

# Lessons Learned from OpenTSDB

Or why OpenTSDB is the way it is  
and how it changed iteratively to  
correct some of the mistakes made

# Key concepts

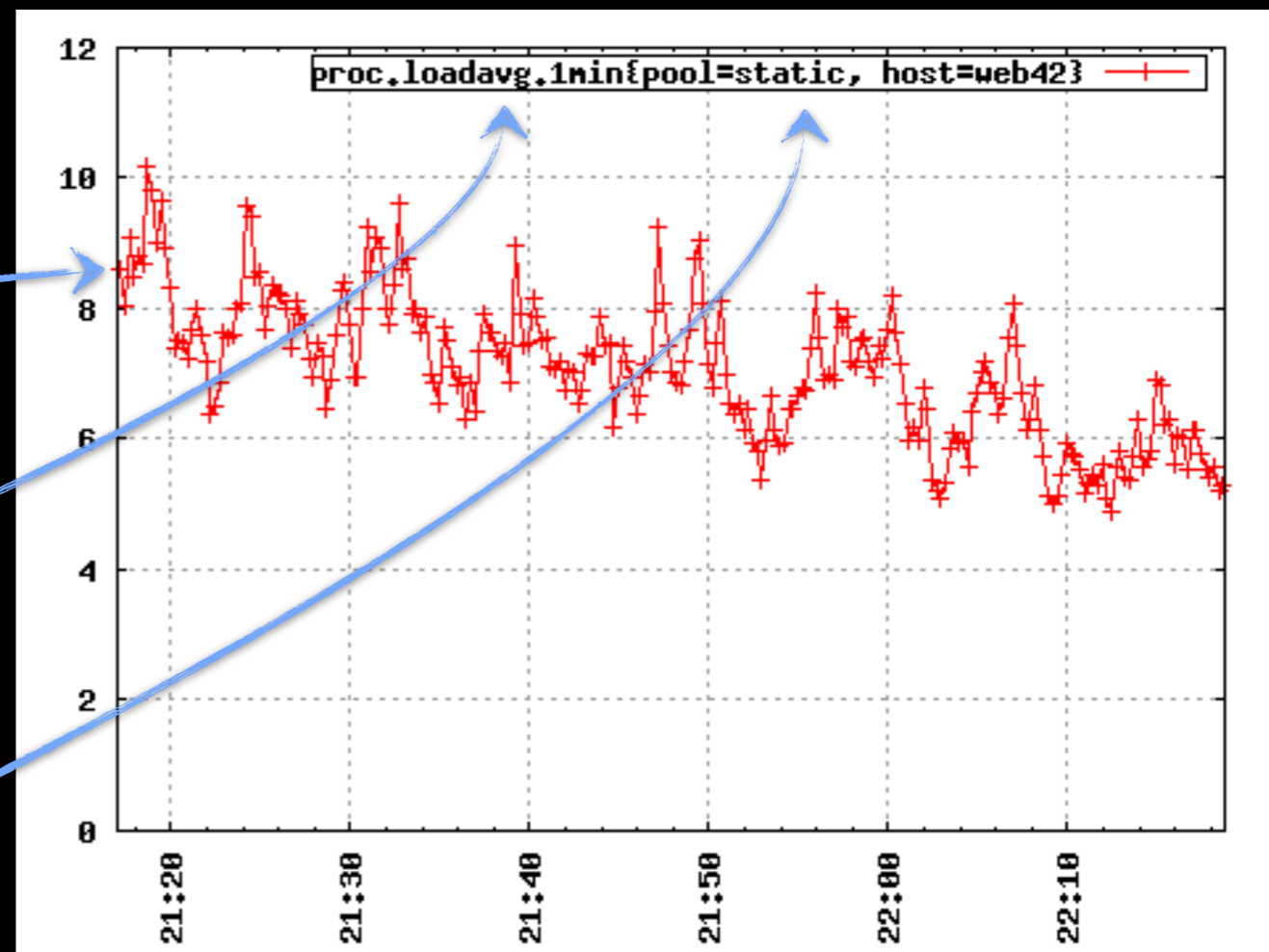
- Data Points  
(time, value)

- Metrics  
`proc.loadavg.1m`

- Tags  
`host=web42 pool=static`

- Metric + Tags = Time Series

- Order of magnitude:  $> 10^6$  time series,  $> 10^{12}$  data points



```
put proc.loadavg.1m 1234567890 0.42 host=web42 pool=static
```

# OpenTSDB @ StumbleUpon

- Main production monitoring system for ~2 years
- Storing hundreds of billions of data points
- Adding over 1 billion data points per day
- 13000 data points/s → 130 QPS on HBase
- If you had a 5 node cluster, this load would hardly make it sweat



# Do's

- **Wider** rows to seek faster  
before: ~4KB/row, after: ~20KB
- Make writes **idempotent** and **independent**  
before: start rows at arbitrary points in time  
after: align rows on 10m (then 1h) boundaries
- Store **more data per KeyValue**  
Remember you pay for the key along *each* value  
in a row, so large keys are *really* expensive

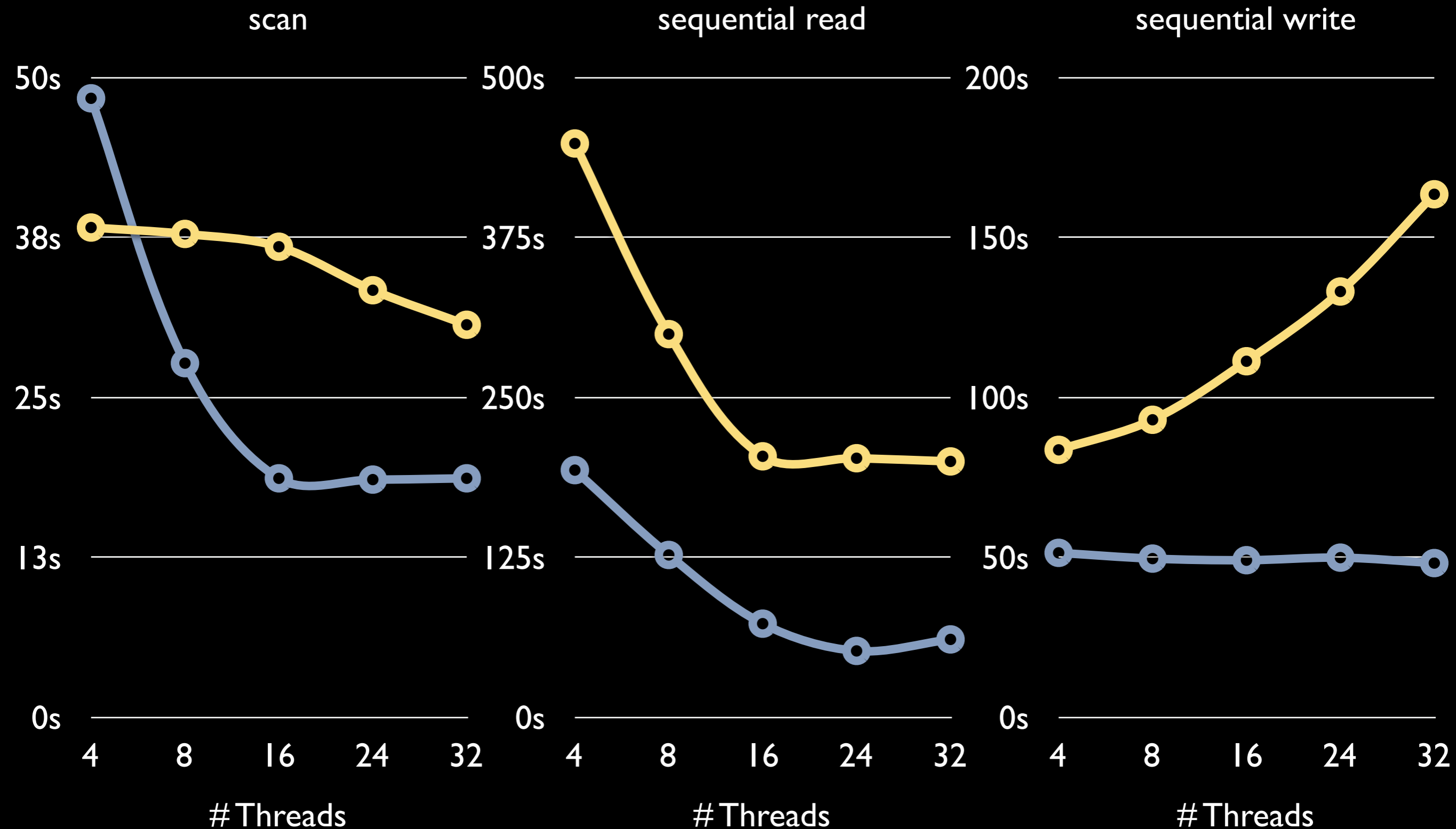
# Don'ts

- Use `HTable` / `HTablePool` in app servers  
`asynchbase` + `Netty` or `Finagle` = performance++
- Put variable-length fields in composite keys  
They're hard to scan
- Exceed a few hundred regions per `RegionServer`  
“Oversharding” introduces overhead and makes recovering from failures more expensive

See detailed benchmark at [goo.gl/8at5V](http://goo.gl/8at5V)

# Use asyncbase

HTable asyncbase



# How OpenTSDB came to be the way it is

## Questions:

- How to store time series data efficiently in HBase?
- How to enable concurrent writes without synchronization between the writers?
- How to save space/memory when storing hundreds of billions of data items in HBase?

# Time Series Data in HBase

Take 1

Key	Column	
1234567890	1	
1234567892	2	
1234567894	3	

Annotations:

- don't care (points to the empty right column)
- values (points to the values in the right column)
- timestamps (points to the keys in the left column)

Simplest design: only 1 time series, 1 row with a single KeyValue per data point.

Supports time-range scans.



# Time Series Data in HBase

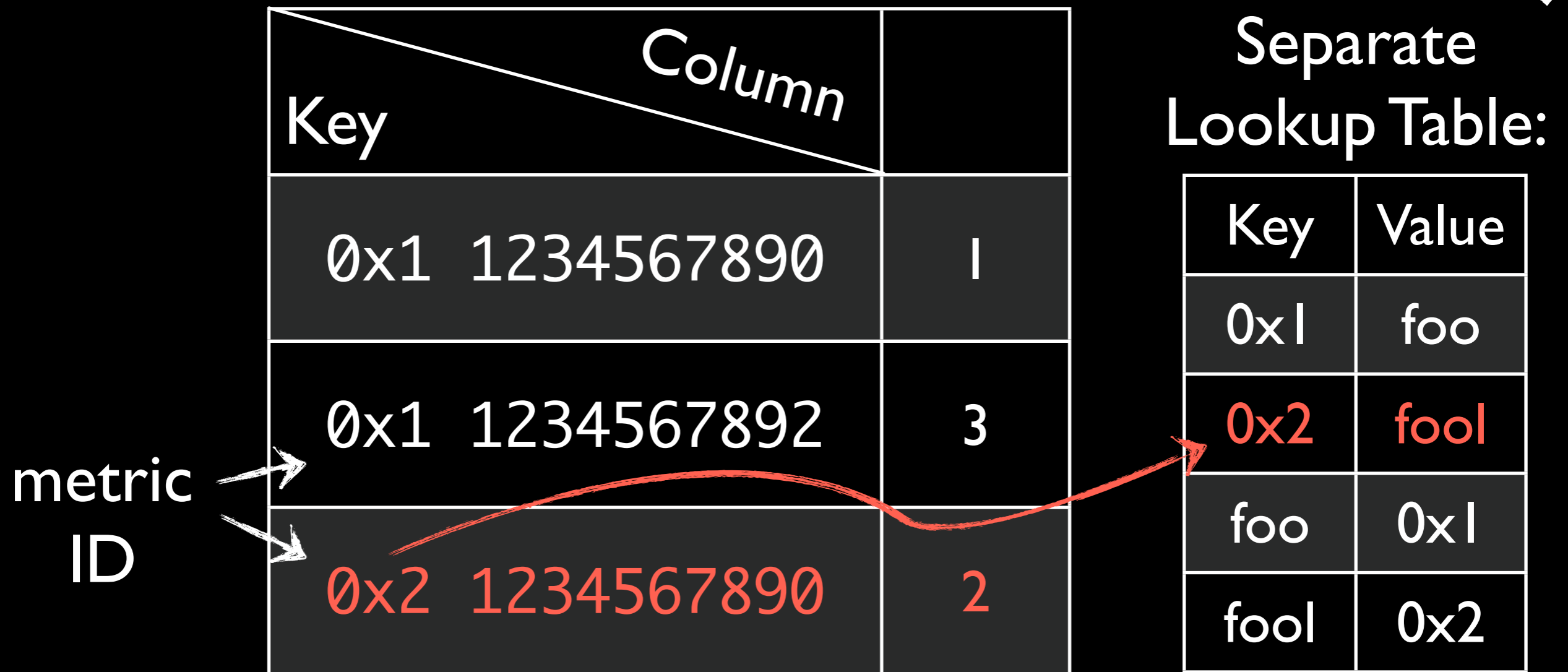
Take 2

	Key	Column
	foo 1234567890	1
	foo 1234567892	3
metric name	foo1 1234567890	2

Metric name first in row key for data locality.  
Problem: can't store the metric as text in row key  
due to space concerns

# Time Series Data in HBase

Take 3

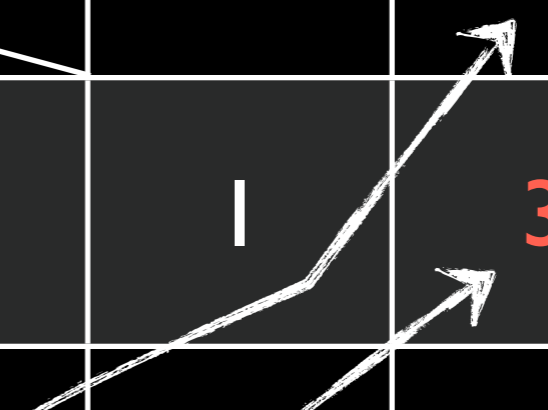


Use a separate table to assign unique IDs to metric names (and tags, not shown here). IDs give us a predictable length and achieve desired data locality.

# Time Series Data in HBase

Take 4

Key \ Column	+0	+2
0x1 1234567890	1	3
0x1 1234567892	3	
0x2 1234567890	2	



Reduce the number of rows by storing multiple consecutive data points in the same row.

Fewer rows = faster to seek to a specific row.

# Time Series Data in HBase

Take 4

Misleading  
table  
representation

Key	Column	+0	+2
0x1 1234567890		1	3
0x1 1234567892		3	
0x2 1234567890		2	

Gotcha #1: wider rows don't save any space\*

Actual table  
stored

Key	Colum	Value
0x1 1234567890	+0	1
0x1 1234567890	+2	3
0x2 1234567890	+0	2

\* Until magic prefix  
compression happens in  
upcoming HBase 0.94

# Time Series Data in HBase

Take 4

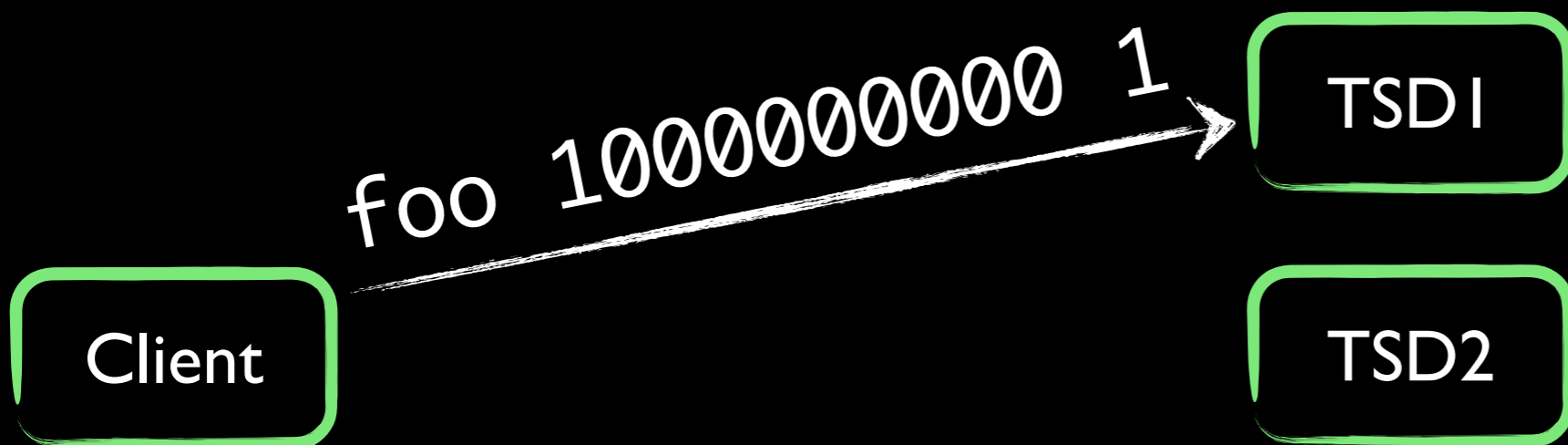
Key \ Column	+0	+2
0x1 1234567890	1	3
0x1 1234567892	3	
0x2 1234567890	2	

Devil is in the details: when to start new rows?  
Naive answer: start on first data point, after some time start a new row.

# Time Series Data in HBase

Take 4

Key	Column	+0	
0x1	10000000000	1	

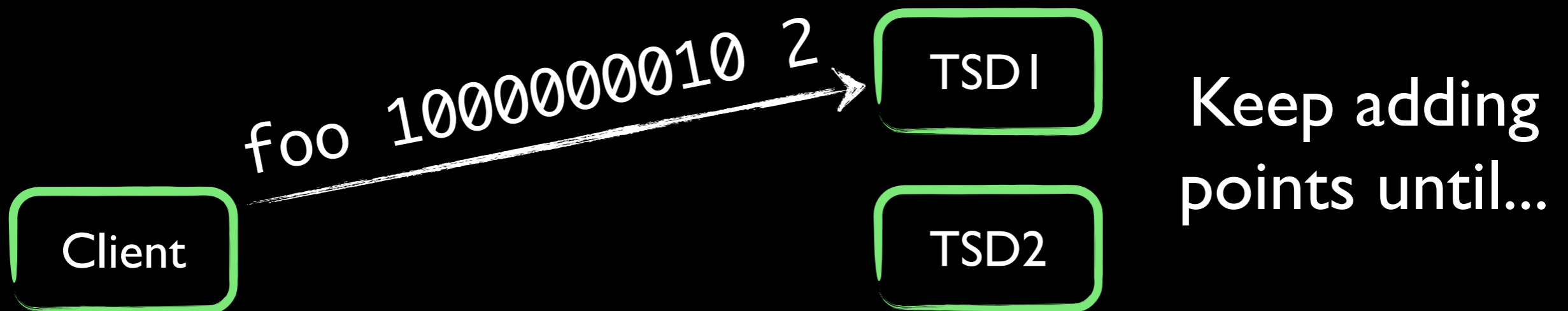


First data point:  
Start a new row

# Time Series Data in HBase

Take 4

Key \ Column	+0	+10	...
0x1 1000000000	1	2	...



# Time Series Data in HBase

Take 4

Key \ Column	+0	+10	...	+599
0x1 1000000000	1	2	...	42





# Time Series Data in HBase

Take 4

Key \ Column	+0	+10	...	+599
0x1 1000000000	1	2	...	42
0x1 1000000600		51		



# Time Series Data in HBase

Take 4

Key \ Column	+0	
0x1 1234567890	1	

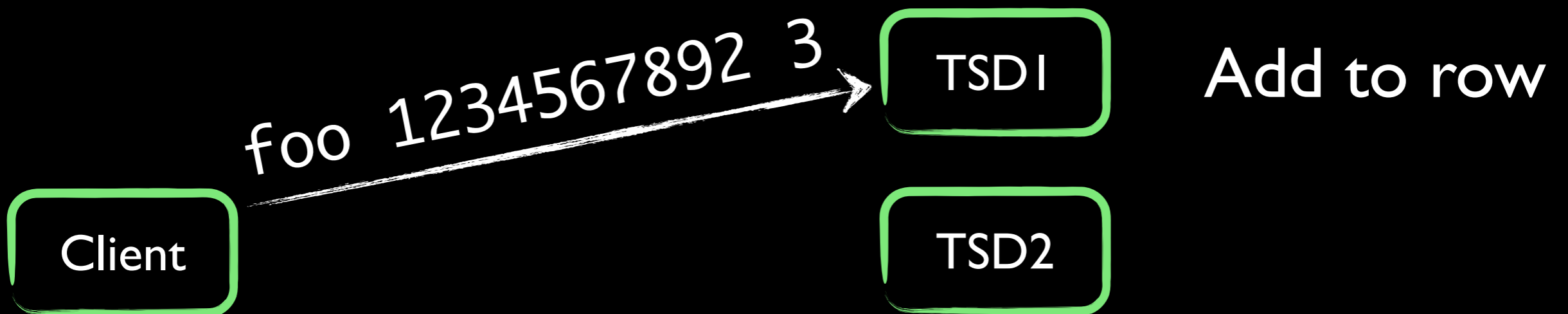
But this scheme fails with multiple TSDs



# Time Series Data in HBase

Take 4

Key \ Column	+0	+2
0x1 1234567890	1	3



# Time Series Data in HBase

Take 4

Key \ Column	+0	+2
0x1 1234567890	1	3
0x1 1234567892	3	

Oops!

Maybe a connection failure occurred, client is retransmitting data to another TSD



# Time Series Data in HBase

Take 5

Base  
timestamp  
always a  
multiple of  
600

Key \ Column	+90	+92
0x1 1234567800	1	3
0x2 1234567800	2	

In order to scale easily and keep TSD stateless,  
make writes independent & idempotent.  
New rule: rows are aligned on 10 min. boundaries

# Time Series Data in HBase

# Time Series Data in HBase

Take 6

Key \ Column	+1890	+1892
0x1 1234566000	1	3
0x2 1234566000	2	

Remember: wider rows don't save any space!

Actual table stored

Key	Column	Value
0x1 1234566000	+1890	1
0x1 1234566000	+1892	3
0x2 1234566000	+1890	2

Key is easily 4x bigger than column + value and repeated

# Time Series Data in HBase

Take 7

Key \ Column	+1890	+1890	+1892	+1892
0x1 1234566000	1	1	3	3
0x2 1234566000	2			

Solution: "compact" columns by concatenation

Key	Column	Value
0x1 1234566000	+1890	1
0x1 1234566000	+1890,+1892	1, 3
0x1 1234566000	+1892	3
0x2 1234566000	+1890	2

Actual table stored

Space savings on disk and in memory are huge: data is 4x-8x smaller!



Fork me on GitHub

# Questions ?

## opentsdb.net

### Summary

- Use asynchbase
- Wider table > Taller table
- Make writes idempotent
- Compact your data
- Use Netty or Finagle
- Short family names
- Make writes independent
- Have predictable key sizes



StumbleUpon



*Think this is cool?  
We're hiring*

Benoît "tsuna" Sigoure  
[tsuna@stumbleupon.com](mailto:tsuna@stumbleupon.com)