This slide intentionally left blank.

Karmasphere

Time Series or Causal Analysis Without Limits!

Shevek
shevek@karmasphere.com

Karmasphere

Or:
How to Break the Stock Market

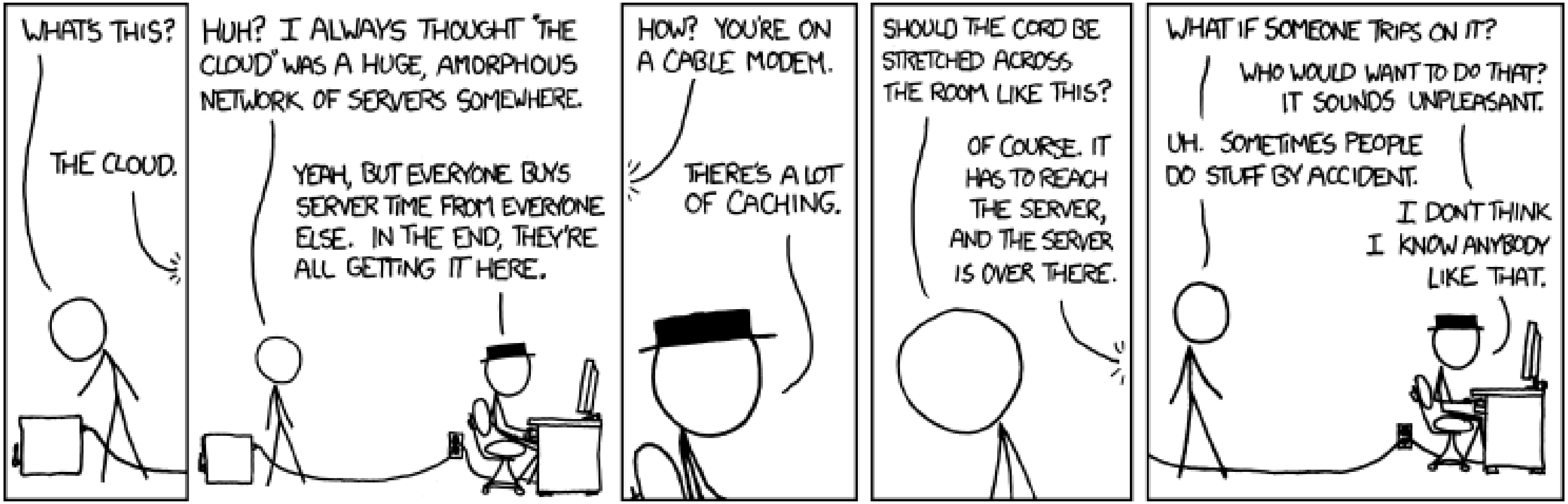
It wasn't me
nobody@citimorgansachs.com

Introduction

- My background.
  - Compilers and languages
  - Algorithmic design.
  - First principles.
- Nobody understands a Brit.
  - I swear at a tremendous speed.
  - Slow me down.

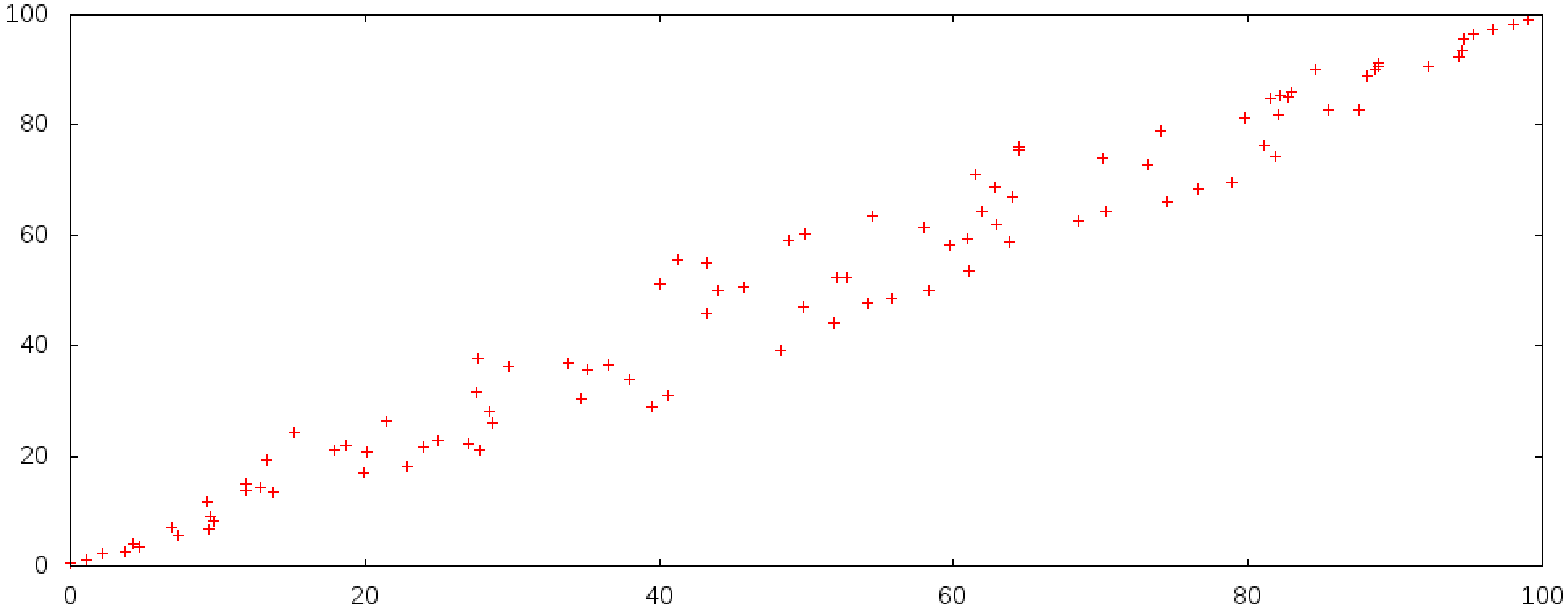
This Talk

- Ask questions, shout, throw things.
  - Don't take life too seriously!
- The objective is to enable you.
  - Not to show off what I did.



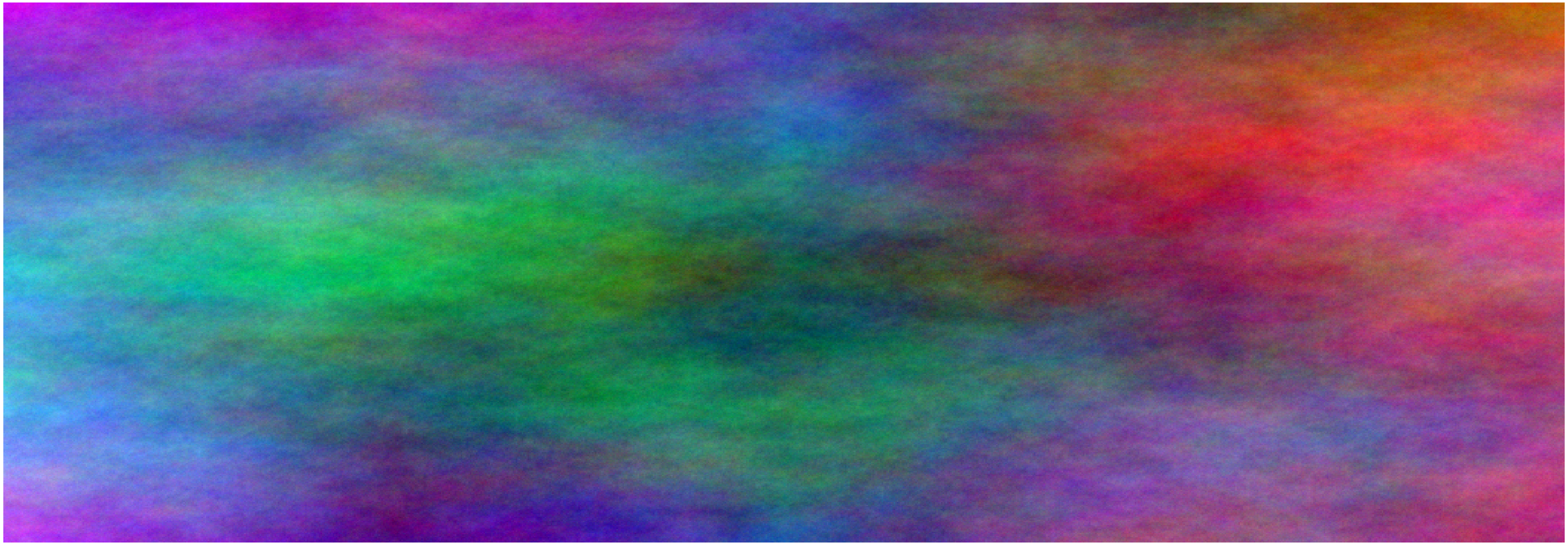
Searching for Similarities

- Correlation.
  - Allows us to predict the properties of a new discrete datum.



Searching for Patterns

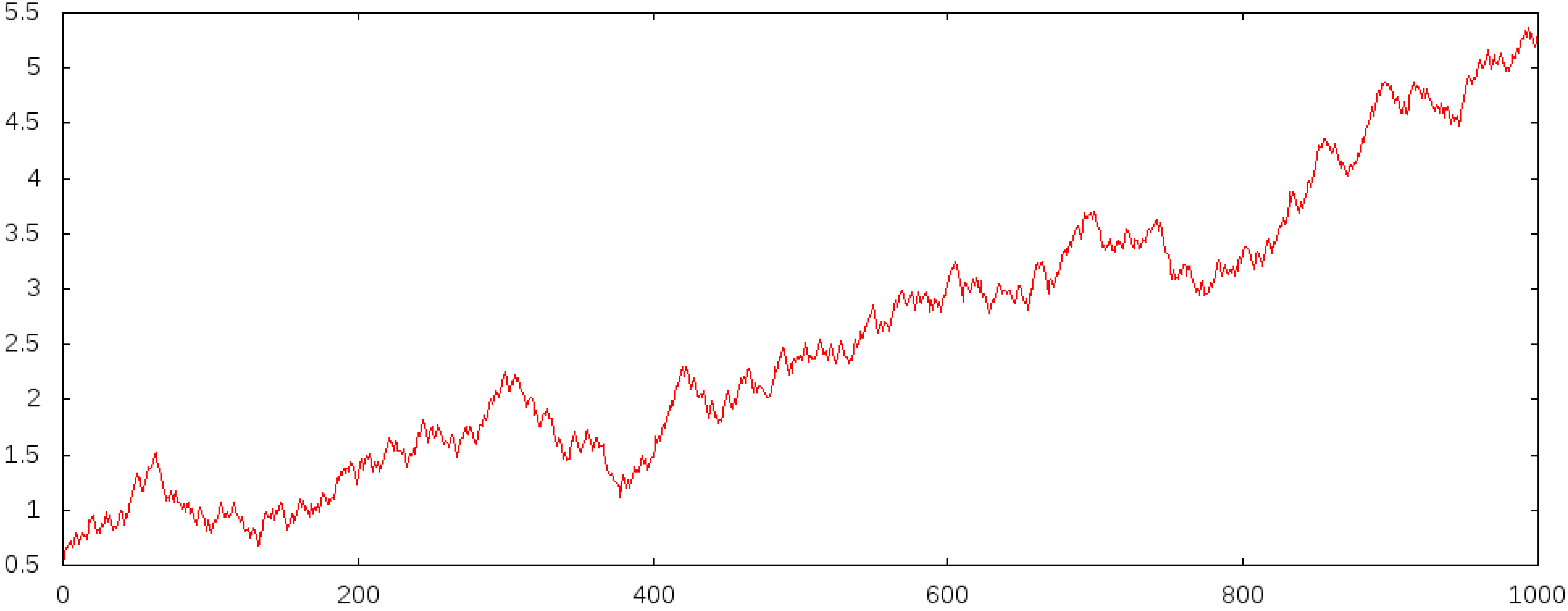
- Blueish.
- Kind of reddish at the top right.
- And there's a greenish area in the middle.



OK, let's not kill ourselves here.

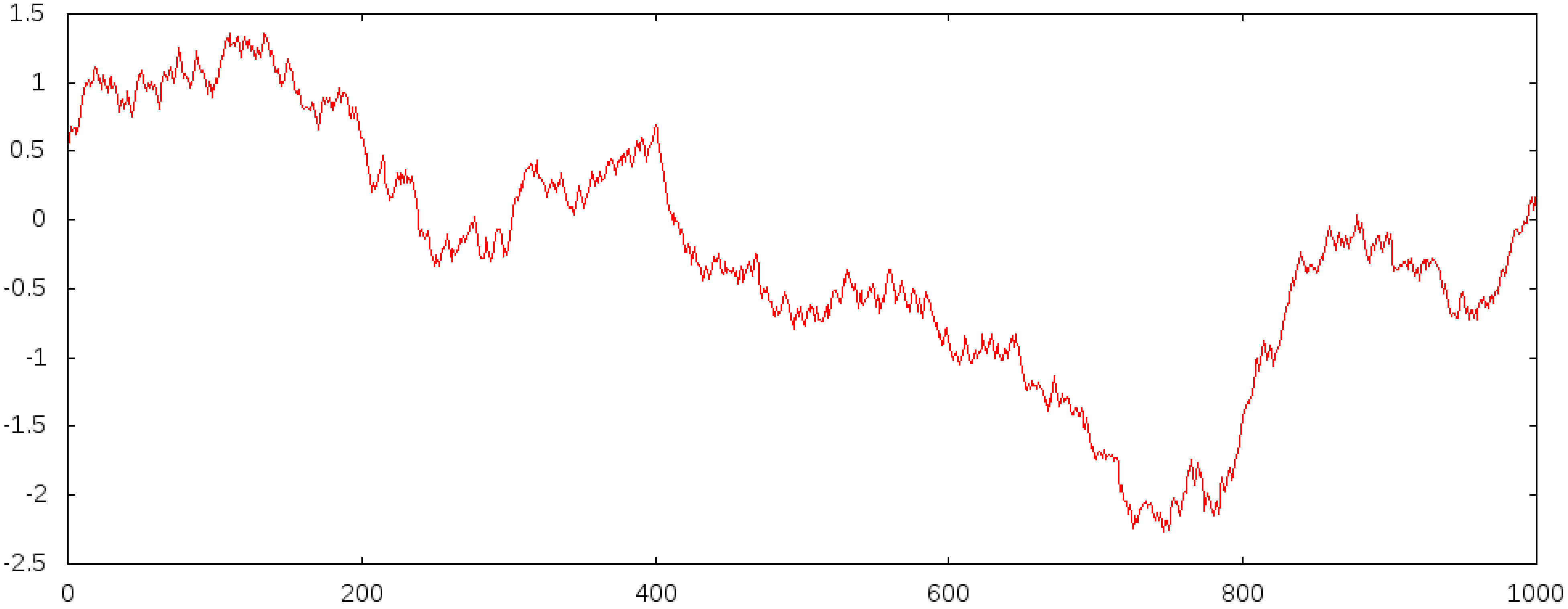
Searching for Trends

- Trends and functions of a time-base.
  - What happened next?
  - Bankers might recognise this pattern, from 1997.



And Correctly Identifying Them

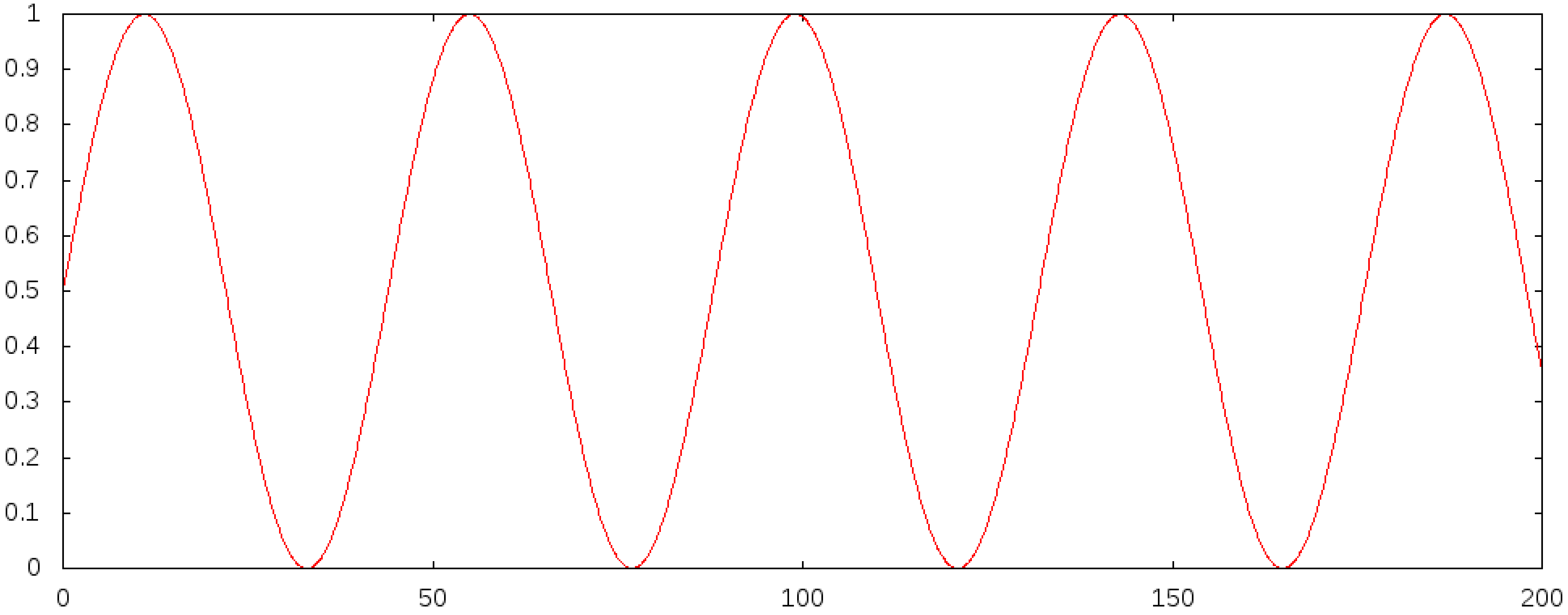
- This happened next.
  - Another pattern from our banks, circa 2002.



Trends suck. What else can we look for?

Predicting the Future

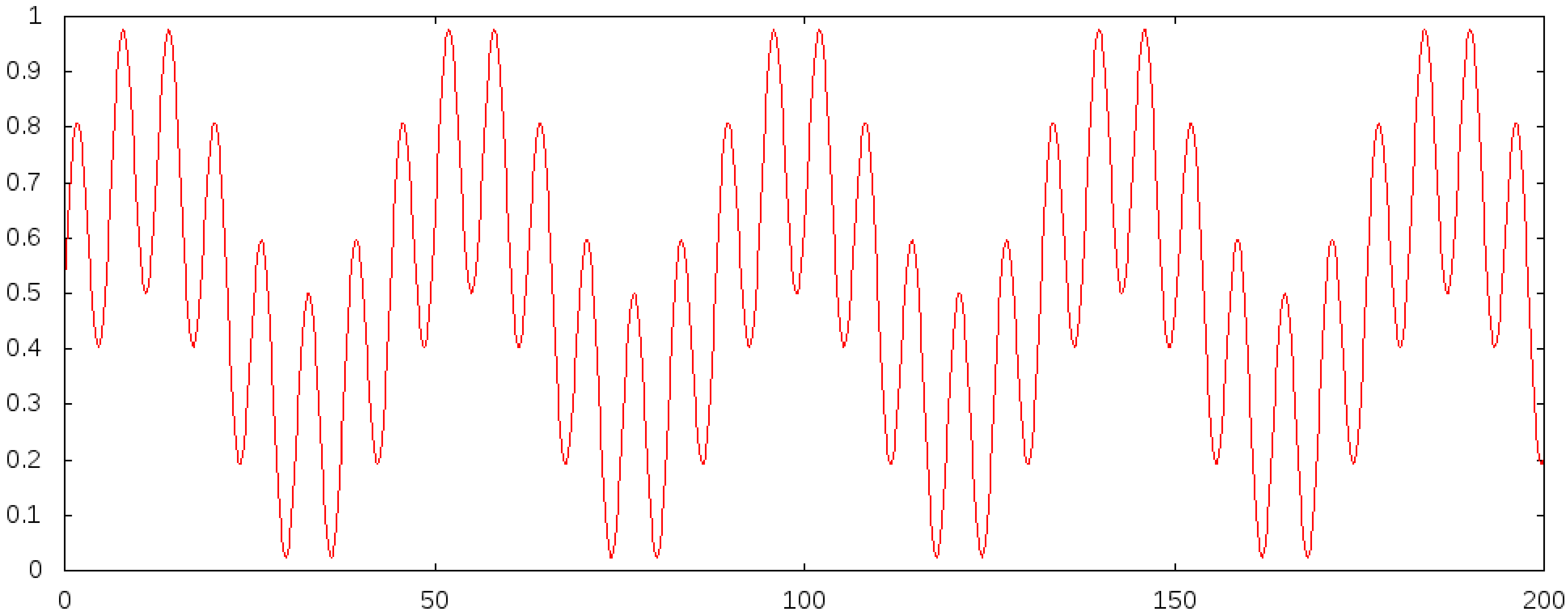
- What happens next?
  - It's a sine wave.



We can see known functions.

A Compound Example

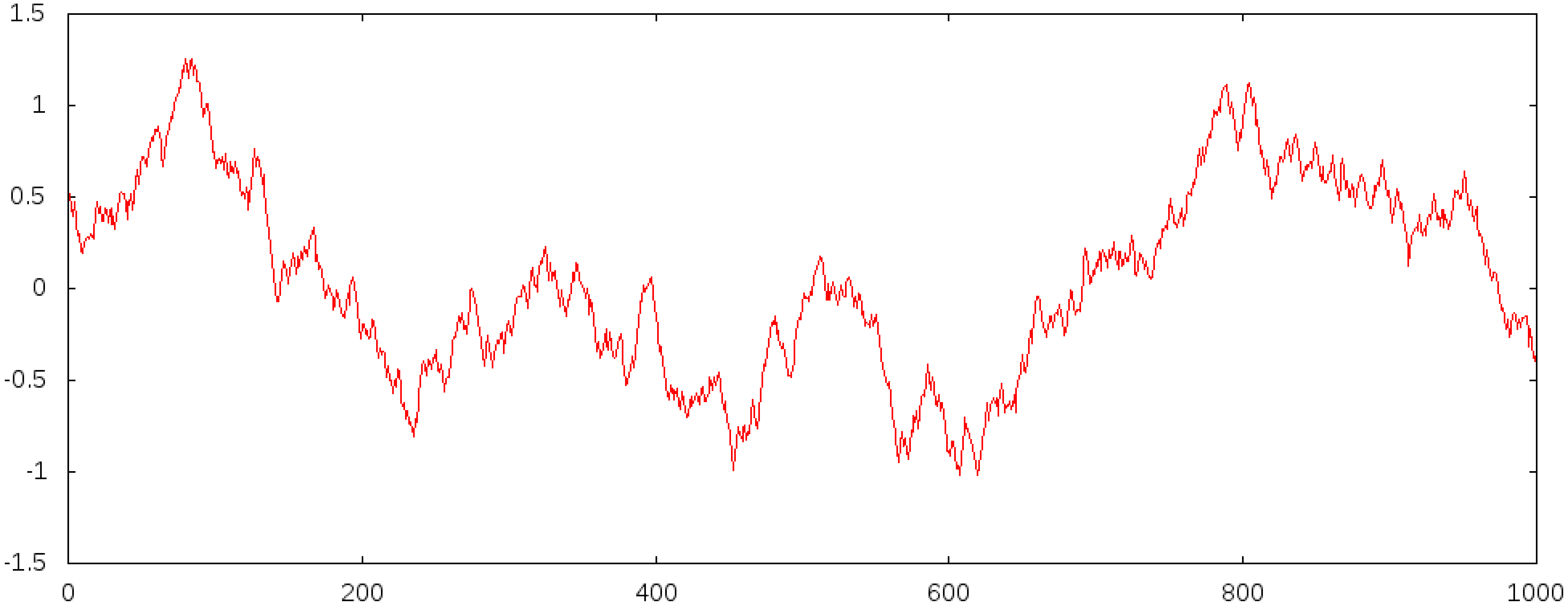
- What happens next?
  - Multiple things are happening.
  - We can discover and distinguish them.



We can detect complex similarities.

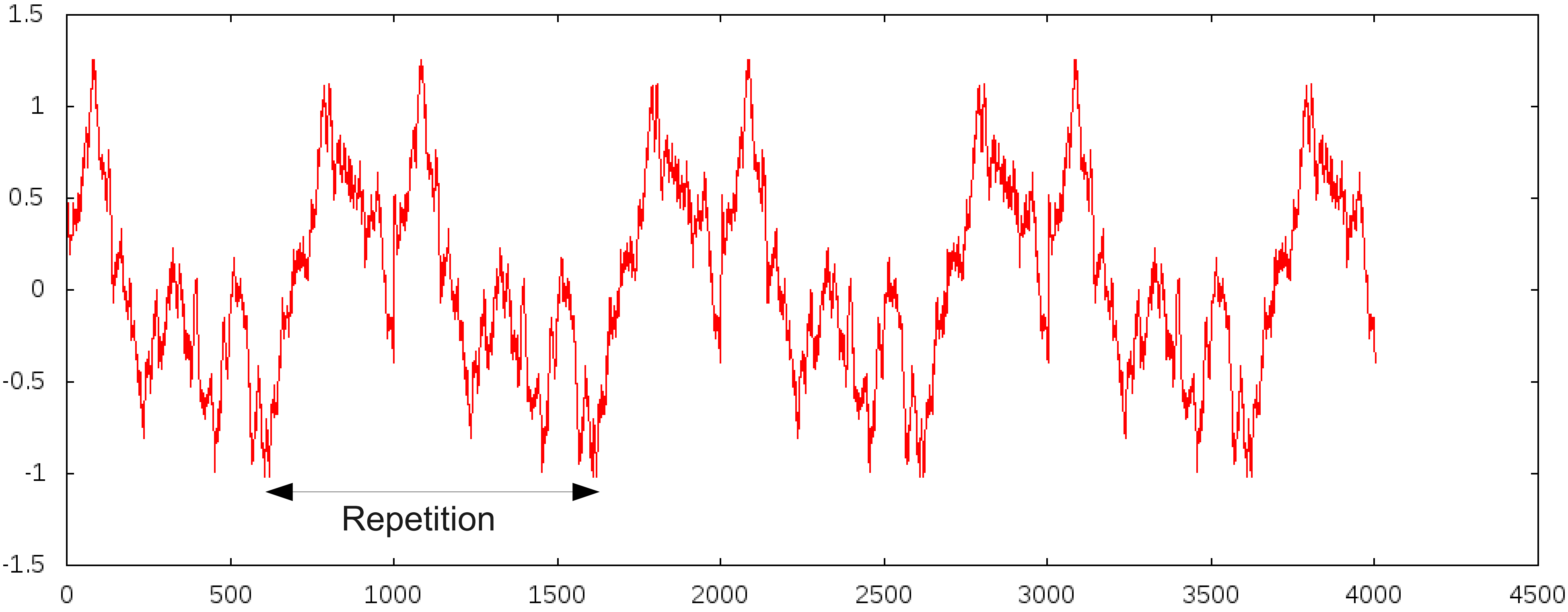
A Complex Example

- What happens next?



A Periodic Example

- It's periodic.
  - Just not analytic.
  - We can still detect this automatically.



And we can spot repetition.

Time Series Analysis

- Similarity of two functions.
  - Use one function to predict another.
  - Describe the unknown function in terms of the known function.

- Similarity of two functions at an offset in time.
  - Compute the offset as well as the relationship.

What Can We Match Against?

- We need one predictable function.
    - Known analytic function, e.g. sine, step, square.
    - Historic data from the same function.

Without Loss Of Generality

- Oh, melody divine!

  \- Major Bloodnok, 1957; (FX: cash register)

- We will consider only one data dimension.
  - You can generalize it yourself later.
    - As an exercise.
      - Homework due Tuesday.
        - You can download the answers from Wikipedia.
          - If you can read this, you're driving too close.

Trivial Causality Analysis

- Group by key (for example, user).
- Order each group by time.
- Match each group against a rule.

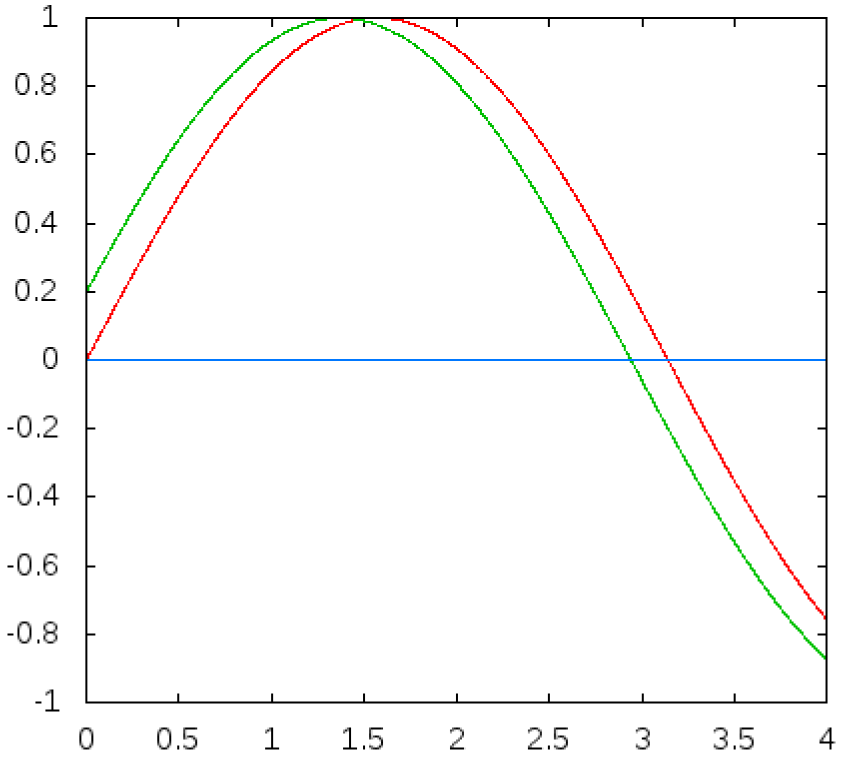
```
user0 →[ page0, page2 ]
user1 →[ page1, page3, purchase ]
```

- This is really a form of correlation, not time series analysis!
  - You can do this in Hive, Pig, Cascading, etc.

Boring! Let's do the real stuff.

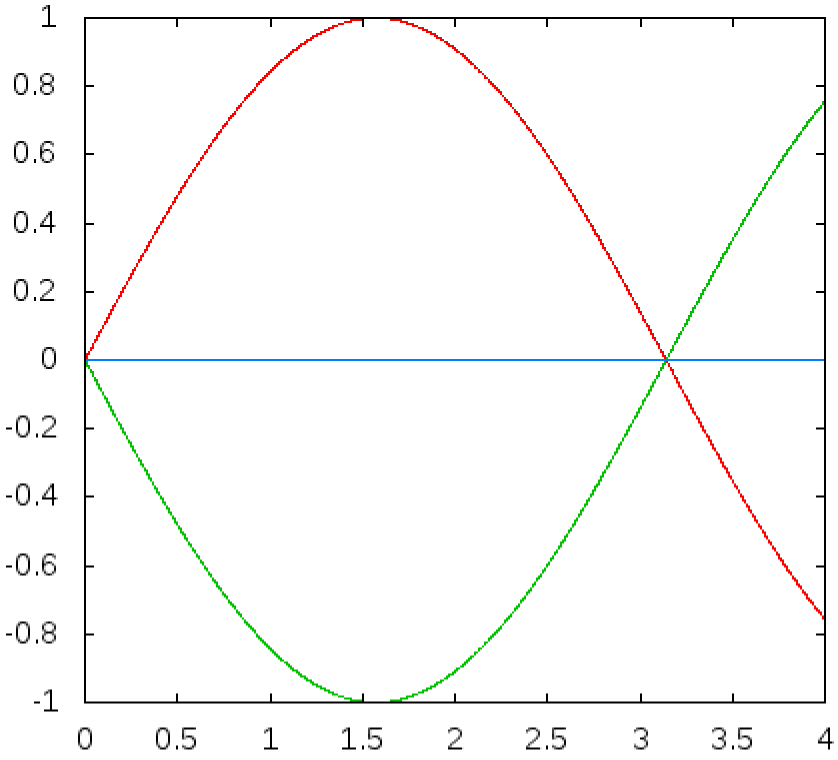
Intuition for the Mathematics

- Similarity



Positive product.

- Dissimilarity

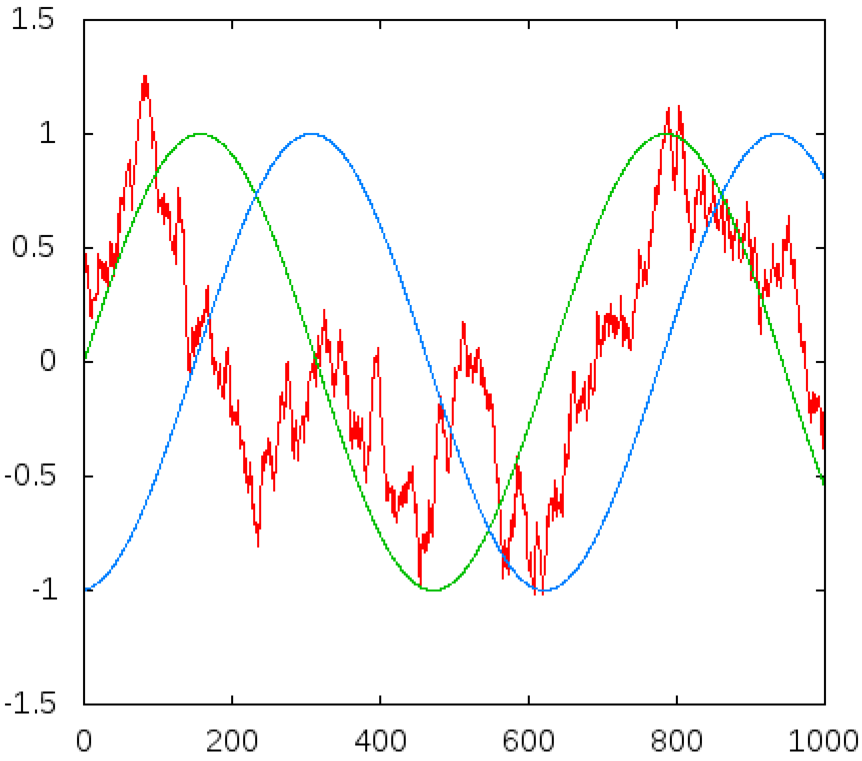


Negative product.

I'm about to cheat, but it doesn't matter.

Finding the Offset

- We compute the correlation at each offset.

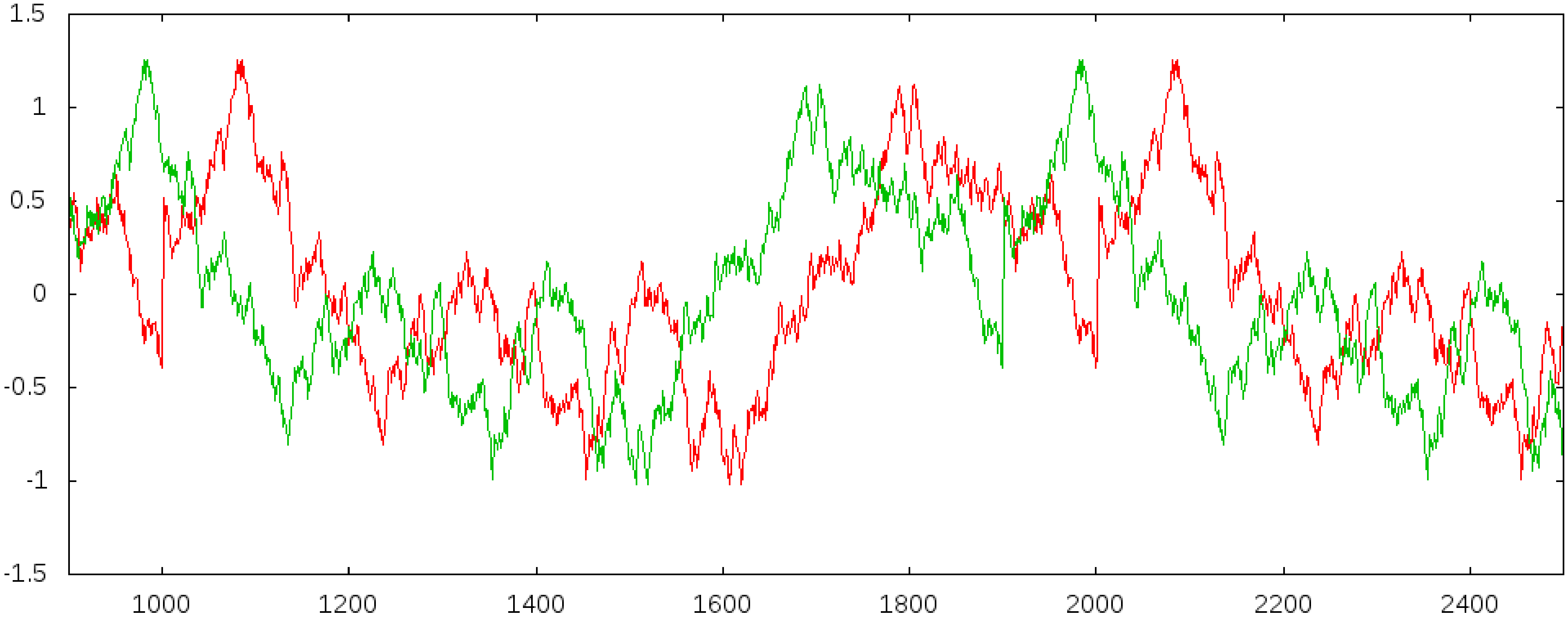


- We restrict the range of offsets using a window.

I kind of cheated, but not by a lot.

Autocorrelation

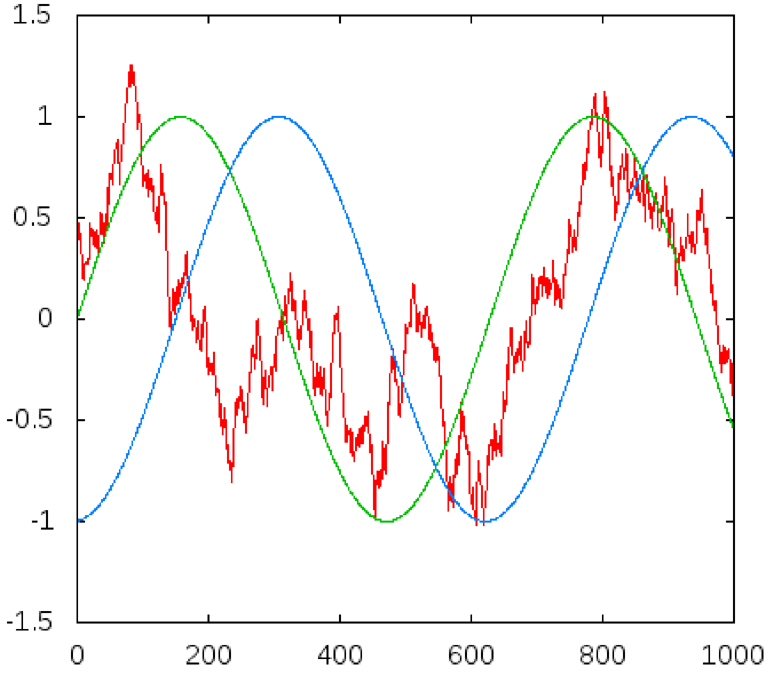
- The same, but correlation against itself.



- The principle of the computation is the same.

The Mathematical Statement

$$X_k = \sum_t f_t \, g_{t+k} \, w_k$$



- X_k is the similarity at offset k.
- t is the time offset.
- w_k is the windowing function.

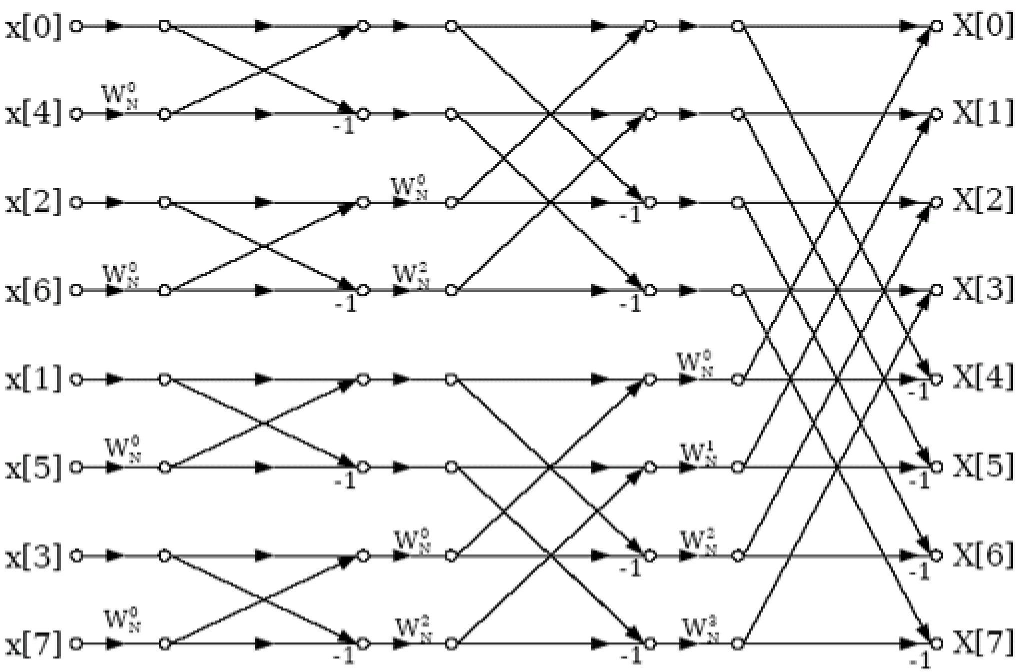
The Classical Algorithm

$$X_k = \sum_t f_t \, g_{t+k} \, w_k$$

- For each offset k:
  - For each time t:
    - Bother. You can't do that efficiently in shared-nothing.

What optimizations are available?

The Cooley-Tukey FFT Algorithm



- Optimized FFT for CPU, not data transfer.
- We can do this, but it's not great.

And that's only FFT, not cross-correlation.

Limitations of Shared Nothing

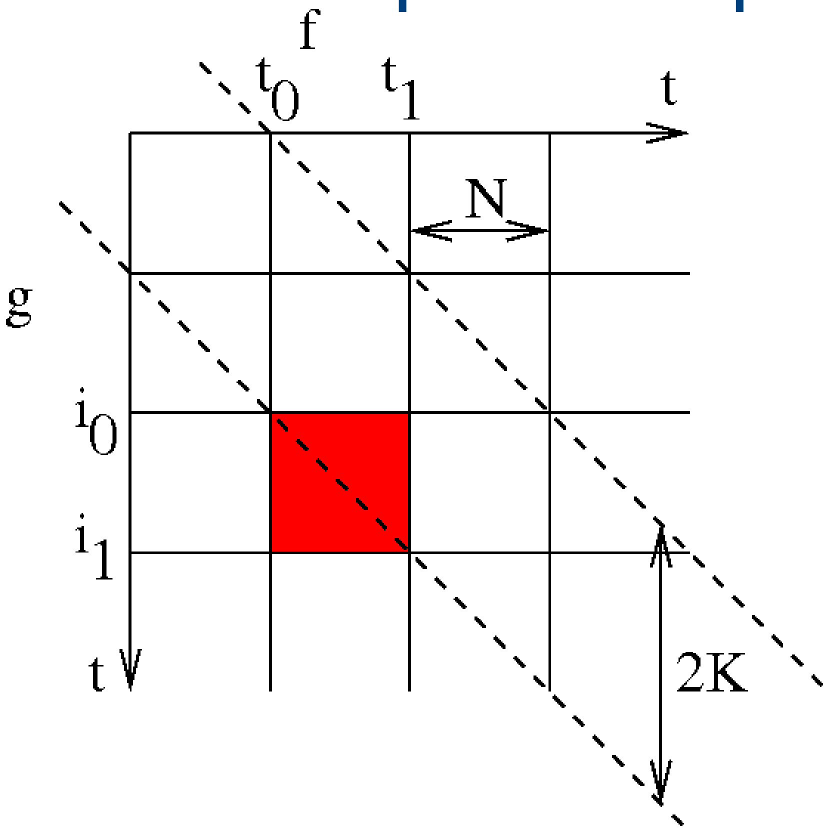
- We can't iterate over the array.
  - We can only see a part of it at a time.
- We have fixed memory size.
  - Memory size should be an input parameter for all big-data programs.
- We don't want $\log_2(n)$ MapReduce jobs.
  - We can do it with 2.

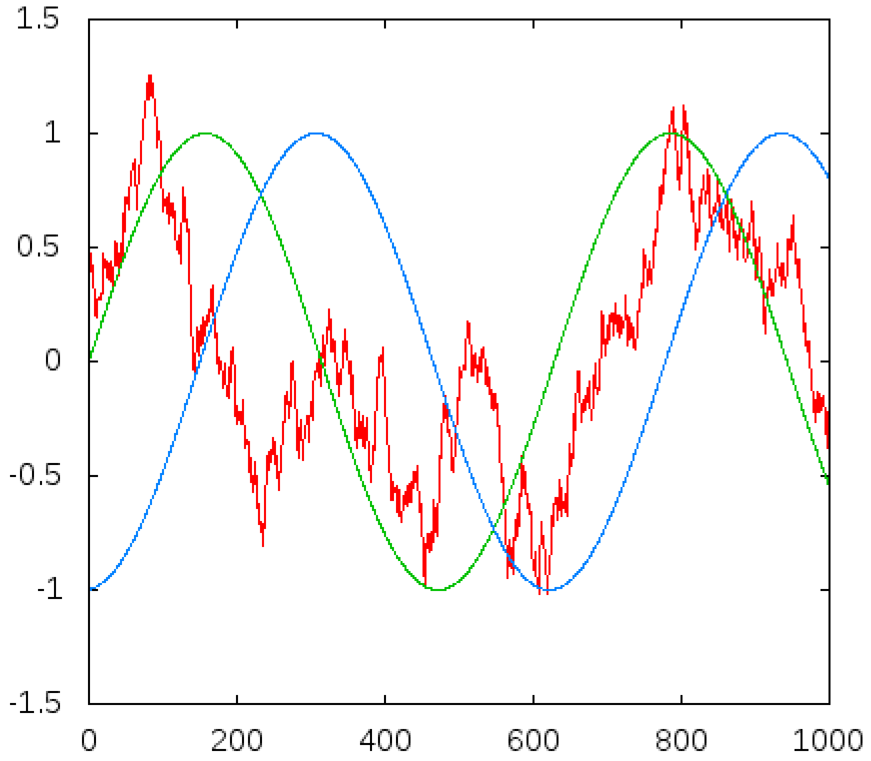
Data Flow Algorithm Design

- We need each element of f to meet each “nearby” element of g.
- We can do this block-wise.
  - Our data is dense, so we tile it.
- We allow, but do not require a windowing function.

The Data Flow Algorithm

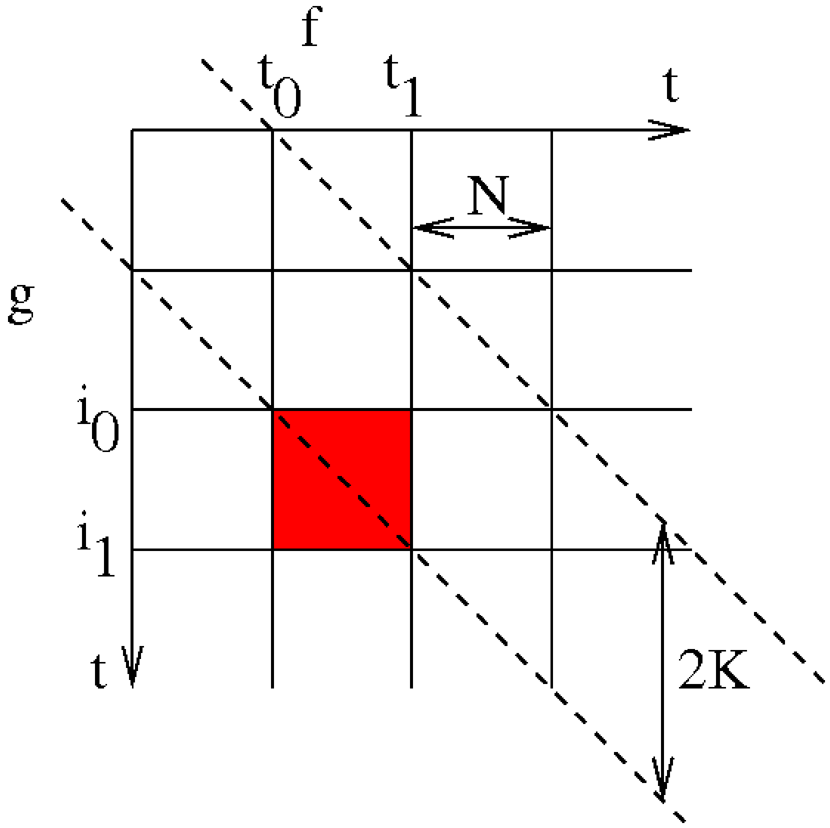
- We split each function into blocks of size N.
- We compute dot-product on each pair of blocks.

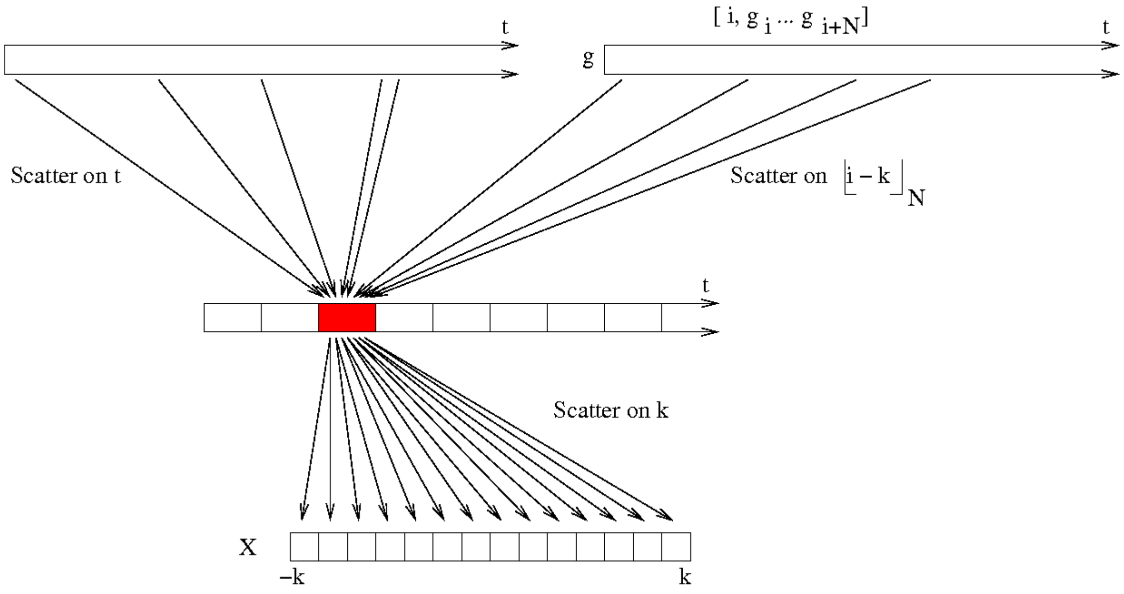




The Data Flow Algorithm

- We send each block of F to a reducer.
- We send each block of G to several reducers.

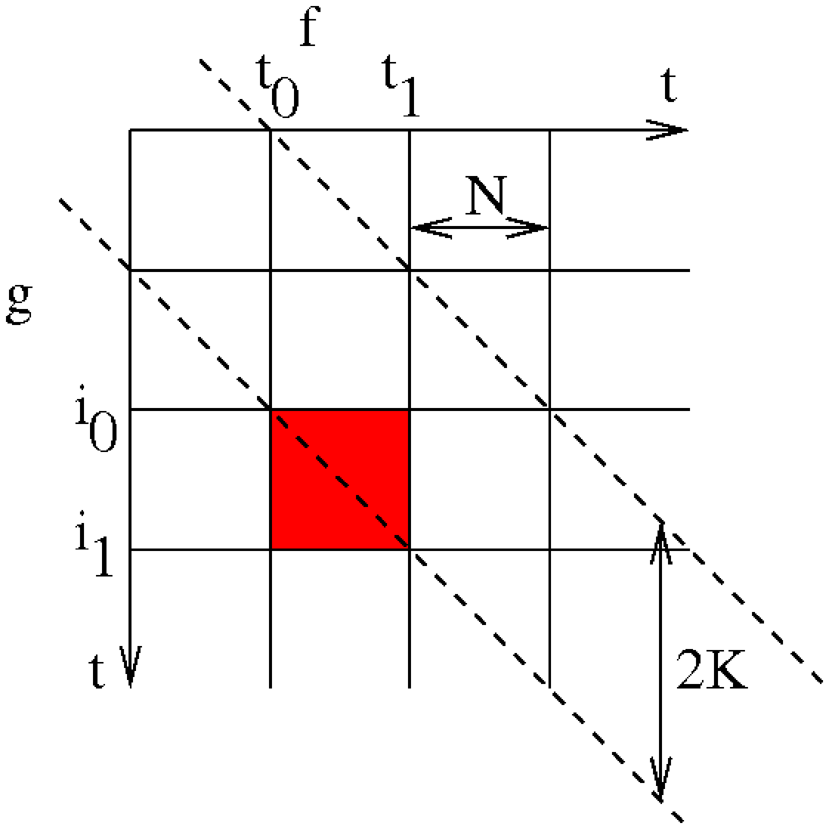


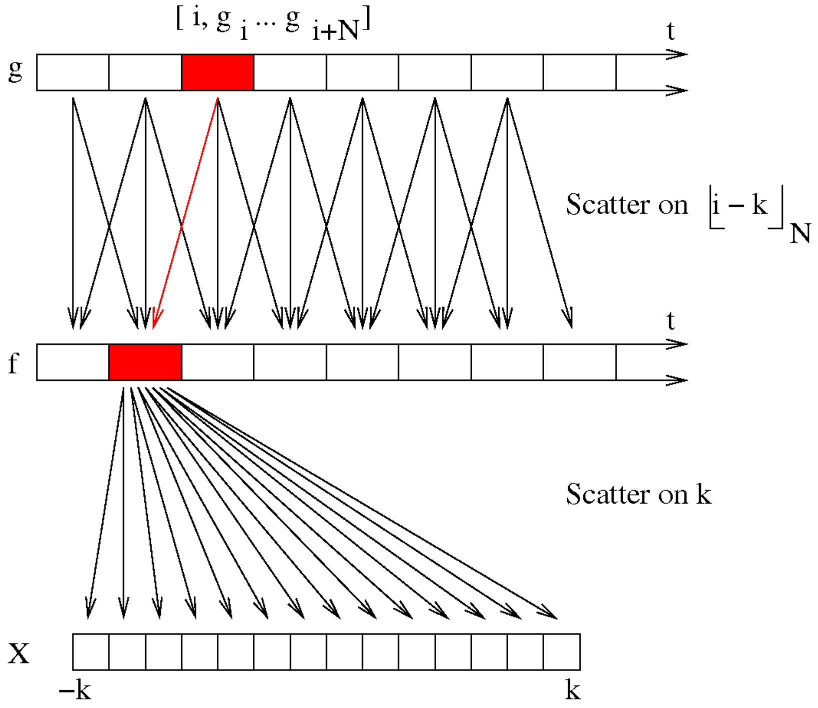


Our cheat doesn't matter any more.

The Data Flow Algorithm

- We could keep F in place, and just move G.
- Each block of G meets each nearby block of F.





MapReduce Formalism

- We will describe MapReduce in terms of four abstract operators:
  - Map
  - Scatter
  - Gather
  - Reduce

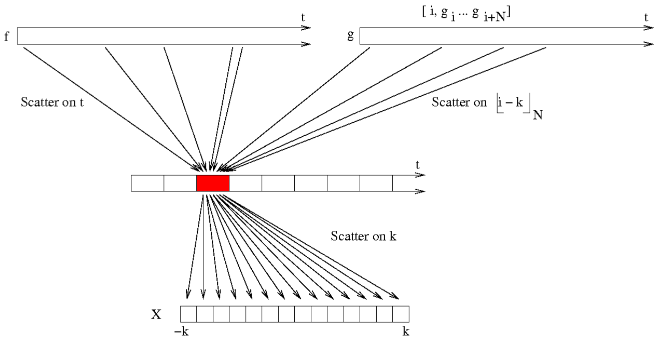
- This is quite a useful way to design jobs.

Aside: Data Flow Machines

- We are actually designing a Gamma workflow.
  - Gamma, 1983, DeWitt et al.
  - FlumeJava, 2009, Chambers et al.
  - Dryad, 2009, Isard and Yu.
  - etc.

MapReduce Implementation

$$\begin{tikzcd}
F \arrow[d, "\texttt{map}"] & & & G \arrow[d, "\texttt{map}(1)"] & \\
{[t_0, f_{t_0} \dots f_{t_1}]} \arrow[d, "\texttt{scatter}"] & \forall t_0 : N|t_0 & & {[u_0, g_{u_0} \dots g_{u_1}]} \arrow[d, "\texttt{scatter}"] & \forall u_0 : N|u_0 \\
{[t_0, \lfloor u \rfloor_N] \mapsto [t_0, f_{t_0} \dots f_{t_1}]} \arrow[rrd, squiggly, "\texttt{shuffle}"'] & & & {[\lfloor t \rfloor_N, u_0] \mapsto [u_0, g_{u_0} \dots g_{u_1}]} \arrow[ld, squiggly, "\texttt{shuffle}"] & \forall t \in [u_0 - K, u_0 + K] \\
& & \begin{array}{c} {[t_0, u] \mapsto [t_0, f_{t_0} \dots f_{t_1}],} \\ {[t, u_0] \mapsto [u_0, g_{u_0} \dots g_{u_1}]} \end{array} \arrow[d, "\texttt{gather}"] & & \\
& & {[t_0, u_0] \mapsto \{[t_0, f_{t_0} \dots f_{t_1}], [u_0, g_{u_0} \dots g_{u_1}]\}} \arrow[d, "\texttt{reduce}(2)"] & & \text{Here, } \lfloor u_0 + k \rfloor_N = t_0. \\
& & k = u - t,\, f_t g_u w_k \arrow[d, "\texttt{scatter}"] & & \forall t, u \text{ having } w_k \neq 0 \\
& & k \mapsto f_t g_u w_k, \dots \arrow[d, "\texttt{gather}"] & & \\
& & k \mapsto \{f_t g_u w_k, \dots\} \arrow[d, "\texttt{combine}"] & & \\
& & k \mapsto \sum f_t g_u w_k \arrow[d, squiggly, "\texttt{shuffle}"] & & \text{Partial sum.} \\
& & k \mapsto \sum f_t g_u w_k, \dots \arrow[d, "\texttt{gather}"] & & \text{Partial sum.} \\
& & k \mapsto \{\sum f_t g_u w_k, \dots\} \arrow[d, "\texttt{reduce}"] & & \text{Partial sum.} \\
& & X_k = \sum f_t g_u w_k & & \text{Full sum.}
\end{tikzcd}$$



(Same shape!)

$\forall x : N|x$ means “for each multiple of N”.

Analysis of the Algorithm

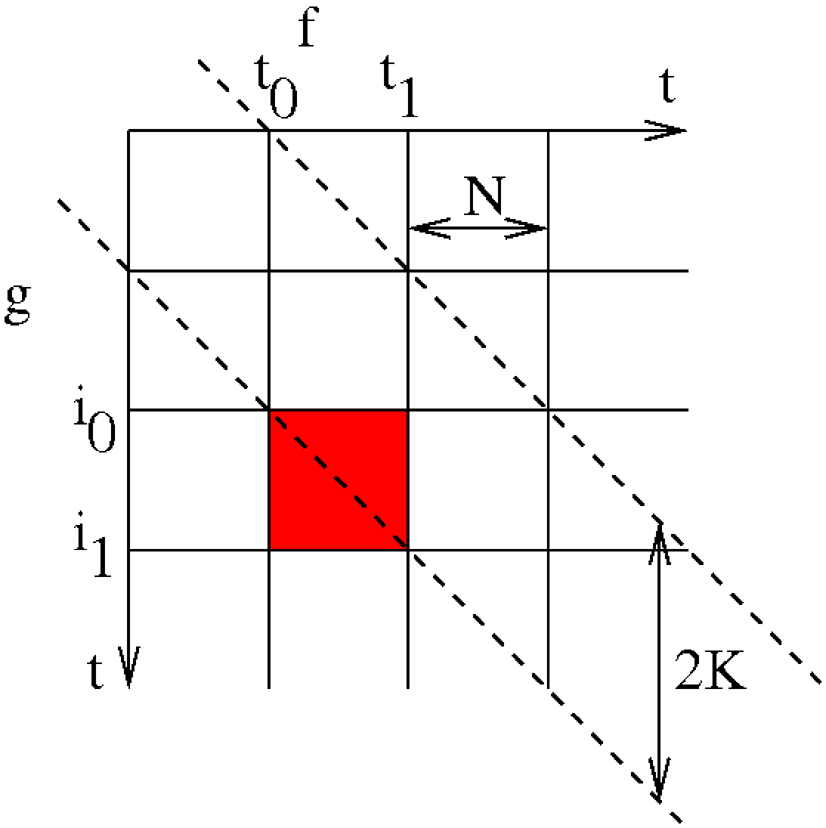
- We are optimizing for:
  - Number of jobs or stages.
  - Amount of I/O.
  - Amount of CPU.

- We can produce faster variations of the algorithm for standard specialist cases.

And now for the variations.

Variation no 1 in Eb Major

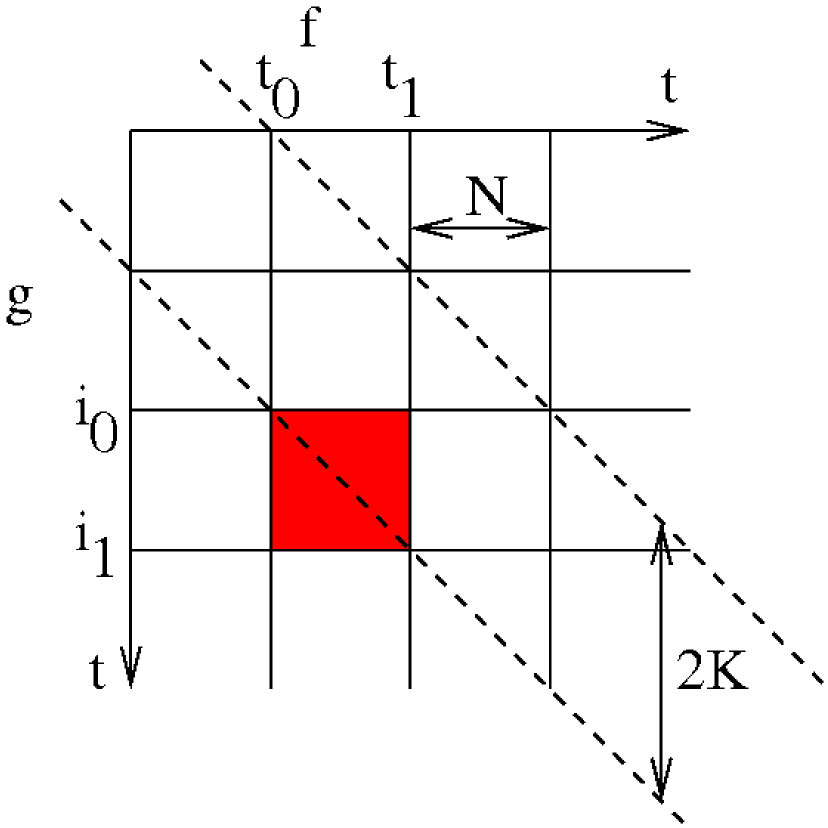
- If the window size is bounded:



If K is bounded, we transfer F and G $\lceil \frac{2K}{N} \rceil$ times.

Variation no 2 in C Minor

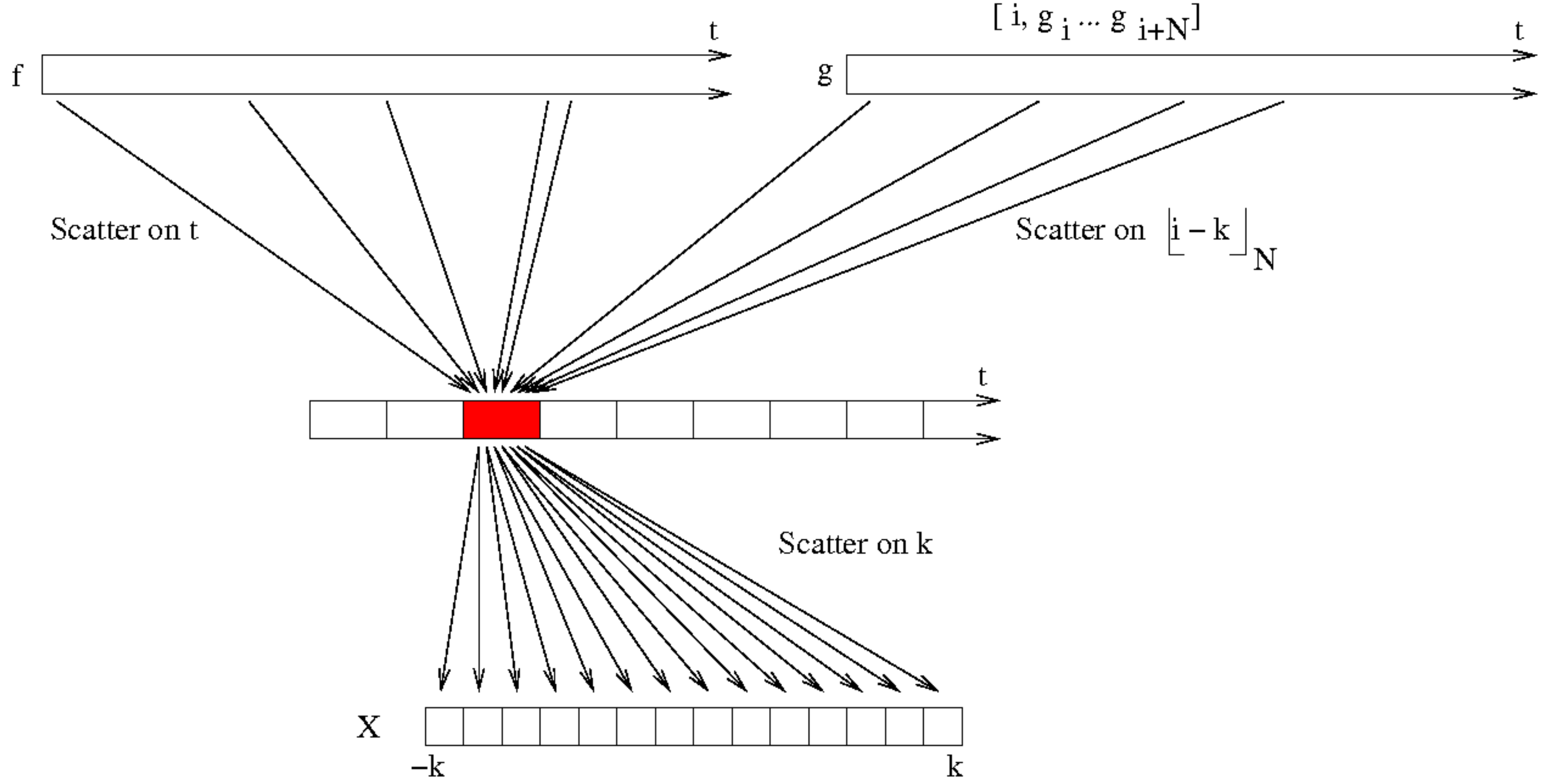
- If the window size is not bounded:



If K is not bounded, then we transfer each function $\frac{|F|}{N}$ times.

Variation no 3 in Bb Major

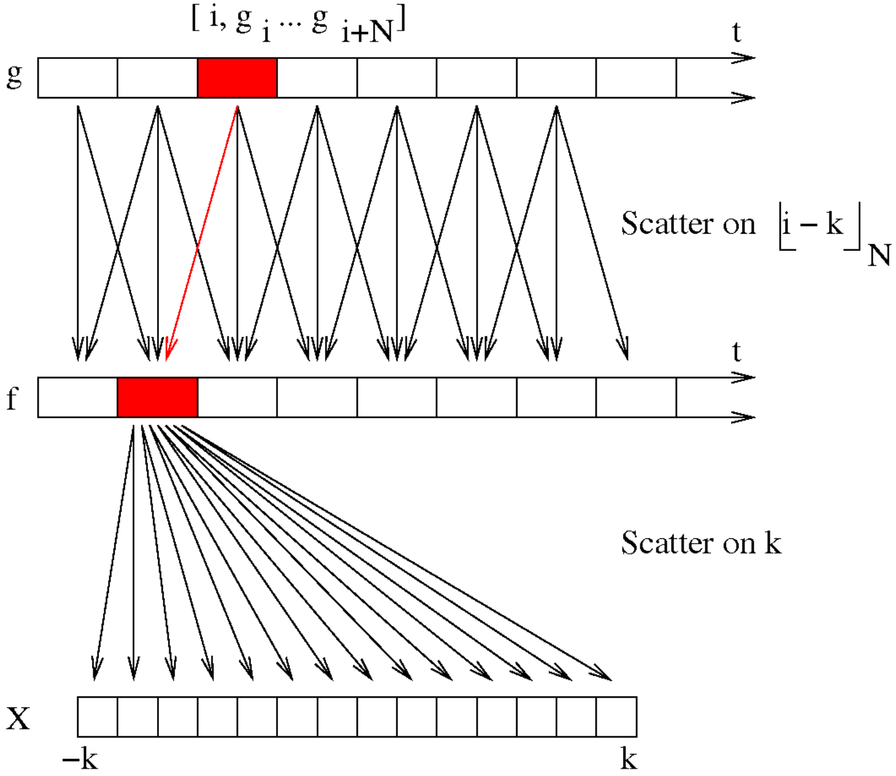
- If the known function is implicit:



If g is implicit, then we never transfer either function.

Variation no 4 in G Minor

- If we have placed reducers:



If we transfer each part of G to the node containing the pertinent part of F, we never transfer F at all, but we still transfer G $\left\lceil \frac{2K}{N} \right\rceil$ times. This requires:

- Explicit placement of reducers, or ability to selectively read from F directly.
- Either fixed-size records or some other way to identify the location of each block of F by index.

Conclusions

- We can do real time series analysis in MapReduce.
- A family of faster variants exists which provides for the specialized cases.
- MRSG is a useful way to think about shared-nothing.
  - That was the original point!

Also, I kind of enjoyed it, and I learned a lot.

Questions, Errata, Heckling

I can't help but use this slide. My friend drew it.

Karmasphere

The Leader in Big Data Intelligence Software

www.karmasphere.com