Sparse Adapton

Low-Overhead Incremental Computation

By Kyle Headley

kyle.headley@colorado.edu

Incremental Computation

A computation is incremental if repeating it with a changed input is faster than from-scratch recomputation

Spreadsheet

Spell check

Web Page

Adapton – General Purpose Incremental Computation

- Memoization
- Demand driven computation
- Dynamic dependency tracking
- Demand driven incremental updates

Problem:

General purpose incremental computation relies heavily on memoized results, which requires a lot of memory

Problem:

General purpose incremental computation relies heavily on memoized results, which requires a lot of memory

Solution:

Memoize fewer results!

It works!

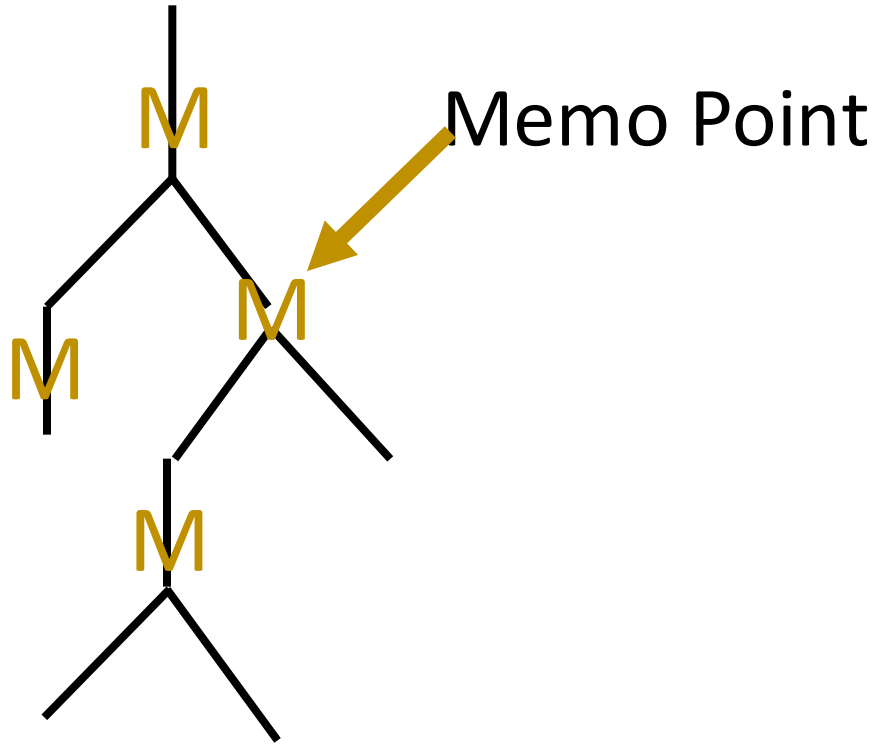
- Required memory is reduced
- Memory management is eased
- Update times are decreased

We gain control over the balance between speed and memory

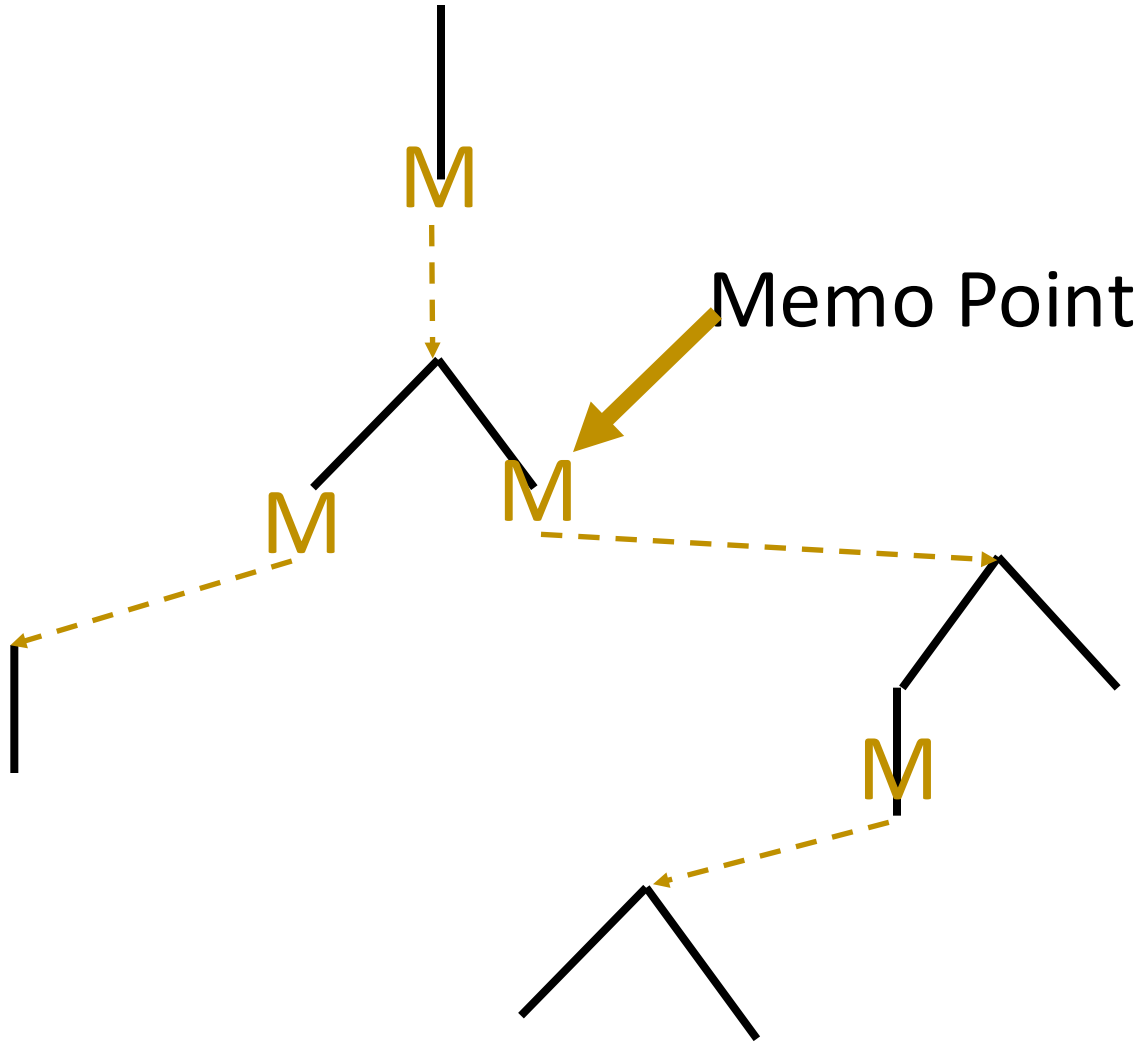
But of course, there are complications…

We can no longer use the strategy ‘memorize every function call’, we have to make a choice.

Working with data structures



Working with data structures



D – Data

M – Memo Point

D D D D D D D D D D D D D D D D D

D – Data

M – Memo Point

D D D D D D D D D D D D D D D D D

Insert Memo Points Evenly

D DMD DMD DMD DMD DMD DMD DMD D

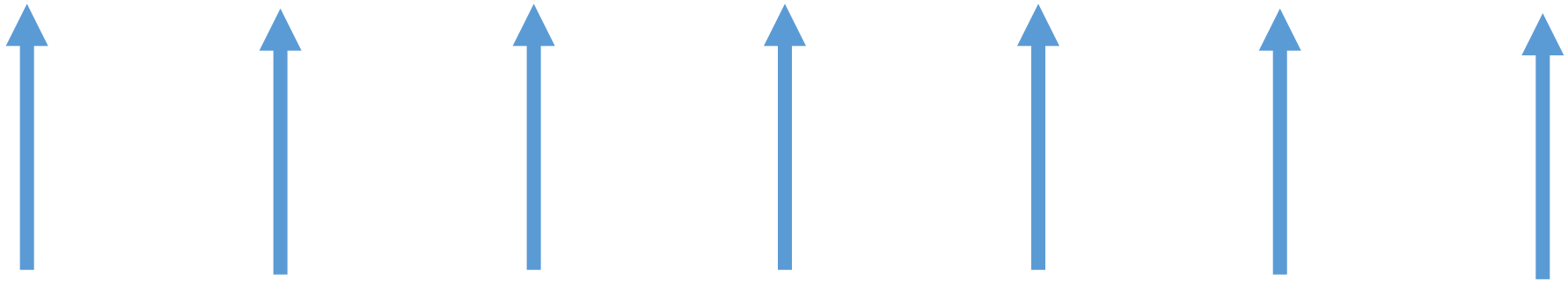
D – Data

M – Memo Point

D D D D D D D D D D D D D D D D D D

Insert Memo Points Evenly

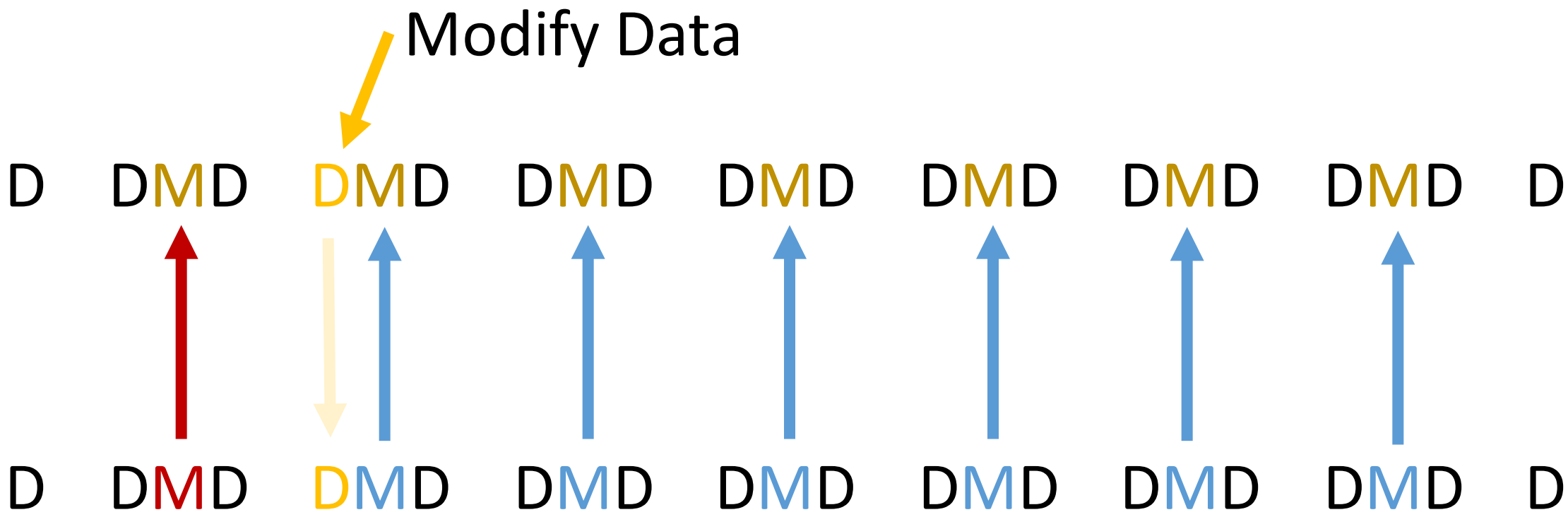
D DMD DMD DMD DMD DMD DMD DMD D



D DMD DMD DMD DMD DMD DMD DMD D

D – Data

M – Memo Point

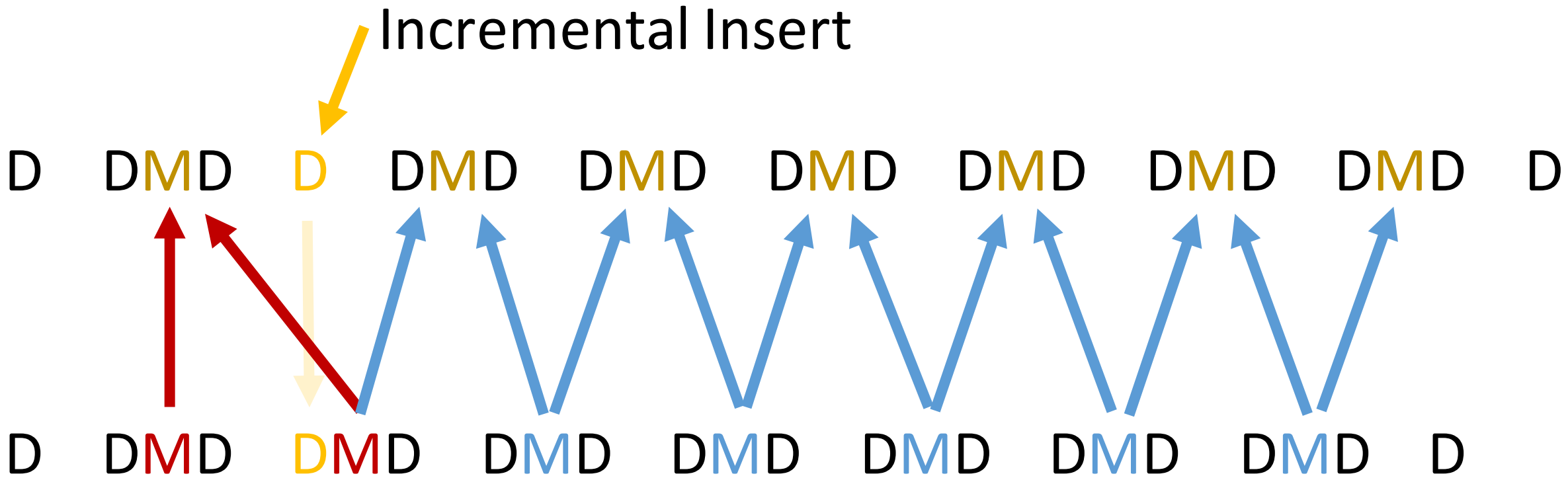


D – Data

M – Memo Point

D – Data

M – Memo Point



D – Data
M – Memo Point

Probabilistic distribution

D D D D D D D D D D D D D D D D D

Hash and Insert Memo Points

D DMD D D DMD DMD D D D DMD D D D

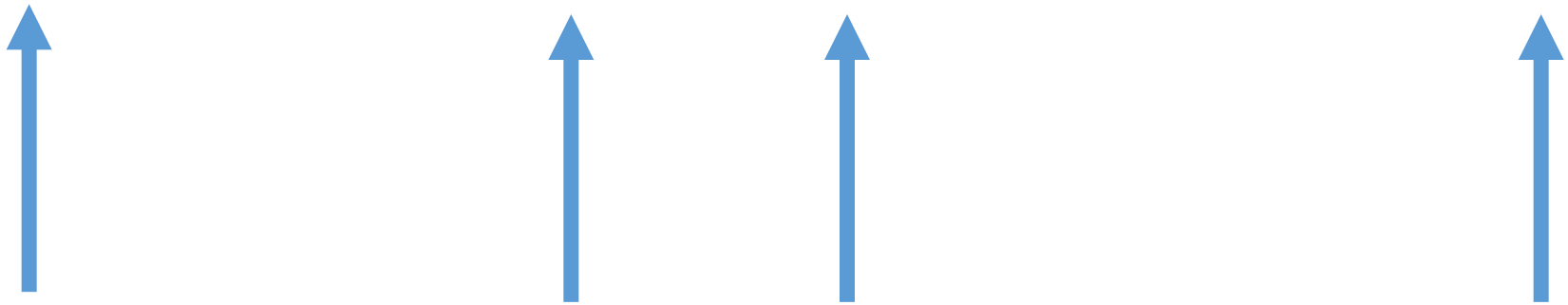
D – Data

M – Memo Point

D D D D D D D D D D D D D D D D D

Hash and Insert Memo Points

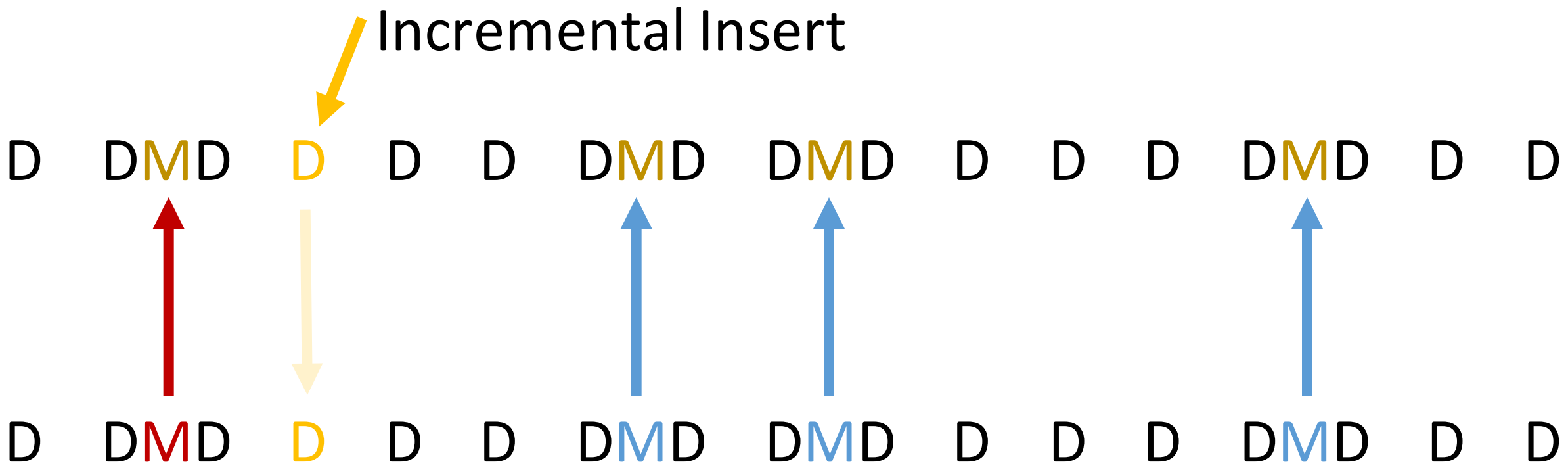
D DMD D D DMD DMD D D D DMD D D D



D DMD D D DMD DMD D D D DMD D D D

D – Data

M – Memo Point

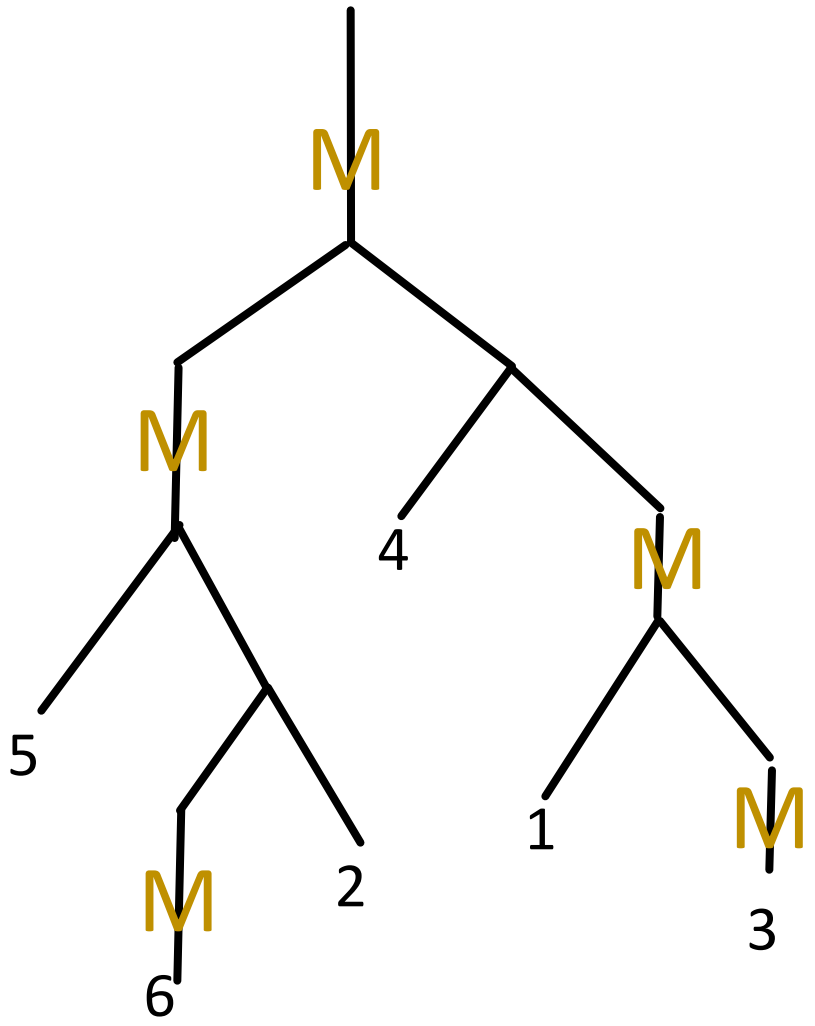


Now we’re committed to referencing a previous memo point to create a new one

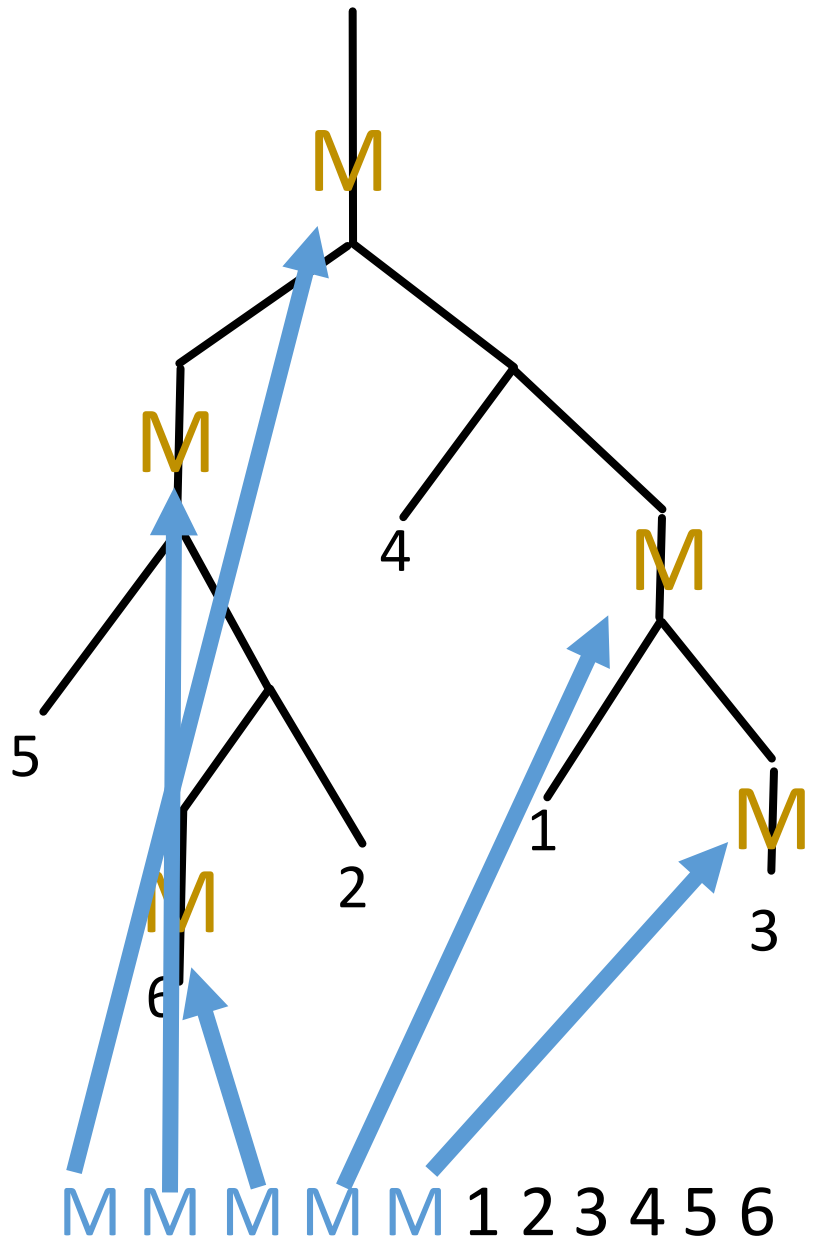
Divide and Conquer Example

Input list M5M6 2M4M1M3

Build Balanced Tree



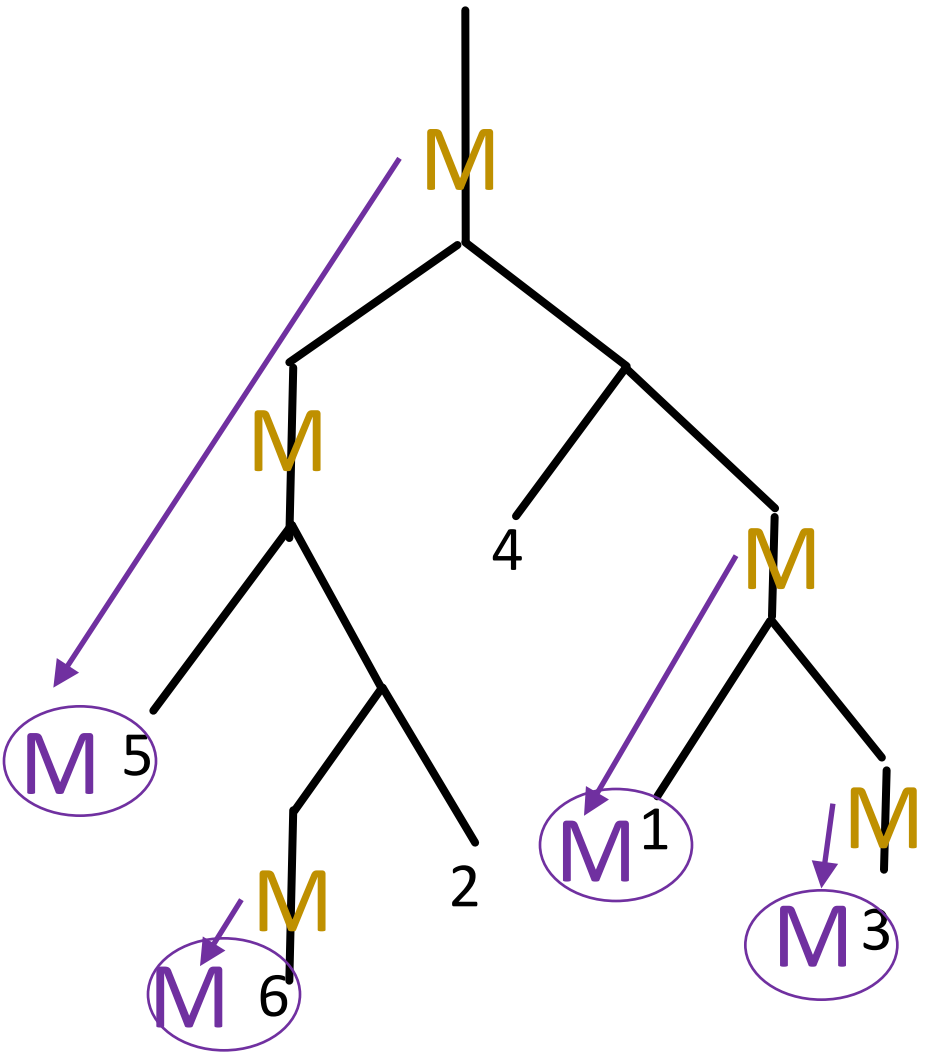
Build Balanced Tree



Poor Results:

Emit a memo point when you encounter a memo point.

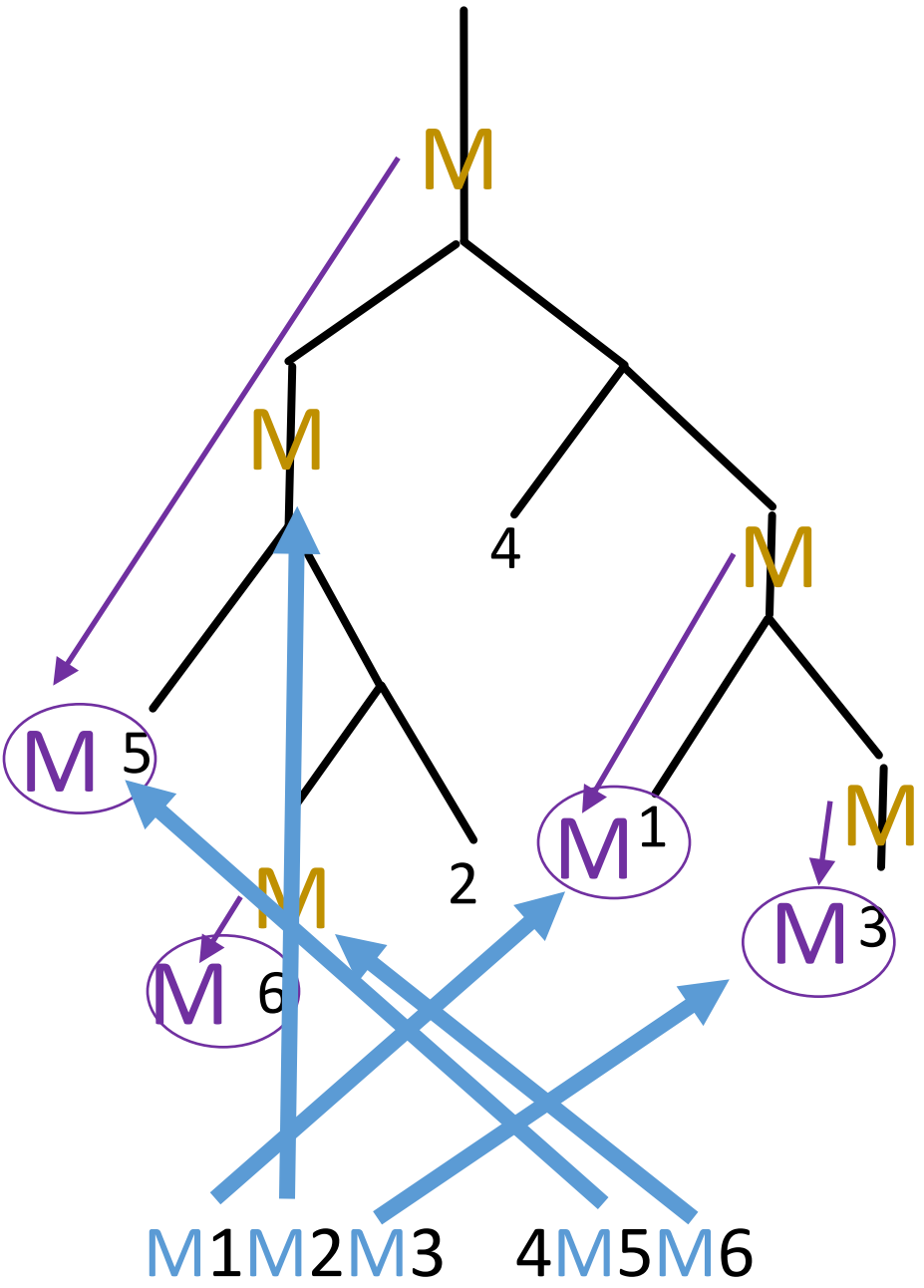
Build Balanced Tree



Pass References to memo points as parameters:

First class names

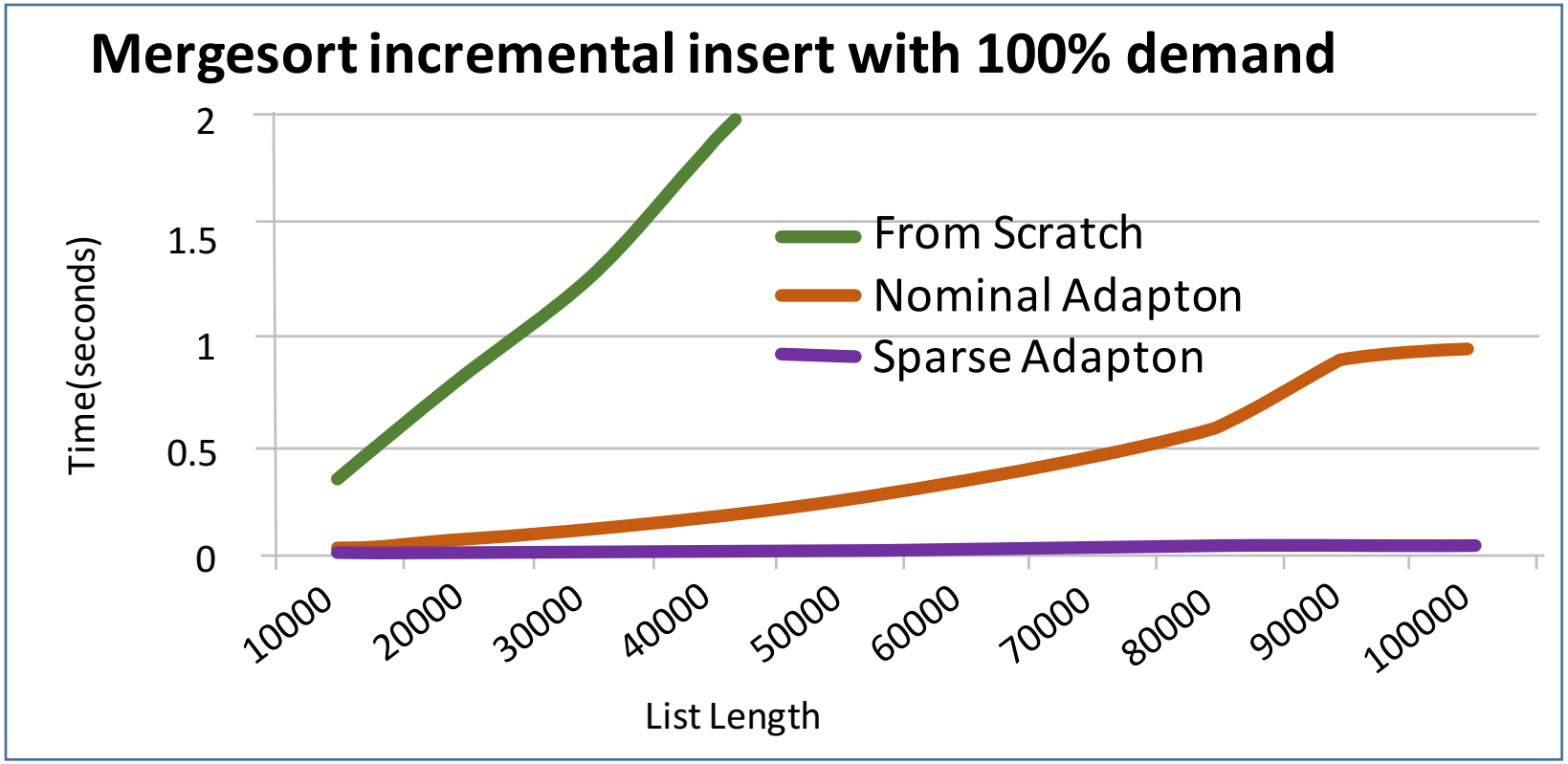
Build Balanced Tree



Good Results:

Maintain association through merge steps

Success!



Onward

- Increasingly complex algorithms
- Automation through libraries
- Interactivity of common tools

Conclusion

- Increase incremental efficiency by managing overhead
- Memo point placement matters
- Associate memo points with data

kyle.headley@colorado.edu