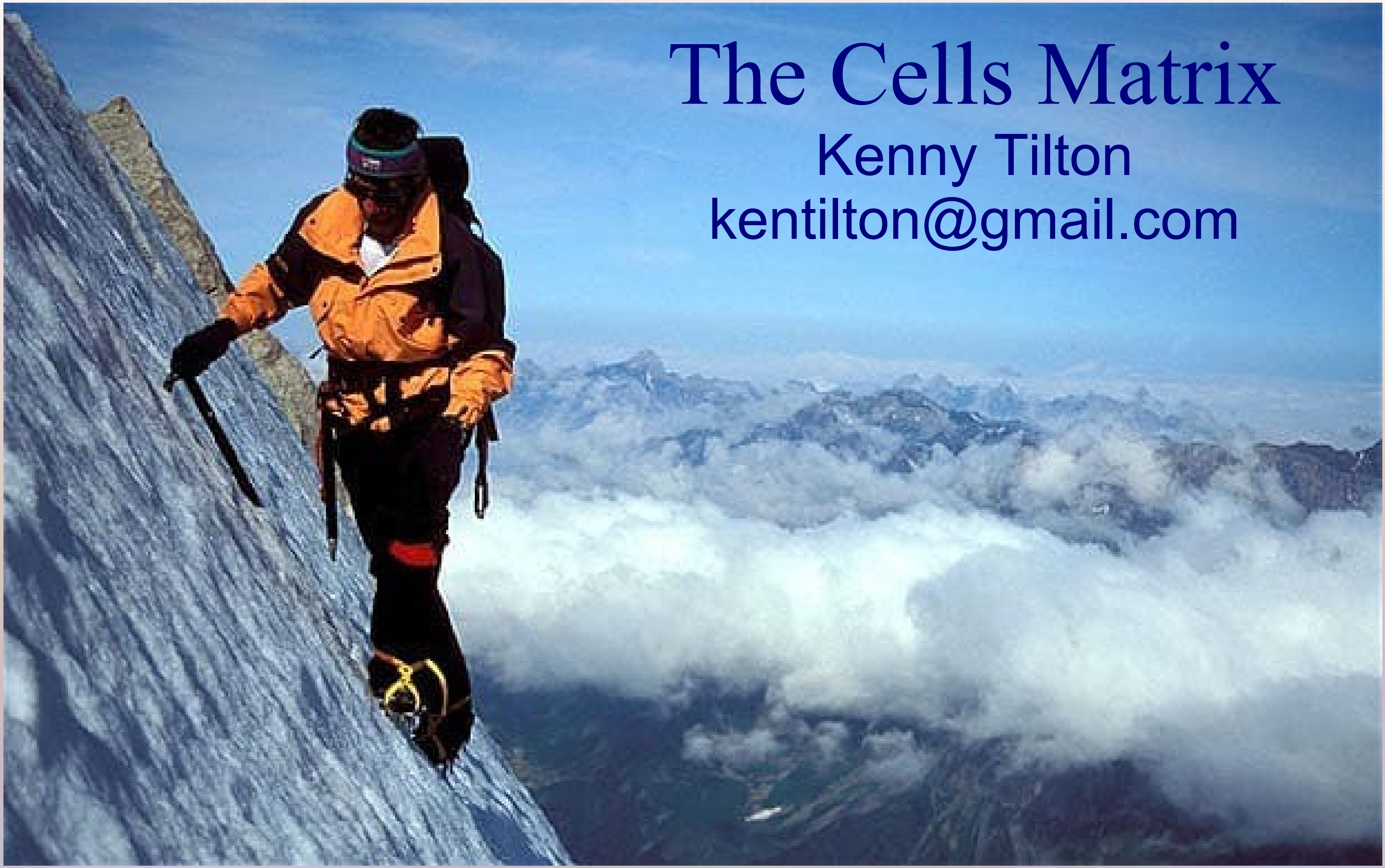


The Cells Matrix

Kenny Tilton
kentilton@gmail.com

These slides with notes:

https://github.com/kennytilton/cells

Something Is Going To Happen

The Wins

- GUI geometry made trivial
- Yes Silver Bullet
- The Grail of Object Reuse
- Callback Hell eliminated
- Transparent persistence
- Streams without ReactX
- Web frameworks eliminated

The Experience

- Declarative
- Transparent subscribe/notify
- Naturally Optimal Efficiency
- Easy to get right
- Easy to debug
- All automatically
- Fun

Data flow

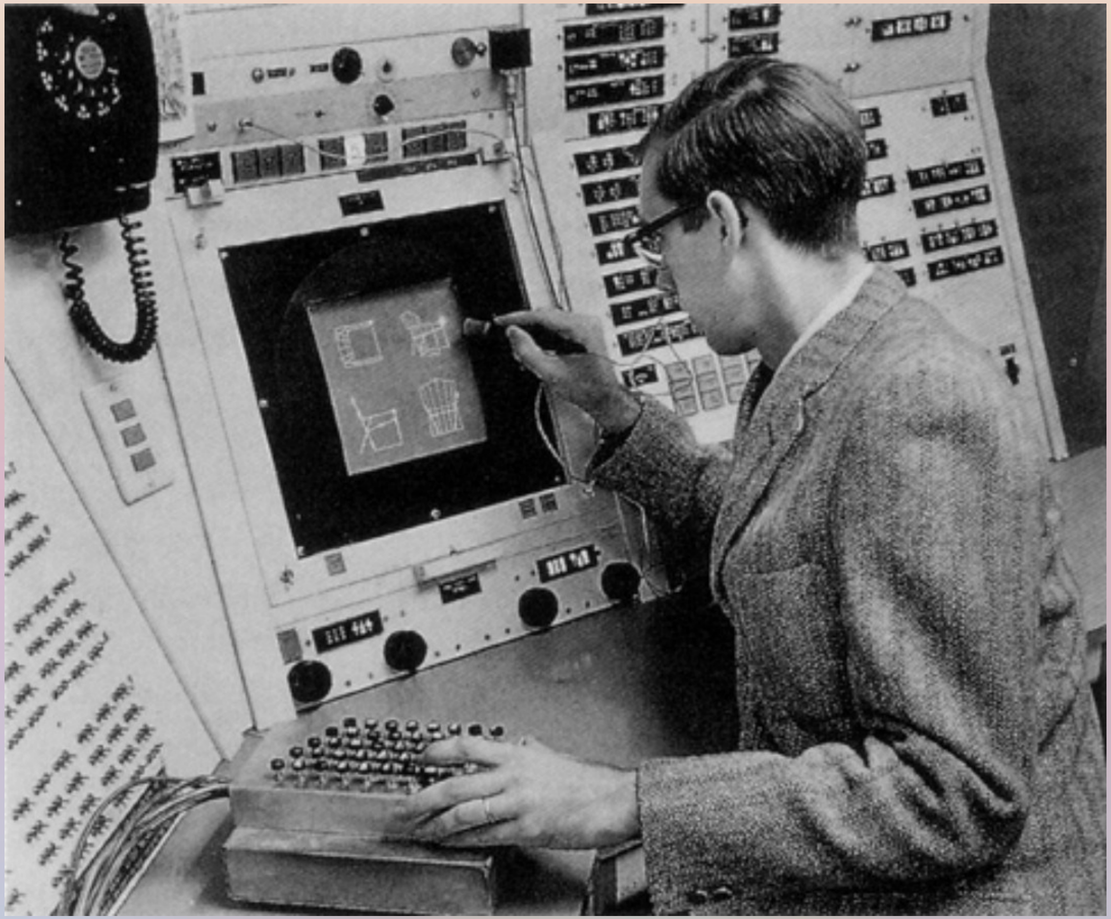
(*f* s*) ~> s

X ~> s

s ~> X

It is 1963. What is he doing?

Play Video



“Unforeseen uses are turning up.”

Sketchpad as Declarative

"The major feature which distinguishes a Sketchpad drawing from paper and pencil drawing is the user's ability to *specify* to Sketchpad mathematical *conditions* on already drawn parts of his drawing which will be *automatically satisfied* by the computer to make the drawing take the exact shape desired."

Ivan Sutheland
"Sketchpad: A Man-Machine Graphical Communication System"
Proceedings—Spring Joint Computer Conference, 1963

Constraints

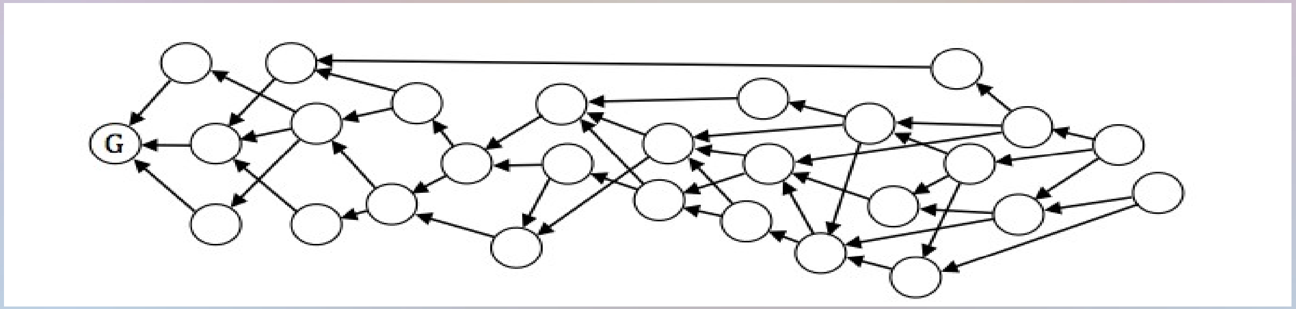
A bridge too far.

Multi-way and partial constraints

Constraint: A must equal B + C
Constraint: B = 2, C = 40.
What is A?

Now change C to 30.
Add constraints B < 10 and B is prime.
What is A?

Now solve this:



25 Years of Logic Programming

“One common problem is the sometimes unpredictable behavior of the constraint model: even small changes in a program can lead to a dramatic change in the performance. This is because the process of performance debugging, designing and improving constraint programs is currently not well understood. Related to this problem is the long learning curve that novices have experienced. While simple applications can be built almost immediately, it can take a long time to become familiar with the full power of a constraint system.”

Rossi, Francesca

In “Constraint (Logic) Programming: A Survey on Research and Applications”, 1997

Cells the Accident: Step #1

```
(make-instance 'text-edit
  :right (lambda (self)
           (+ (left self)
              (fontStringWidth (text self)
                (math-font *app*))))
```

- Functional/declarative: easier to write/debug
- Property to property: efficient, minimal change
- Instance specific: object re-use

Step #2: Transparency

Great, but how to we *read* “lr”? `(funcall (right self) self)`? What if “right” is 42 instead of a function? We would have to read the slot and see if it is a function. We need an accessor!

```
(defmethod right ((self geo-box))
  (let ((sv (slot-value self 'right)))
    (if (functionp sv)
      (funcall sv self)
      sv)))
```

Functional is slow. Step #3.

Cache the computation (so now we need state):

```
(defstruct cell rule value)

(defmethod right ((self geo-box))
   (b-if cell (slot-cell self 'right)
     (if (unboundp (value cell))
       (setf (value cell)
          (funcall rule self)))
       (value cell))
     (slot-value self 'right)))
```

Full sweep is still slow. Step #4a.

Track dependents so we can notify them:

```
(defstruct cell rule value users)

(defmethod right ((self geo-box))
  (b-if cell (slot-cell self 'right)
     (progn
        (when *user*
           (c-record-user cell *user*))
        (let ((*user* cell))
           (setf (value cell)
             (funcall rule cell))))
     (slot-value self 'right)))
```

Step #4b: Closing the Loop

```
(defmethod (setf right) (new-val (self geo-box))
  (b-if cell (slot-cell self 'right)
    (prog1 new-val
      (when (not (eql new-val (value cell)))
         (c-observe 'right self ;; s->X
                    new-val (value cell))
         (setf (value cell) new-val)
         (dolist (user (users cell))
            (c-recompute user))))
    (setf (slot-value self 'right) new-val))))
```

Transparent, specific, automatic state change.

Step #5: Tell the World

Once we recalculate everything, how do we manifest the new values? Observers.

```
(defobserver right ((self geo-box) newv oldv)
   (qdraw:invalidate-rect (rectangle self)))
```

Step #0: In the Beginning

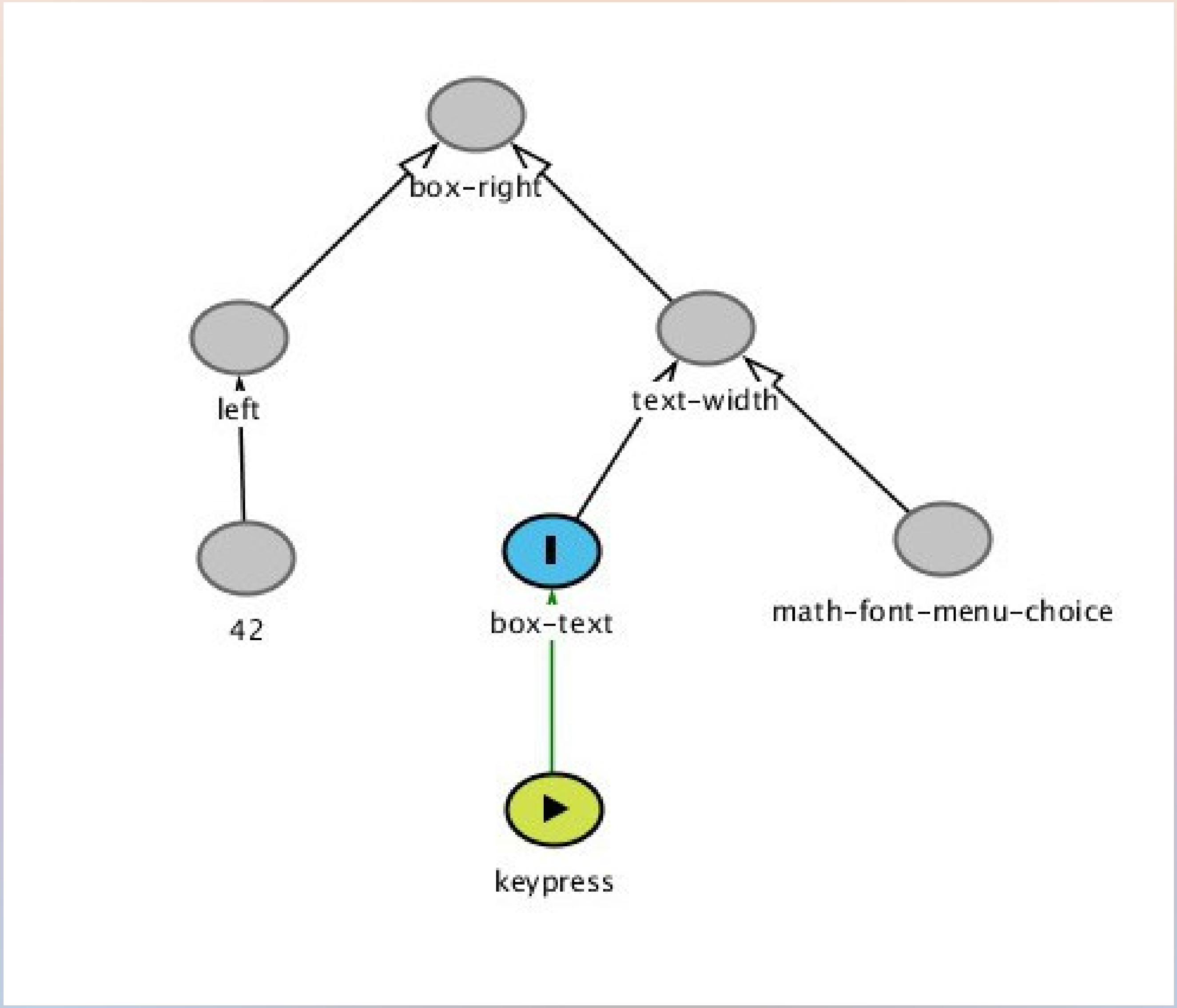
Where did the data *start*?

```
(defn todo-entry-field []
  (input {
      :class       "new-todo"
      :autofocus   true
      :placeholder "What needs doing?"
      :text (c-input “”)
      :oninput  (c? (fn [evt]
                    (setf (text self)
                      (form/getValue (.-target evt))))))}))
```

The Call Stack Inverted

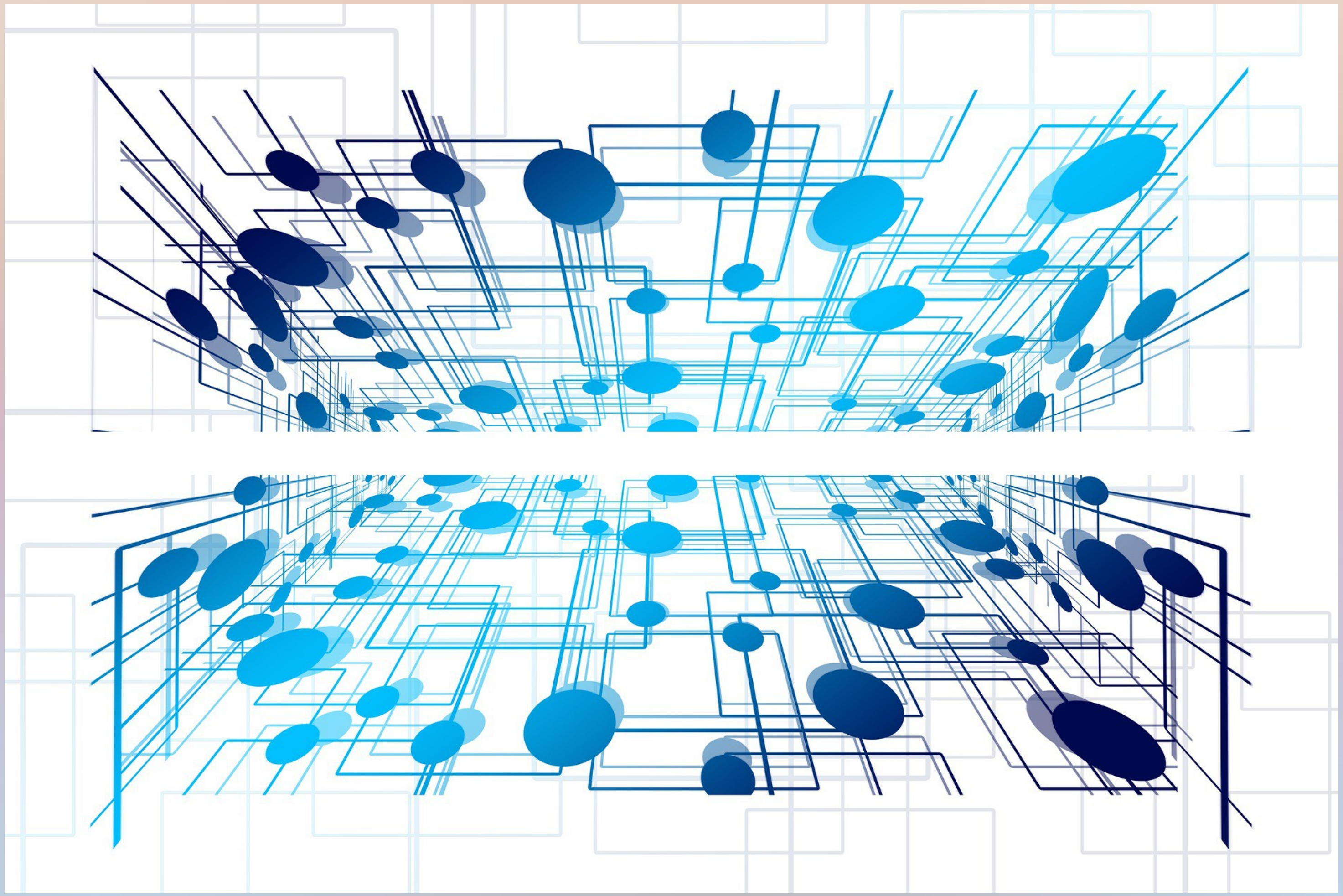
Inversion Goggles.

Hello, DAG.


box-right
left
text-width
42
box-text
math-font-menu-choice
keypress

Hello, Matrix

ma·trix ˈmātriks noun an environment in which something else takes form. Origin: Latin, female animal used for breeding, parent plant, from matr-, mater



TodoMX in the Matrix

Look for:
- Data flow
- Efficiency
- Transparency
- Web components
- Object re-use
- Functional state
- Debuggability
- No web framework

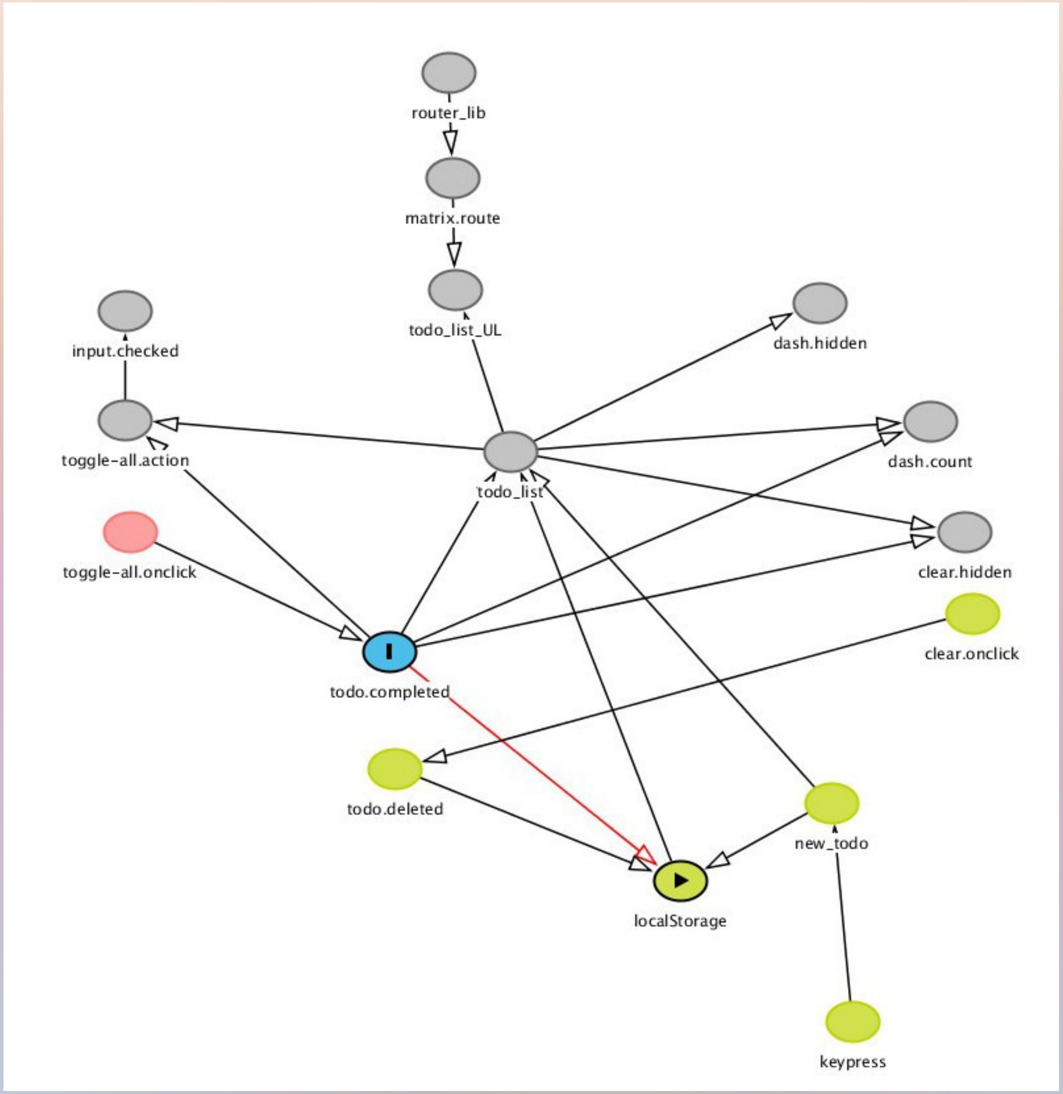
Demo

Five Changes

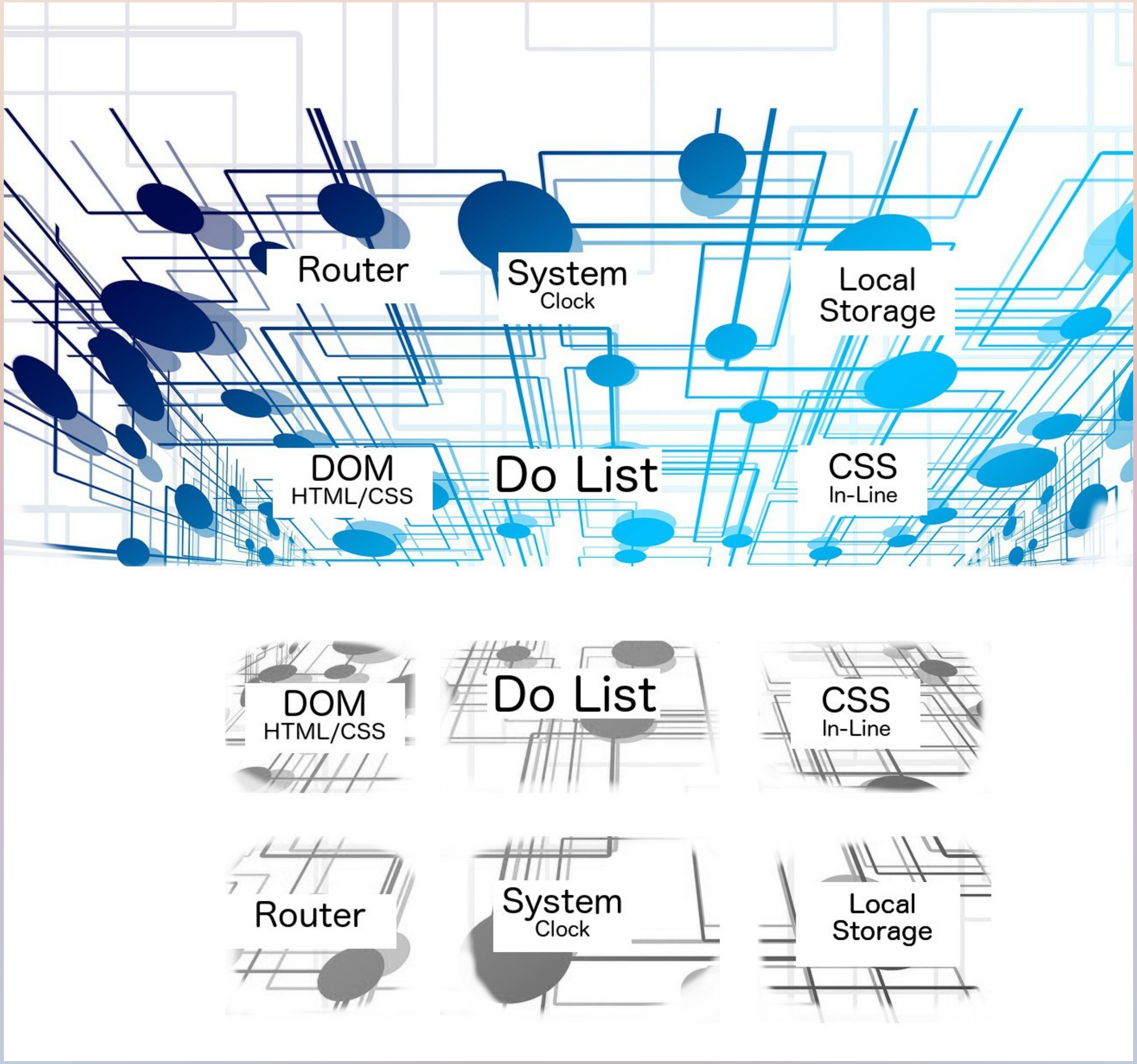
```
(td-toggle-completed! todo)
```

Pop quiz: How?

TodoMX in the Matrix

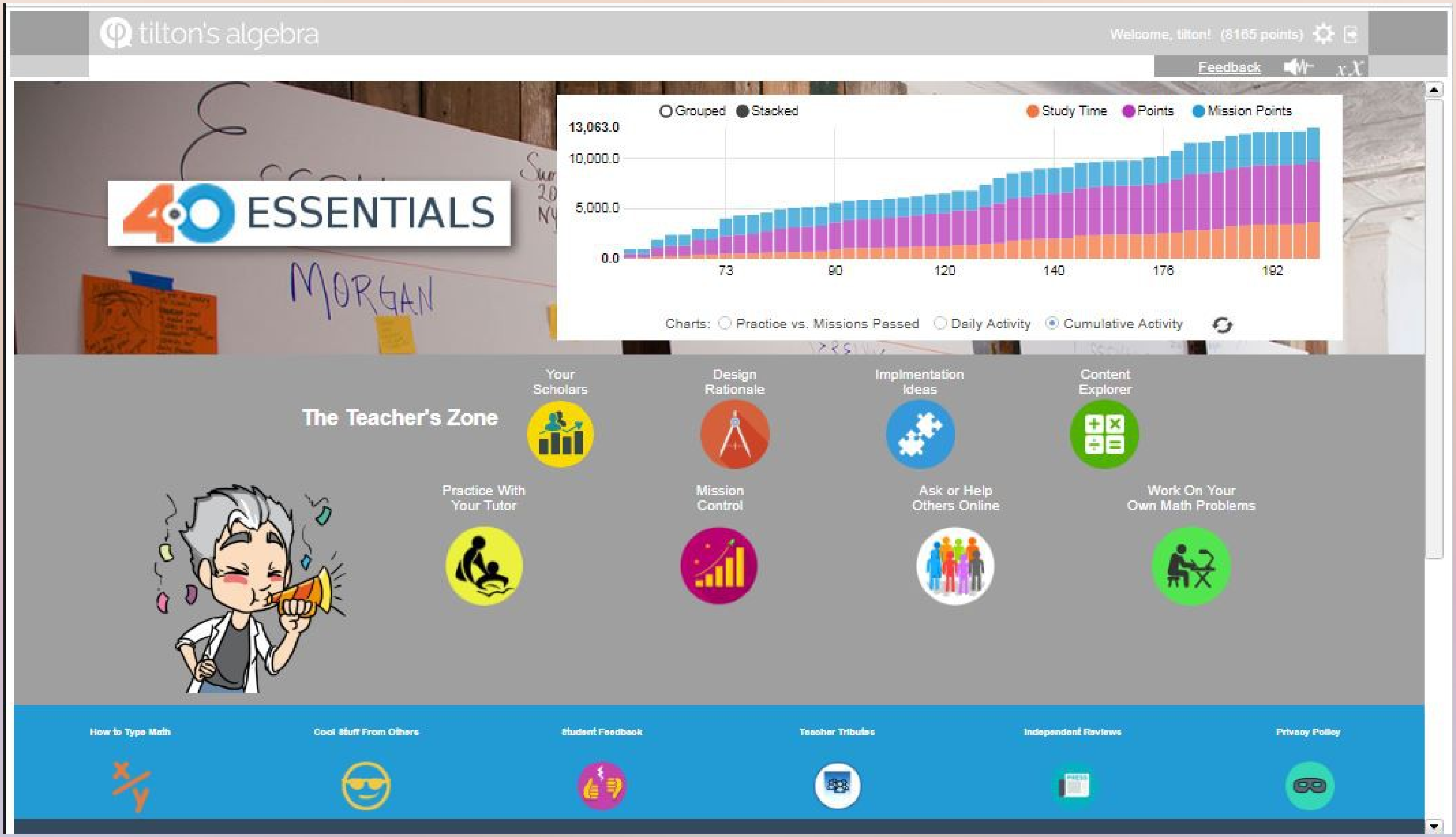


The TodoMX Matrix



This Is Not Simple

tiltonsalgebra.com

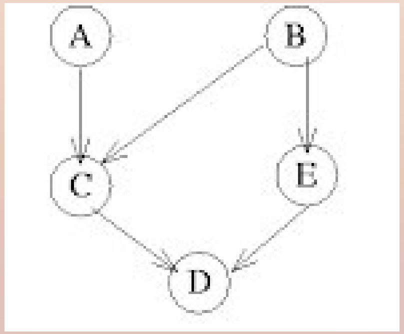


And CliniSys(tm), a Clinical Drug Trial Manager.
80kloc of Common Lisp/qooxdoo.

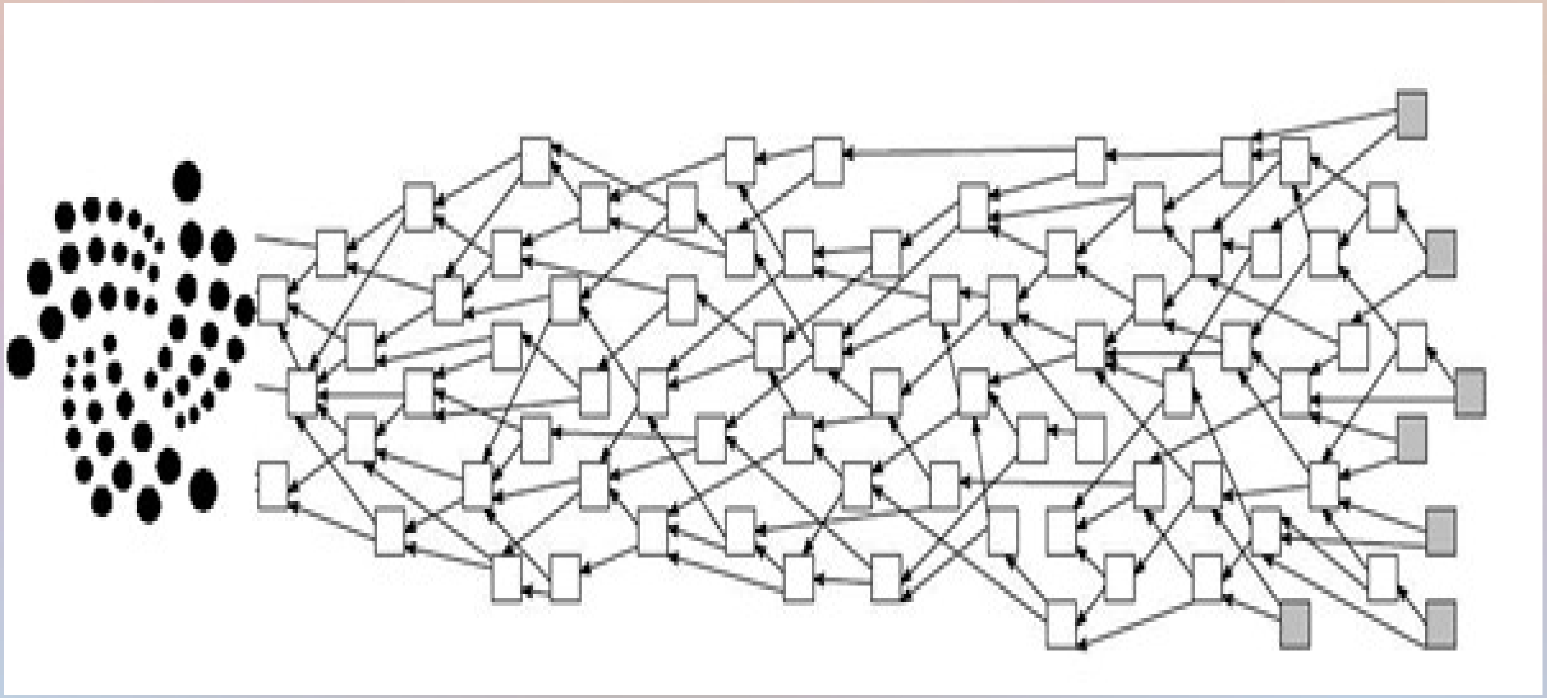
Fred Brooks Was Right.

About the Problem.

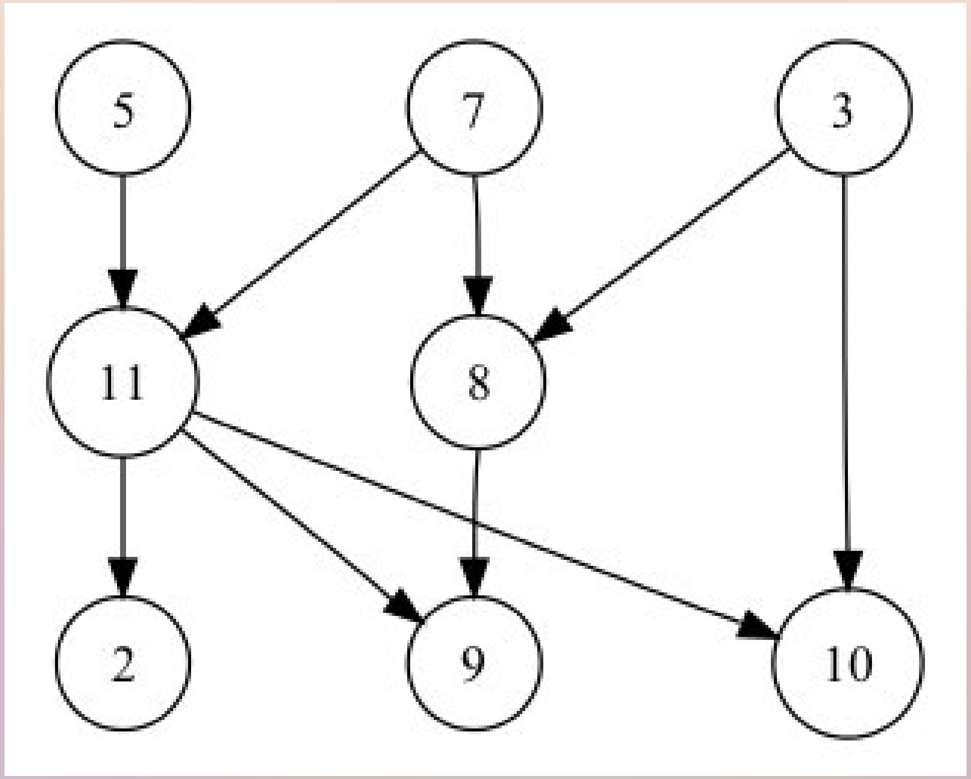
Not Bad.


$$\begin{tikzcd}
A \arrow[d] & & B \arrow[dll] \arrow[d] \\
C \arrow[dr] & & E \arrow[dl] \\
& D &
\end{tikzcd}$$

Good Luck.



But Sean Parent Saw This...



Pop quiz: Notice anything?

“...a large class of applications.”

It is not just for user interfaces.

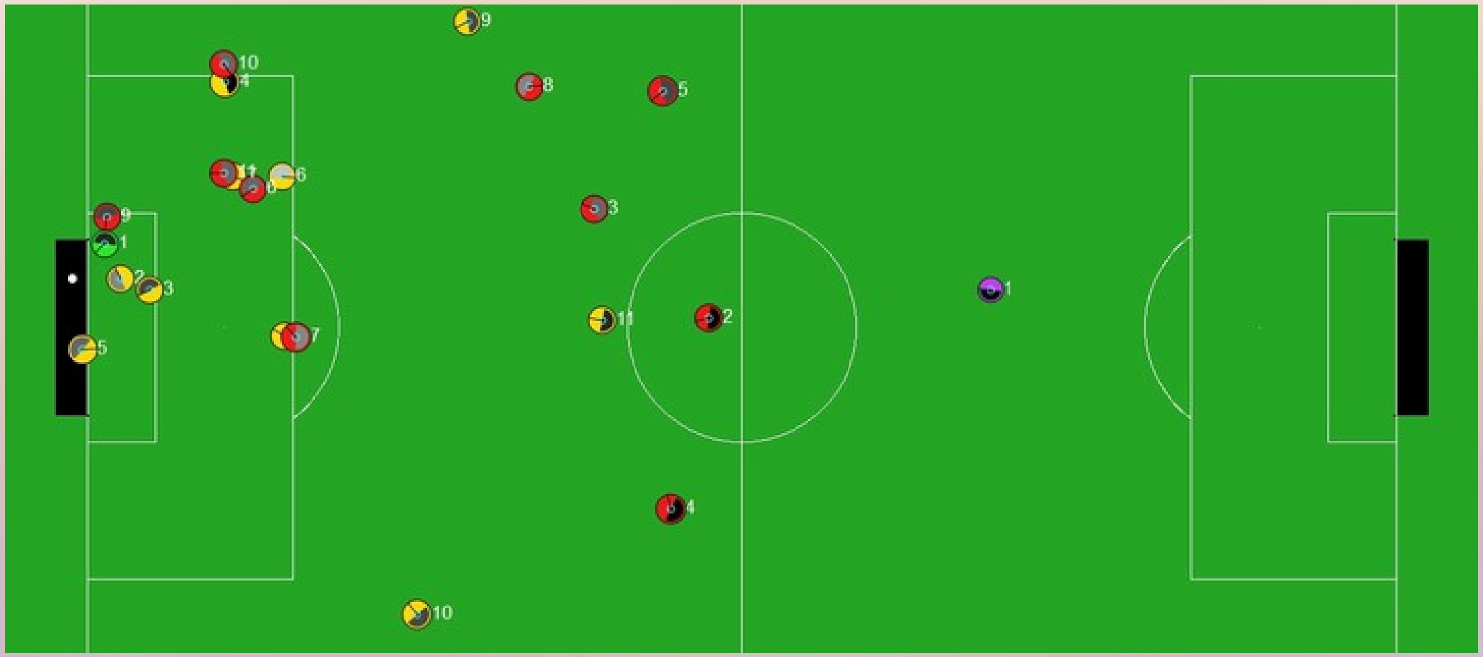
“It is useful for any application involving an interesting amount of long-lived state and a stream of unpredictable inputs.”

http://smuglispweeny.blogspot.com/2008/02/cells-manifesto.html

- User interfaces: Yes
- Real-time process control: Yes
- ETL: Not so much
- RoboCup Simulation League: Yes

Glitches and RoboCup Sim

The Cells are dead! Long live Cells!



One (UDP) input: a visual sensory dump every 100ms.

The Glitch

$$\begin{tikzcd}
X & \\
& C \arrow[ul, dashed] \\
& B \arrow[u] \\
A \arrow[uuu] \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[rr, "\text{Known}"] & & {} \\
{} \arrow[rr, dashed, "\text{Conditional}"] & & {}
\end{tikzcd}$$

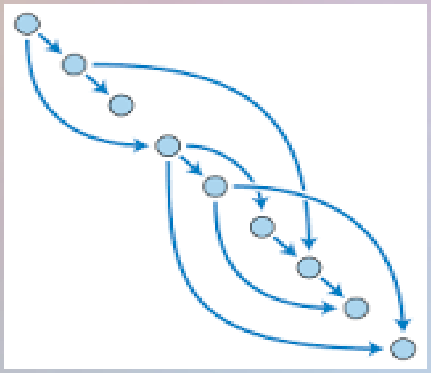
Puzzle: How ensure C is computed before X when A changes?

Anti-glitch schemes

- Topological sort
- Mark and sweep (MobX)
- “Dirty” counting (MobX originally)
- Integer pulse/clock (Cells)

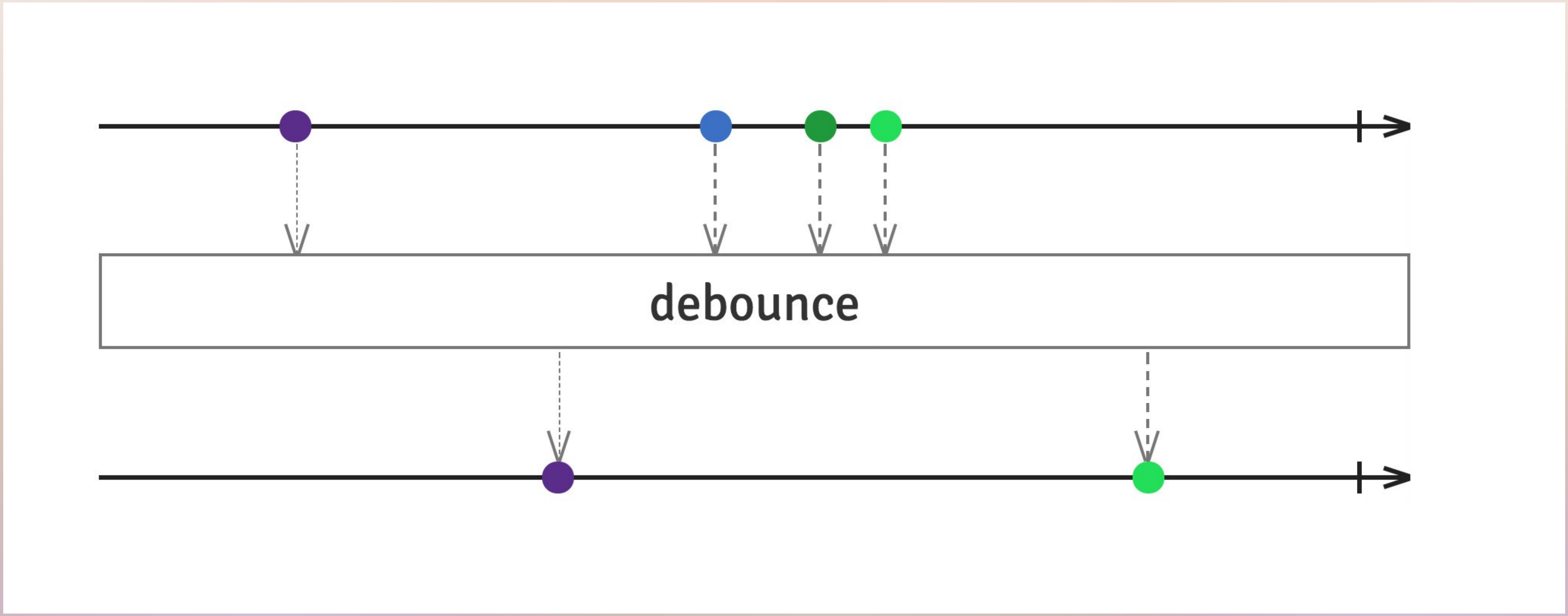
Puzzle: Why This?

Why did RoboCup Sim fail on glitches when a 79kloc clinical drug trial system did not?

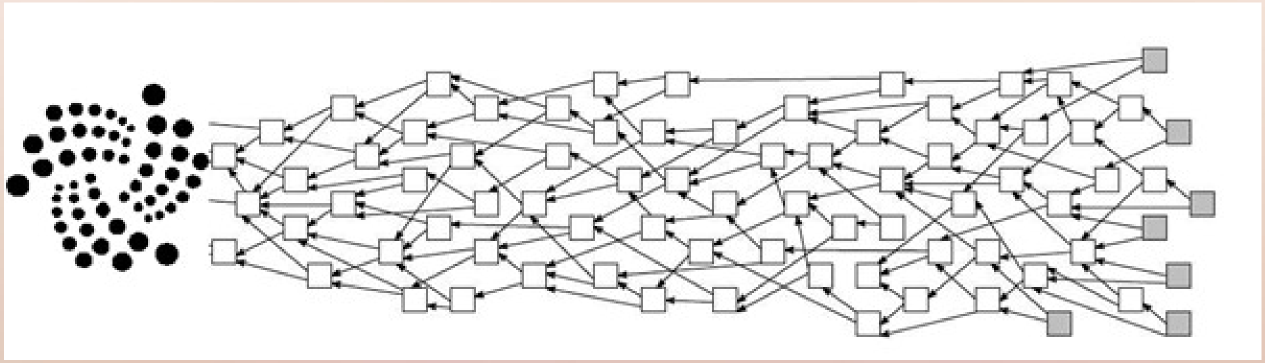


ReactiveX/RxJS

Streamthink, or Data flow is not the only game in town.



Synapses as Streams



syn·apse [sin-aps, si-naps] Physiology noun - A junction between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter.

```
(let [sensor (cI nil)
      alarm (cF (if (with-synapse (:delta-sensor [prior (atom nil)])
                      (when-let [reading (<cget sensor)]
                       (let [delta (Math/abs
                                     (if @prior
                                         (- reading @prior)
                                         0))]
                         (when (> delta 5)
                            (reset! prior reading)
                            delta))))
                  :on :off)))]
     ...etc...)
```

The Reactive Field

I look for declarative, transparent authoring, point-to-point dependency, automatically executed glitch-free. Avoid "lifting" RX.

- Common Lisp Garnet’s KR; cool but quiescent.
- Philip Eby’s Python Cells clone, Trellis;
- Adobe’s C++ Adam;
- Lifting dataflow: CLJS Javelin;
- Michael Weststrate’s Javascript MobX
- CL Screamer (constraints)
- FrTIme/FlapJax: also lifting

“Lifting” Data Flow

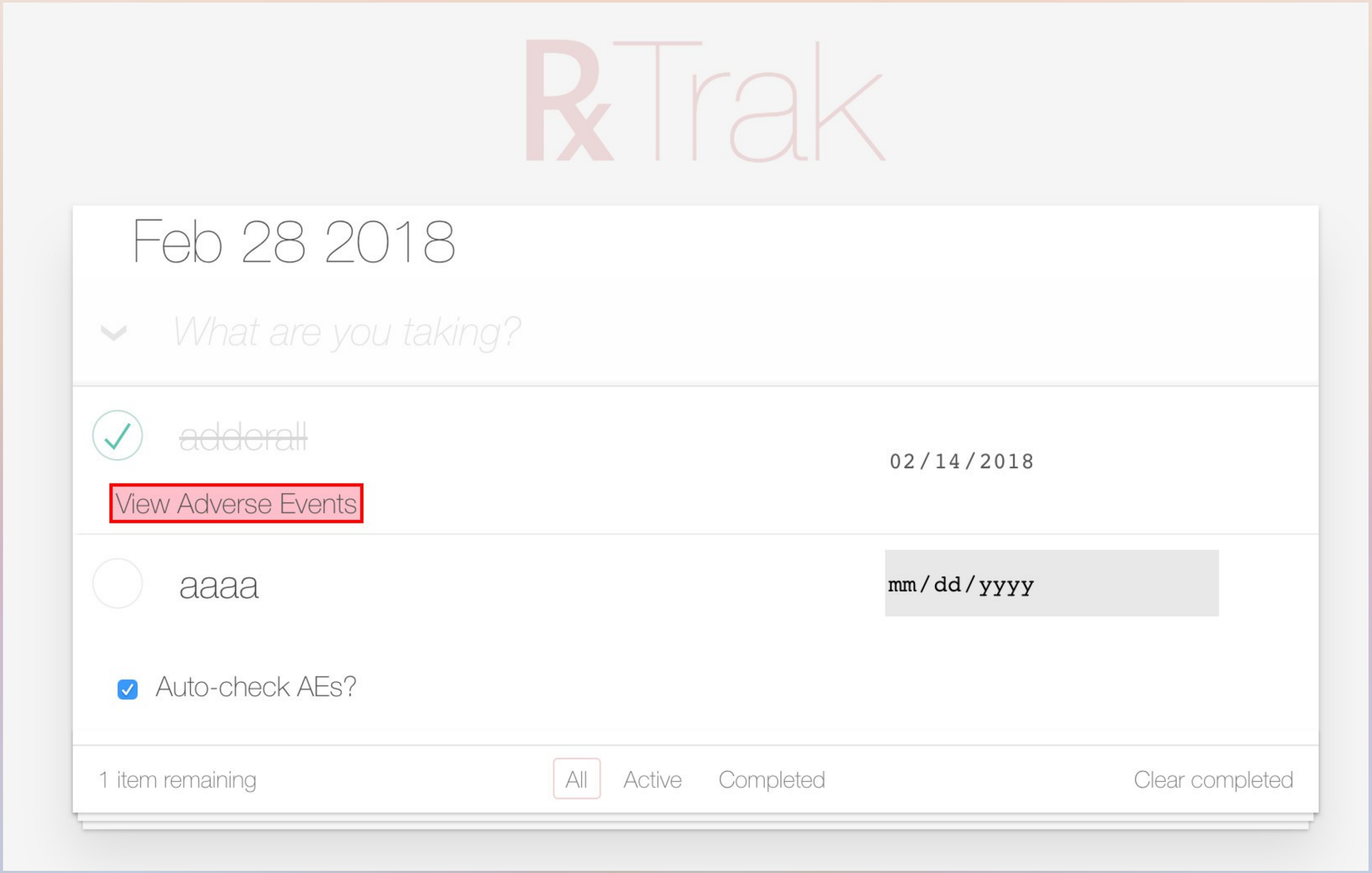
X := (if A B C) where all vars are Cells.

Pop quiz: On how many Cells does X depend?

```
(defn get-Z [] Z)
X := (+ Y (get-Z))
```

Trick quiz: On how many Cells does X depend?

RxTrak


RxTrak

Feb 28 2018

What are you taking?

~~adderall~~ 02/14/2018

View Adverse Events

aaaa mm/dd/yyyy

☑ Auto-check AEs?

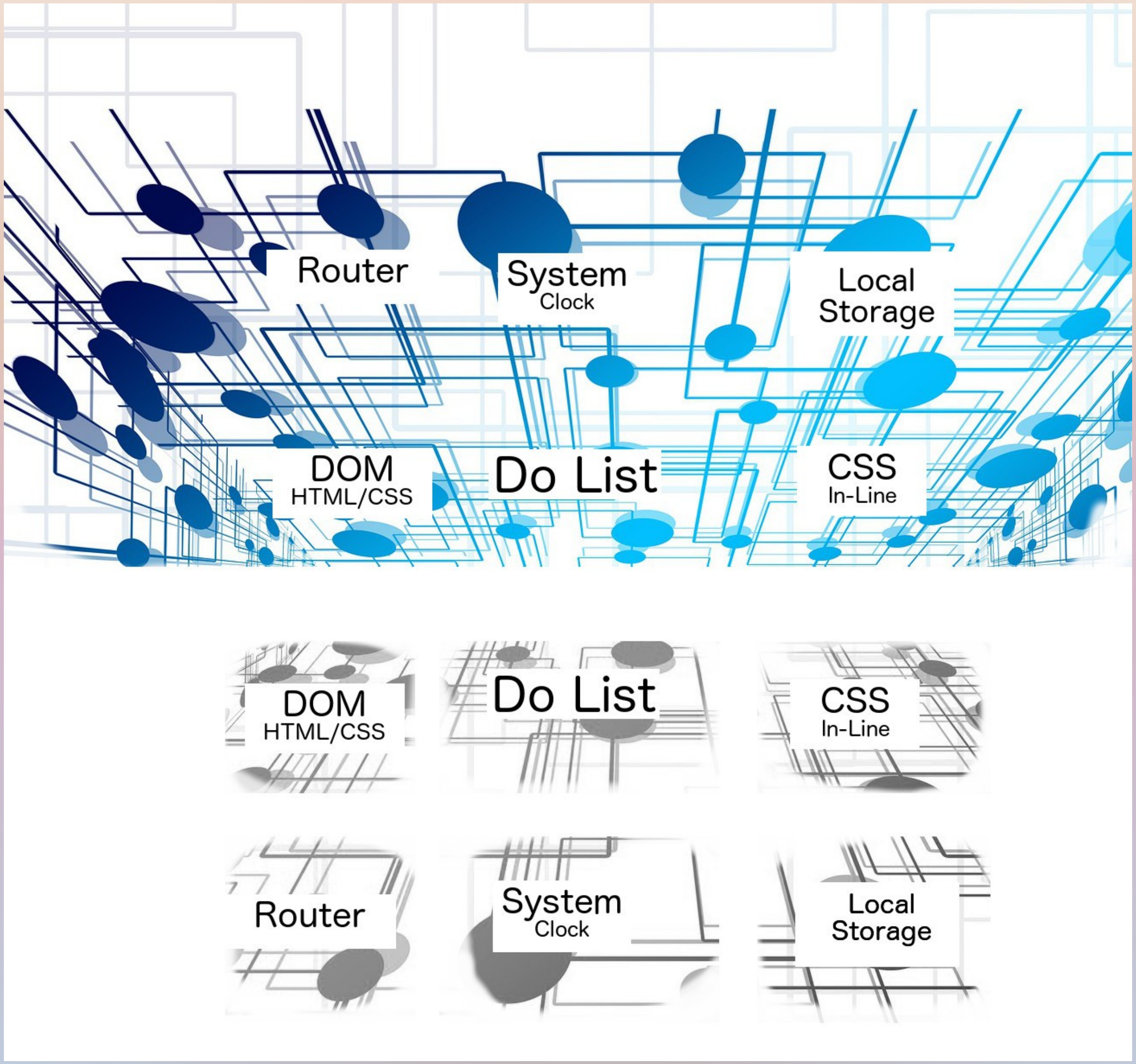
1 item remaining | All | Active | Completed | Clear completed

Callback Hell Meets Data Flow

```
(defn xhr-send [xhr]
  (go
    (let [response (<! (client/get (<mget xhr :uri)
                                   {:with-credentials? false}))]
      (with-cc :xhr-handler-sets-responded
        (mset!> xhr :response
          {:status (:status response)
           :body   (if (:success response)
                     ((:body-parser @xhr)
                      (:body response))
                     [(:error-code response)
                      (:error-text response)])})))))
```

Look for mset!>

The TodoMX Matrix



The Benefits

GUI geometry made trivial
Yes Silver Bullet
The Grail of Object Reuse
Efficiency for free
Callback Hell eliminated
Web frameworks obvisted
ReactX/RxJS for free
Reliability
Debugability
Fun

Reactive Programming

s <~ (*f* s*)

s <= x

*side-effects** <~ s

Puzzle: What elemental quality of nature does the above capture making it so powerful?

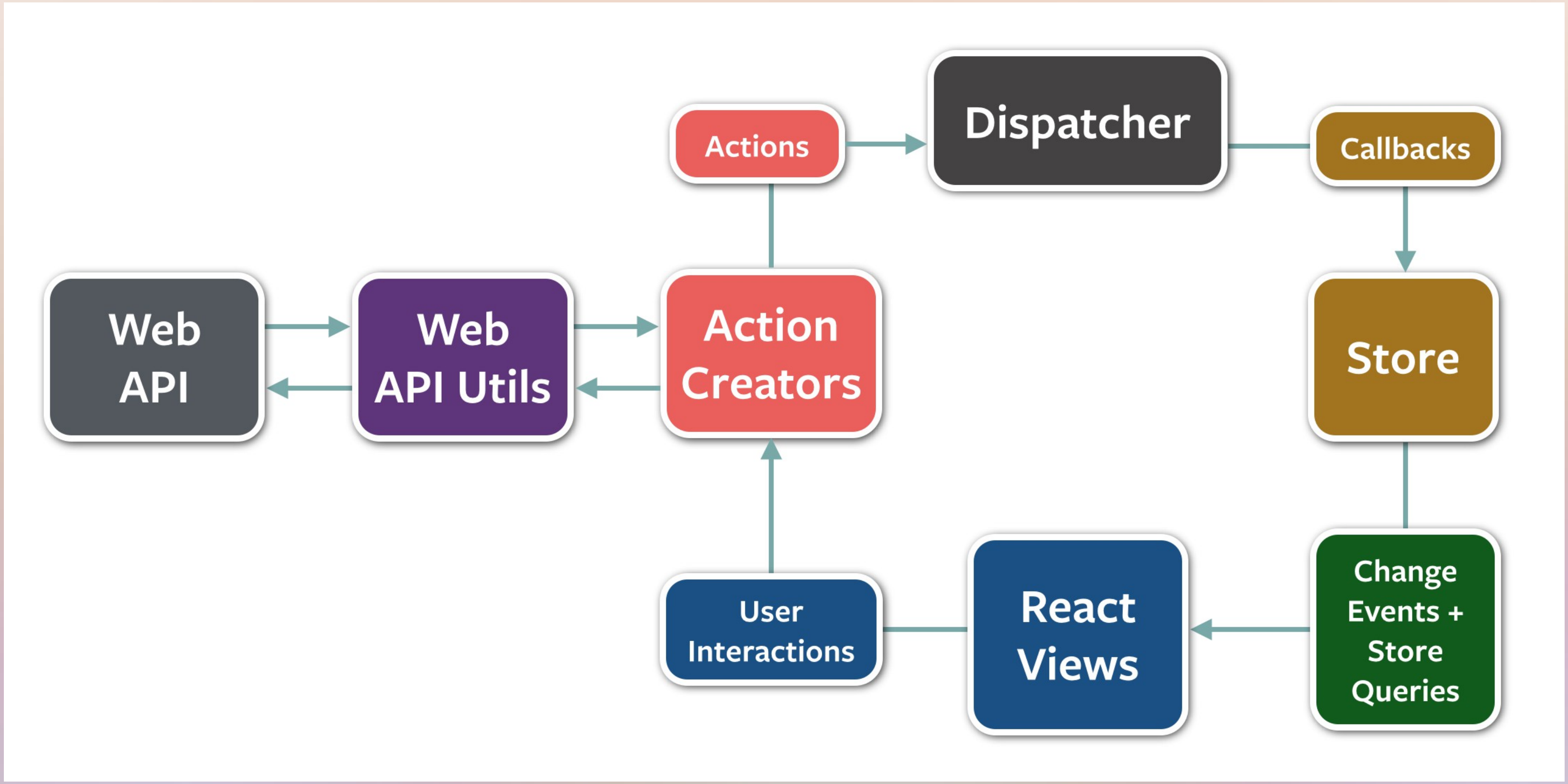
- Causation
- Communication
- Animation

Something Is Going To Happen

ReactJS Evangelized Declarative

```
class TodoList extends React.Component {
  render() {
    return (
      <ul>
        {this.props.items.map(items => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
```

Gross Reactivity: Elm/Flux/Redux



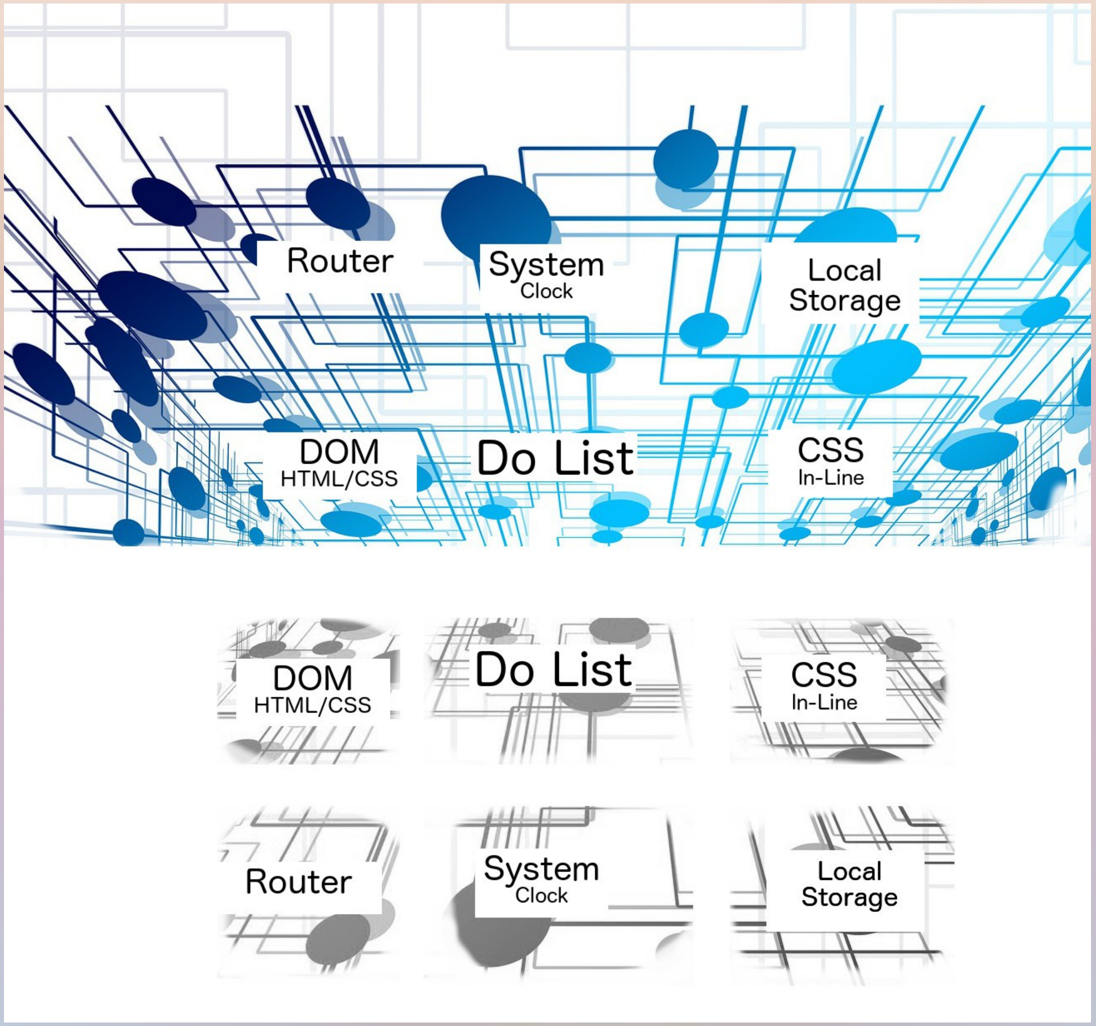
Ouch.

MobX Evangelizing Point Data Flow

- Binding.scala
- Clojurescript Javelin
- Scheme FrTime and FlapJax
- Python Trellis
- Common Lisp Cells
- CLJ/S Matrix

They Will Find It

The last piece of the puzzle: Matrix



Appendix

More slides with more information on reactive programming follow.

Common Lisp

Clojure

Evaluating Reactive Libraries

- Is it a “lifting” library? Problem.
- Is pub-sub explicit? Awkward.
- Is it really Reactive? ReactJS is not, but it is declarative which is nice.
- Does it have “glitches”? Not the end of the world, but others are glitch-free so why not?

FrTime, FlapJax

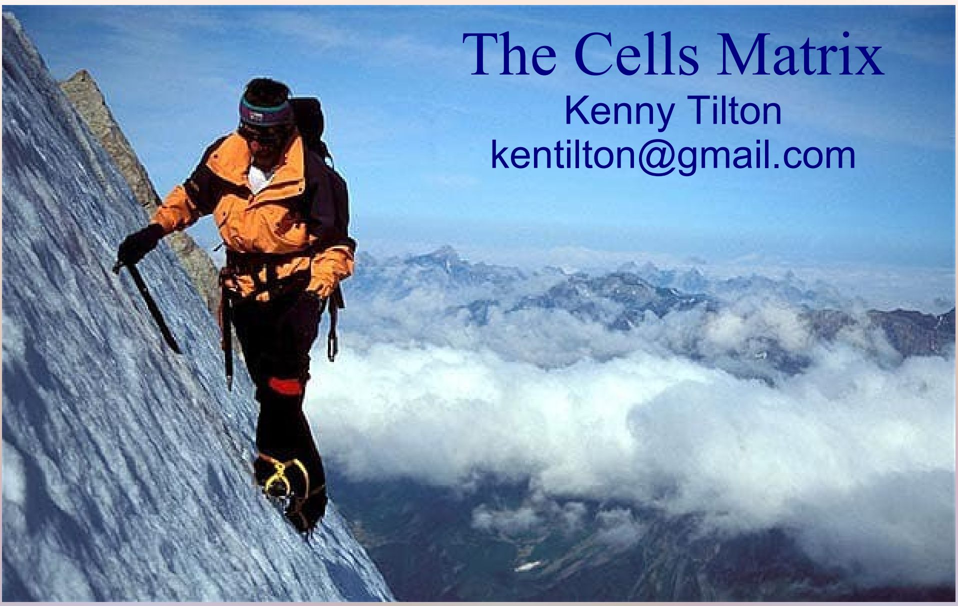
- FrTime: Scheme FRP: http://docs.racket-lang.org/frtime/
- FlapJax: FRP Web framework, a Javascript translation of FrTime: http://www.flapjax-lang.org
- Javelin (ClojureScript)
- “lifting” implementation has efficiency and coding issues: (if A B C) and (+ 40 (compute-with box)),

Trellis for Pythonistas

- Phillip Eby's development by Cells
- http://peak.telecommunity.com/DevCenter/Trellis
- https://pypi.python.org/pypi/Trellis/0.7a1
- David Gelerntner's “Trellis” from “Mirror Worlds”: https://global.oup.com/academic/product/mirror-worlds-9780195079067?cc=us&lang=en&

Functional Reactive Programming (FRP)

- Conan Eliot Lambda-Jam 2015 https://youtu.be/j3Q32brCUAI
- Continuous time, yes
- Graphs and dataflow, no.



The Cells Matrix

Kenny Tilton
kentilton@gmail.com

These slides with notes:

https://github.com/kennytilton/cells

Erhard Loretan. Died pair climbing when his partner fell and pulled him to his death. She survived.

Something Is Going To Happen

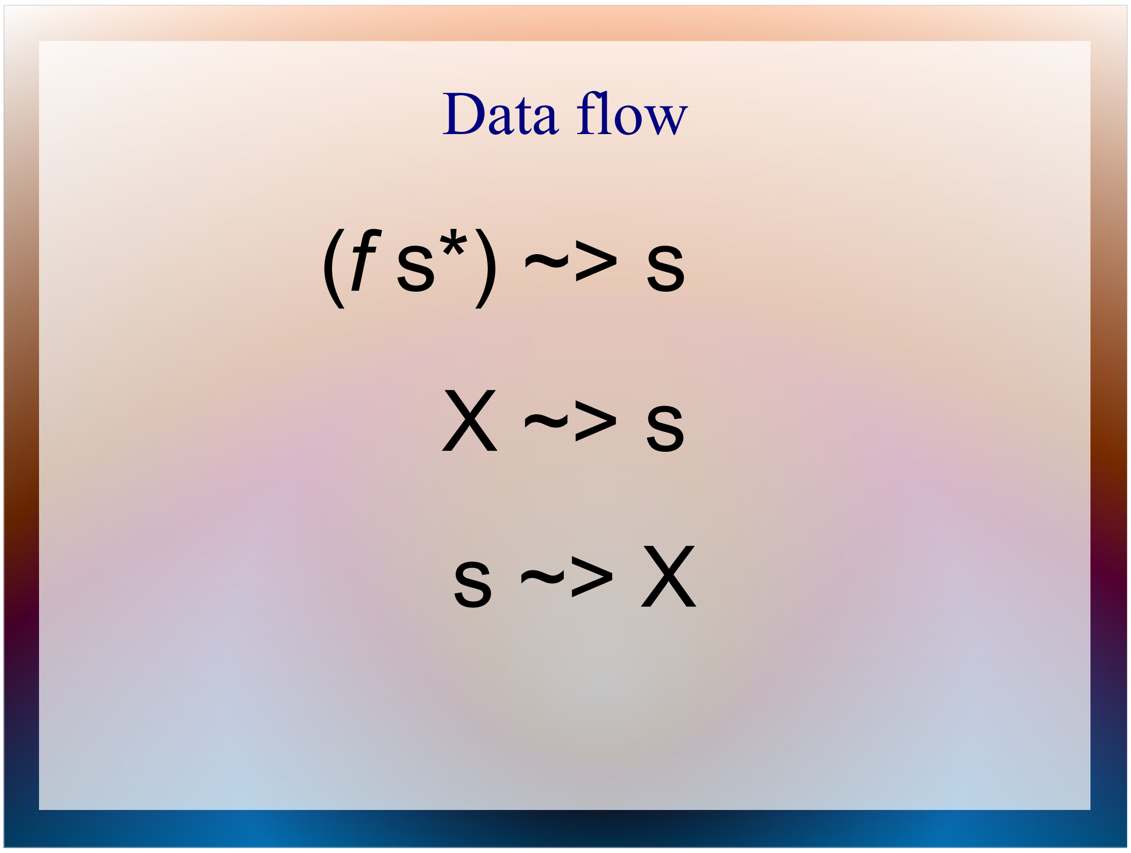
Great scene from 2010, sequel to 2001 Space Odyssey: https://youtu.be/V5HA-umweZ0

The Wins

- GUI geometry made trivial
- Yes Silver Bullet
- The Grail of Object Reuse
- Callback Hell eliminated
- Transparent persistence
- Streams without ReactX
- Web frameworks eliminated

The Experience

- Declarative
- Transparent subscribe/notify
- Naturally Optimal Efficiency
- Easy to get right
- Easy to debug
- All automatically
- Fun

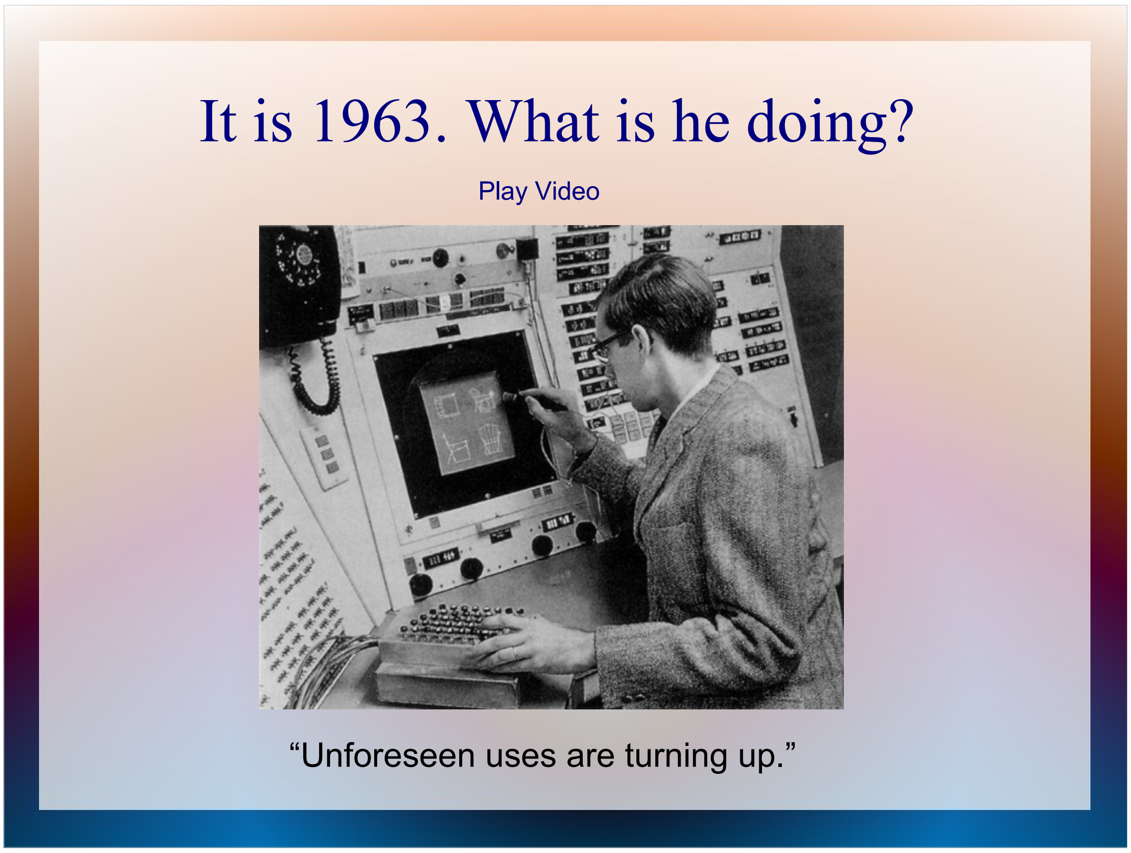


Data flow

$$(f\ s^{*}) \mathrel{\sim>} s$$

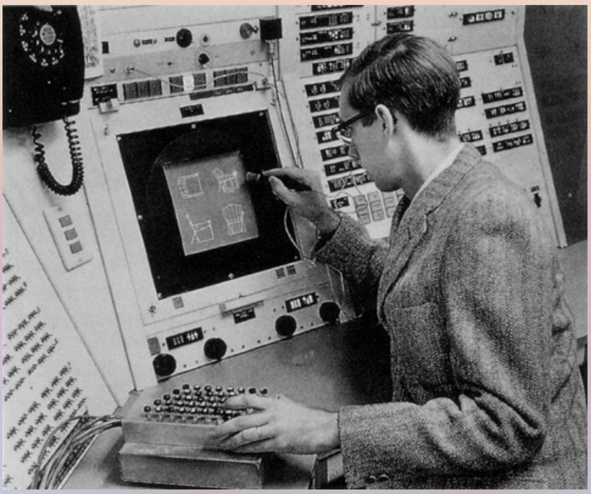
$$X \mathrel{\sim>} s$$

$$s \mathrel{\sim>} X$$



It is 1963. What is he doing?

Play Video



“Unforeseen uses are turning up.”

Ivan's objective was simply to use a computer to help a mechanical engineer with a design drawing.

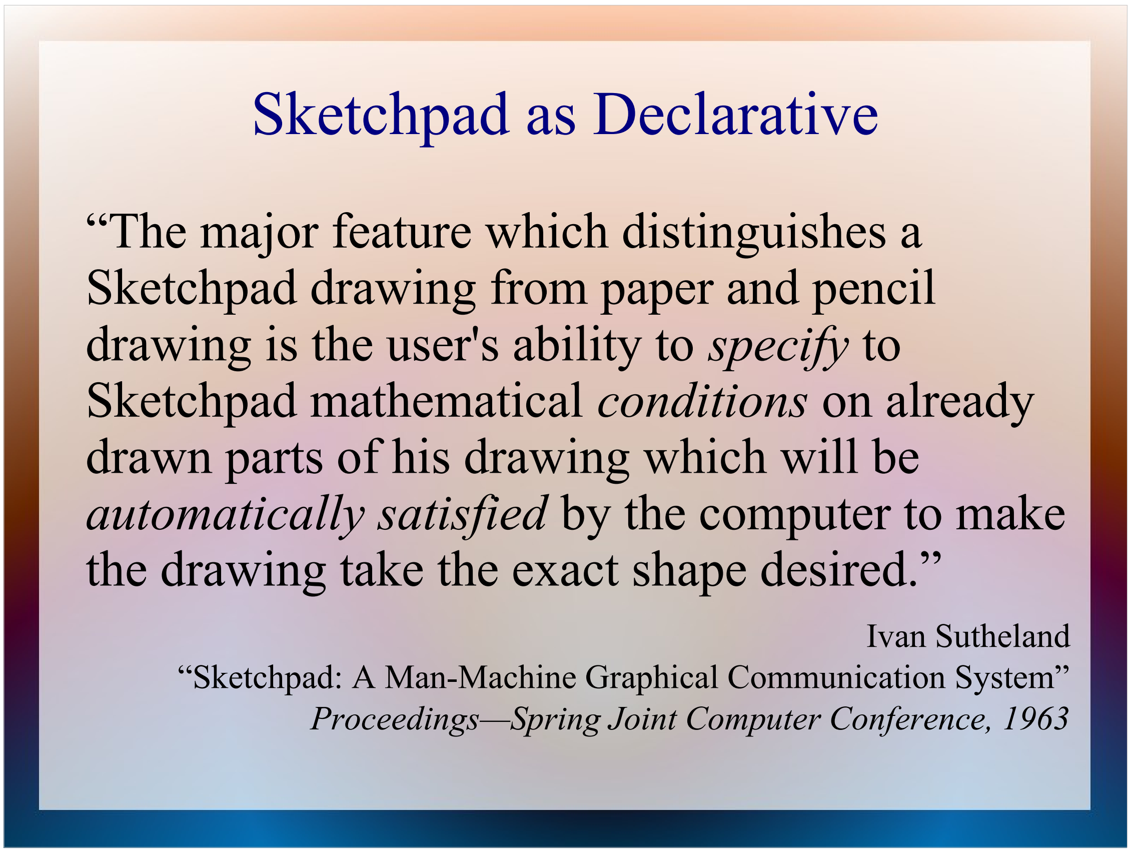
Some software was so hard to use that engineers were more effective using paper. To give SketchPad an edge over paper, Ivan let the user declare constraints such as “these segments are horizontal” or “I am drawing an arc”. A constraint solver translated sloppy light pen gestures into precise coordinates to honor the declarations. Fun tech note: constraints were declared by throwing toggle switches.

- Age 15 wrote biggest app for SIMON: 8 feet of paper tape
- 1963: Sketchpad, an app as a PhD thesis.
- First “mouse” (a light pen)
- Constraint solver
- Turing Award

Thesis: Re-released/-formatted 2003 https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-574.pdf

Turing award: : https://amturing.acm.org/award_winners/sutherland_3467412.cfm

- Demo: https://youtu.be/57wj8diYpgY
- TV show: https://youtu.be/USyoT_Ha_bA



Sketchpad as Declarative

"The major feature which distinguishes a Sketchpad drawing from paper and pencil drawing is the user's ability to *specify* to Sketchpad mathematical *conditions* on already drawn parts of his drawing which will be *automatically satisfied* by the computer to make the drawing take the exact shape desired."

Ivan Sutheland
"Sketchpad: A Man-Machine Graphical Communication System"
Proceedings—Spring Joint Computer Conference, 1963

Constraints

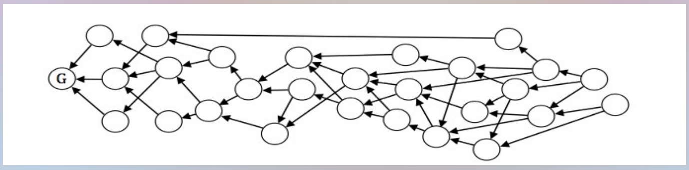
A bridge too far.

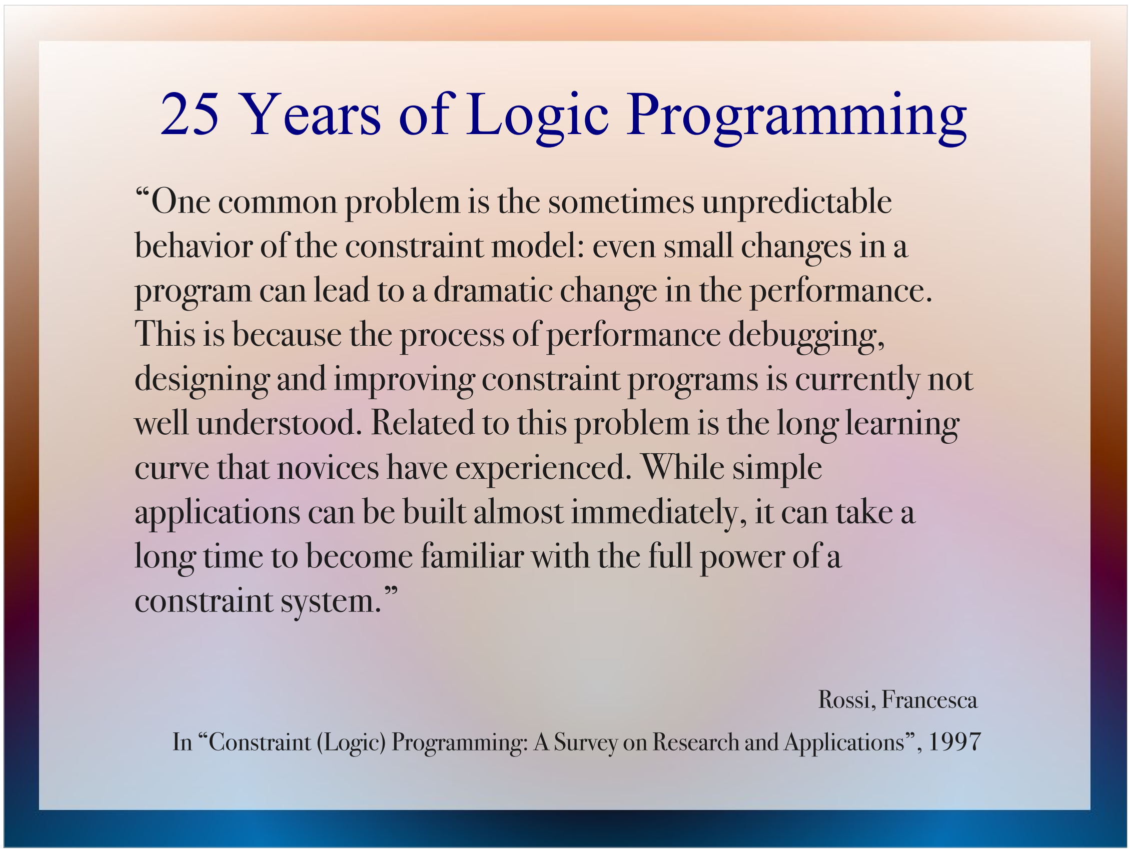
Multi-way and partial constraints

Constraint: A must equal B + C
Constraint: B = 2, C = 40.
What is A?

Now change C to 30.
Add constraints B < 10 and B is prime.
What is A?

Now solve this:





25 Years of Logic Programming

"One common problem is the sometimes unpredictable behavior of the constraint model: even small changes in a program can lead to a dramatic change in the performance. This is because the process of performance debugging, designing and improving constraint programs is currently not well understood. Related to this problem is the long learning curve that novices have experienced. While simple applications can be built almost immediately, it can take a long time to become familiar with the full power of a constraint system."

Rossi, Francesca

In "Constraint (Logic) Programming: A Survey on Research and Applications", 1997

Rossi's quote appeared in this retrospective
https://books.google.com/books?id=9jhtCQAAQBAJ

Cells the Accident: Step #1

```
(make-instance 'text-edit

  :right (lambda (self)
           (+ (left self)
              (fontStringWidth (text self)
                (math-font *app*)))))
```

- Functional/declarative: easier to write/debug
- Property to property: efficient, minimal change
- Instance specific: object re-use

Step #2: Transparency

Great, but how to we *read* “lr”? `(funcall (right self) self)`? What if “right” is 42 instead of a function? We would have to read the slot and see if it is a function. We need an accessor!

```
(defmethod right ((self geo-box))
   (let ((sv (slot-value self 'right)))
      (if (functionp sv)
         (funcall sv self)
         sv)))
```

Functional is slow. Step #3.

Cache the computation (so now we need state):

```
(defstruct cell rule value)

(defmethod right ((self geo-box))
   (b-if cell (slot-cell self 'right)
     (if (unboundp (value cell))
       (setf (value cell)
          (funcall rule self)))
       (value cell))
     (slot-value self 'right)))
```

Full sweep is still slow. Step #4a.

Track dependents so we can notify them:

```
(defstruct cell rule value users)

(defmethod right ((self geo-box))
  (b-if cell (slot-cell self 'right)
      (progn
         (when *user*
            (c-record-user cell *user*))
         (let ((*user* cell))
            (setf (value cell)
              (funcall rule cell))))
      (slot-value self 'right)))
```

Step #4b: Closing the Loop

```
(defmethod (setf right) (new-val (self geo-box))
  (b-if cell (slot-cell self 'right)
    (prog1 new-val
      (when (not (eql new-val (value cell)))
         (c-observe 'right self ;; s->X
                    new-val (value cell))
         (setf (value cell) new-val)
         (dolist (user (users cell))
            (c-recompute user))))
    (setf (slot-value self 'right) new-val))))
```

Transparent, specific, automatic state change.

Step #5: Tell the World

Once we recalculate everything, how do we manifest the new values? Observers.

```
(defobserver right ((self geo-box) newv oldv)
   (qdraw:invalidate-rect (rectangle self)))
```

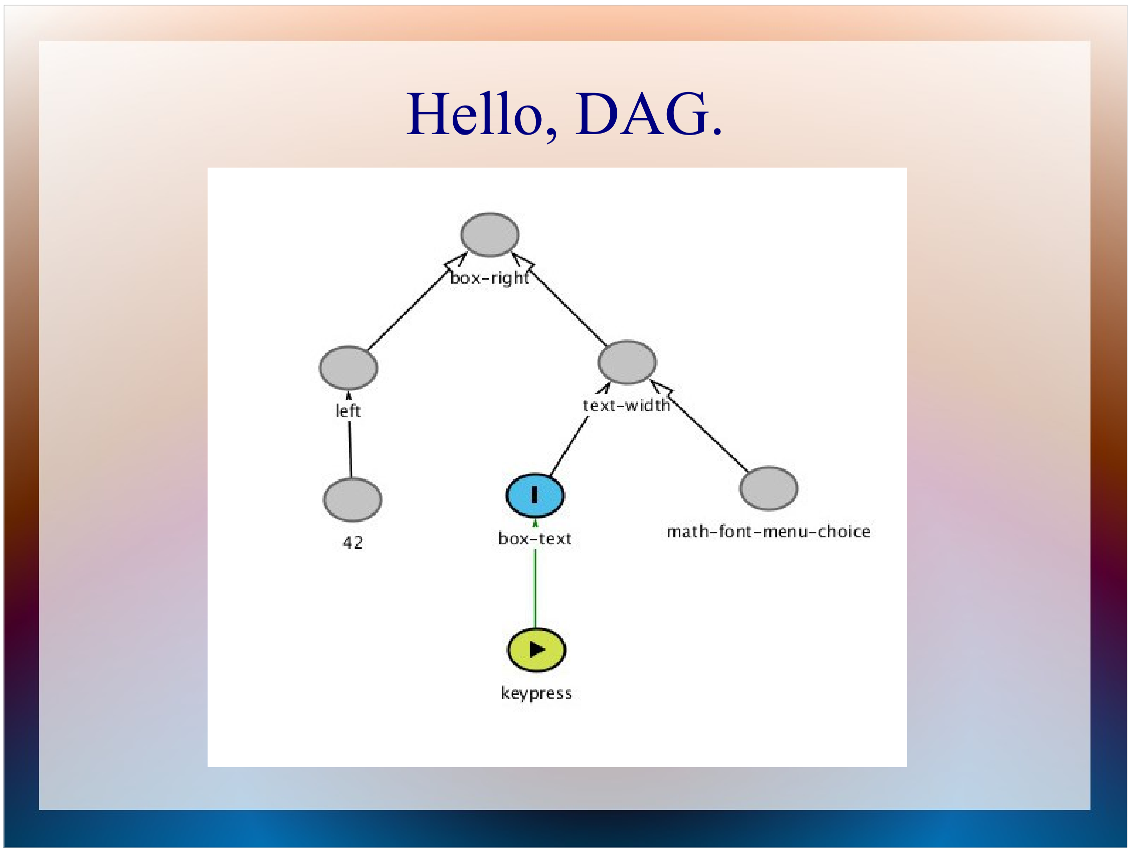
Step #0: In the Beginning

Where did the data *start*?

```
(defn todo-entry-field []
  (input {
      :class       "new-todo"
      :autofocus   true
      :placeholder "What needs doing?"
      :text (c-input “”)
      :oninput  (c? (fn [evt]
                       (setf (text self)
                         (form/getValue (.-target evt)))))}))
```

The Call Stack Inverted

Inversion Goggles.



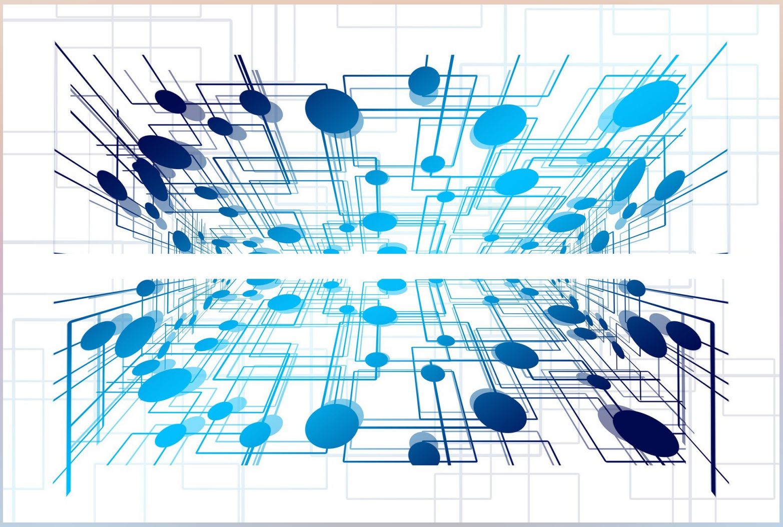
Hello, DAG.

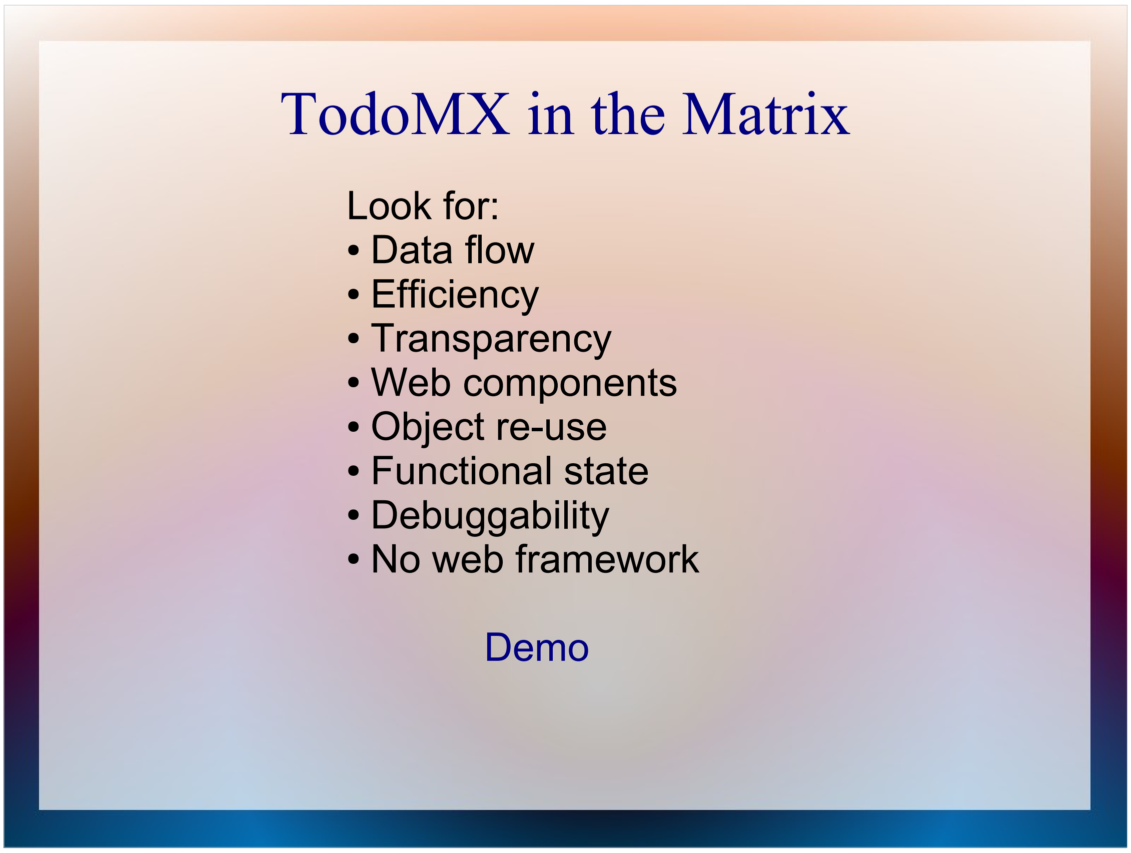
42 1 @0.341,0.267
box-right 1 @0.437,0.008
box-text O @0.468,0.263
keypress E @0.469,0.414
left 1 @0.338,0.138
math-font-menu-choice 1 @0.632,0.256
text-width 1 @0.533,0.133

42 left
box-text text-width
keypress box-text
left box-right
math-font-menu-choice text-width
text-width box-right

Hello, Matrix

ma·trix ˈmātriks noun an environment in which something else takes form. Origin: Latin, female animal used for breeding, parent plant, from matr-, mater





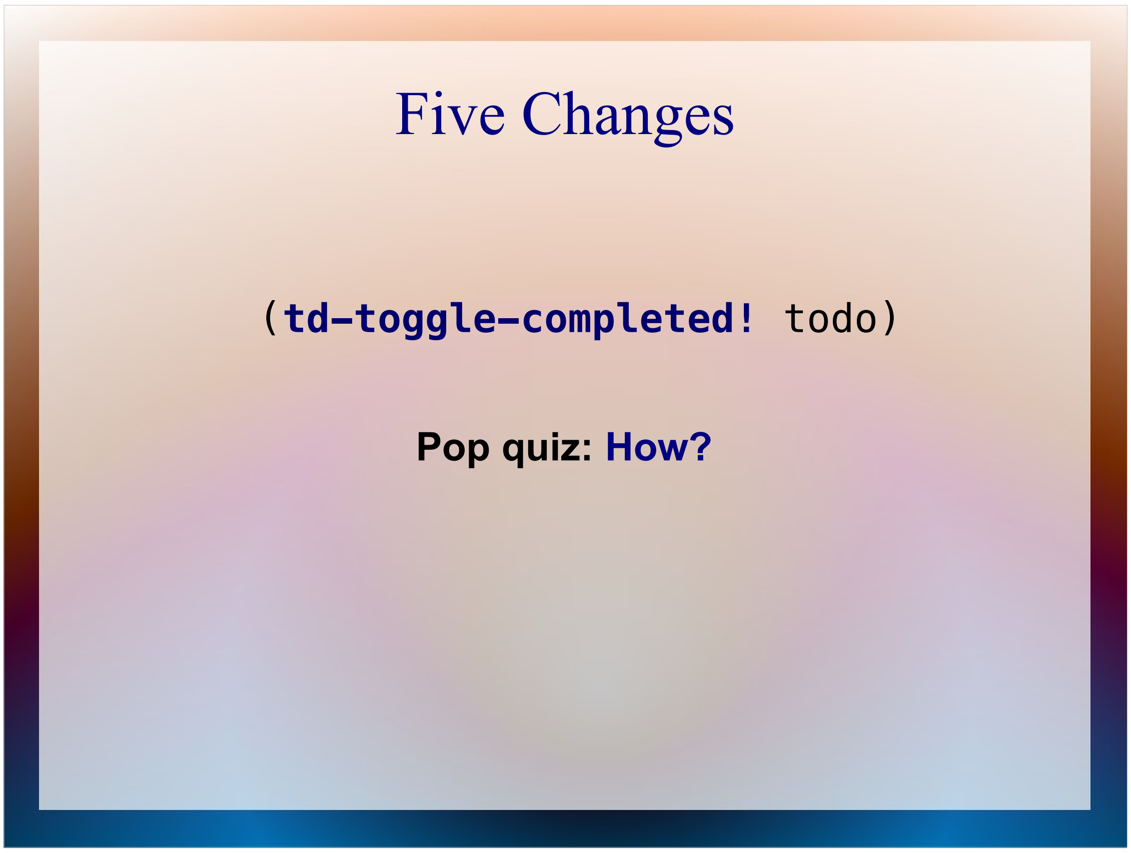
TodoMX in the Matrix

Look for:
- Data flow
- Efficiency
- Transparency
- Web components
- Object re-use
- Functional state
- Debuggability
- No web framework

Demo

DAGitty v2.3 code:

```
clear.hidden 1 @0.921,0.474
clear.onclick 1 @0.927,0.560
dash.count 1 @0.895,0.357
dash.hidden 1 @0.812,0.232
input.checked 1 @0.289,0.241
keypress 1 @0.837,0.989
localStorage E @0.707,0.841
matrix.route 1 @0.536,0.101
new_todo 1 @0.821,0.759
router_lib 1 @0.533,-0.010
todo.completed O @0.488,0.600
todo.deleted 1 @0.492,0.723
todo_list 1 @0.579,0.389
todo_list_UL 1 @0.538,0.219
toggle-all.action 1 @0.289,0.356
toggle-all.onclick 1 @0.293,0.474

clear.onclick todo.deleted
keypress new_todo
```



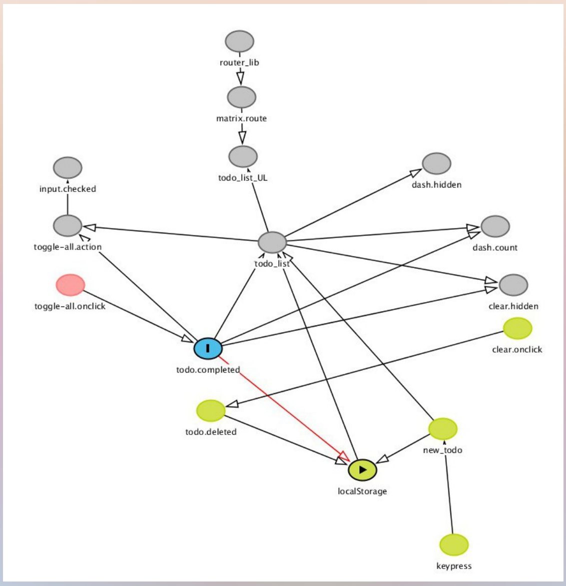
Five Changes

`(td-toggle-completed! todo)`

Pop quiz: How?

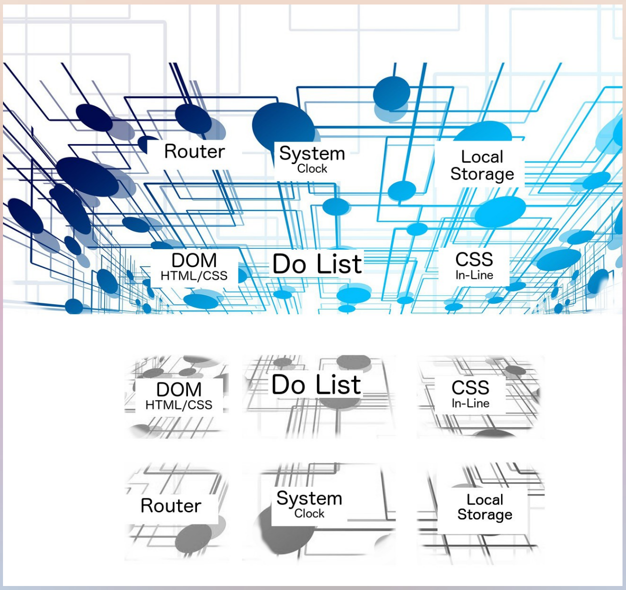
```
DAGitty v2.3 code:
clear.hidden 1 @0.921,0.474
clear.onclick 1 @0.927,0.560
dash.count 1 @0.895,0.357
dash.hidden 1 @0.812,0.232
input.checked 1 @0.289,0.241
keypress 1 @0.837,0.989
localStorage E @0.707,0.841
matrix.route 1 @0.536,0.101
new_todo 1 @0.821,0.759
router_lib 1 @0.533,-0.010
todo.completed O @0.488,0.600
todo.deleted 1 @0.492,0.723
todo_list 1 @0.579,0.389
todo_list_UL 1 @0.538,0.219
toggle-all.action 1 @0.289,0.356
toggle-all.onclick 1 @0.293,0.474

clear.onclick todo.deleted
keypress new_todo
```

TodoMX in the Matrix



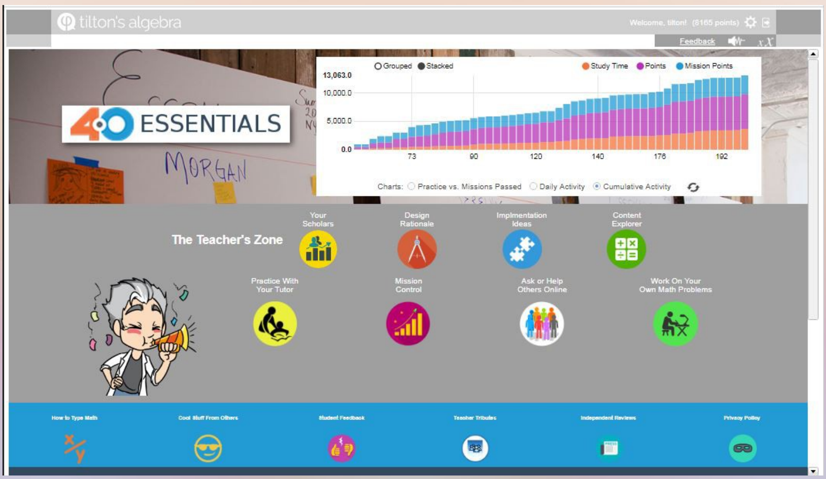
```
DAGitty v2.3 code:
clear.hidden 1 @0.921,0.474
clear.onclick 1 @0.927,0.560
dash.count 1 @0.895,0.357
dash.hidden 1 @0.812,0.232
input.checked 1 @0.289,0.241
keypress 1 @0.837,0.989
localStorage E @0.707,0.841
matrix.route 1 @0.536,0.101
new_todo 1 @0.821,0.759
router_lib 1 @0.533,-0.010
todo.completed O @0.488,0.600
todo.deleted 1 @0.492,0.723
todo_list 1 @0.579,0.389
todo_list_UL 1 @0.538,0.219
toggle-all.action 1 @0.289,0.356
toggle-all.onclick 1 @0.293,0.474

clear.onclick todo.deleted
keypress new_todo
```

The TodoMX Matrix



This Is Not Simple

tiltonsalgebra.com



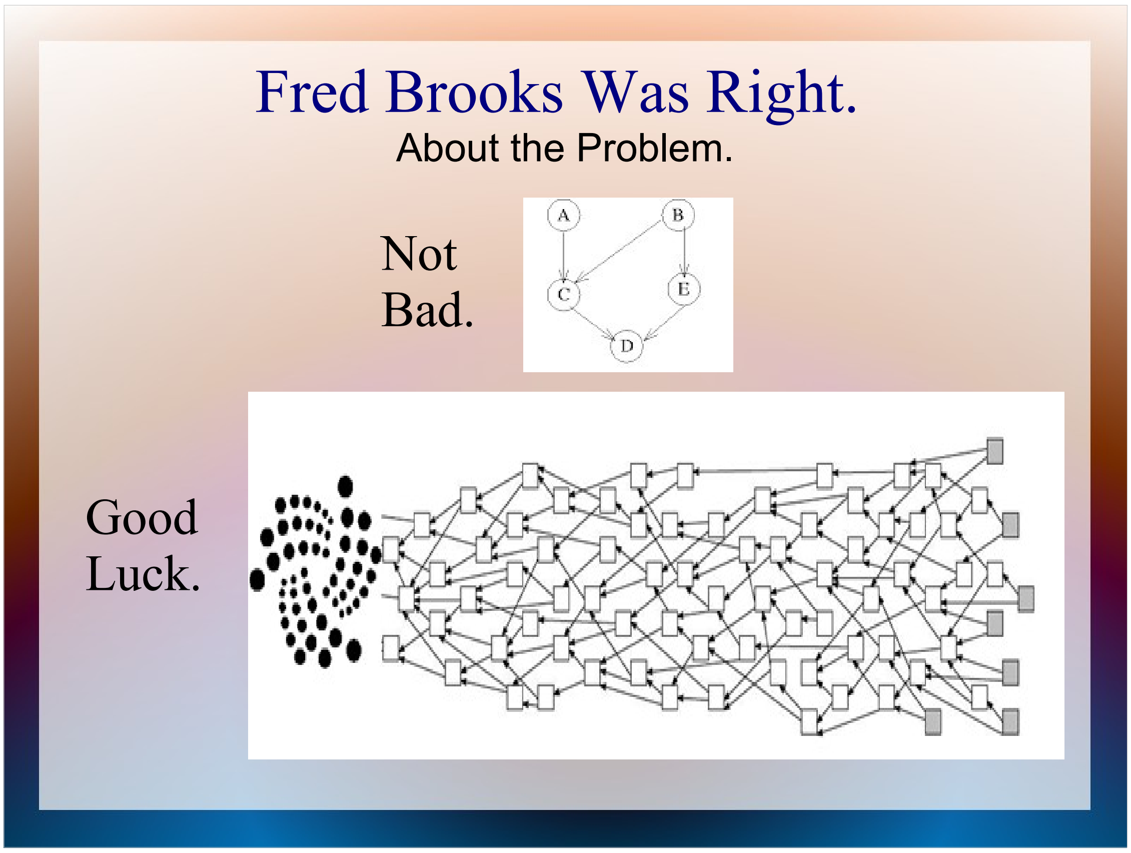
And CliniSys(tm), a Clinical Drug Trial Manager.
80kloc of Common Lisp/qooxdoo.

This app has 1218 distinct formulas. At run-time, a formula has on average 1.5 ependencies. The average formula includes 30 Lisp symbols.

On a busy screen, over 1000 cells (comprising 300 formulas) can be active.

The longest dependency chain is eleven. It works from the type-of the math in the last problem step to the highlighting of that step.

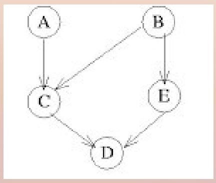
A second experience report was a complete clinical drug trial management system in which the persistent CLOS database was also driven by cells.



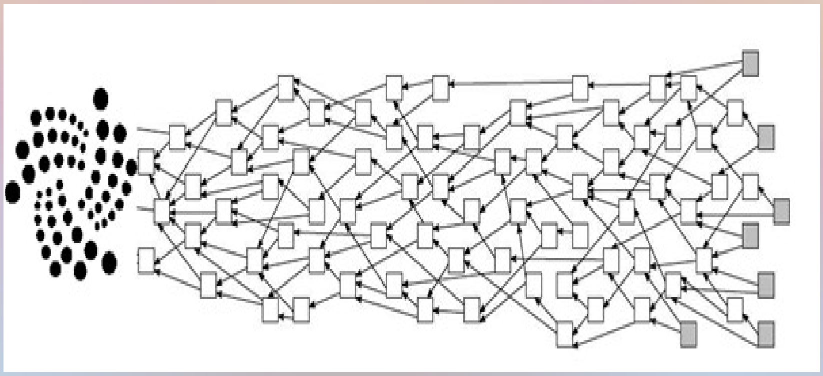
Fred Brooks Was Right.

About the Problem.

Not Bad.

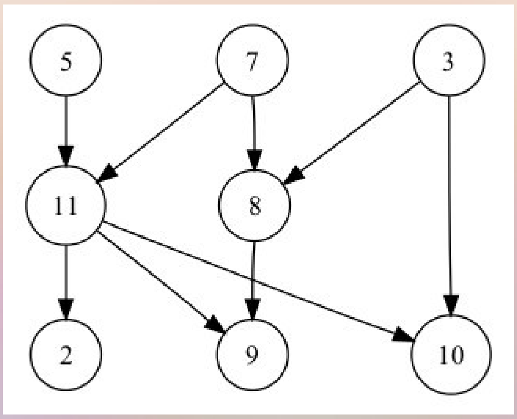


Good Luck.



No Silver Bullet: http://worrydream.com/refs/Brooks-NoSilverBullet.pdf

But Sean Parent Saw This...



Pop quiz: Notice anything?

That is a bipartite graph

The Adobe Adam/property model overview:
http://stlab.adobe.com/group__asl__overview.html#asl_ove

The Google Talk Haskell Question:
https://youtu.be/4moyKUHApq4?t=49m17s

"...a large class of applications."

It is not just for user interfaces.

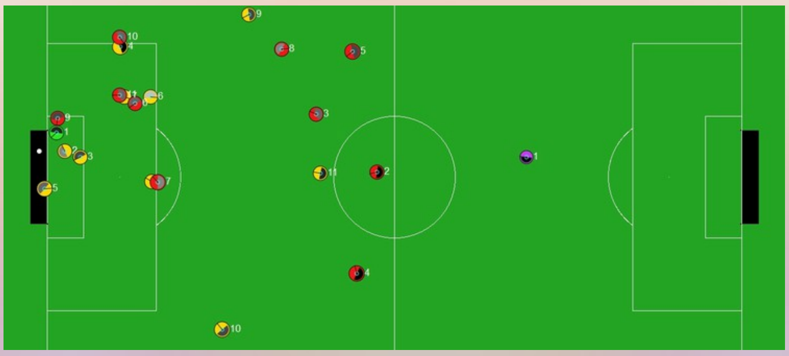
"It is useful for any application involving an interesting amount of long-lived state and a stream of unpredictable inputs."

http://smuglispweeny.blogspot.com/2008/02/cells-manifesto.html

- User interfaces: Yes
- Real-time process control: Yes
- ETL: Not so much
- RoboCup Simulation League: Yes

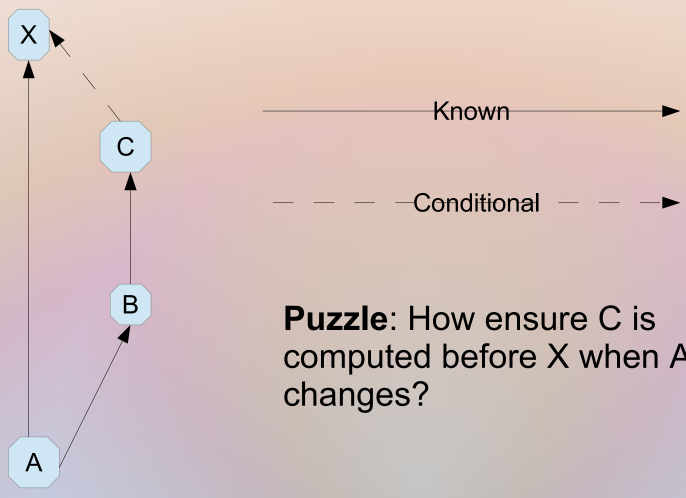
Glitches and RoboCup Sim

The Cells are dead! Long live Cells!



One (UDP) input: a visual sensory dump every 100ms.

The Glitch



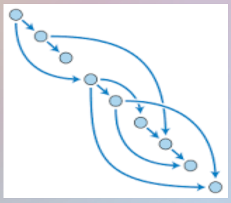
Puzzle: How ensure C is computed before X when A changes?

Anti-glitch schemes

- Topological sort
- Mark and sweep (MobX)
- “Dirty” counting (MobX originally)
- Integer pulse/clock (Cells)

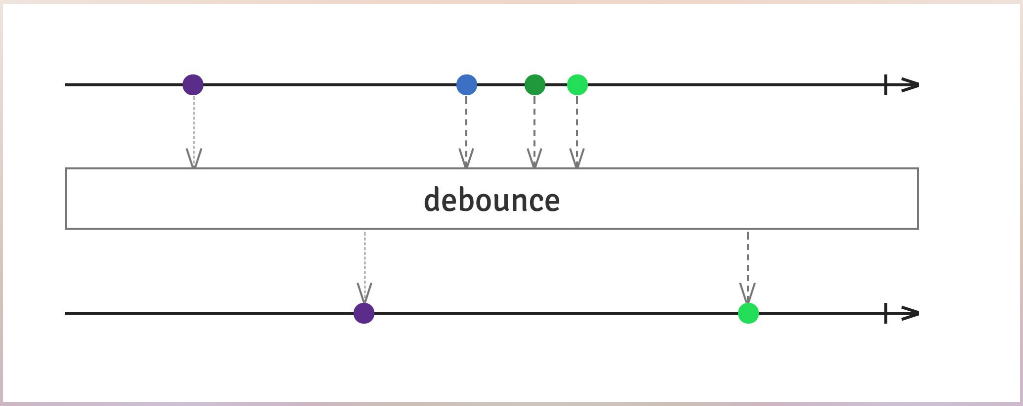
Puzzle: Why This?

Why did RoboCup Sim fail on glitches when a 79kloc clinical drug trial system did not?

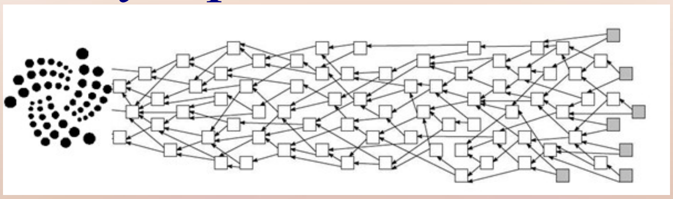


ReactiveX/RxJS

Streamthink, or Data flow is not the only game in town.



Synapses as Streams



syn·apse [sin-aps, si-naps] Physiology noun - A junction between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter.

```
(let [sensor (cI nil)
      alarm (cF (if (with-synapse (:delta-sensor [prior (atom nil)])
                      (when-let [reading (<cget sensor)]
                       (let [delta (Math/abs
                                      (if @prior
                                          (- reading @prior)
                                          0))]
                          (when (> delta 5)
                             (reset! prior reading)
                             delta))))
                   :on :off)))]
     ...etc...)
```

The Reactive Field

I look for declarative, transparent authoring, point-to-point dependency, automatically executed glitch-free. Avoid "lifting" RX.

- Common Lisp Garnet’s KR; cool but quiescent.
- Philip Eby’s Python Cells clone, Trellis;
- Adobe’s C++ Adam;
- Lifting dataflow: CLJS Javelin;
- Michael Weststrate’s Javascript MobX
- CL Screamer (constraints)
- FrTIme/FlapJax: also lifting

“Lifting” Data Flow

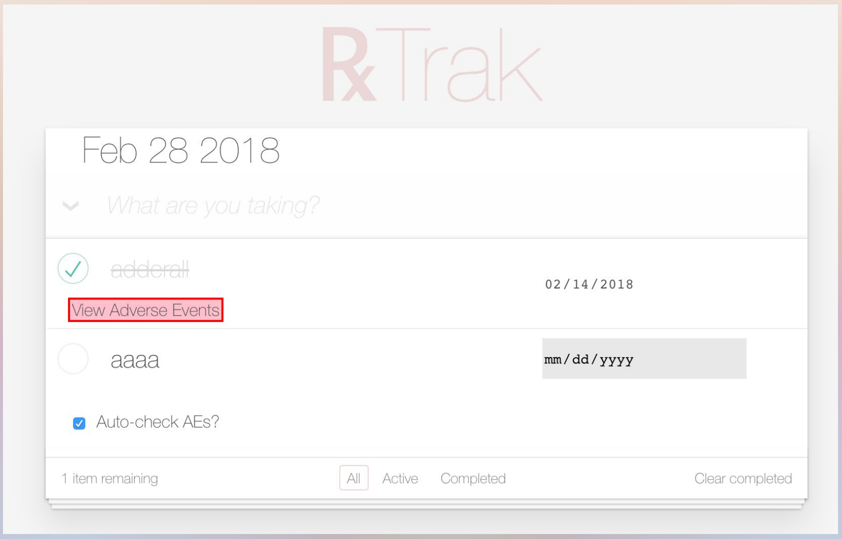
X := (if A B C) where all vars are Cells.

Pop quiz: On how many Cells does X depend?

```
(defn get-Z [] Z)
X := (+ Y (get-Z))
```

Trick quiz: On how many Cells does X depend?

RxTrak



RxTrak

Feb 28 2018

What are you taking?

✓ ~~adderall~~ 02/14/2018

View Adverse Events

aaaa mm/dd/yyyy

✓ Auto-check AEs?

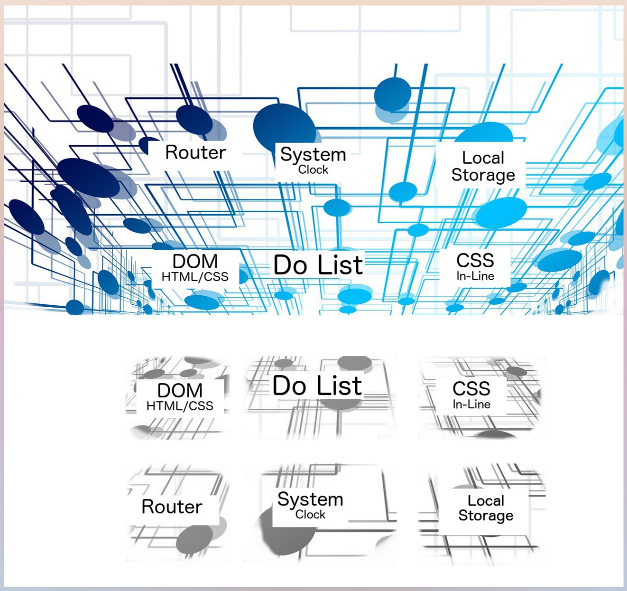
1 item remaining | All | Active | Completed | Clear completed

Callback Hell Meets Data Flow

```
(defn xhr-send [xhr]
  (go
    (let [response (<! (client/get (<mget xhr :uri)
                                   {:with-credentials? false}))]
      (with-cc :xhr-handler-sets-responded
        (mset!> xhr :response
          {:status (:status response)
           :body   (if (:success response)
                     ((:body-parser @xhr)
                      (:body response))
                     [(:error-code response)
                      (:error-text response)])})))))
```

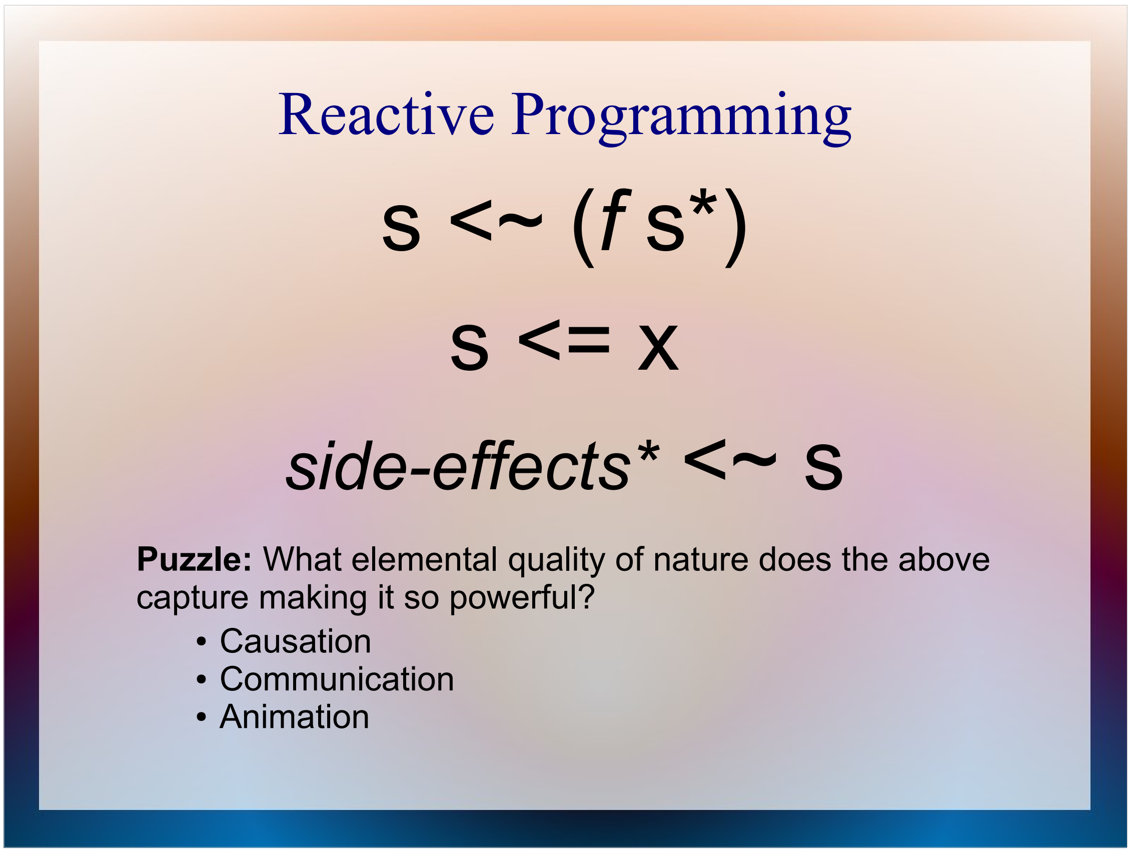
Look for `mset!>`

The TodoMX Matrix



The Benefits

GUI geometry made trivial
Yes Silver Bullet
The Grail of Object Reuse
Efficiency for free
Callback Hell eliminated
Web frameworks obvisted
ReactX/RxJS for free
Reliability
Debugability
Fun



Reactive Programming

s <~ (*f* s*)

s <= x

*side-effects** <~ s

Puzzle: What elemental quality of nature does the above capture making it so powerful?

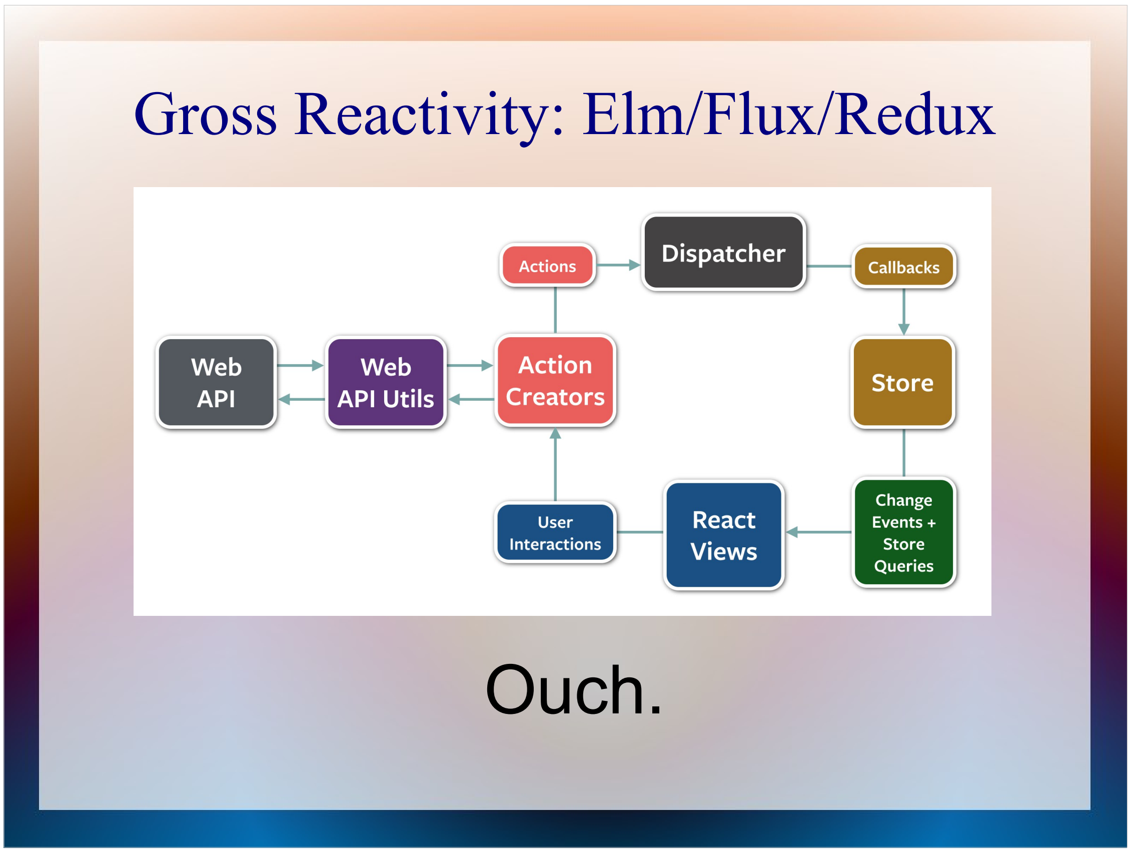
- Causation
- Communication
- Animation

Something Is Going To Happen

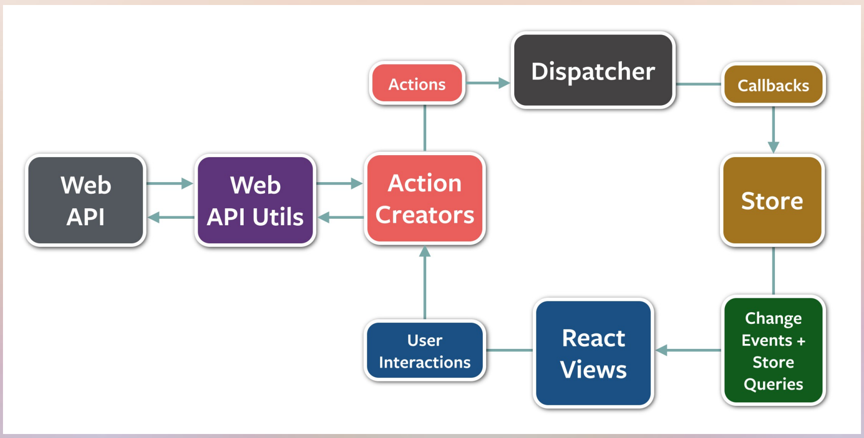
Great scene from 2010, sequel to 2001 Space Odyssey: https://youtu.be/V5HA-umweZ0

ReactJS Evangelized Declarative

```
class TodoList extends React.Component {
  render() {
    return (
      <ul>
        {this.props.items.map(items => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
```

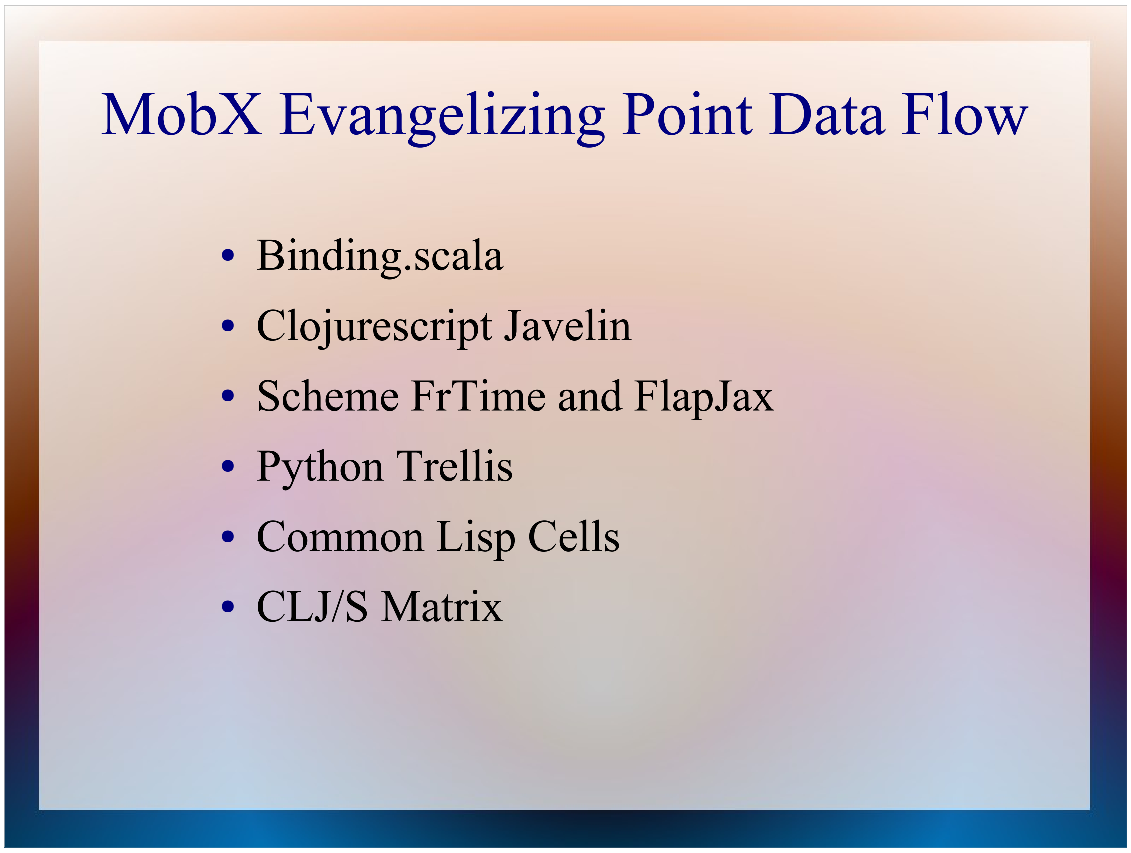


Gross Reactivity: Elm/Flux/Redux



Ouch.

Flux: https://github.com/facebook/flux

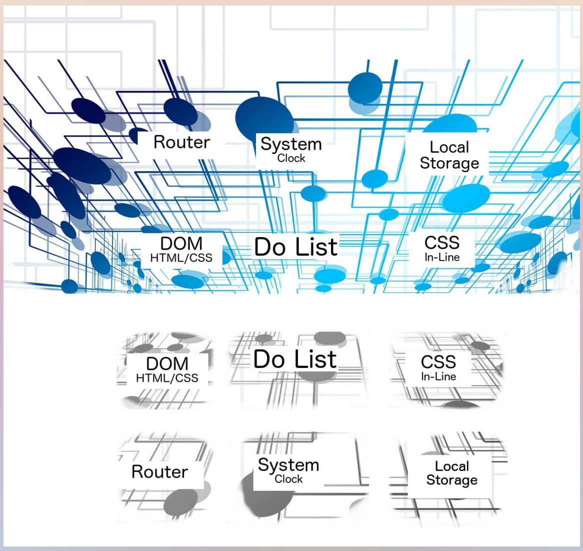


MobX Evangelizing Point Data Flow

- Binding.scala
- Clojurescript Javelin
- Scheme FrTime and FlapJax
- Python Trellis
- Common Lisp Cells
- CLJ/S Matrix

They Will Find It

The last piece of the puzzle: Matrix



Appendix

More slides with more information on reactive programming follow.

Common Lisp

Clojure

Evaluating Reactive Libraries

- Is it a “lifting” library? Problem.
- Is pub-sub explicit? Awkward.
- Is it really Reactive? ReactJS is not, but it is declarative which is nice.
- Does it have “glitches”? Not the end of the world, but others are glitch-free so why not?

FrTime, FlapJax

- FrTime: Scheme FRP: http://docs.racket-lang.org/frtime/
- FlapJax: FRP Web framework, a Javascript translation of FrTime: http://www.flapjax-lang.org
- Javelin (ClojureScript)
- “lifting” implementation has efficiency and coding issues: (if A B C) and (+ 40 (compute-with box)),

Trellis for Pythonistas

- Phillip Eby's development by Cells
- http://peak.telecommunity.com/DevCenter/Trellis
- https://pypi.python.org/pypi/Trellis/0.7a1
- David Gelerntner's “Trellis” from “Mirror Worlds”: https://global.oup.com/academic/product/mirror-worlds-9780195079067?cc=us&lang=en&

Functional Reactive Programming (FRP)

- Conan Eliot Lambda-Jam 2015 https://youtu.be/j3Q32brCUAI
- Continuous time, yes
- Graphs and dataflow, no.