Where should I travel next? Modeling multi-destination trips with Recurrent Neural Networks.

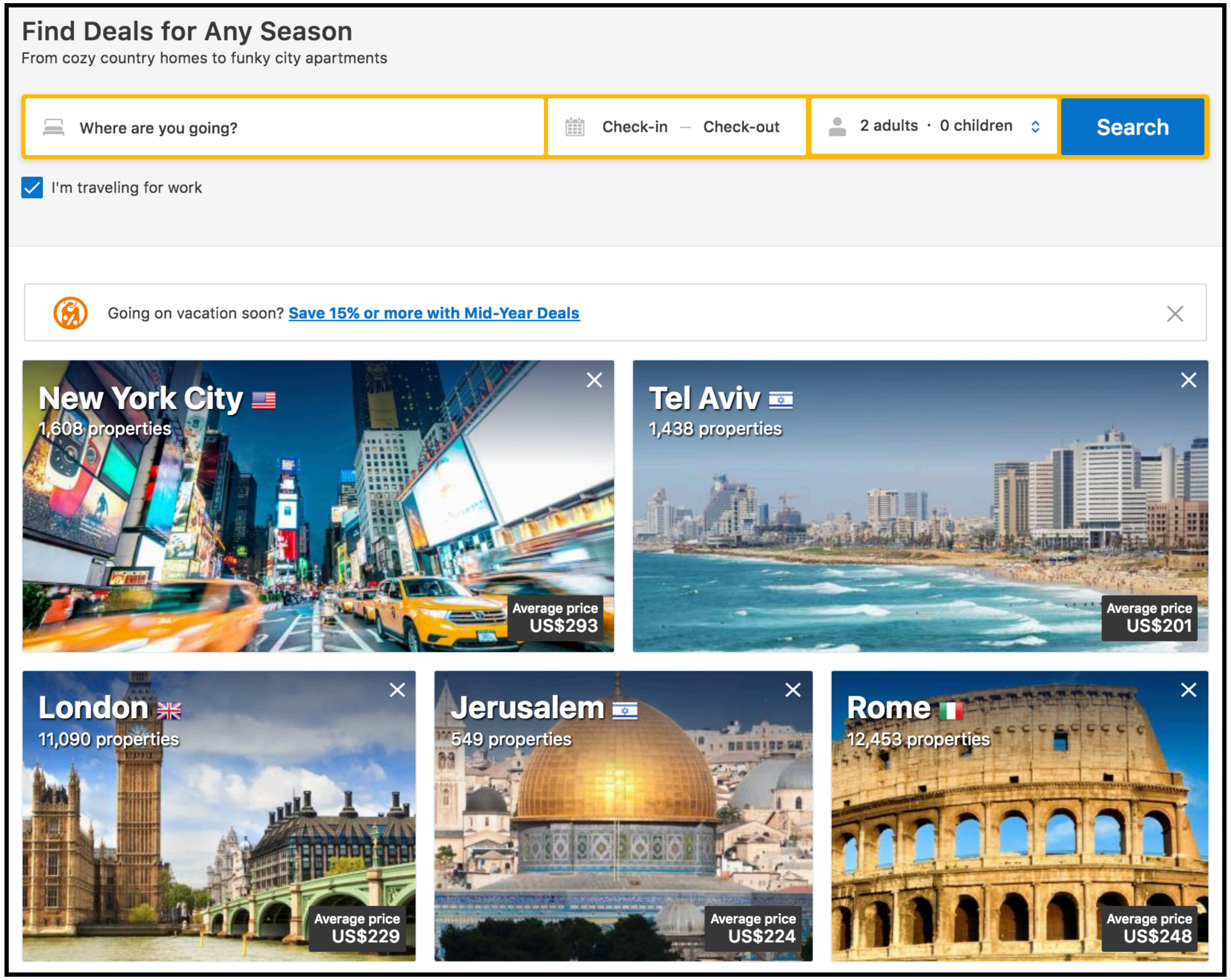
Pavel Levin
Sr. Data Scientist - Booking.com

- Who am I?
    - **Data Scientist at Booking.com for about 4 years**
    - Studied Applied Math (McGill University, University of Ottawa)
    - Prior to Booking: medical research (psychiatry, pain research, drug resistance modeling, cancer trials)
- About Booking.com
    - Founded in 1996; part of Booking Holdings (NASDAQ:BKNG) from 2005
    - Over 29 million listings including everything from apartments and holiday homes to hotels and resorts, and even igloos and tree houses with 164,000,000+ guest reviews
    - **140,000+ destinations** in 230 countries and territories
    - **1,550,000+ rooms booked every 24 hours**
    - 17,000+ employees in over 200 offices in 70 countries

Booking.com

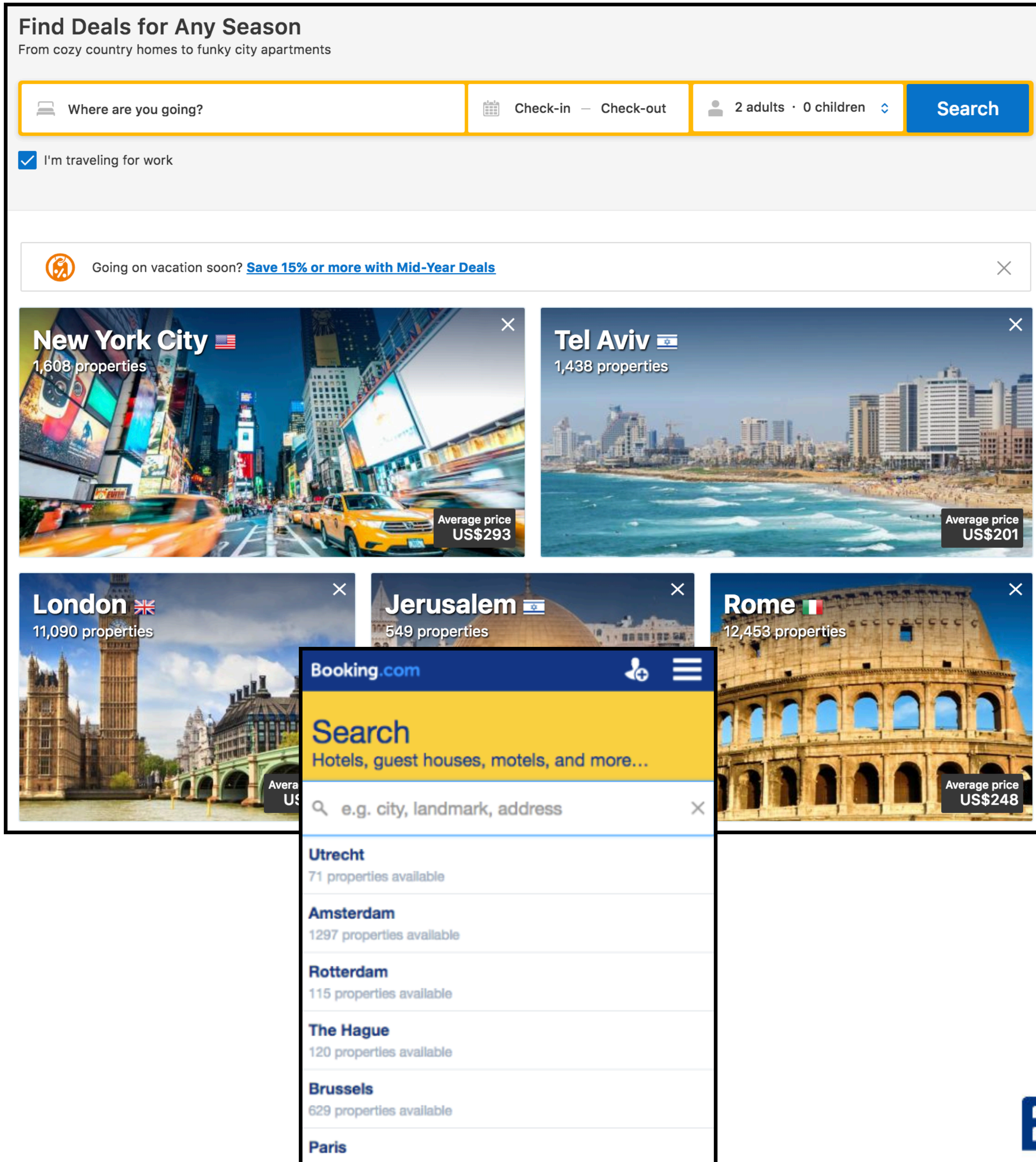
Recommending destinations

Recommending destinations



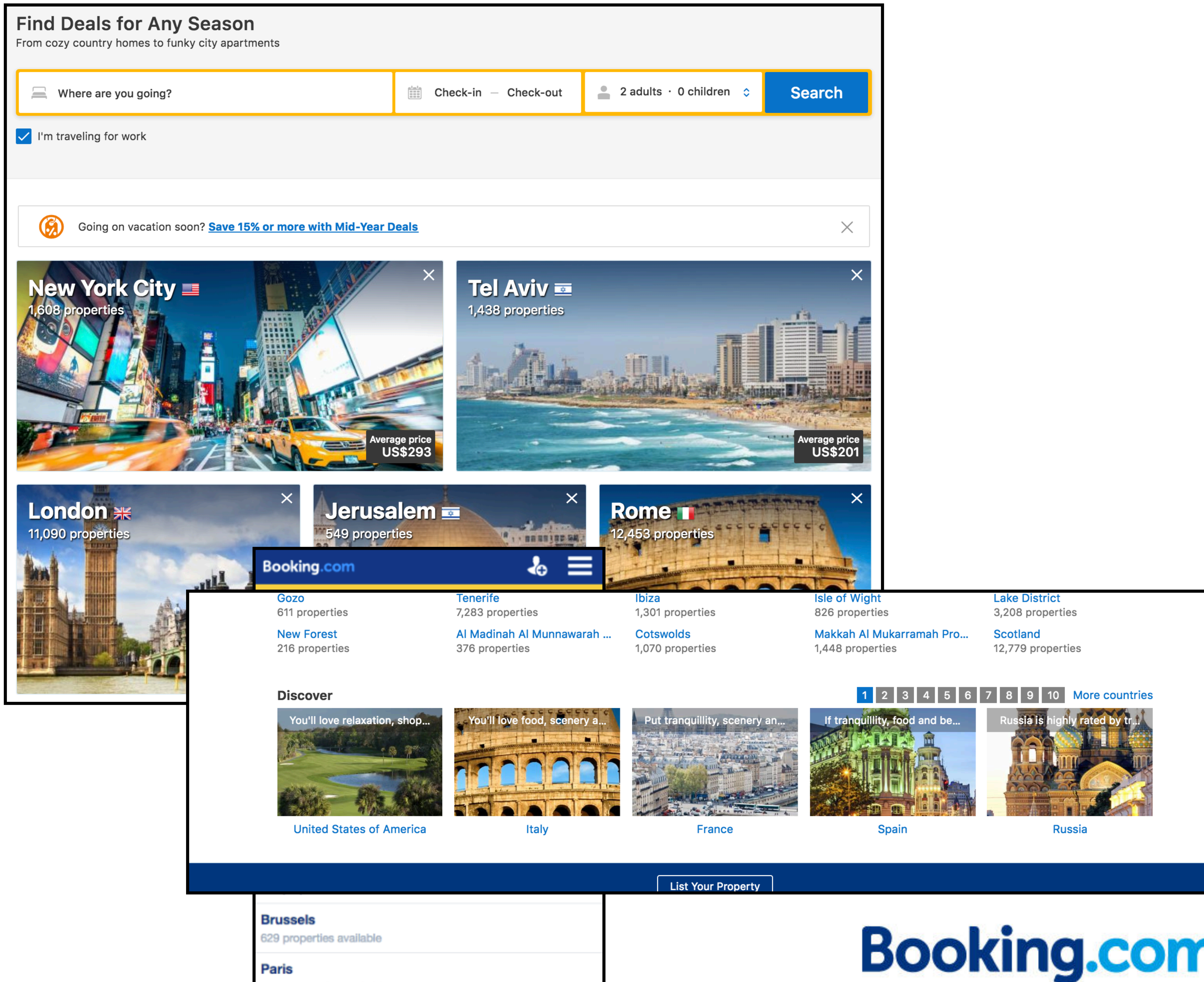
Booking.com

Recommending destinations



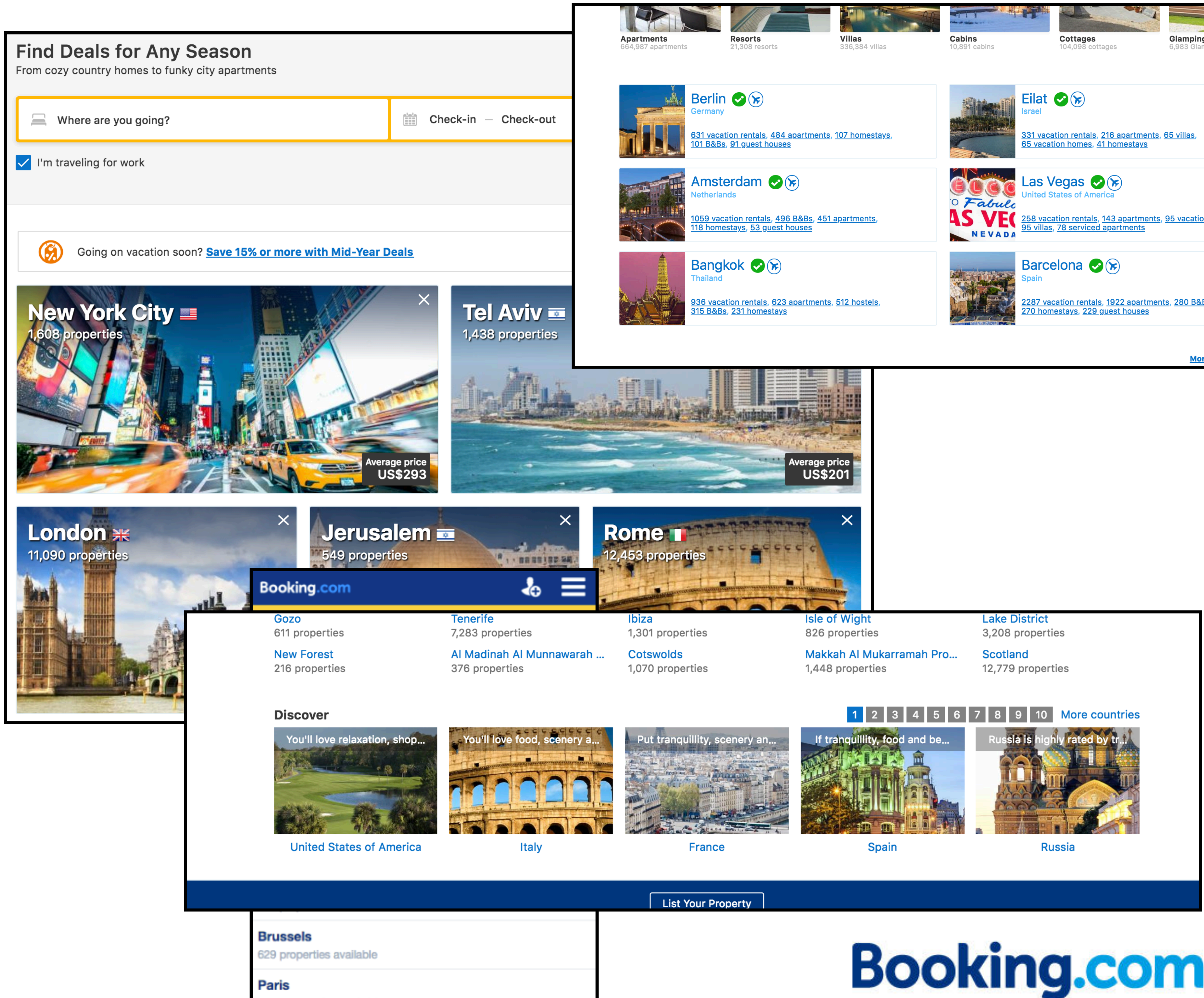
Booking.com

Recommending destinations



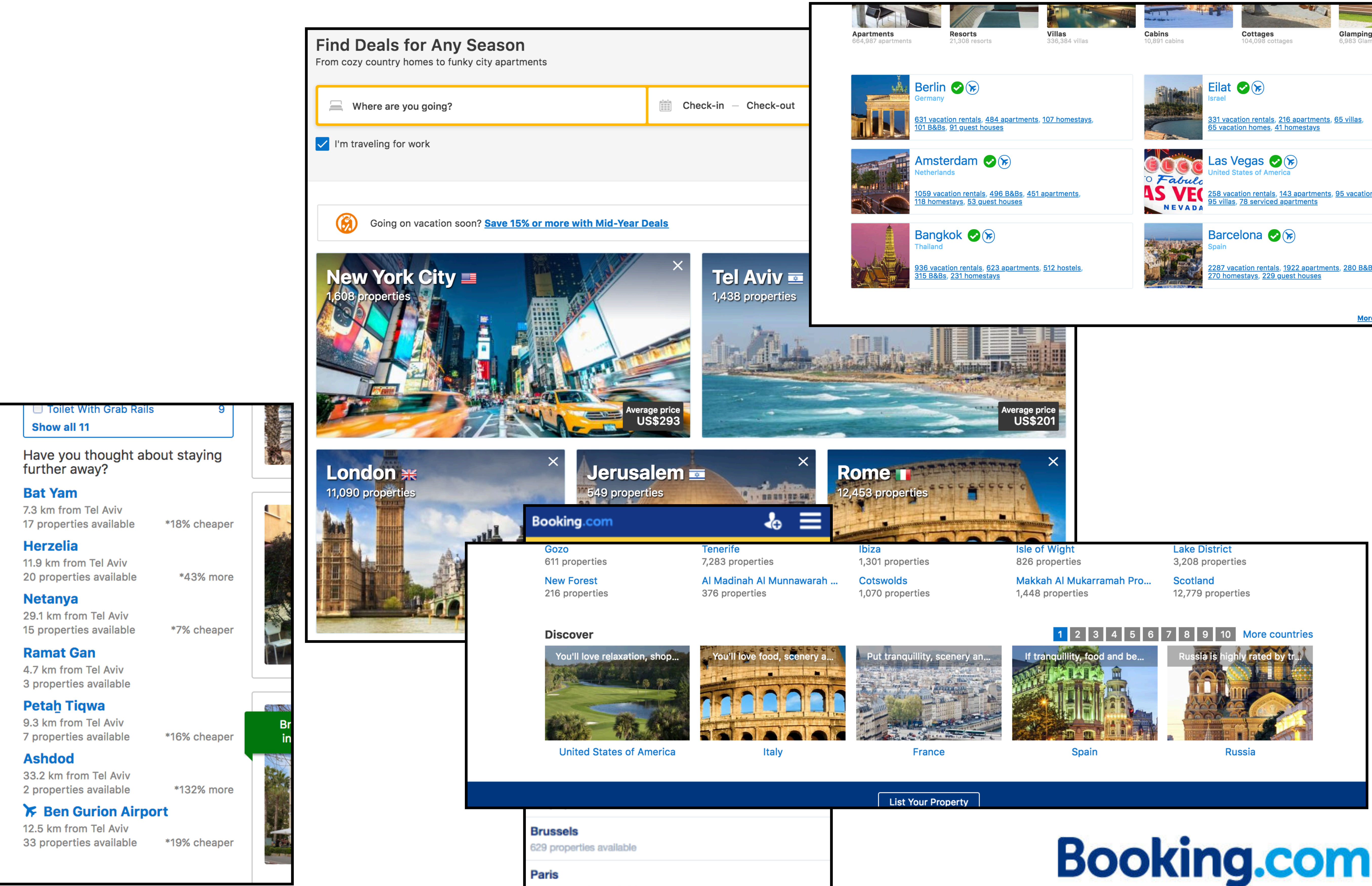
Booking.com

Recommending destinations

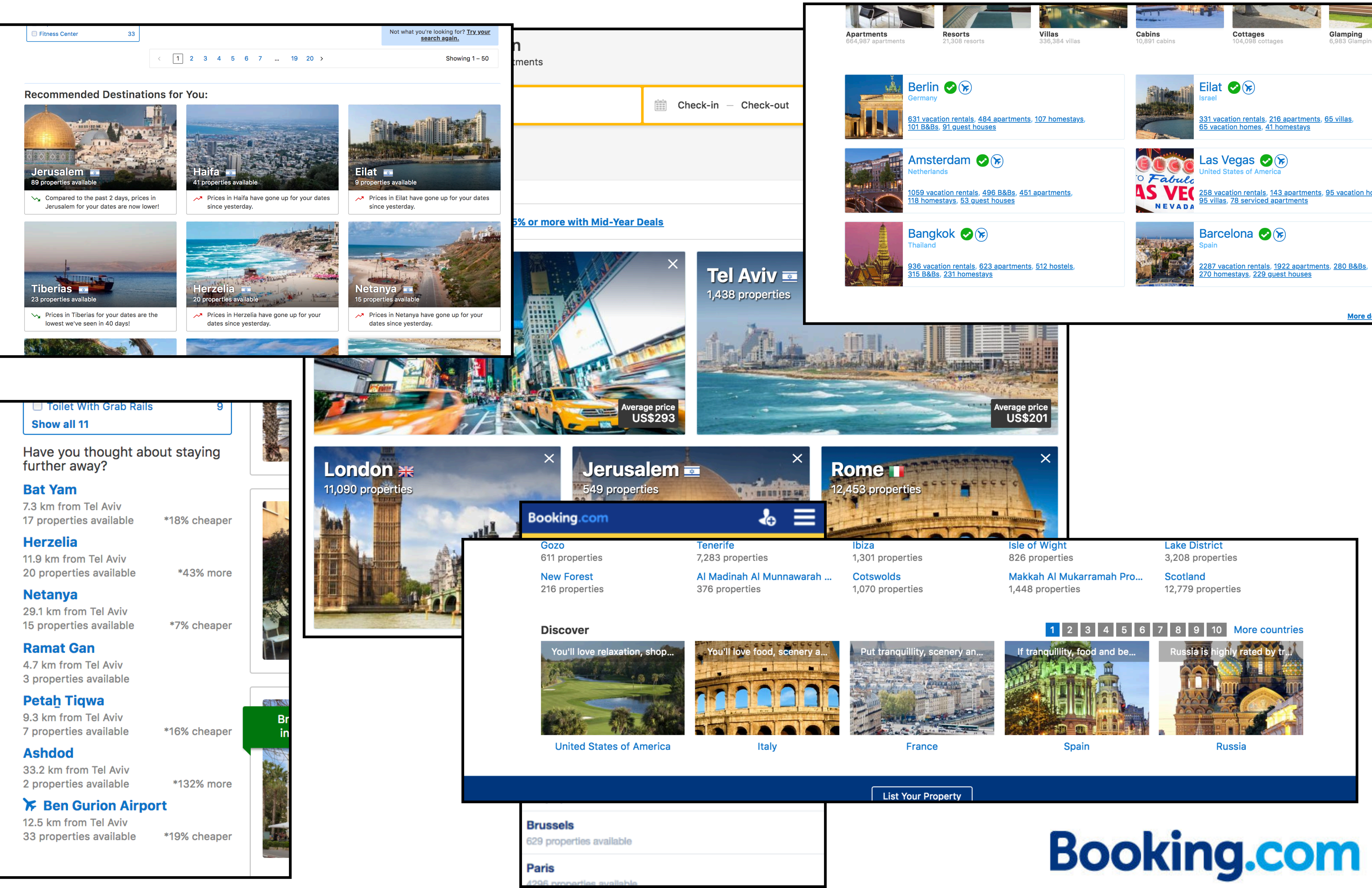


Booking.com

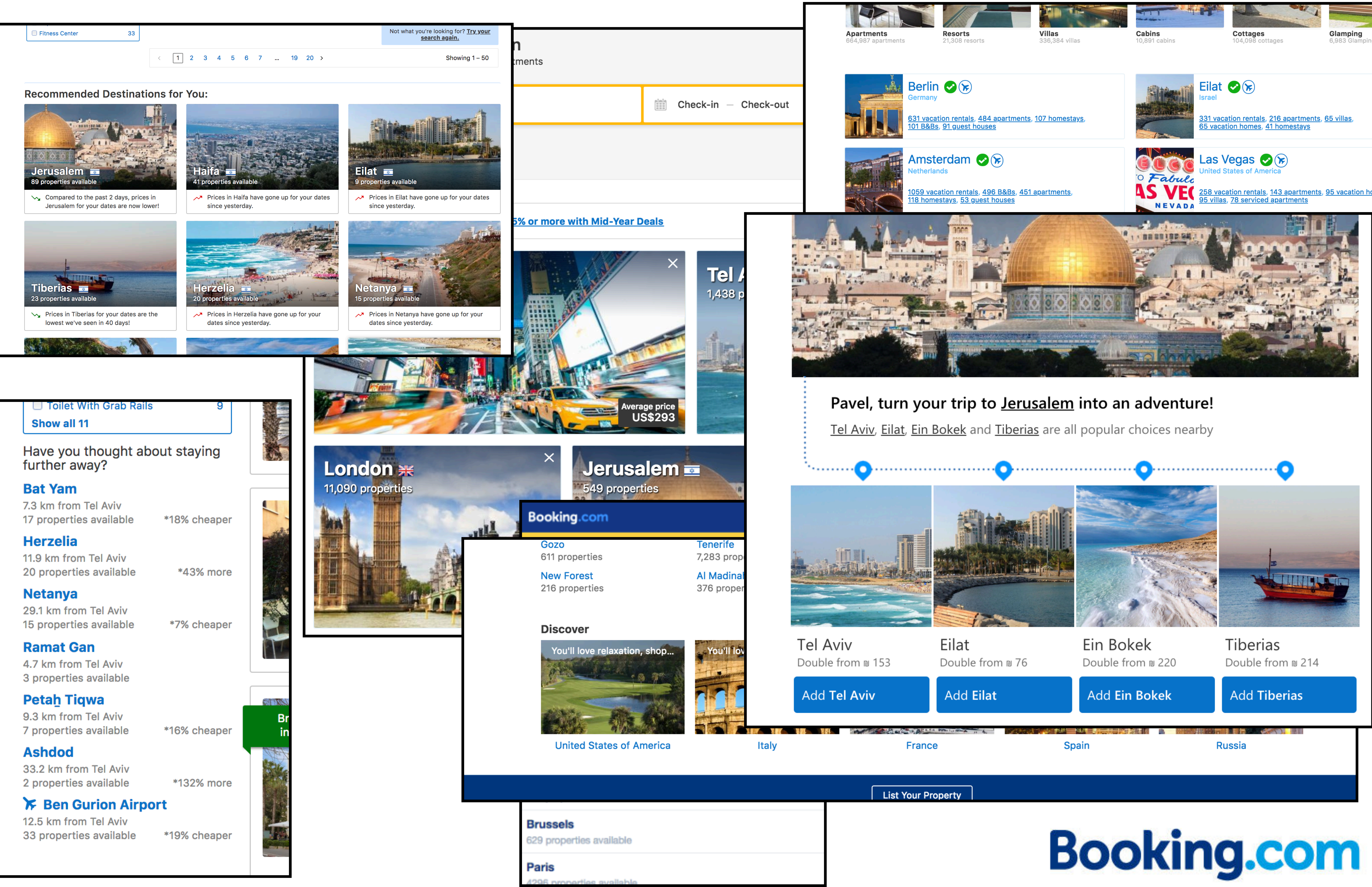
Recommending destinations



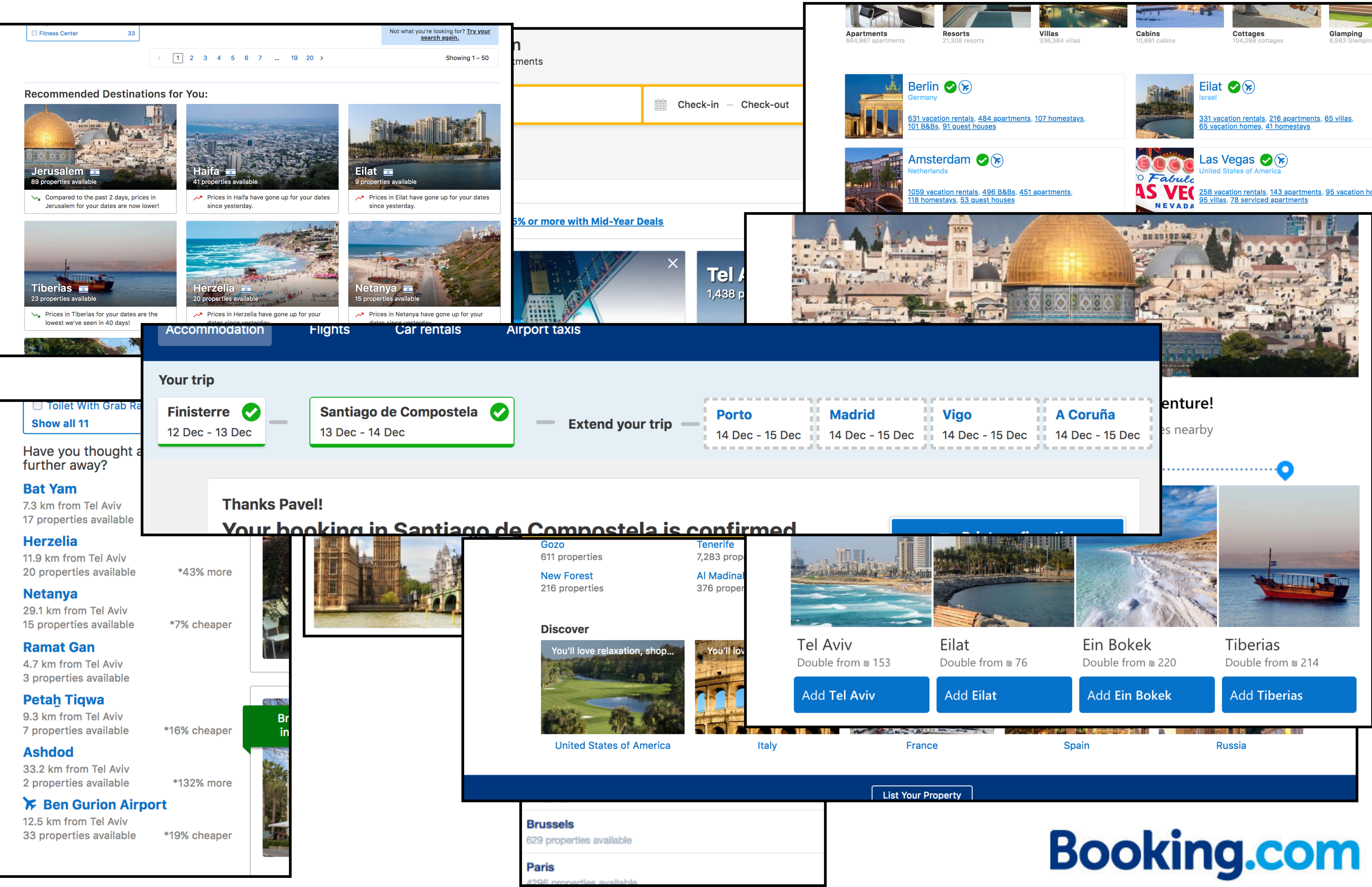
Recommending destinations



Recommending destinations



Recommending destinations



What we’ll cover

- Travel as a sequence modeling problem
  - Sequence modelling
  - The multi-destination trip problem
  - RNNs
- Implementation details
  - data collection, batching, training
  - practical considerations
  - putting it in production
- Some use cases
- Beyond single-step predictions

Sequence modeling

Sequence modeling

Typical example: language model

Sequence modeling

Typical example: language model

- Sentence scoring:

$$\Pr\{\textit{She likes reading books}\} > \Pr\{\textit{She likes reading bookings}\}$$

Booking.com

Sequence modeling

Typical example: language model

- Sentence scoring:

$$\Pr\{\textit{She likes reading books}\} > \Pr\{\textit{She likes reading bookings}\}$$

- Machine Translation:

$$\Pr\{\textit{The apartment offers a terrace.} \mid \textit{Het appartement beschikt over een terras.}\}$$
$$> \Pr\{\textit{Free WiFi access throughout is available.} \mid \textit{Het appartement beschikt over een terras.}\}$$

Booking.com

Sequence modeling

Typical example: language model

- Sentence scoring:

$$\Pr\{\textit{She likes reading books}\} > \Pr\{\textit{She likes reading bookings}\}$$

- Machine Translation:

$$\Pr\{\textit{The apartment offers a terrace}. | \textit{Het appartement beschikt over een terras.}\}$$
$$> \Pr\{\textit{Free WiFi access throughout is available}. | \textit{Het appartement beschikt over een terras.}\}$$

- Image captioning:

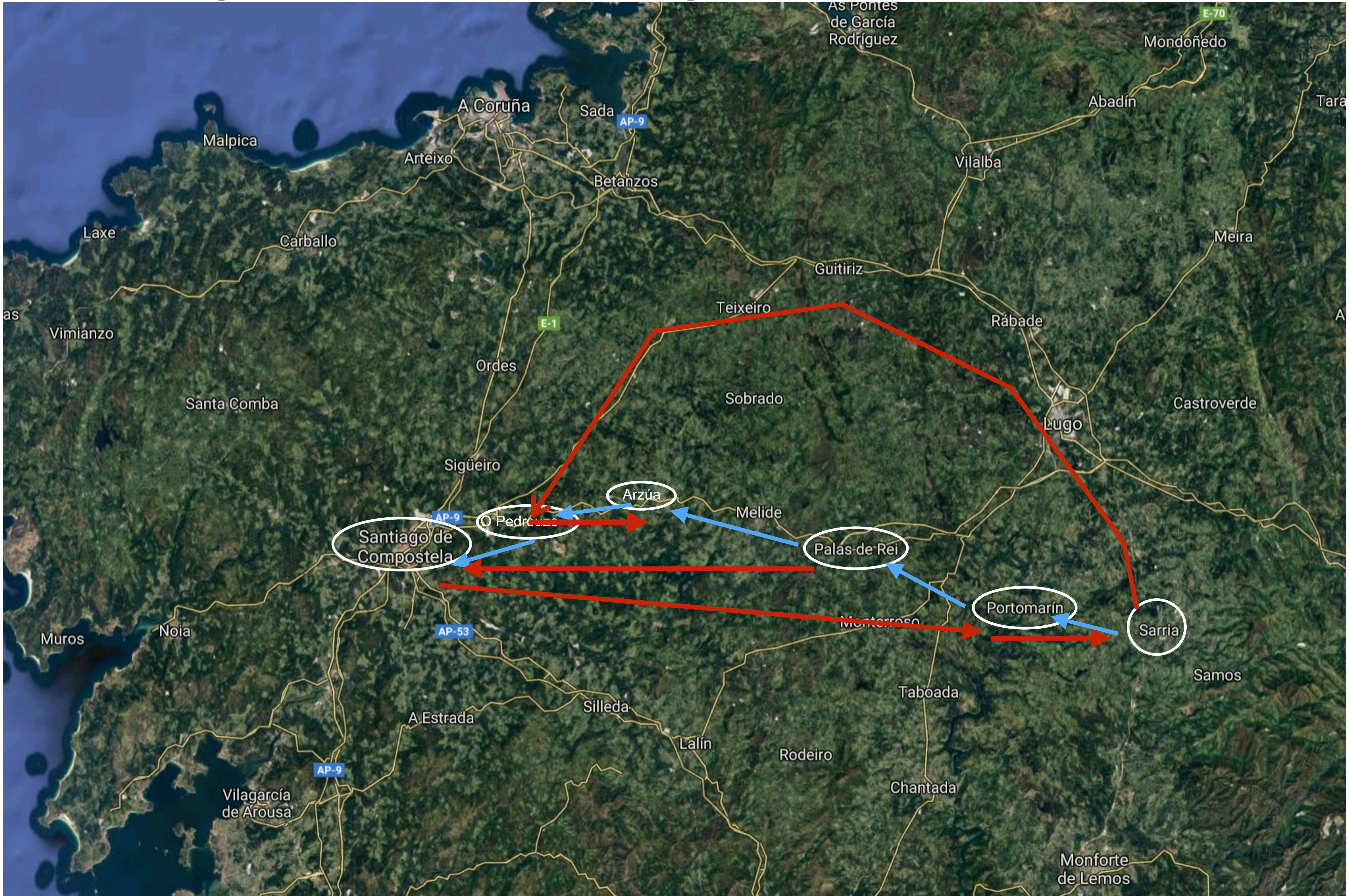
$\Pr\{\textit{Room with a fireplace} |$  $\} > \Pr\{\textit{Room with a private pool} |$  $\}$

Booking.com

Trips are sequences too!

Trips are sequences too!

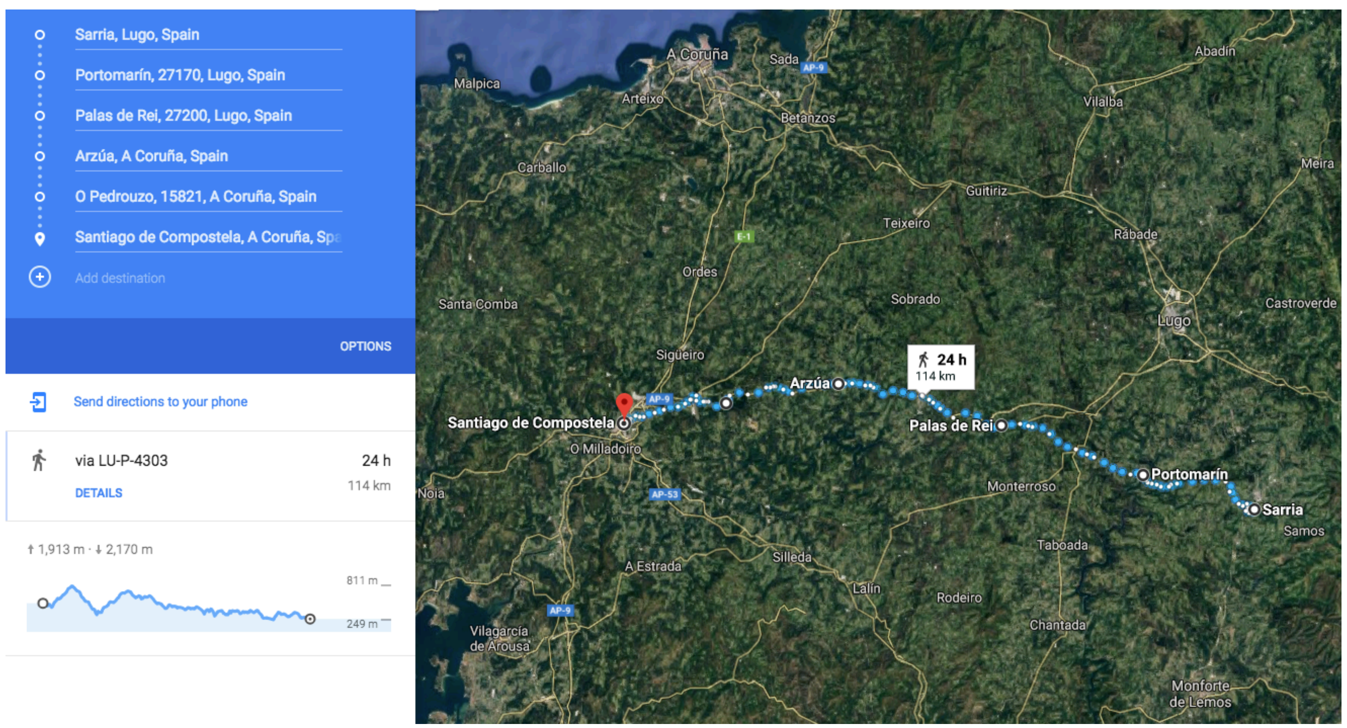
Trips are sequences too!



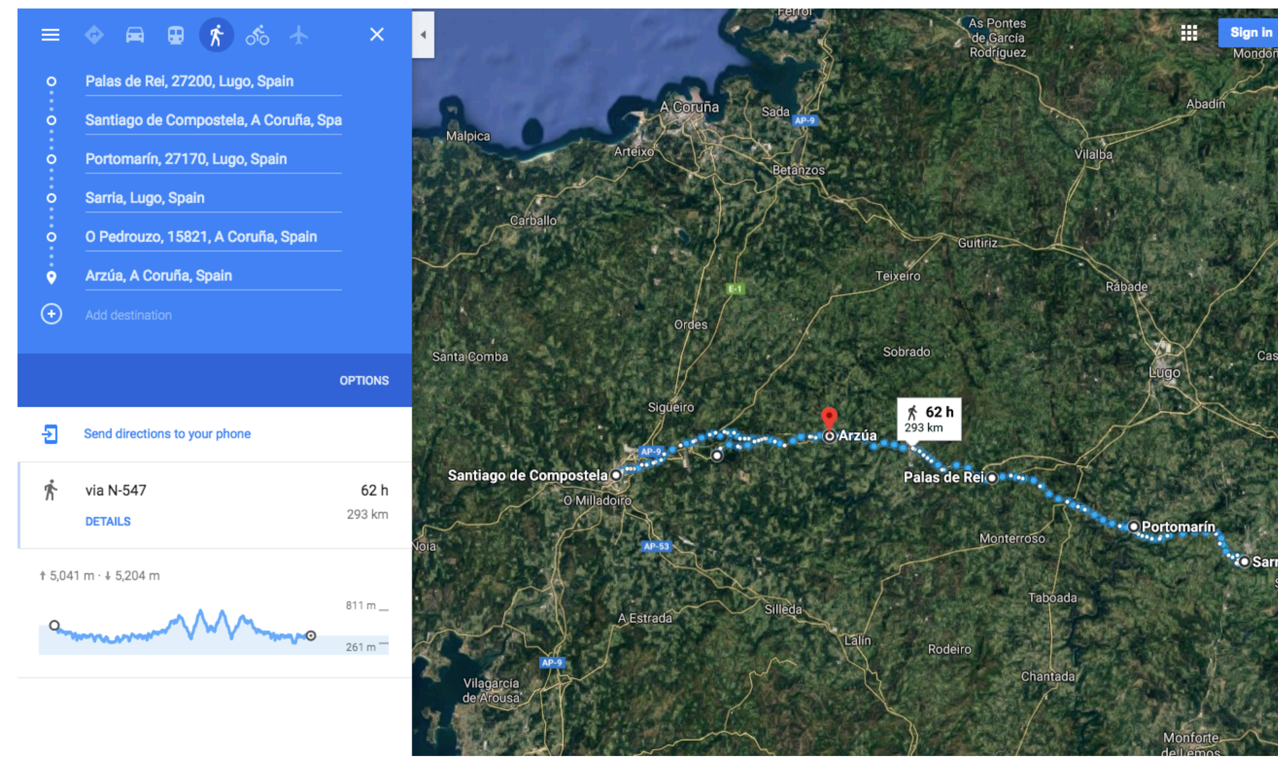
Trips are sequences too!

$$\Pr\{(Sarria \rightarrow Portomarin \rightarrow Palas\ de\ Rei \rightarrow Arzua \rightarrow Pedrouzo \rightarrow Santiago\ de\ Compostela)\}$$

$$> \Pr\{(Palas\ de\ Rei \rightarrow Santiago\ de\ Campostela \rightarrow Portmarin \rightarrow Sarria \rightarrow Pedrouzo \rightarrow Arzua)\}$$



>

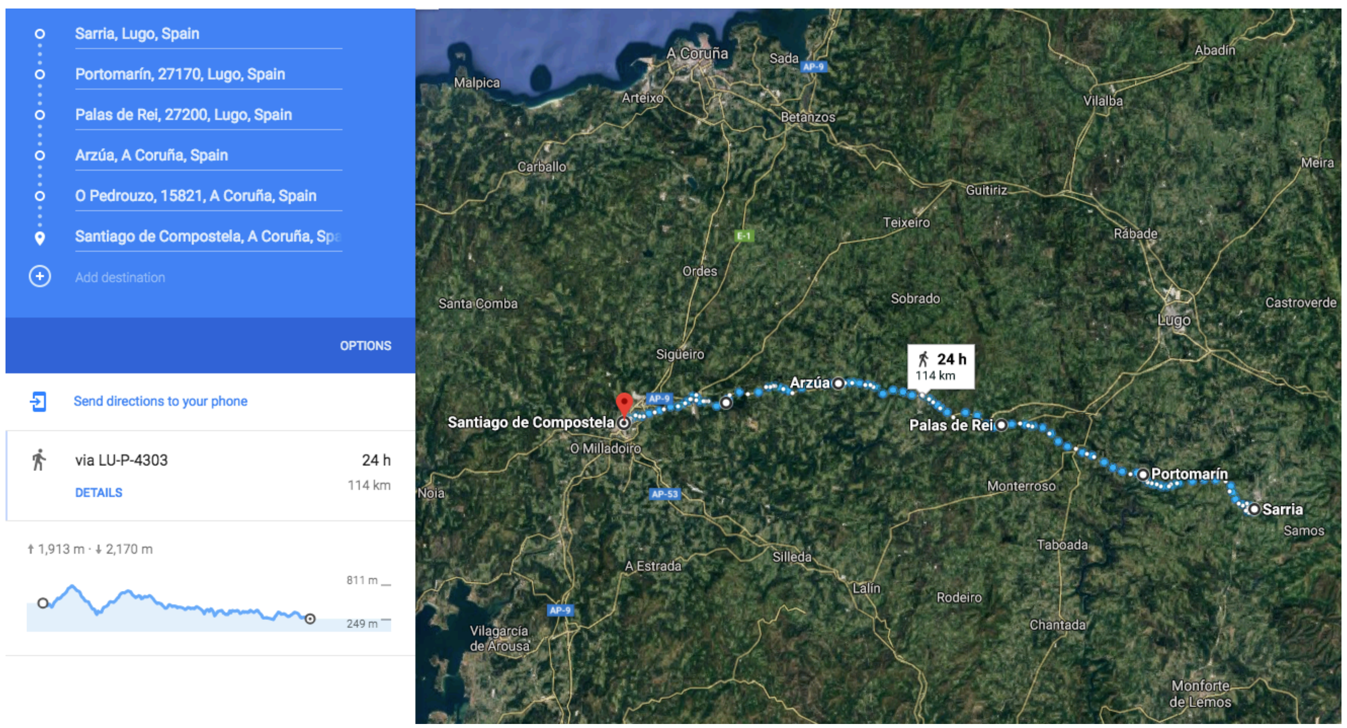


Booking.com

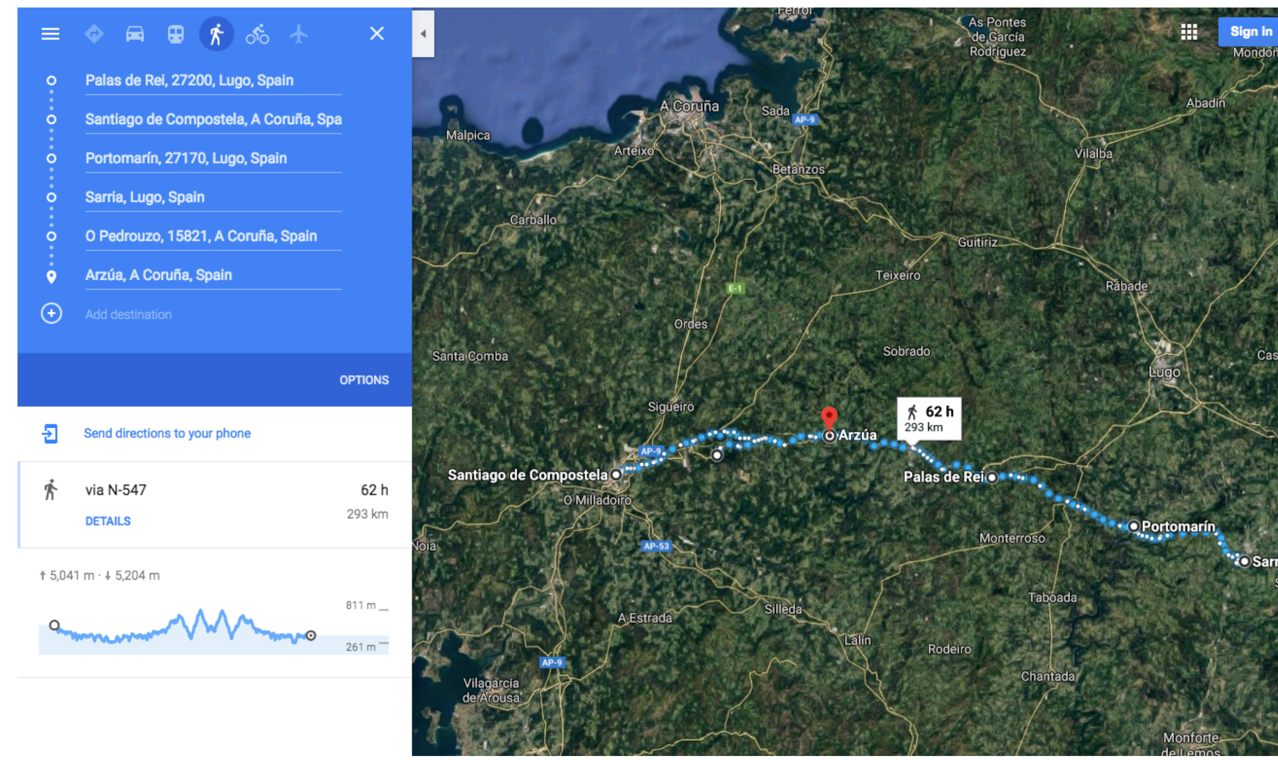
Trips are sequences too!

$$\Pr\{(Sarria \rightarrow Portomarin \rightarrow Palas\ de\ Rei \rightarrow Arzua \rightarrow Pedrouzo \rightarrow Santiago\ de\ Compostela)\}$$

$$> \Pr\{(Palas\ de\ Rei \rightarrow Santiago\ de\ Campostela \rightarrow Portmarin \rightarrow Sarria \rightarrow Pedrouzo \rightarrow Arzua)\}$$



>



Similarly:

$$\Pr\{next\_dest = \textbf{Santiago de Compostela} \mid so\_far = (Sarria, Portomarin, Palas\ de\ Rei, Arzua, Pedrouzo)\}$$

$$> \Pr\{next\_dest = \textbf{Barcelona} \mid so\_far = (Sarria, Portomarin, Palas\ de\ Rei, Arzua, Pedrouzo)\}$$

Booking.com

The multi-leg trip problem

- We have some censored information about the trip (e.g. first few legs)
- Want to predict missing legs
- This work: predicting forward legs only

The multi-leg trip problem

- We have some censored information about the trip (e.g. first few legs)
- Want to predict missing legs
- This work: predicting forward legs only

The problem:

Prediction $\text{argmax}_{d_n \in D} P(d_n \,|\, d_1, \ldots, d_{n-1}; \mathbf{\Theta})$

Learning $\text{argmax}_{\mathbf{\Theta}} \mathbb{E}[P(d_k \,|\, d_1, \ldots d_{k-1}; \mathbf{\Theta})]$

<u>Notation:</u>

D	set of all known destinations
$\mathbf{d} = (d_1, d_2, \ldots, d_n)$	trip (ordered list of destinations)
$\mathbf{\Theta}$	model parameters
$P(\mathbf{d} \,|\, \mathbf{\Theta})$	probability of the trip/ likelihood of the model
$P(d_n \,|\, d_1, \ldots, d_{n-1}; \mathbf{\Theta})$	next leg probability

The multi-leg trip problem

- We have some censored information about the trip (e.g. first few legs)
- Want to predict missing legs
- This work: predicting forward legs only

The problem:

Prediction $\text{argmax}_{d_n \in D} P(d_n \mid d_1, \ldots, d_{n-1}; \boldsymbol{\Theta})$

Learning $\text{argmax}_{\boldsymbol{\Theta}} \mathbb{E}[P(d_k \mid d_1, \ldots d_{k-1}; \boldsymbol{\Theta})]$

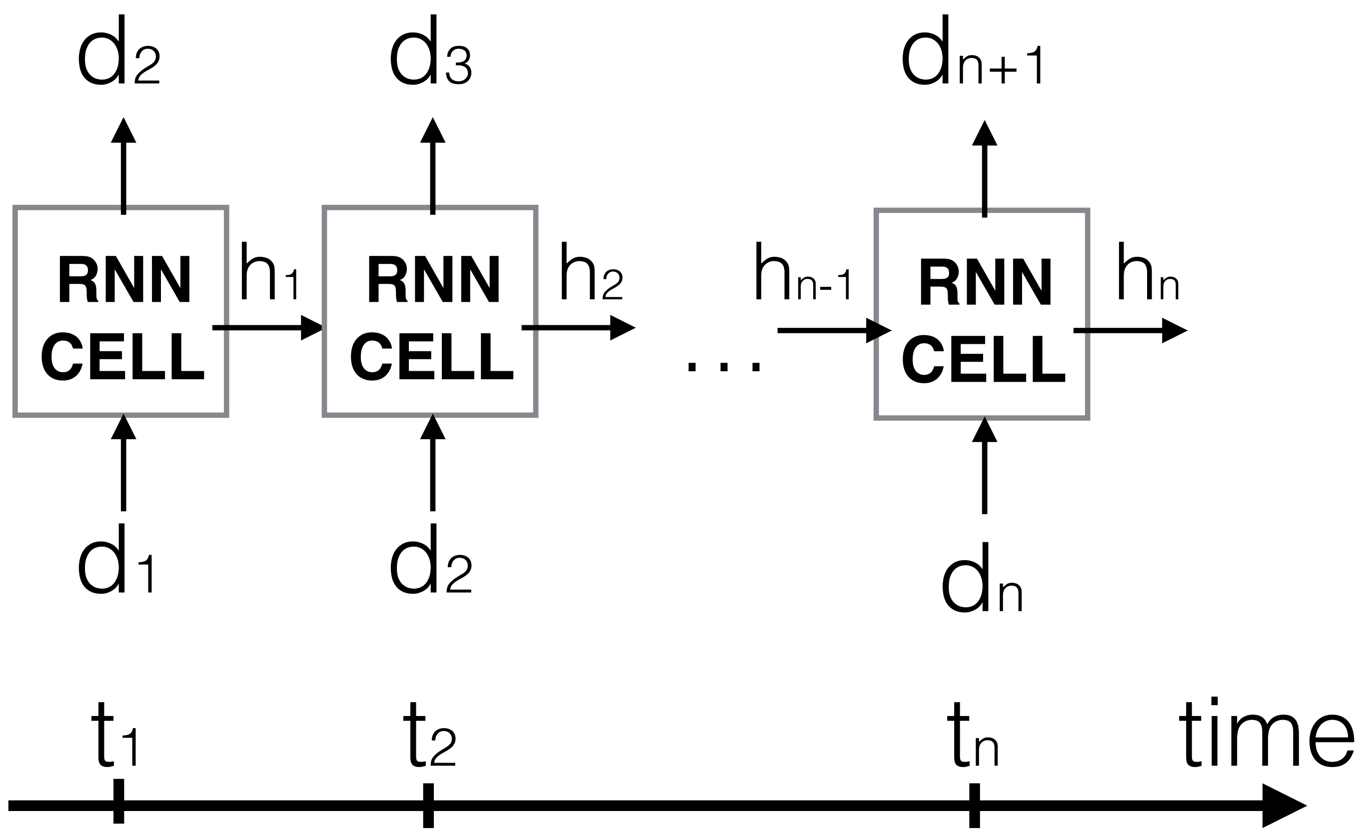
Notation:

D	set of all known destinations
$\mathbf{d} = (d_1, d_2, \ldots, d_n)$	trip (ordered list of destinations)
$\boldsymbol{\Theta}$	model parameters
$P(\mathbf{d} \mid \boldsymbol{\Theta})$	probability of the trip/ likelihood of the model
$P(d_n \mid d_1, \ldots, d_{n-1}; \boldsymbol{\Theta})$	next leg probability

Can use RNNs to solve!

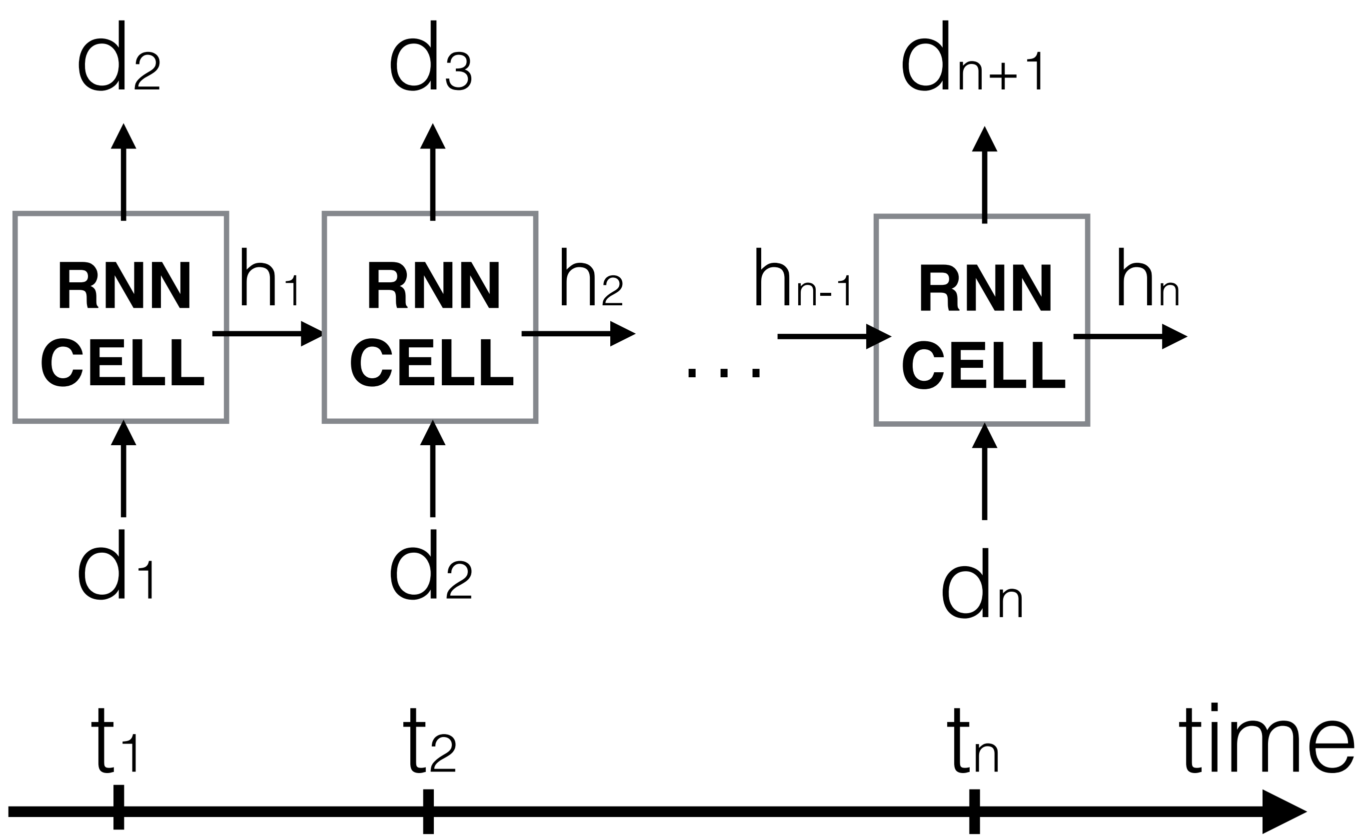
Booking.com

What's an RNN?

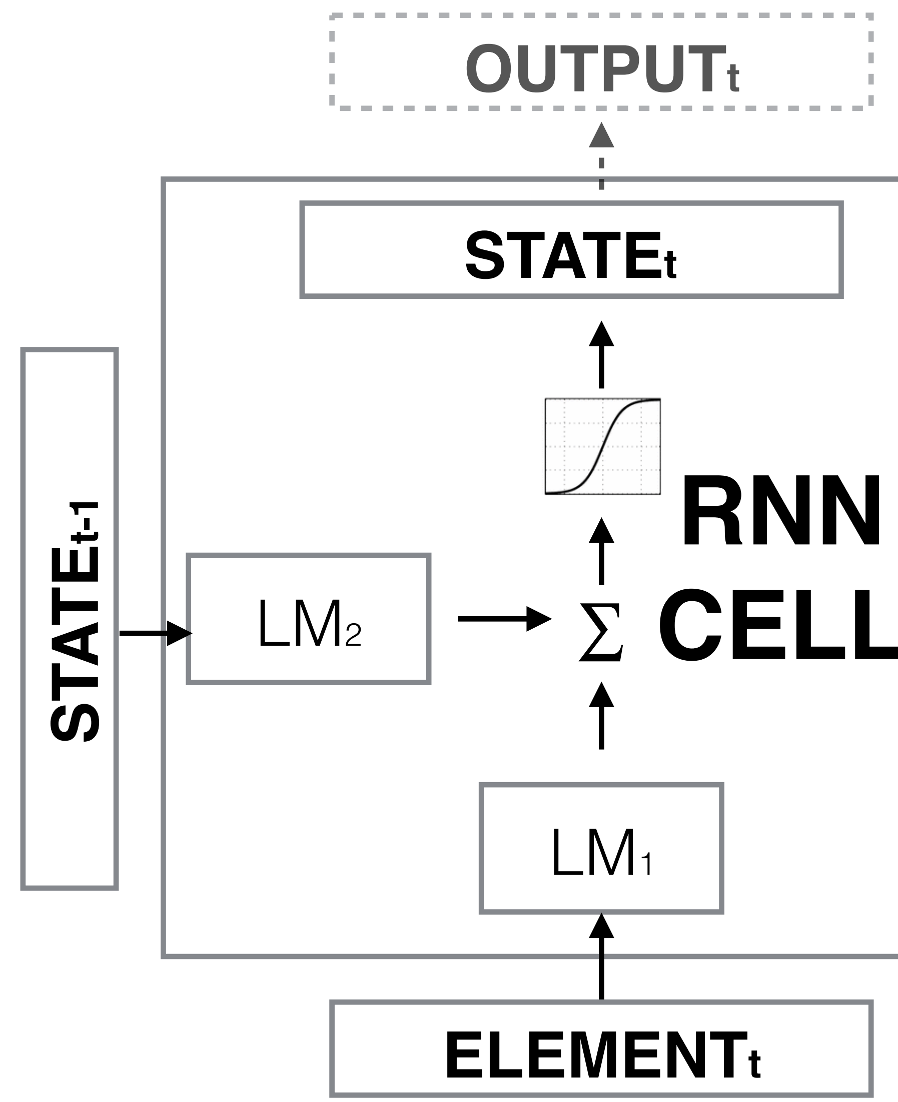


d2
d3
dn+1
RNN CELL
h1
RNN CELL
h2
hn-1
RNN CELL
hn
. . .
d1
d2
dn
t1
t2
tn
time

What's an RNN?

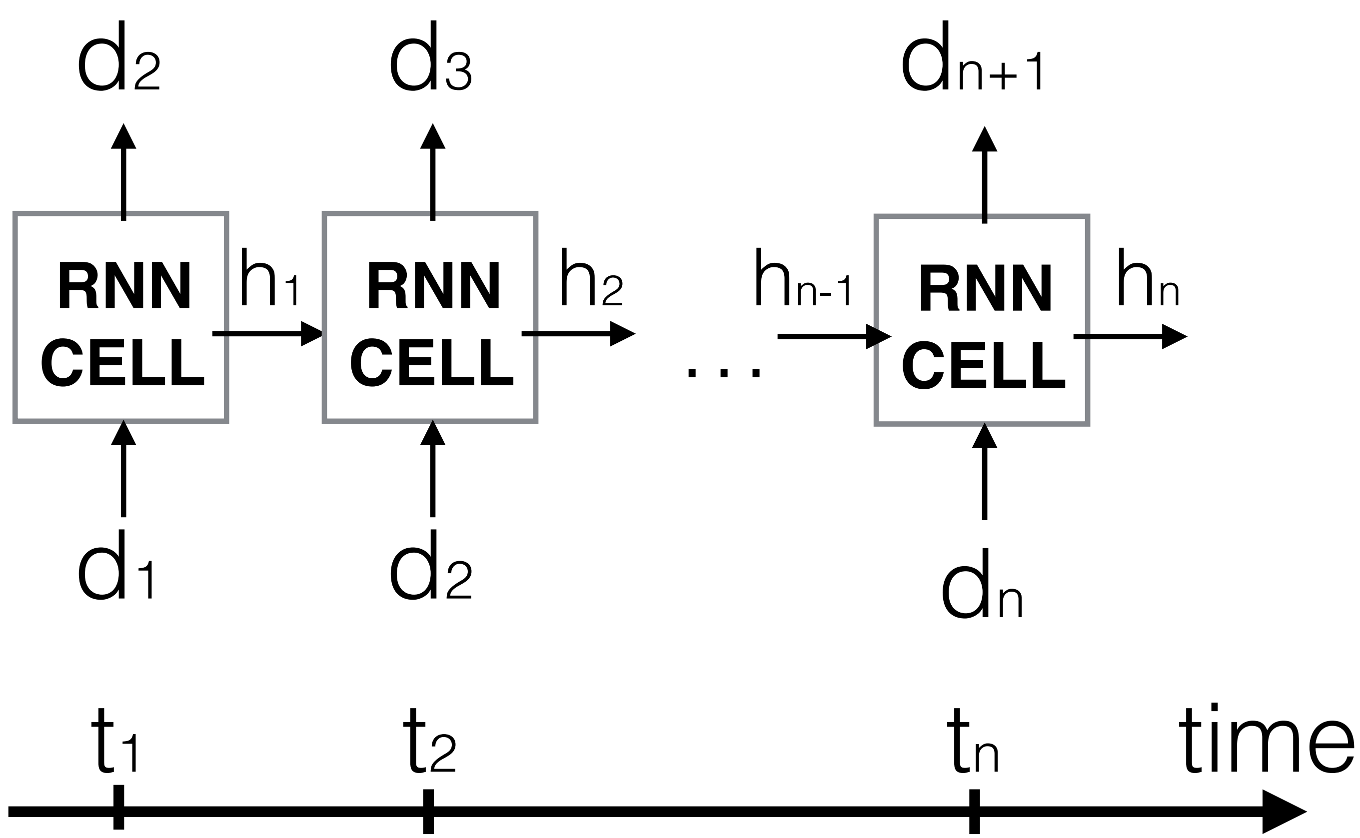


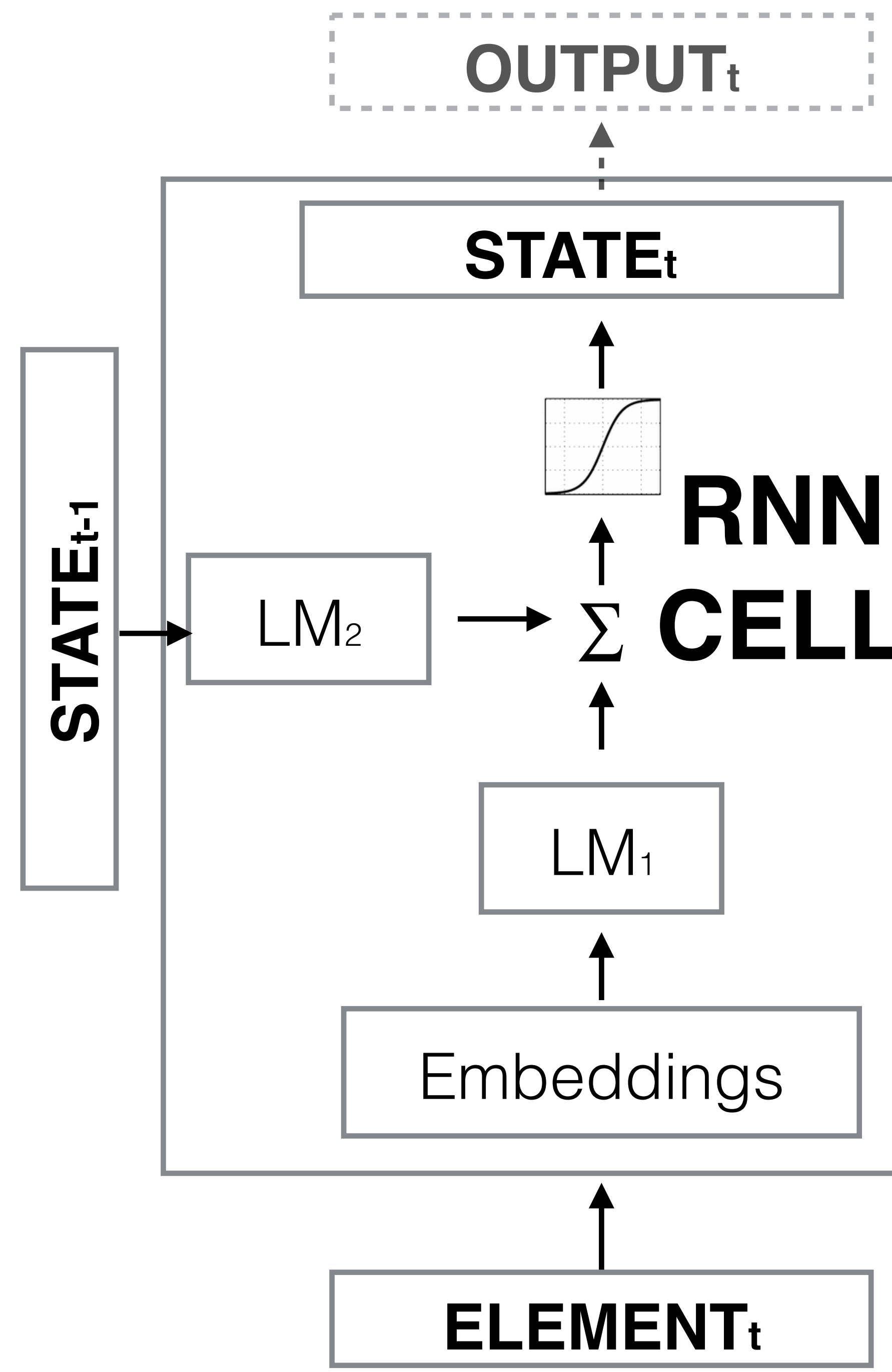
d2
d3
dn+1
RNN CELL
h1
RNN CELL
h2
hn-1
RNN CELL
hn
d1
d2
dn
t1
t2
tn
time



OUTPUTt
STATEt
RNN CELL
STATEt-1
LM2
Σ
LM1
ELEMENTt

What's an RNN?





Booking.com

Learning RNN

Learning RNN

The RNN model:

$$x_t = \Theta_d d_t$$
$$h_t = \tanh(\Theta_x(x_t) + \Theta_h(h_{t-1}) + \theta_h)$$
$$y_t = \sigma(\Theta_y h_t + \theta_y)$$

Learning RNN

The RNN model:

$$x_t = \Theta_d d_t$$
$$h_t = \tanh(\Theta_x(x_t) + \Theta_h(h_{t-1}) + \theta_h)$$
$$y_t = \sigma(\Theta_y h_t + \theta_y)$$

What are we optimizing?

Booking.com

Learning RNN

The RNN model:

$$x_t = \Theta_d d_t$$
$$h_t = \tanh(\Theta_x(x_t) + \Theta_h(h_{t-1}) + \theta_h)$$
$$y_t = \sigma(\Theta_y h_t + \theta_y)$$

What are we optimizing?

For any given sequence **d**, we want

$$-\frac{1}{|\mathbf{d}|}\sum_{t=1}^{|\mathbf{d}|} \log(P[RNN(d_t, h_{t-1}) = d_{t+1}])$$ to be as small as possible

Learning RNN

The RNN model:

$$x_t = \Theta_d d_t$$
$$h_t = \tanh(\Theta_x(x_t) + \Theta_h(h_{t-1}) + \theta_h)$$
$$y_t = \sigma(\Theta_y h_t + \theta_y)$$

What are we optimizing?

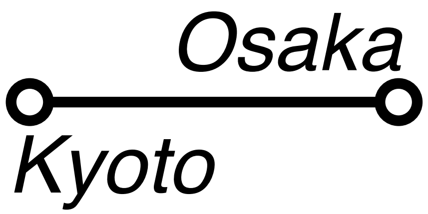
For any given sequence **d**, we want

$$-\frac{1}{|\mathbf{d}|}\sum_{t=1}^{|\mathbf{d}|} \log(P[RNN(d_t, h_{t-1}) = d_{t+1}])$$

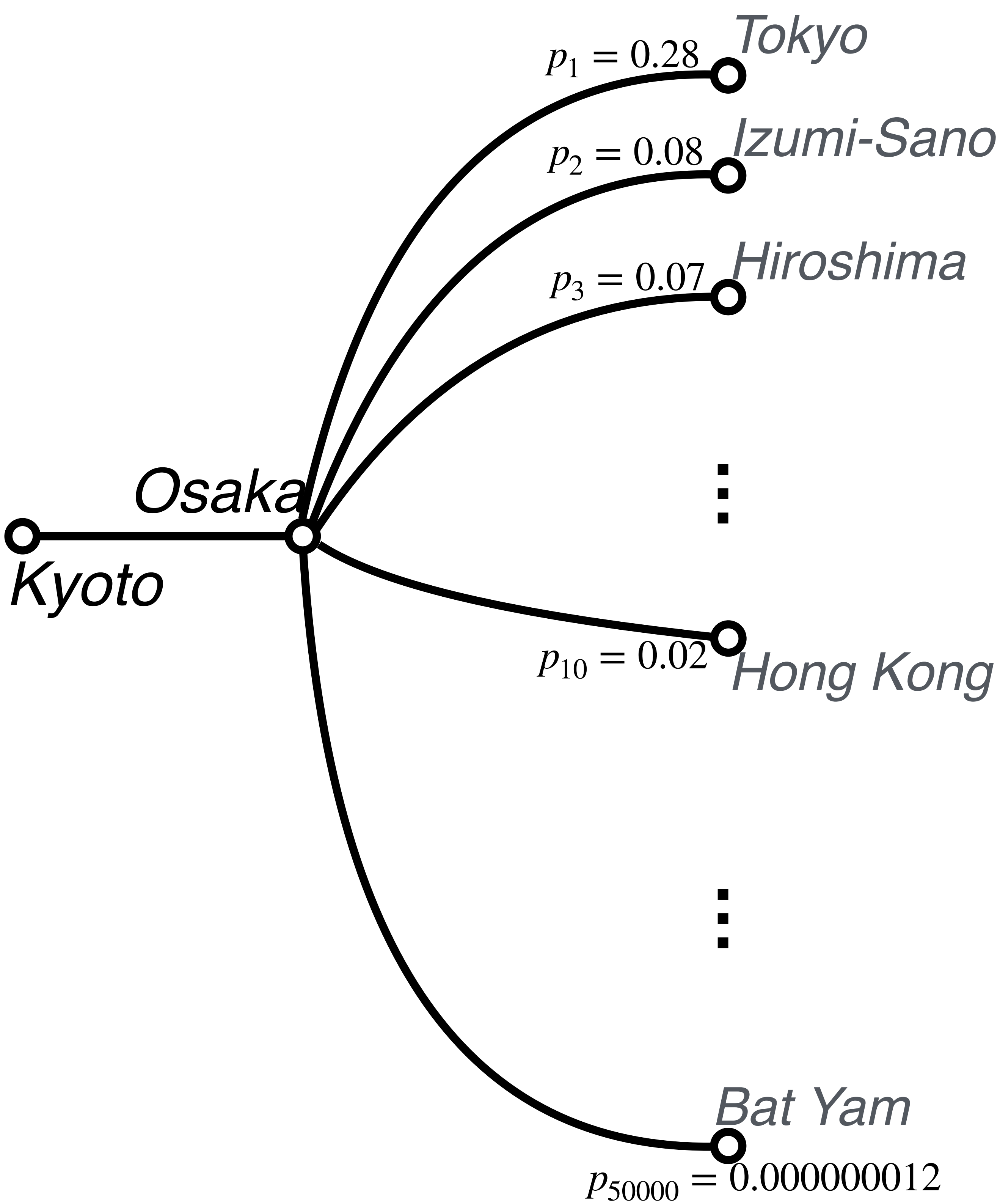
to be as small as possible

i.e. minimize average **log loss** for next element classification
aka minimize average **Cross Entropy** loss
aka minimize **perplexity**

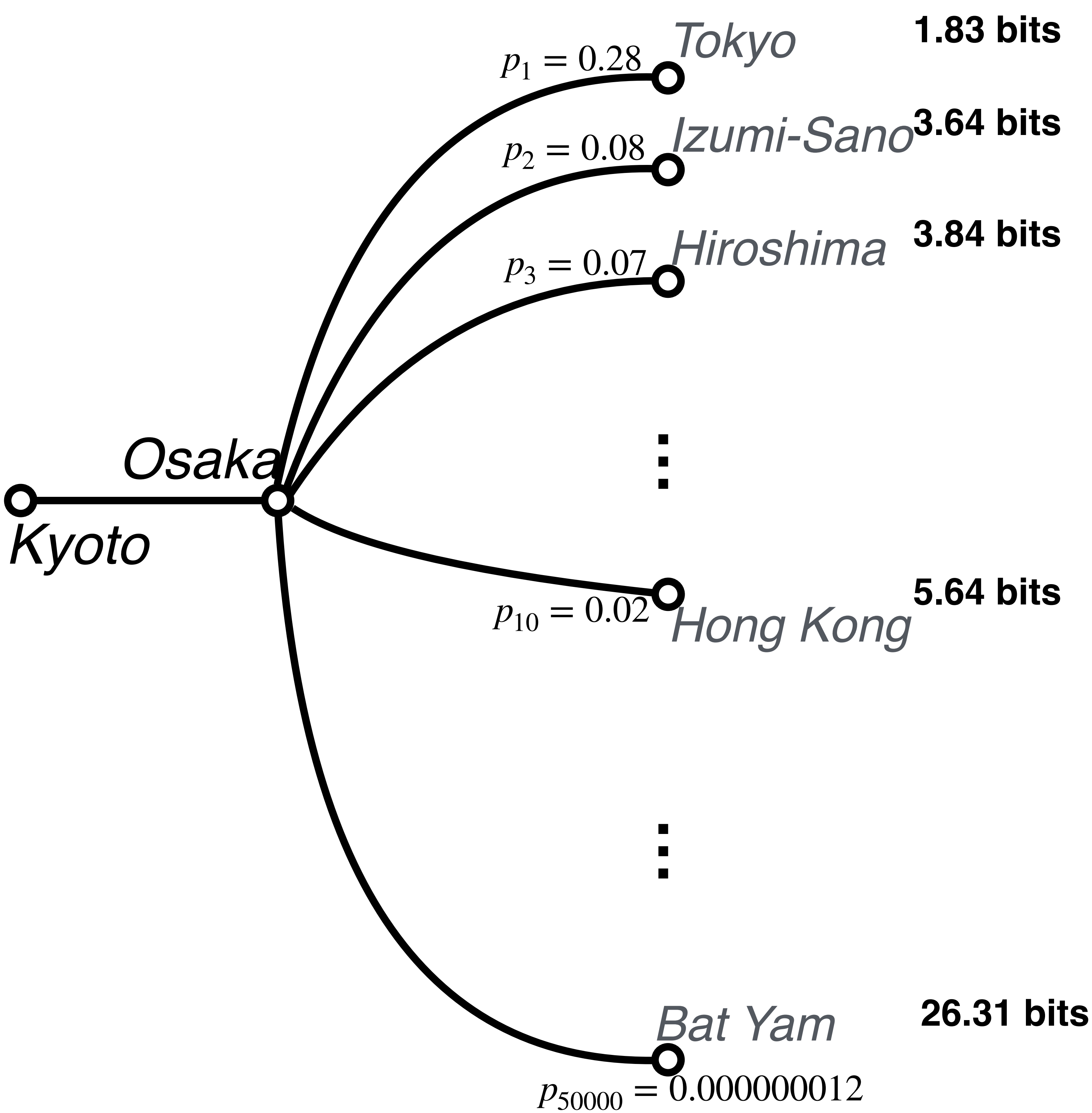
Perplexity: intuition



Perplexity: intuition

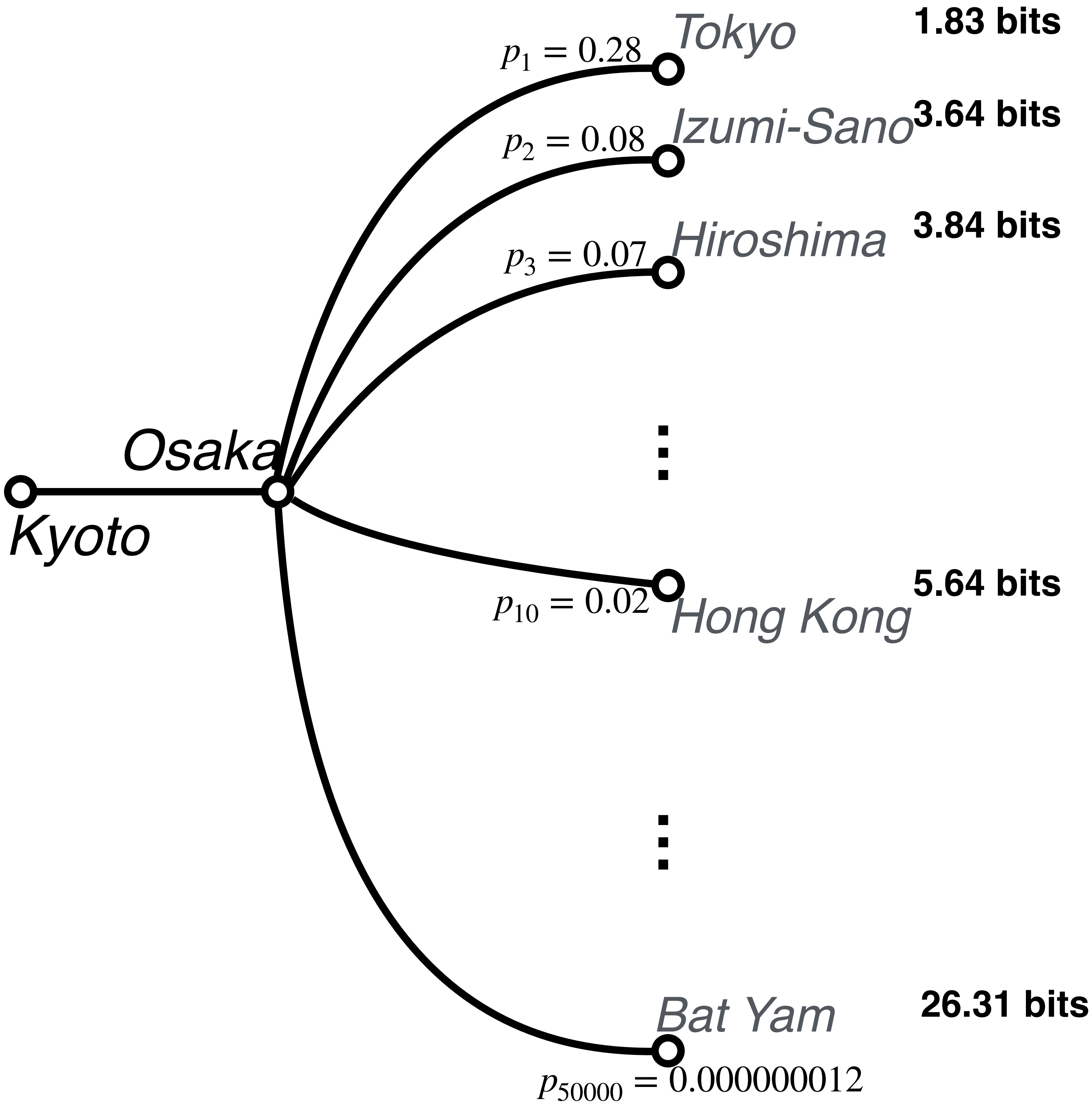


Perplexity: intuition



Booking.com

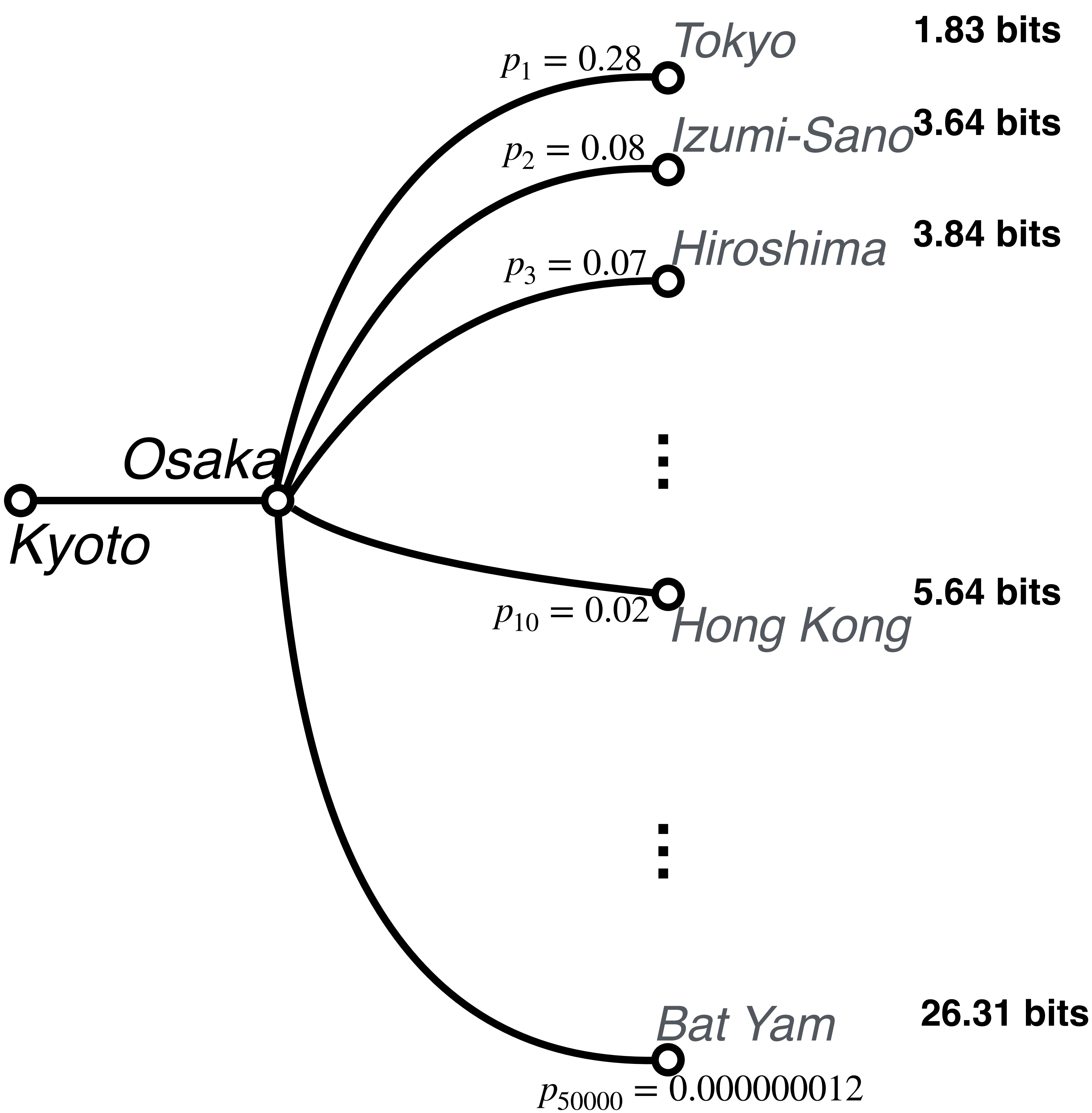
Perplexity: intuition



$$H = 0.28 \times 1.83 + 0.08 \times 3.64 + \ldots$$
$$+(1.2e - 8) * 26.31 = \mathbf{5.26}$$

Booking.com

Perplexity: intuition

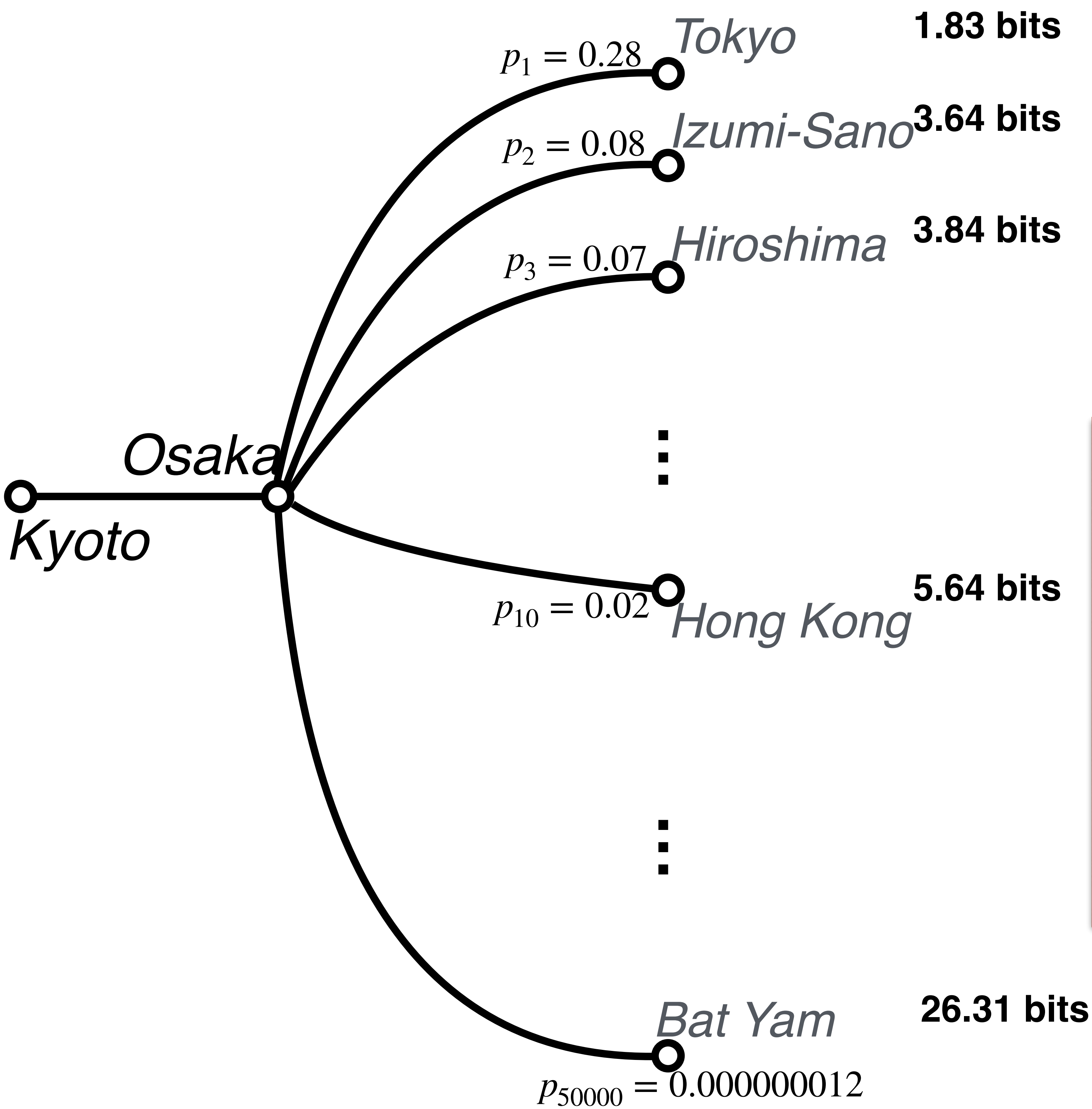


$$H = 0.28 \times 1.83 + 0.08 \times 3.64 + \ldots$$
$$+(1.2e-8) * 26.31 = \mathbf{5.26}$$

$$2^{5.26} = \mathbf{38.4}$$

Booking.com

Perplexity: intuition



$$H = 0.28 \times 1.83 + 0.08 \times 3.64 + \ldots$$
$$+ (1.2e - 8) * 26.31 = \mathbf{5.26}$$

$$2^{5.26} = \mathbf{38.4}$$

Uncertainty of flipping a fair 38.4-sided coin!

Perplexity: intuition

$$\mathbf{PPL} = 2^{H}$$

$$H(\tilde{p}, q) = -\sum_{x} \tilde{p}(x) \log_2 q(x)$$

- How many typical likely next steps can I take?
- Average effective branching factor of the next leg probability tree
- Lower perplexity = more predictable sequences
- Find model which is the least perplexed by the data (likelihood principle)

Booking.com

Perplexity: intuition

$$\mathbf{PPL} = 2^{H}$$

Our loss function!

$$H(\tilde{p}, q) = -\sum_{x} \tilde{p}(x) \log_2 q(x)$$

- How many typical likely next steps can I take?
- Average effective branching factor of the next leg probability tree
- Lower perplexity = more predictable sequences
- Find model which is the least perplexed by the data (likelihood principle)

Implementation

The Gory Details

Booking.com

Data collection

- ~23 million trips from 2017
- Only look at top 50k destinations
- COLLECT() bookings into trips
- Remove consecutive duplicates (stays in different hotels of the same city)
- Randomize and dump into a text file

Booking.com

Data collection

- ~23 million trips from 2017
- Only look at top 50k destinations
- COLLECT() bookings into trips
- Remove consecutive duplicates (stays in different hotels of the same city)
- Randomize and dump into a text file

sketch in Spark

```python
# from pyspark.sql import functions as f
# from pyspark.sql.functions import udf
# from pyspark.sql import Window
# from pyspark.sql.types import ArrayType, StringType
# from functools import reduce

(spark.sql('''SELECT user_id, checkin, checkout, destination_id from Reservations''')
  .join(top_destinations, how='left_semi', on= 'destination_id') # DataFrame with top destinations
  .withColumn('res_rank', f.row_number().over(book_window_checkin)) # Window.partitionBy('user_id').orderBy('checkin')
  .withColumn('next_checkin', f.lead('checkin').over(book_window_checkin))
  .withColumn('days_to_next', f.datediff(f.col('next_checkin'), f.col('checkout')))
  .withColumn('prev_checkout', f.lag('checkout').over(book_w_checkin))
  .withColumn('days_since_last', f.datediff(f.col('checkin'), f.col('prev_checkout')))

  .filter('(days_to_next <= %i) OR (days_since_last <= %i)'%(ALLOWED_GAP, ALLOWED_GAP)) # ALLOWED_GAP=0: perfect trips only
  .orderBy(['user_id','checkin'])
  .withColumn('trip_id_', f.when(f.col('days_since_last')>ALLOWED_GAP,1).otherwise(0))
  .withColumn('trip_id', f.sum('trip_id_').over(book_window_checkin))
  .drop('trip_id_')
  .groupBy('user_id','trip_id')
  .agg(remove_duplicates_udf(f.collect_list('destination_id')).alias('trip'))
  .withColumn('trip_length', f.size('trip'))
  .filter('trip_length > 1')
  .orderBy(f.rand())
 )

# remove_duplicates_udf = f.udf(lambda x:reduce(lambda x,y:((x if x[-1]==y else x+[y]) if isinstance(x, list) else ([x] if x==y else [x,y])), x)
#                              , ArrayType(StringType()))
```

Data collection

- ~23 million trips from 2017
- Only look at top 50k destinations
- COLLECT() bookings into trips
- Remove consecutive duplicates (stays in different hotels of the same city)
- Randomize and dump into a text file

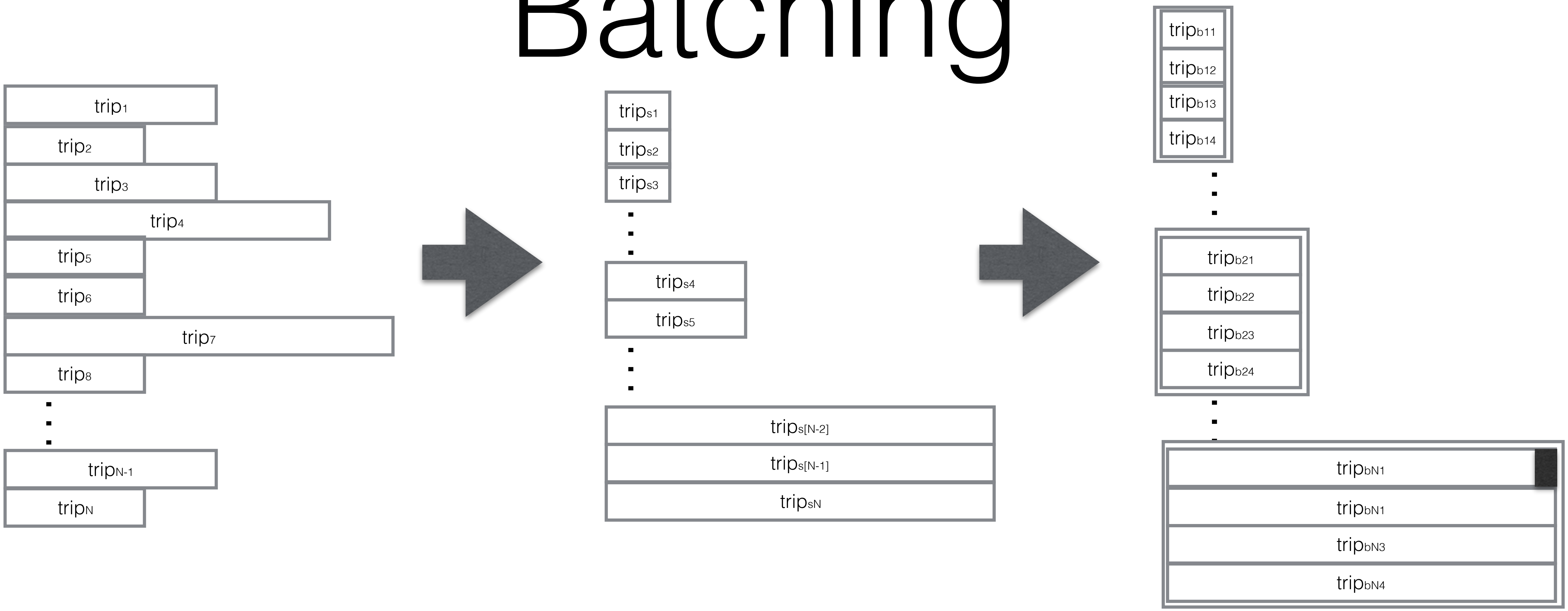
```
# from pyspark.sql import functions
# from pyspark.sql.functions import
# from pyspark.sql import Window
# from pyspark.sql.types import Arr
# from functools import reduce

(spark.sql('''SELECT user_id, check
  .join(top_destinations, how='left
  .withColumn('res_rank', f.row_num                                          checkin')
  .withColumn('next_checkin', f.lea
  .withColumn('days_to_next', f.dat
  .withColumn('prev_checkout', f.la
  .withColumn('days_since_last', f.

  .filter('(days_to_next <= %i) OR (days_since_last <= %i)' %(ALLOWED_GAP, ALLOWED_GAP)) # ALLOWED_GAP=0: perfect trips only
  .orderBy(['user_id','checkin'])
  .withColumn('trip_id_', f.when(f.col('days_since_last')>ALLOWED_GAP,1).otherwise(0))
  .withColumn('trip_id', f.sum('trip_id_').over(book_window_checkin))
  .drop('trip_id_')
  .groupBy('user_id','trip_id')
  .agg(remove_duplicates_udf(f.collect_list('destination_id')).alias('trip'))
  .withColumn('trip_length', f.size('trip'))
  .filter('trip_length > 1')
  .orderBy(f.rand())
 )

# remove_duplicates_udf = f.udf(lambda x:reduce(lambda x,y:((x if x[-1]==y else x+[y]) if isinstance(x, list) else ([x] if x==y else [x,y])), x)
#                               , ArrayType(StringType()))
```

Approximately:
23 million trips

90% <5 legs
10% 5+ legs

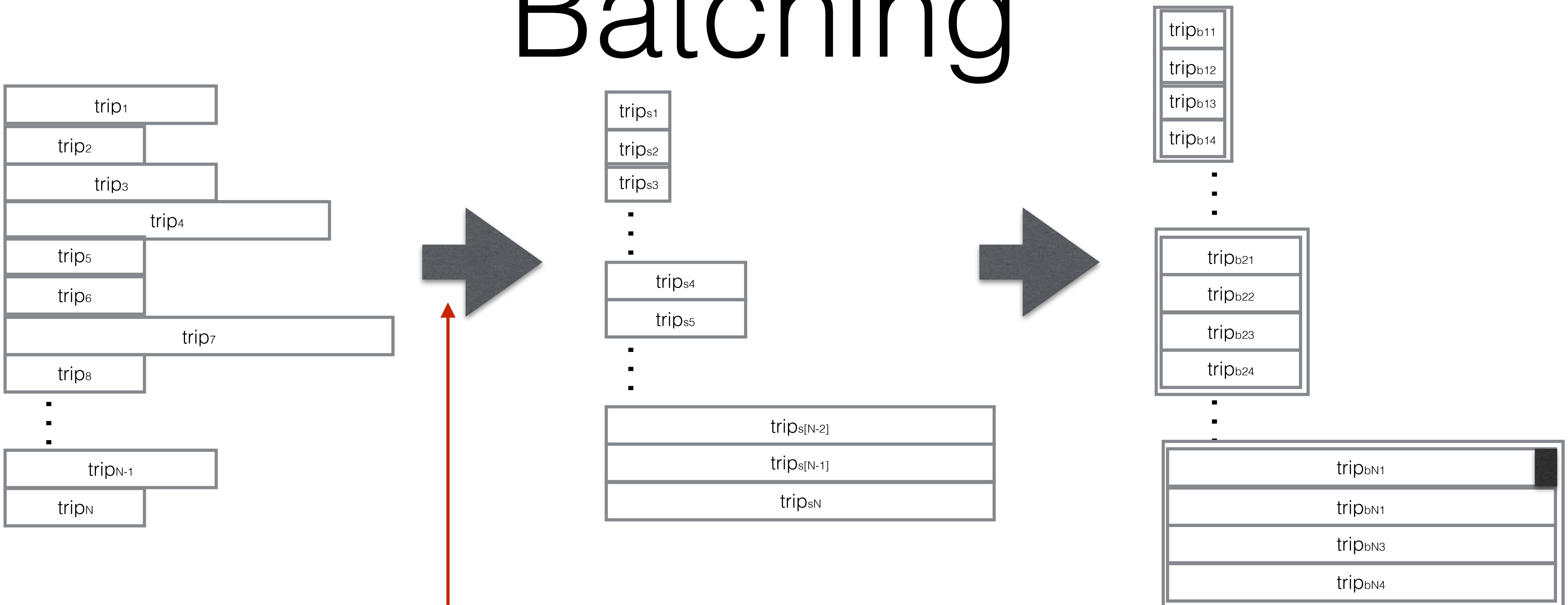
Booking.com

Batching



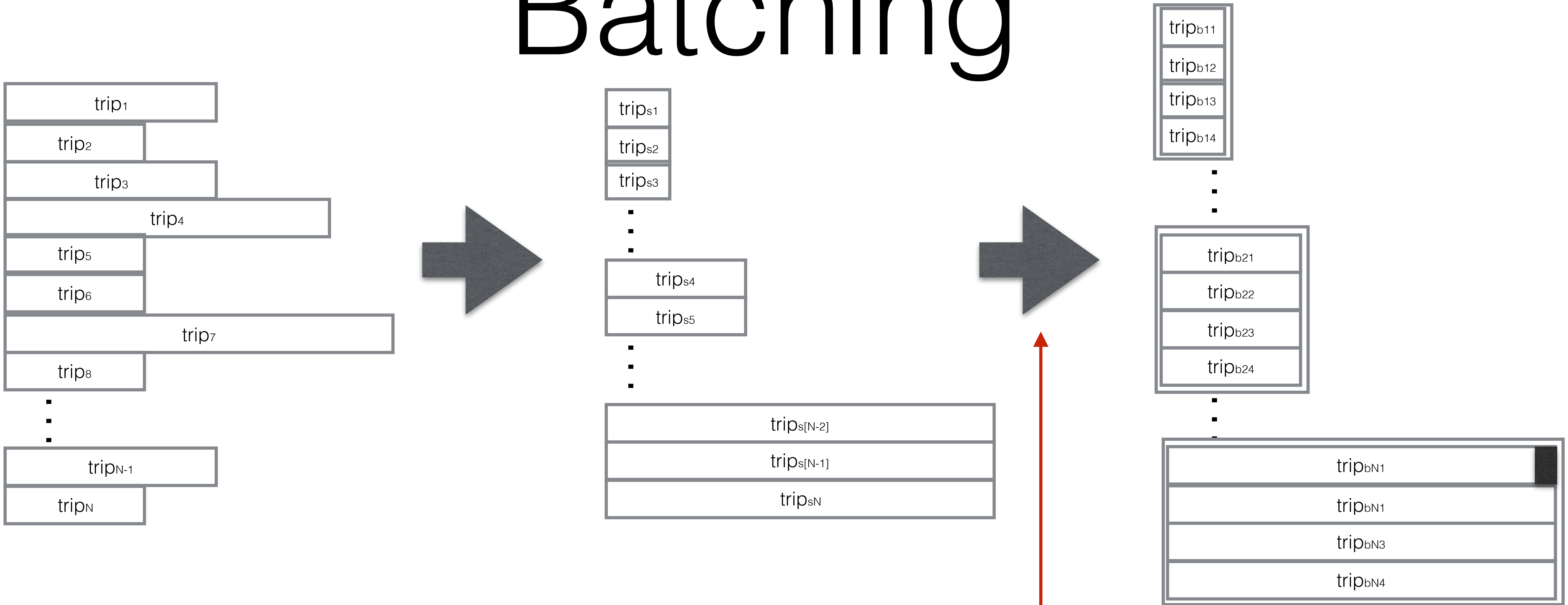
Booking.com

Batching




```
with open(DATA_FILE) as f:
    for line in f:
        lines.append([word2id.get(_, word2id['<UNK>']) for _ in (line.split() + ['<EOS>'])])
        length = len(lines[-1])
        if length > MAX_SEQ_LEN:
            lines[-1] = lines[-1][:MAX_SEQ_LEN]
            length = MAX_SEQ_LEN
        lens.append(length)
        order = np.argsort(lens)
lines_sorted = sorted(lines, key=len)
```

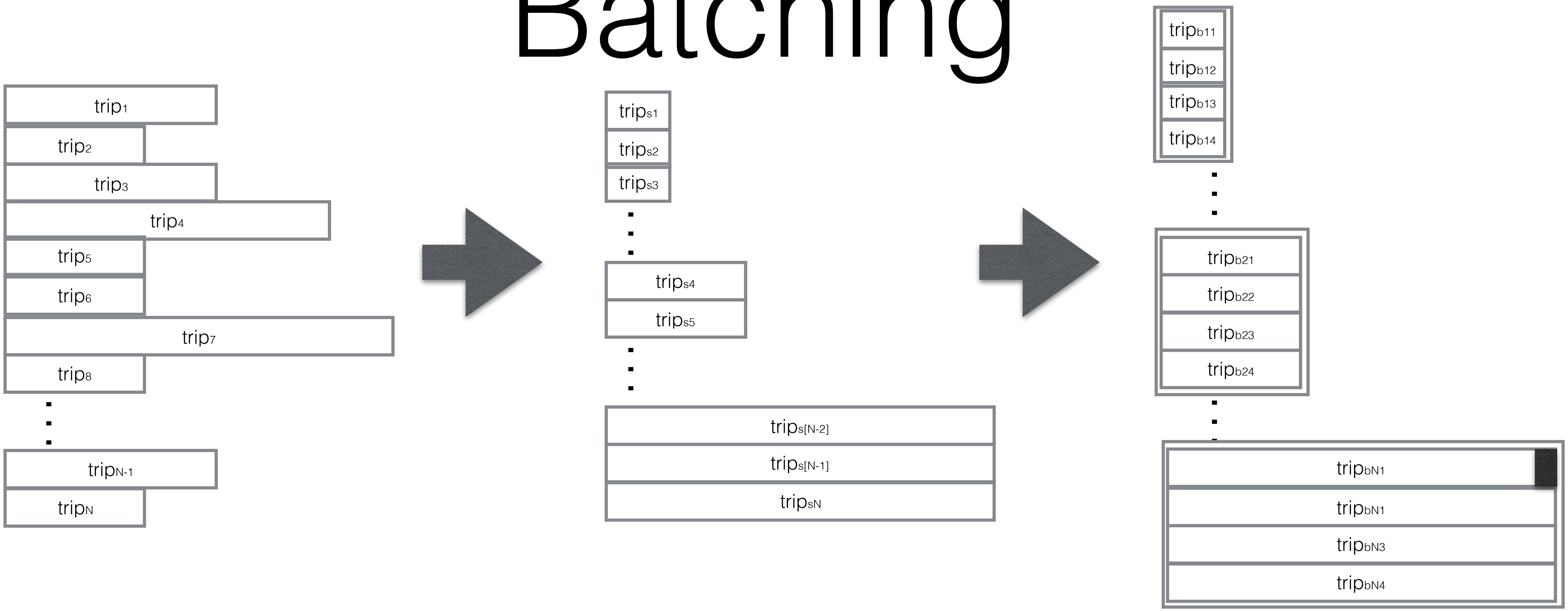
Batching

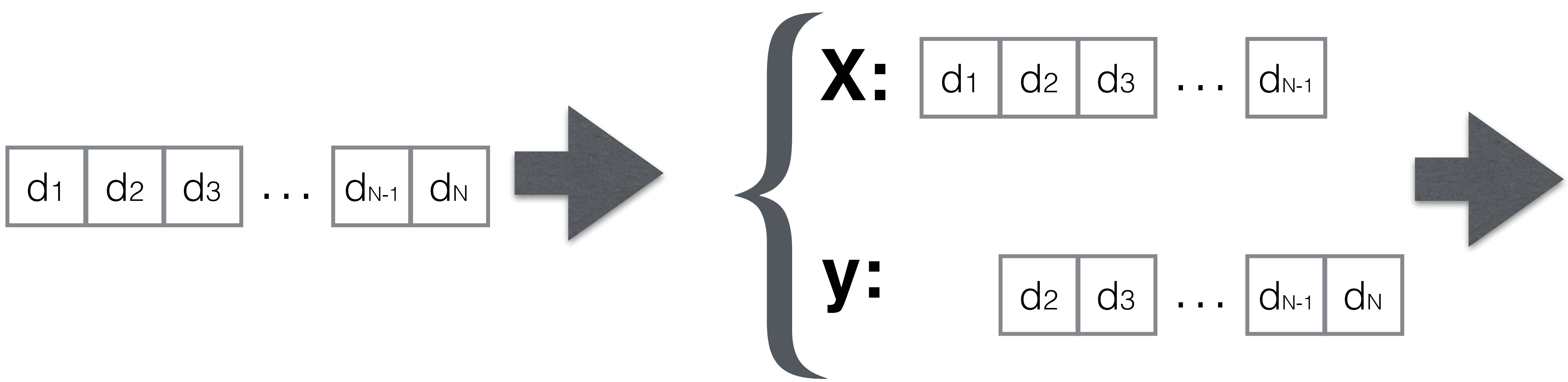



```python
# from torch.autograd import Variable
# if CUDA_ENABLED:
#     from torch.cuda import LongTensor
# else:
#     from torch import LongTensor

# GETTING BATCH i
lines = lines_sorted[i:i+batch_size]
max_len = np.max([len(_) for _ in lines])
data = LongTensor([pad_sequence(_ , max_len) for _ in lines])
source, target = (Variable(data.narrow(1, 0, data.size(1)-1).t())
                  ,Variable(data.narrow(1, 1, data.size(1)-1).t().contiguous().view(-1))
                 )
```

Booking.com

Batching





	x	y
1	d_1	d_2
2	d_2	d_3
	. . .	
N-1	d_{N-1}	d_N

Booking.com

The model

```
class RNNLM(nn.Module):
        def __init__(self, vocab_size, emb_dim=500, rnn_dim=500, n_layers=2, dropout=0.3, rnn_dropout=0.3, tie_weights=False):
                super(RNNLM, self).__init__()
                self.drop = nn.Dropout(dropout)
                self.embedding = nn.Embedding(vocab_size, emb_dim)
                self.rnn = nn.RNN(input_size=emb_dim, hidden_size=rnn_dim, num_layers=n_layers, dropout=rnn_dropout)
                self.dense = nn.Linear(rnn_dim, vocab_size)

                if tie_weights:
                        self.dense.weight = self.embedding.weight
                self.vocab_size = vocab_size
                self.emb_dim = emb_dim
                self.rnn_dim = rnn_dim
                self.n_layers = n_layers
                self.dropout = dropout
                self.rnn_dropout = rnn_dropout
                self.tie_weights = tie_weights

        def forward(self, input, h0=None):
                out, hidden = self.rnn(self.drop(self.embedding(input)), h0)
                out = self.dense(self.drop(out.view(out.size(0)*out.size(1), out.size(2))))
                return out, hidden
```

The model

```python
class RNNLM(nn.Module):
        def __init__(self, vocab_size, emb_dim=500, rnn_dim=500, n_layers=2, dropout=0.3, rnn_dropout=0.3, tie_weights=False):
                super(RNNLM, self).__init__()
                self.drop = nn.Dropout(dropout)
                self.embedding = nn.Embedding(vocab_size, emb_dim)
                self.rnn = nn.RNN(input_size=emb_dim, hidden_size=rnn_dim, num_layers=n_layers, dropout=rnn_dropout)
                self.dense = nn.Linear(rnn_dim, vocab_size)

                if tie_weights:
                        self.dense.weight = self.embedding.weight
                self.vocab_size = vocab_size
                self.emb_dim = emb_dim
                self.rnn_dim = rnn_dim
                self.n_layers = n_layers
                self.dropout = dropout
                self.rnn_dropout = rnn_dropout
                self.tie_weights = tie_weights

        def forward(self, input, h0=None):
                out, hidden = self.rnn(self.drop(self.embedding(input)), h0)
                out = self.dense(self.drop(out.view(out.size(0)*out.size(1), out.size(2))))
                return out, hidden
```

The model

```python
class RNNLM(nn.Module):
        def __init__(self, vocab_size, emb_dim=500, rnn_dim=500, n_layers=2, dropout=0.3, rnn_dropout=0.3, tie_weights=False):
                super(RNNLM, self).__init__()
                self.drop = nn.Dropout(dropout)
                self.embedding = nn.Embedding(vocab_size, emb_dim)
                self.rnn = nn.RNN(input_size=emb_dim, hidden_size=rnn_dim, num_layers=n_layers, dropout=rnn_dropout)
                self.dense = nn.Linear(rnn_dim, vocab_size)

                if tie_weights:
                        self.dense.weight = self.embedding.weight
                self.vocab_size = vocab_size
                self.emb_dim = emb_dim
                self.rnn_dim = rnn_dim
                self.n_layers = n_layers
                self.dropout = dropout
                self.rnn_dropout = rnn_dropout
                self.tie_weights = tie_weights

        def forward(self, input, h0=None):
                out, hidden = self.rnn(self.drop(self.embedding(input)), h0)
                out = self.dense(self.drop(out.view(out.size(0)*out.size(1), out.size(2))))
                return out, hidden
```

$$\Theta_d = \Theta_y$$

Published as a conference paper at ICLR 2017

TYING WORD VECTORS AND WORD CLASSIFIERS: A LOSS FRAMEWORK FOR LANGUAGE MODELING

Hakan Inan, Khashayar Khosravi
Stanford University
Stanford, CA, USA
{inanh,khosravi}@stanford.edu

Richard Socher
Salesforce Research
Palo Alto, CA, USA
rsocher@salesforce.com

ABSTRACT

Recurrent neural networks have been very successful at predicting sequences of words in tasks such as language modeling. However, all such models are based

Booking.com

Training

- Get fresh batch (inputs, outputs)
- Get predictions
- Calculate loss on the predictions
- Backpropagate
- Update model parameters

Training

- Get fresh batch (inputs, outputs)
- Get predictions
- Calculate loss on the predictions
- Backpropagate
- Update model parameters

pytorch sketch

```
for epoch in range(1, TOTAL_EPOCHS):
    for i, batch_n in enumerate(shuffled_batches):
        data_in, targets = get_batch(batch_n, batch_size)
        model.zero_grad()
        output = model(data_in)
        loss = criterion(output.view(-1, n_destinations), targets)
        loss.backward()
        optimizer.step()
        losses.append(loss.data[0])
```

Training demo

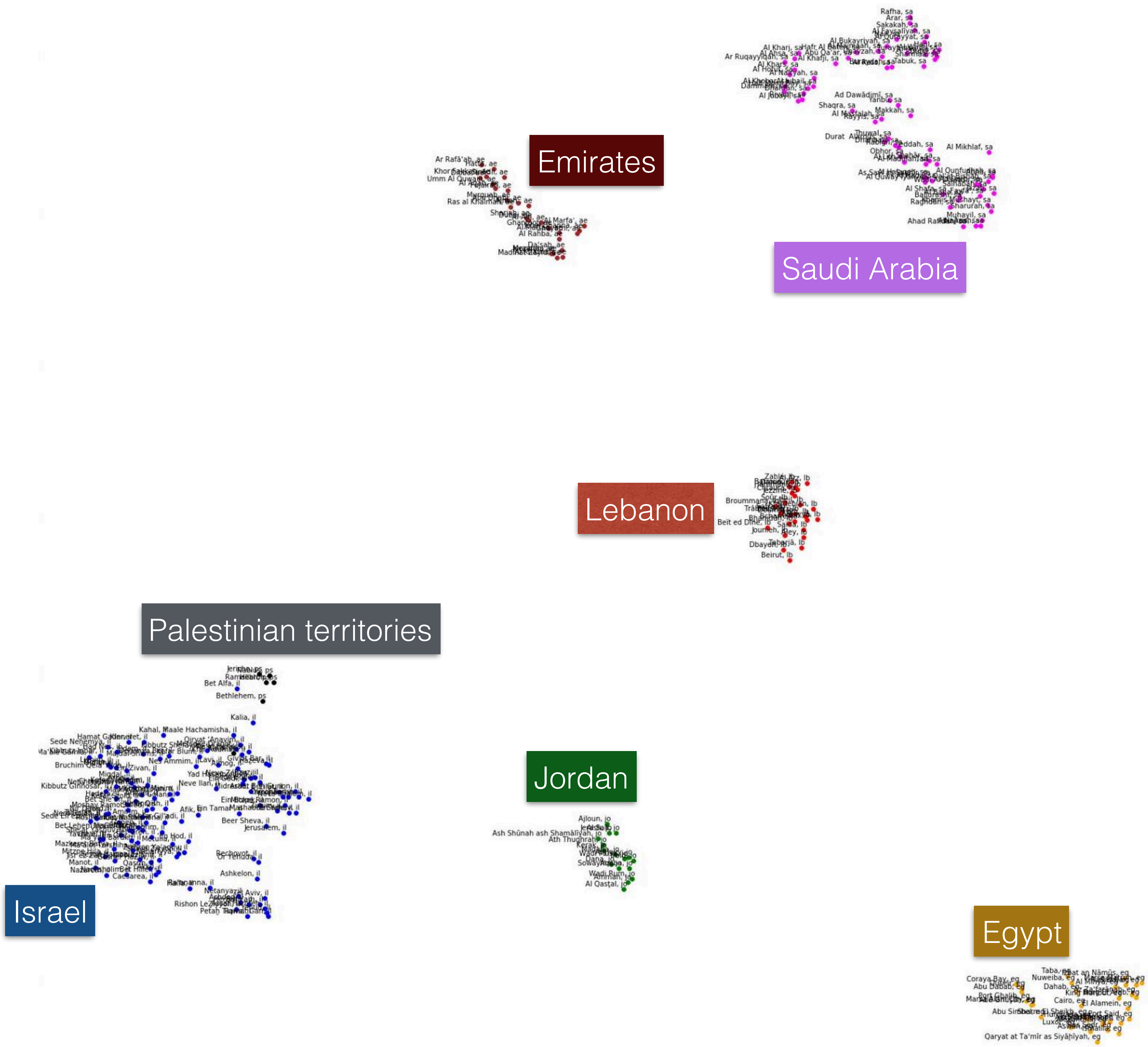
http://localhost:8888/notebooks/jerusalem_demo.ipynb

Training demo

http://localhost:8888/notebooks/jerusalem_demo.ipynb

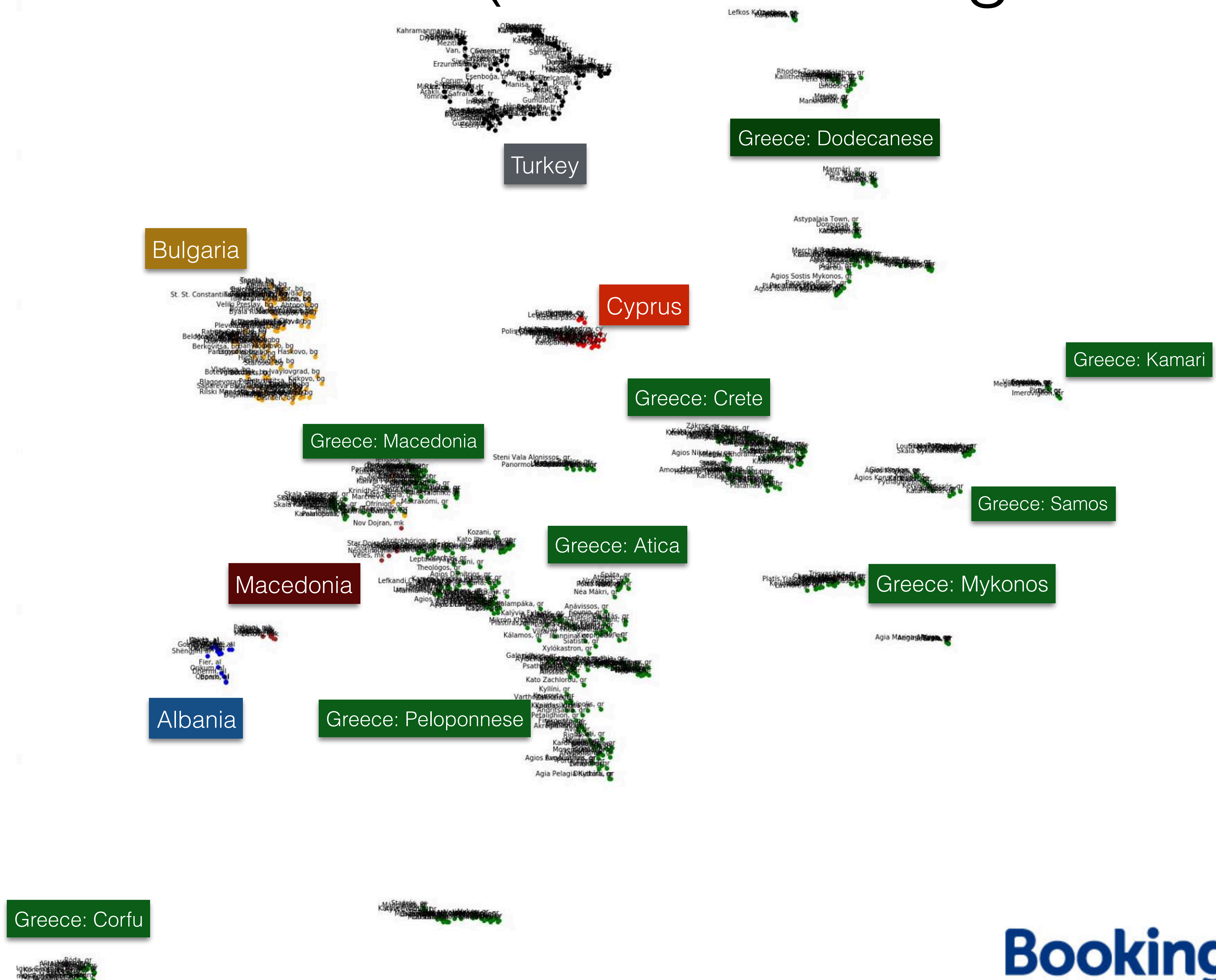
Similar model open source (from official pytorch examples):
https://github.com/pytorch/examples/tree/master/word_language_model

Destination embeddings learned by the RNN model (middle east)



Booking.com

Destination embeddings learned by the RNN model (Greece + neighbors)



Booking.com

Handling unknown input

- Maintain destination mapping (MySQL table)

- Map all known destinations to the 50k most popular ones based on geography

- use kd-Tree for fast nearest neighbor search

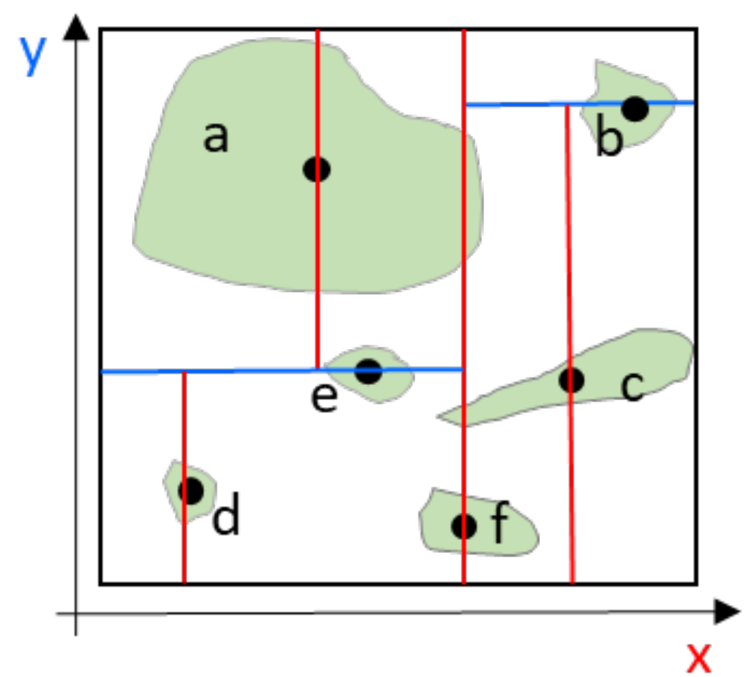
Booking.com

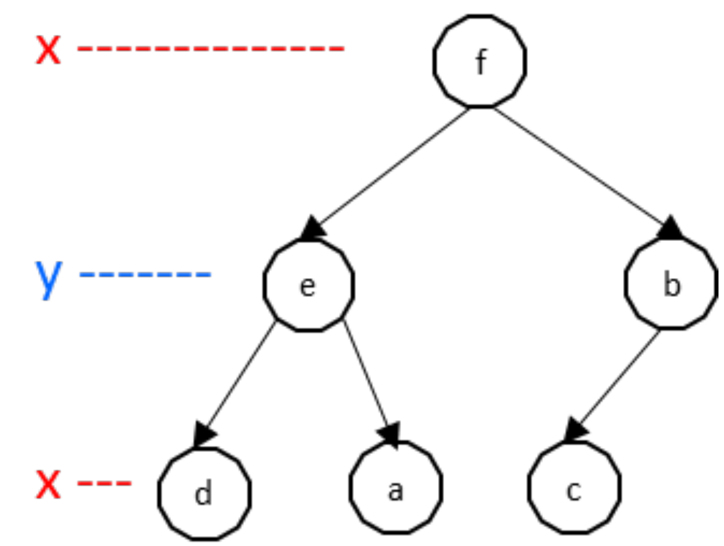
Handling unknown input

- Maintain destination mapping (MySQL table)

- Map all known destinations to the 50k most popular ones based on geography

- use kd-Tree for fast nearest neighbor search

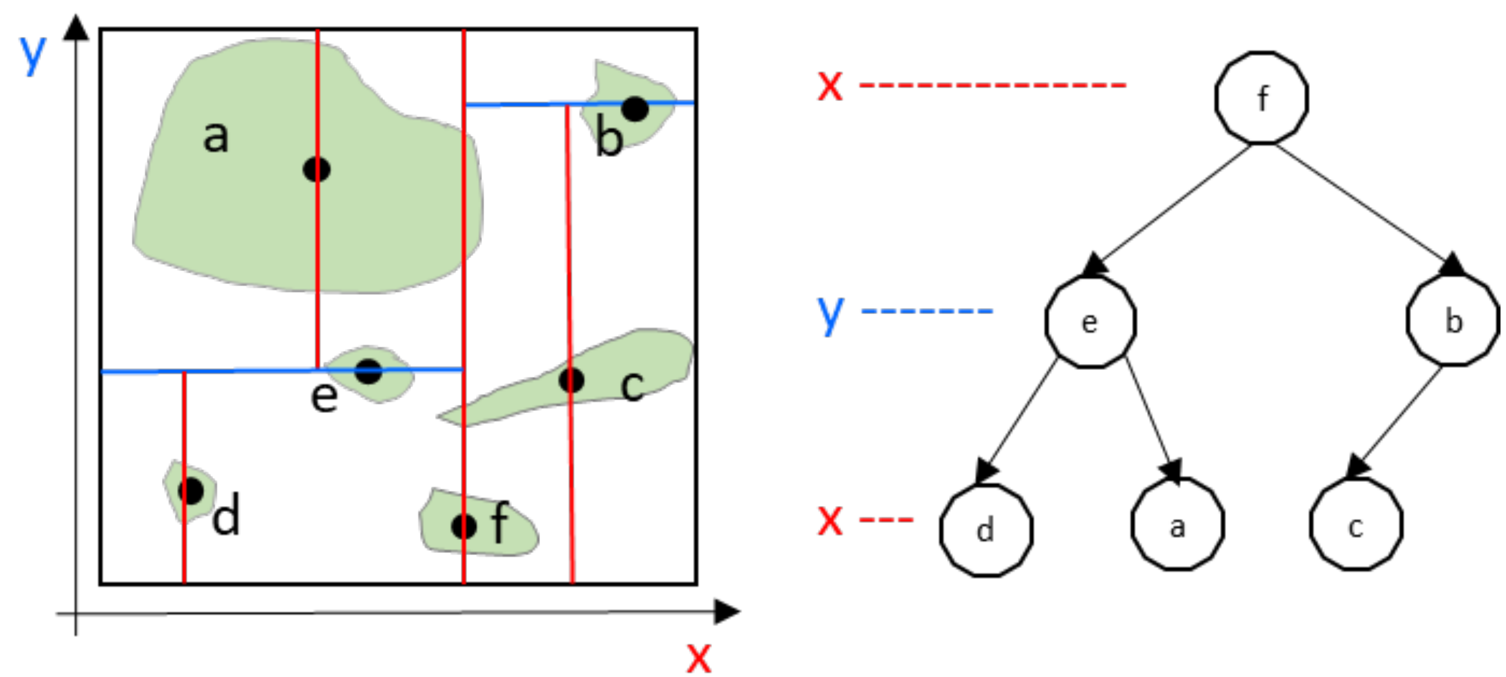




Booking.com

Handling unknown input

- Maintain destination mapping (MySQL table)
- Map all known destinations to the 50k most popular ones based on geography
- use kd-Tree for fast nearest neighbor search

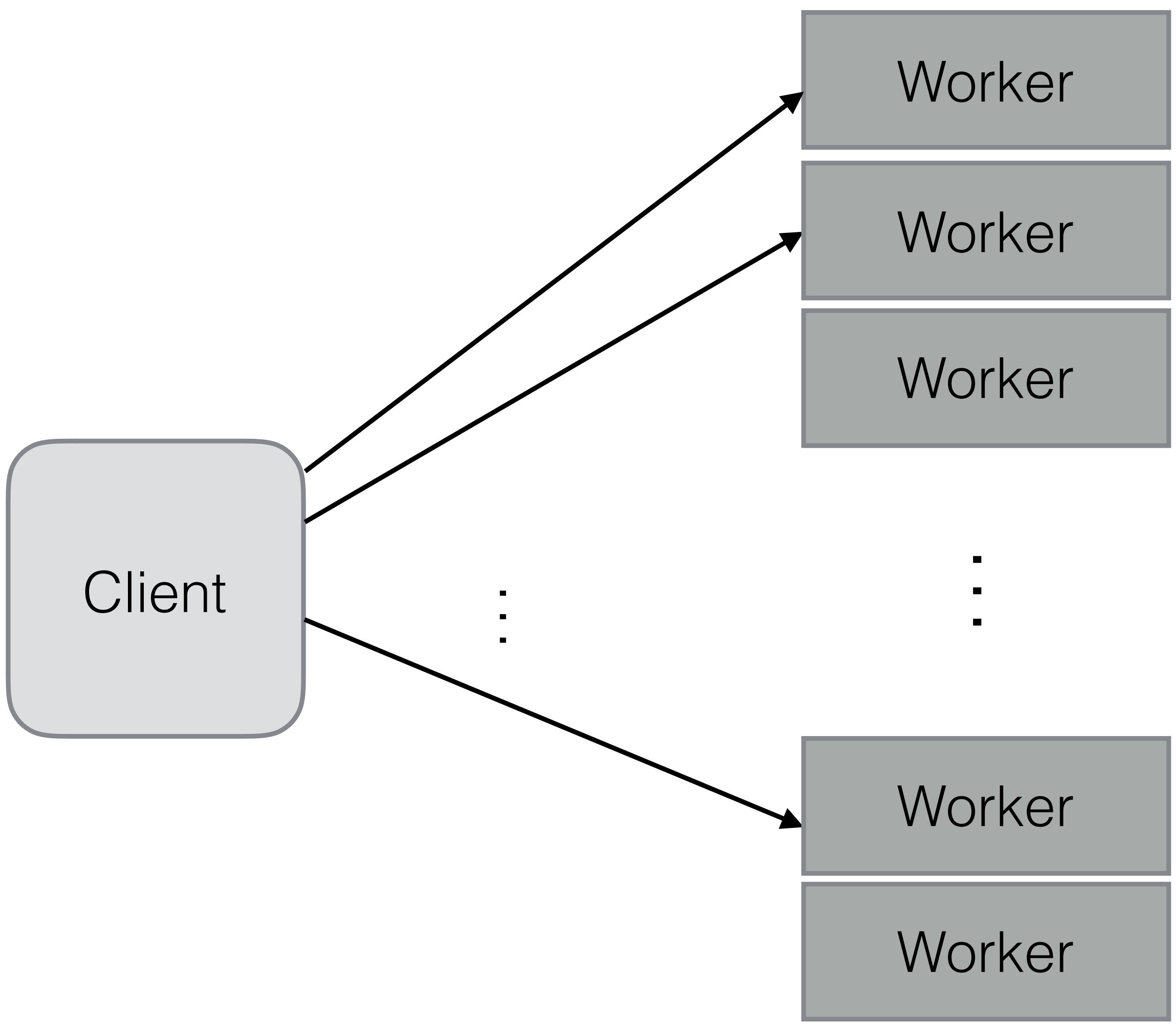


Python sketch

```
from scipy.spatial import cKDTree
dest_coords = (destionation_data
               [['destination_id','latitude','longitude']]
               .values
               )
coords = [[lat, lon] for destination_id, lat, lon in dest_coords]
kdtree = cKDTree(coords)

all_known_destinations['closest'] = (
    all_known_destinations
    .apply(lambda x: dest_coords[kdtree.query([x.latitude, x.longitude]), 0]
           , axis=1)
)
```

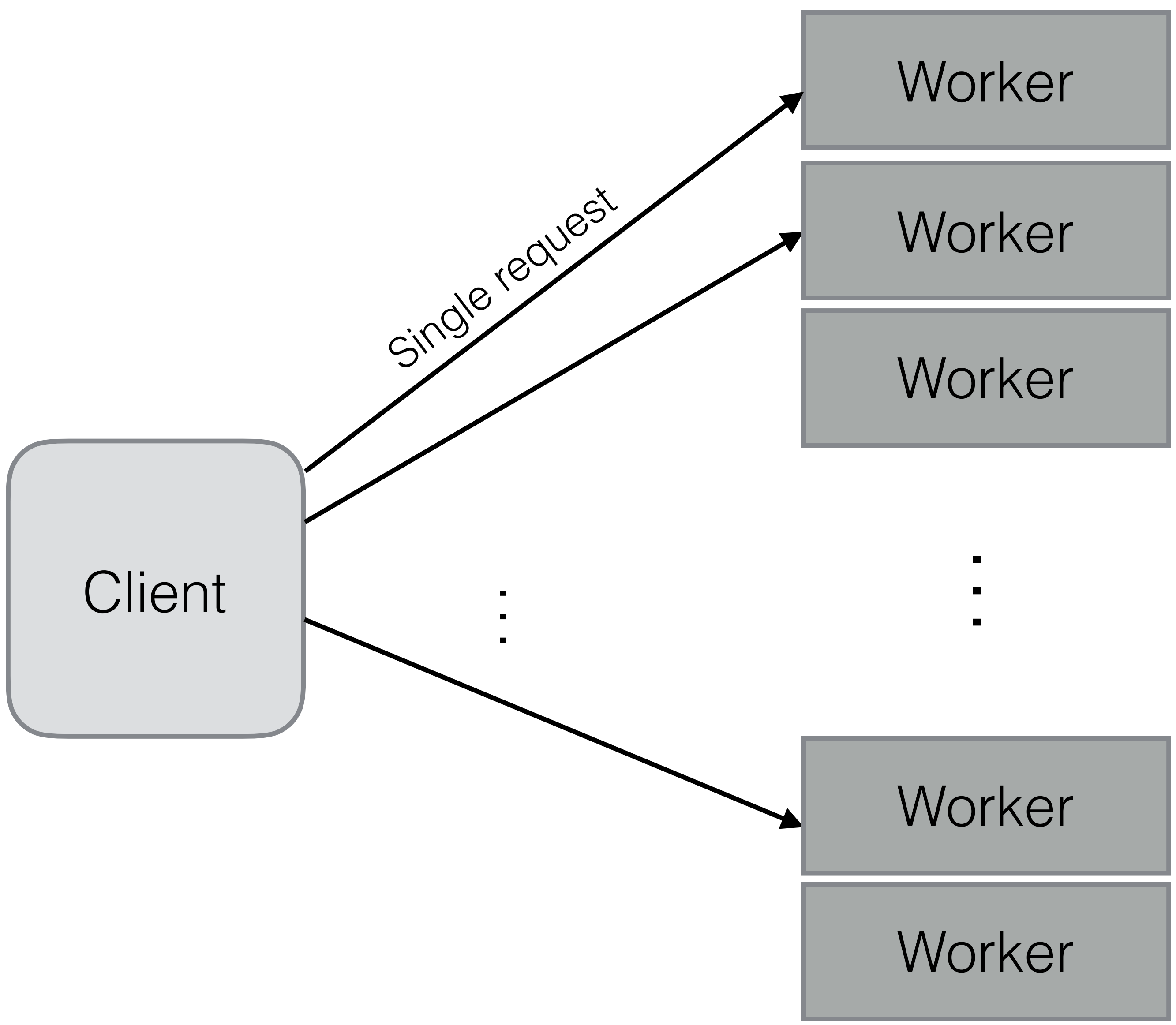
Putting it in production

General architecture



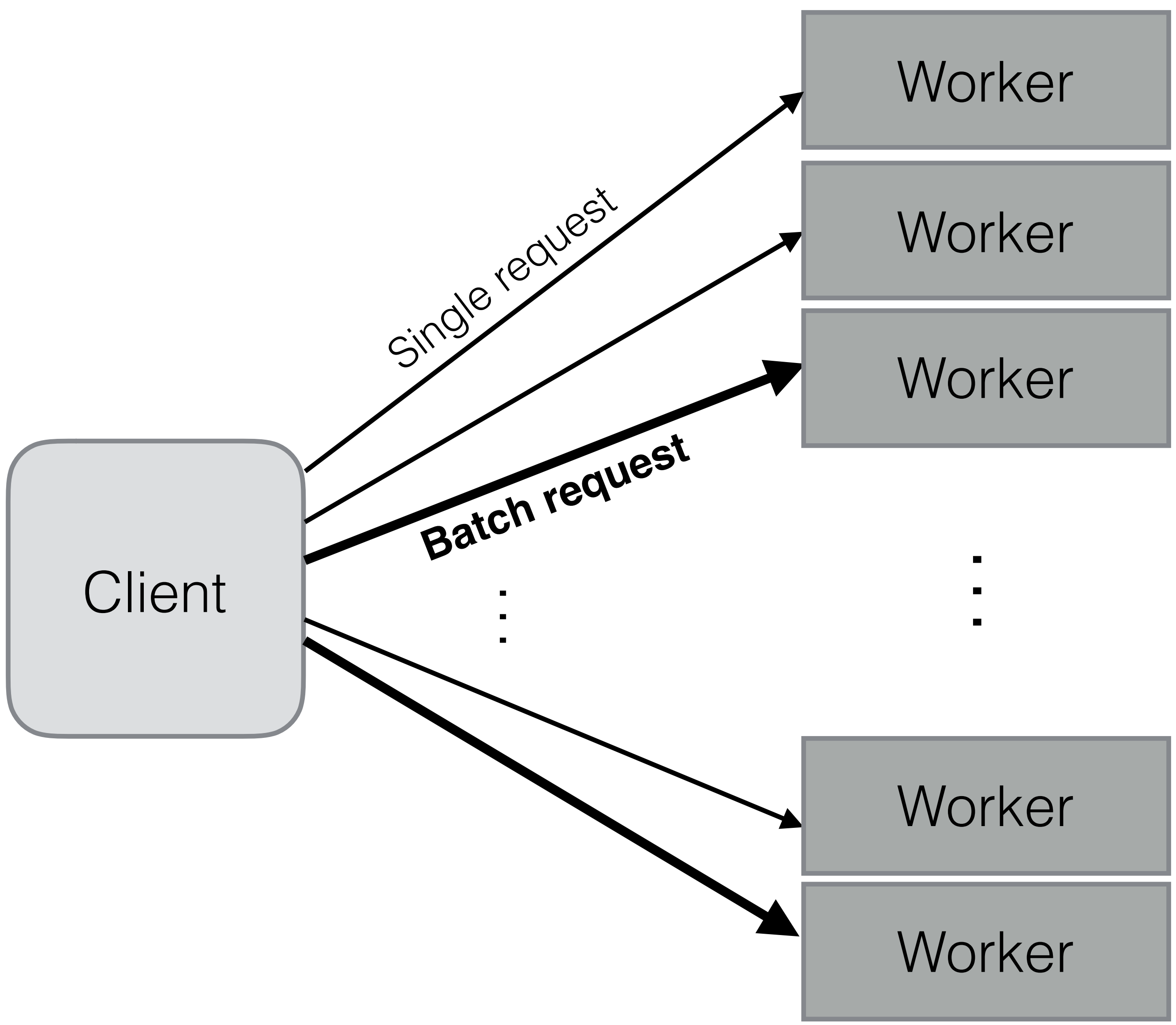
Booking.com

General architecture



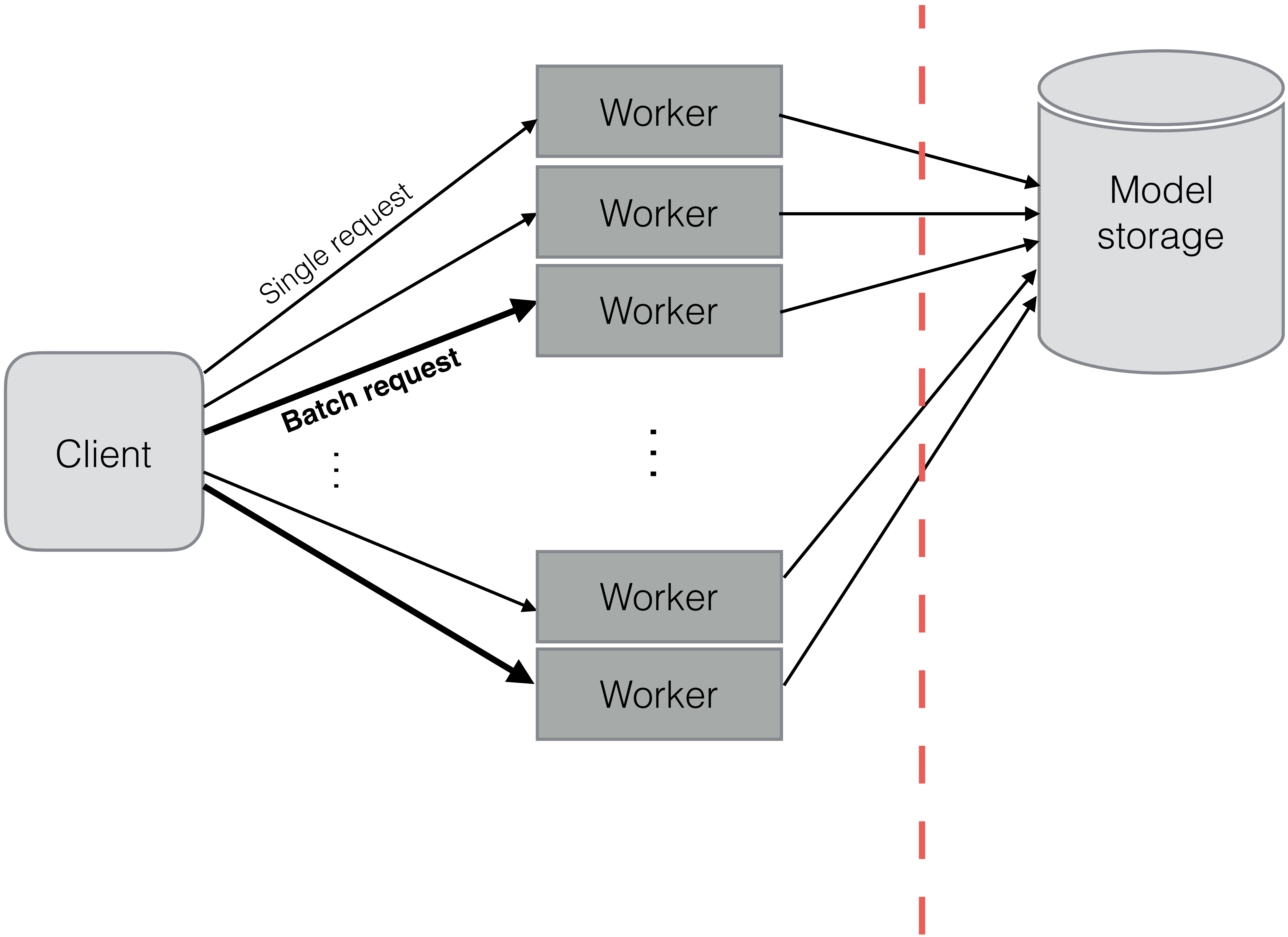
Booking.com

General architecture



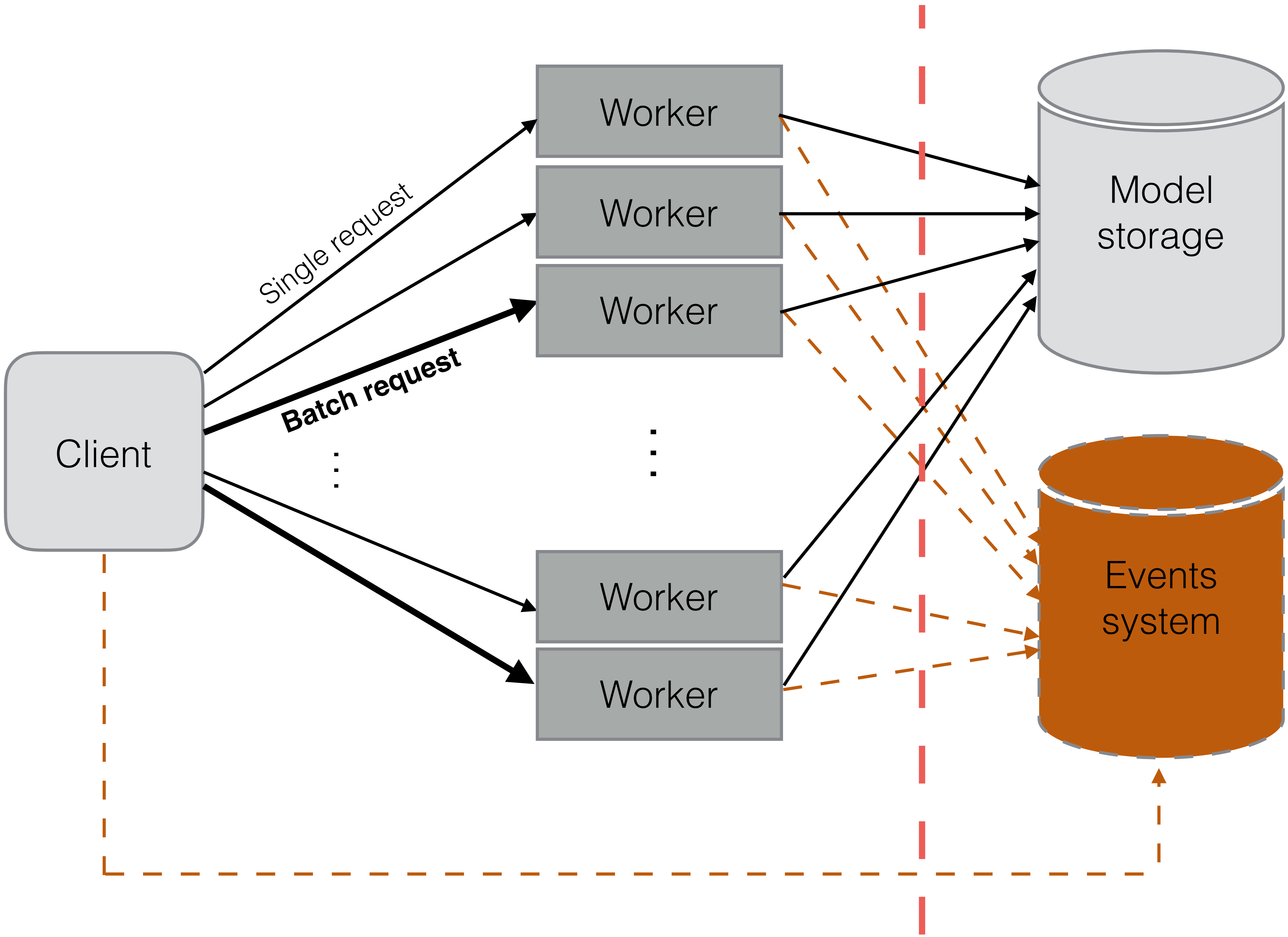
Booking.com

General architecture



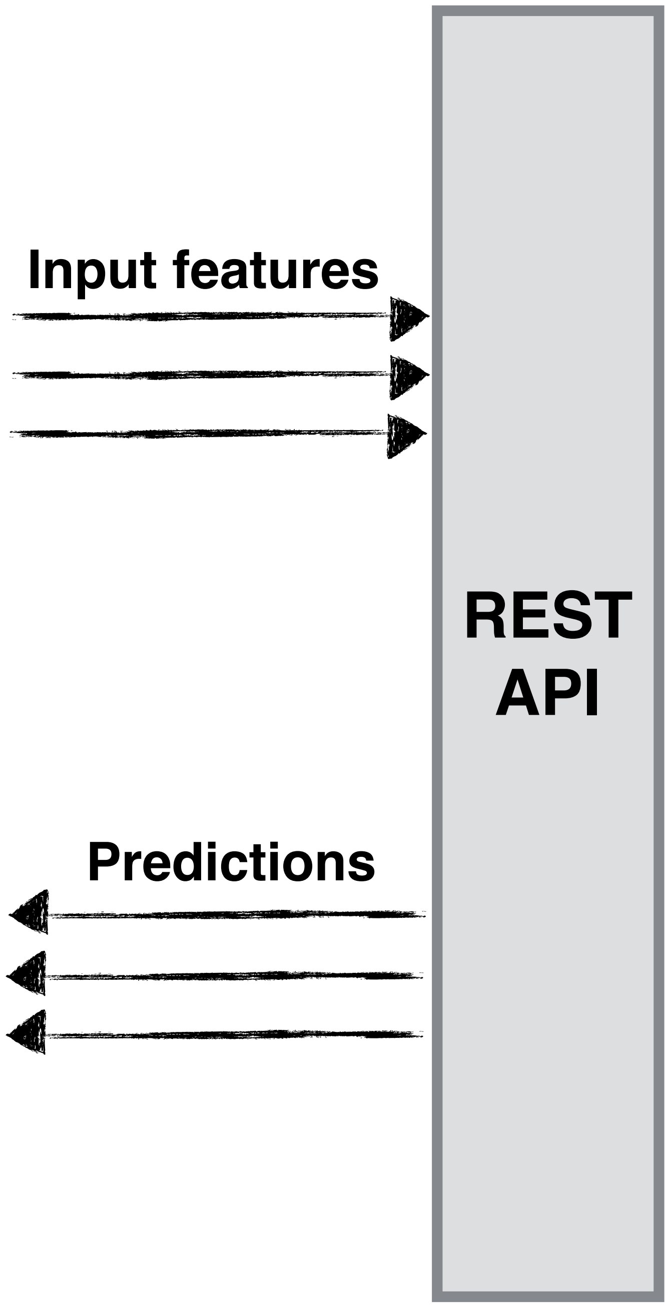
Booking.com

General architecture

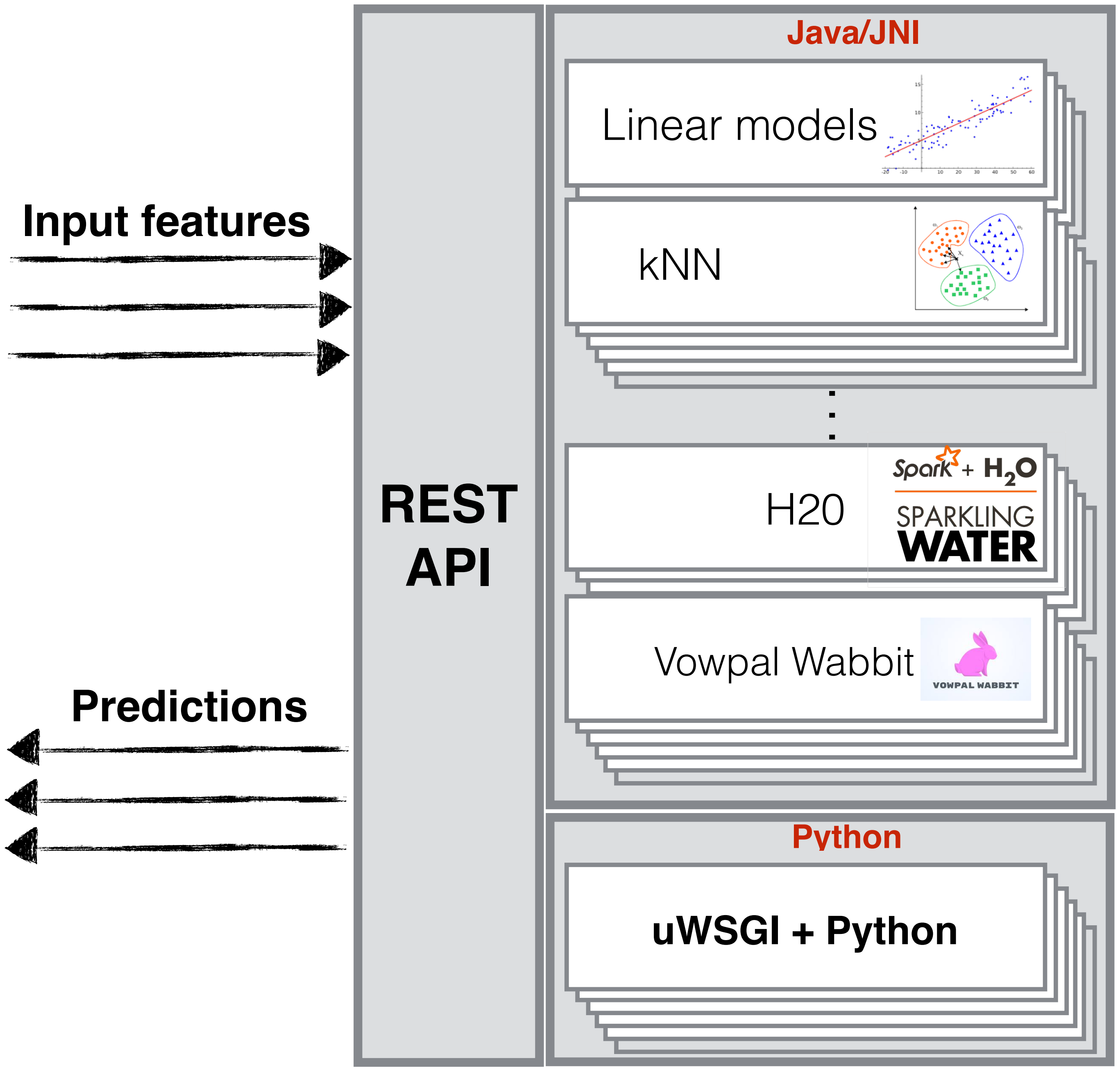


Booking.com

Worker structure

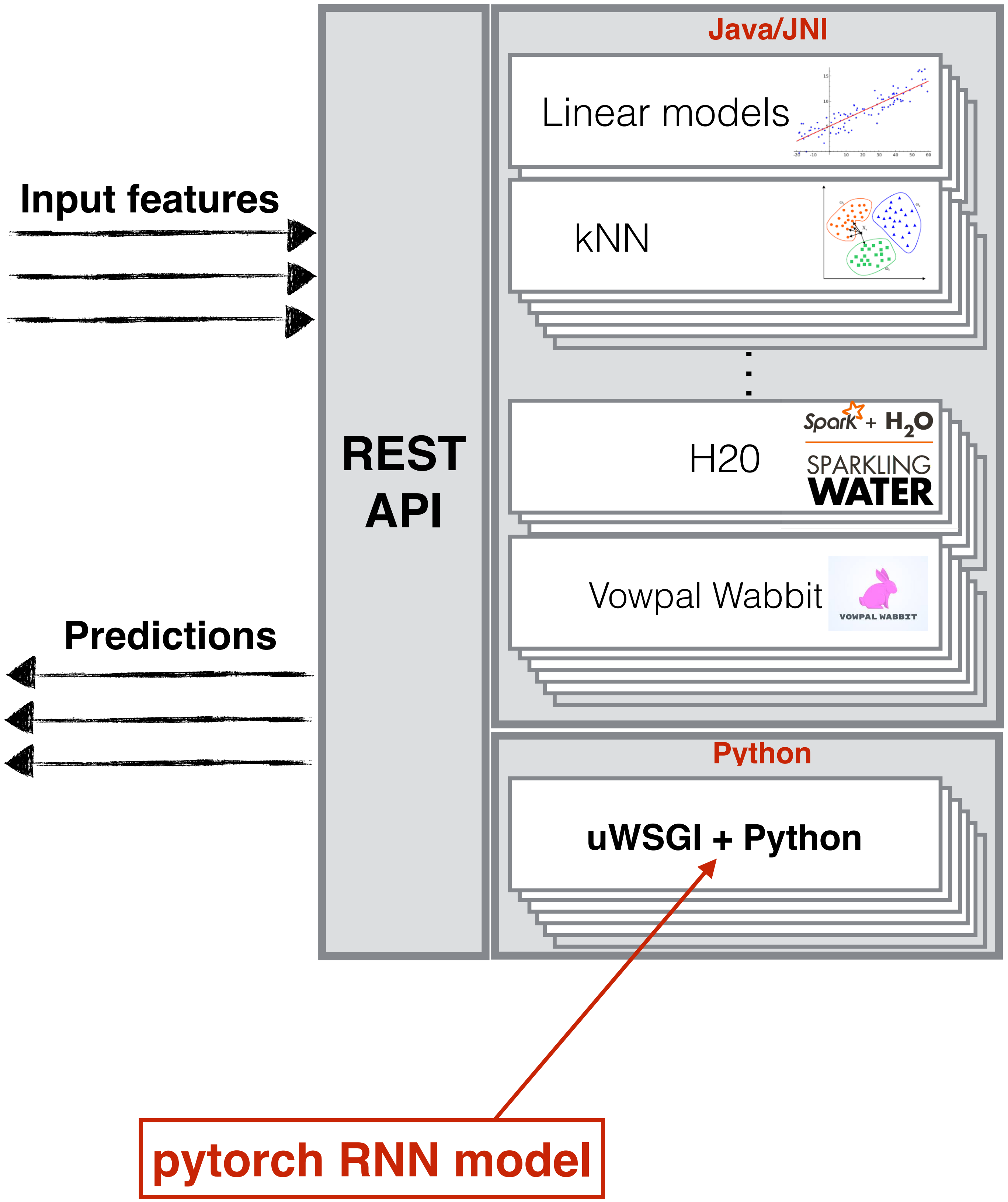


Worker structure



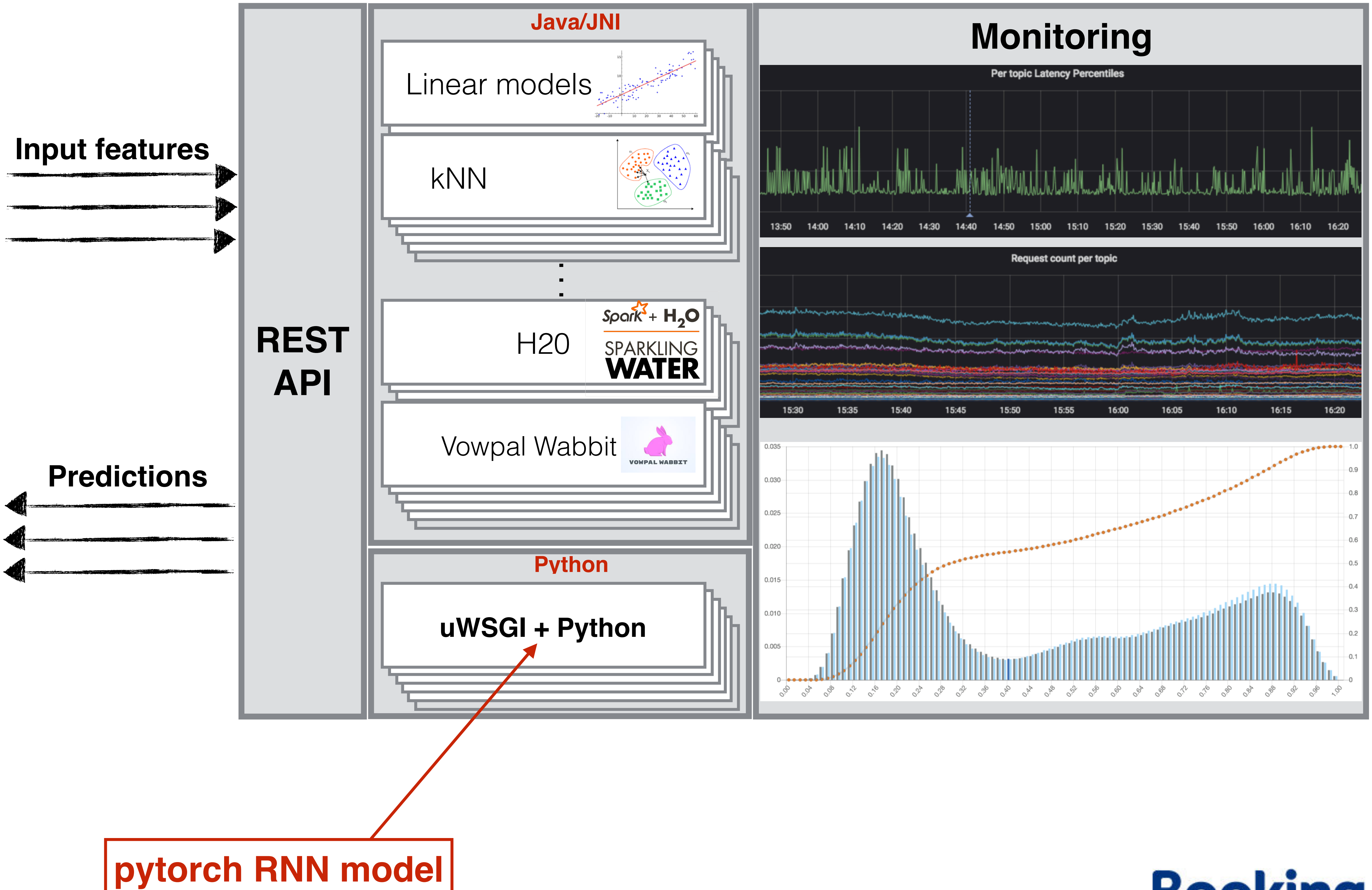
Booking.com

Worker structure



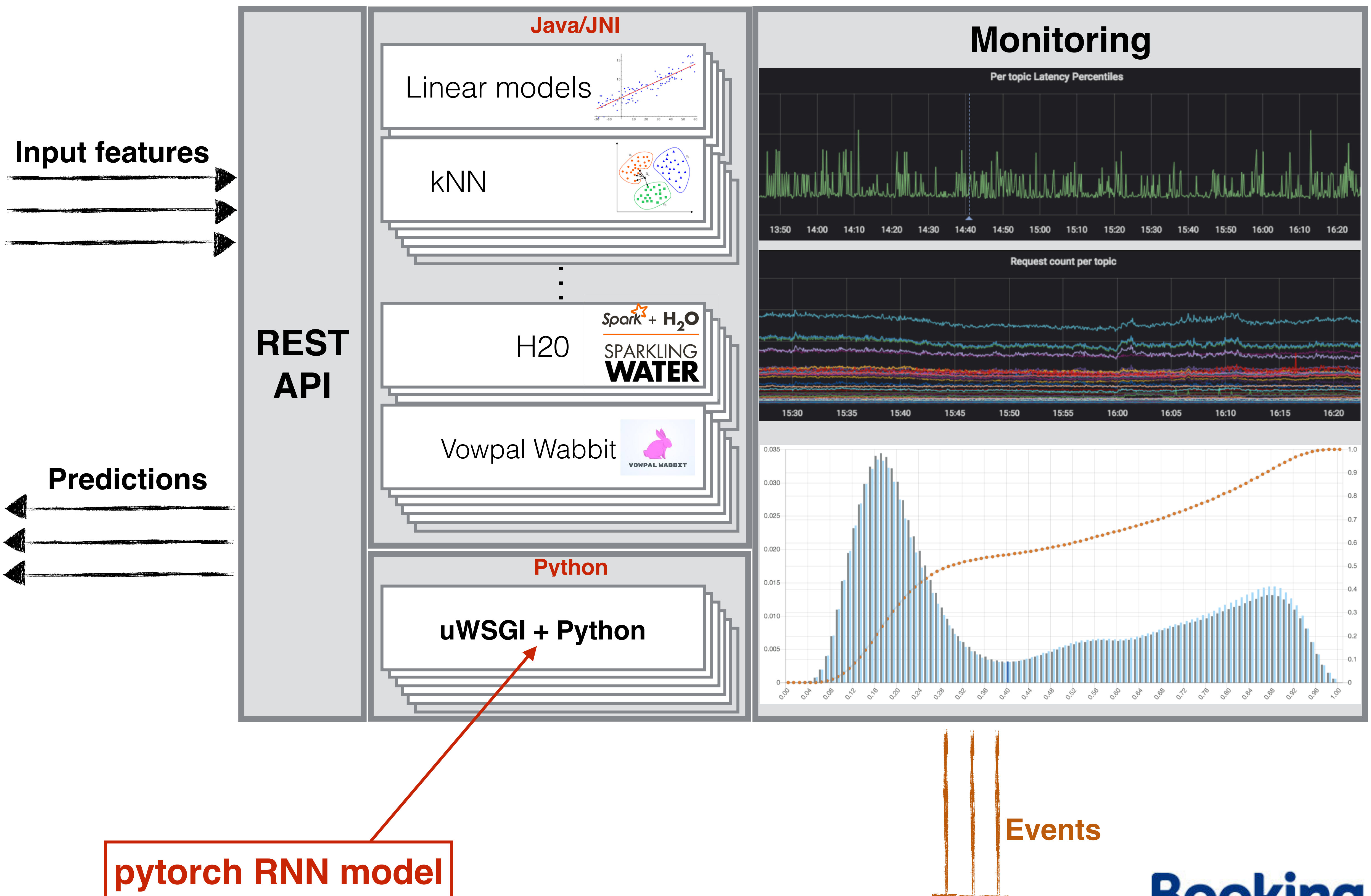
Booking.com

Worker structure

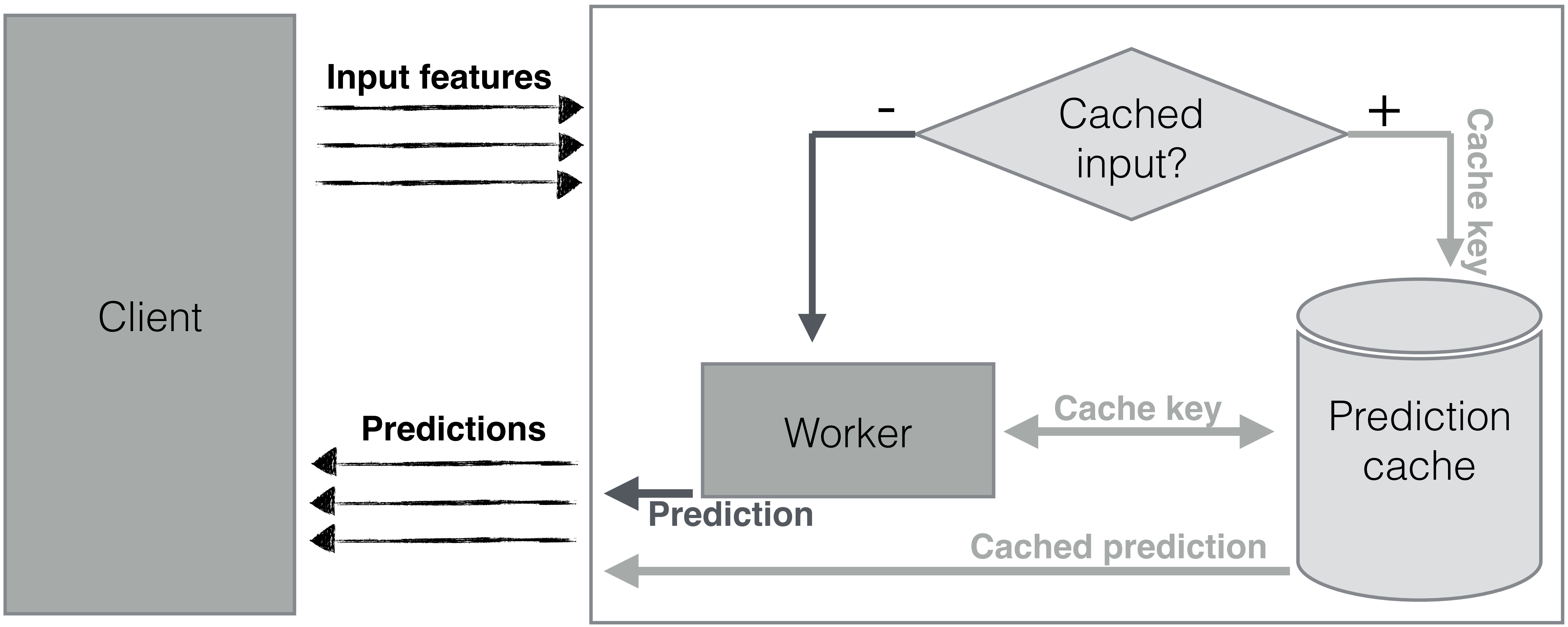


Booking.com

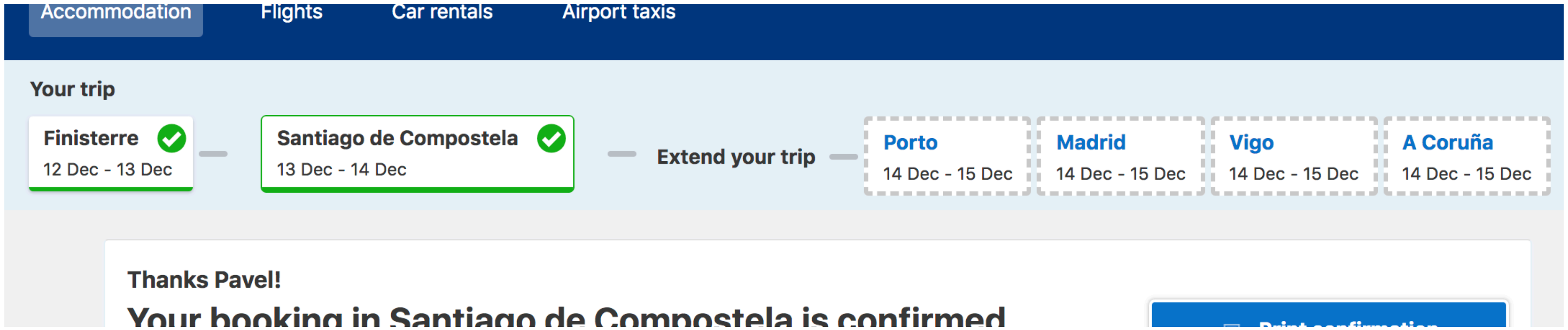
Worker structure



Caching common inputs



Some use cases



- Website: trip extension bar
- Email Marketing: personalized destination suggestions
- Mobile notifications [WiP]

Booking.com

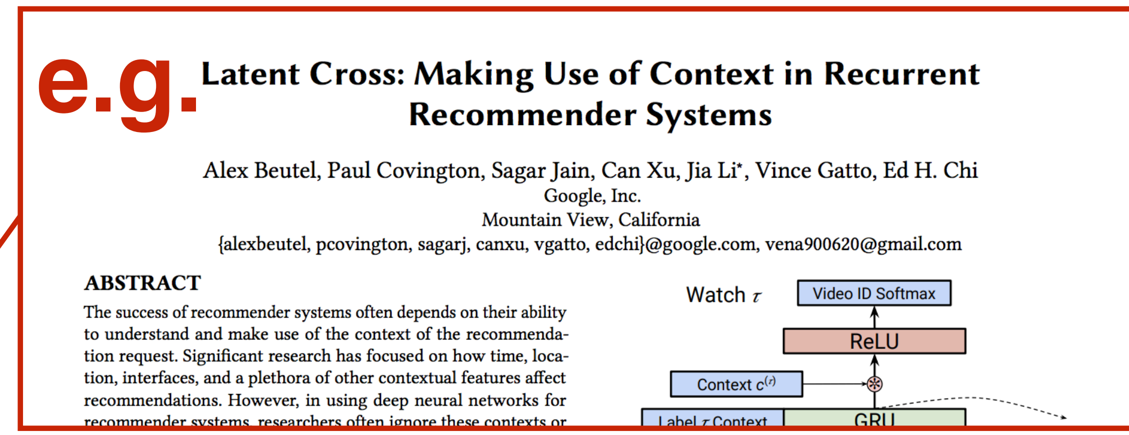
Beyond single-step predictions

- Beam decoding for multiple steps
- Handling gaps properly
  - Sequence scoring
  - Direct modeling
- Additional user/trip/city-level features
- Hierarchical encoders to model multiple trips

Booking.com

Beyond single-step predictions

- Beam decoding for multiple steps
- Handling gaps properly
  - Sequence scoring
  - Direct modeling
- Additional user/trip/city-level features
- Hierarchical encoders to model multiple trips


e.g. **Latent Cross: Making Use of Context in Recurrent Recommender Systems**

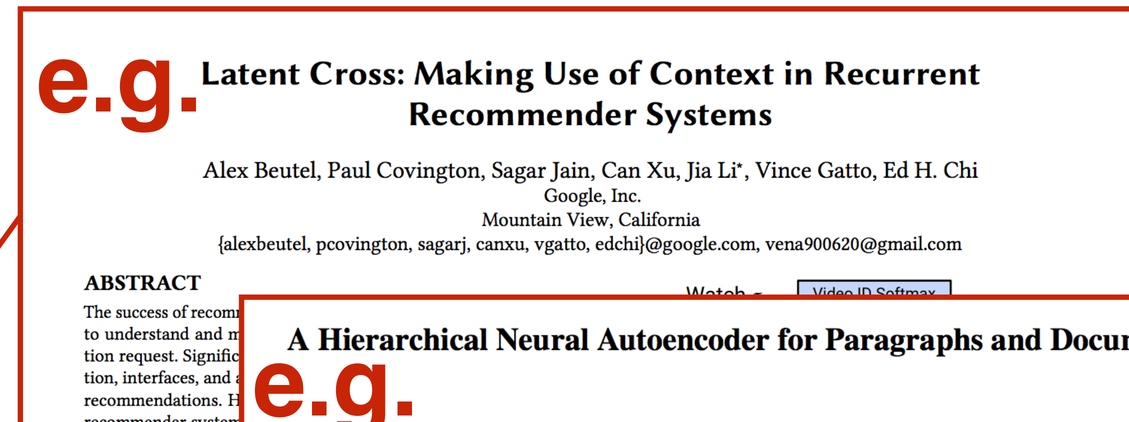
Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li*, Vince Gatto, Ed H. Chi
Google, Inc.
Mountain View, California
{alexbeutel, pcovington, sagarj, canxu, vgatto, edchi}@google.com, vena900620@gmail.com

ABSTRACT
The success of recommender systems often depends on their ability to understand and make use of the context of the recommendation request. Significant research has focused on how time, location, interfaces, and a plethora of other contextual features affect recommendations. However, in using deep neural networks for recommender systems, researchers often ignore these contexts or

Booking.com

Beyond single-step predictions

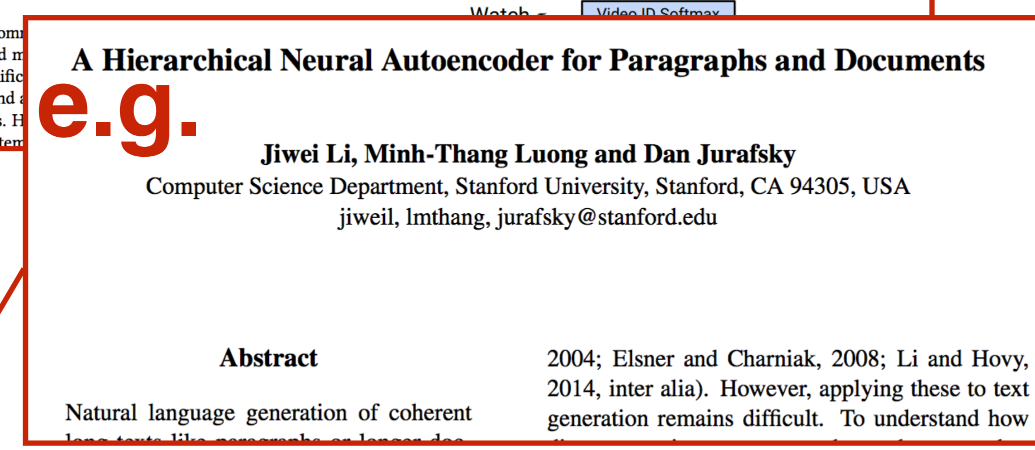
- Beam decoding for multiple steps
- Handling gaps properly
  - Sequence scoring
  - Direct modeling
- Additional user/trip/city-level features
- Hierarchical encoders to model multiple trips


e.g.

Latent Cross: Making Use of Context in Recurrent Recommender Systems

Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li*, Vince Gatto, Ed H. Chi
Google, Inc.
Mountain View, California
{alexbeutel, pcovington, sagarj, canxu, vgatto, edchi}@google.com, vena900620@gmail.com

ABSTRACT


e.g.

A Hierarchical Neural Autoencoder for Paragraphs and Documents

Jiwei Li, Minh-Thang Luong and Dan Jurafsky
Computer Science Department, Stanford University, Stanford, CA 94305, USA
jiweil, lmthang, jurafsky@stanford.edu

Abstract

Natural language generation of coherent

2004; Elsner and Charniak, 2008; Li and Hovy, 2014, inter alia). However, applying these to text generation remains difficult. To understand how

Booking.com

Thank you!

We're hiring!
https://workingatbooking.com

https://booking.ai

Booking.com