the kingdom Fungi lacking chlorophyll and feeding on organ unicellular or multicellular organisms to spore-bearing syncytia.

Popularity percentile:: 60%

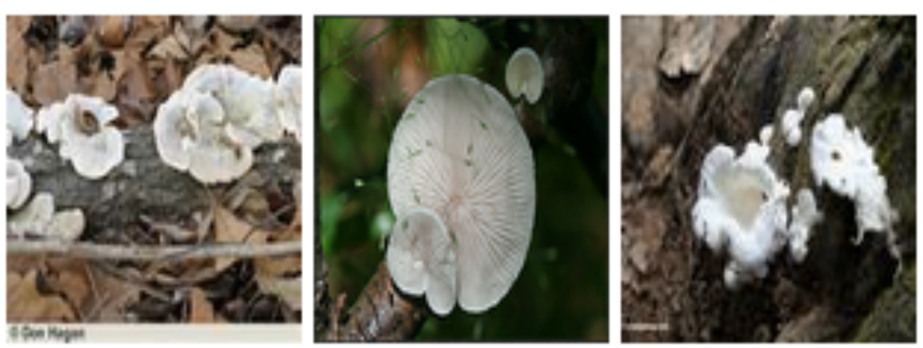
Depth in WordNet: 5

Synset: honey mushroom, honey fungus, Armillariella mellea

Definition: a honey-colored edible mushroom commonly associated with the roots of tr do not eat raw.

Popularity percentile:: 56%

Depth in WordNet: 8



Synset: white fungus, Saprolegnia ferax

Definition: a fungus that attacks living fish and tadpoles and spawn causing white fung hyphae on especially peripheral parts (as fins).

Popularity percentile:: 53%

Depth in WordNet: 6

Synset: sac fungus

Definition: any of various ascomycetous fungi in which the spores are formed in a sac

Popularity percentile:: 49%

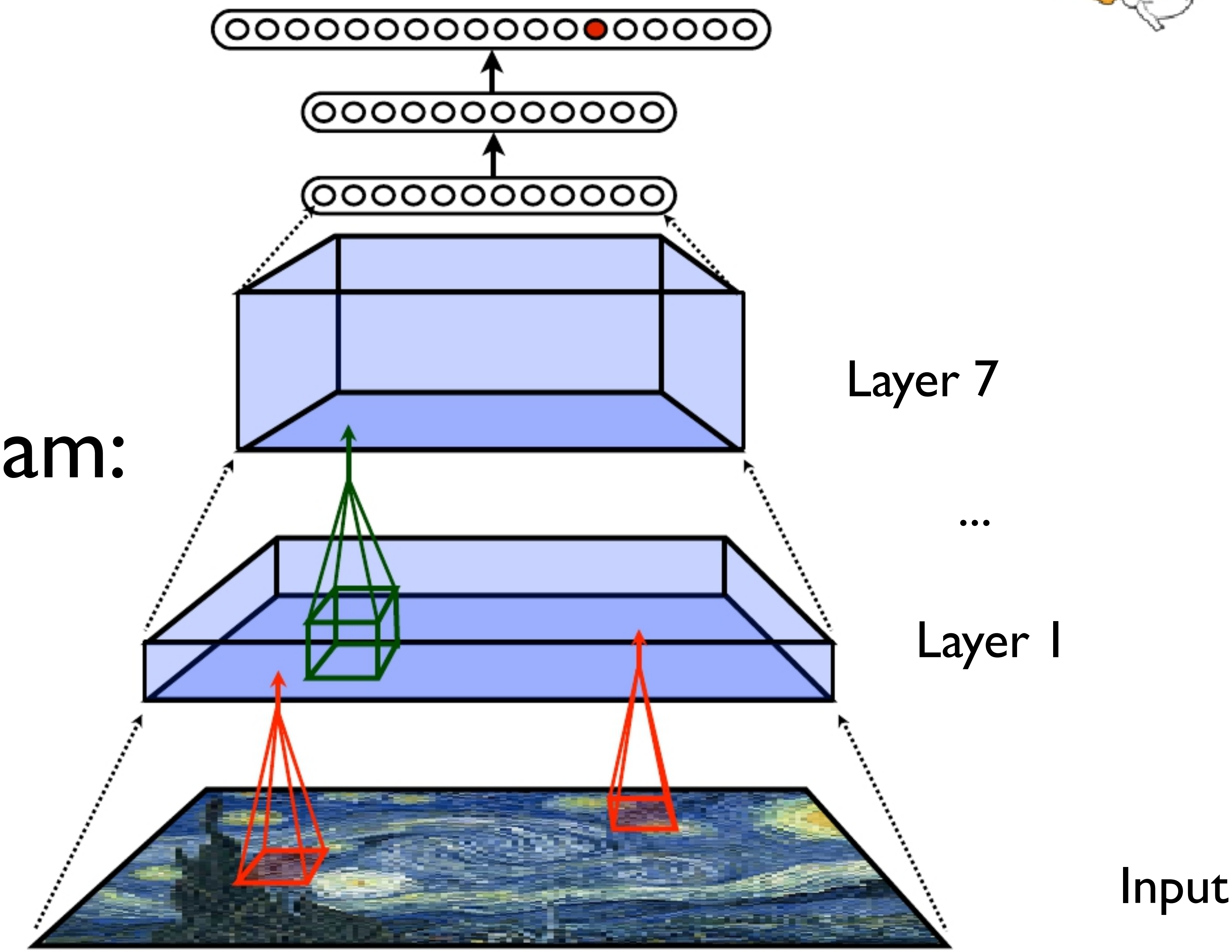
Depth in WordNet: 6

ImageNet

2012 Model

8 layers

U Toronto Team: Krizhevsky, Sutskever & Hinton



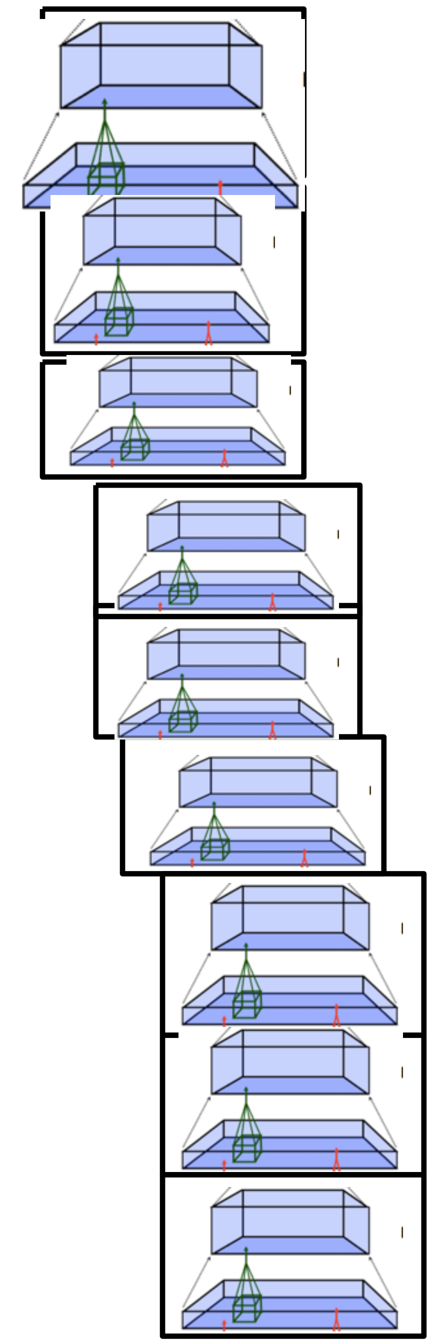
16.4% top-5 error rate

ImageNet

2014 Model

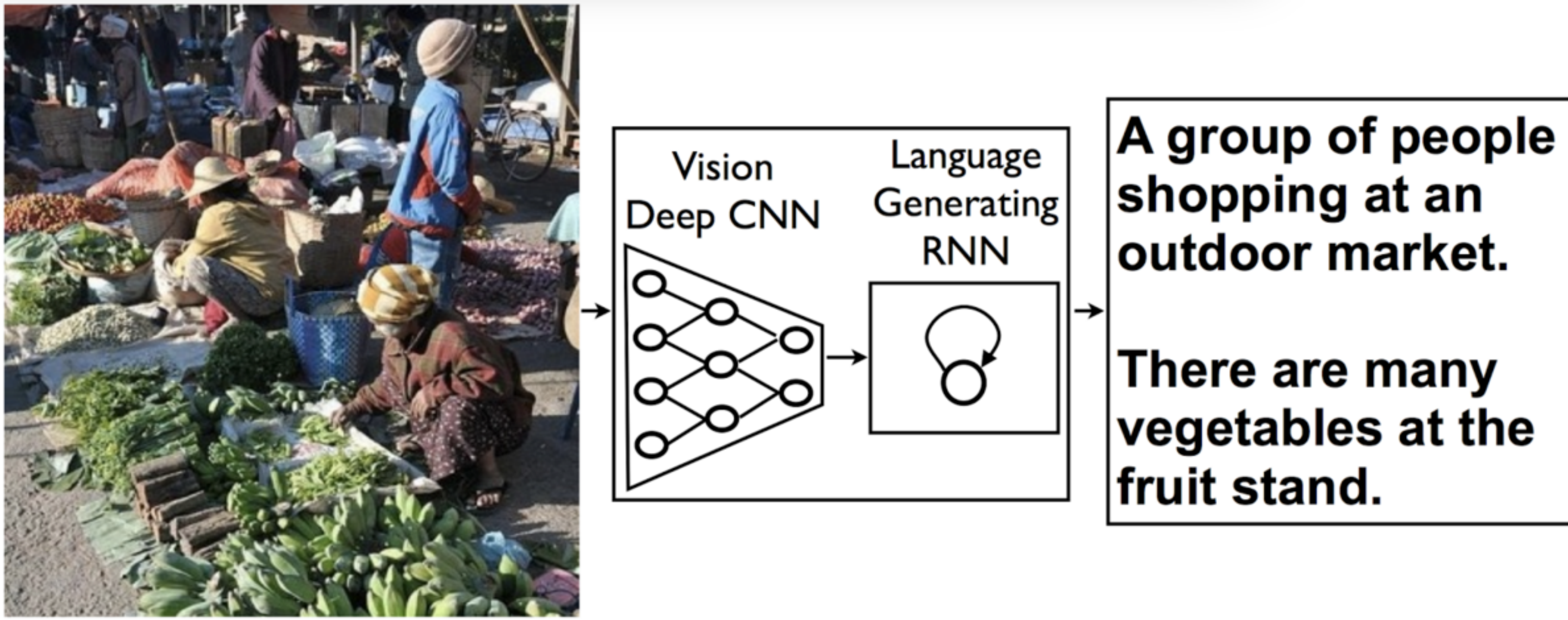
24 layers

Google Team



6.6% top-5 error rate

End-to-End Caption Writing



Human: Three different types of pizza on top of a stove.
Machine: Two pizzas sitting on top of a stove.

A couple of giraffe standing next to each other

A reflection of a dog in a side view mirror

A man riding a skateboard

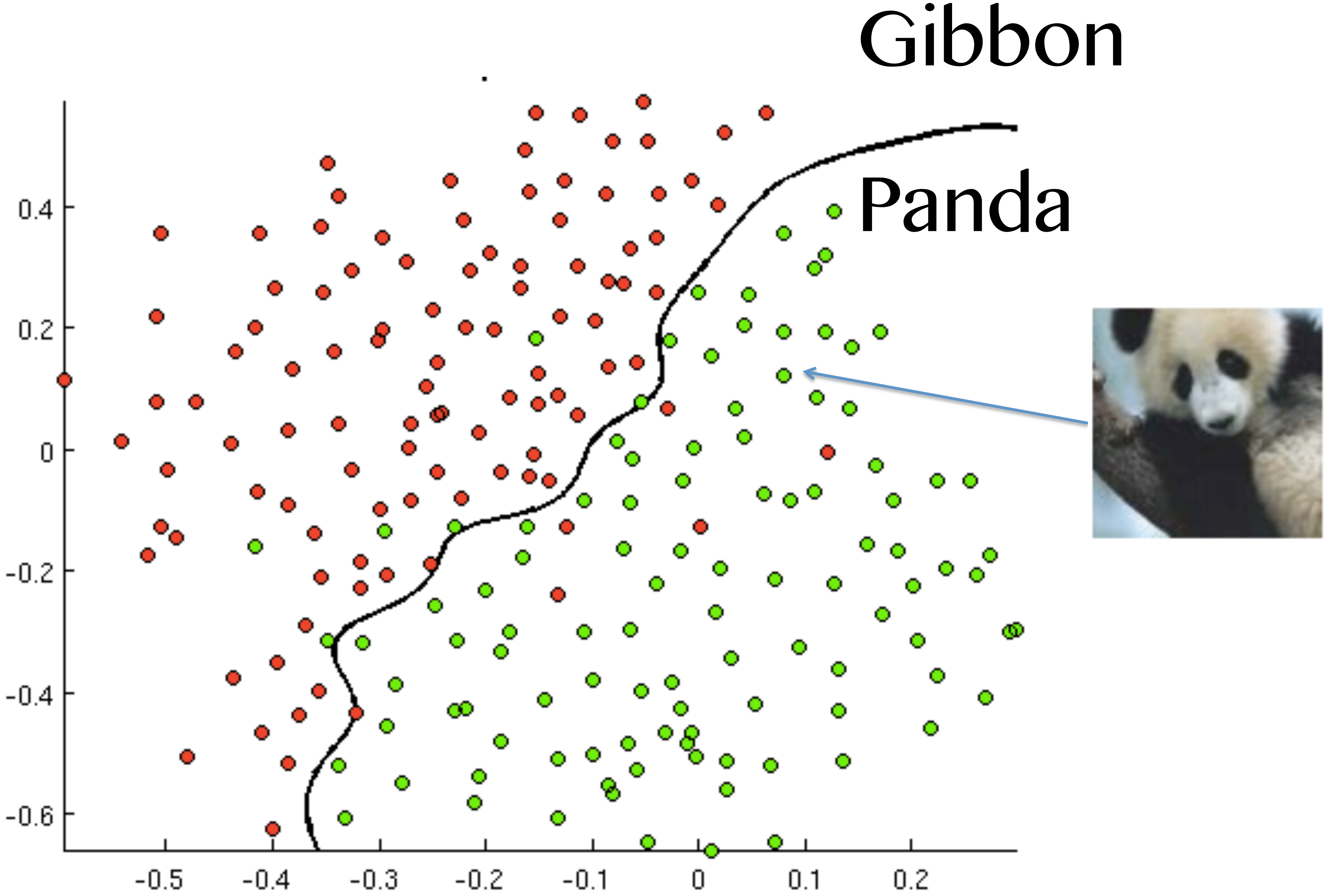
Challenges for Machine Learning Systems

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

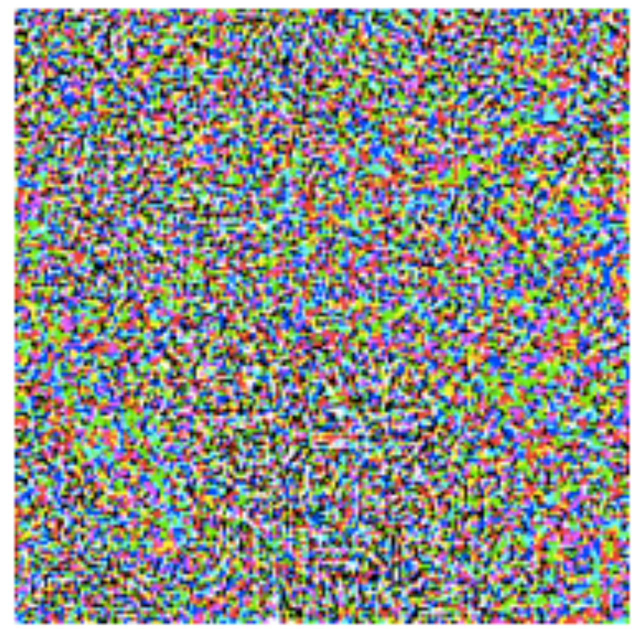
Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

ABSTRACT

Several machine learning models, including neural networks, consistently misclassify *adversarial examples*—inputs formed by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. Early attempts at explaining this phenomenon focused on nonlinearity and overfitting. We argue instead that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature. This explanation is supported by new quantitative results while giving the first explanation of the most intriguing fact about them: their generalization across architectures and training sets. Moreover, this view yields a simple and fast method of generating adversarial examples. Using this approach to provide examples for adversarial training, we reduce the test set error of a maxout network on the MNIST dataset.


Gibbon
Panda
0.4
0.2
0
-0.2
-0.4
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2

$+ .007 \times$



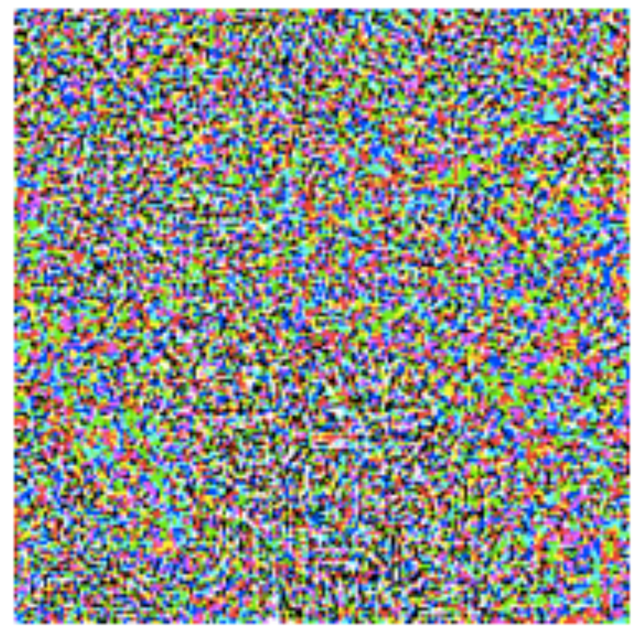
$\boldsymbol{x}$

"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$+ .007 \times$



$=$

$\boldsymbol{x}$

"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$\boldsymbol{x} +$
$\epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

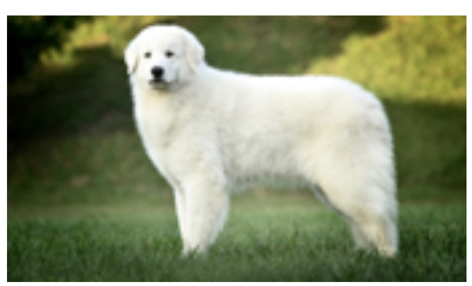
"gibbon"
99.3 % confidence

Chimp

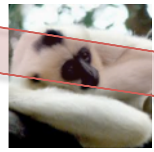
Gibbon

Panda

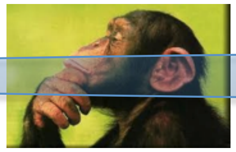
Kuvasz Dog



Sifaka Lemur

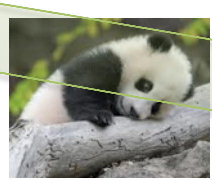
 Gibbon

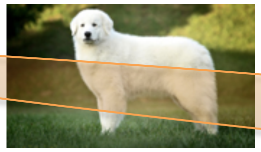
?

 Chimp

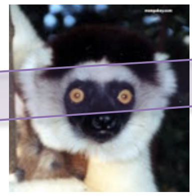
?

?

 Panda

 Kuvasz Dog

?

 Sifaka Lemur

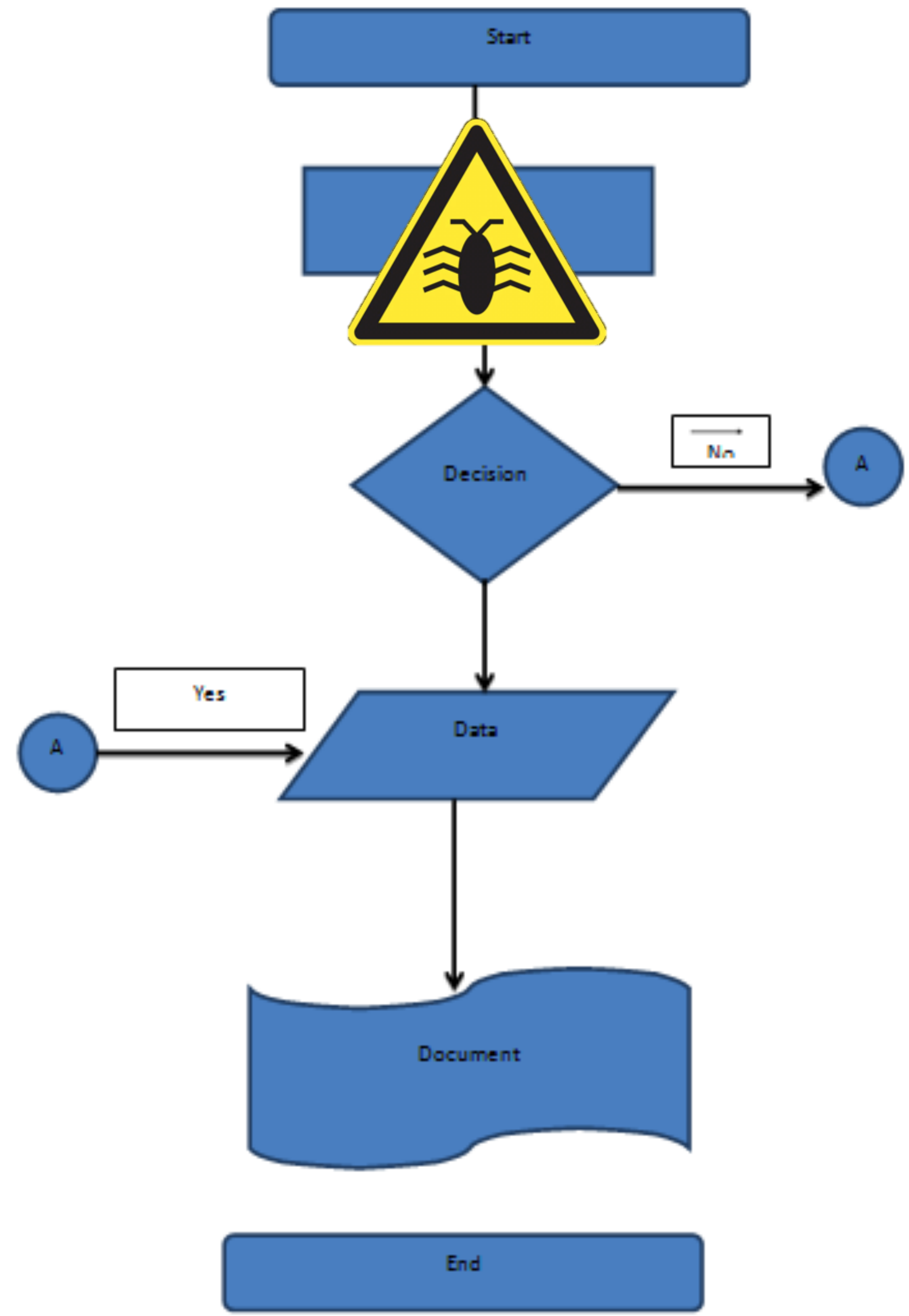
Machine Learning:
The High-Interest Credit Card of Technical Debt

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov,
Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young
{dsculley,gholt,dgg,edavydov}@google.com
{toddphillips,ebner,vchaudhary,mwyoung}@google.com
Google, Inc

Abstract

Machine learning offers a fantastically powerful toolkit for building complex systems quickly. This paper argues that it is dangerous to think of these quick wins as coming for free. Using the framework of *technical debt*, we note that it is remarkably easy to incur massive ongoing maintenance costs at the system level when applying machine learning. The goal of this paper is highlight several machine learning specific risk factors and design patterns to be avoided or refactored where possible. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, changes in the external world, and a variety of system-level anti-patterns.

Lack of Clear Abstraction Barriers



Learning to Divide and Conquer: Applying the L* Algorithm to Automate Assume-Guarantee Reasoning

Corina S. Păsăreanu

Perot Systems, NASA Ames Research Center, N269-230, Moffett Field, CA 94035, USA

Dimitra Giannakopoulou

RIACS, NASA Ames Research Center, N269-230, Moffett Field, CA 94035, USA

Mihaela Gheorghiu Bobaru

Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, Ontario, CANADA M5S 3G4

Jamieson M. Cobleigh [1]

Department of Computer Science, University of Massachusetts, 140 Governor's Drive, Amherst, MA 01003, USA

Howard Barringer

School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK

Lifecycle Verification of the NASA Ames K9 Rover Executive

Dimitra Giannakopoulou[1, 3] **Corina S. Pasareanu**[2, 3] **Michael Lowry**[3] and **Rich Washington**[4]

(1) USRA/RIACS
(2) Kestrel Technology LLC
(3) NASA Ames Research Center, Moffett Field, CA 94035-1000, USA
{dimitra, pcorina, lowry}@email.arc.nasa.gov
(4) Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
rwashington@google.com

Abstract

Autonomy software enables complex, robust behavior in reaction to external stimuli without human intervention. It is typically based on planning and execution technology. Extensive verification is a pre-requisite for autonomy technology to be adopted in high-risk domains. This verification is challenging precisely because of the multitude of behaviors enabled by autonomy technology.

This paper describes the application of advanced verification techniques for the analysis of the Executive subsystem of the NASA Ames K9 Rover. Existing verification tools were extended in order to handle a system the size of the Executive. A divide and conquer approach was critical for scaling. Moreover, verification was performed in close collaboration with the system developers, and was applied during both design and implementation. Our study demonstrates that advanced verification techniques are crucial for real-world planning and execution systems. Moreover, it shows that when verification proceeds hand-in-hand with software development throughout the lifecycle, it can greatly improve the design decisions and the quality of the resulting plan execution system.

and effort since they may involve major changes in the architecture of the system, and possible re-implementation of a large part of it. Therefore, we believe that the verification of a safety critical system should be addressed *as early as possible during its design*, and should go hand-in-hand with later phases of software development.

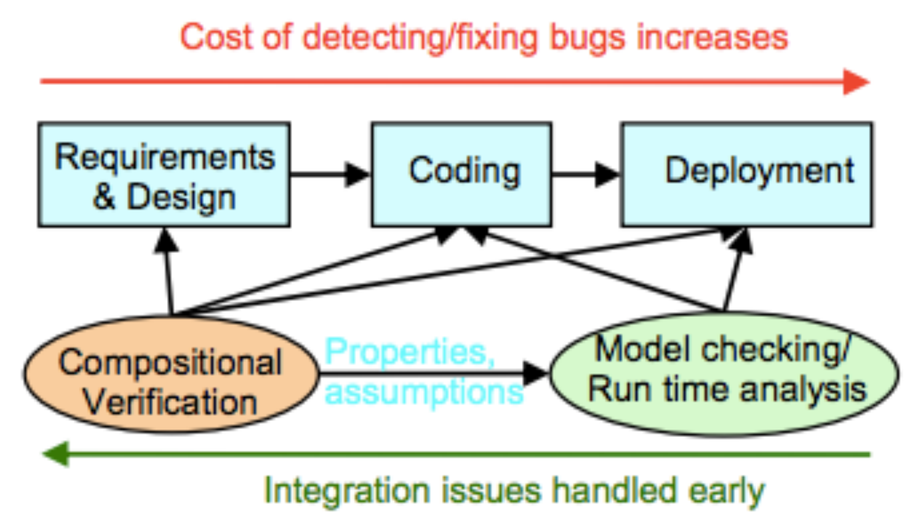
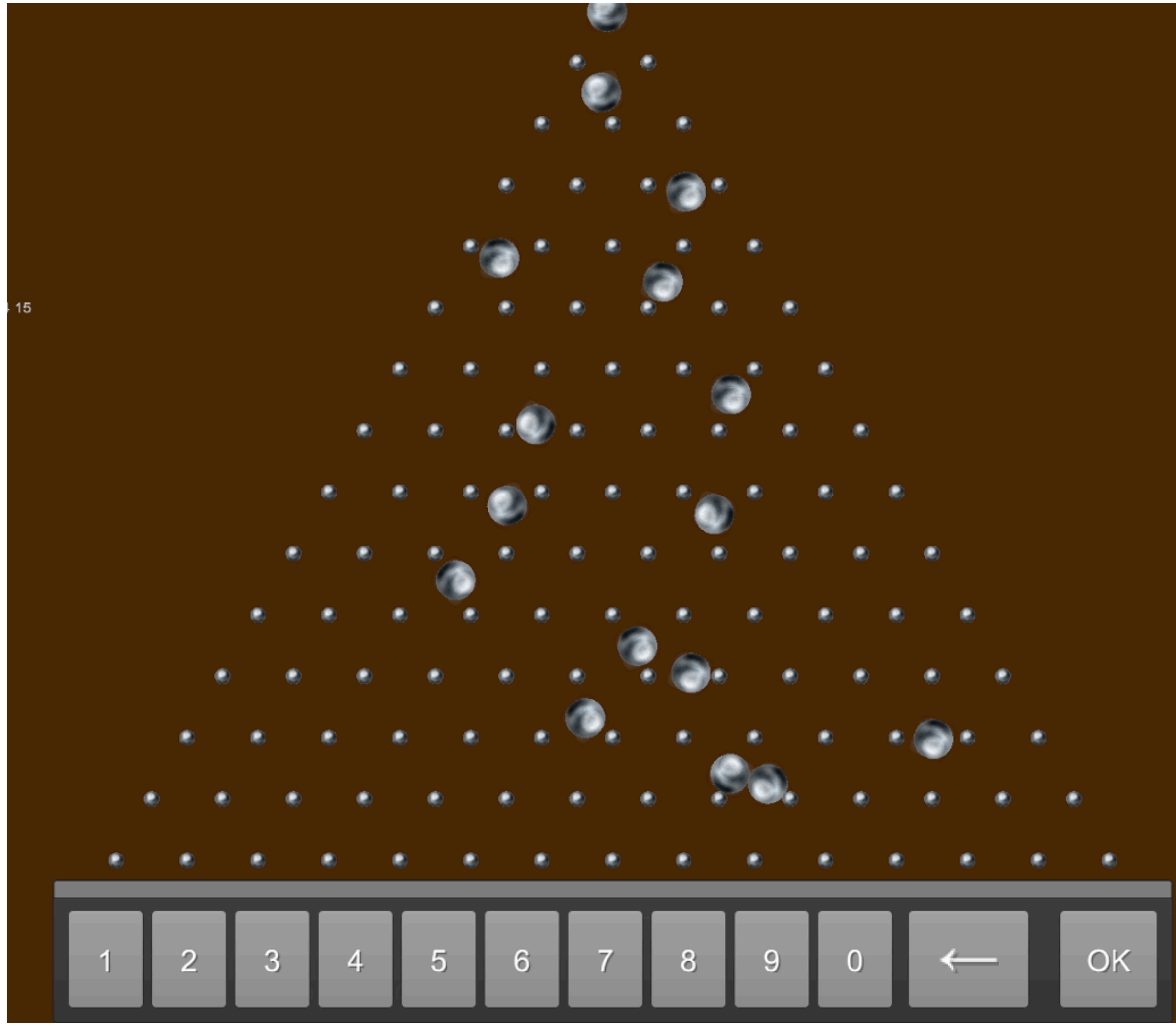


Figure 1. Compositional verification throughout the software lifecycle

Our work advocates the use of a combination of formal analysis techniques and testing to analyze autonomous

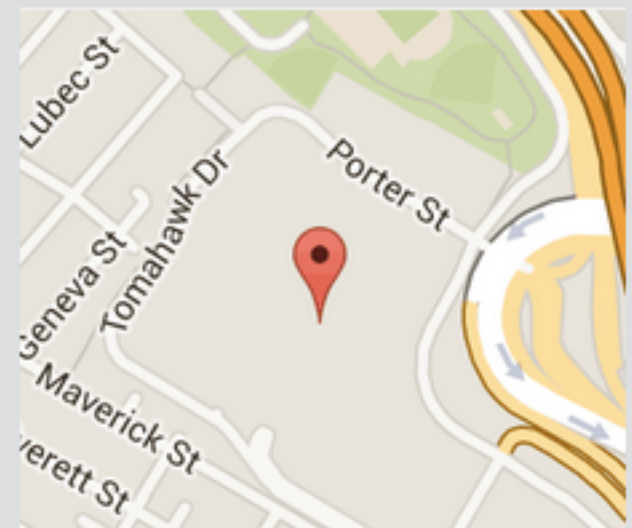
Non-Modularity: Changing Anything Changes Everything



Google Now

Depart now for:

Return Rental Car to Logan Airport
156 Tomahawk Dr,
Boston MA

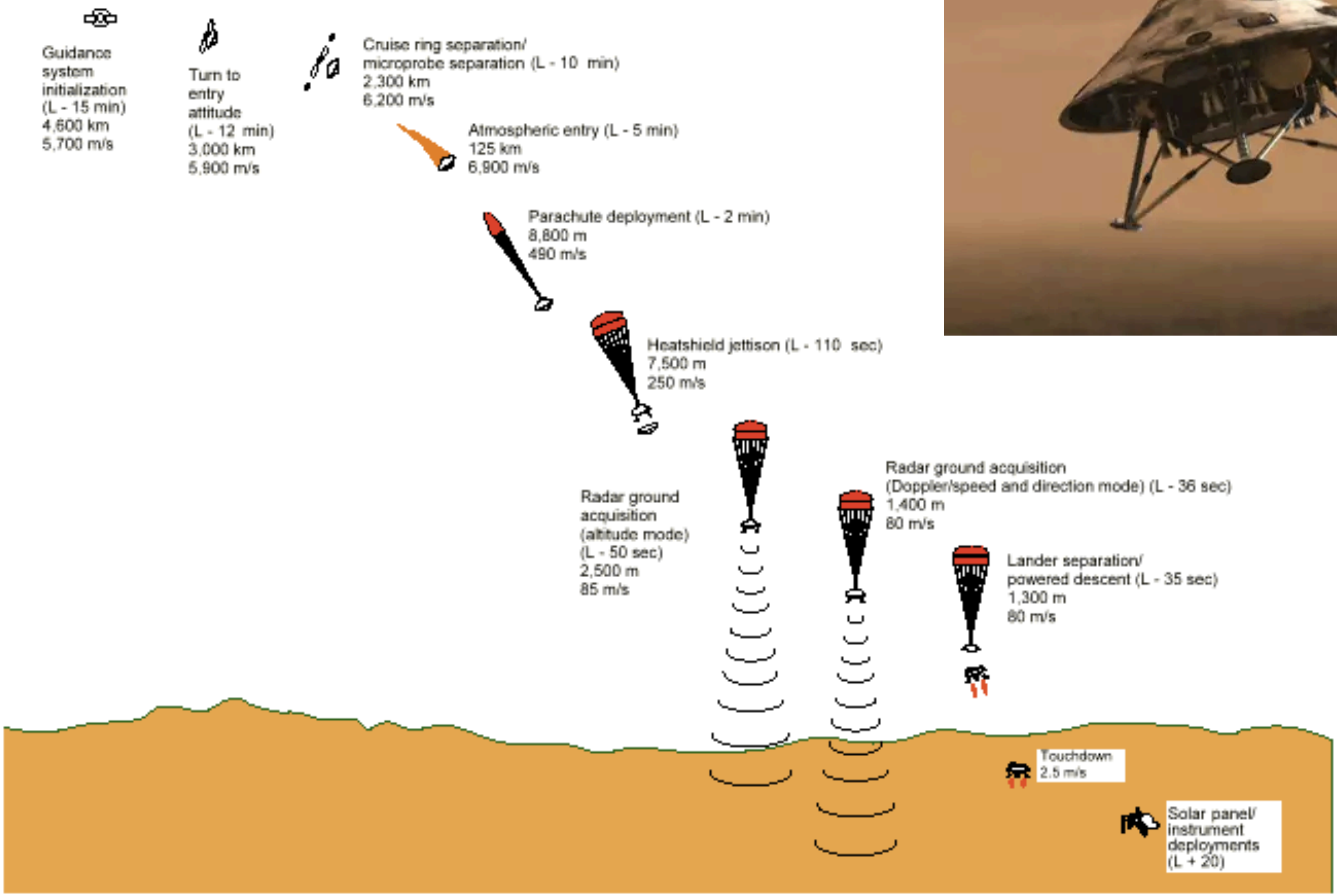


Time of travel:

23 minutes by bicycle

```
event = ExtractEvent(email.body)
trip  = Travel(current.location, event.location, event.time)
CreateAlert(trip)
```

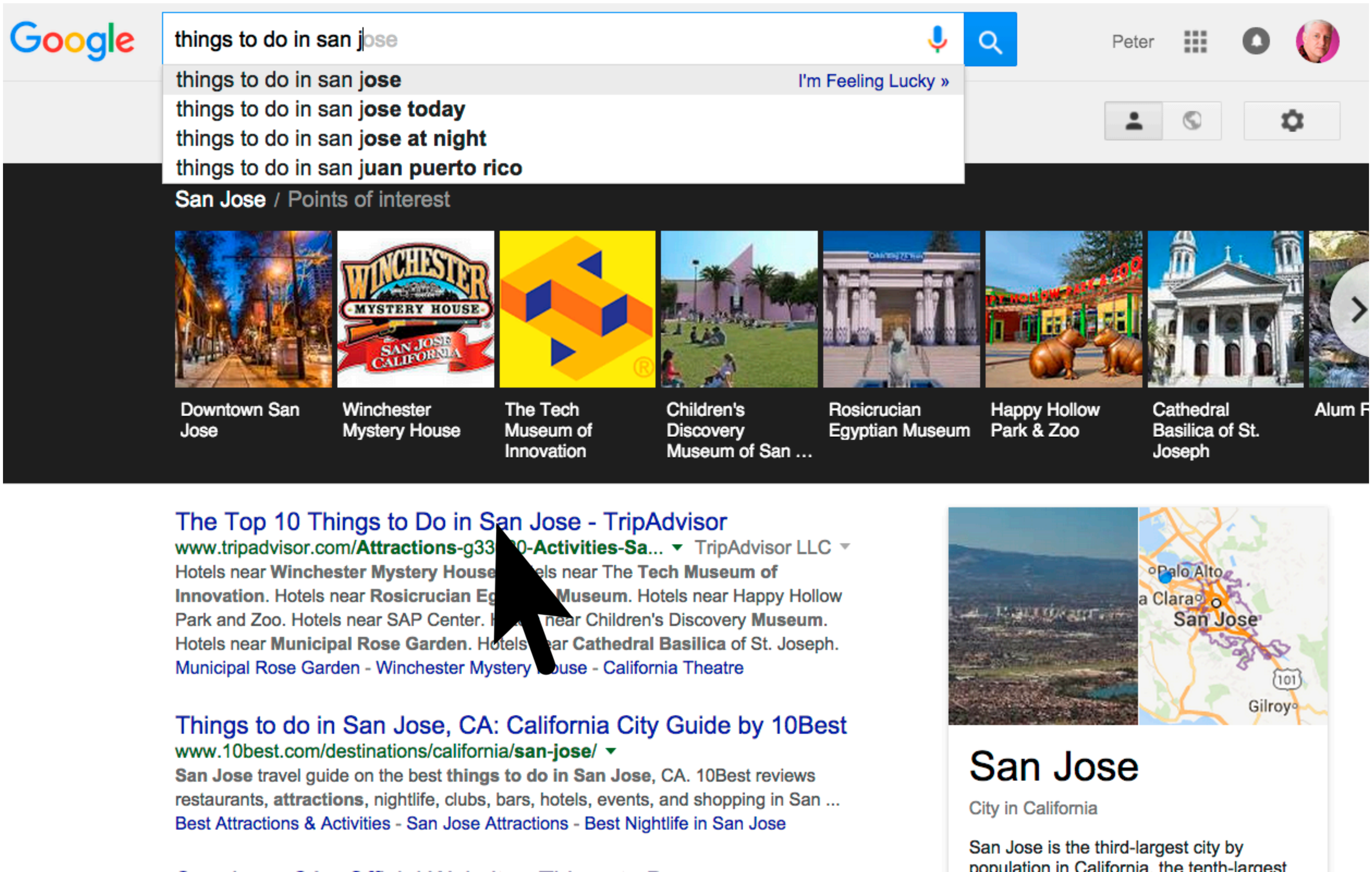
Mars Polar Lander

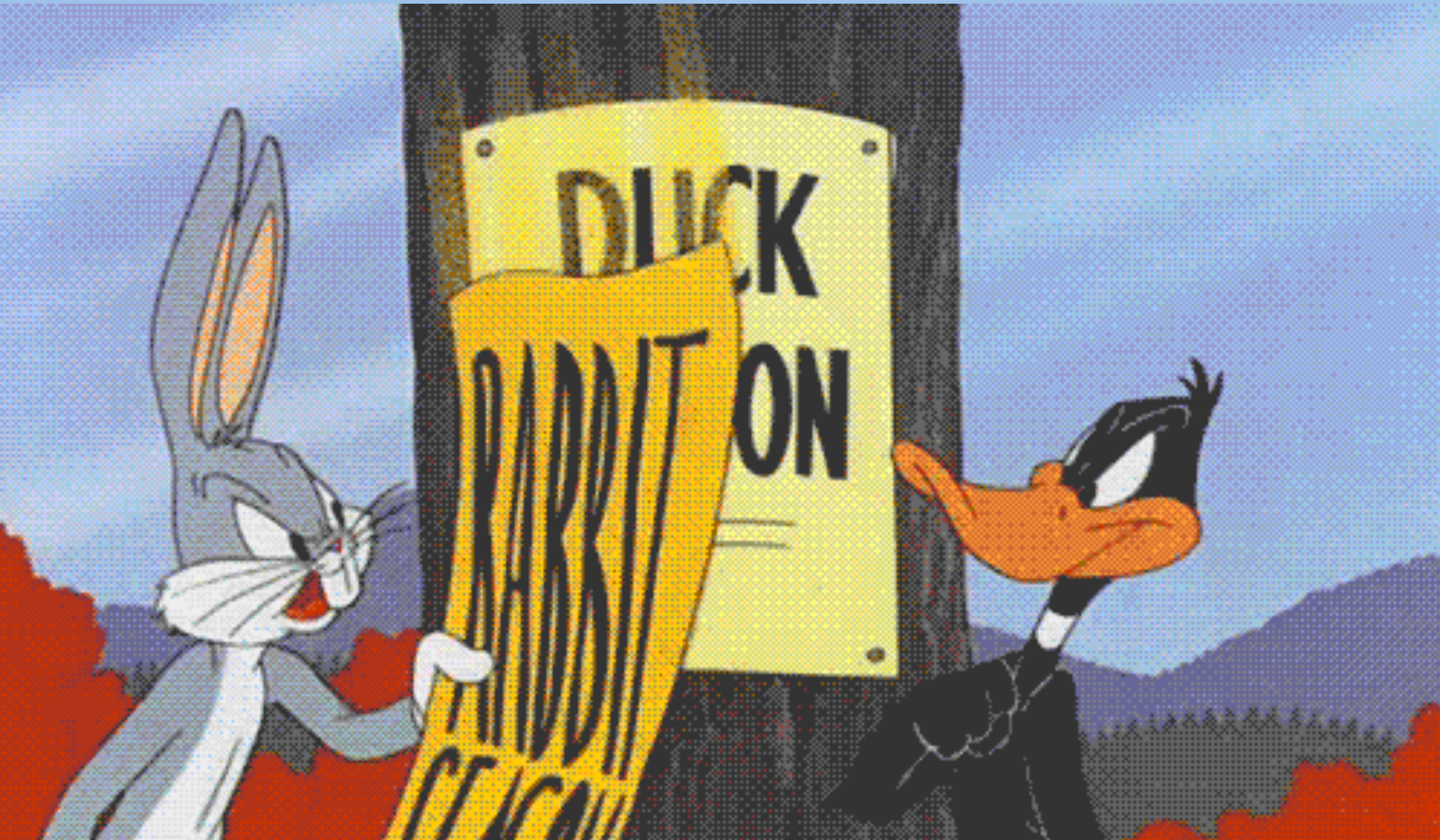


Entry, descent and landing

Nonstationarity

Feedback Loops




DUCK
ON
RABBIT

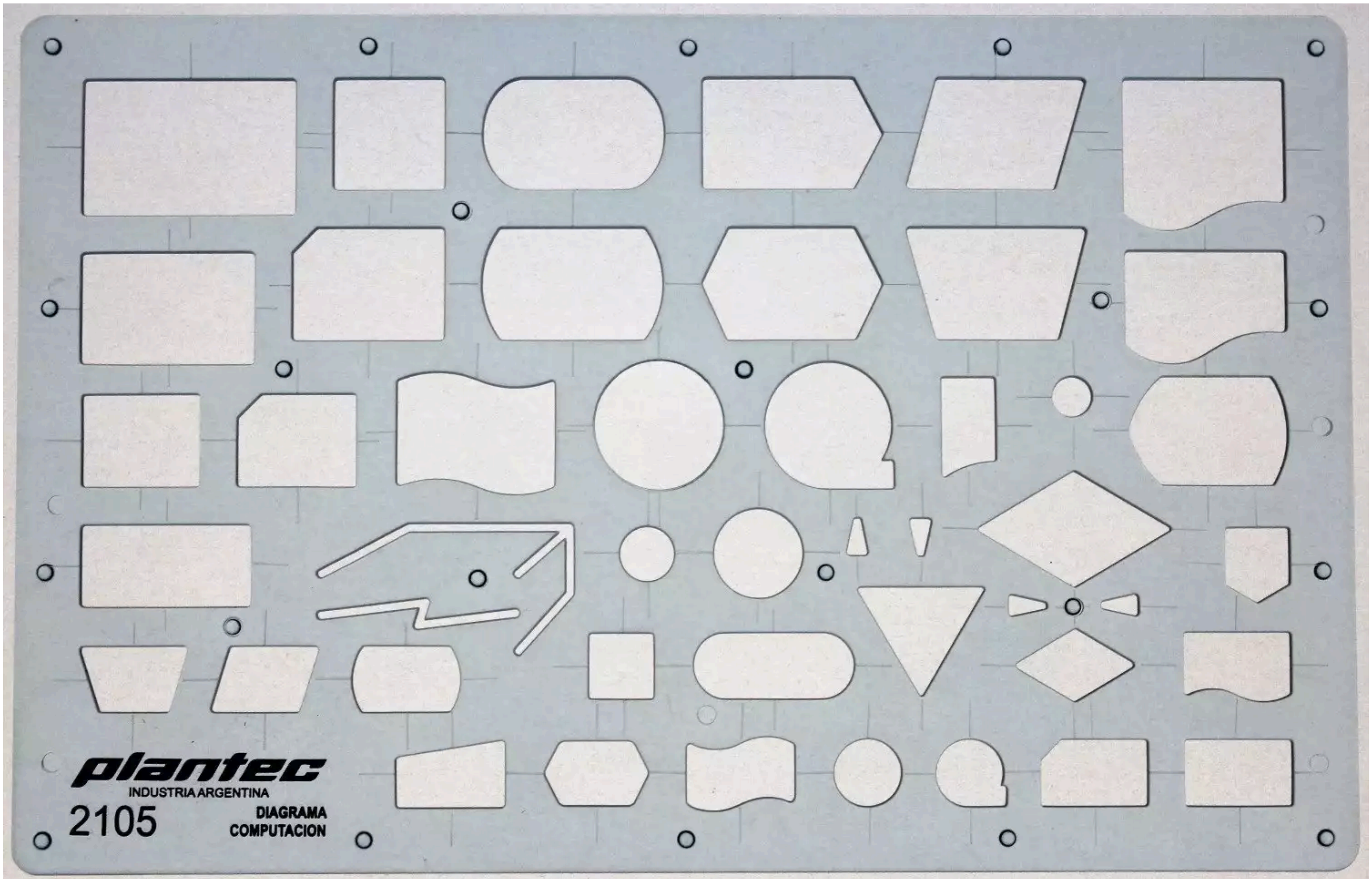
Attractive Nuisance

“Synonyms”

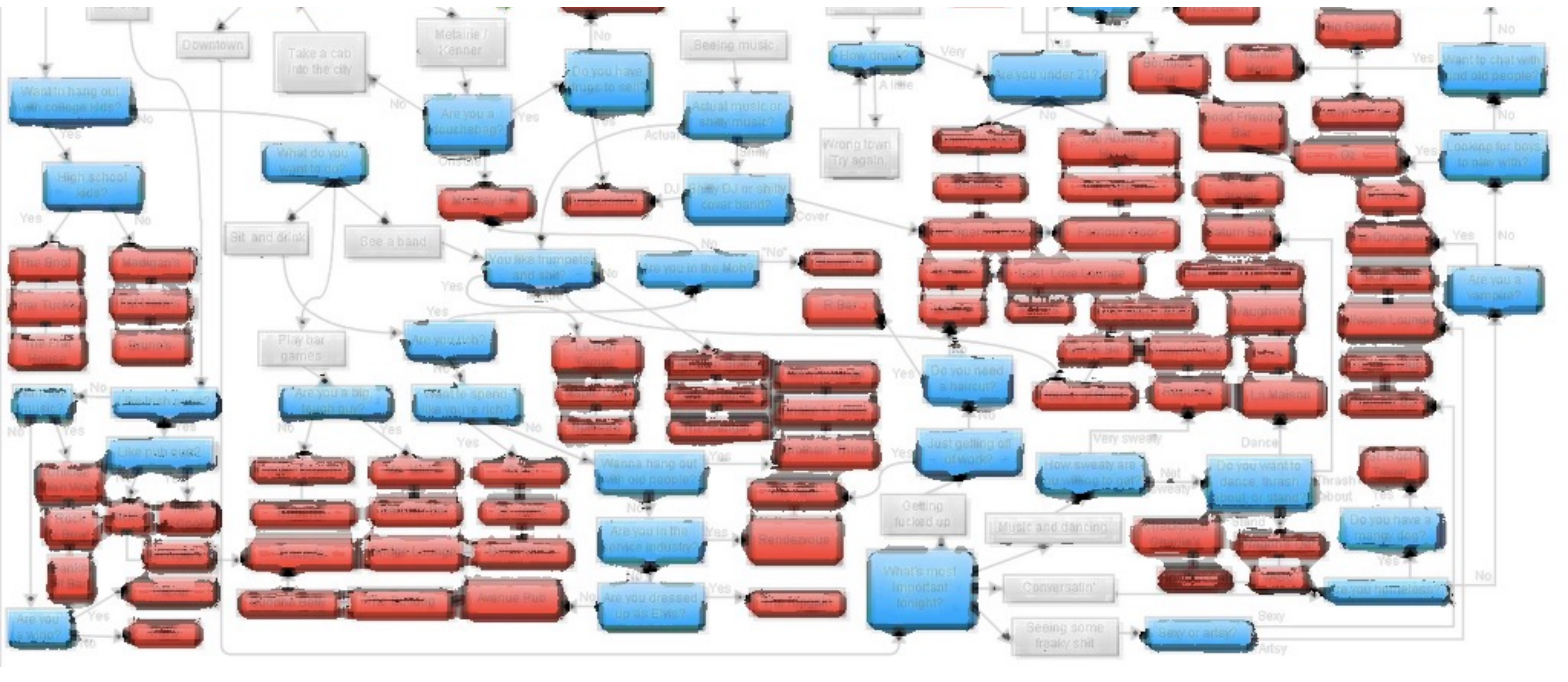
Privacy and Security

Lack of Intuition

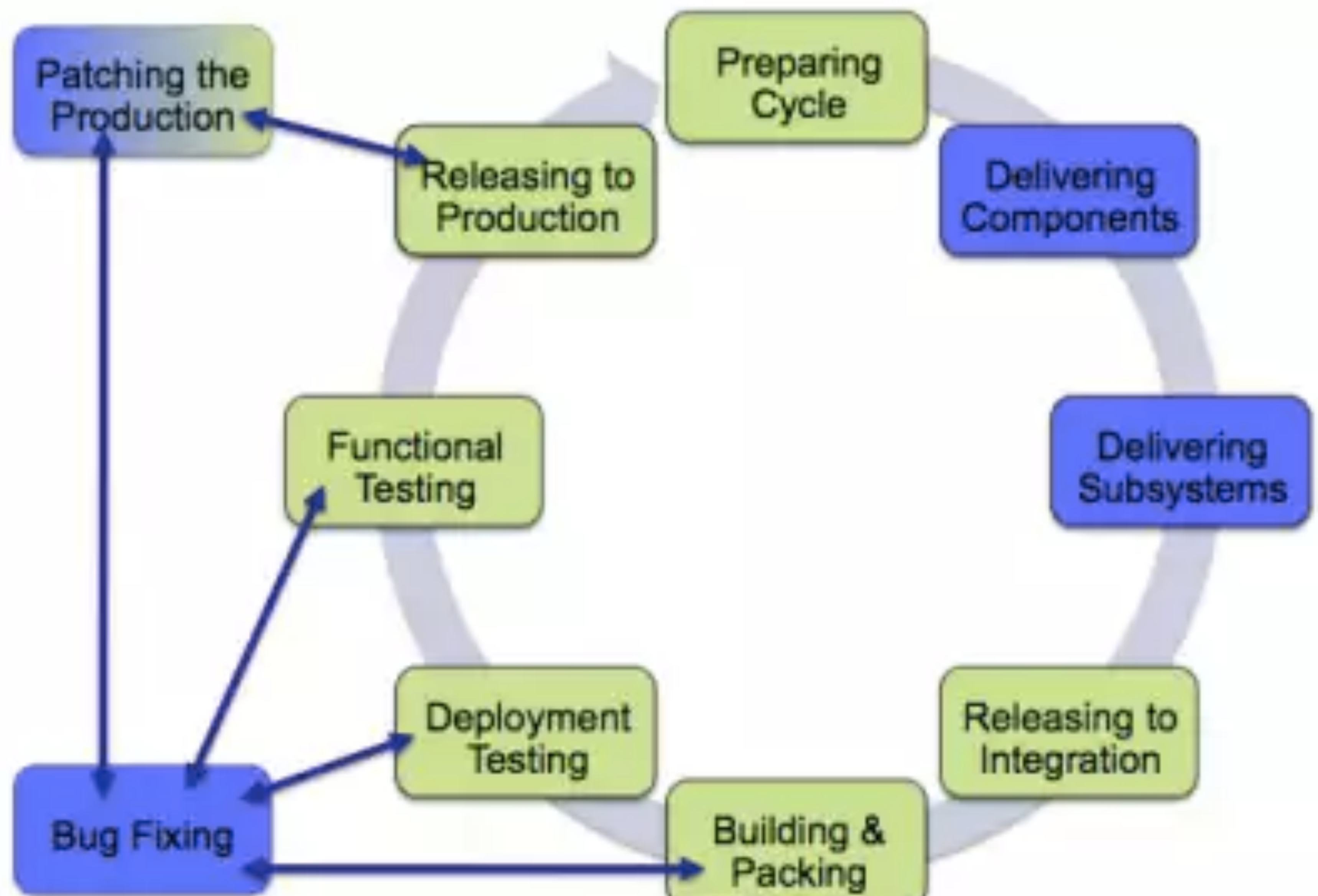
Lack of Tooling



Data/Config Dependencies



Build / Test / Release Cycles



Fundamental Formula of AI

PROBABILISTIC INFERENCE

$$act^* = \text{argmax}_{\text{a in actions}} E(Utility(a, s))$$

PLANNING

STATE ESTIMATION

Concrete Problems in AI Safety

Dario Amodei* Google Brain

Chris Olah* Google Brain

Jacob Steinhardt Stanford University

Paul Christiano UC Berkeley

John Schulman OpenAI

Dan Mané Google Brain

Abstract

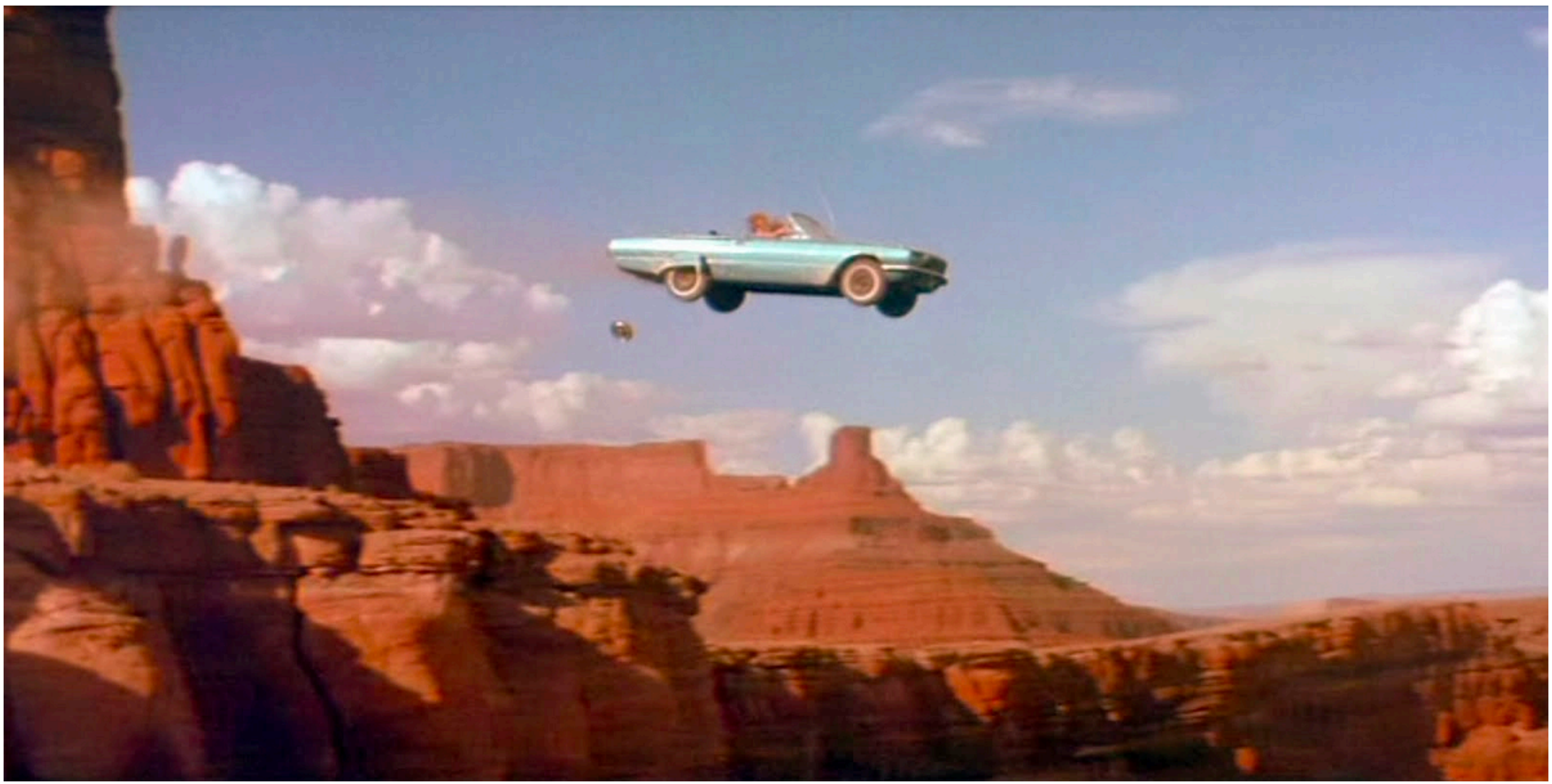
Rapid progress in machine learning and artificial intelligence (AI) has brought increasing attention to the potential impacts of AI technologies on society. In this paper we discuss one such potential impact: the problem of *accidents* in machine learning systems, defined as unintended and harmful behavior that may emerge from poor design of real-world AI systems. We present a list of five practical research problems related to accident risk, categorized according to whether the problem originates from having the wrong objective function ("avoiding side effects" and "avoiding reward hacking"), an objective function that is too expensive to evaluate frequently ("scalable supervision"), or undesirable behavior during the learning process ("safe exploration" and "distributional shift"). We review previous work in these areas as well as suggesting research directions with a focus on relevance to cutting-edge AI systems. Finally, we consider the high-level question of how to think most productively about the safety of forward-looking applications of AI.

Avoiding Negative Side Effects

Avoiding Reward Hacking

Mechanism Design

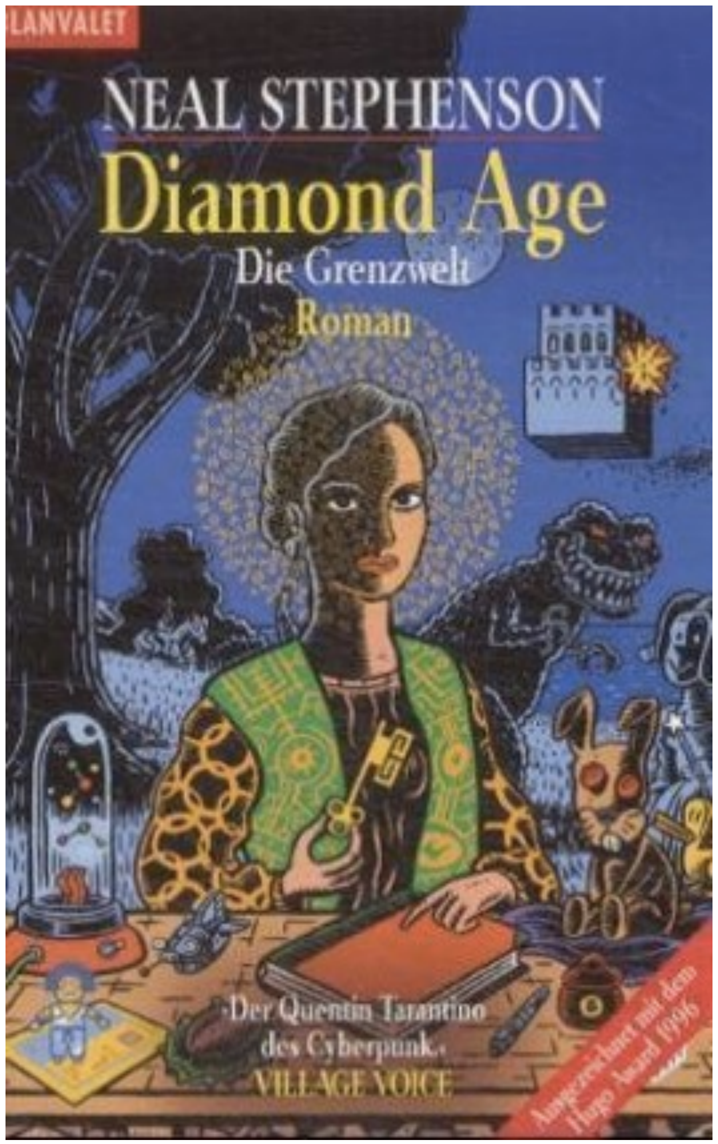
Safe Exploration

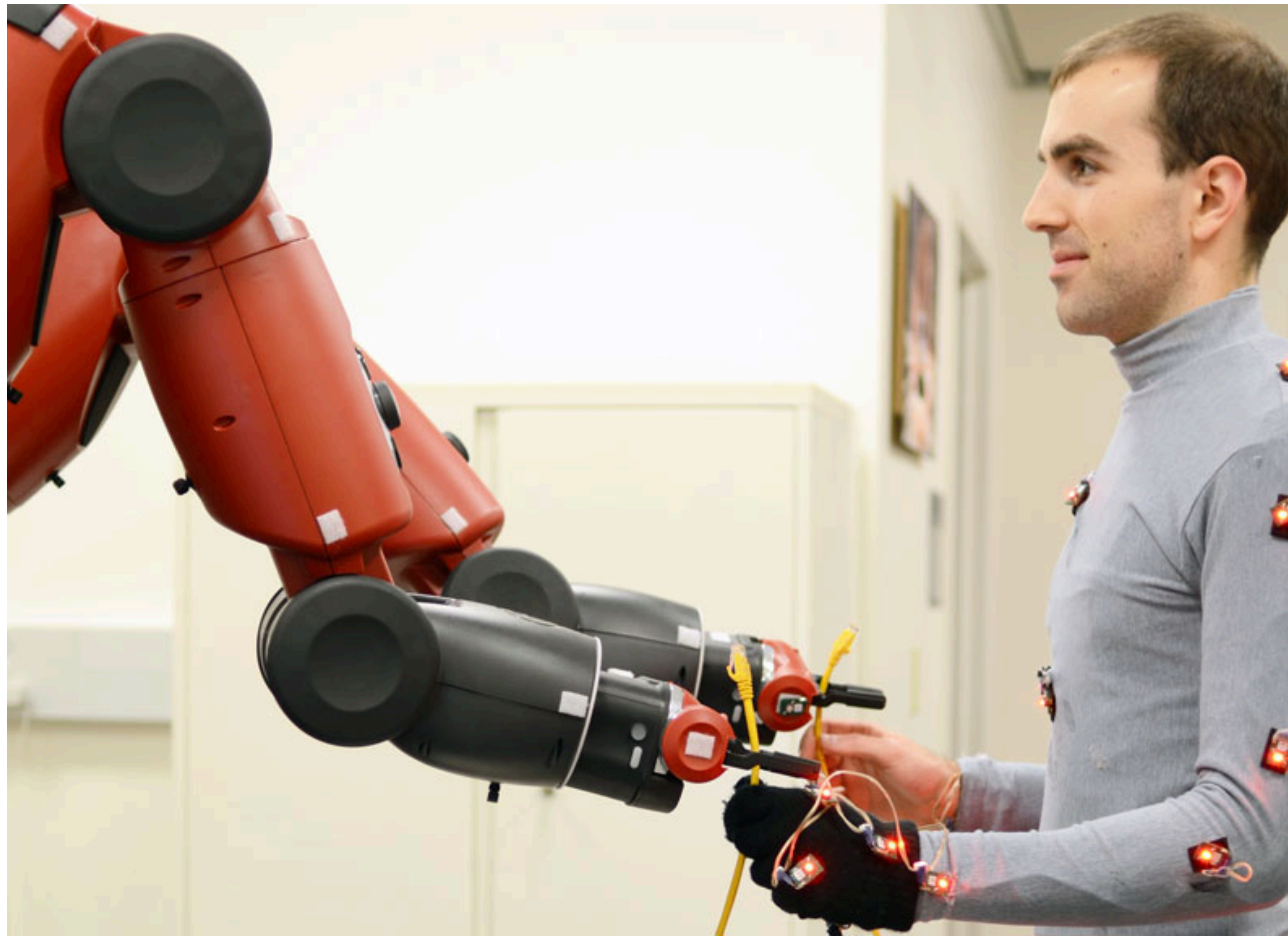


Inattention Valley

Transfer Learning

Scalable Oversight





“the worst ··· except all the others that have been tried”