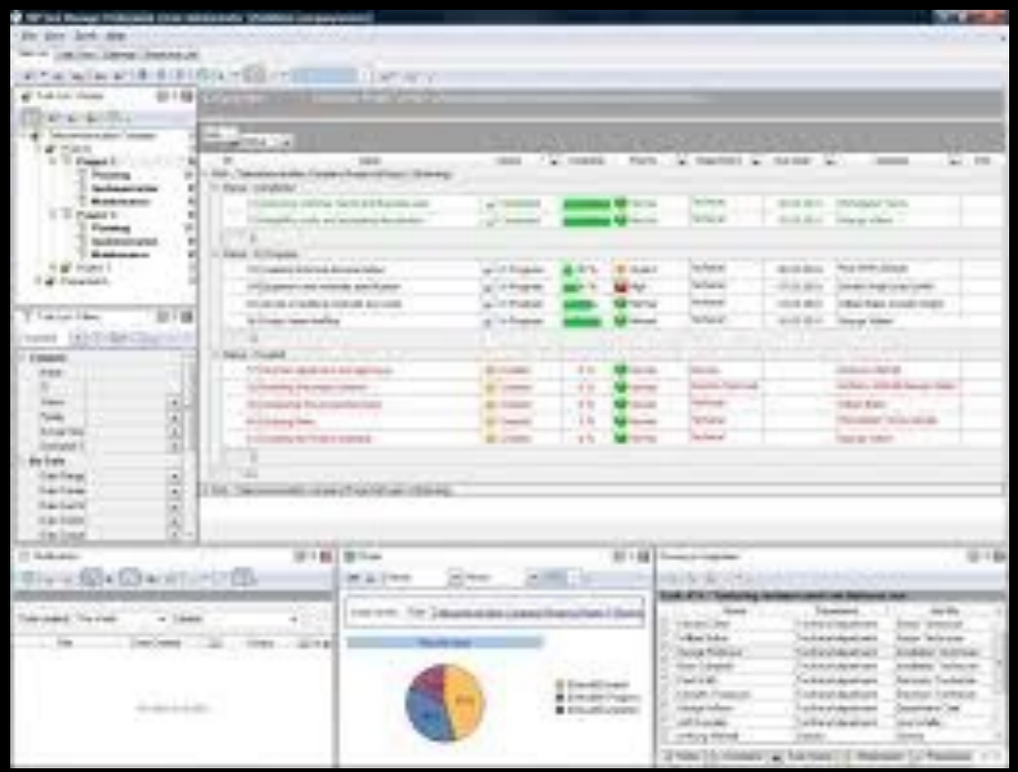
Correct-by-Construction Interactive Software: From Declarative Specifications to Efficient Implementations

Kyle Headley

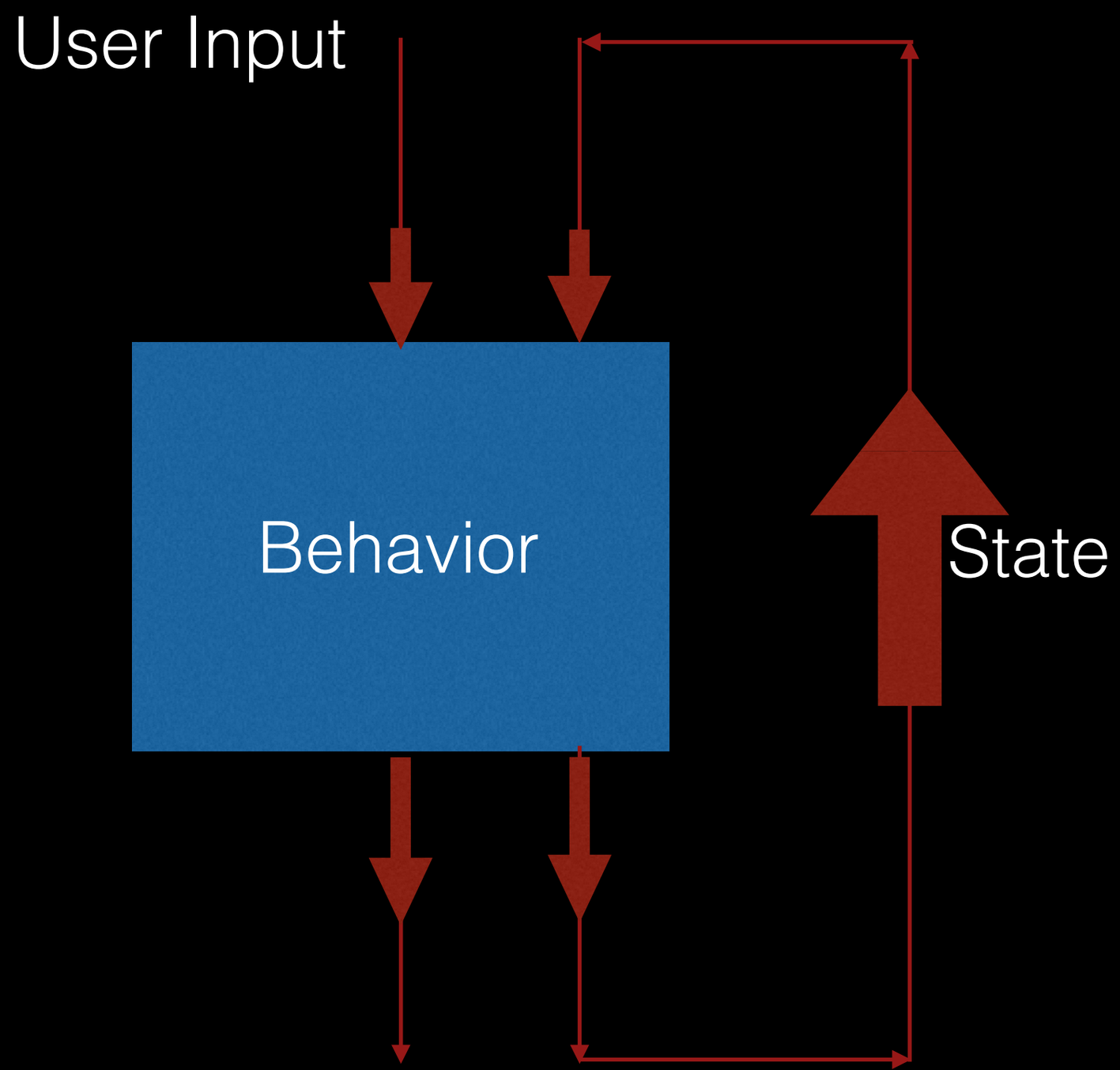
Matthew Hammer

OBT ‘16

Interactive System



Interactive System



Interactive System

Programming interactive experiences is ***complex***

Existing interactive experiences are ***underspecified***

Complex examples

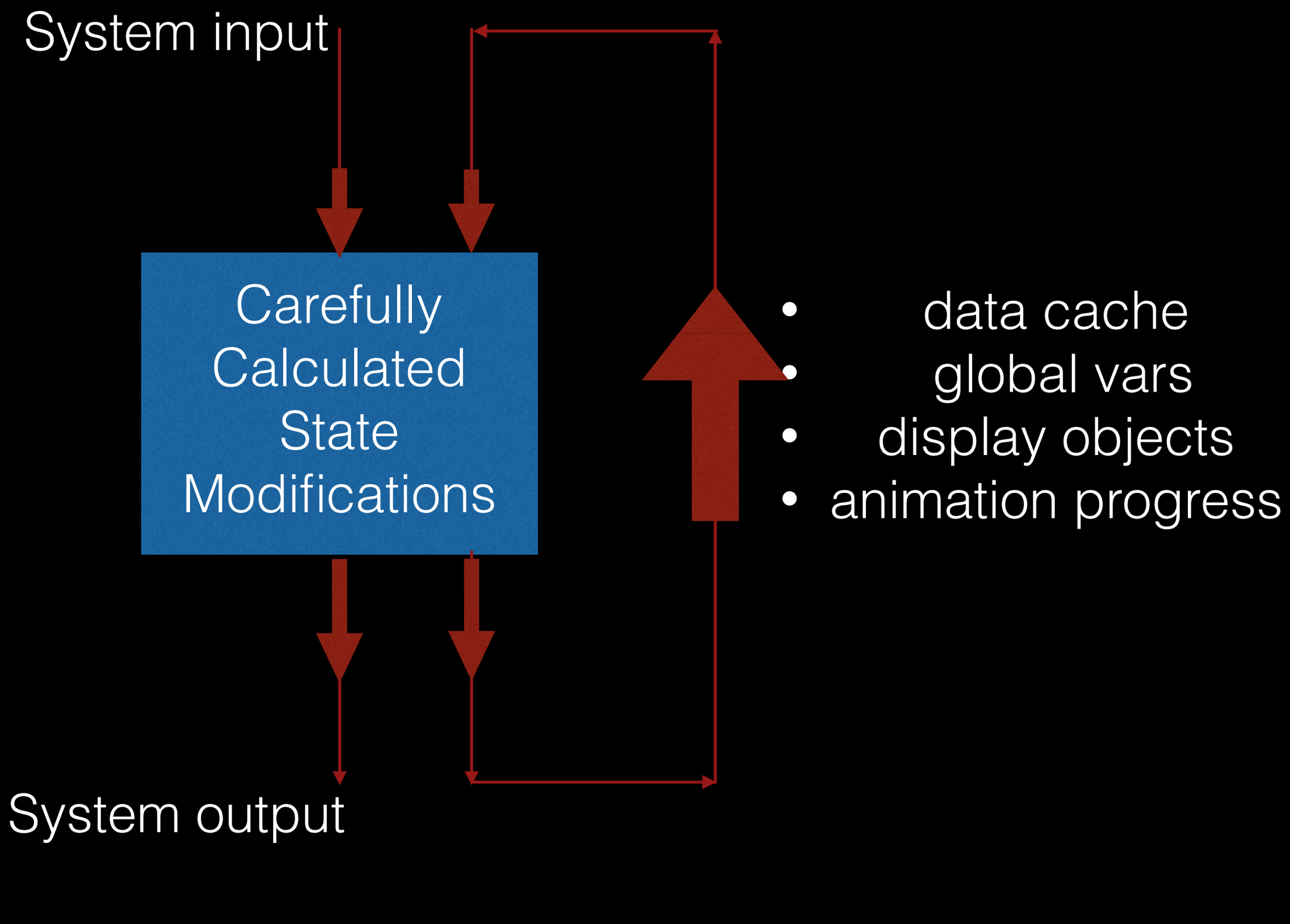
Obvious

- Extending emacs/eclipse

Subtle

- What is the semantics of UNDO?
- Undo with multiple buffers?
- Undo over higher order commands?

Interactive System



Using PL Features, Algorithms, Techniques
for interactive behavior

Programming interactive experiences can be ***simple***

Existing interactive experiences can be ***unambiguous***

Code can be ***independent*** of the framework used

Correct-by-construction Interactive Program

Semantics preserving transformation

Logical Spec

Deterministic functions

Functional Programming

Executable Spec

Library Integration

HCI Considerations

Correct Implementation

Incremental Computation

Optimisations

Responsive Implementation

Correct-by-construction Interactive Program

Logical Spec

Deterministic functions

Functional Programming

Executable Spec

Library Integration

HCI Considerations

Correct Implementation

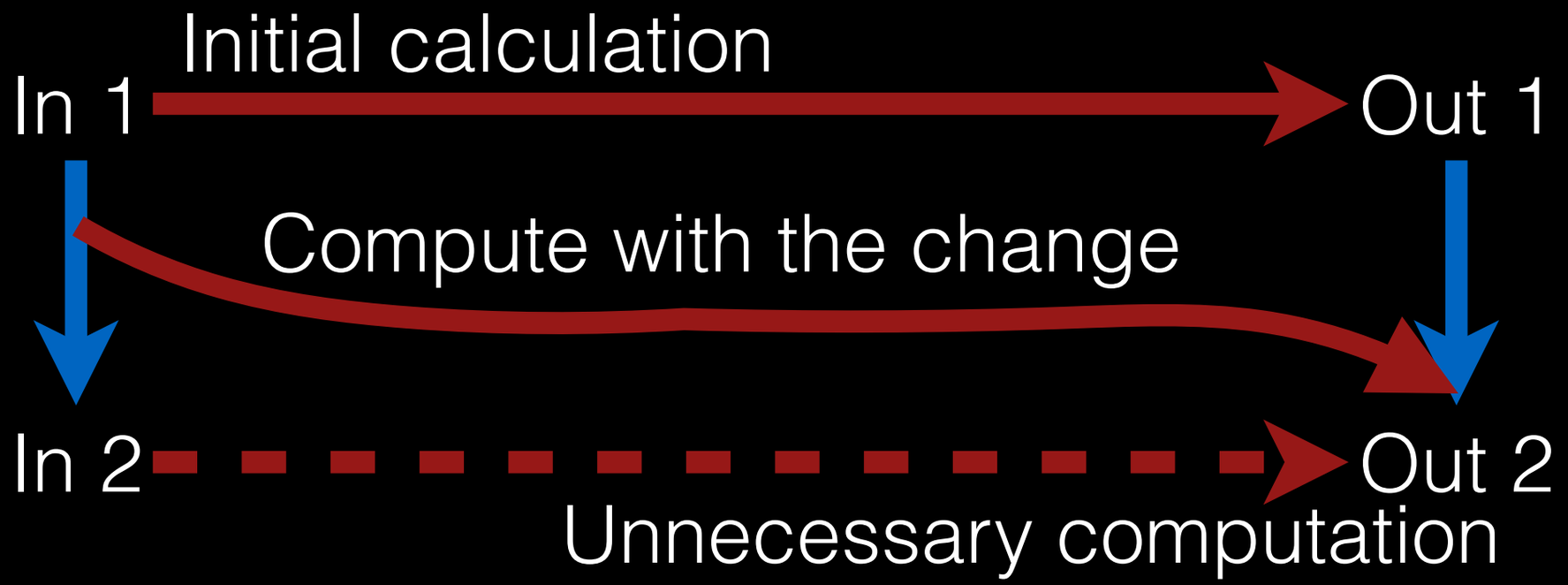
Incremental Computation

Optimisations

Responsive Implementation

Incremental Computation

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



Incremental Computation

Incremental computation can use:

- Memoization / Caching
- Dependency graphs
- Internal efficient data structures

Incremental Computation

An incremental computation library may be simple or complex

```
fn cmdz_of_actions
  <A:Adapton
  ,Acts:TreeT<A,Action>
  ,Cmds:TreeT<A,Command>
  ,Edit:ListEdit<A,Command,Cmds>
  >
  (st: &mut A, acts:Acts::Tree)
   -> (Edit::State, Option<A::Name>) {

...
let z = Edit::insert(st, z, Dir2::Left, c) ;
let z = Edit::ins_cell(st, z, Dir2::Left, nm.clone()) ;
let z = Edit::ins_name(st, z, Dir2::Left, nm) ;
...
```

```
fn foo(a: u32) -> u32 {
   ...
}
memo!(ic_tables, foo, a:a)
```

Proof of Concept: Text Editor

IC_Edit

Features:

- Text creation/removal
- Undo/Redo
- Multiple ‘cursors’

Proof of Concept: Text Editor

Logical Spec

Deterministic functions

Functional Programming

Executable Spec

Library Integration

HCI Considerations

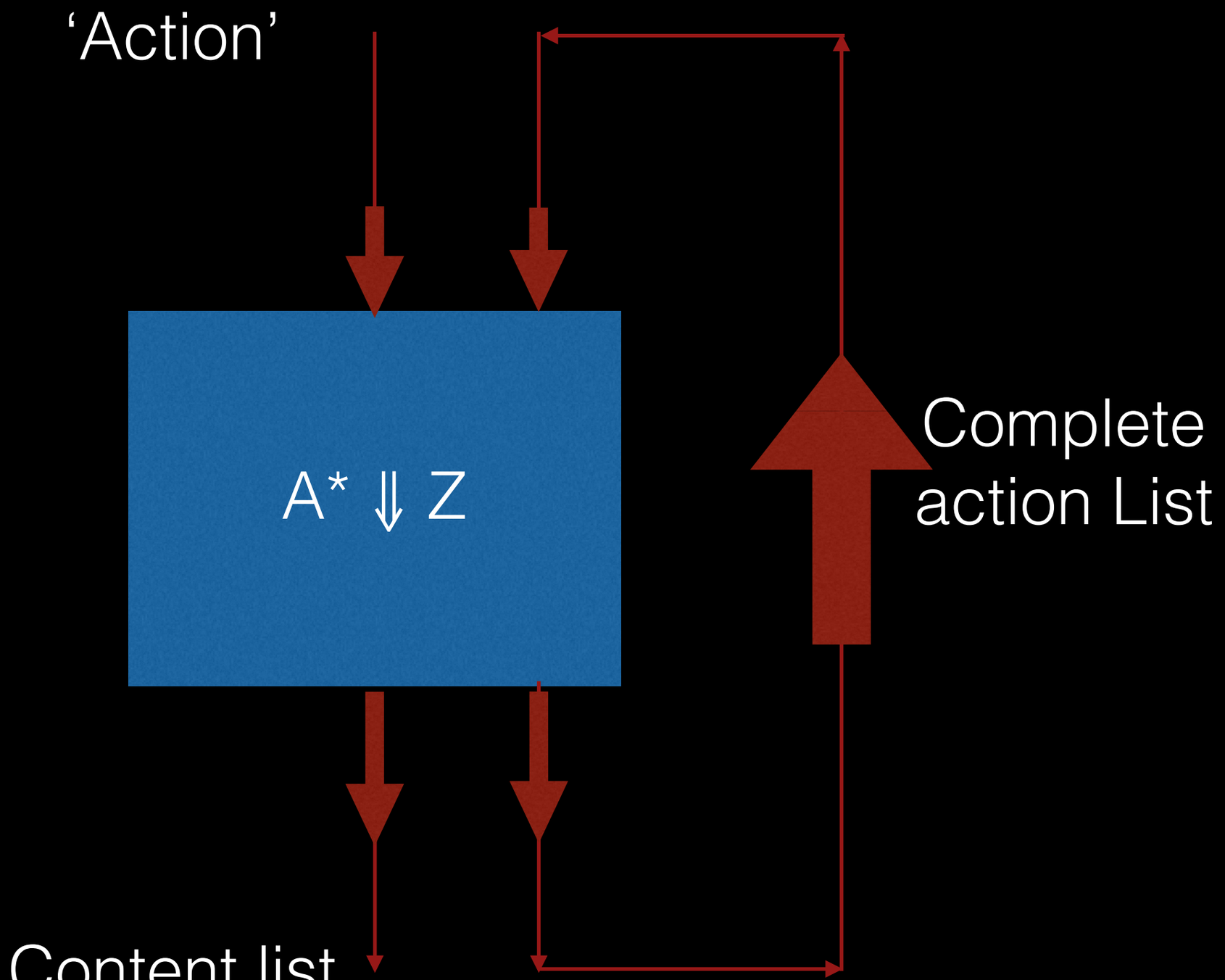
Correct Implementation

Incremental Computation

Optimisations

Responsive Implementation

IC_Edit



IC_Edit

Primitive operations

Edits

Insert, Overwrite, Delete, Undo, Redo

Cursor Management

Move, Make, Jump, Switch, Join

IC_Edit

Primitive operations

Edits

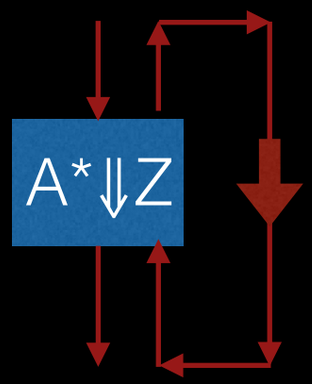
Insert, Overwrite, Delete, Undo, Redo

Cursor Management

Move, Make, Jump, Switch, Join

Demo — Correct Implementation

IC_Edit Spec



Build content from action list

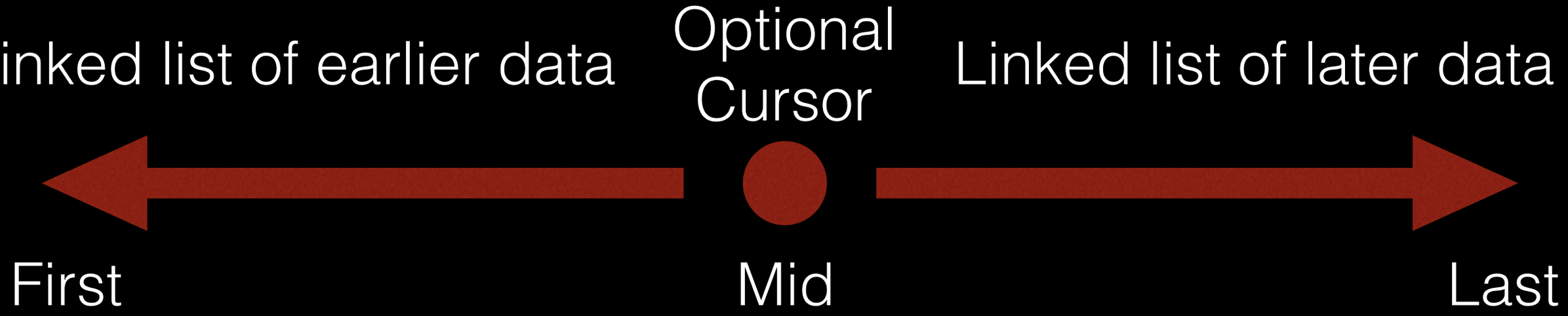
Initial cursor α fresh

Handle Undo/Redo $A^* \Downarrow \langle\!\langle C_1^* \| C_2^* \rangle\!\rangle$

Build content $\langle\, \varepsilon \mid \alpha \mid \varepsilon \,\rangle \vdash \mathrm{reverse}(C_1^*) \Downarrow Z$

$$A^* \Downarrow Z$$

Zipper



Links progress away from cursor

IC_Edit Spec

$$A^* \Downarrow \langle\!\langle C_1^* \| C_2^* \rangle\!\rangle$$

A ::= UNDO | REDO | Command

A^*: Action List

C_1^*: Commands to process

C_2^*: Commands that have been undone

IC_Edit Spec

$$Z_1 \vdash \text{reverse}(C_1^*) \Downarrow Z_2$$

Command ::= Insert | Delete | Overwrite | CursorCommands

Z_1: Initial content zipper

C_1^*: Active commands

Z_2: Final content zipper

Implementation

Which data structure to choose?

Array

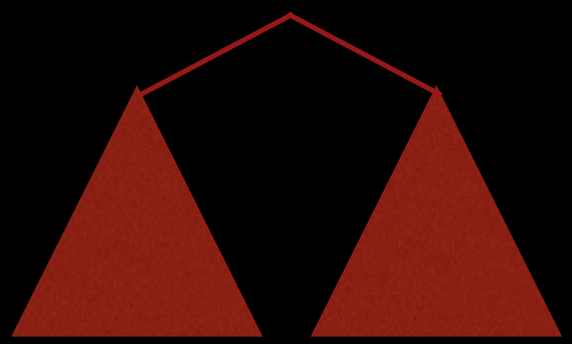
Easy: Random access

Hard: Insert, Search

Zipper

Easy: Insert, local move

Hard: Search

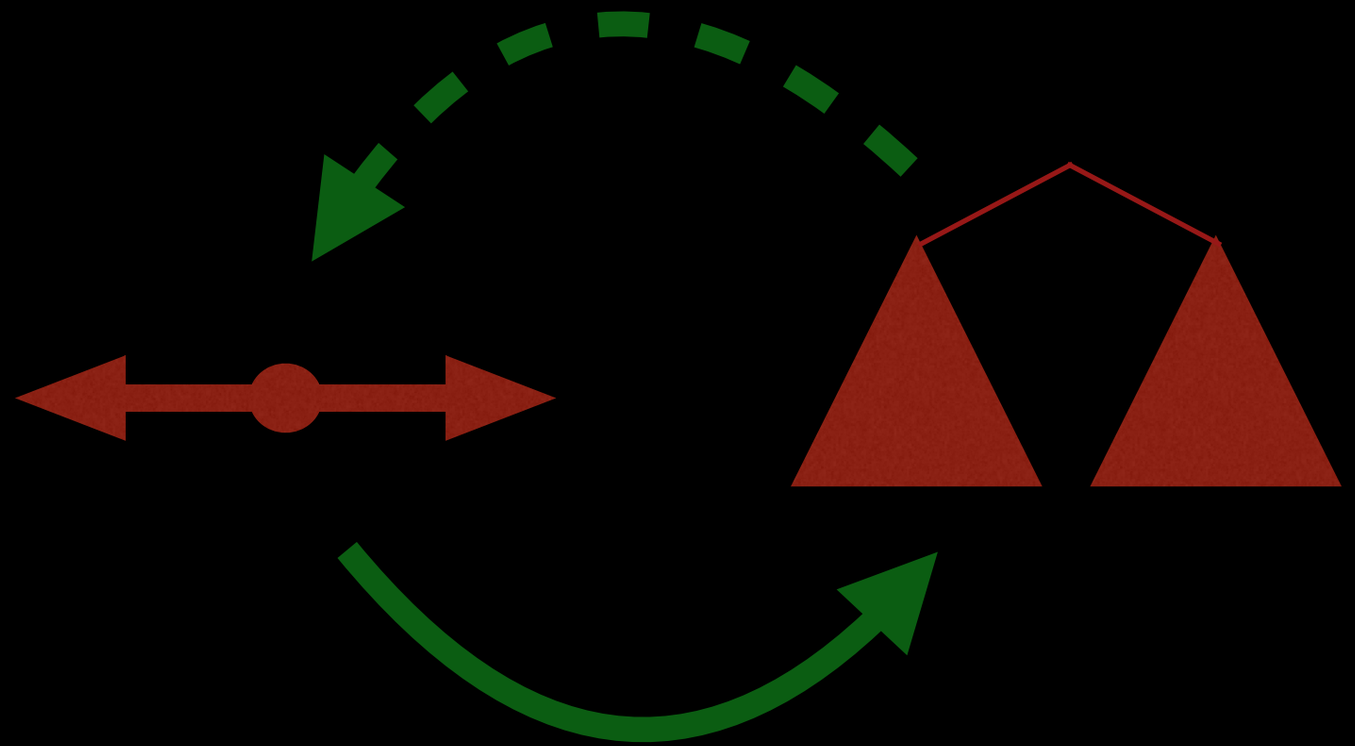


Tree

Easy: Search, global move

Hard: Insert (balancing issues)

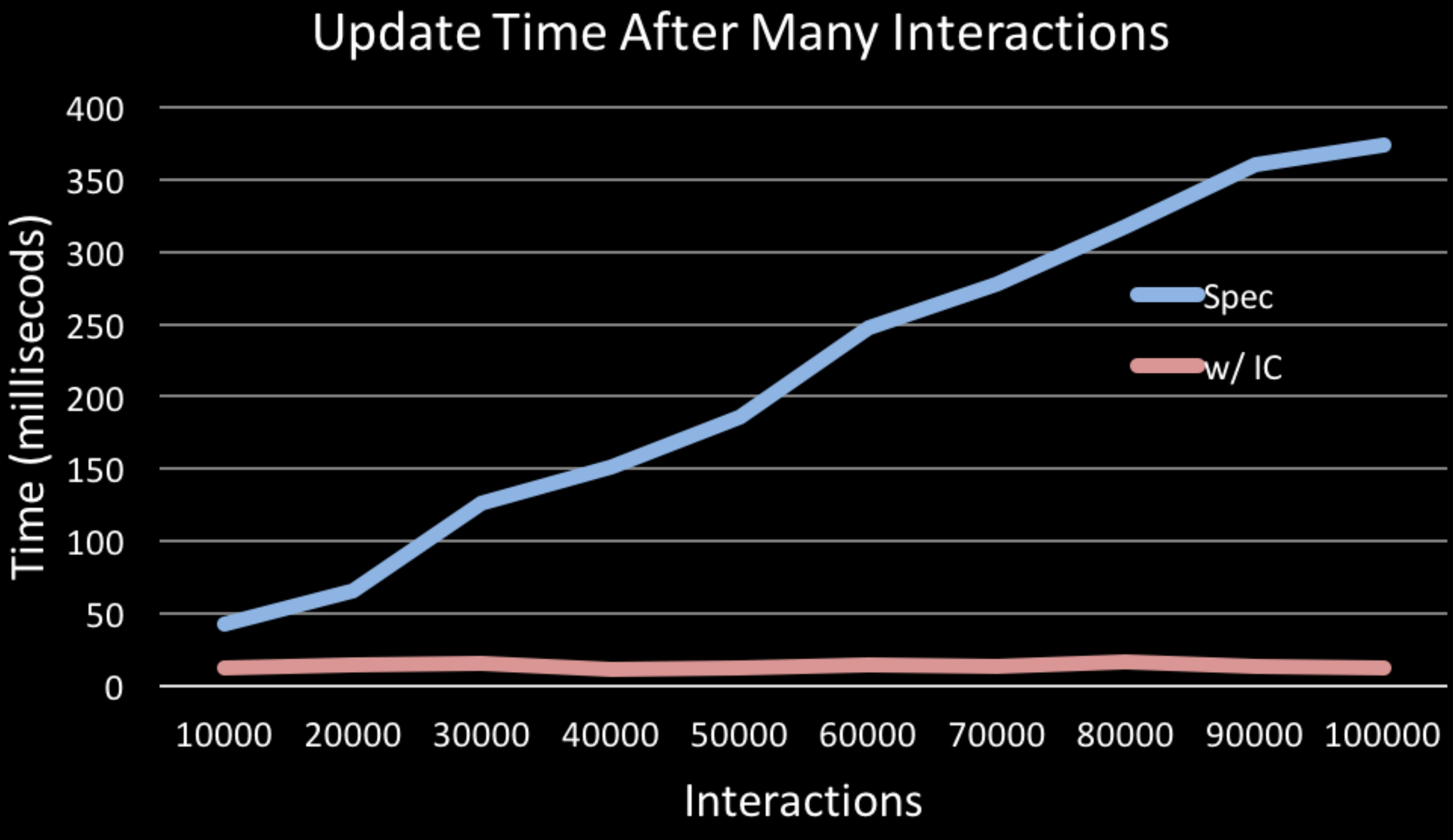
Key Insight



- Use data structure that is easiest for each operation
- Transform to other structures and keep them in cache
- Use incremental computation for optimizations

Implementation

Preliminary Results



IC_Edit Future

Optimise Searching

Reduce overhead

Implement additional features

- Search for words
- Better navigation

Correct-by-construction next steps

Spreadsheet

IDE

Improved Adapton interface

Better composition of techniques

Servo:
Web browser in rust

Take-aways

Use IC to cache useful data structures