Advanced Data Management (CSCI 490/680)

Spatial Data

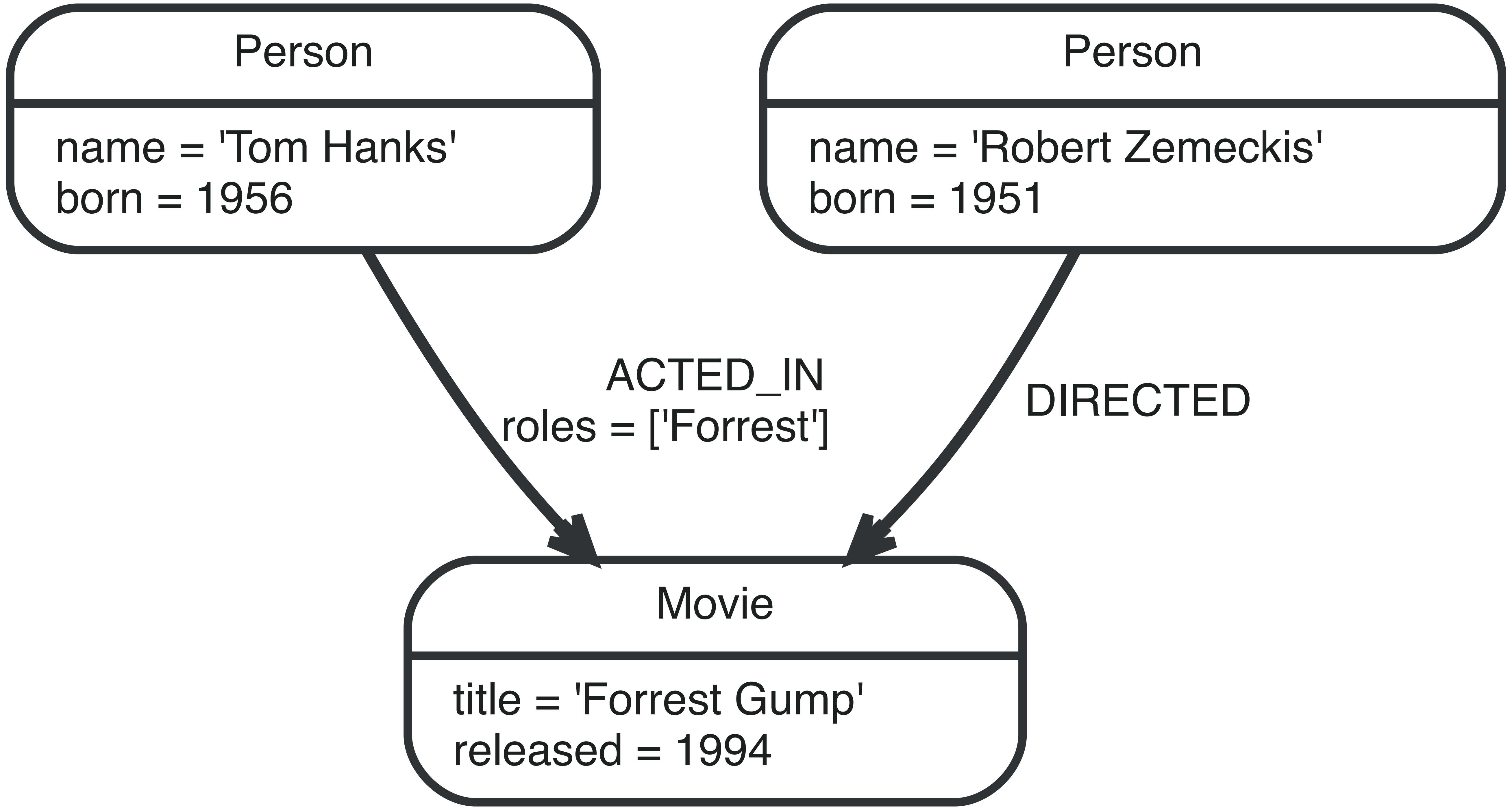
Dr. David Koop

Assignment 5

- Available soon
- Work with time series and spatial data
- Shorter assignment
- Due at the end of the semester

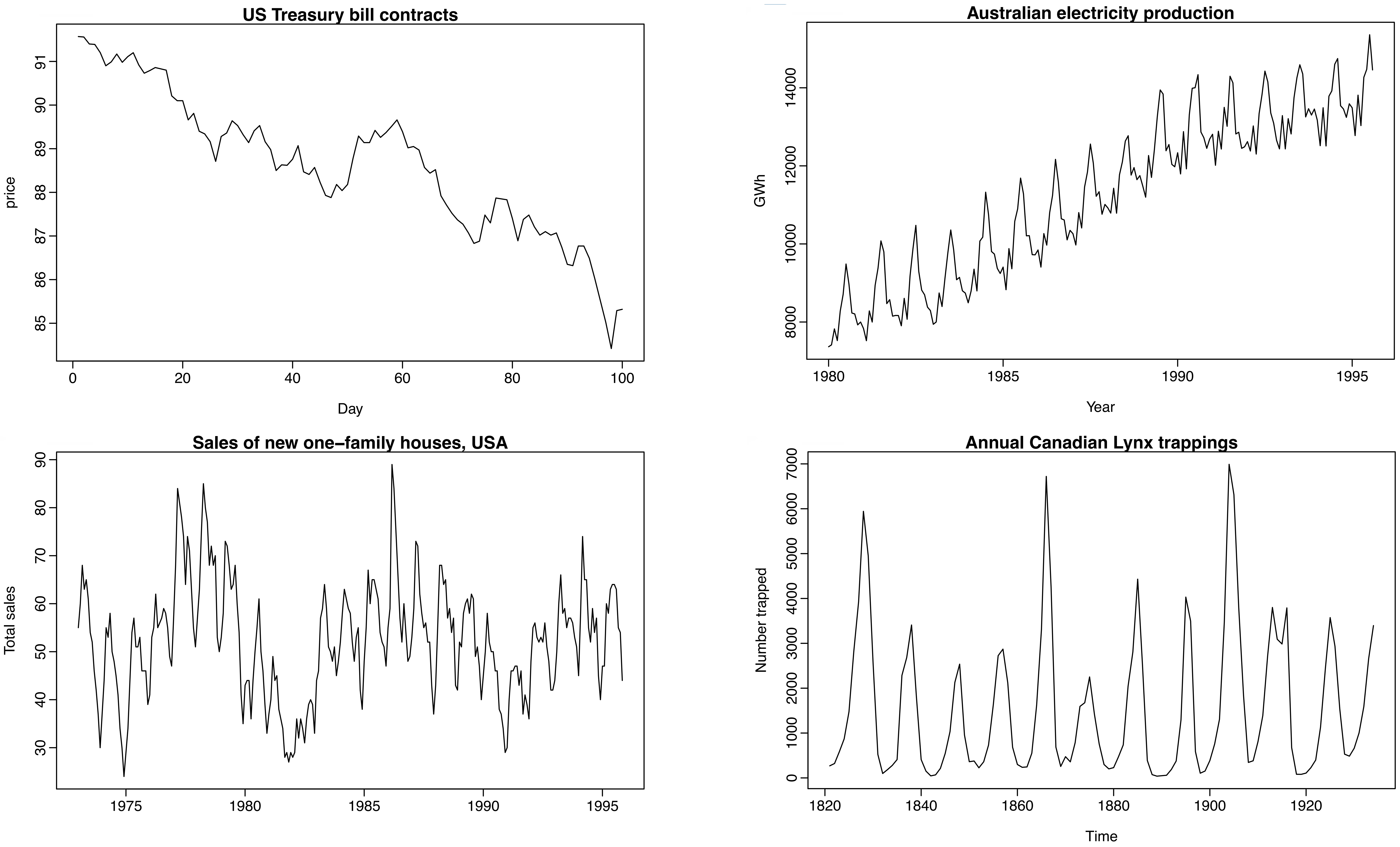
Graph Data

- Each vertex or edge may have properties associated with it
- May include identifiers or classes



[neo4j]

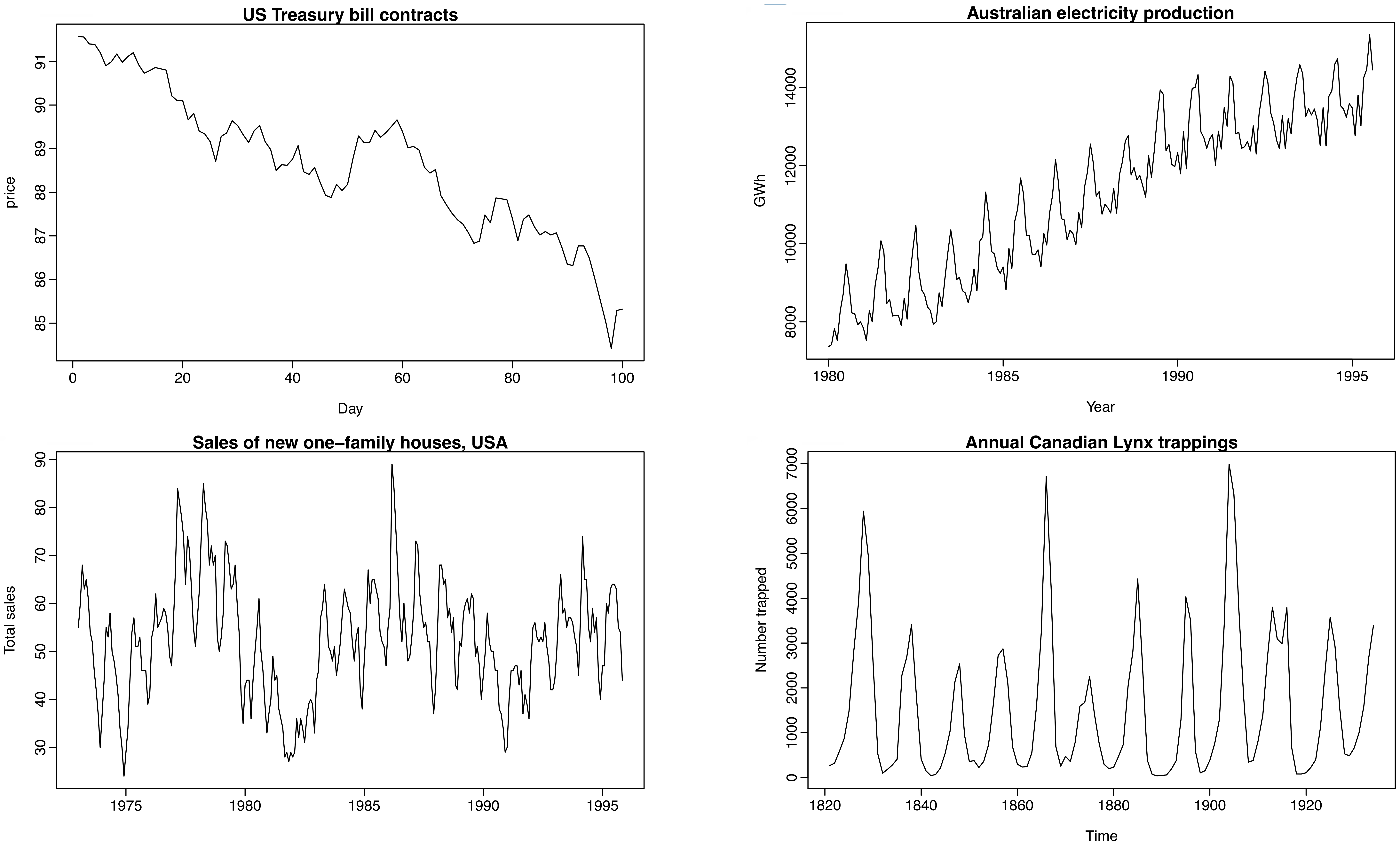
Time Series Data



[R. J. Hyndman]

Time Series Data

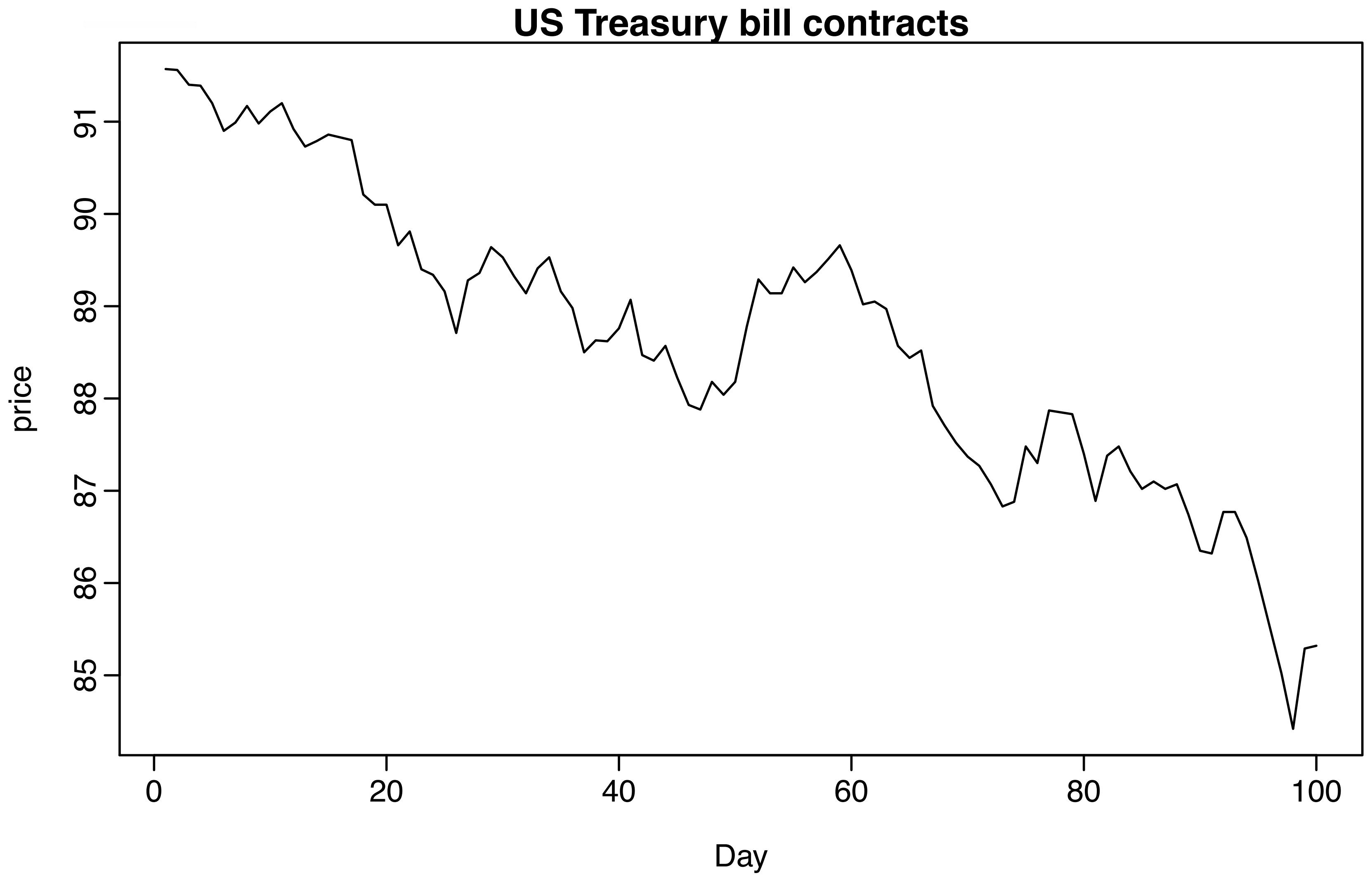
Trend

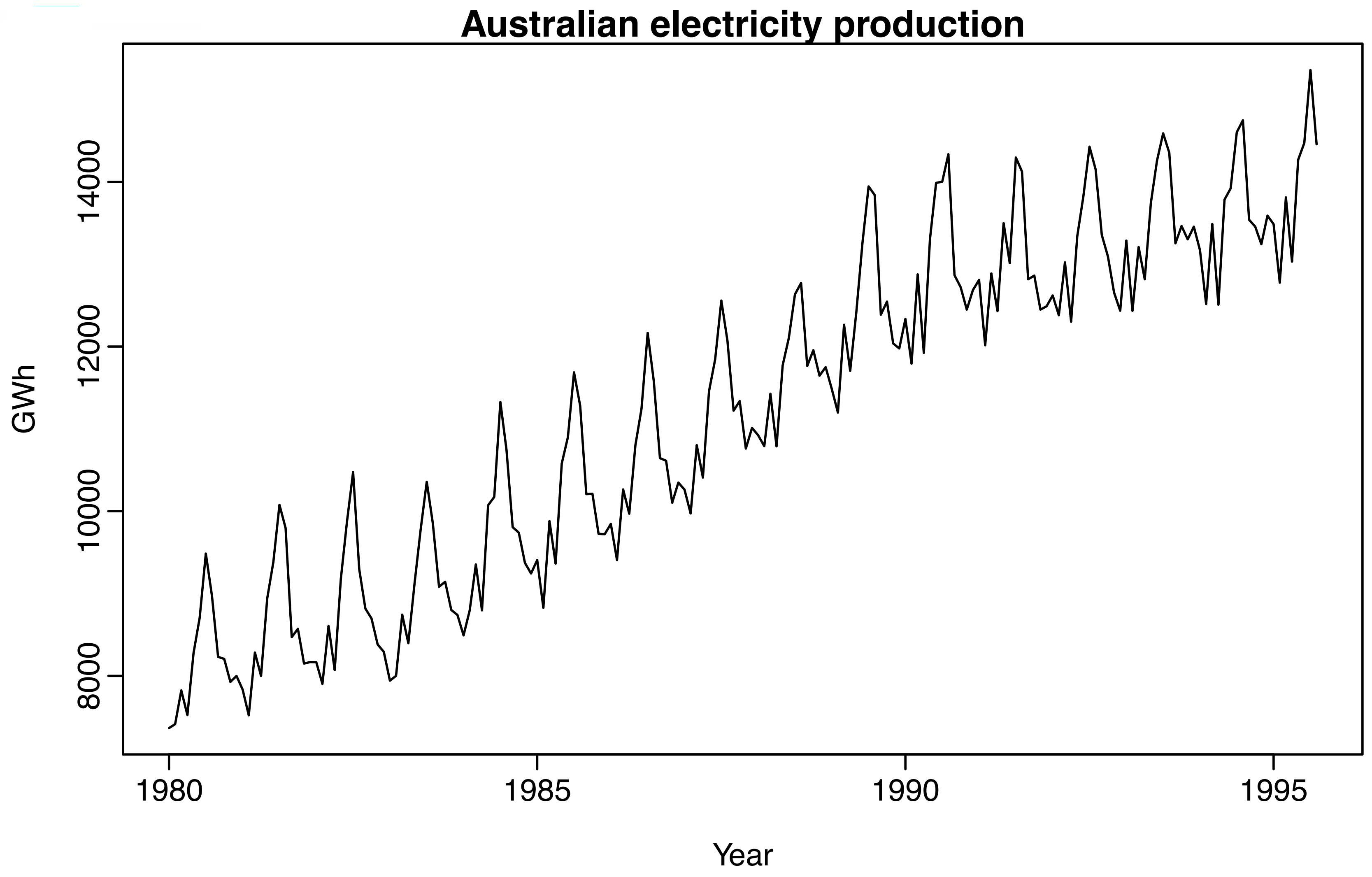


[R. J. Hyndman]

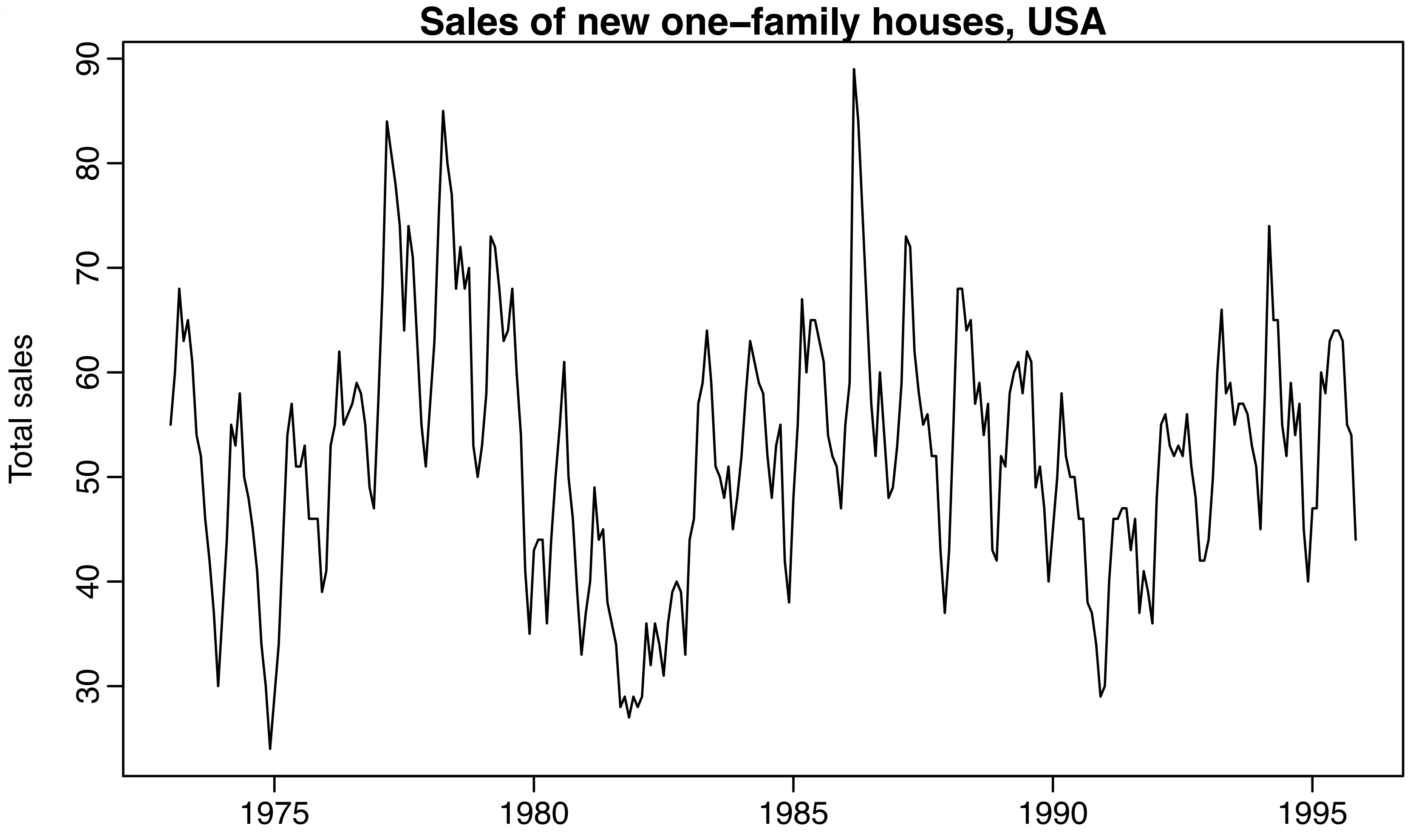
Time Series Data

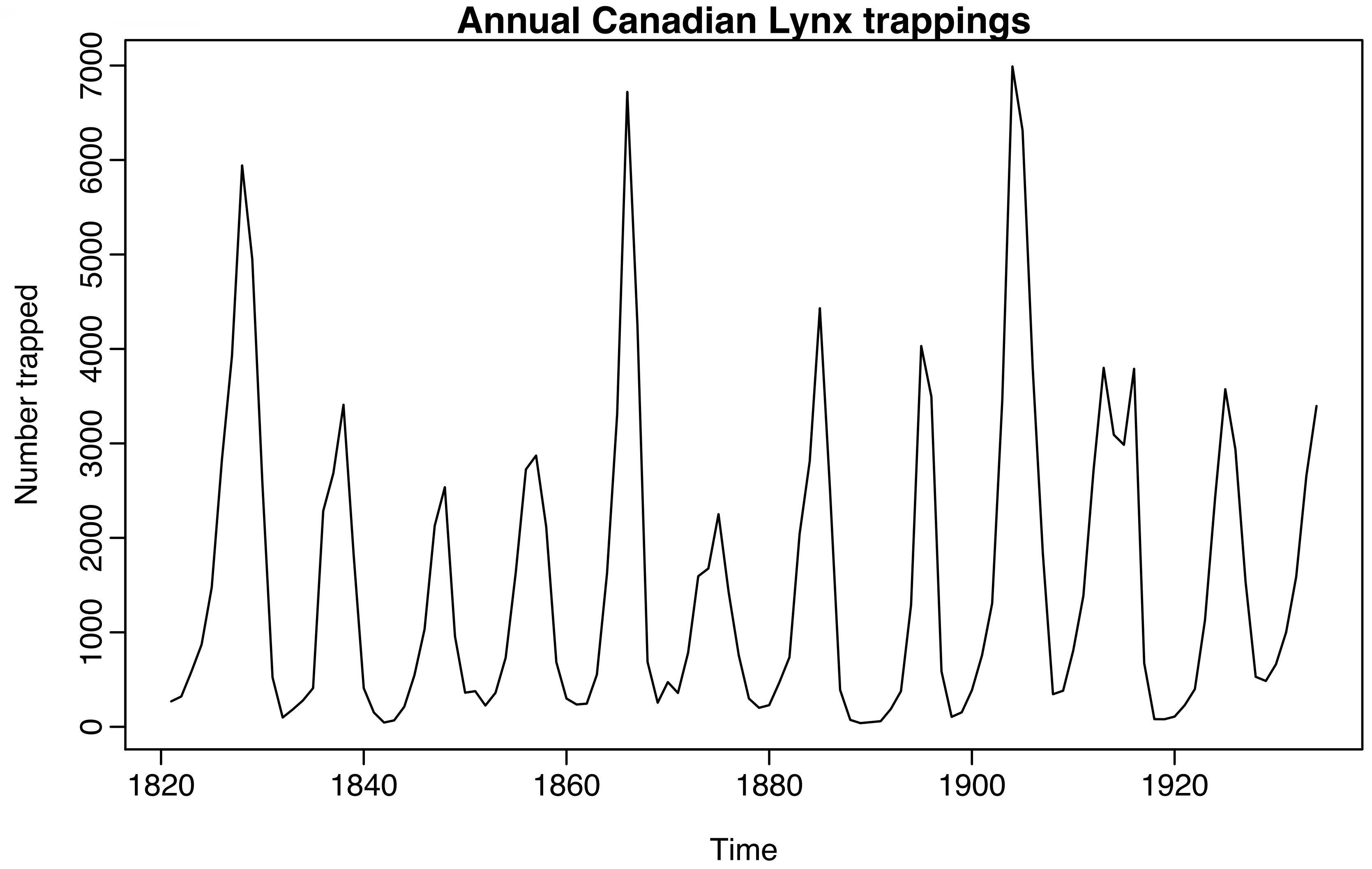
Trend





Trend + Seasonality

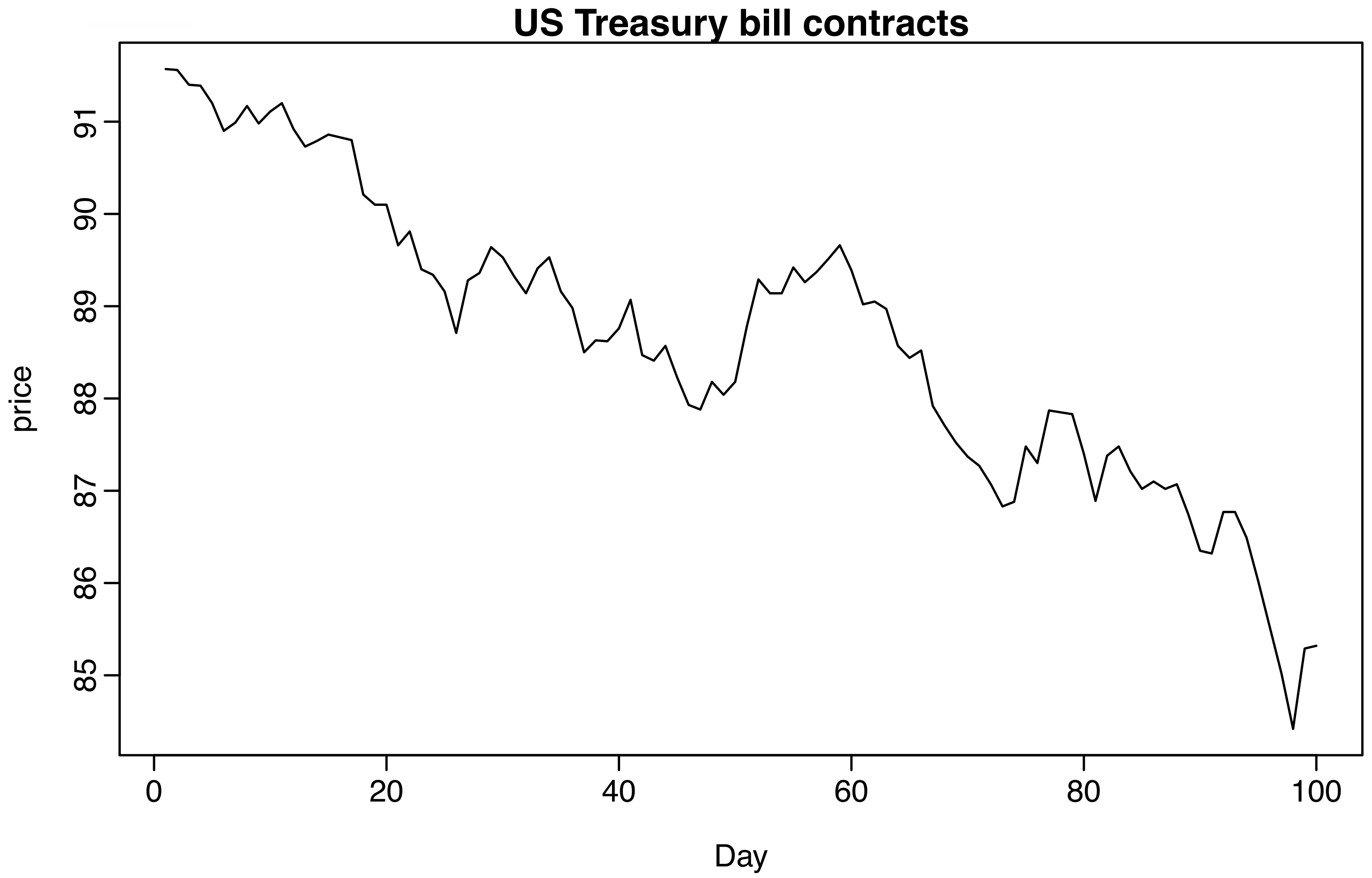


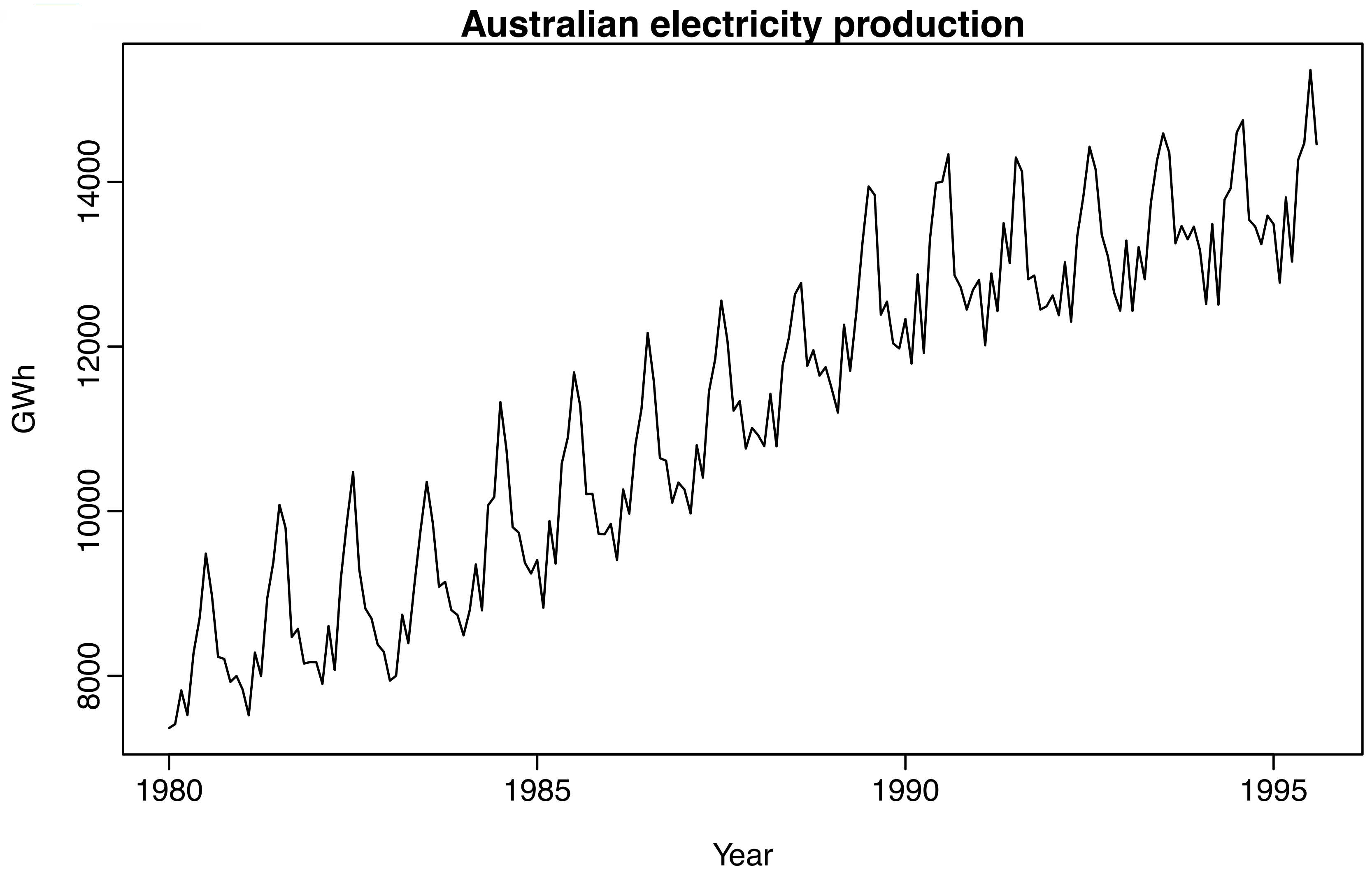


[R. J. Hyndman]

Time Series Data

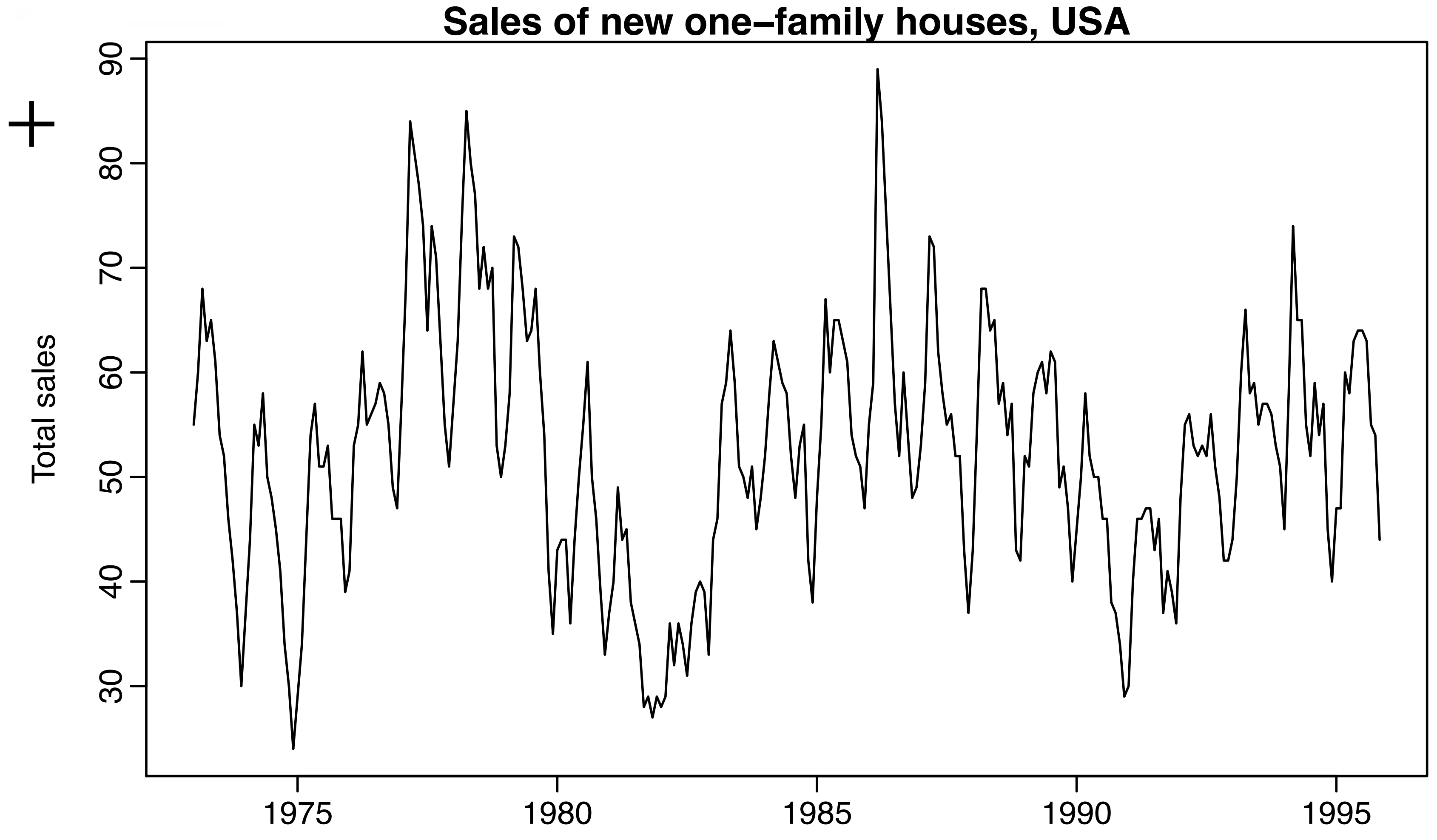
Trend

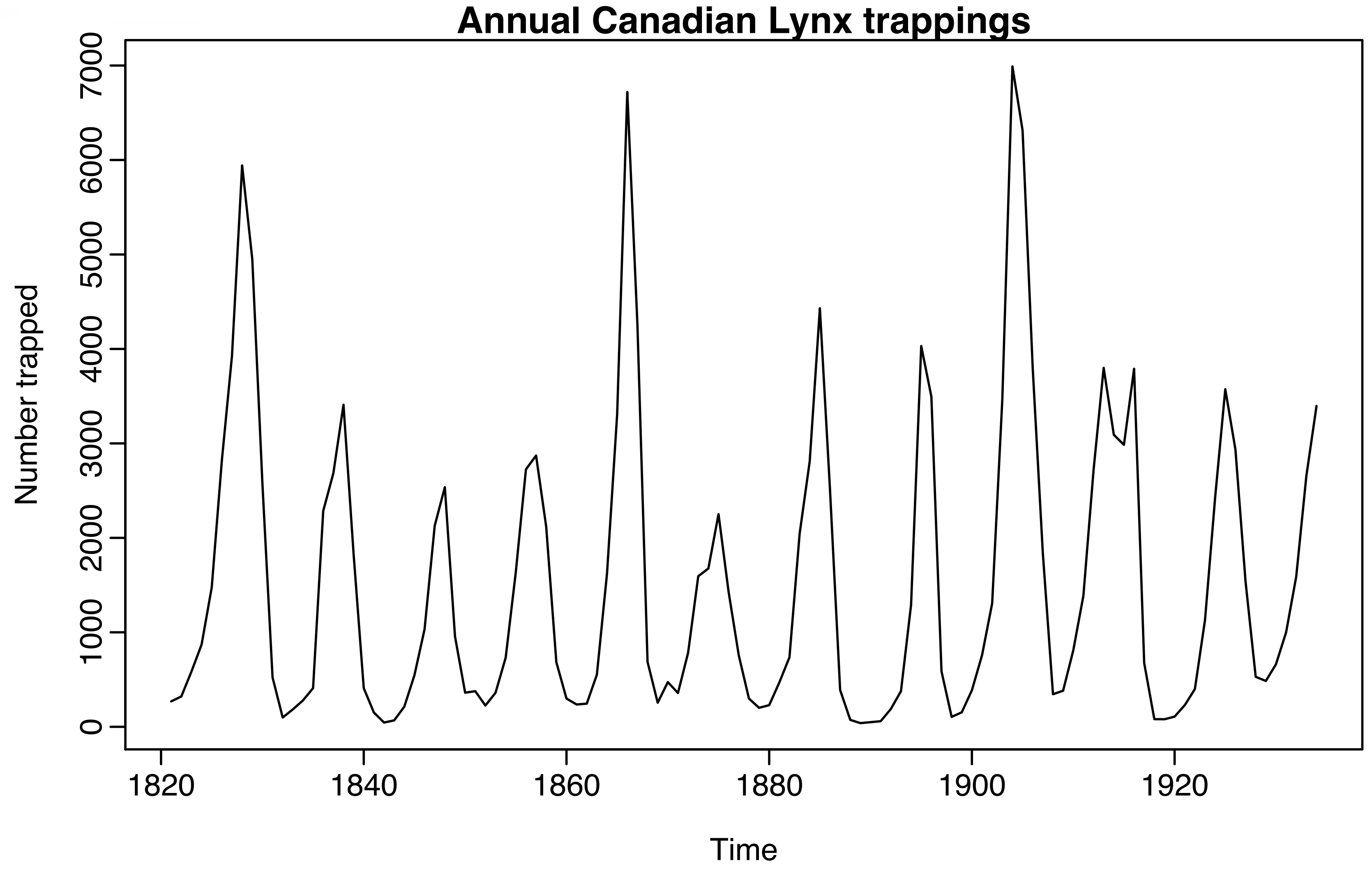




Trend + Seasonality

Seasonality + Cyclic

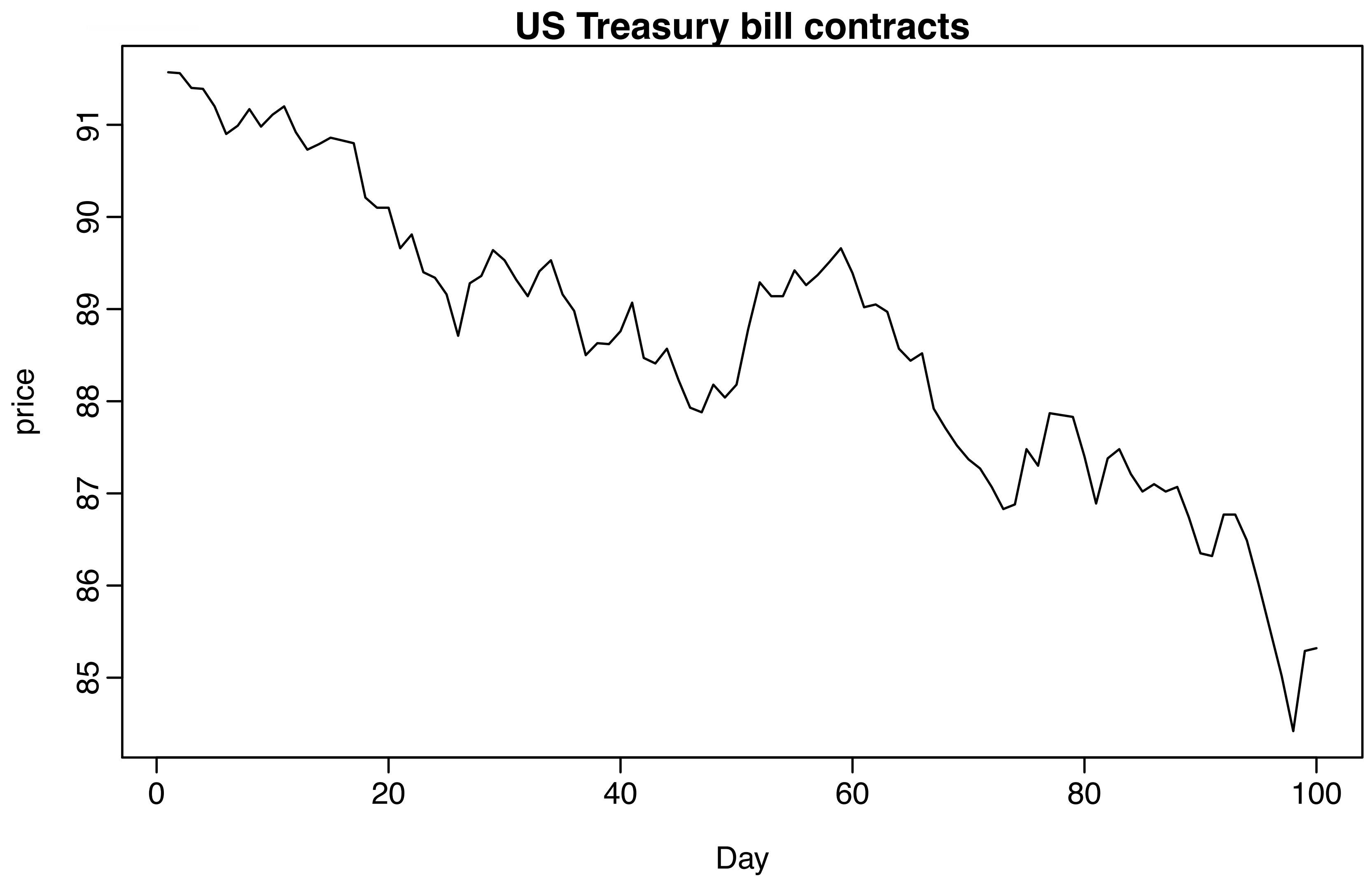


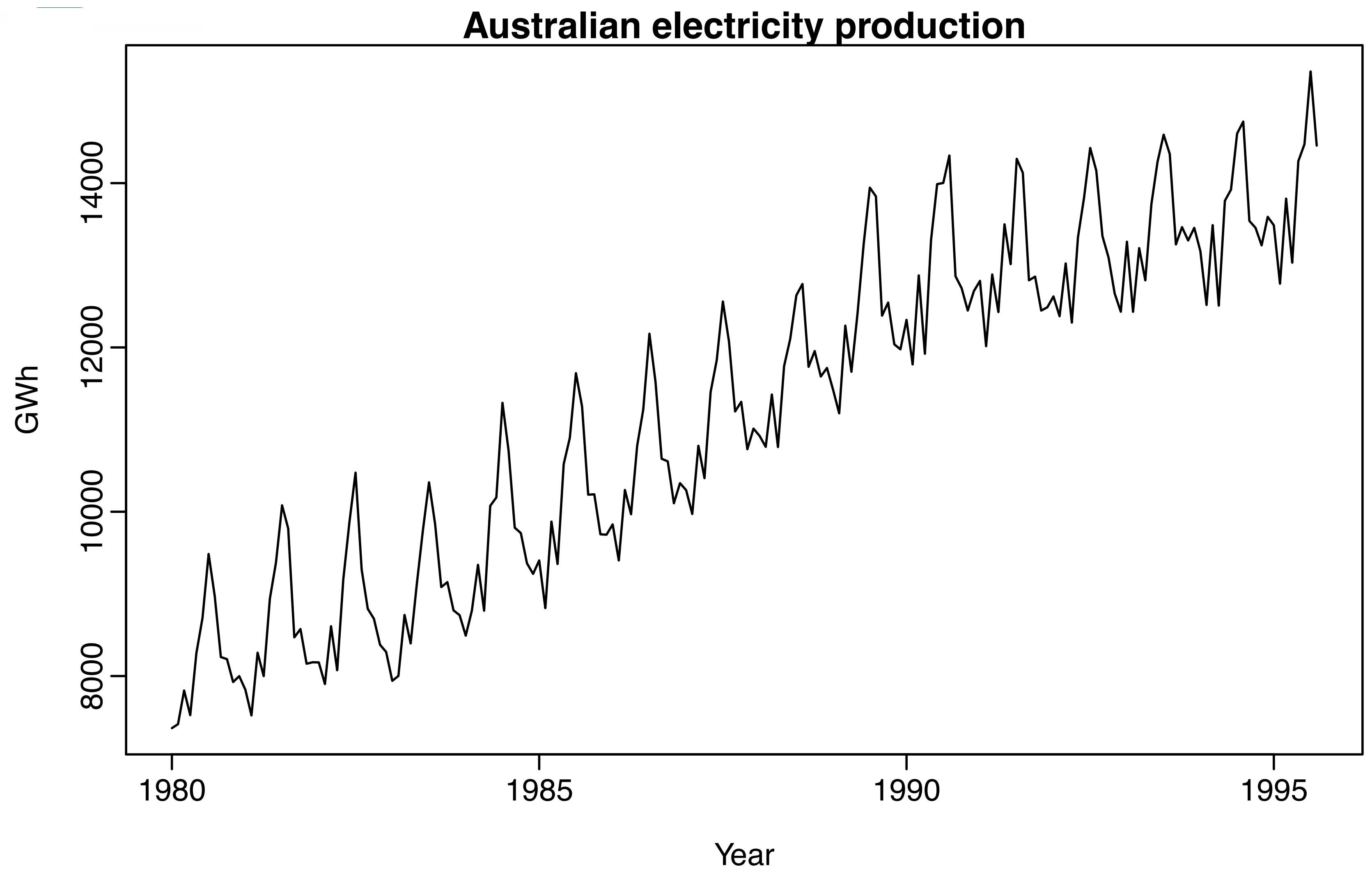


[R. J. Hyndman]

Time Series Data

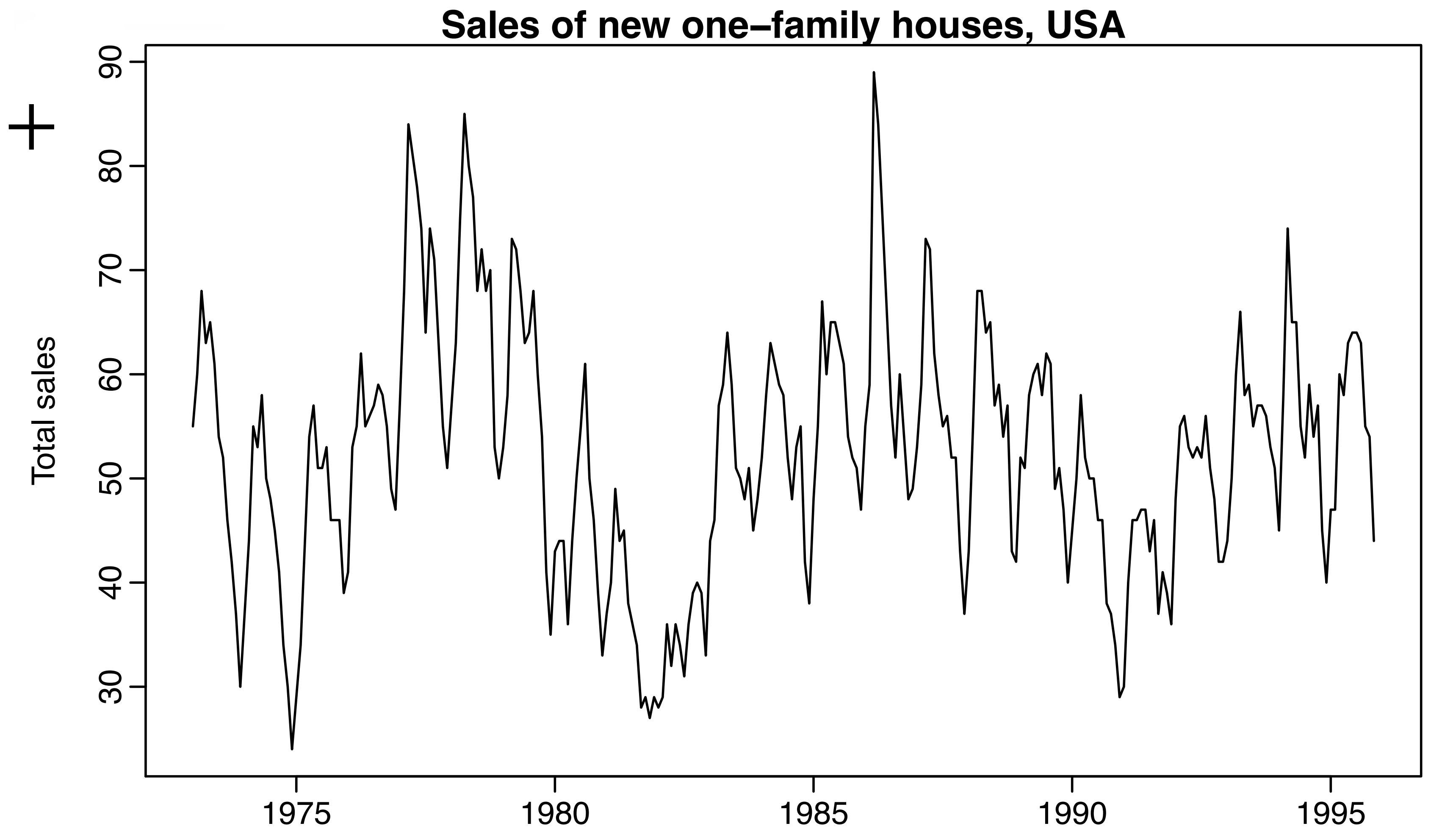
Trend

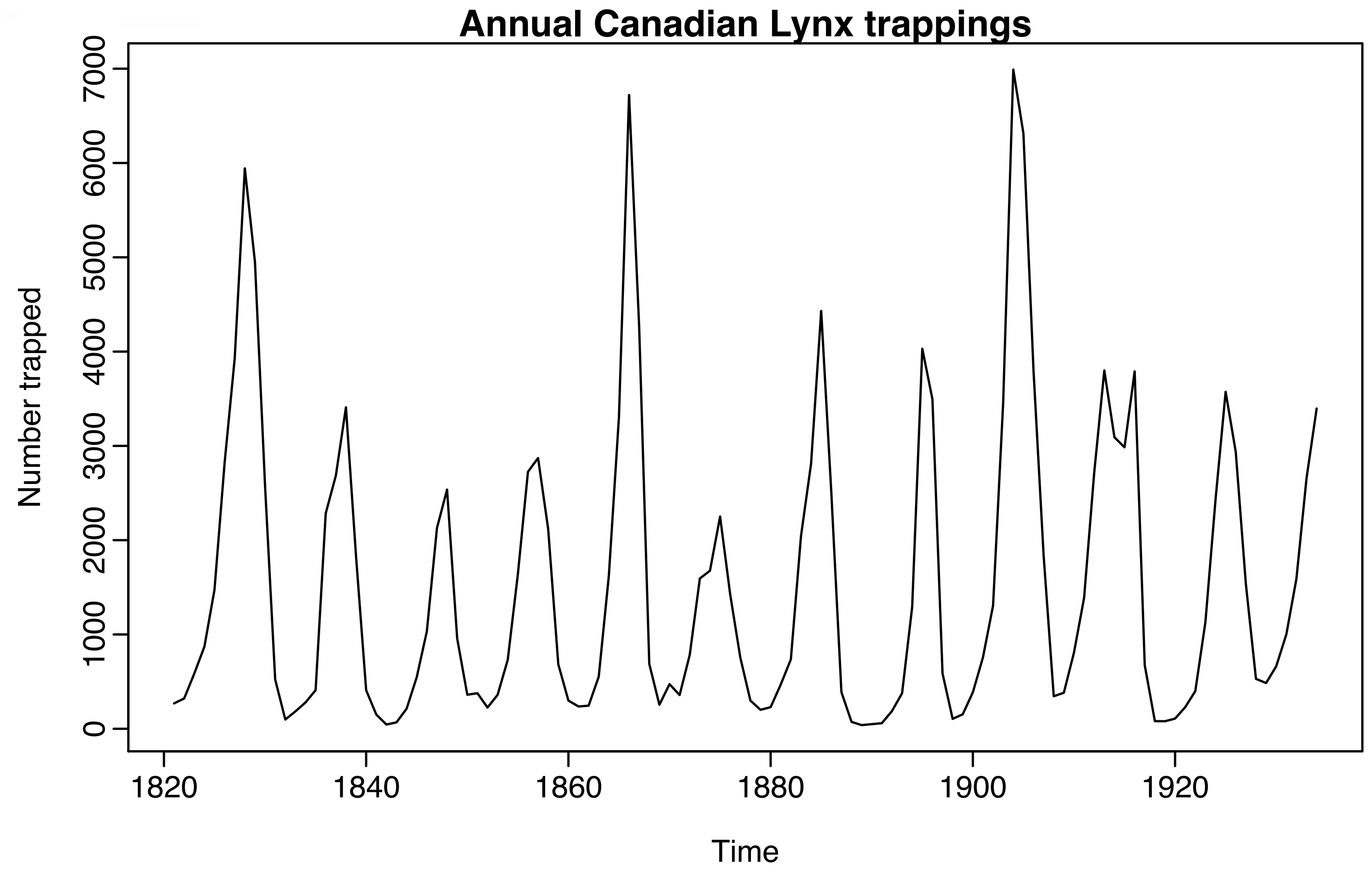




Trend + Seasonality

Seasonality + Cyclic





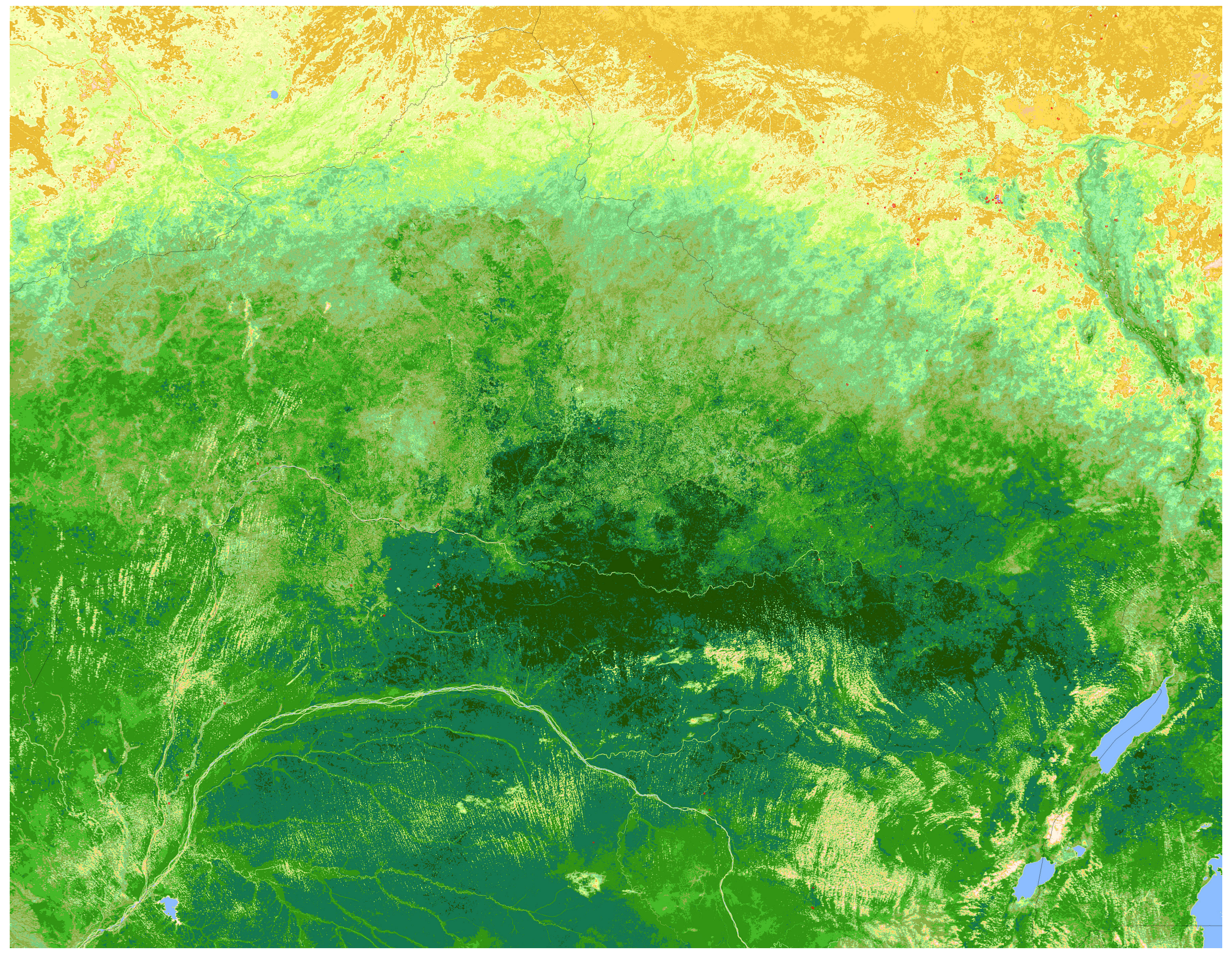
Stationary

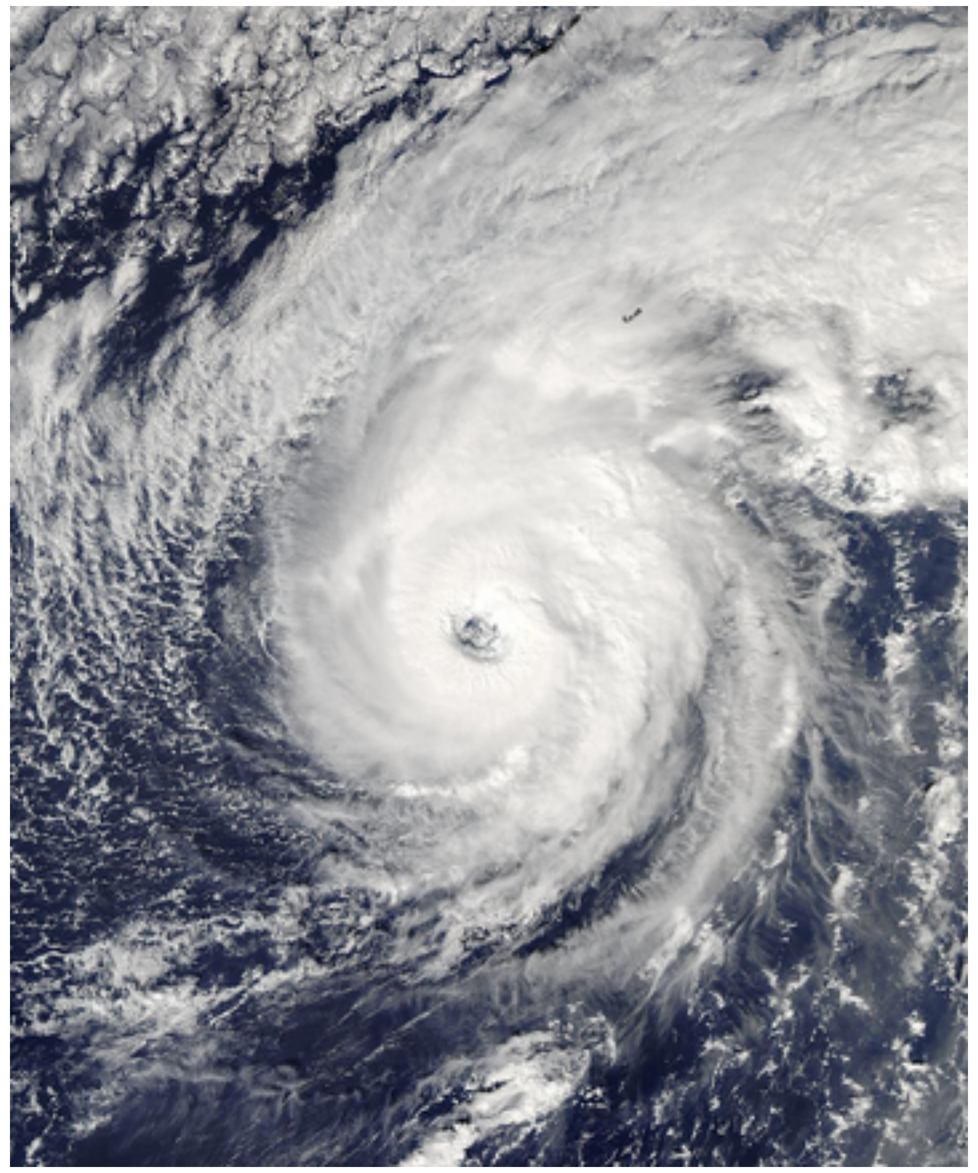
[R. J. Hyndman]

Today: Spatial Data

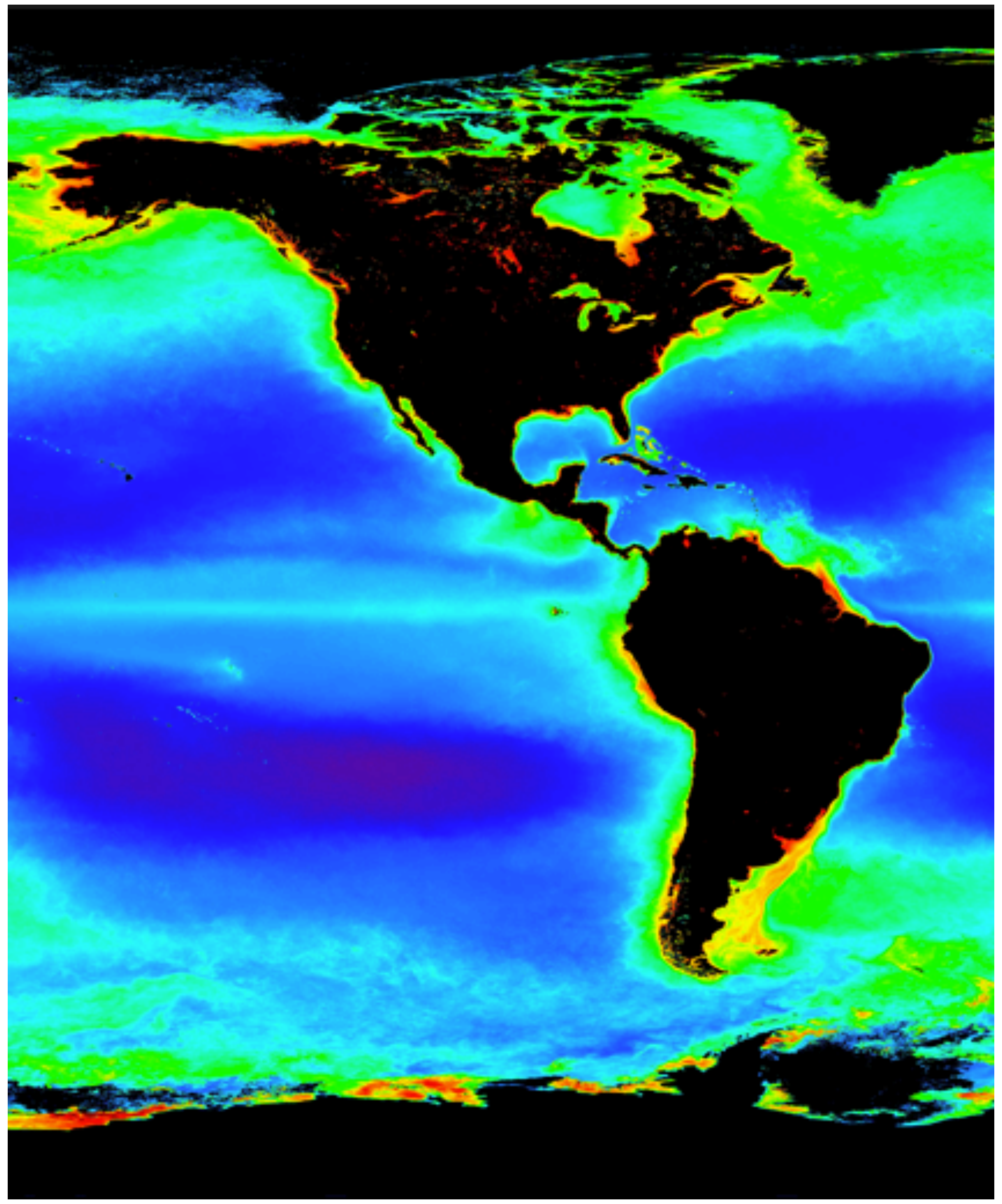
Track hurricanes

Measure vegetation density

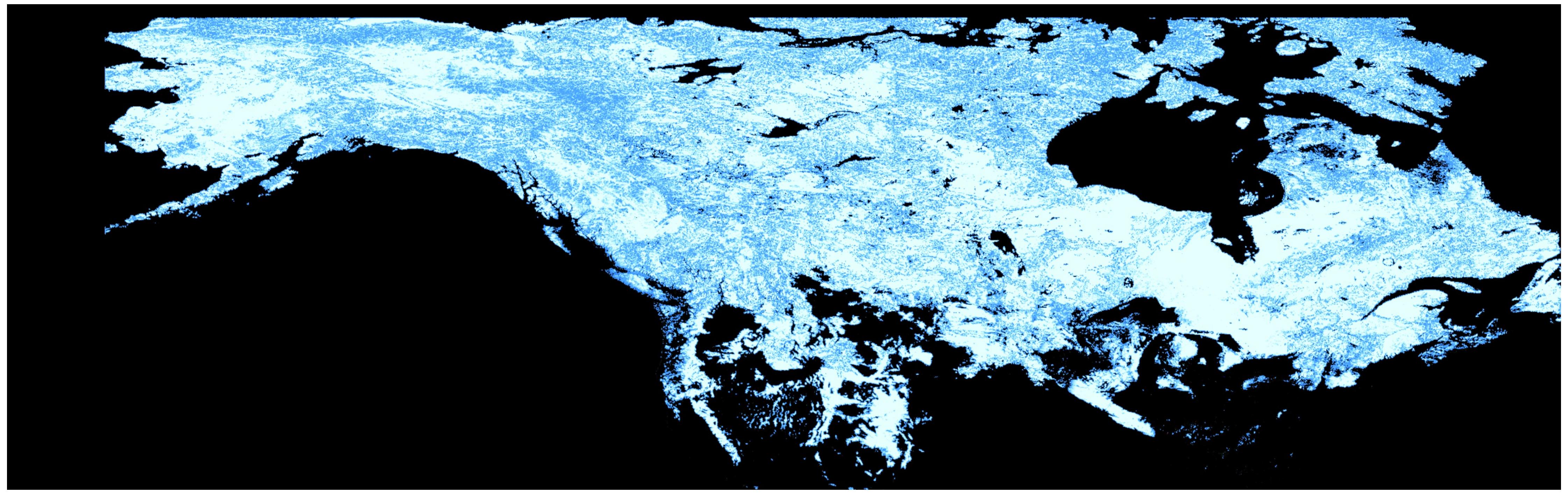




Track phytoplankton populations

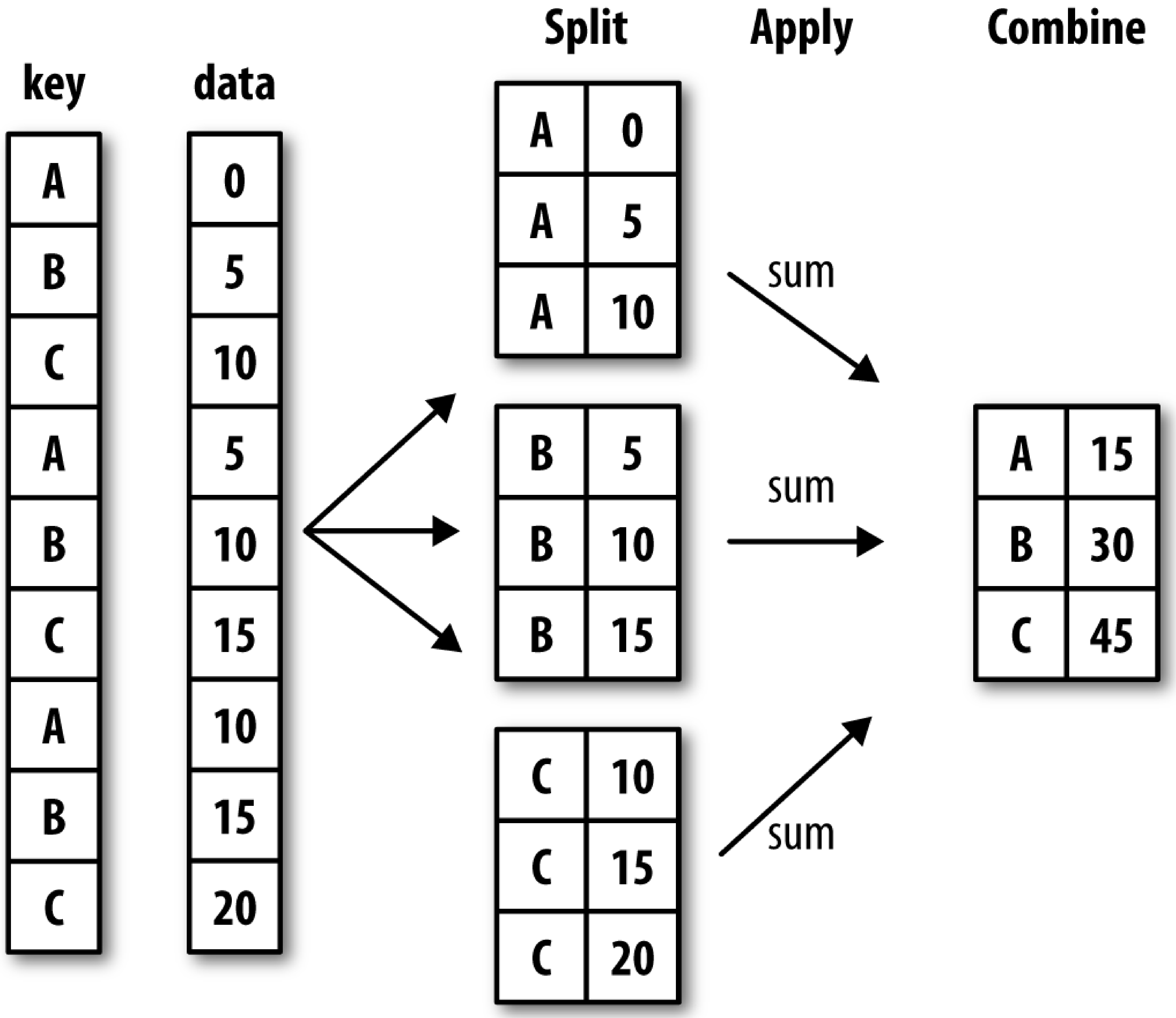


Measure snow melt



[L. Battle, 2017]

Split-Apply-Combine



[W. McKinney, Python for Data Analysis]

Types of GroupBy in pandas

- Aggregation: `agg`
  - `n:1` n group values become one value
  - Examples: mean, min, median
- Apply: `apply`
  - `n:m` n group values become m values
  - Most general (could do aggregation or transform with apply)
  - Example: top 5 in each group, filter
- Transform: `transform`
  - `n:n` n group values become n values
  - Cannot mutate the input

Time series data

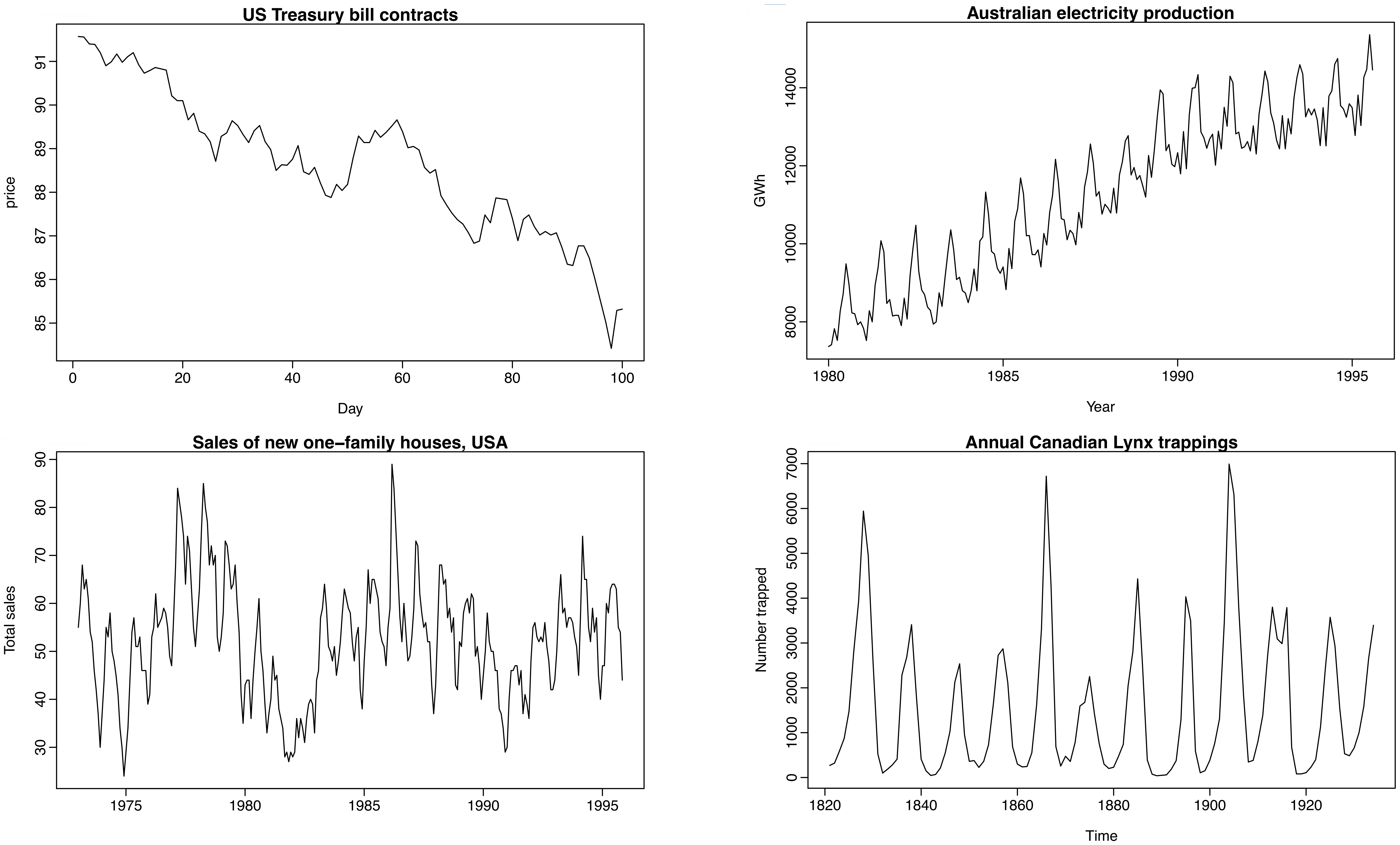
- Technically, it's normal tabular data with a timestamp attached
- But… we have observations of the same values over time, usually **in order**
- This allows more analysis
- Example: Web site database that tracks the last time a user logged in
  - 1: Keep an attribute `lastLogin` that is **overwritten** every time user logs in
  - 2: **Add a new row** with login information every time the user logs in
  - Option 2 takes more storage, but we can also do a lot more analysis!

Features of Time Series Data

- Trend: long-term increase or decrease in the data
- Seasonal Pattern: time series is affected by seasonal factors such as the time of the year or the day of the week (fixed and of known frequency)
- Cyclic Pattern: rises and falls that are not of a fixed frequency
- Stationary: no predictable patterns (roughly horizontal with constant variance)
  - White noise series is stationary
  - Will look the basically the same whenever you observe it

[Hyndman and Athanosopoulos]

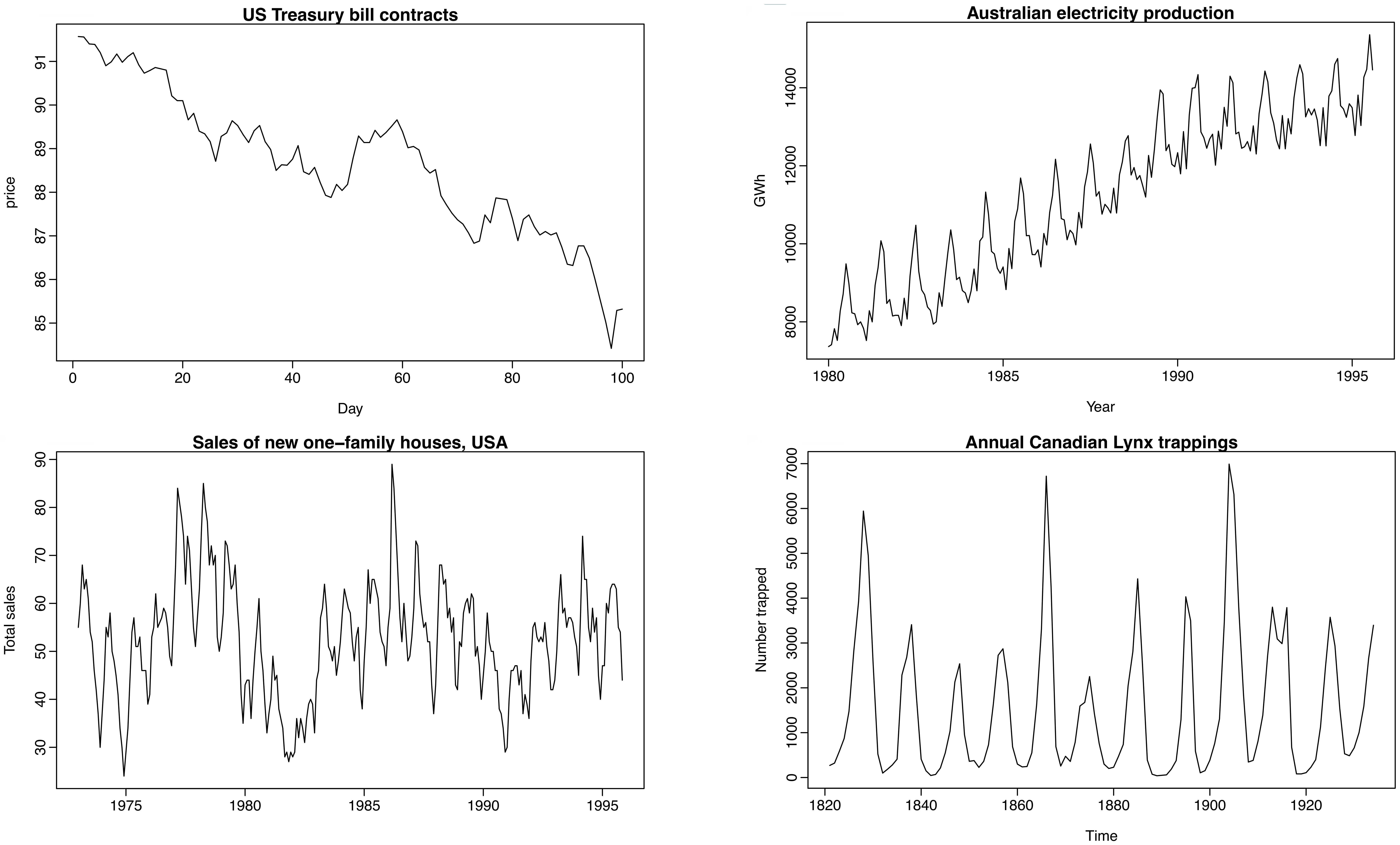
Time Series Data



[R. J. Hyndman]

Time Series Data

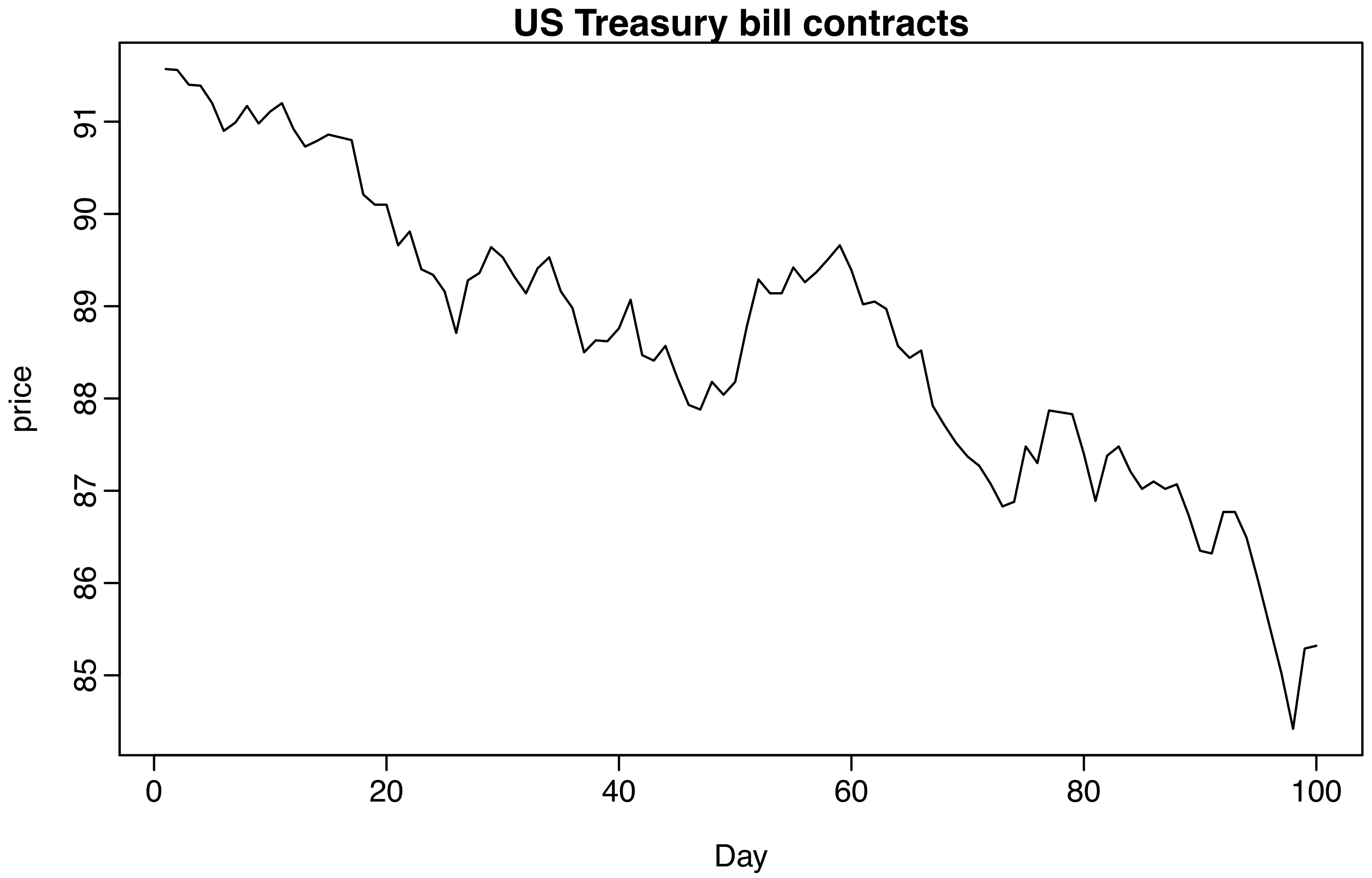
Trend

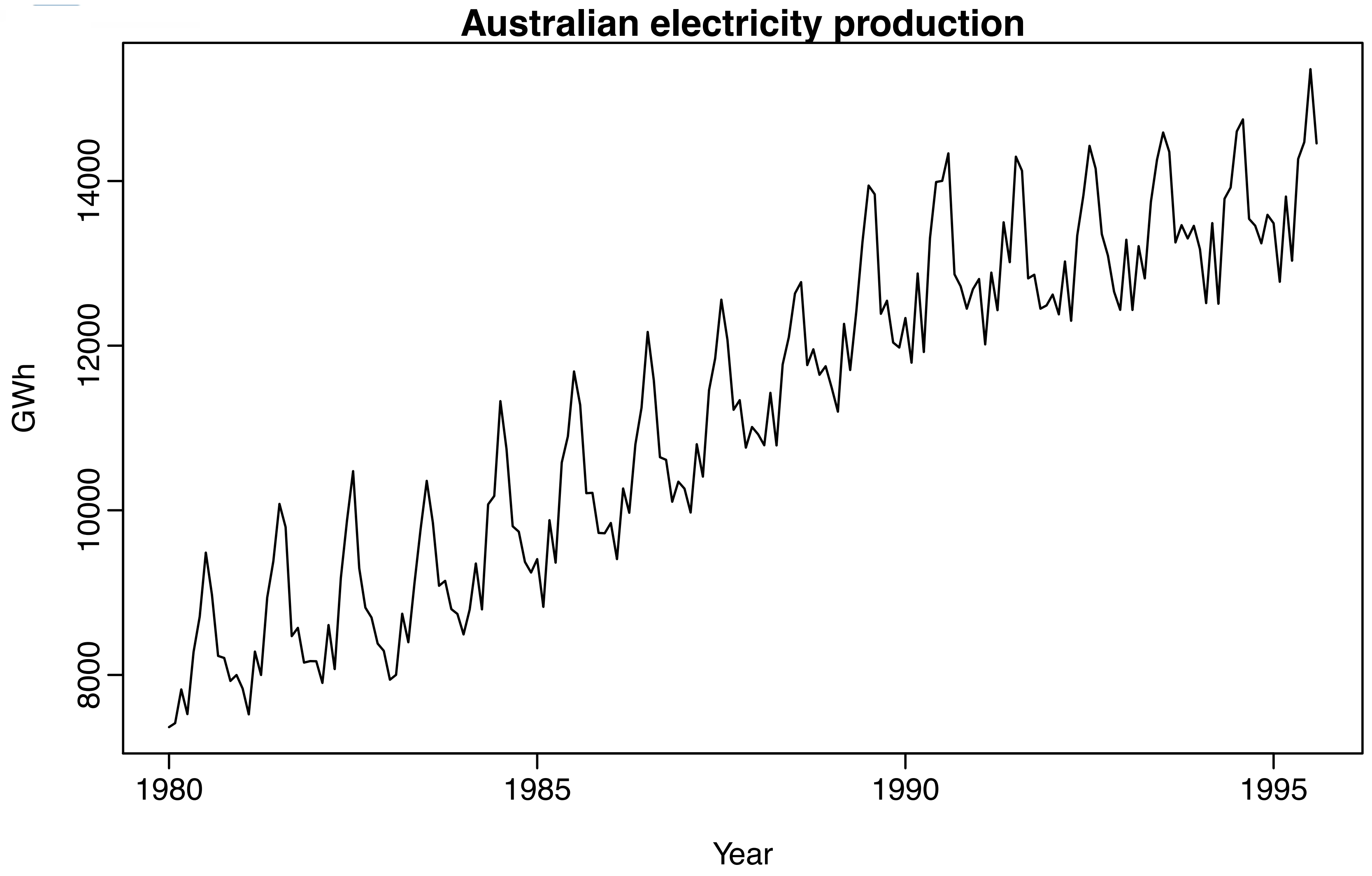


[R. J. Hyndman]

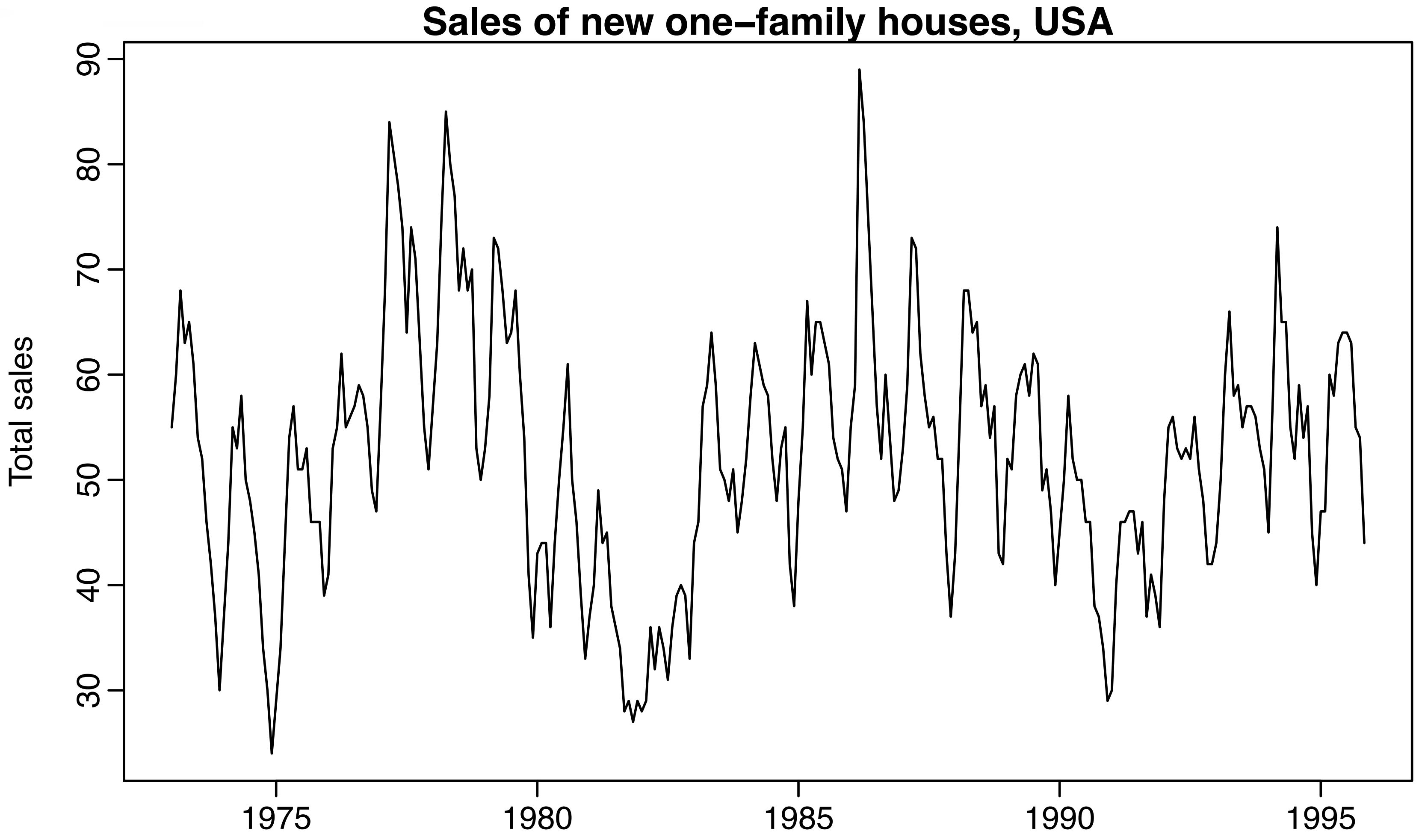
Time Series Data

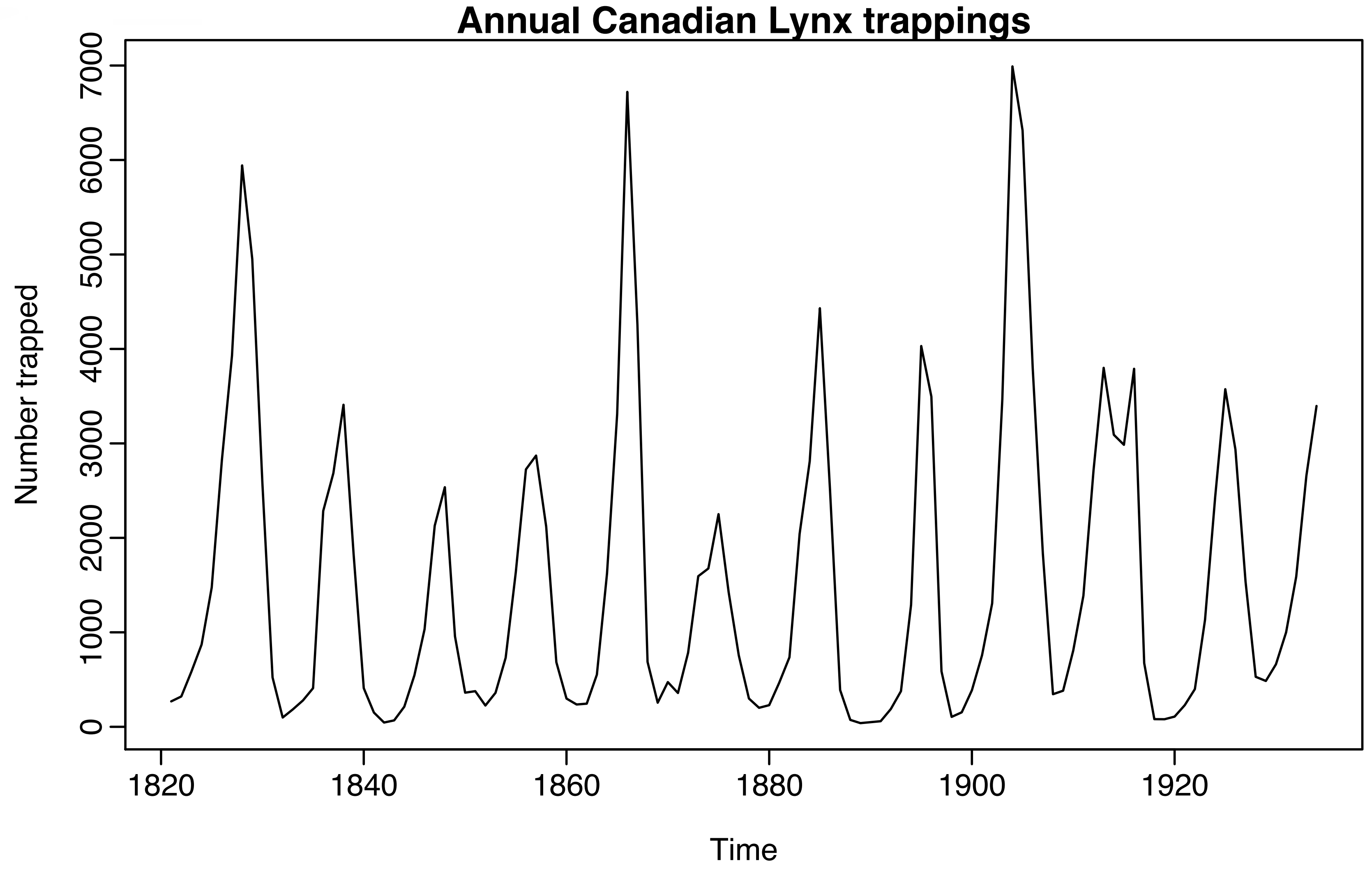
Trend





Trend +
Seasonality

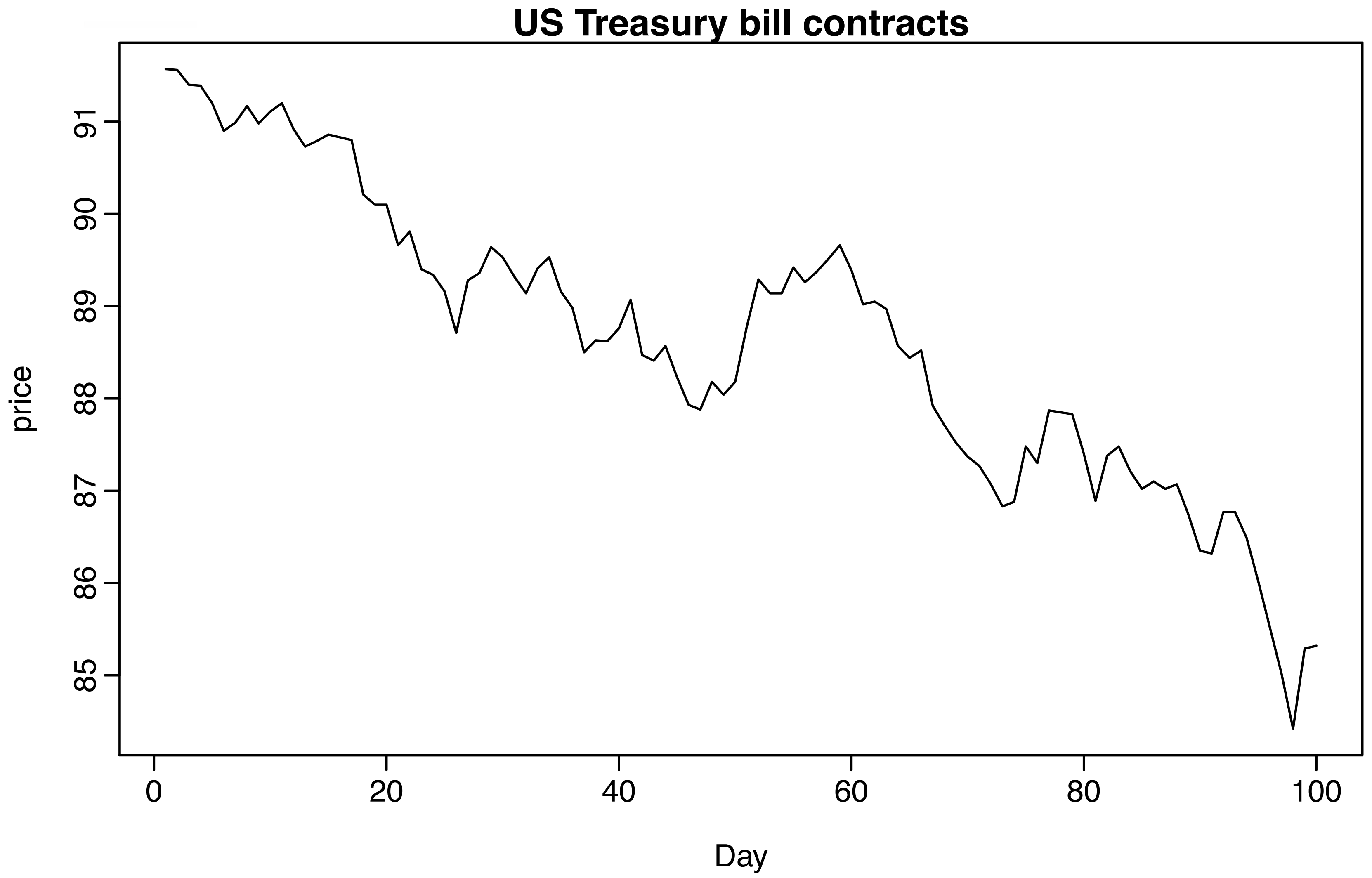


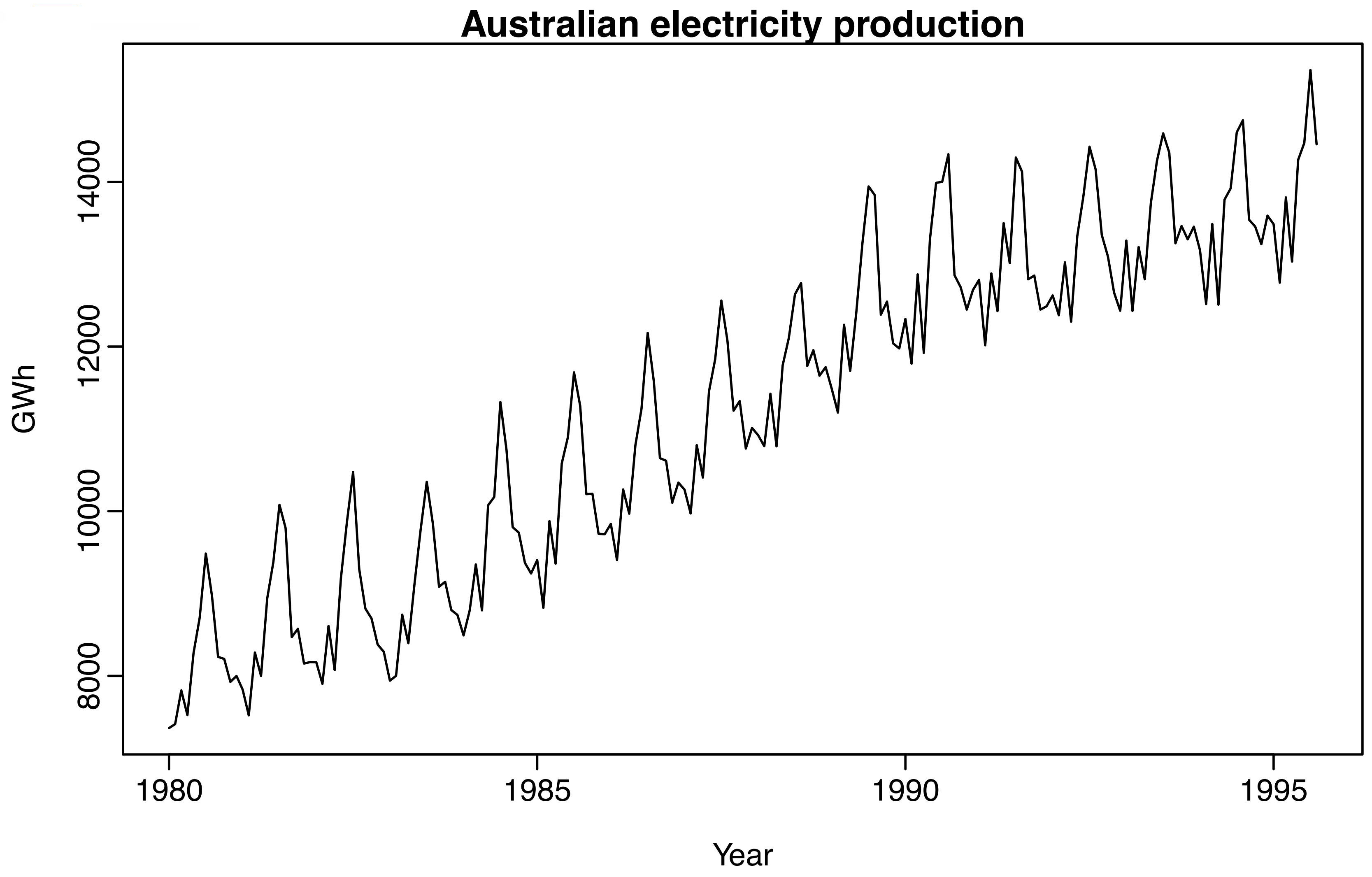


[R. J. Hyndman]

Time Series Data

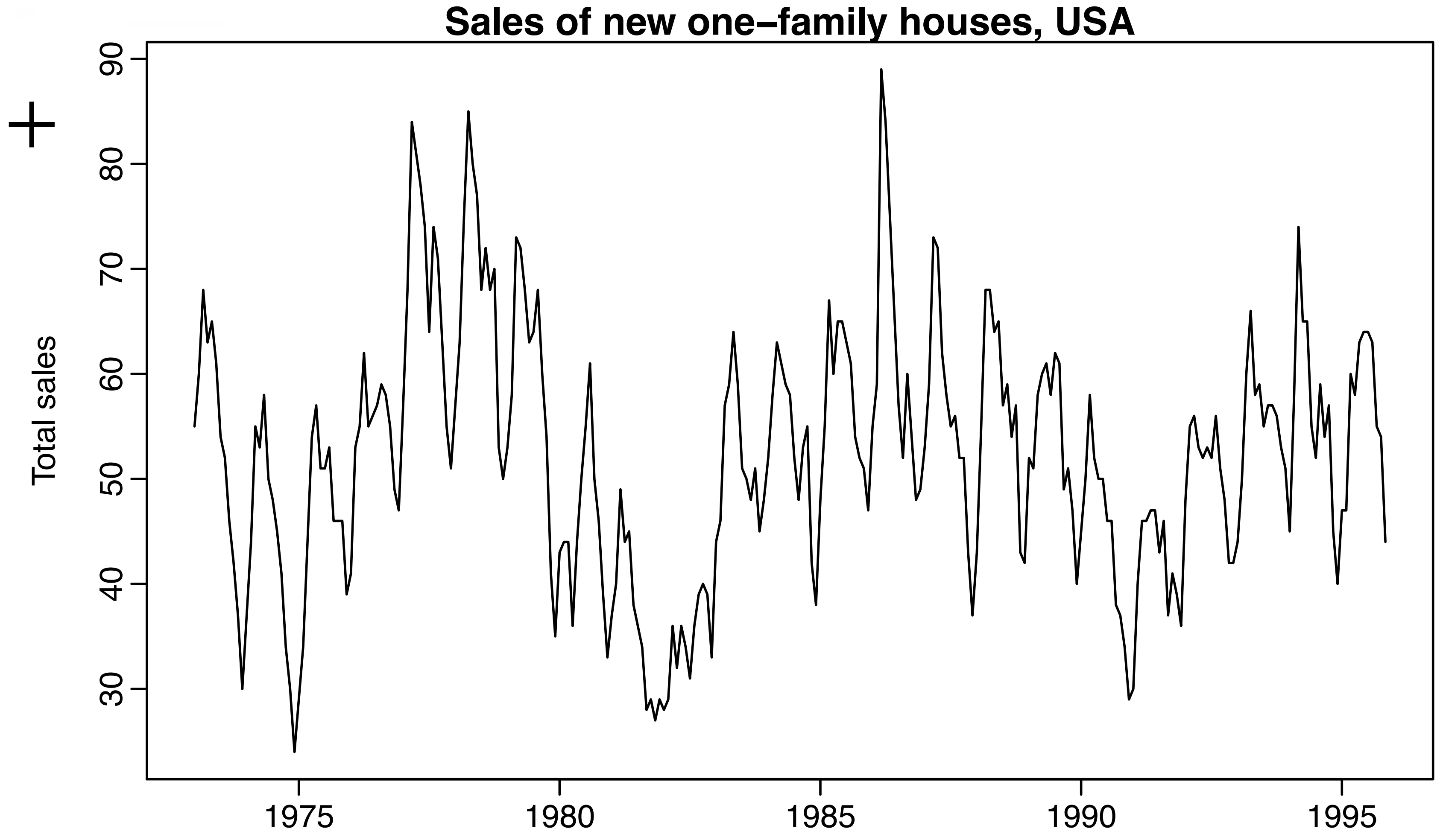
Trend

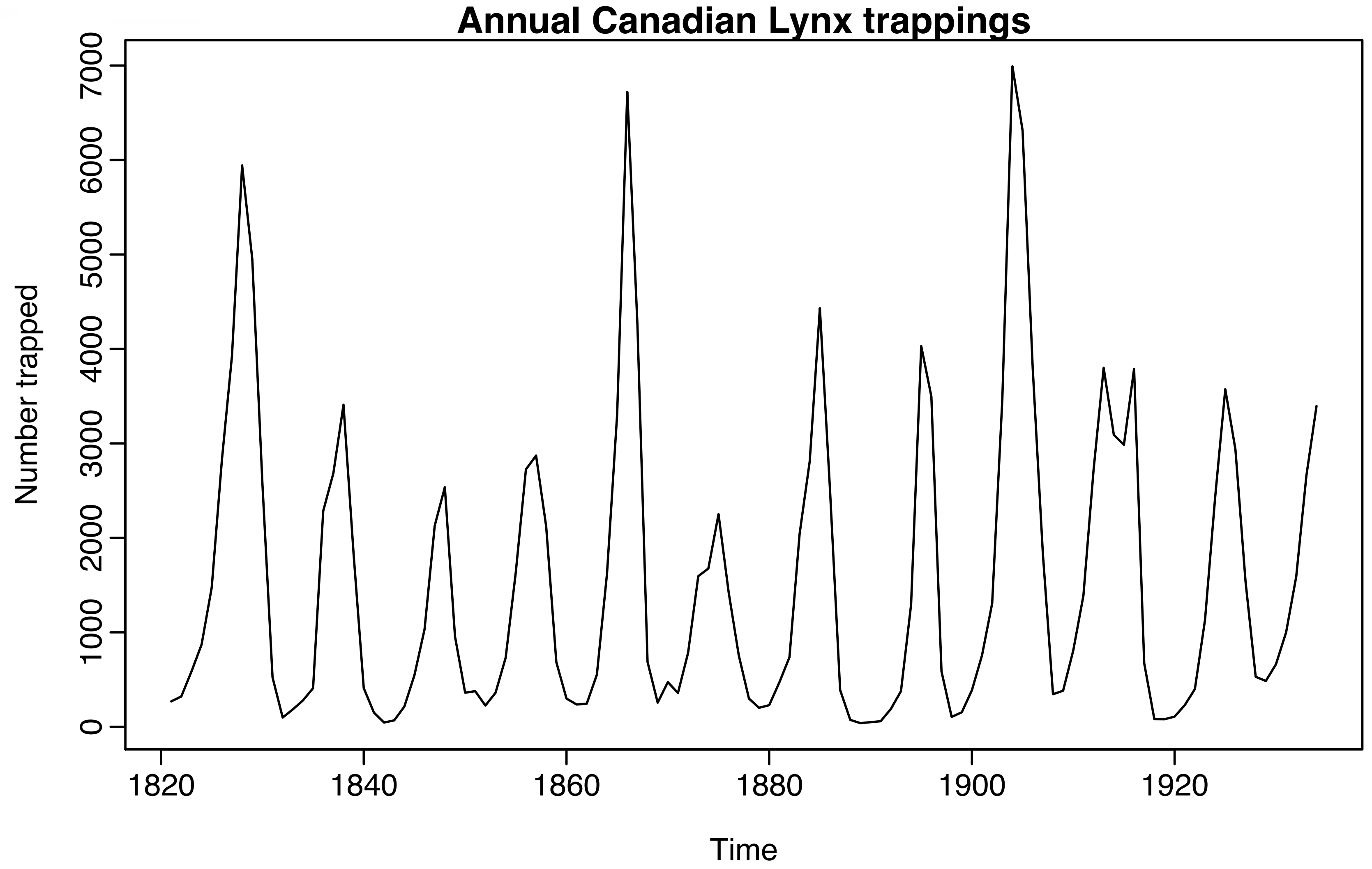




Trend + Seasonality

Seasonality + Cyclic

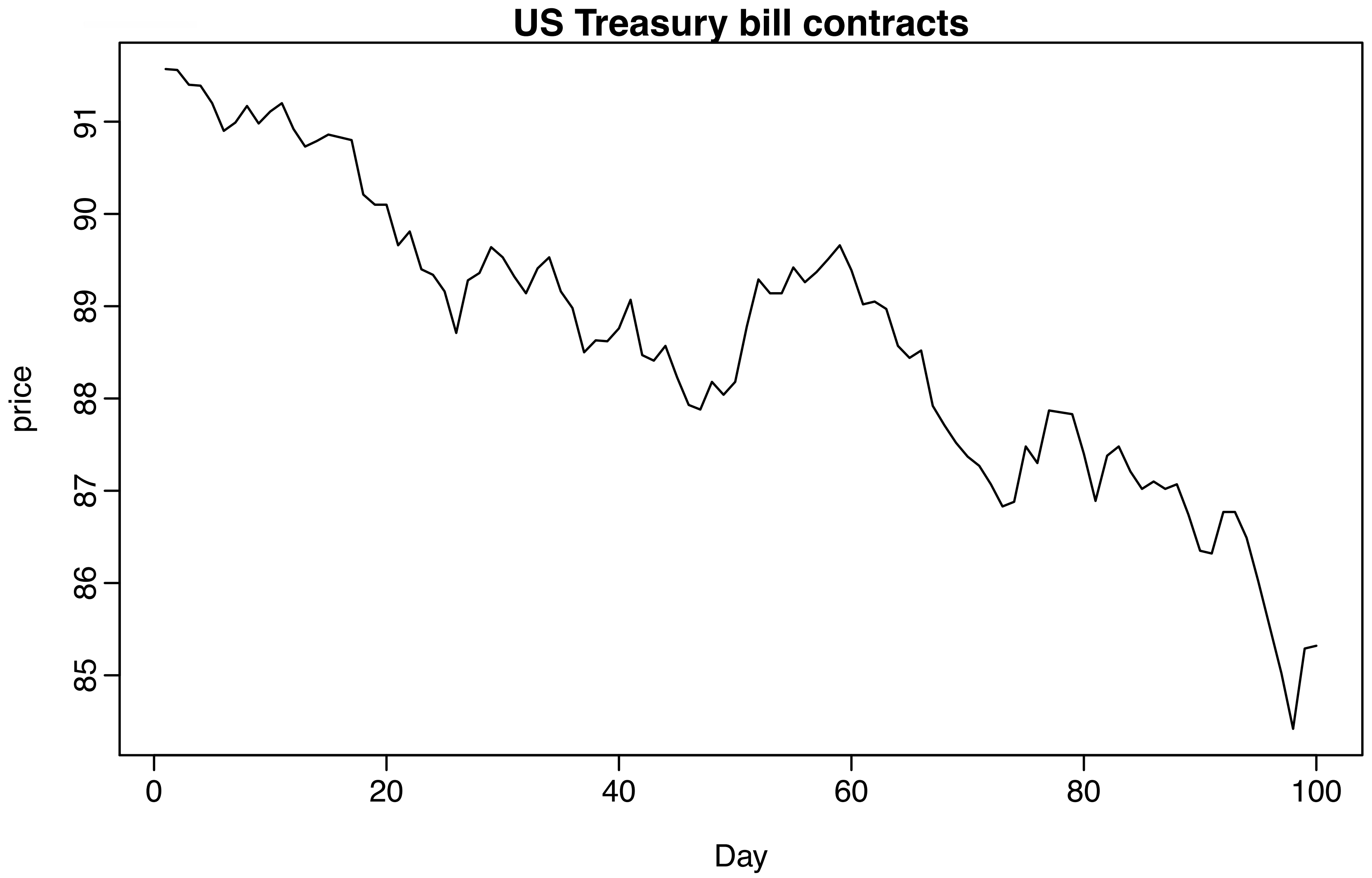


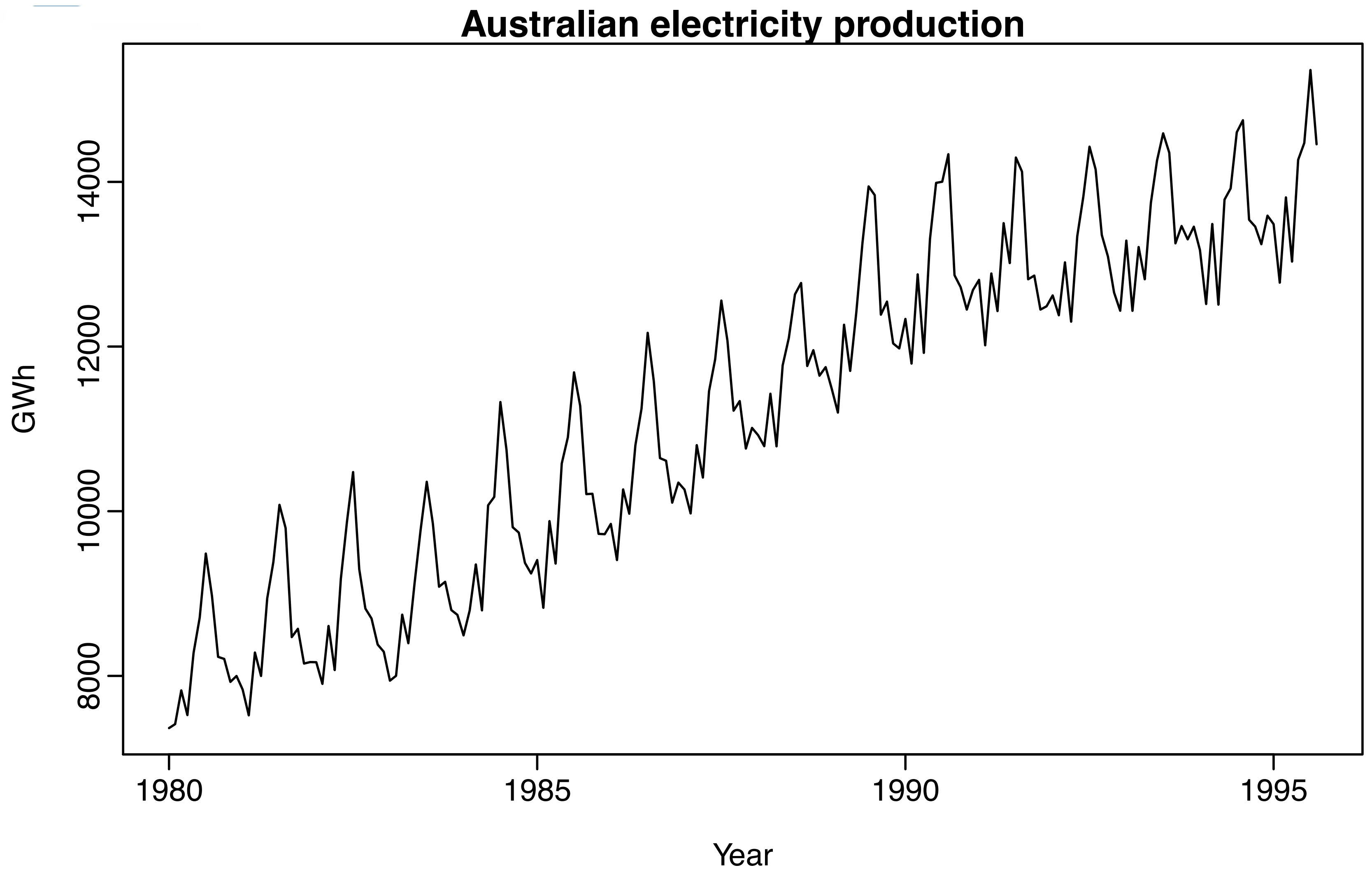


[R. J. Hyndman]

Time Series Data

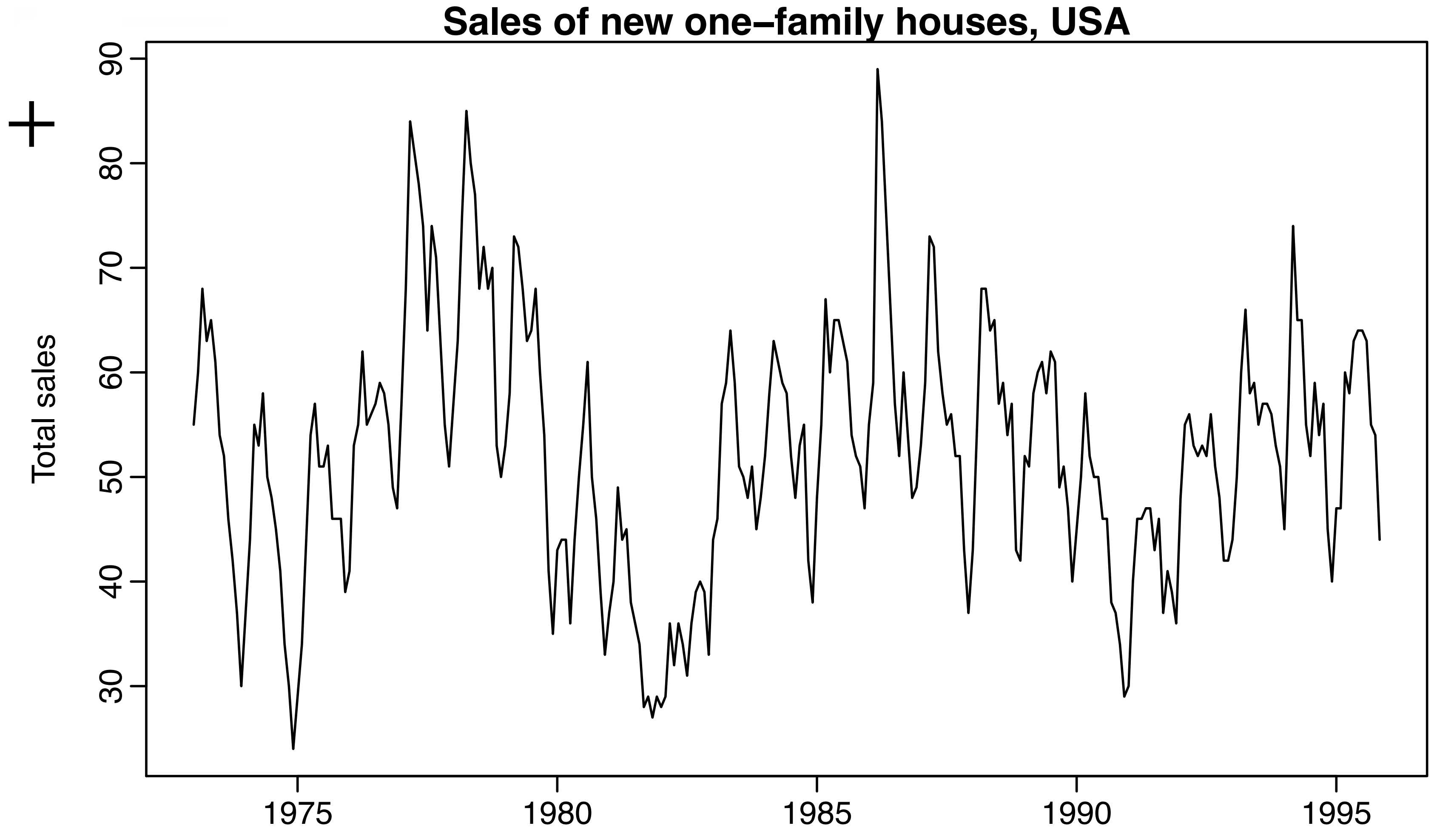
Trend

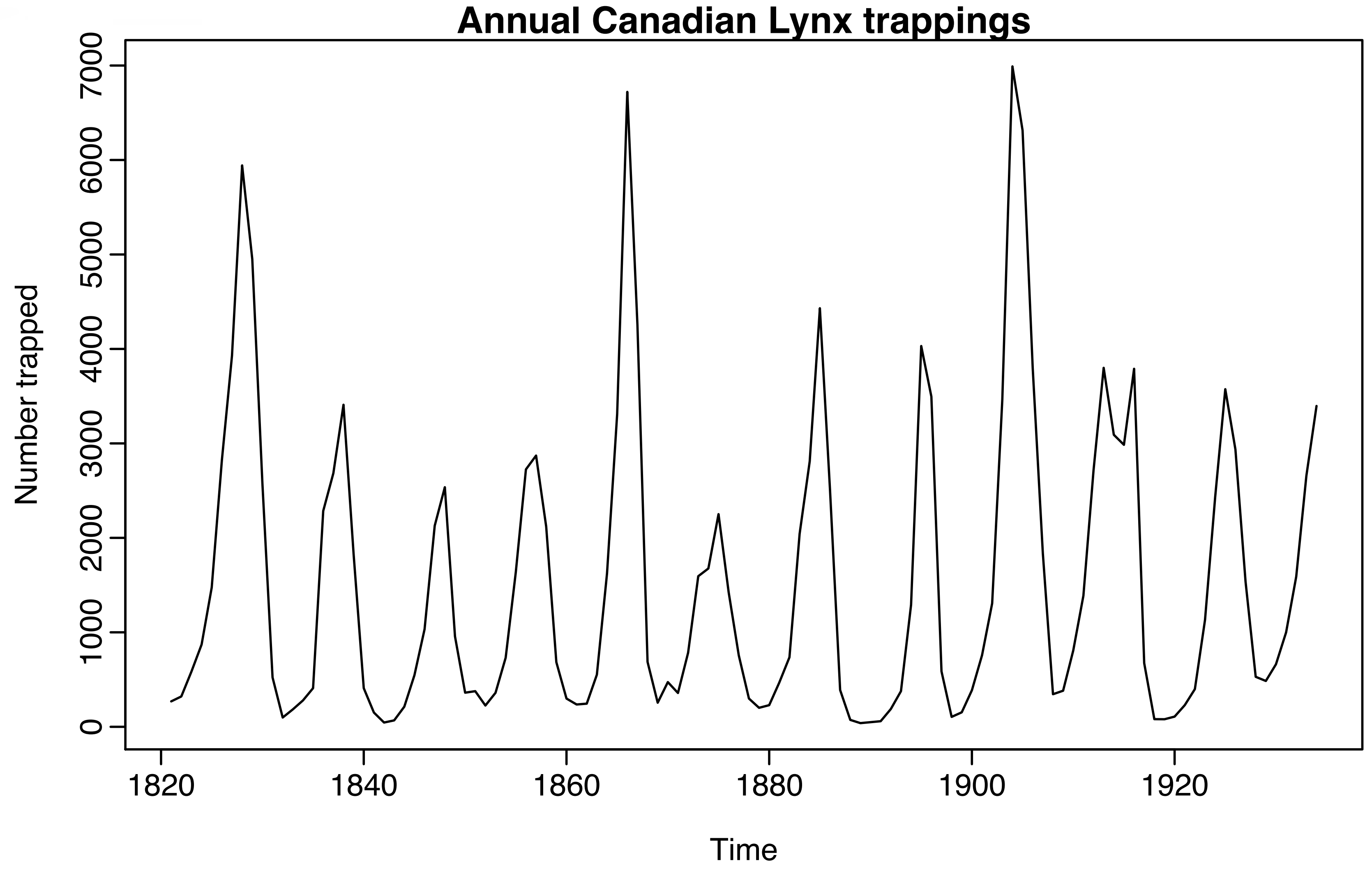




Trend + Seasonality

Seasonality + Cyclic





Stationary

[R. J. Hyndman]

Dates and Times

- What is time to a computer?
  - Can be stored as seconds since Unix Epoch (January 1st, 1970)
- Often useful to break down into minutes, hours, days, months, years…
- Lots of different ways to write time:
  - How could you write `"November 29, 2016"`?
  - European vs. American ordering…
- What about time zones?

DatetimeIndex

- Can use time as an **index**
- ```
  data = [('2017-11-30', 48),
          ('2017-12-02', 45),
          ('2017-12-03', 44),
          ('2017-12-04', 48)]
  dates, temps = zip(*data)
  s = pd.Series(temps, pd.to_datetime(dates))
  ```
- Accessing a particular time or checking equivalence allows any string that can be interpreted as a date:
  - `s['12/04/2017']` or `s['20171204']`
- Using a less specific string will get all matching data:
  - `s['2017-12']` returns the three December entries
  ```

Timedelta

- Compute differences between dates
- Lives in `datetime` module
- `diff = parse_date("1 Jan 2017") - datetime.now().date()`
  `diff.days`
- Also a `pd.Timedelta` object that take strings:
  - `datetime.now().date() + pd.Timedelta("4 days")`
- Also, Roll dates using anchored offsets

```
from pandas.tseries.offsets import Day, MonthEnd

now = datetime(2011, 11, 17)
In [107]: now + MonthEnd(2)
Out[107]: Timestamp('2011-12-31 00:00:00')
```

Time Zones

- Why?
- Coordinated Universal Time (UTC) is the standard time (basically equivalent to Greenwich Mean Time (GMT)
- Other time zones are UTC +/- a number in [1,12]
- Dekalb is UTC-6 (aka US/Central)

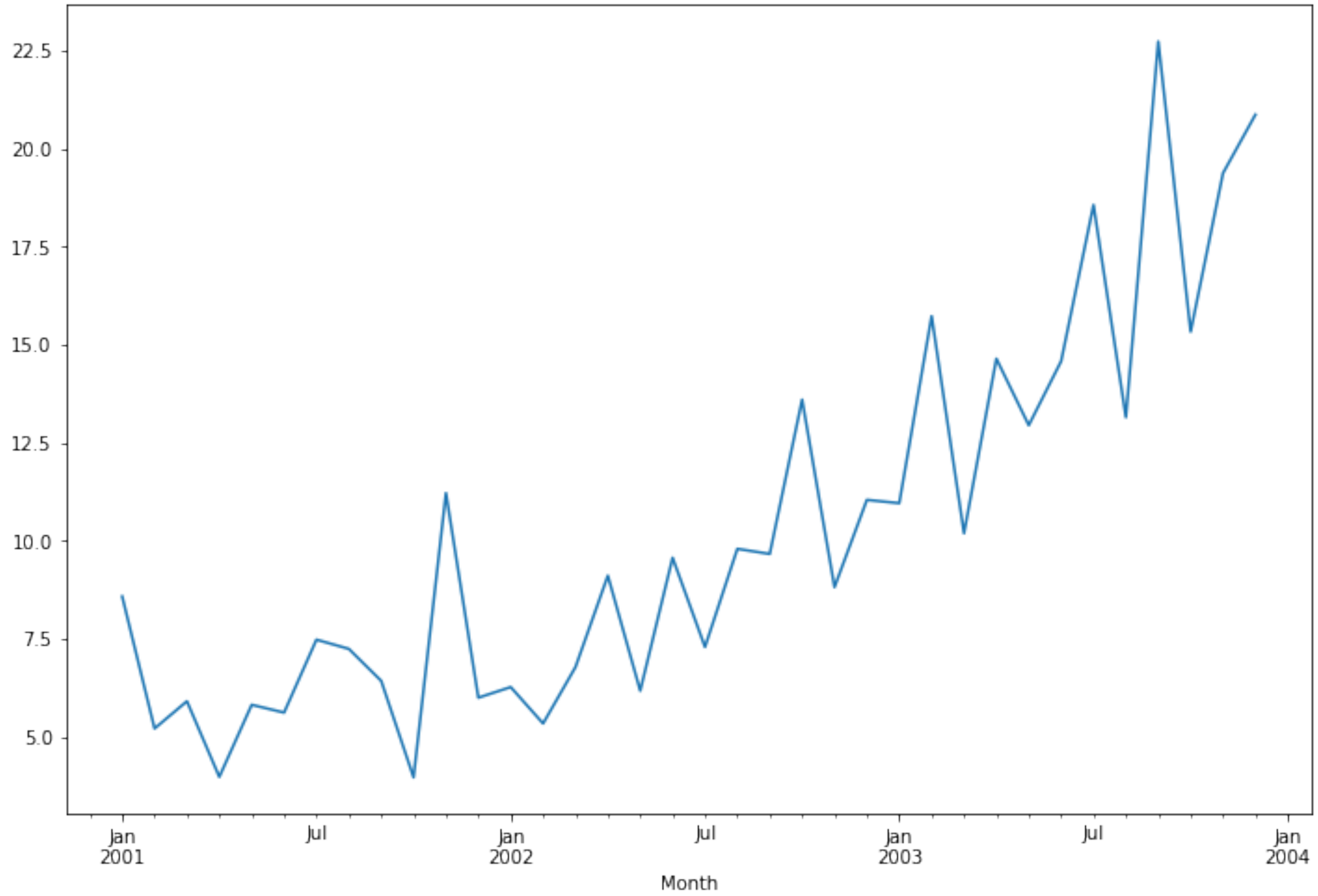
Resampling

- Could be
  - downsample: higher frequency to lower frequency
  - upsample: lower frequency to higher frequency
  - neither: e.g. Wednesdays to Fridays
- resample method: e.g. `ts.resample('M').mean()`

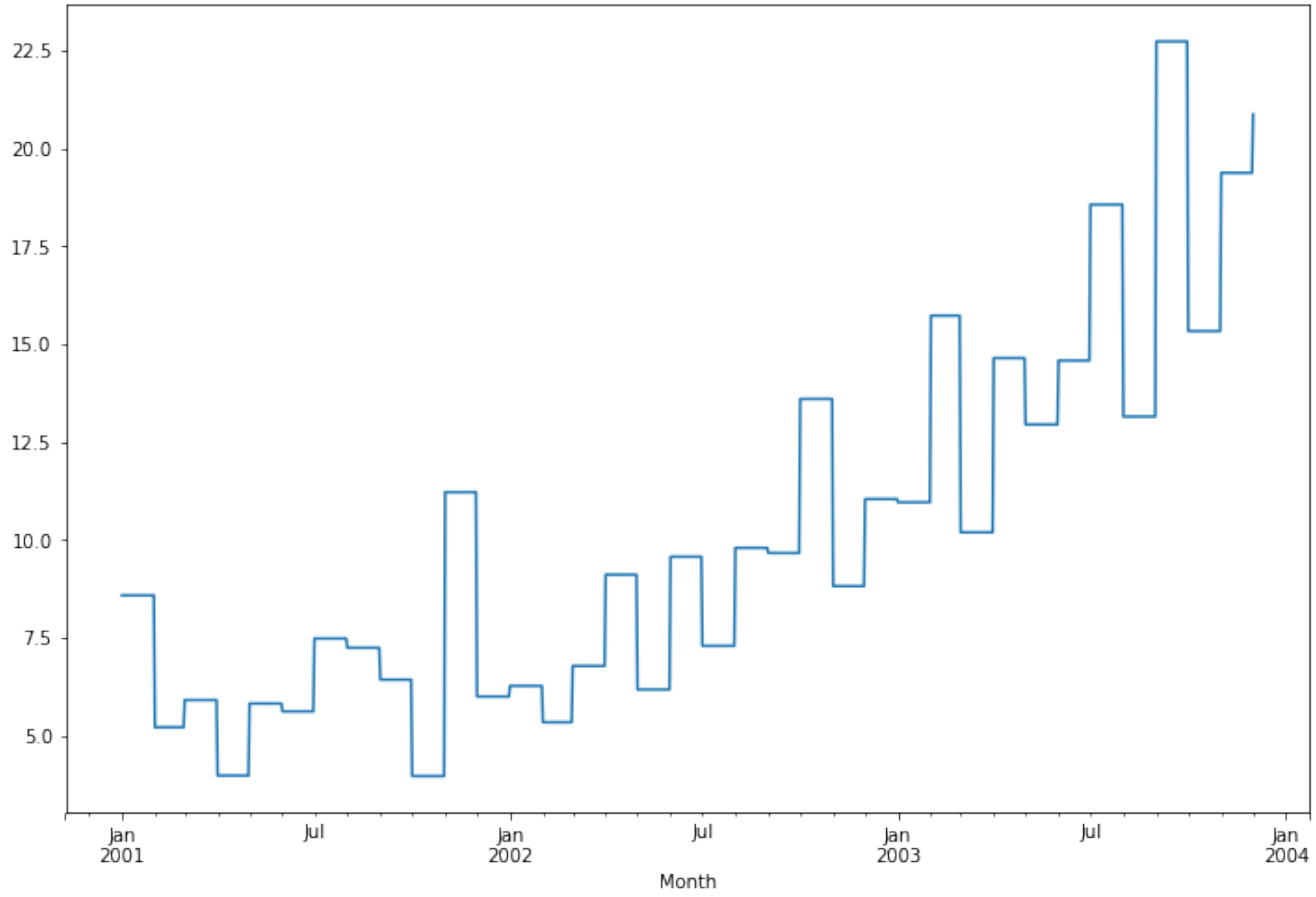
Argument	Description
`freq`	String or DateOffset indicating desired resampled frequency (e.g., 'M', '5min', or `Second(15)`)
`axis`	Axis to resample on; default axis=0
`fill_method`	How to interpolate when upsampling, as in `'ffill'` or `'bfill'`; by default does no interpolation
`closed`	In downsampling, which end of each interval is closed (inclusive), `'right'` or `'left'`
`label`	In downsampling, how to label the aggregated result, with the `'right'` or `'left'` bin edge (e.g., the 9:30 to 9:35 five-minute interval could be labeled `9:30` or `9:35`)
`loffset`	Time adjustment to the bin labels, such as `'-1s'` / `Second(-1)` to shift the aggregate labels one second earlier
`limit`	When forward or backward filling, the maximum number of periods to fill
`kind`	Aggregate to periods (`'period'`) or timestamps (`'timestamp'`); defaults to the type of index the time series has
`convention`	When resampling periods, the convention (`'start'` or `'end'`) for converting the low-frequency period to high frequency; defaults to `'end'`

[W. McKinney, Python for Data Analysis]

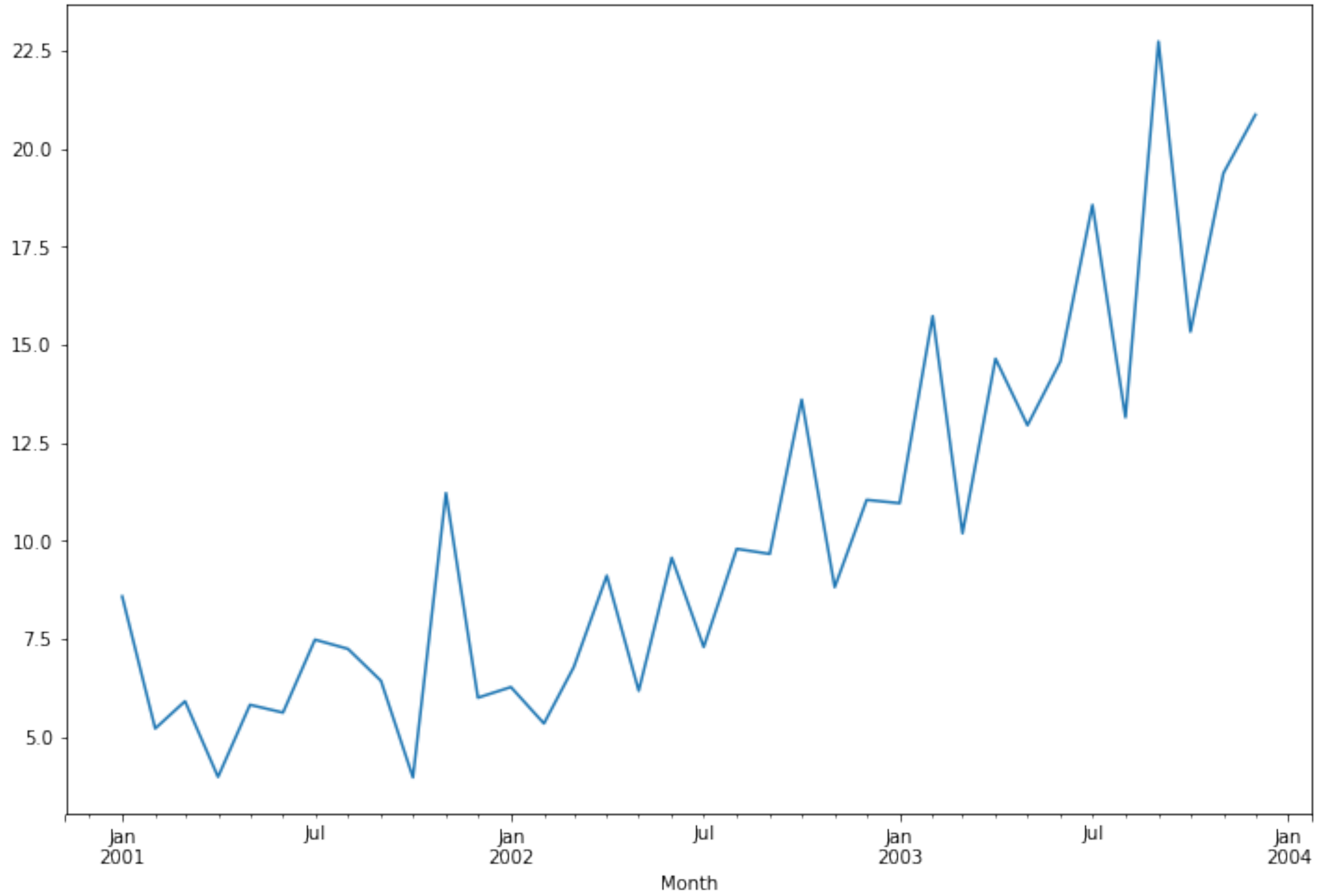
Sales Data by Month



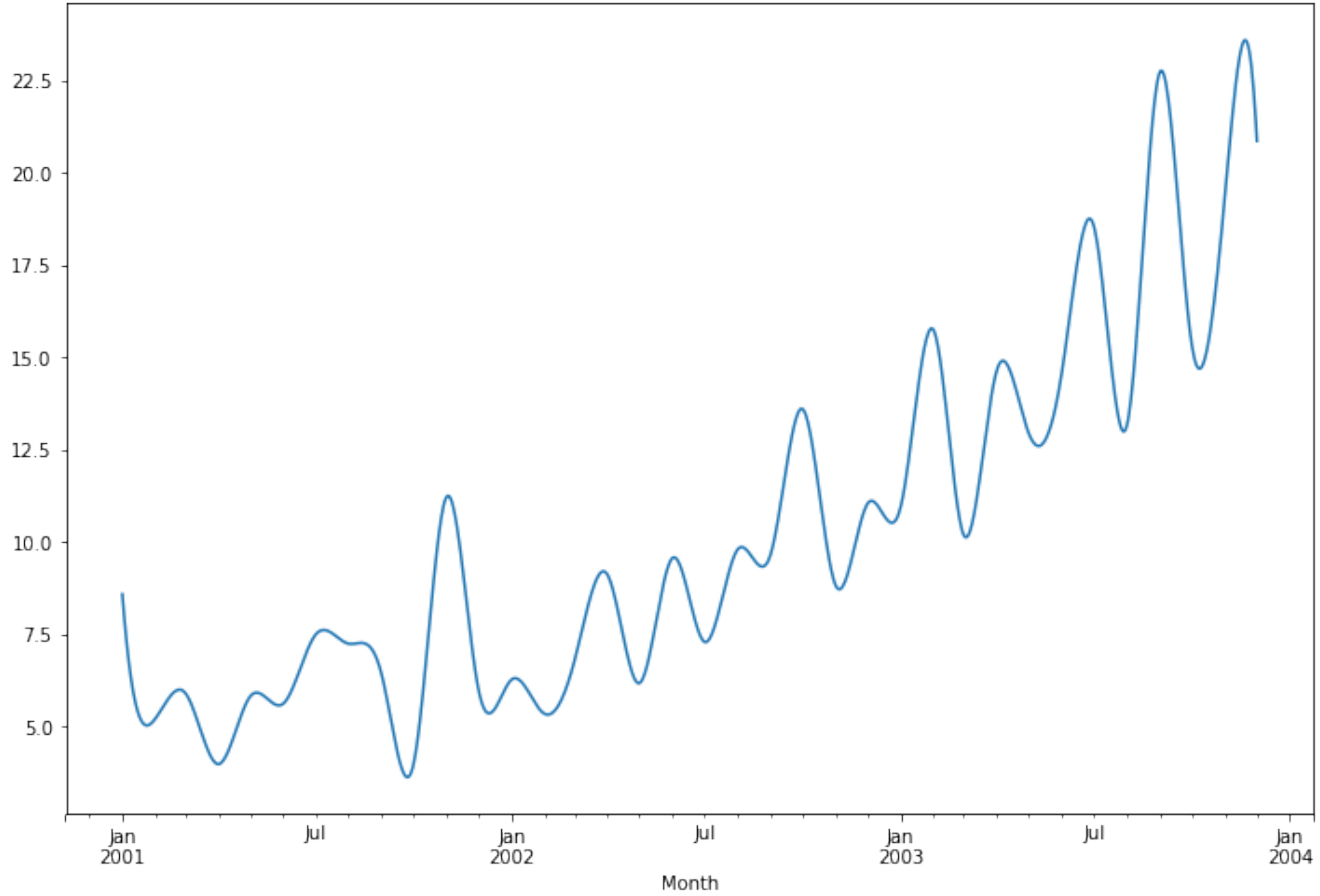
Resampled Sales Data (ffill)



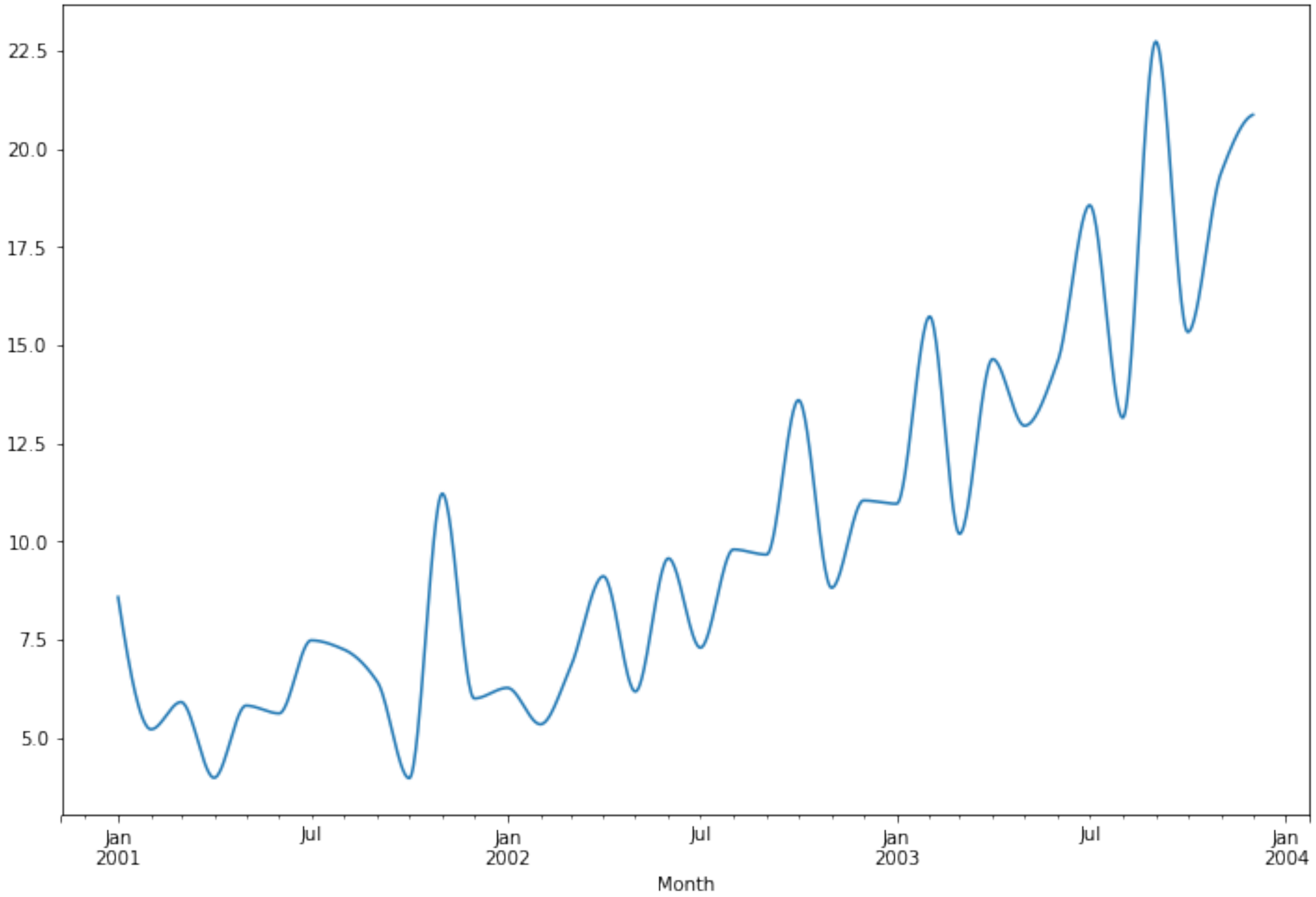
Resampled with Linear Interpolation (Default)



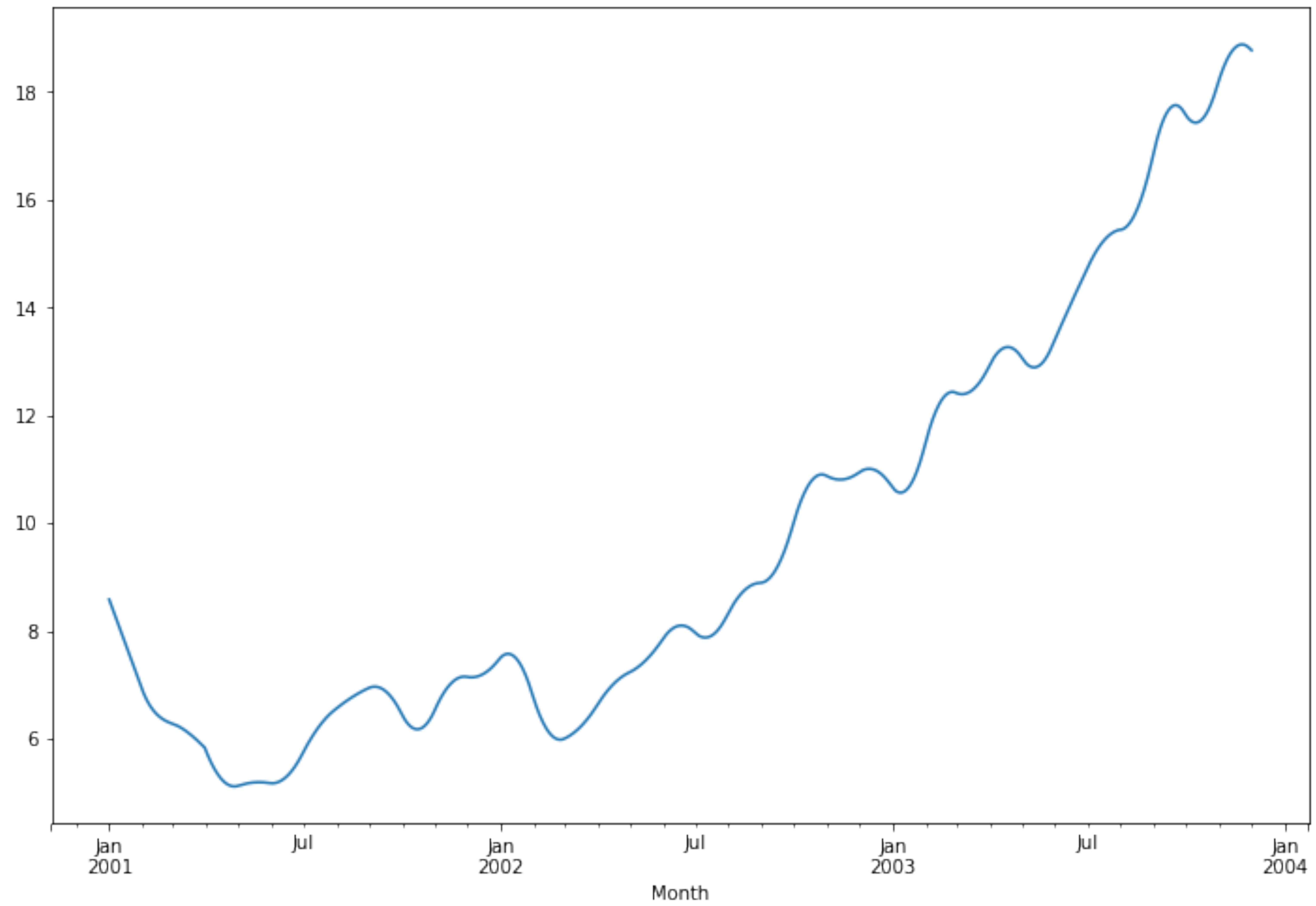
Resampled with Cubic Interpolation



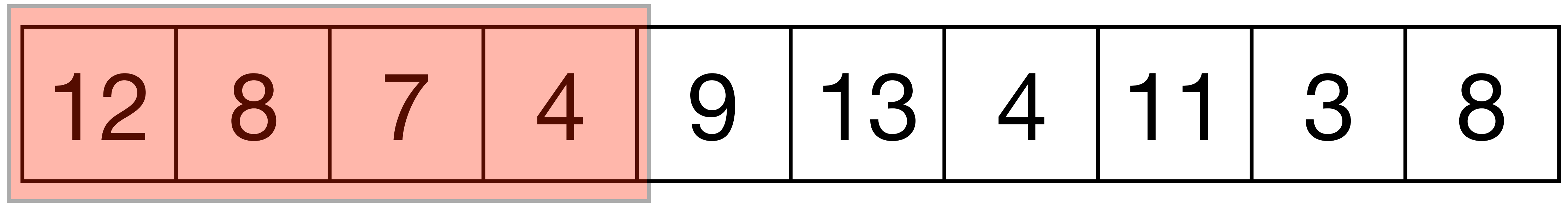
Piecewise Cubic Hermite Interpolating Polynomial



90-Day Rolling Window (Mean)



Rolling Window Calculations



Rolling Window Calculations

12	8	7	4	9	13	4	11	3	8

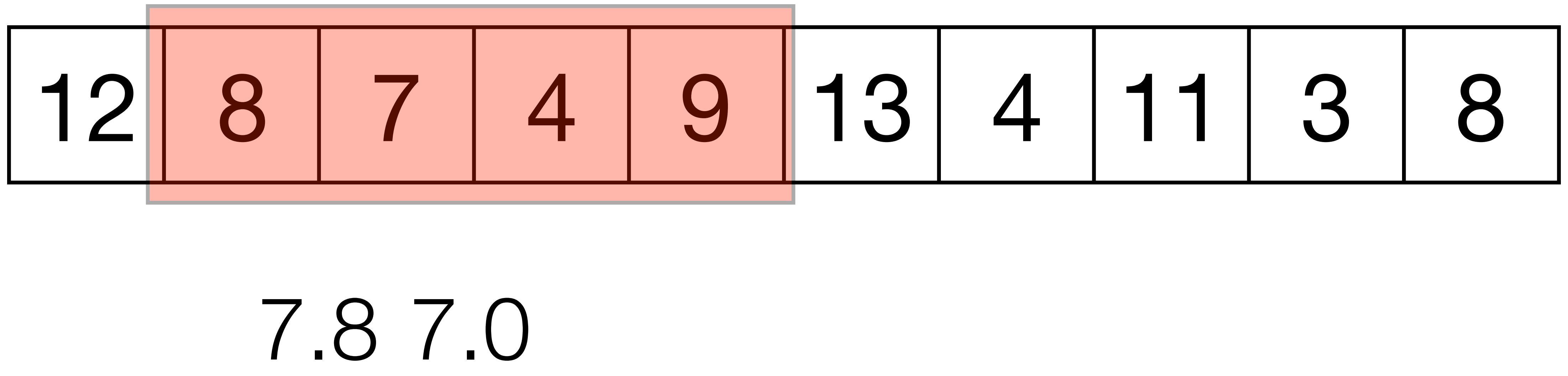
7.8

Rolling Window Calculations

12	8	7	4	9	13	4	11	3	8

7.8

Rolling Window Calculations



Rolling Window Calculations

12	8	7	4	9	13	4	11	3	8

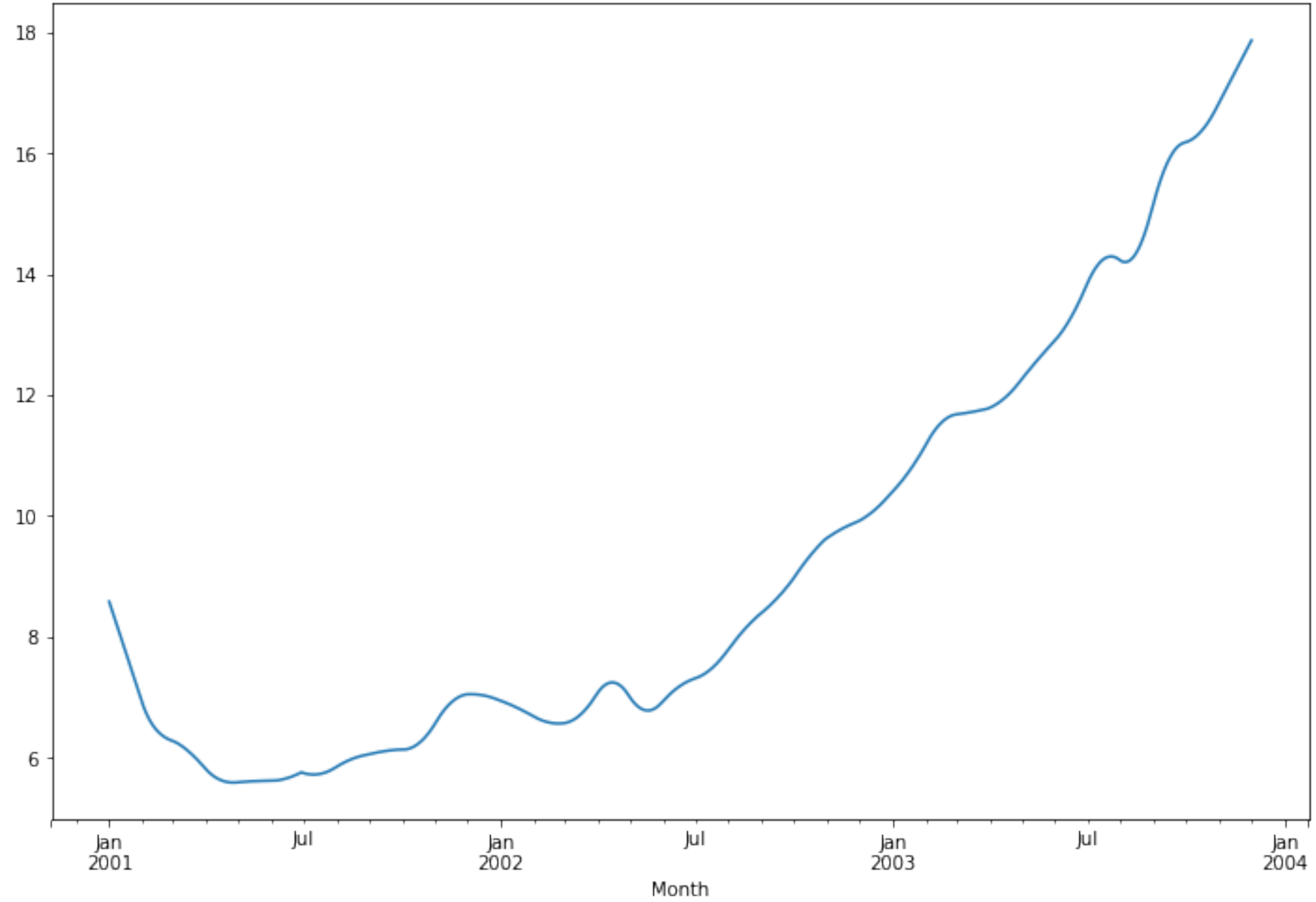
7.8 7.0

Rolling Window Calculations

12	8	7	4	9	13	4	11	3	8

7.8 7.0 8.3

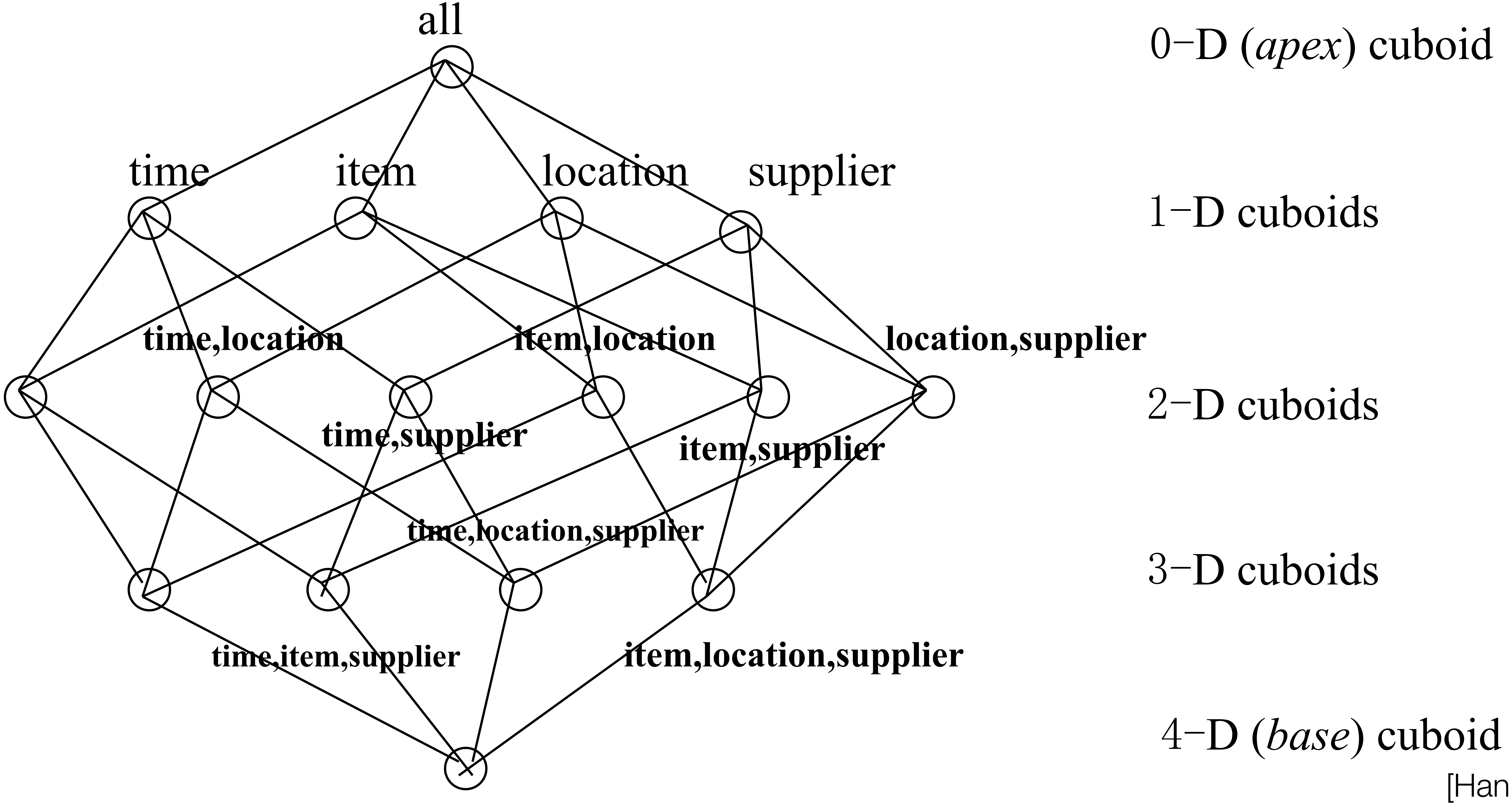
180-Day Rolling Window (Mean)



Data Cubes

J. Han, M. Kamber, and J. Pei

Data Cube: A Lattice of Cuboids



[Han et al., 2011]

Cube Operations

- Roll-up: aggregate up the given hierarchy
- Drill-down: refine down the given hierarchy
- Roll-up and drill-down are "inverses"

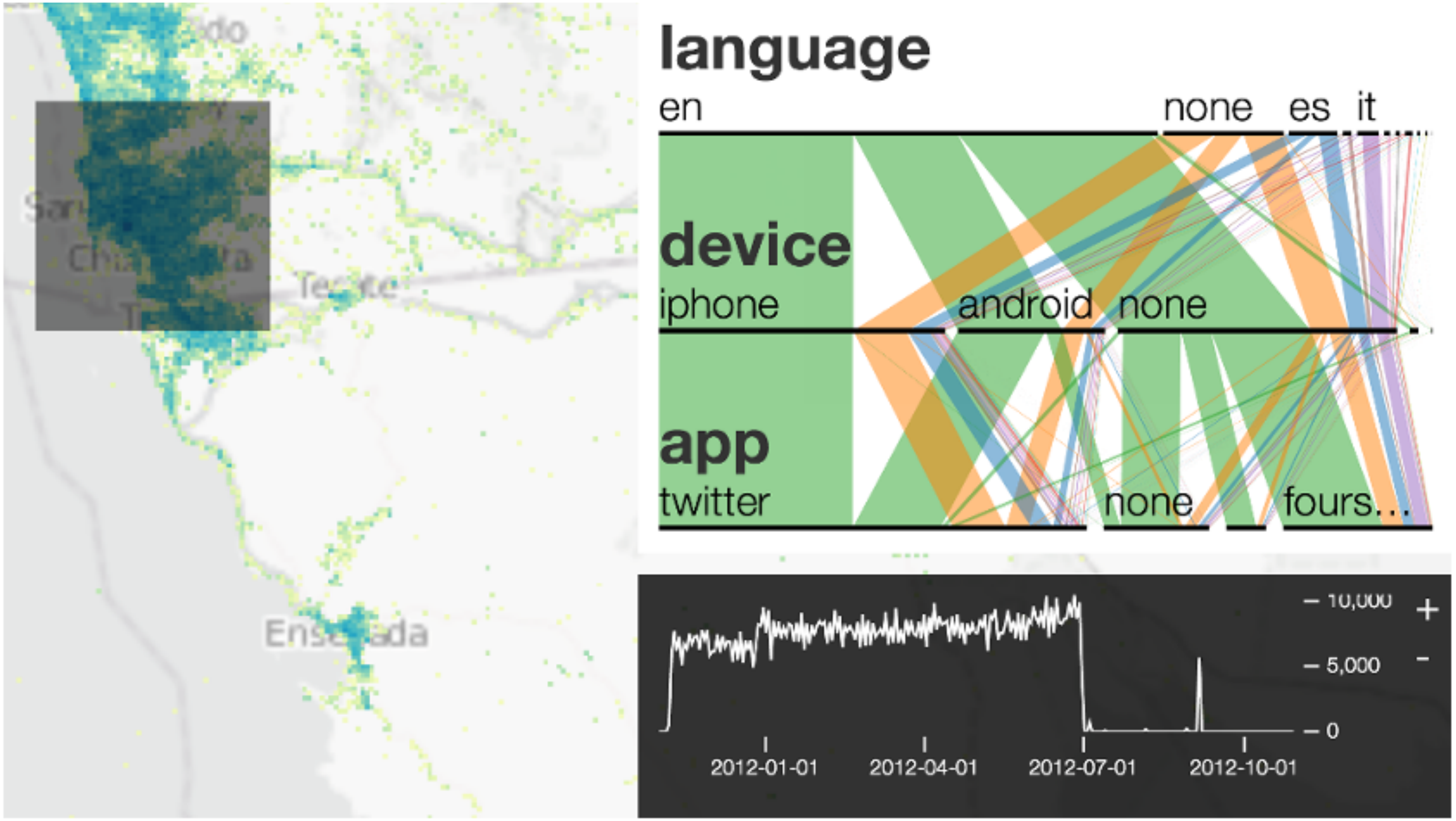
Spatial Data Exploration Motivation

L. Battle

Nanocubes for Real-Time Exploration of Spatiotemporal Datasets

L. Lins, J. T. Klosowski, and C. Scheidegger

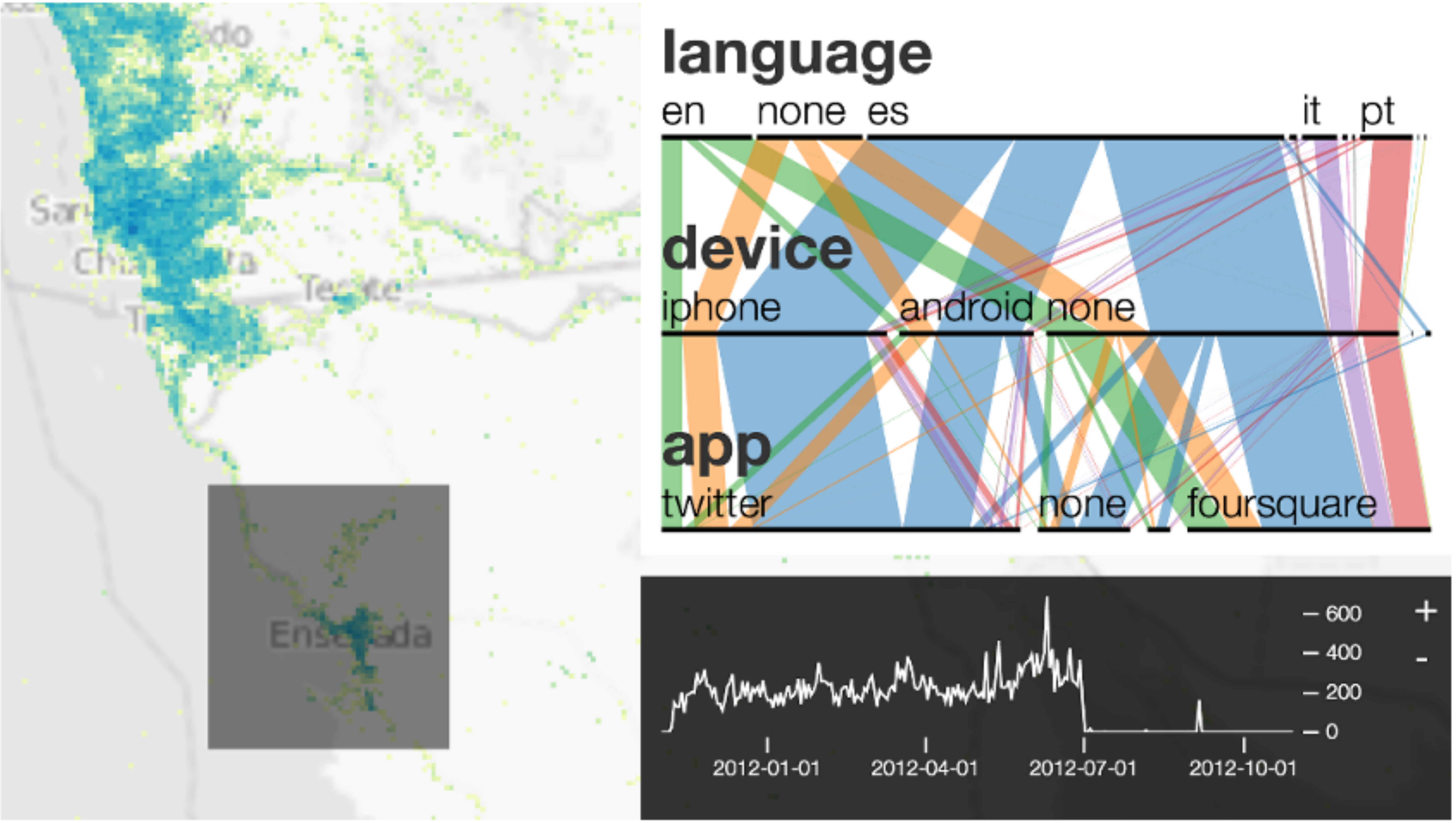
Goal: Interactive Exploration of Data Cubes



Linked view of tweets in San Diego, US

[Lins et. al, 2013]

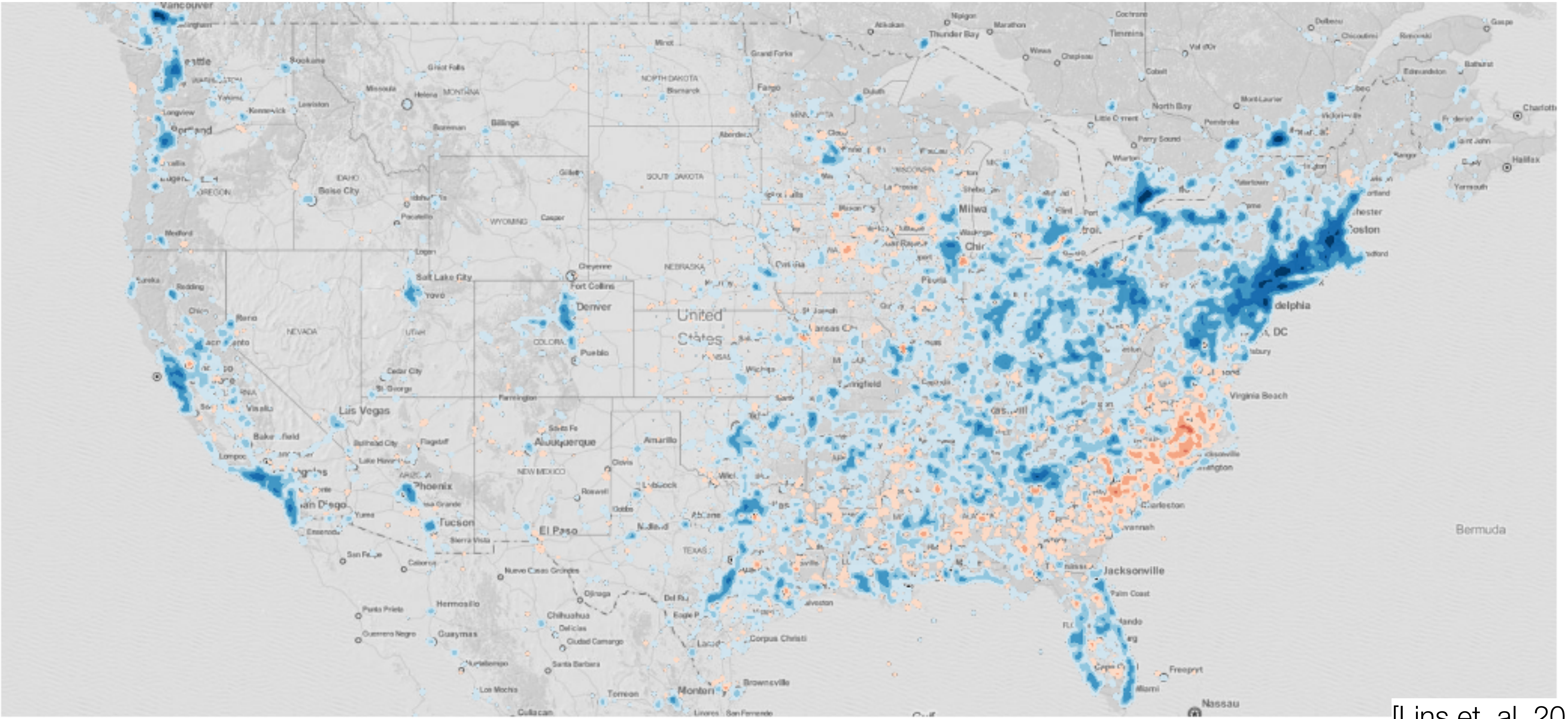
Move to Another Location



Linked view of tweets in Ensenada, Mexico

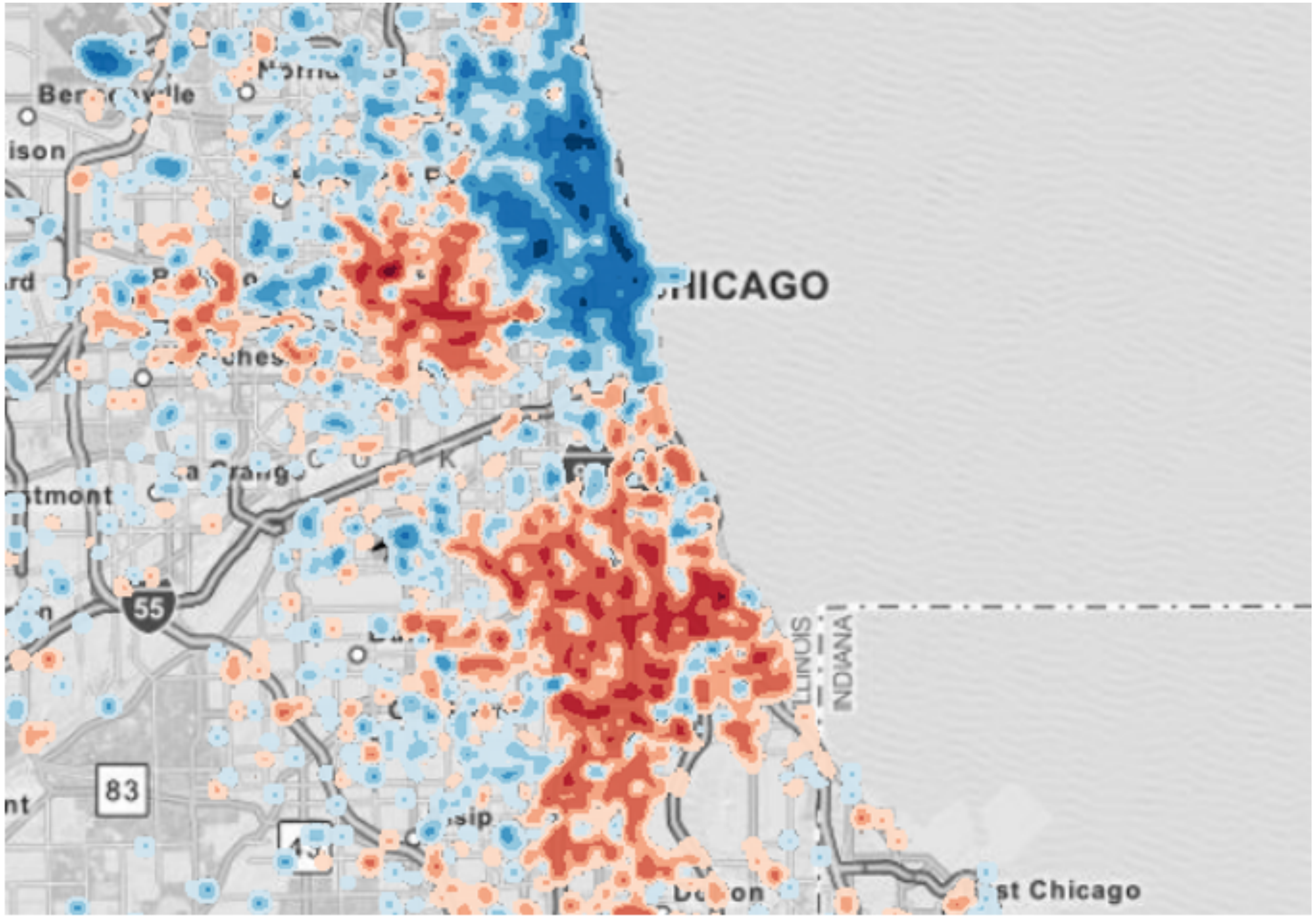
[Lins et. al, 2013]

iPhone vs. Android Map



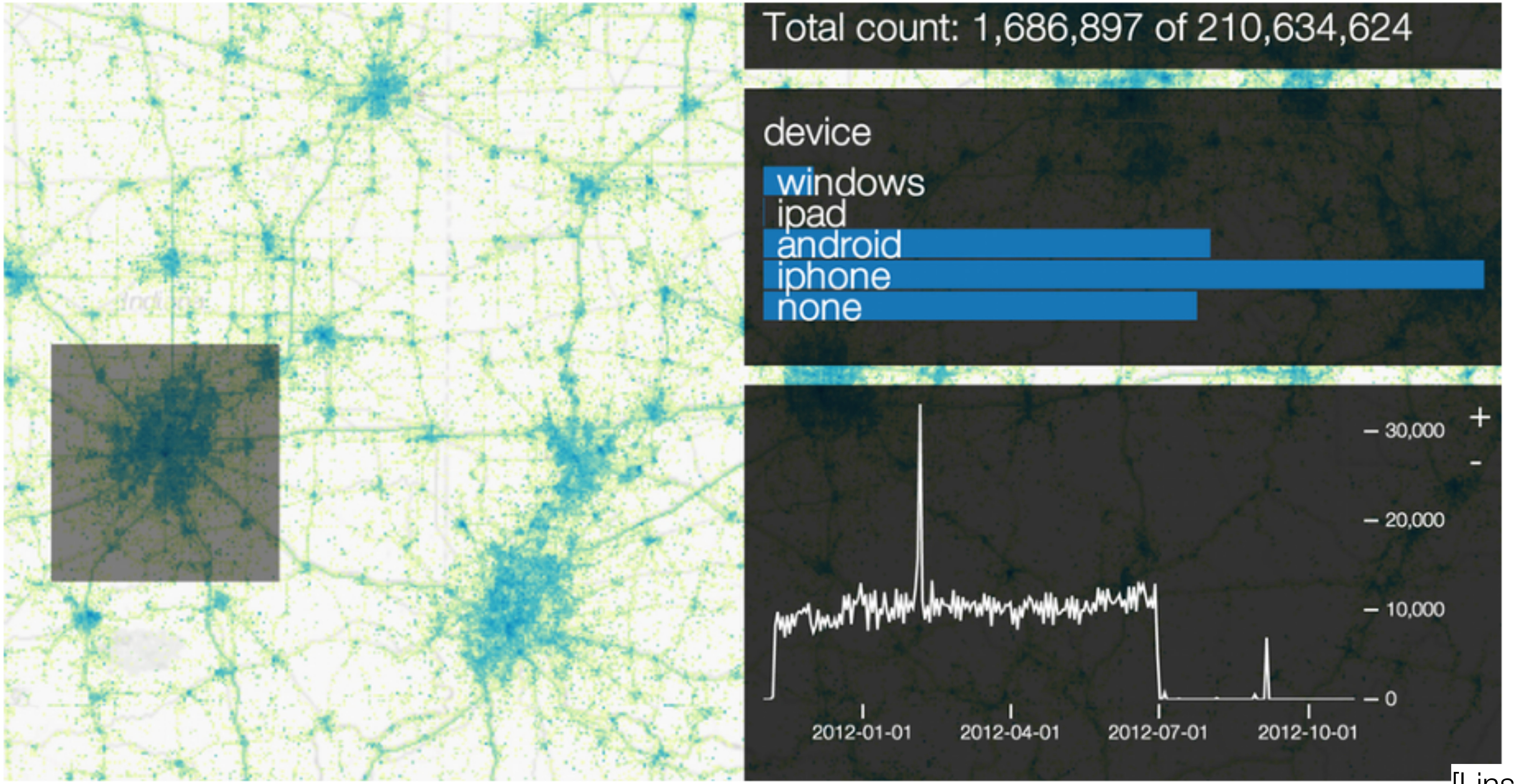
[Lins et. al, 2013]

Zoom into Chicago



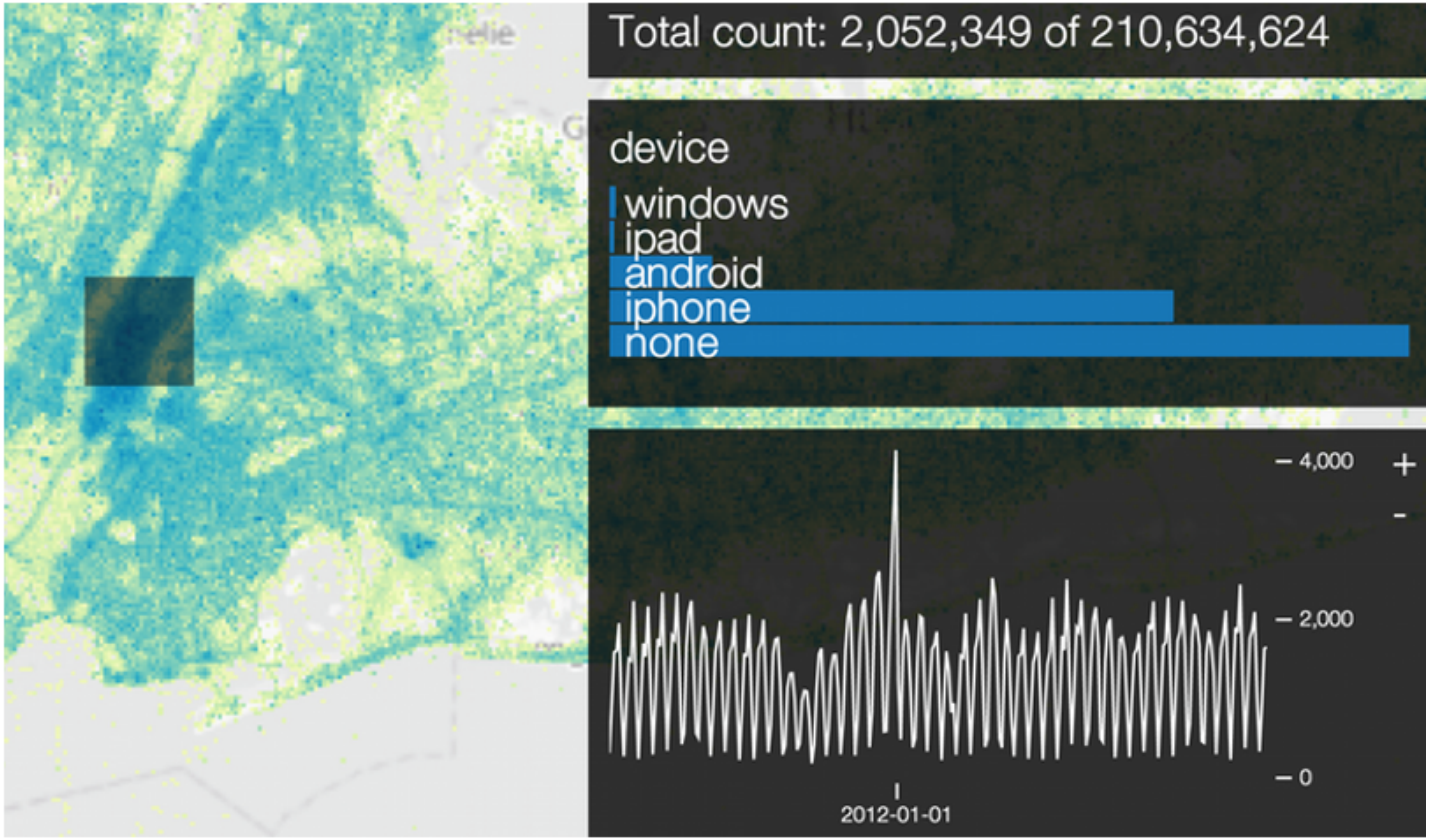
[Lins et. al, 2013]

SuperBowl in Indianapolis



[Lins et. al, 2013]

New Year's Eve in Manhattan



[Lins et. al, 2013]

Aggregations on Spatiotemporal Data

- Spatial: e.g. counting events in a spatial region (world or San Fran.)
- Temporal: e.g. queries at multiple scares (hour, day, week, month)
- Seek to address Visual Information Seeking Mantra:
- Overview first, zoom and filter, details-on-demand
- Multidimensional:
  - Latitude, Longitude, Time + **more**

Data Cube Aggregations

Relation A

Country	*Device*	*Language*
US	Android	en
US	iPhone	ru
South Africa	iPhone	en
India	Android	en
Australia	iPhone	en

Aggregation B

Country	*Device*	*Language*	*Count*
All	*All*	*All*	5

Group By on *Device, Language* C

Country	*Device*	*Language*	*Count*
All	Android	en	2
All	iPhone	en	2
All	iPhone	ru	1

Cube on *Device, Language* D

Country	*Device*	*Language*	*Count*
All	*All*	*All*	5
All	Android	*All*	2
All	iPhone	*All*	3
All	*All*	en	4
All	*All*	ru	1
All	iPhone	ru	1
All	Android	en	2
All	iPhone	en	2

Equivalent to Group By on all possible subsets of {*Device, Language*}

[Lins et. al, 2013]

Nanocube Queries

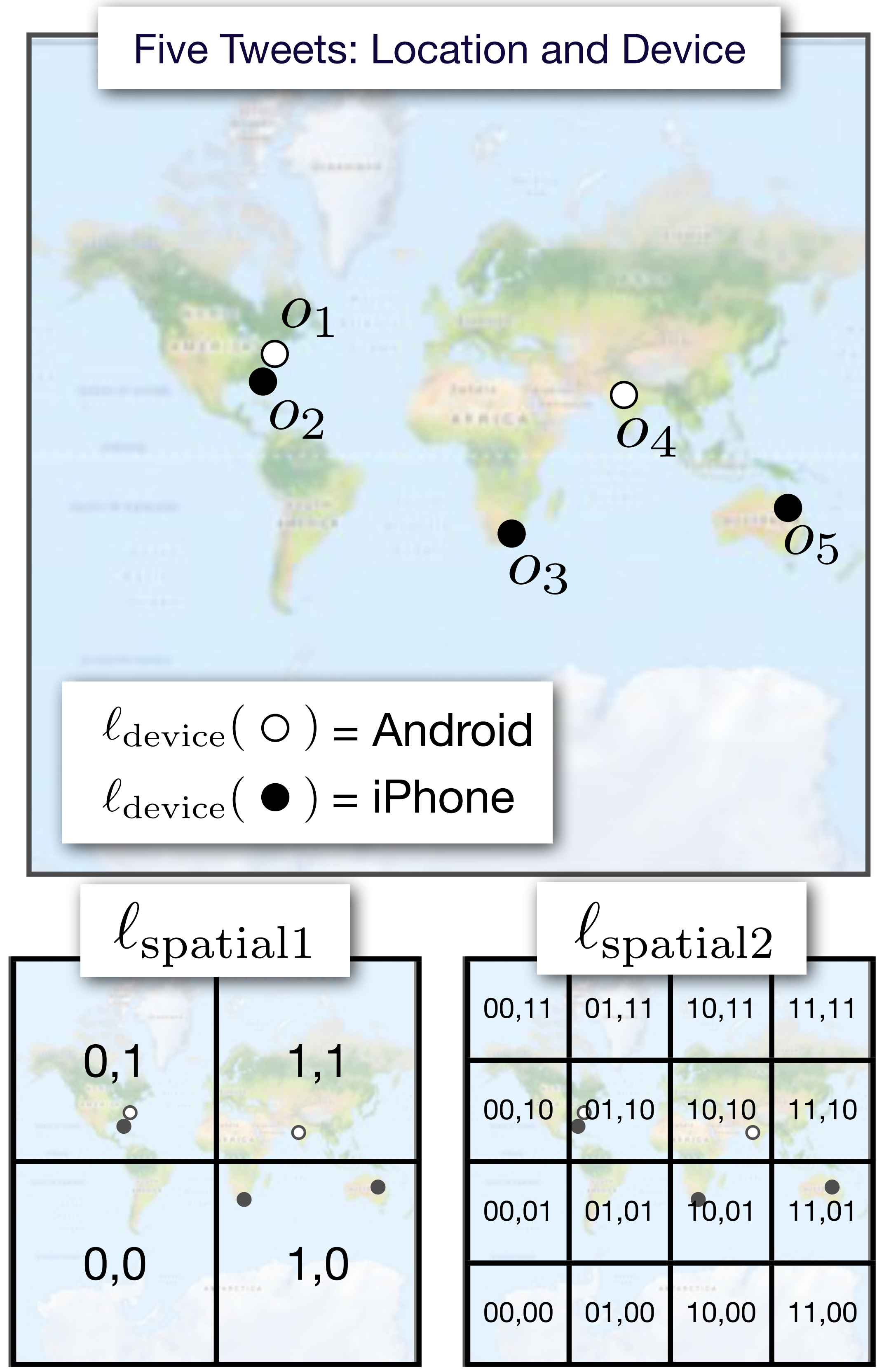
- Representing natural language queries as data cube queries

Natural language query	s		c		t		URL
count of all Delta flights	R	U	R	{ Delta }	R	U	/where/carrier=Delta
count of all Delta flights in the Midwest	R	Midwest	R	{ Delta }	R	U	/region/Midwest/where/carrier=Delta
count of all flights in 2010	R	U	D		R	2010	/field/carrier/when/2010
time-series of all United flights in 2009	R	U	R	{ United }	D	2009	/tseries/when/2009/where/carrier=United
heatmap of Delta flights in 2010	D	tile0	R	{ Delta }	R	2010	/tile/tile0/when/2010/where/carrier=Delta

- s = space, c = category, t = time
- R = rollup, D = drill down
- <value> after RD = subset of dimension's domain, U = universe
- Note that time queries are stored in an array of cumulative counts

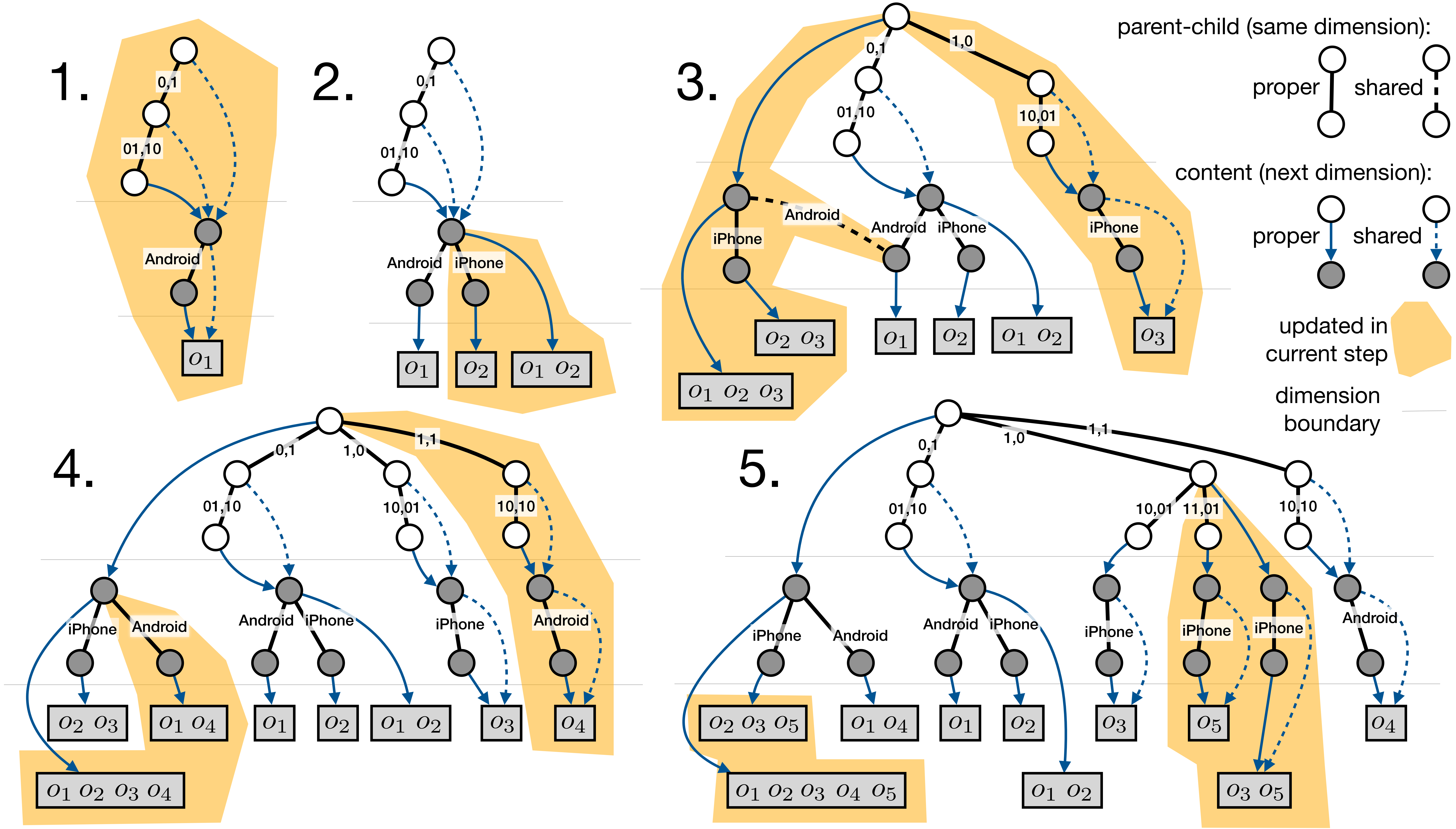
[Lins et. al, 2013]

Building a Nanocube



Indexing Schema

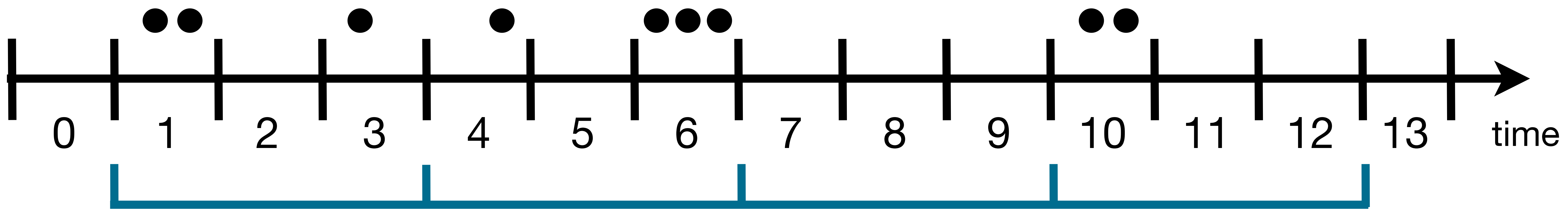
$$S = [\,[\ell_{\text{spatial1}}, \ell_{\text{spatial2}}], [\ell_{\text{device}}]\,]$$



[Lins et. al, 2013]

Summed-area Table

- Every node in the previous figure stores an array of timestamped counts like this:



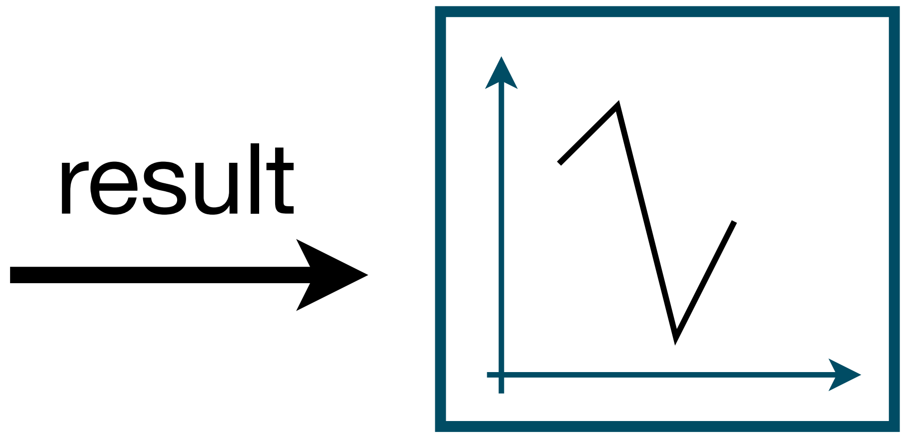
query/tseries/1/3/4

start at bin 1, use buckets of 3 bins each, and collect 4 of these buckets

solve using...

bin: 1 accum: 2	bin: 3 accum: 3	bin: 4 accum: 4	bin: 6 accum: 7	bin: 10 accum: 9

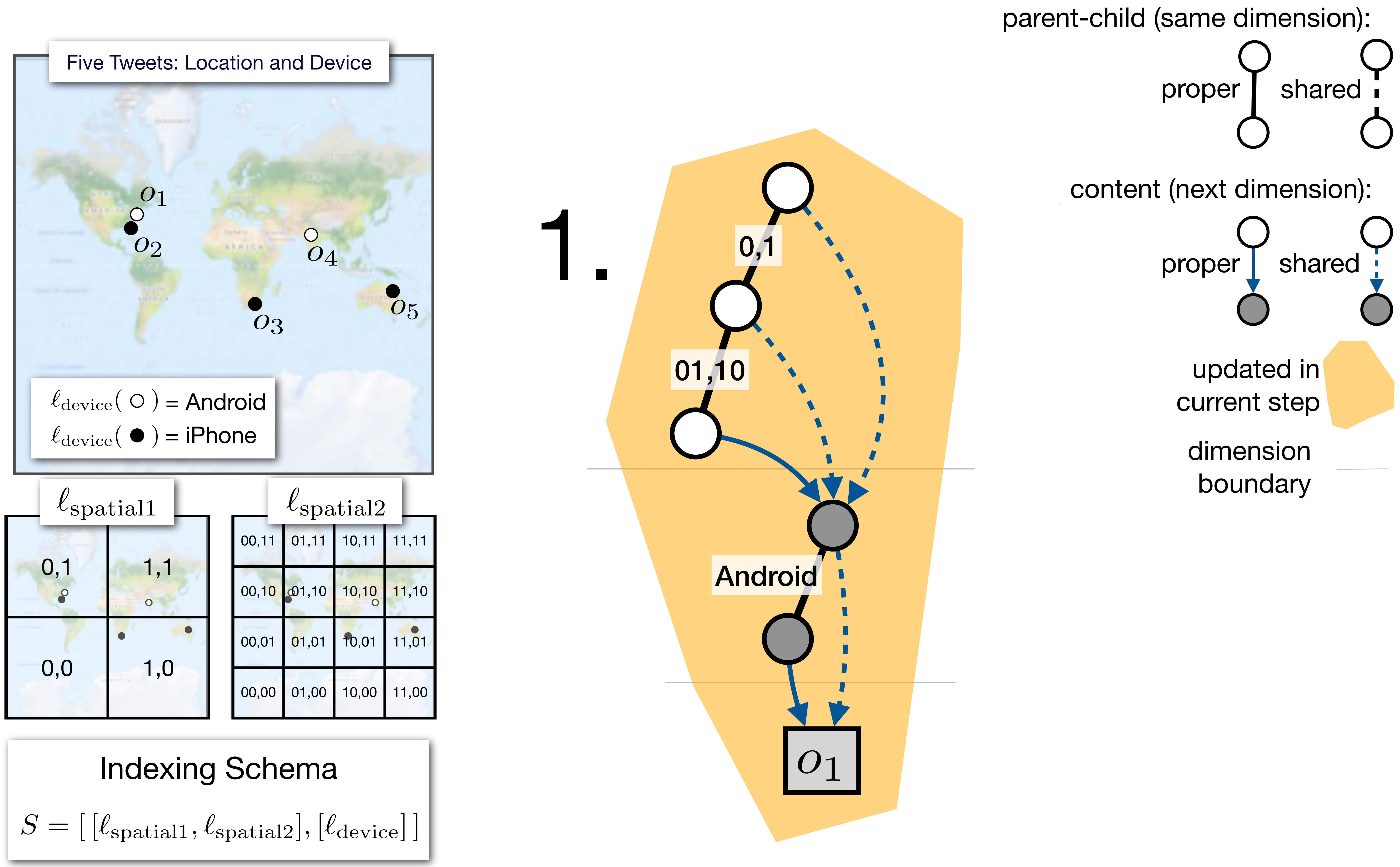
A Summed Table Sparse Representation for Counts



3	4	0	2

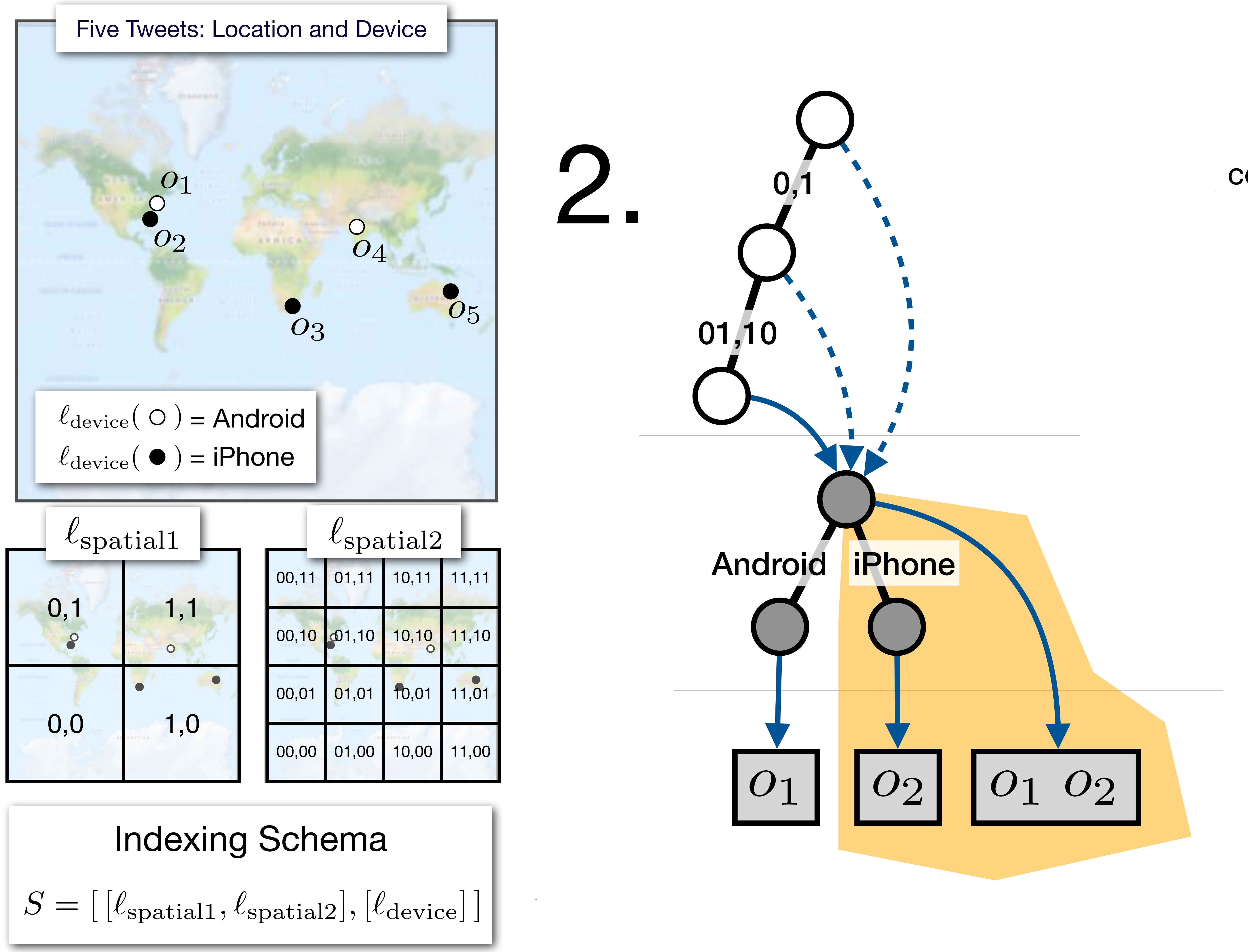
[Lins et. al, 2013]

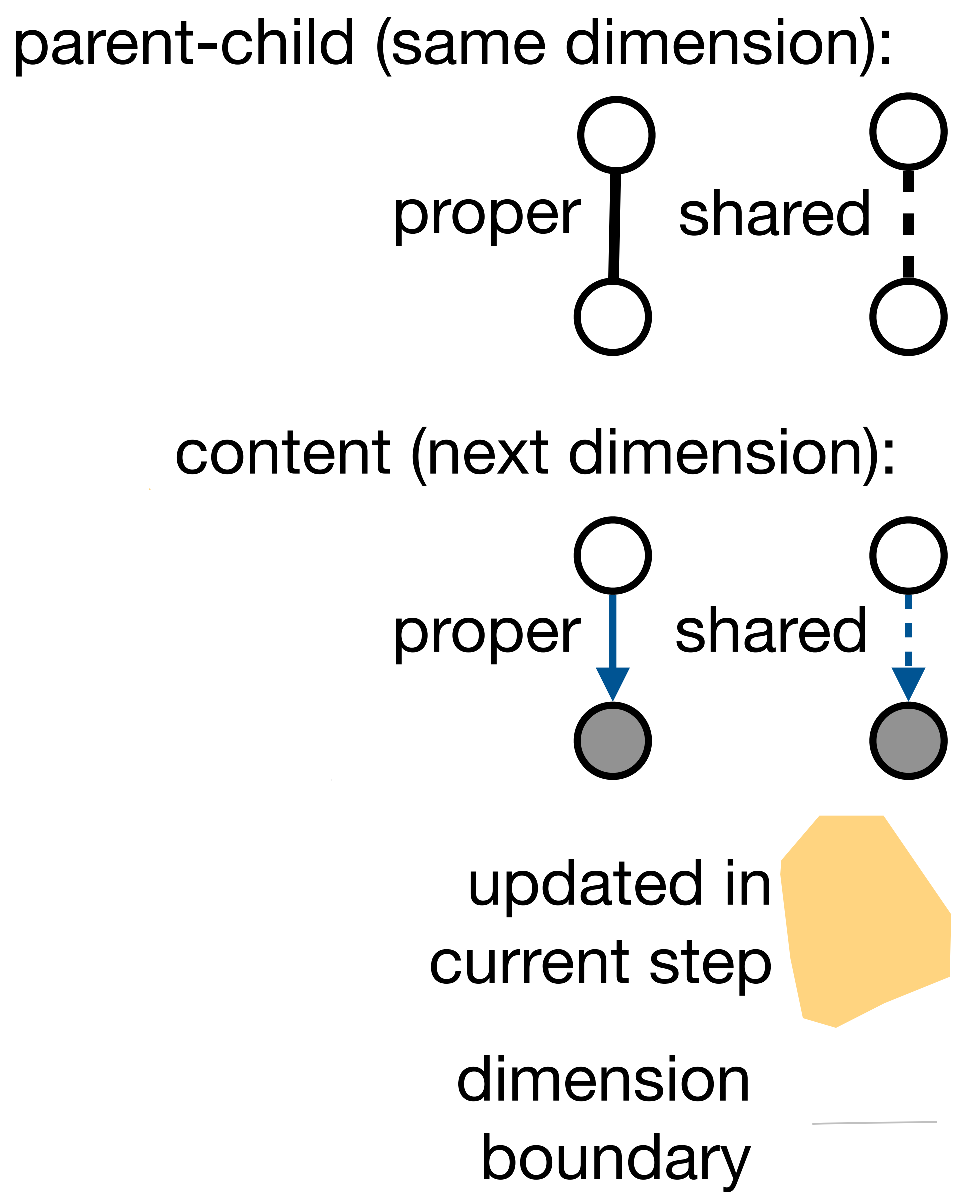
Building a Nanocube: Step 1



[Lins et. al, 2013]

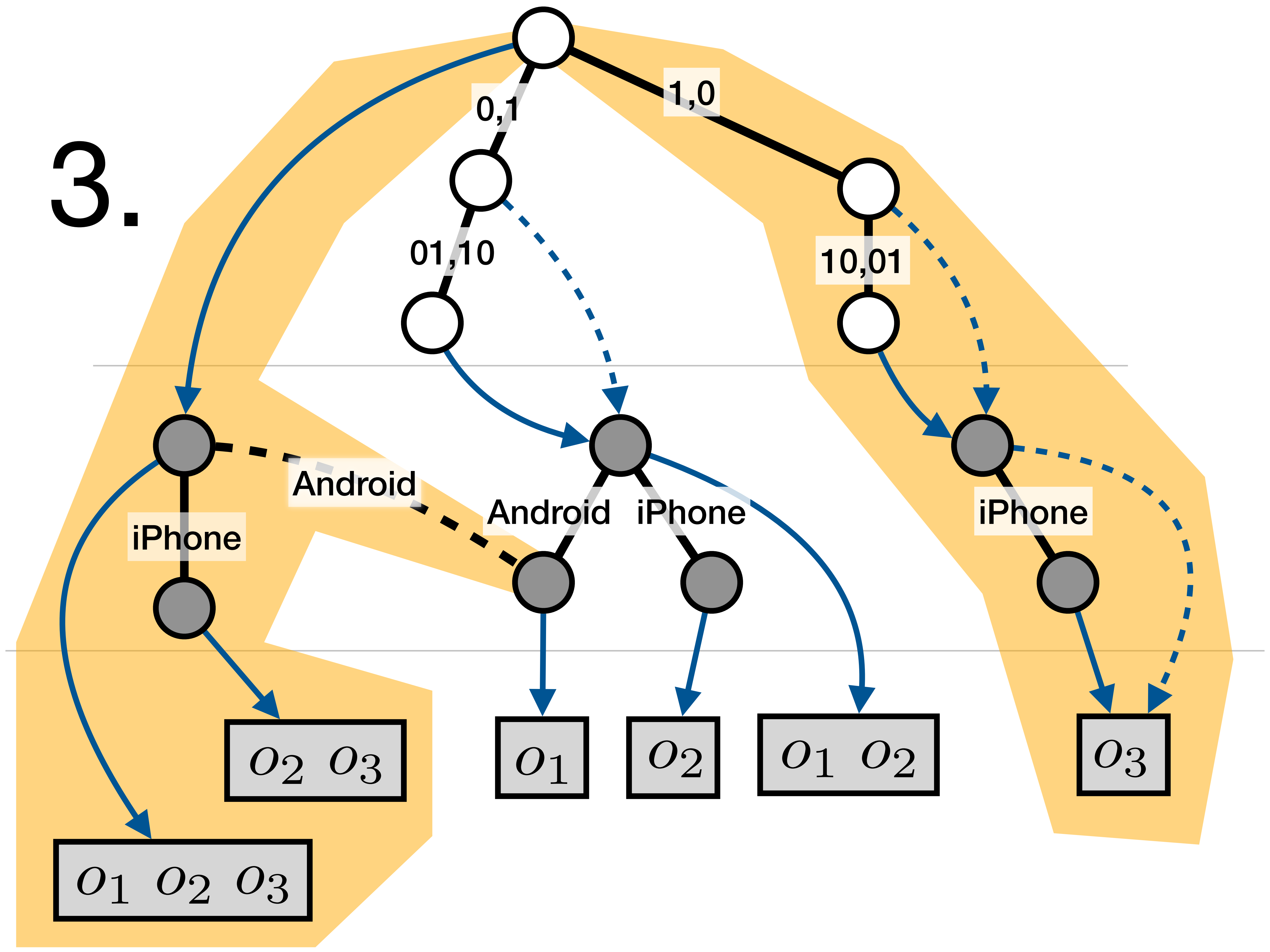
Building a Nanocube: Step 2

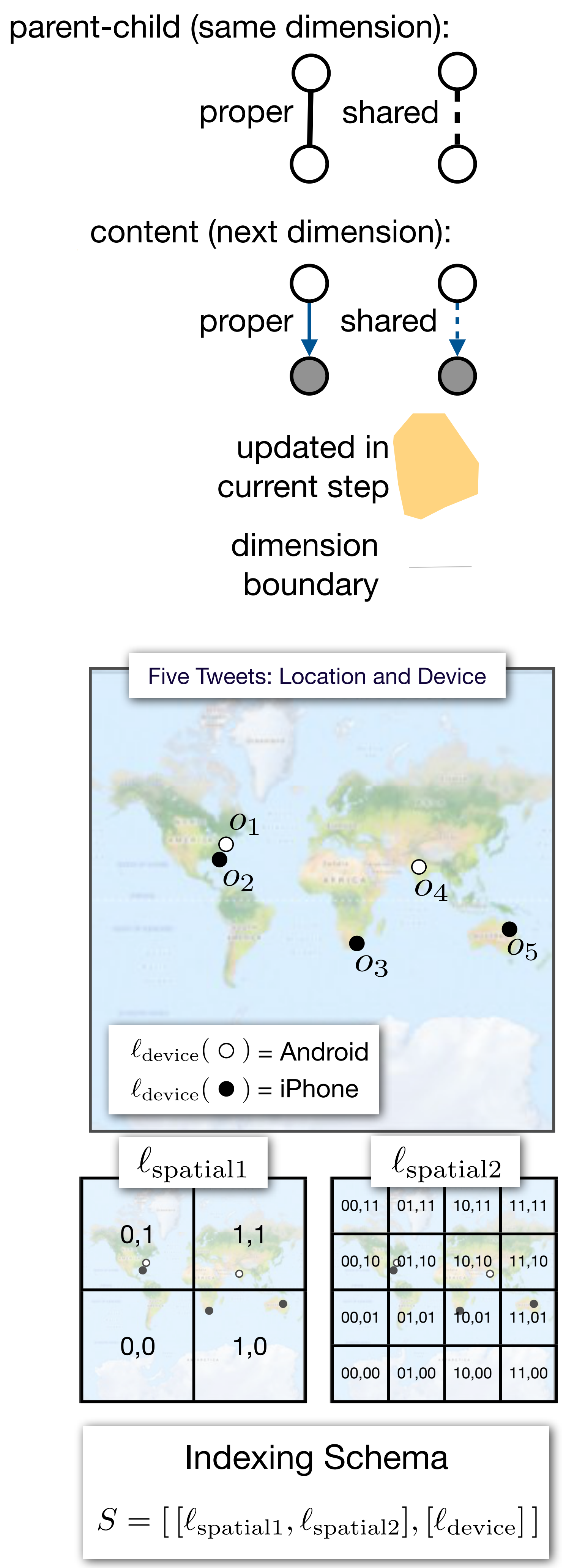




[Lins et. al, 2013]

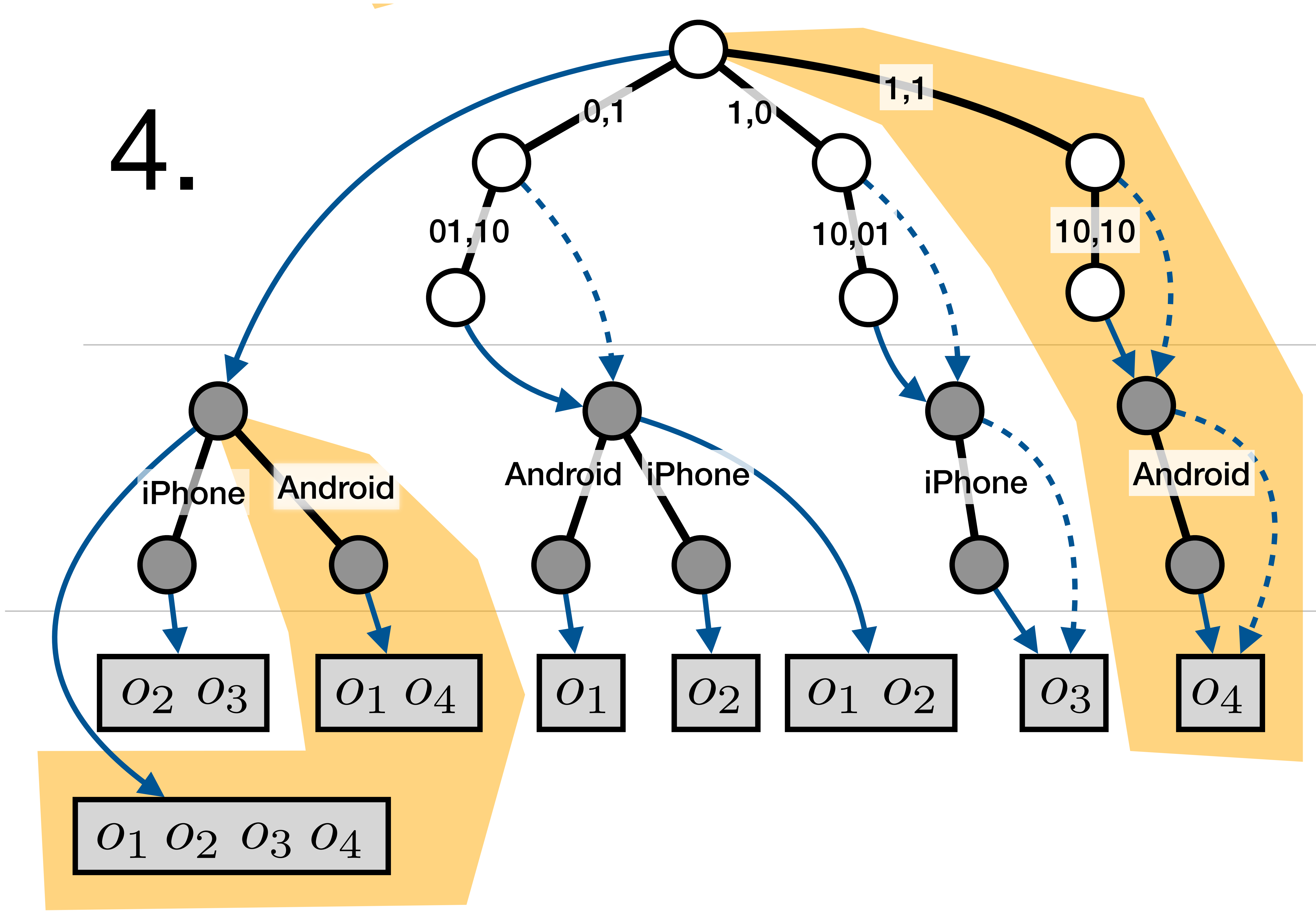
Building a Nanocube: Step 3

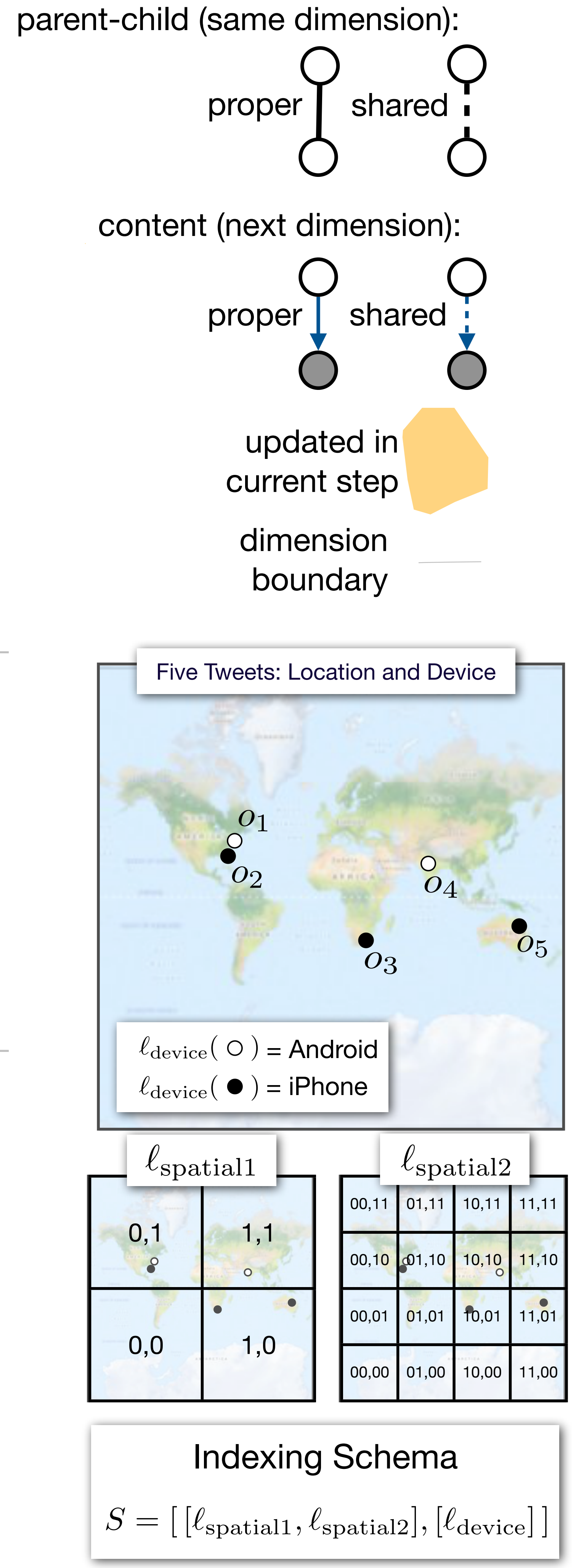




[Lins et. al, 2013]

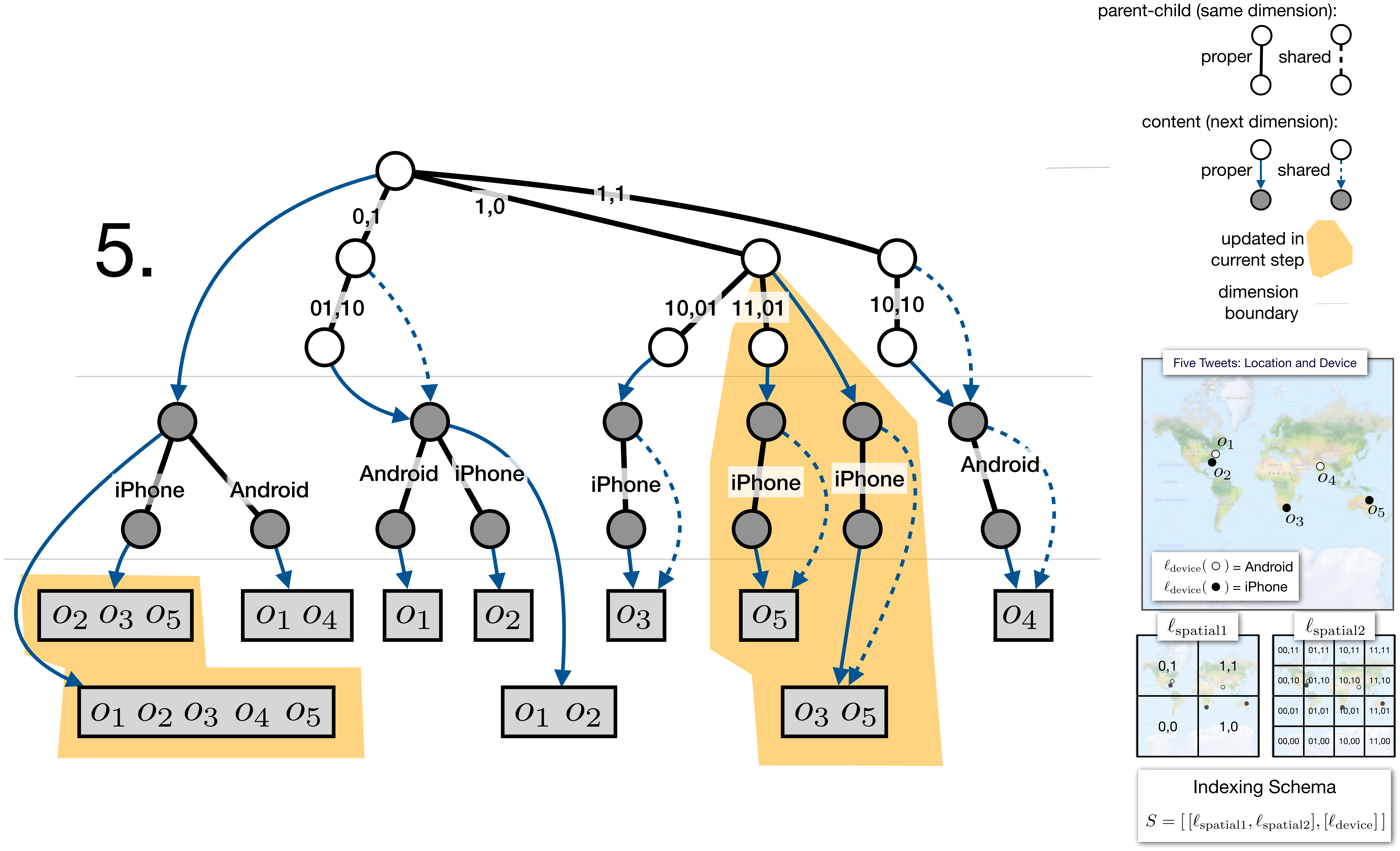
Building a Nanocube: Step 4





[Lins et. al, 2013]

Building a Nanocube: Step 5

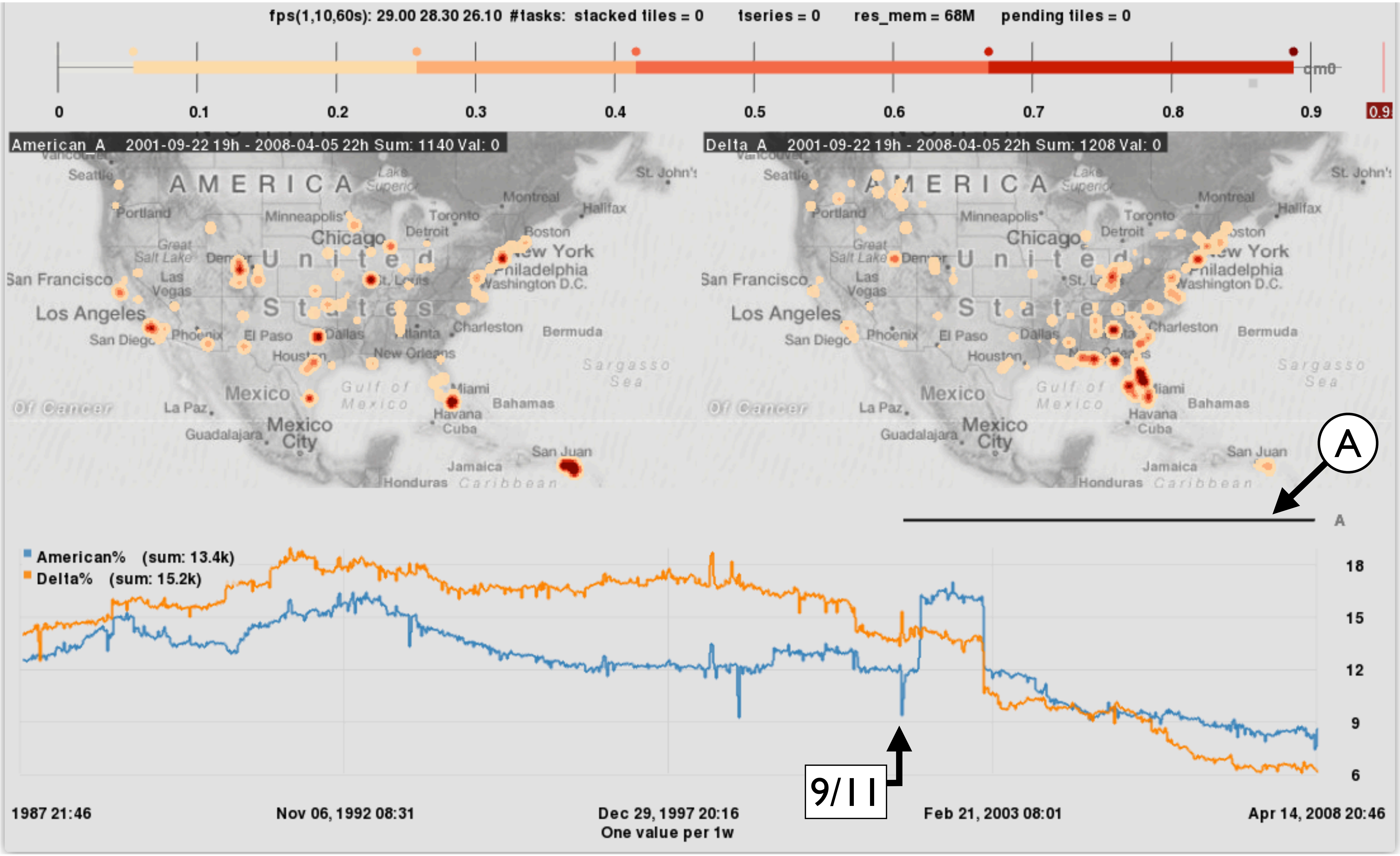


[Lins et. al, 2013]

Nanocubes Discussion

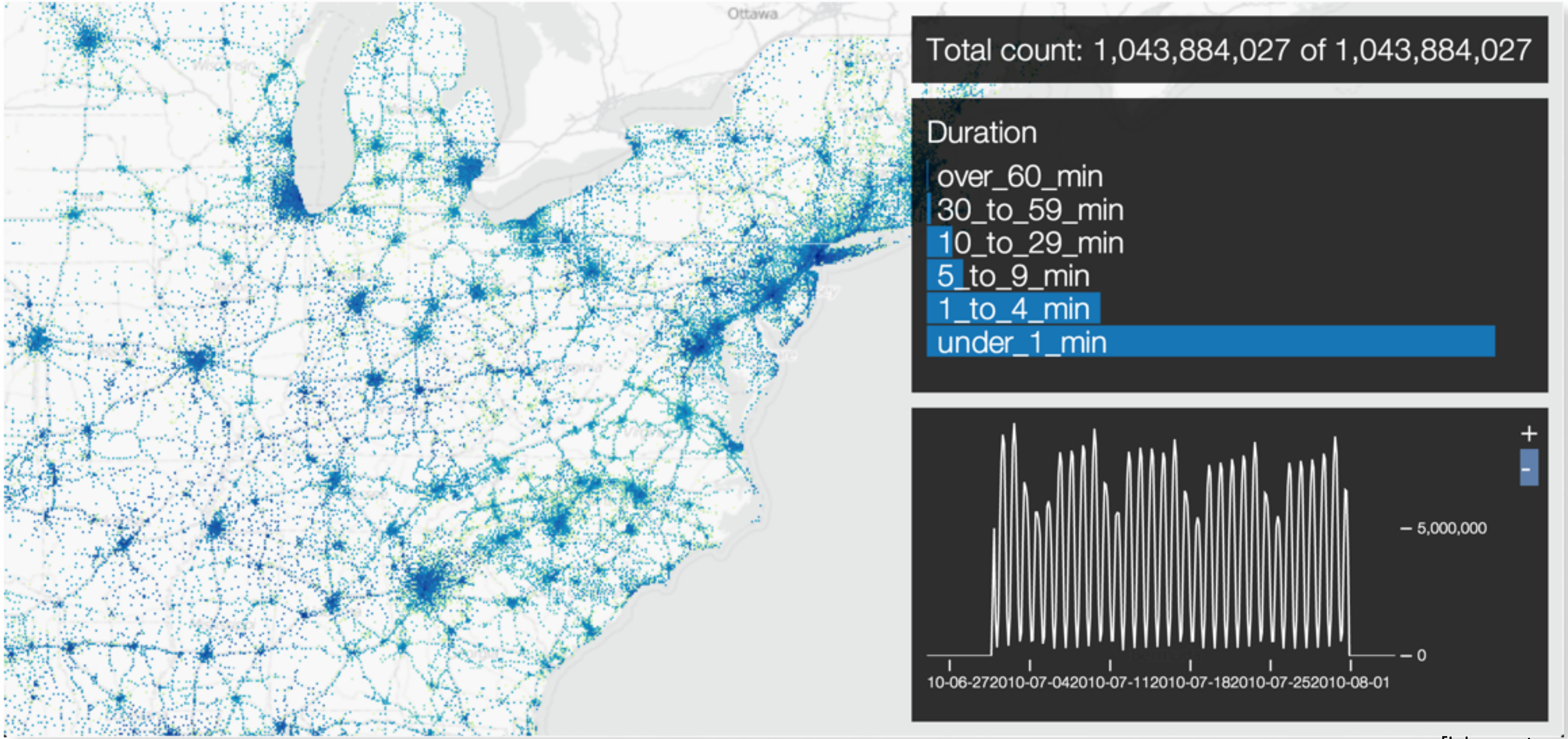
- Save space by organizing the data in a manner that takes advantage of data sparseness
- Limited to one spatial dimension, one temporal dimension
- Precompute **once**, then exploration has **low latency**

Example: American vs. Delta



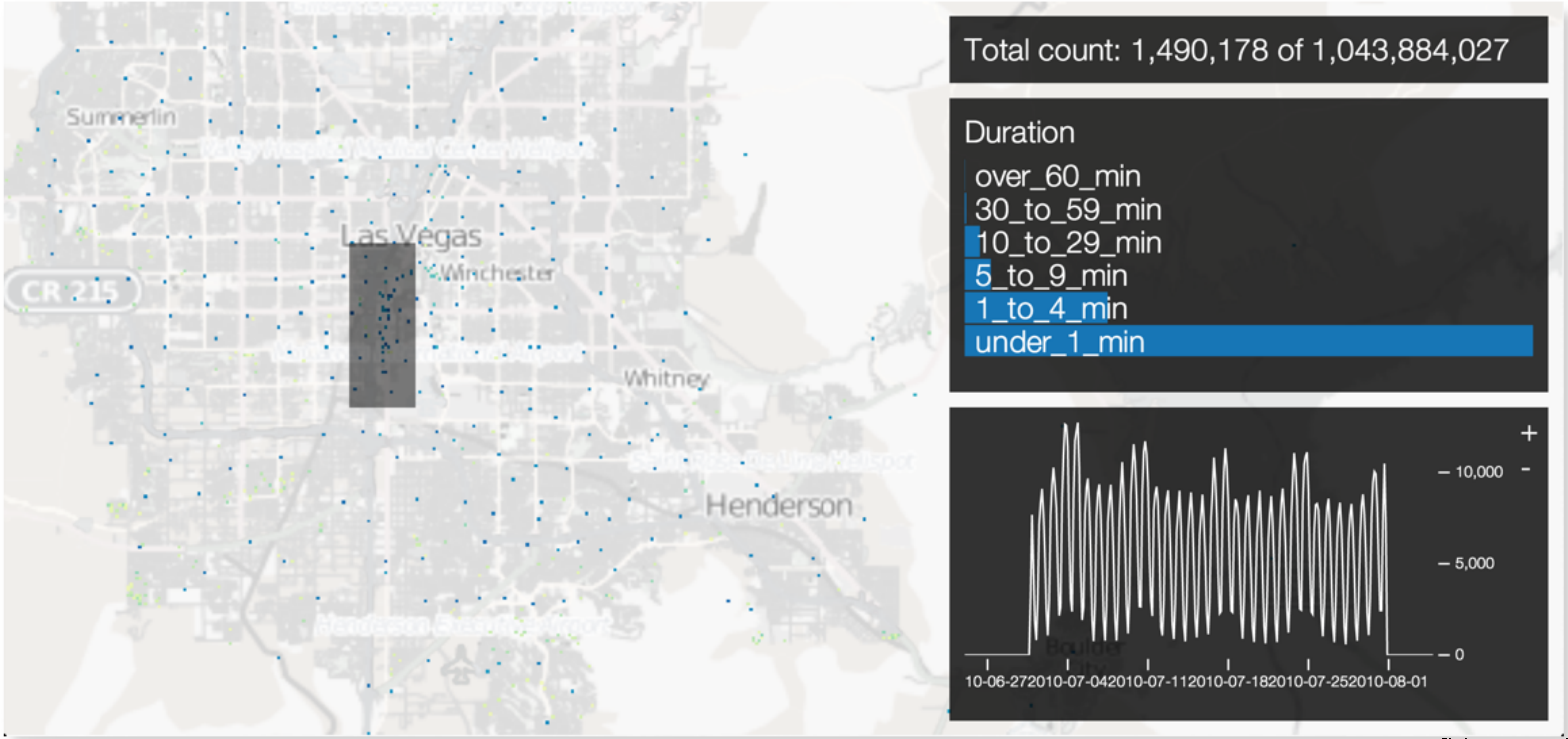
[Lins et. al, 2013]

Example: Cell Data Records



[Lins et. al, 2013]

Example: Cell Data Records



[Lins et. al, 2013]

TopKube: A Rank-Aware Data Cube for Real-Time Exploration of Spatiotemporal Data

F. Miranda, L. Lins, J. T. Klosowski, and C. T. Silva

TopKube: What about Top-k and Rankings?

- Aggregates are interesting
- Also, often interested in **top-k** answers given particular criteria
- …or **rankings**
- Search over time and space but find specific examples
- TopKube is a rank-aware data structure that computes top-k queries with low latency so interactive exploration is possible

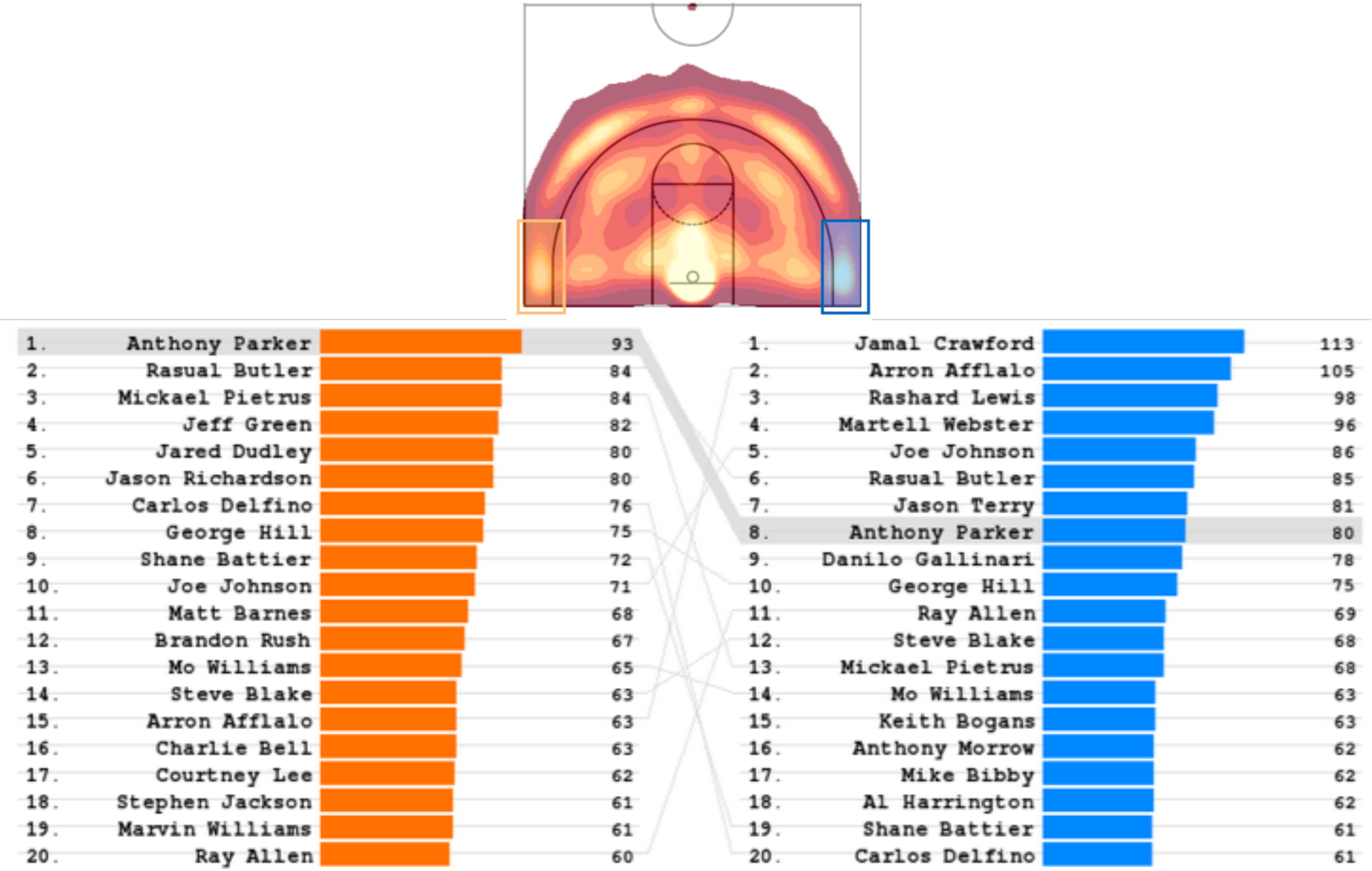
Example: Basketball

- Shots by time, number of points scored, and location on the court

team	player	time	pts	x	y
CLE	L. James	5	0	13	28
BOS	R. Rondo	5	2	38	26
CLE	L. James	7	3	42	35

- Query: Ranked list of the 50 players who took the most shots
  - `SELECT player,count(*) AS shots FROM table GROUP BY player ORDER BY shots DESC LIMIT 50`
- Query: Rank the top 50 players by points made:
  - `SELECT player,sum(pts) AS points FROM table GROUP BY player ORDER BY points DESC LIMIT 50`

Ranking by Shot Location



[F. Miranda et al., 2017]

TopKube vs. Nanocubes

- Product bin: the combination of selections from dimensions
- Nanocubes maps each product bin ((01,10), iPhone) to a **time series**

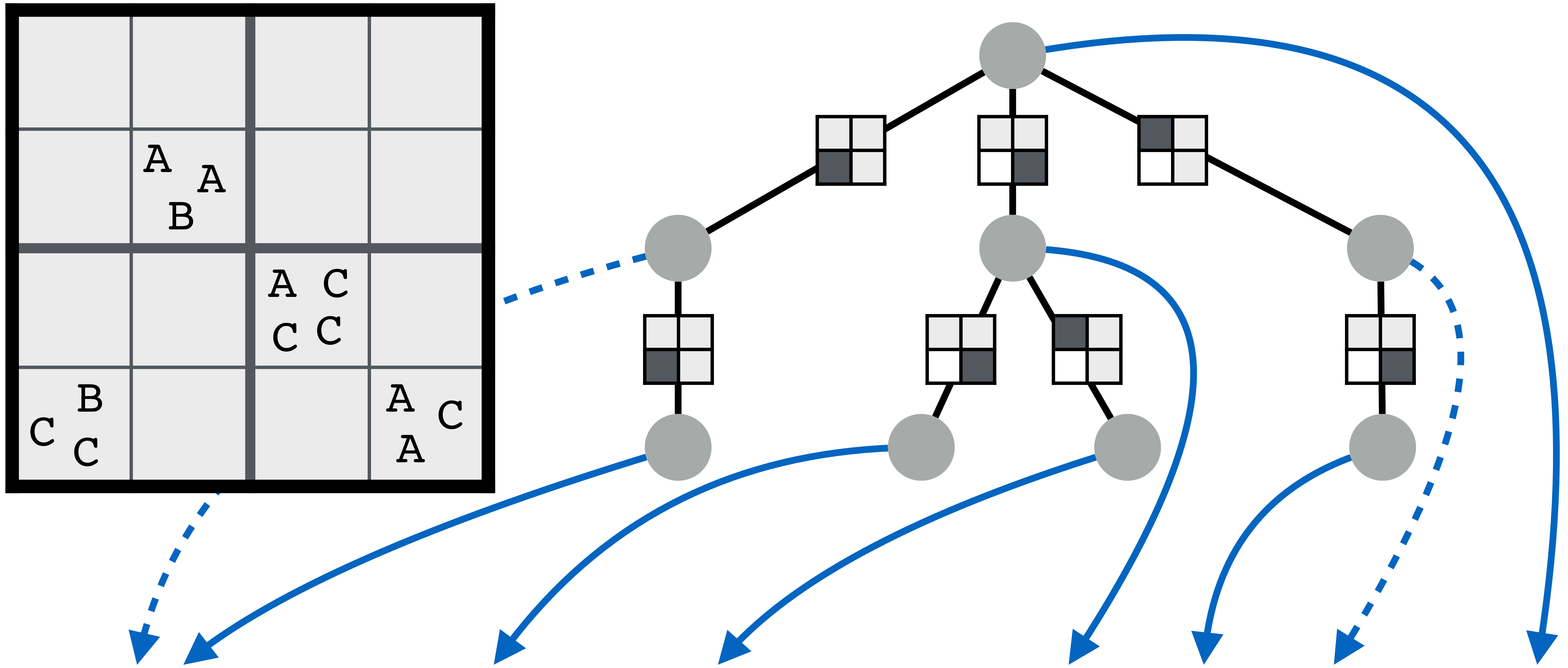
$$\beta \mapsto ((t_1, v_1), (t_2, v_1 + v_2), \ldots, (t_m, v_1 + \ldots + v_m))$$

- TopKube maps each product bin to **rank-aware multi-set**

$$\beta \mapsto \left\{ \mathrm{lst} = ((q_1, v_1, \sigma_1), \ldots, (q_j, v_j, \sigma_j)), \mathrm{sum} = \sum_{i=1}^{j} v_i \right\}$$

- q_i is the ith smallest key that appears in product bin
- v_i is the value of the measure for key q_i in the product bin
- σ_i is the index of the key with its largest value

Example: One Spatial Dim. and A,B,C events



q	v	σ
B	1	2
C	2	1
sum 3		

q	v	σ
A	2	1
C	1	2
sum 3		

q	v	σ
A	1	2
C	3	1
sum 4		

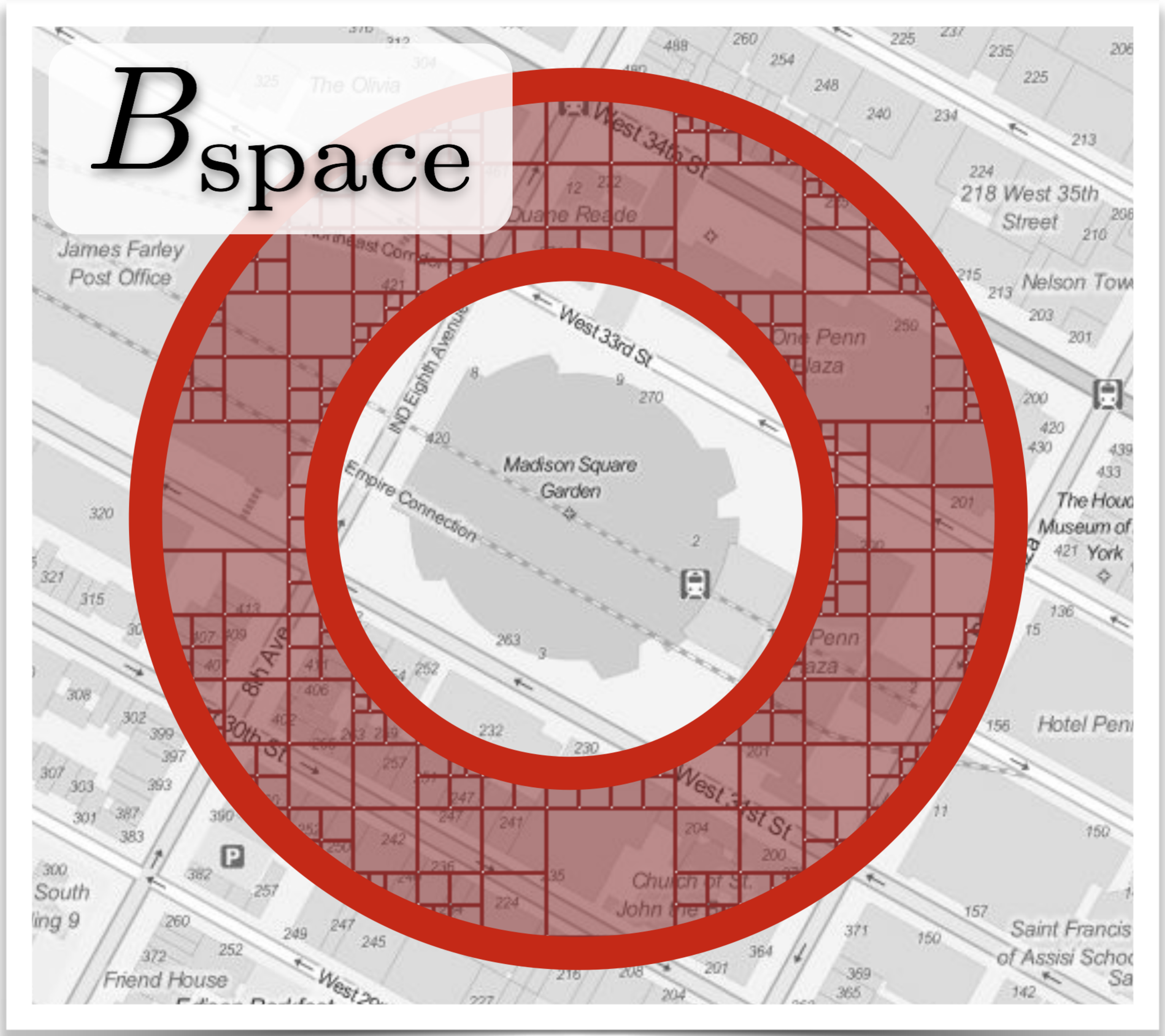
q	v	σ
A	3	2
C	4	1
sum 7		

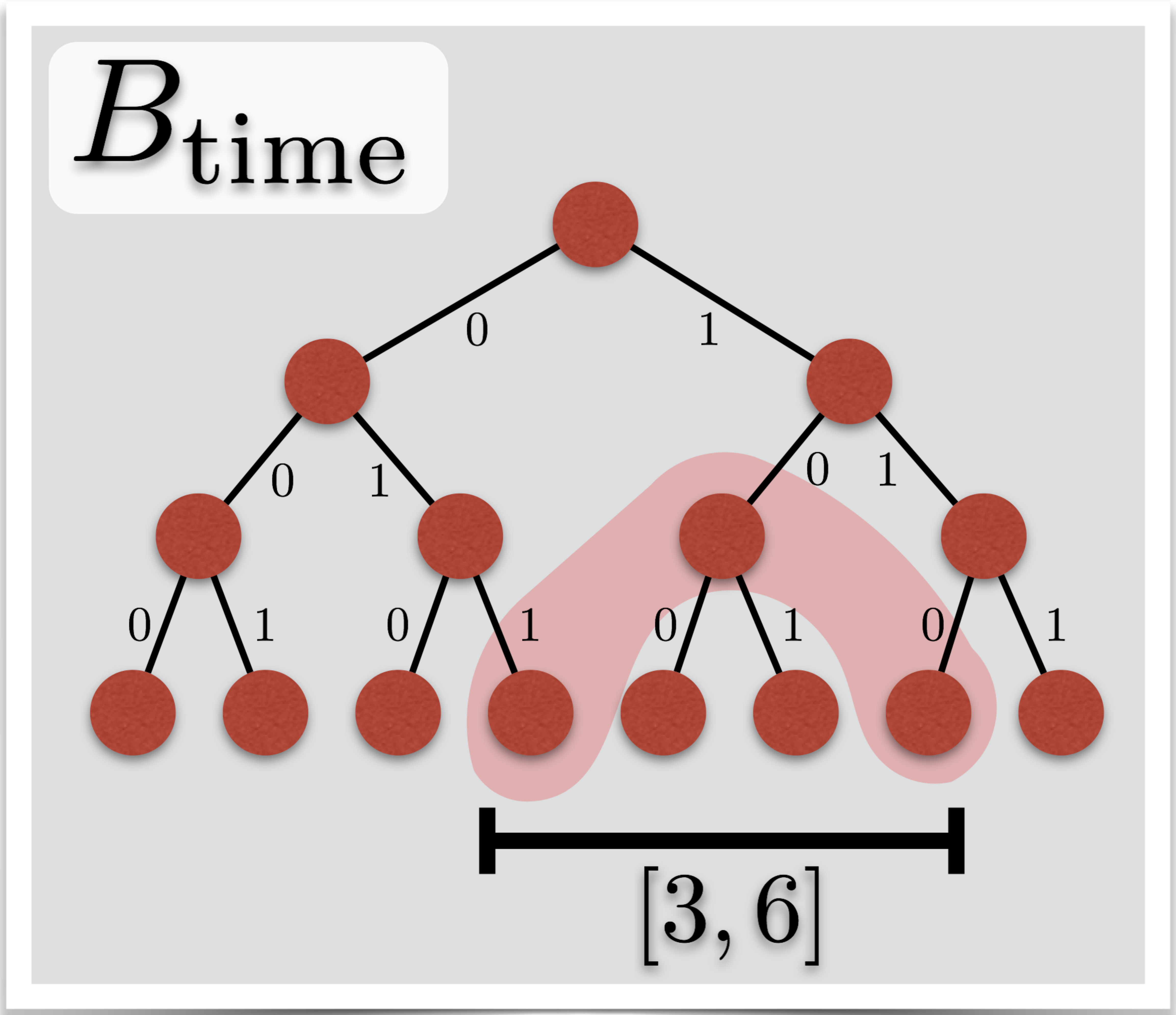
q	v	σ
A	2	1
B	1	2
sum 3		

q	v	σ
A	5	3
B	2	1
C	6	2
sum 13		

[F. Miranda et al., 2017]

Problem: Lots of Bins!



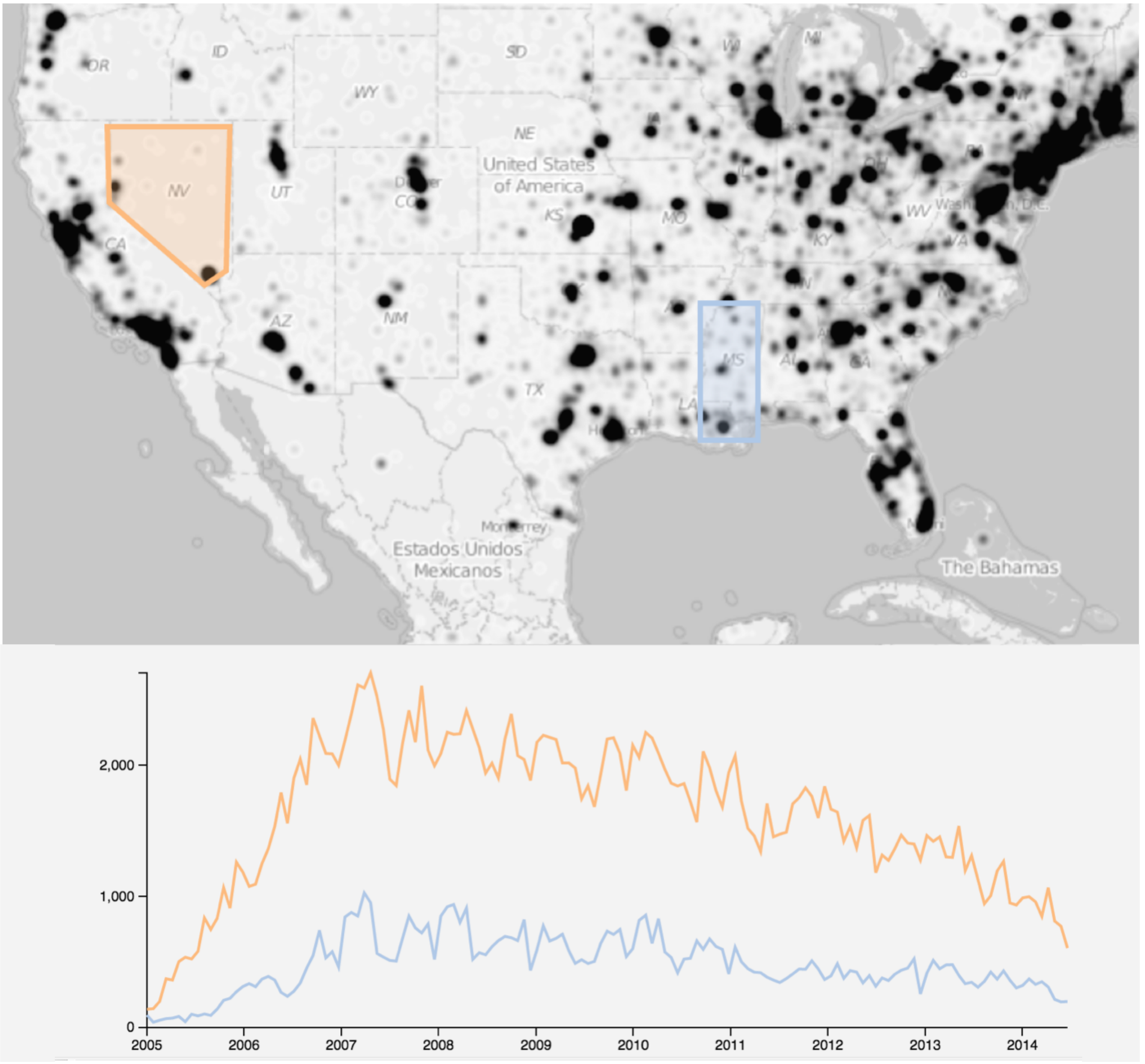


[F. Miranda et al., 2017]

Three Algorithms to Merge Bins

- Threshold: don't do a full scan, use extra information about ranking
- Sweep: Use a priority queue where the product bin with the current smallest key is on the top
- Hybrid:
  - Threshold has best theoretical guarantee but some sparse cases can be faster
  - Use Sweep on small input lists, Threshold on denser problem

Top-edited Wikipages in Nevada and Mississippi

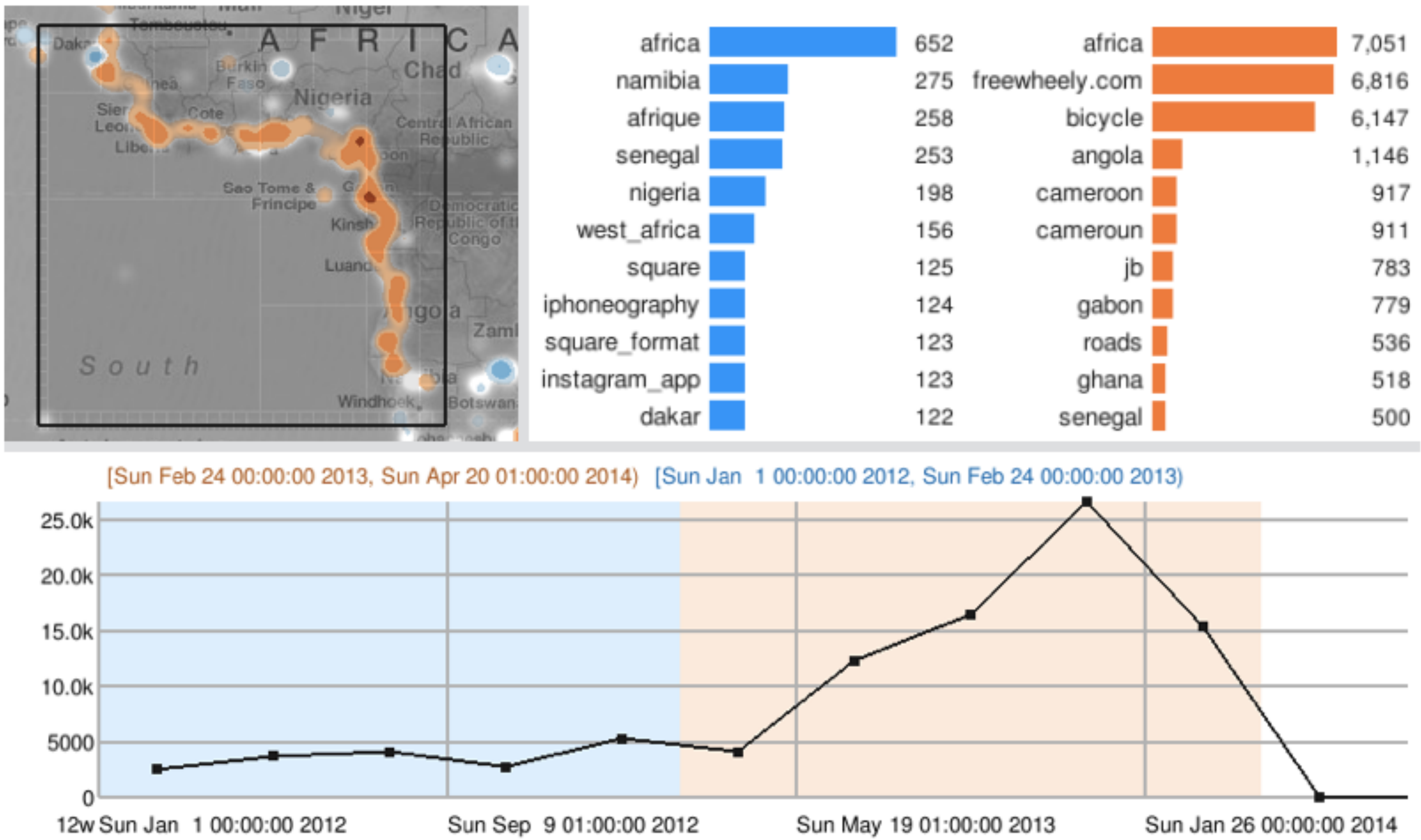


Baton_Rouge,_Louisiana	323
University_of_Mississippi...	230
Mississippi	216
Jackson,_Mississippi	208
Louisiana_State_University...	189
Mississippi_State_University...	169
WVLA-TV	158
Ole_Miss_Rebels_football...	155
List_of_Star_Wars_books...	131
Louisiana	122
New_Orleans_Saints	107

Reno,_Nevada	303
Early_Christianity	284
Comparison_of_the_AK-47_and_M1...	273
Las_Vegas_Academy	225
Timeline_of_Christianity...	216
Las_Vegas	204
Council_of_Jerusalem	192
Paul_the_Apostle	190
University_of_Nevada,_Las_Vega...	189
Nevada	188
Antinomianism	188

[F. Miranda et al., 2017]

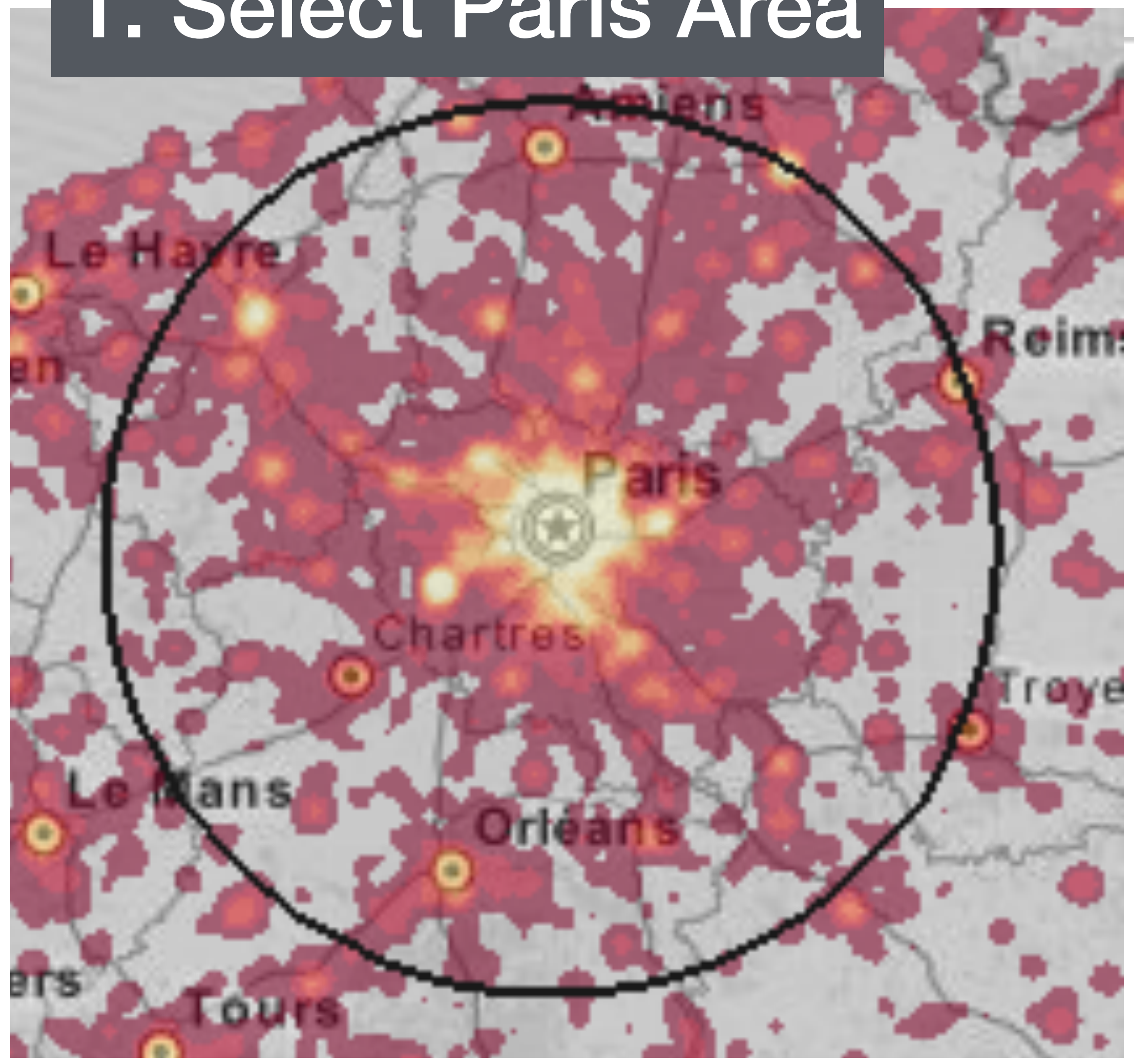
Geolocated Flickr tags in Africa



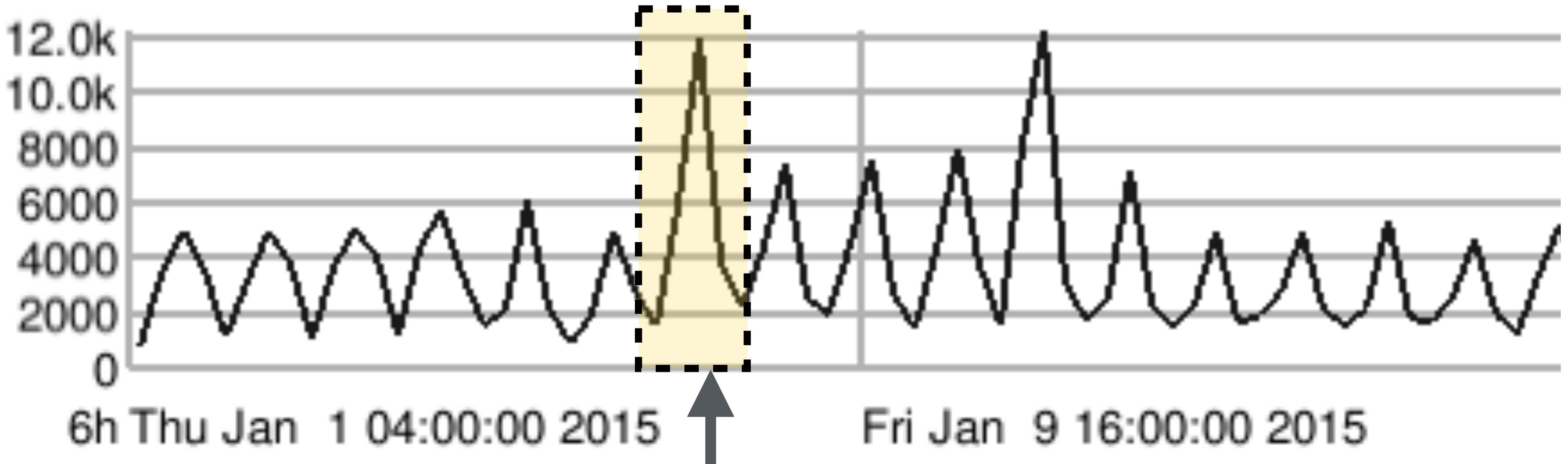
[F. Miranda et al., 2017]

Top Hashtags in Paris related to Charlie Hebdo

1. Select Paris Area

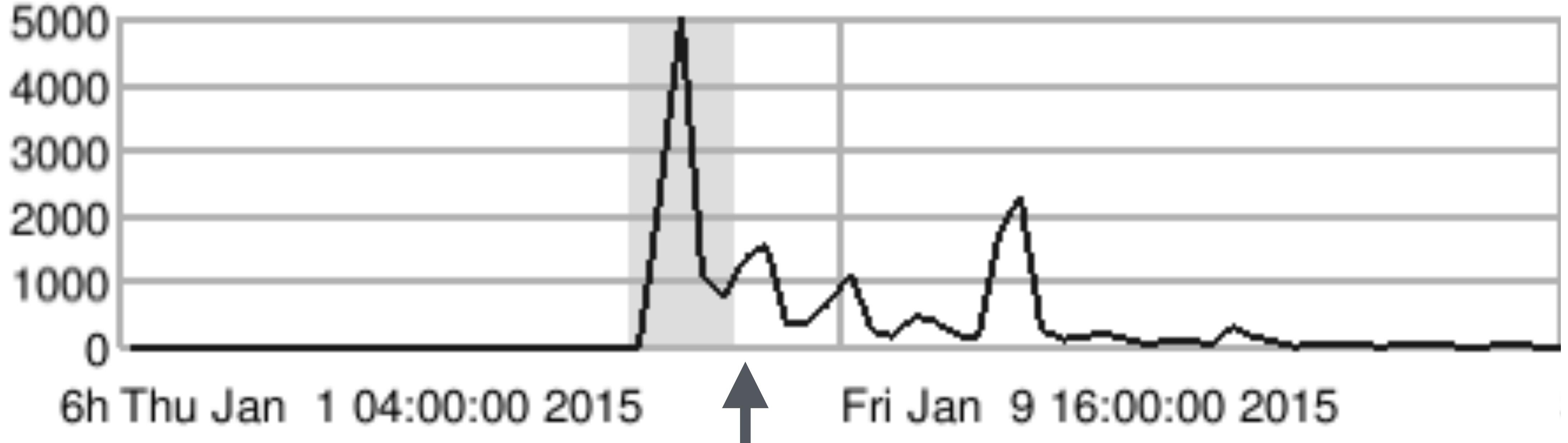


2. Observe Uncommon Spike on Wed. Jan 7, 2015



3. Select this Spike and Observe Top-10 Hashtags

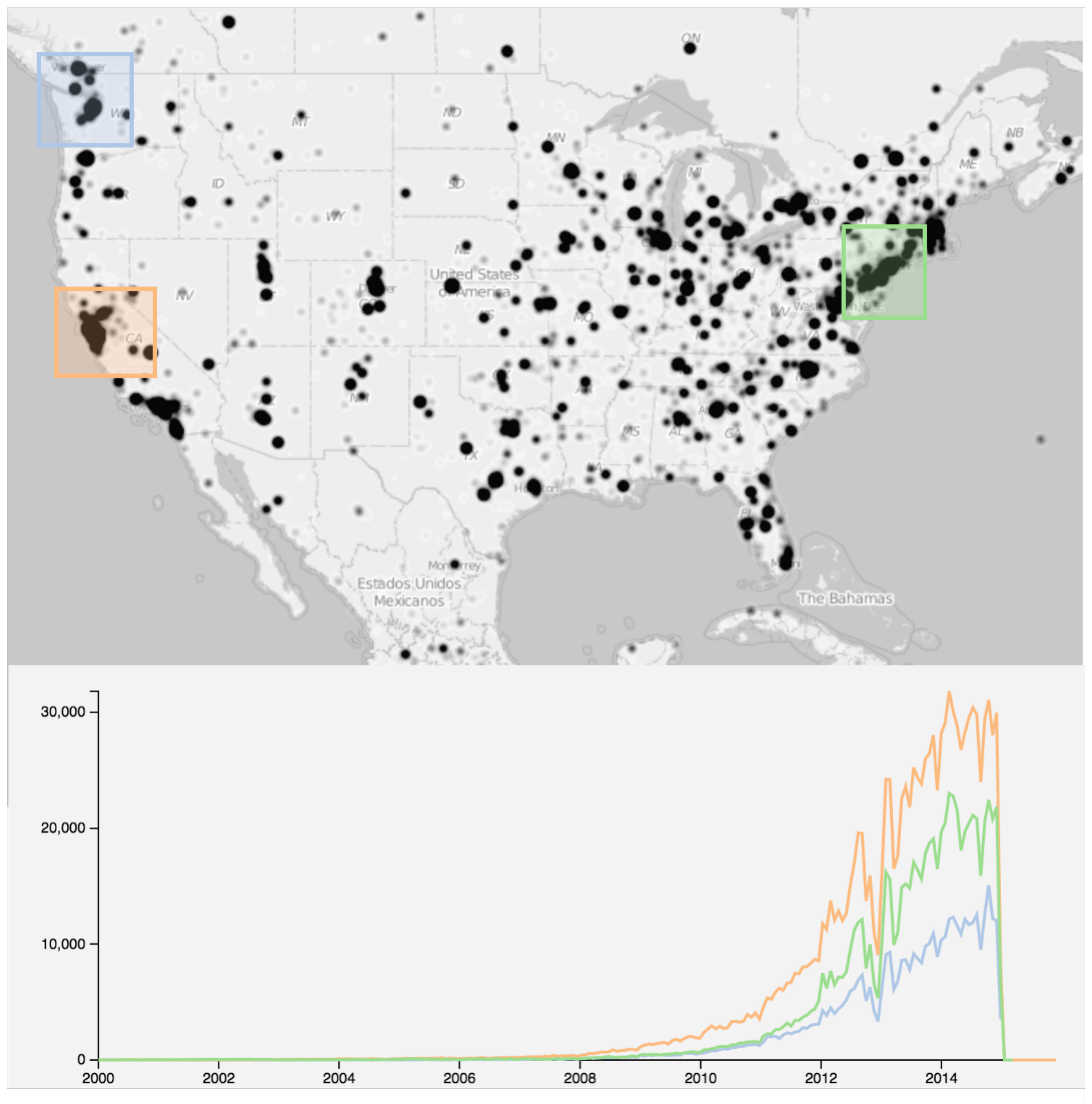
1.	#jesuischarlie	4,456
2.	#charliehebdo	4,190
3.	#lrt	1,146
4.	#paris	607
5.	#gagnetaplace	447
6.	#charliehebdo	418
7.	#off	402
8.	#lt	335
9.	#noussommescharlie	197
10.	#rip	187



4. Select Charlie Hebdo's Top Hashtags and Observe its Temporal Volume Pattern

[F. Miranda et al., 2017]

GitHub Top commits near urban centers



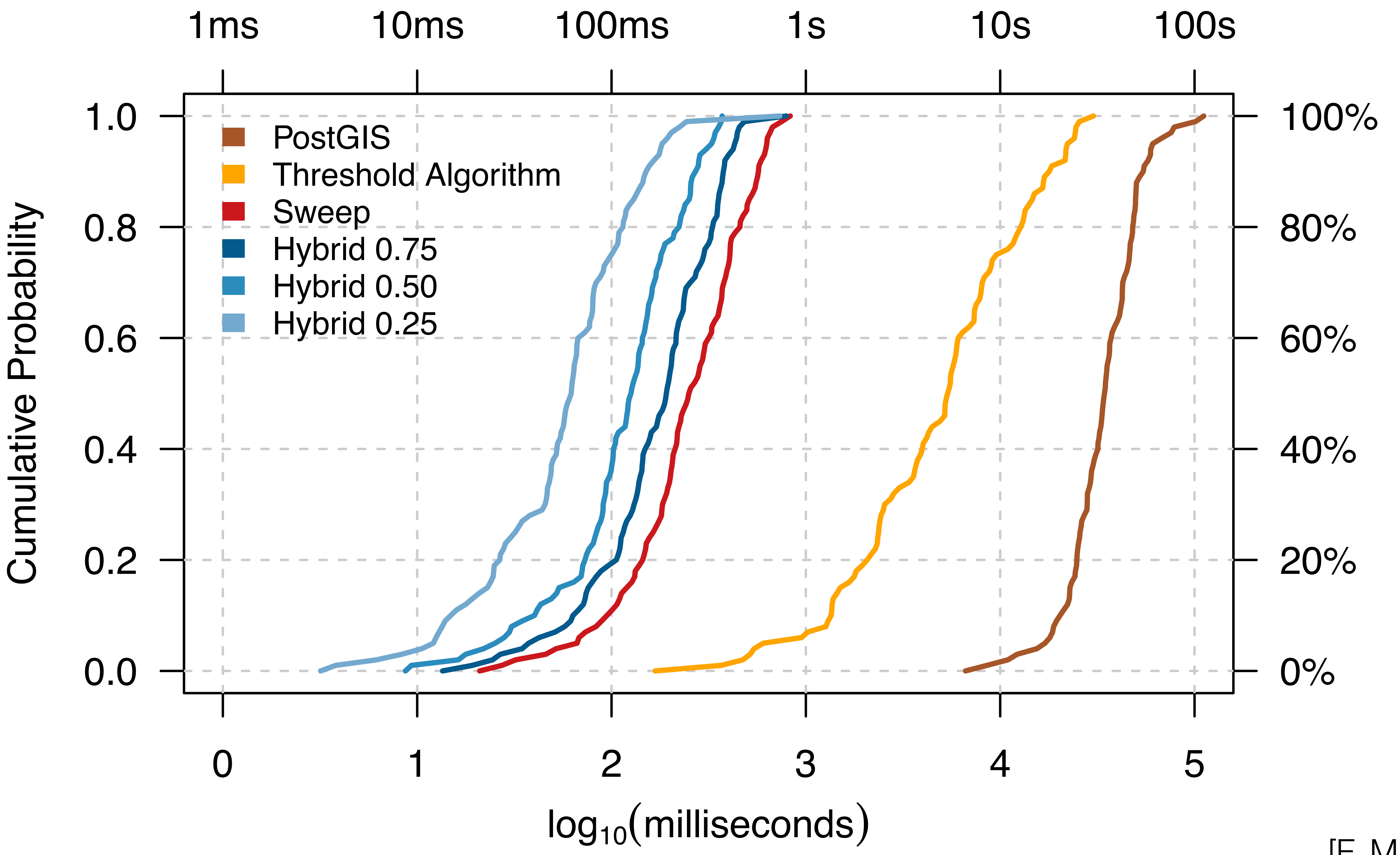
dotfiles	4102
homebrew	1780
chef	1297
Synapse-Repo...	1155
mapnik	1105
NzbDrone	1057
rails	961
zamboni	843
SynapseWebCl...	793
rubygems	790
mozilla-cent...	758
beets	738
leiningen	728
guzzle	717
groovy-eclip...	707

dotfiles	10022
rust	2341
llvm-project...	2321
llvm-project...	2306
titanium_mob...	2266
git	2031
node	1837
gaia	1830
wireshark-ht...	1745
spark	1741
phabricator	1720
mozilla-cent...	1709
zamboni	1631
gecko-dev	1531
docs	1522

dotfiles	8908
postgresql	2122
mongo	2107
pubsub	1761
docs	1243
neon-rtk-gps...	1222
pandas	1175
homebrew	1165
julia	1102
sample_app	973
astrometry.n...	948
.emacs.d	947
VTK	921
website	858
Cinder	749

[F. Miranda et al., 2017]

Evaluation

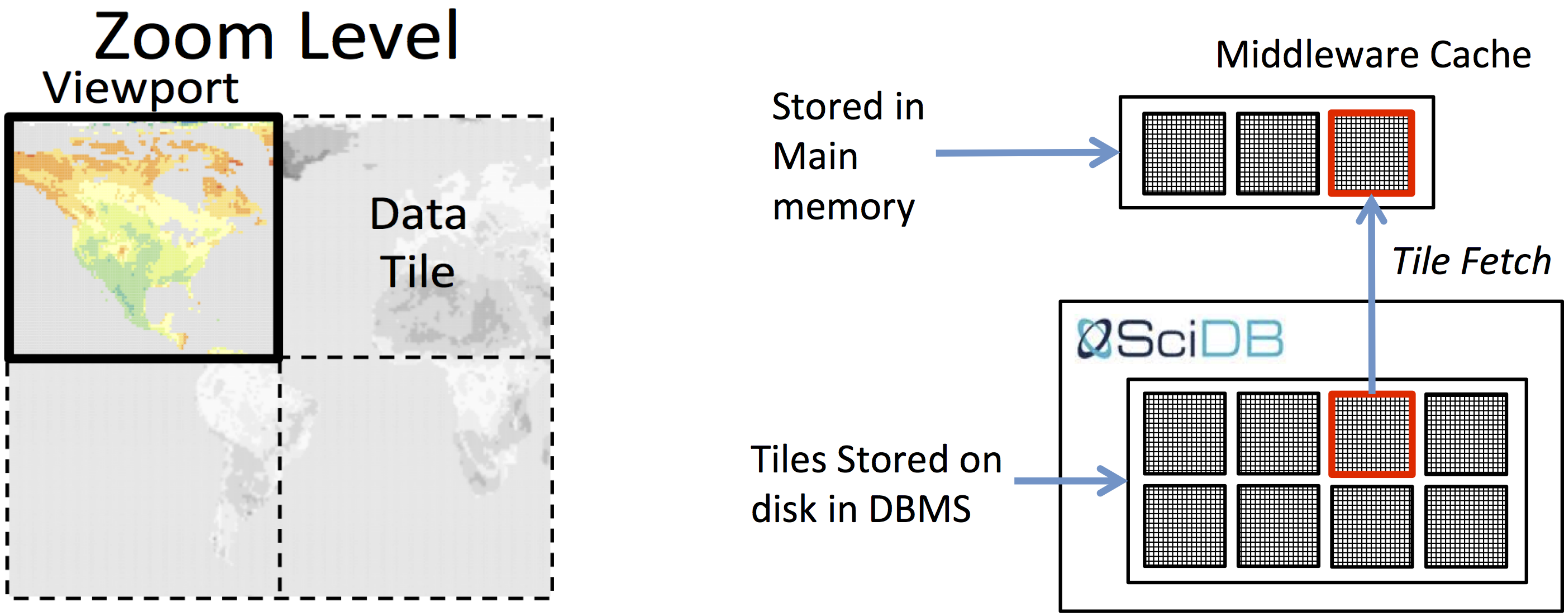


[F. Miranda et al., 2017]

ForeCache

L. Battle

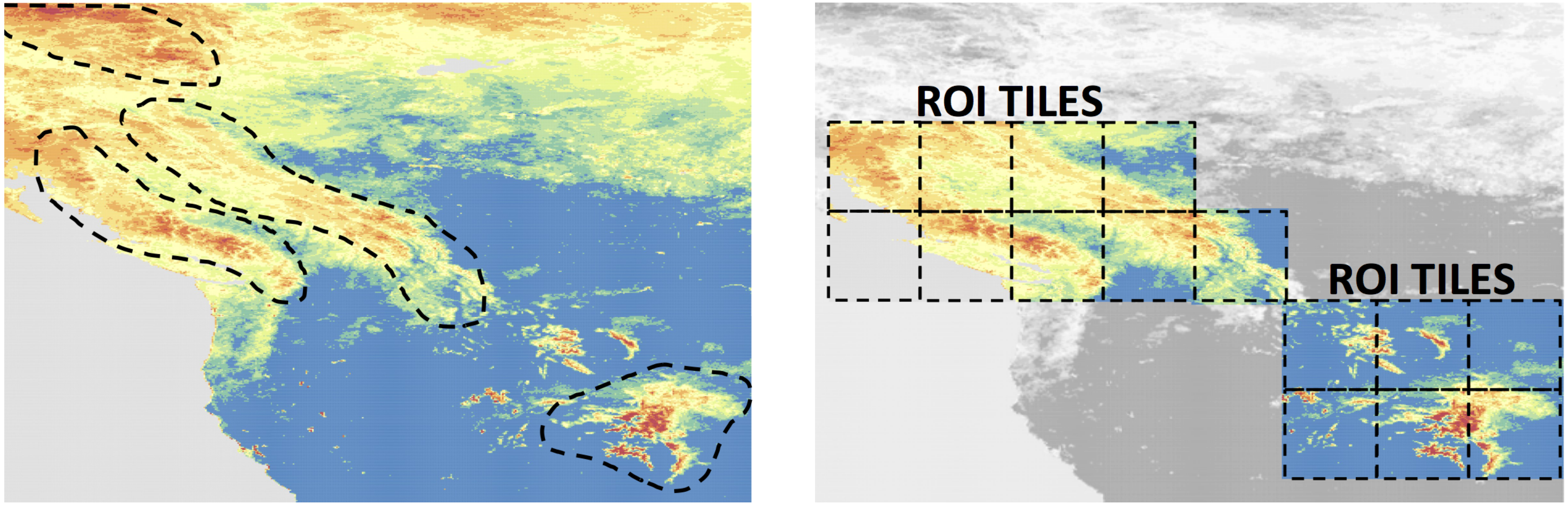
Tile Storage in ForeCache



[Battle et al., 2016]

ForeCache

- Predict which tiles a user will need next and prefetch those
  - Use common patterns (zoom, pan)
  - Use regions of interest (ROIs)



[Battle et al., 2016]