Building an

EXPERIMENTATION

platform in Clojure

@nid90

@sriharisriraman

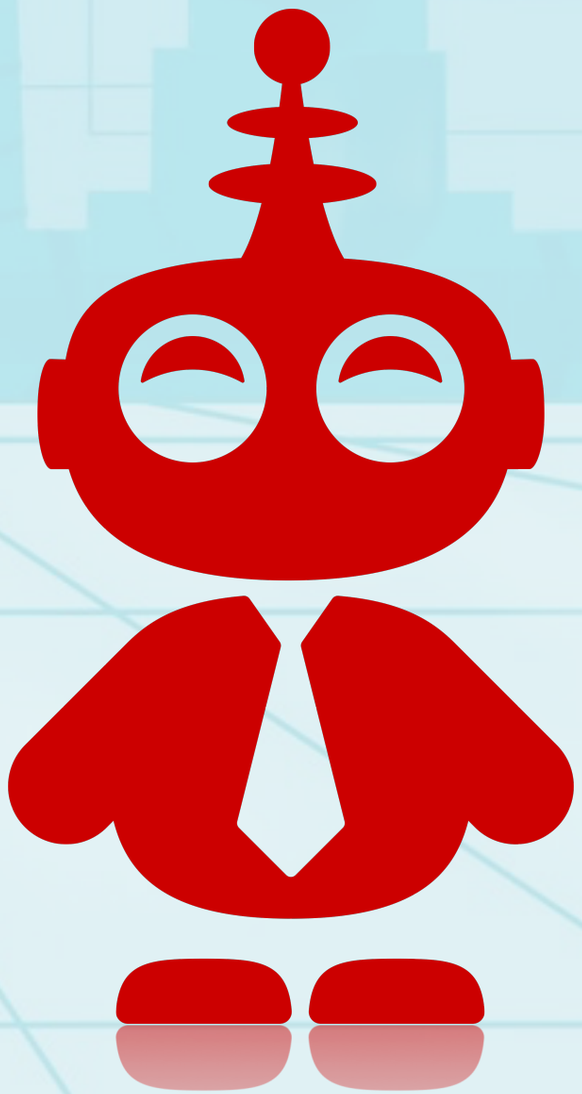
nilenso

@nid90

@sriharisriraman

what we built

- built at Staples-SparX
- one box serving all Staples's experimentations
- 8 GB of data per day
- 5 million sessions a day
- 500 requests per second
- SLA of 99.9th percentile at 10ms



what you will learn

- values of different experiments setup
- how to efficiently use traffic
- some nice things about clojure
- building assembly lines using core.async
- putting a complex system under simulation testing

structure of the talk

1. explaining experimentation
2. implementation
3. simulation testing

explaining experimentation

the experimental method.

experimentation is the step in the scientific method that helps people decide between two or more competing explanations – or hypotheses.

experimentation in business

- a process for where business ideas can be evaluated at scale, analyzed scientifically and in a consistent manner

- data driven decisions

hypotheses

- “a red button will be more compelling than a blue button”
- algorithms, navigation flows
- measurement of overall performance of an entire product

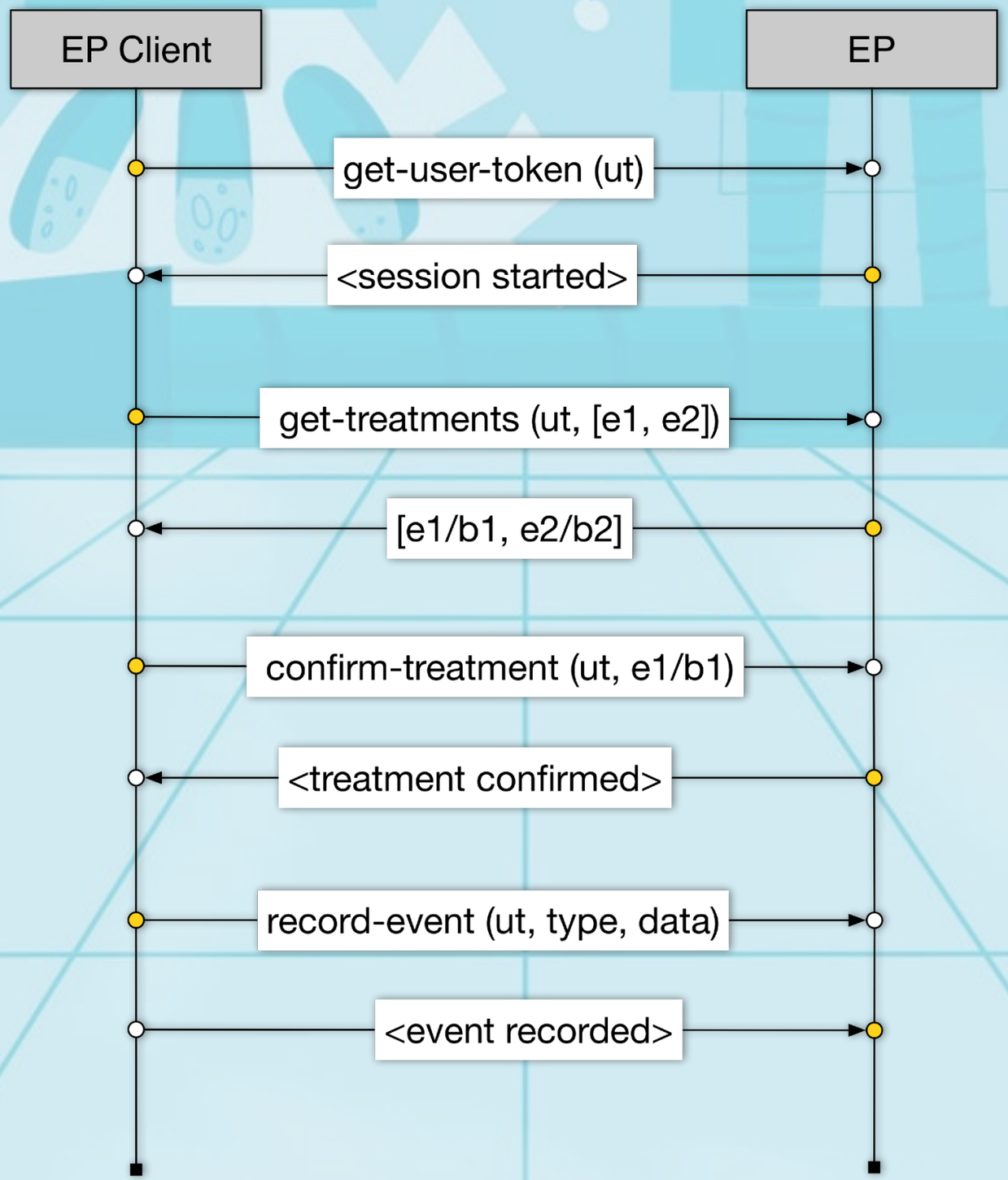
treatment

- values for the variables in the system under investigation
- control (no treatment) vs test (some treatment)
- red/blue/green

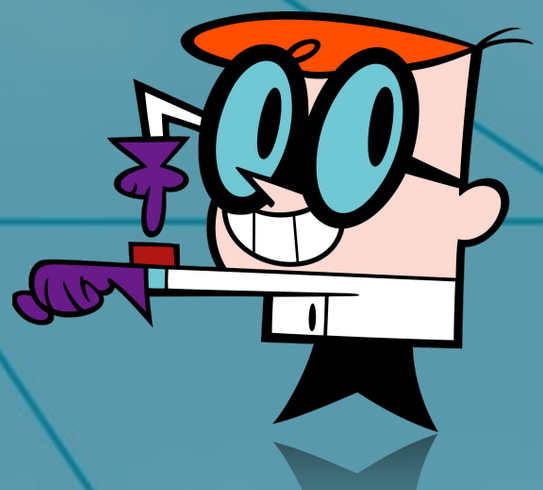
coverage

- effect of external factors (business rules, integration bug, etc.)
- fundamental in ensuring a precise measurement
- design: not covered by default

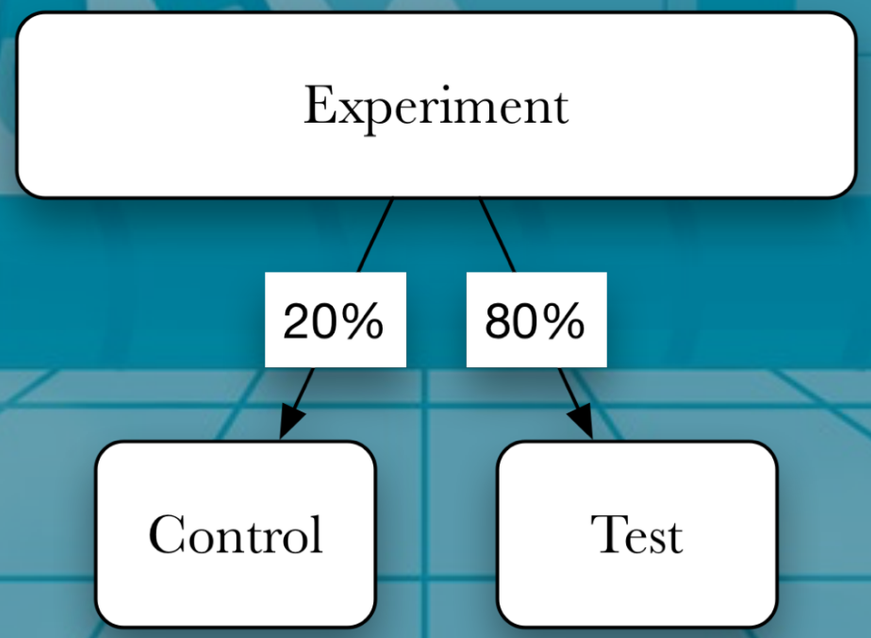
sequence of interactions



experiment infrastructure

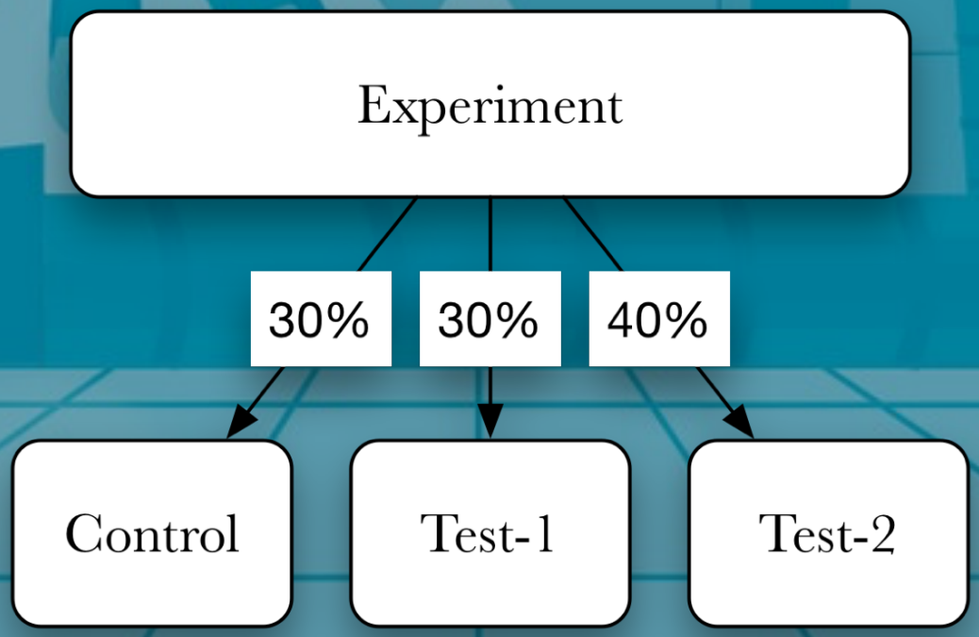


A/B



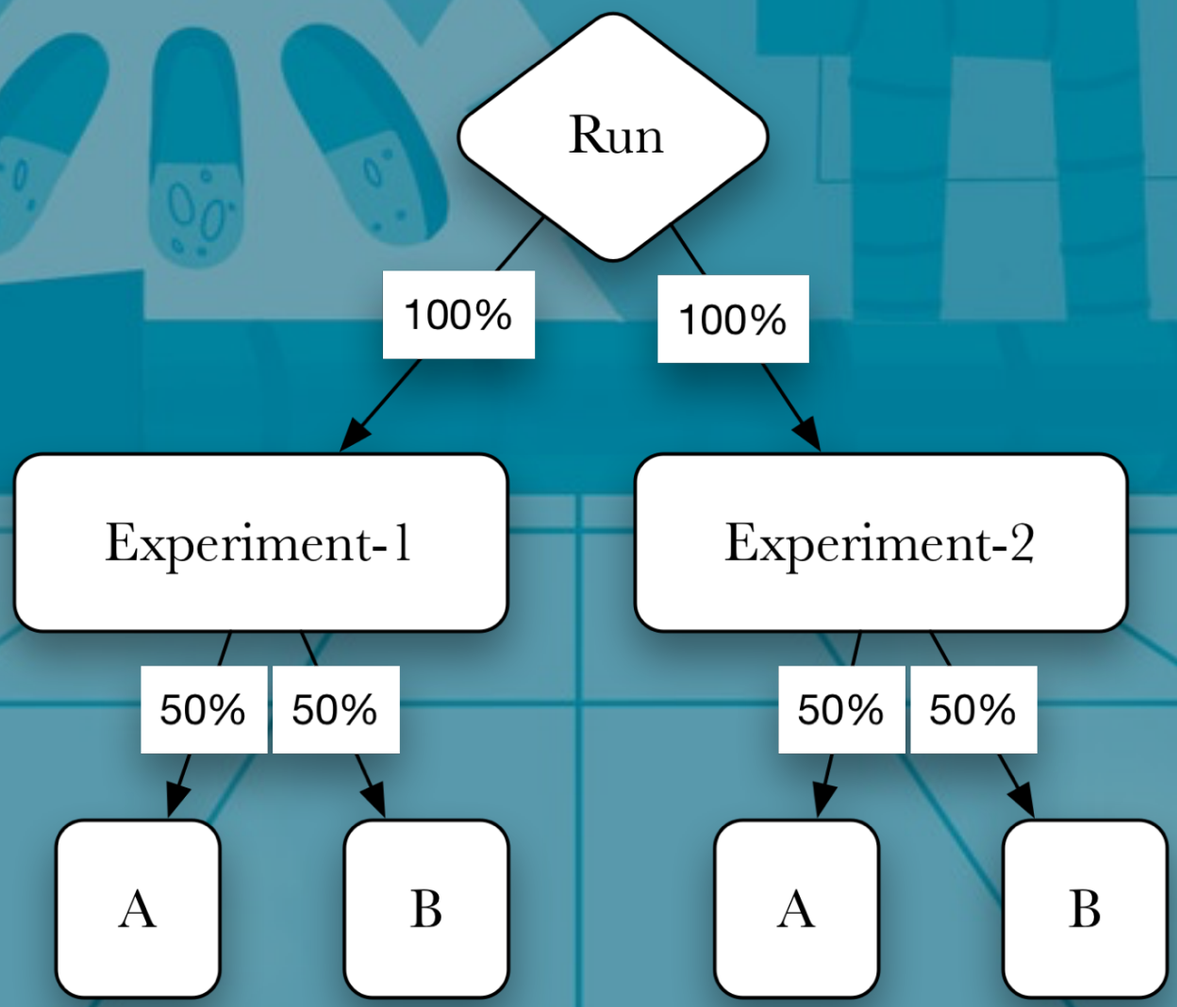
traffic is split

A/B/C



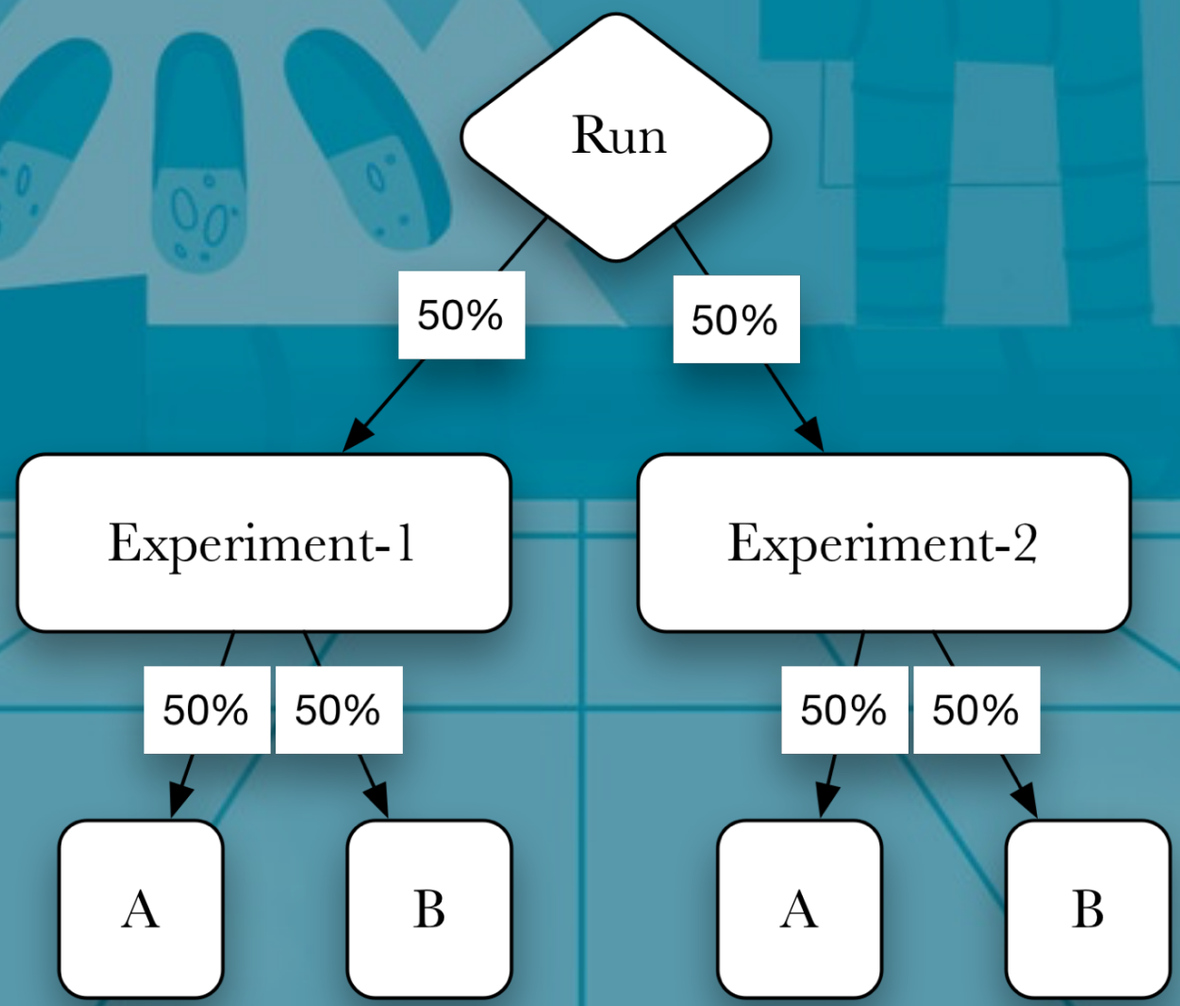
no limitation in the number of treatments you can associate to an experiment

messy



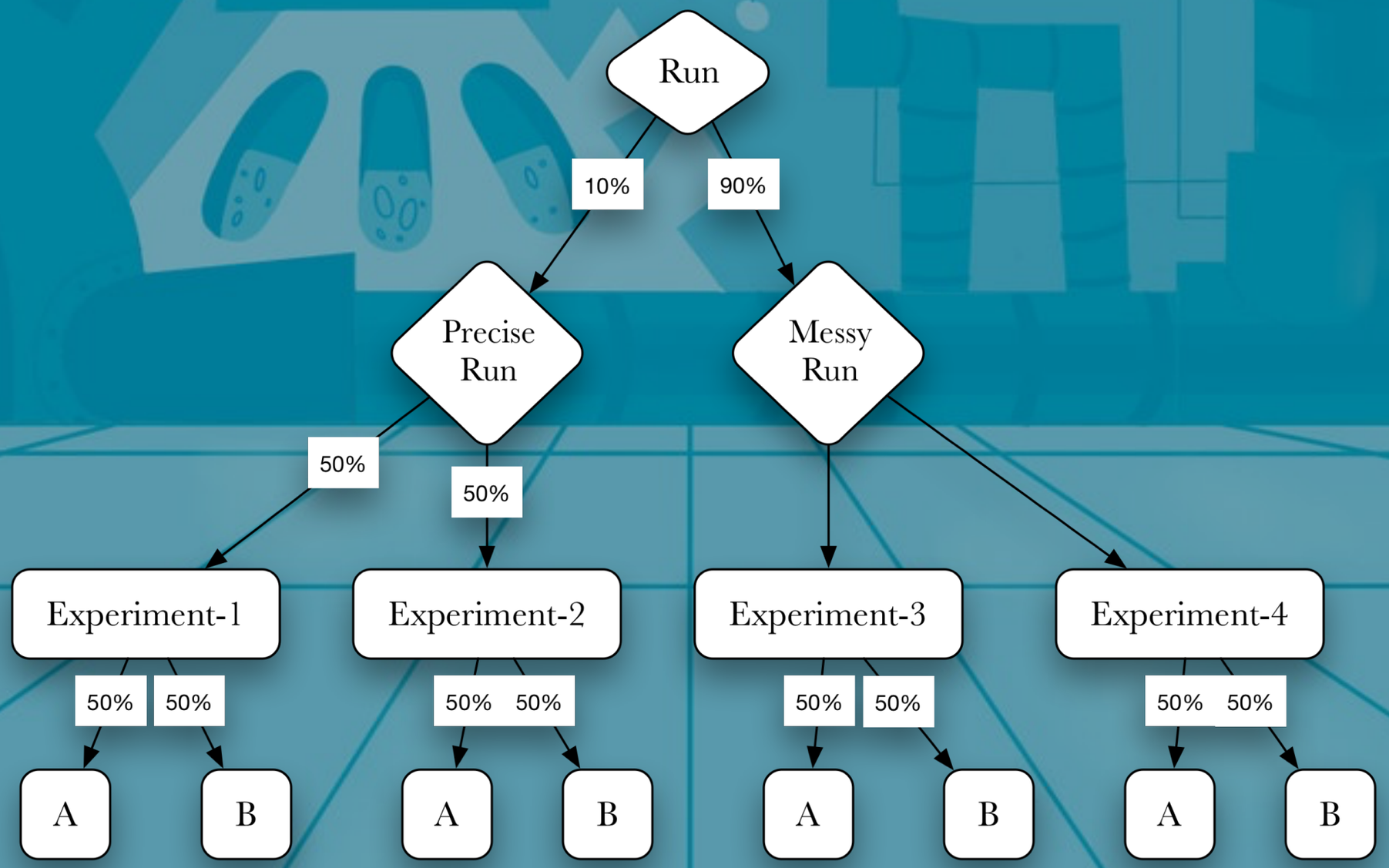
testing orthogonal hypotheses

precise



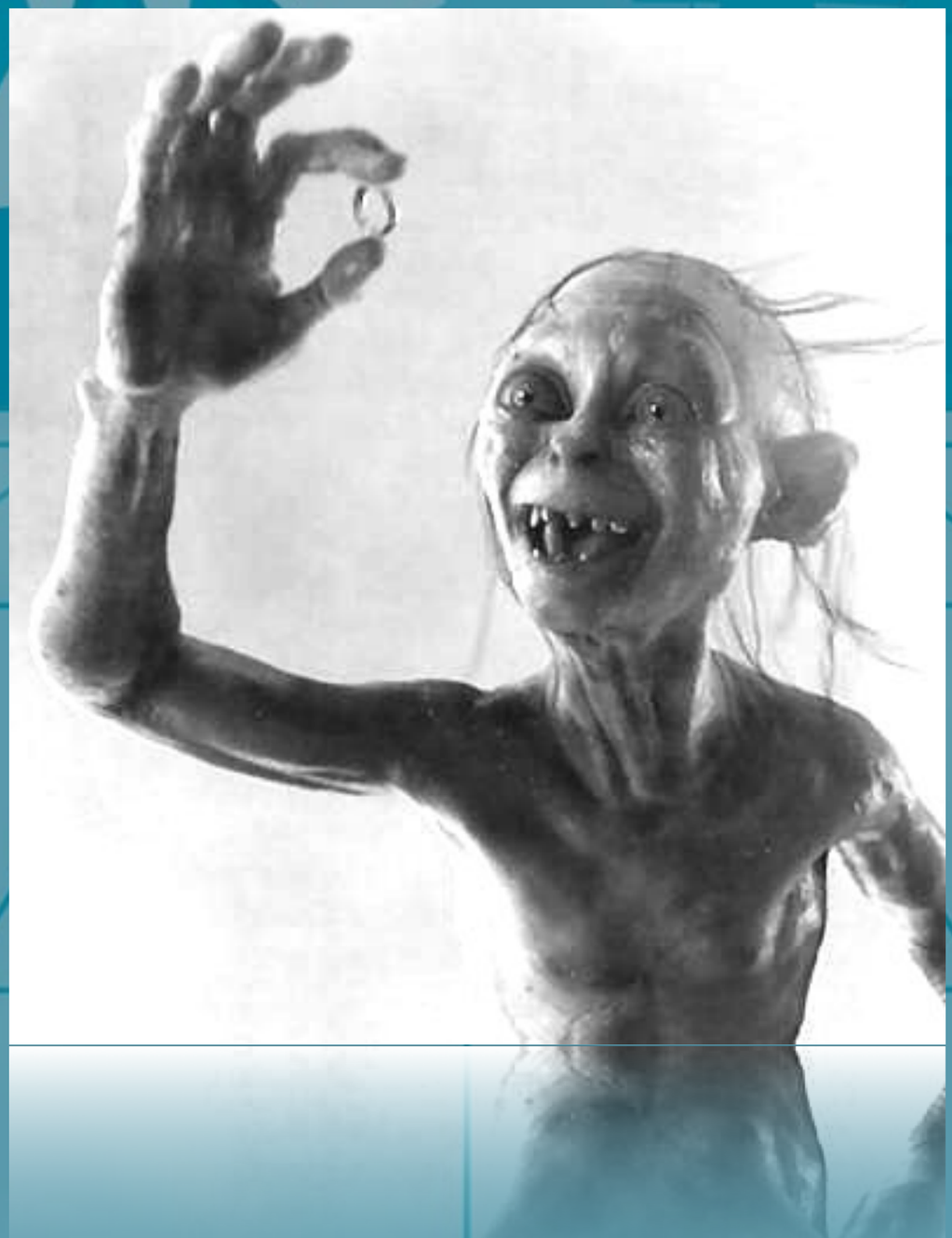
testing non-orthogonal hypotheses

messy/precise

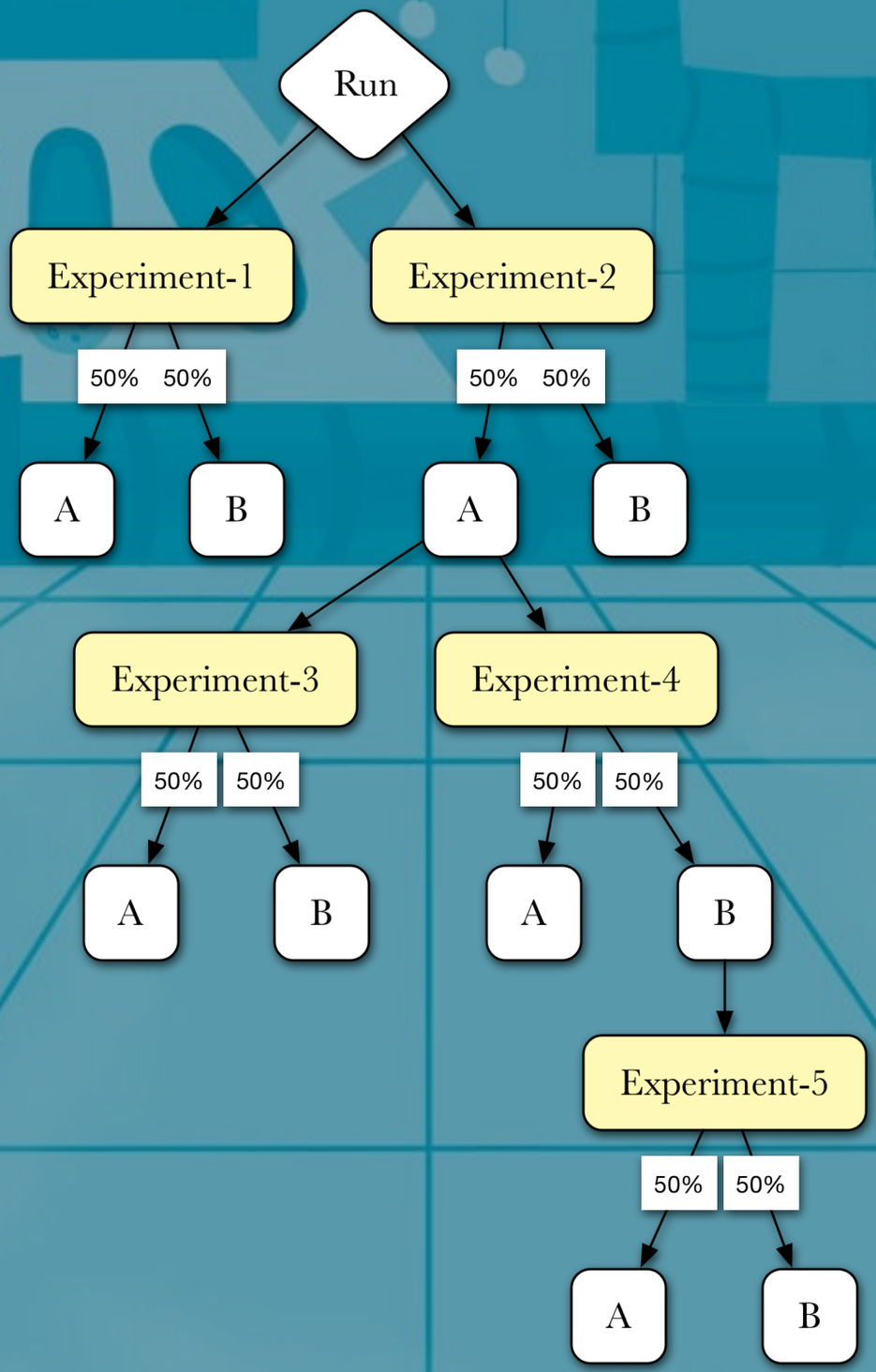


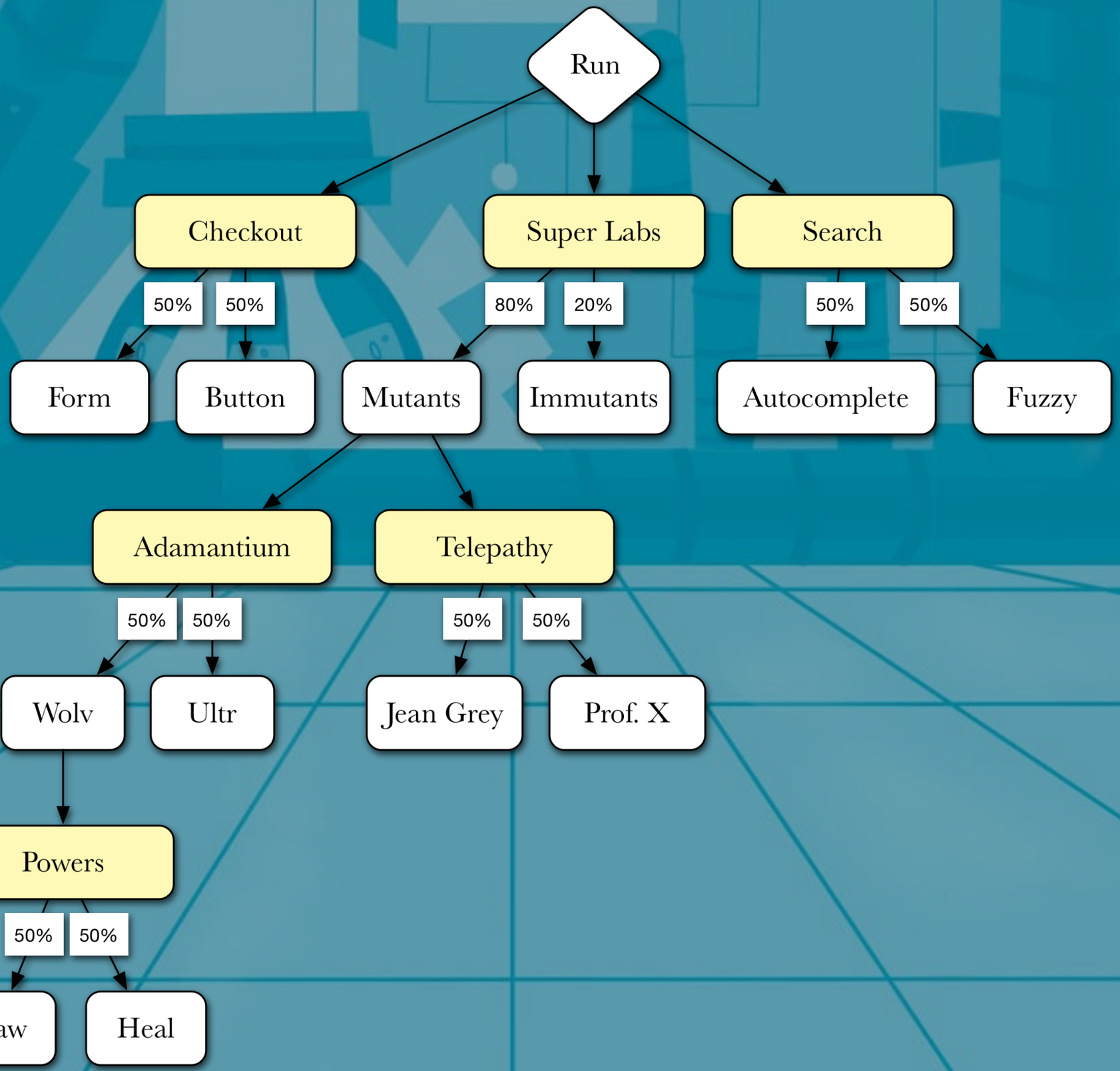
first version of experiment infrastructure

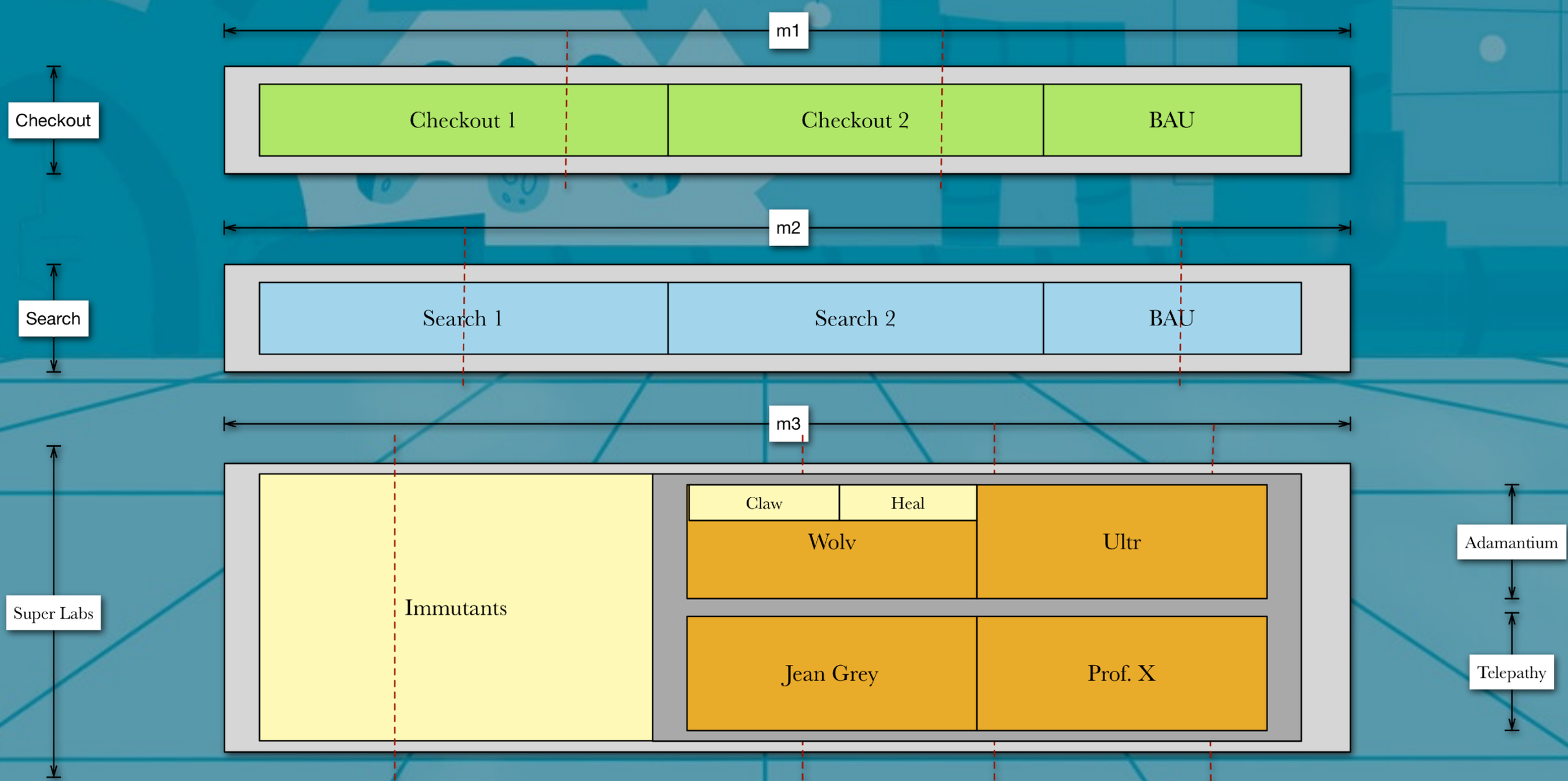
traffic is precious



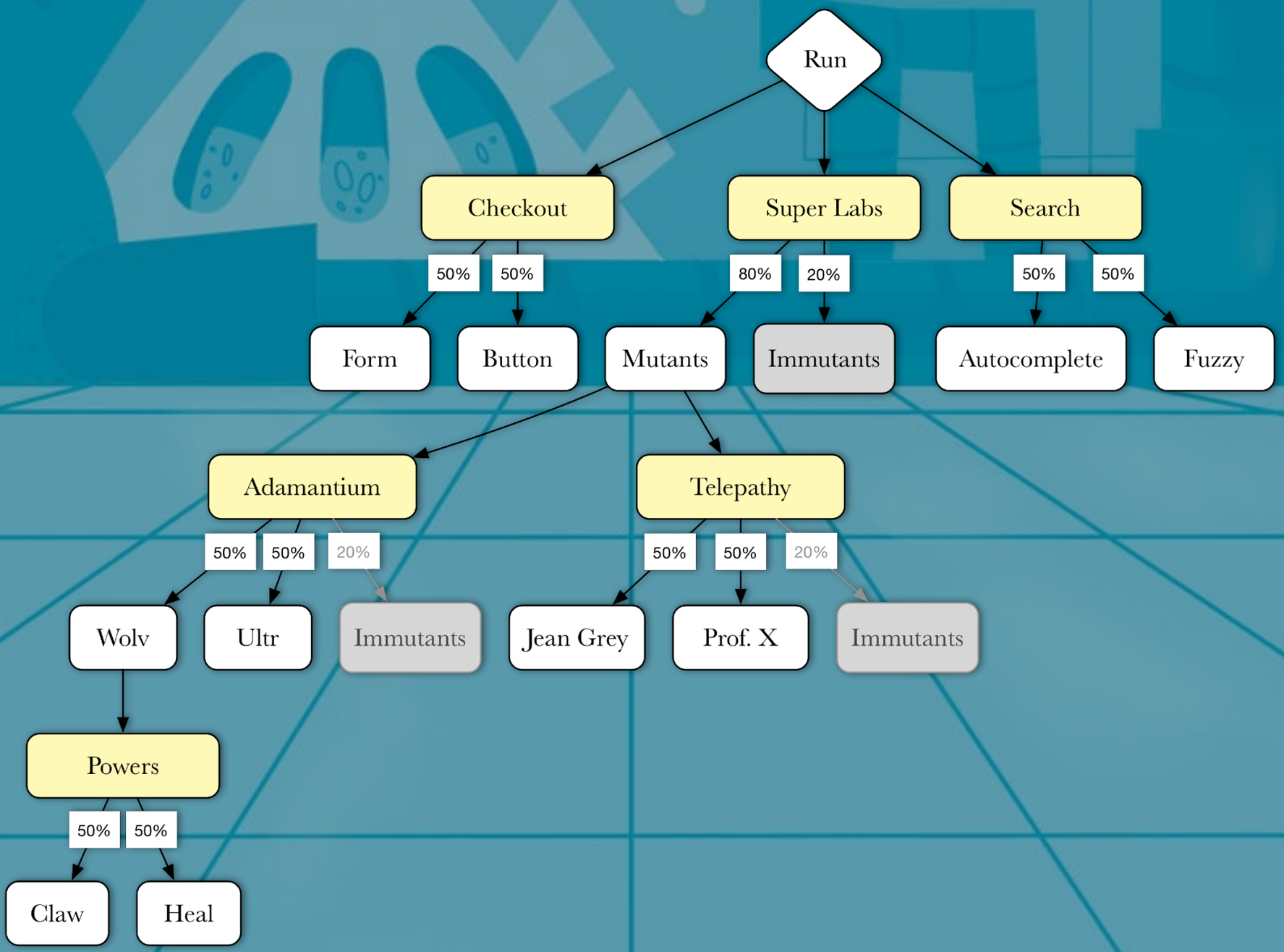
nested


Run
Experiment-1
Experiment-2
50%
50%
50%
50%
A
B
A
B
Experiment-3
Experiment-4
50%
50%
50%
50%
A
B
A
B
Experiment-5
50%
50%
A
B

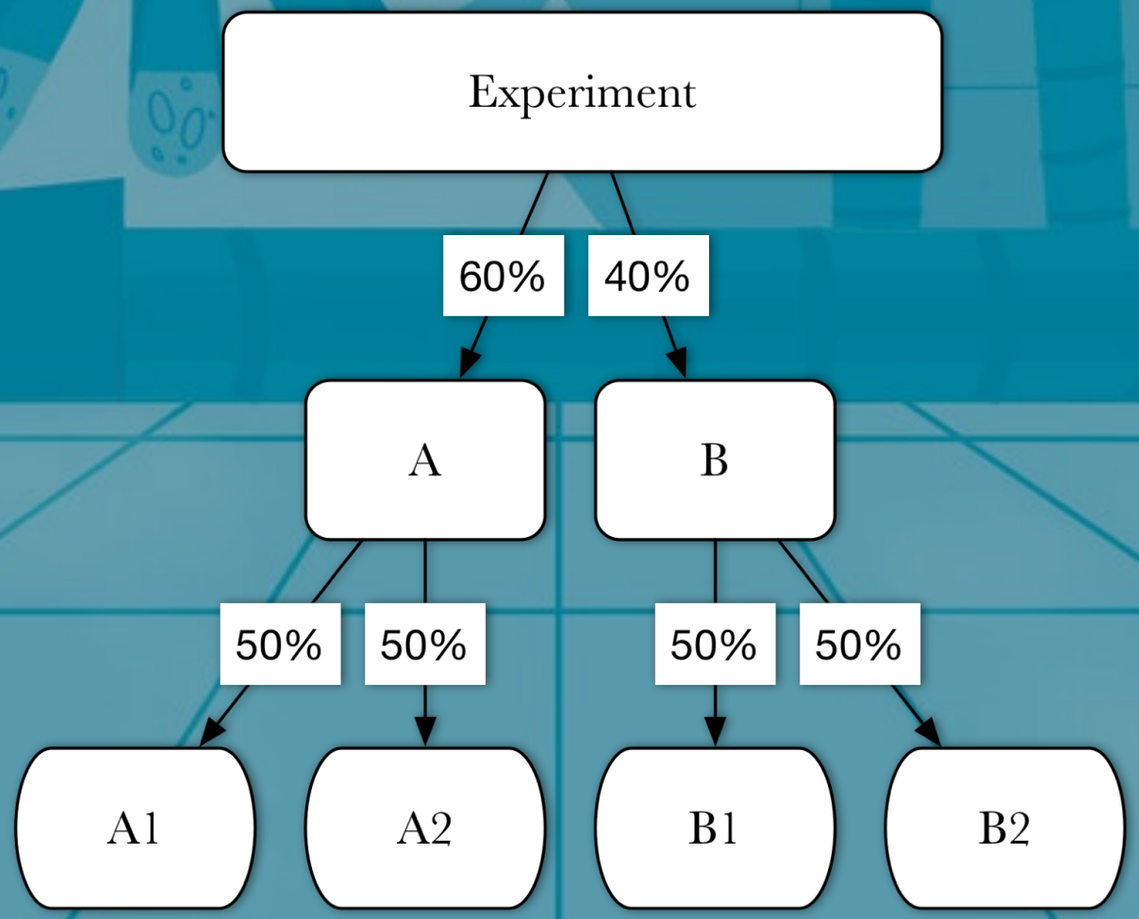

Run
Checkout
Super Labs
Search
50%
50%
80%
20%
50%
50%
Form
Button
Mutants
Immutants
Autocomplete
Fuzzy
Adamantium
Telepathy
50%
50%
50%
50%
Wolv
Ultr
Jean Grey
Prof. X
Powers
50%
50%
Claw
Heal


m1
Checkout
Checkout 1
Checkout 2
BAU
m2
Search
Search 1
Search 2
BAU
m3
Super Labs
Immutants
Claw
Heal
Wolv
Ultr
Jean Grey
Prof. X
Adamantium
Telepathy

shared bucket



A/A



null hypothesis test

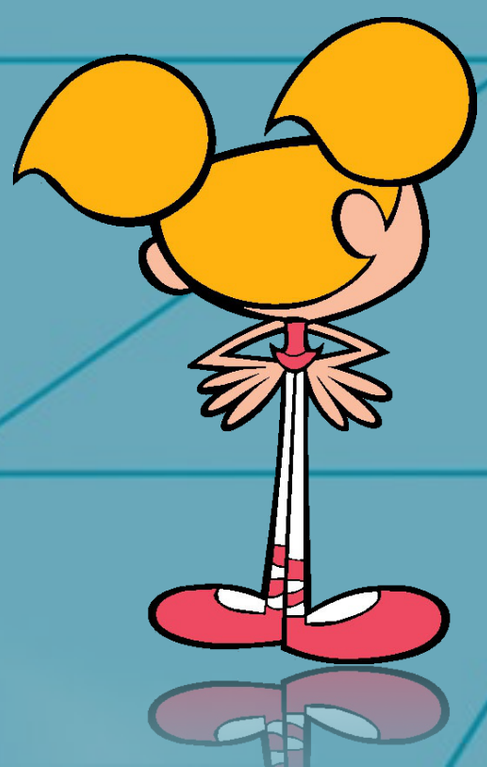
why build ep?

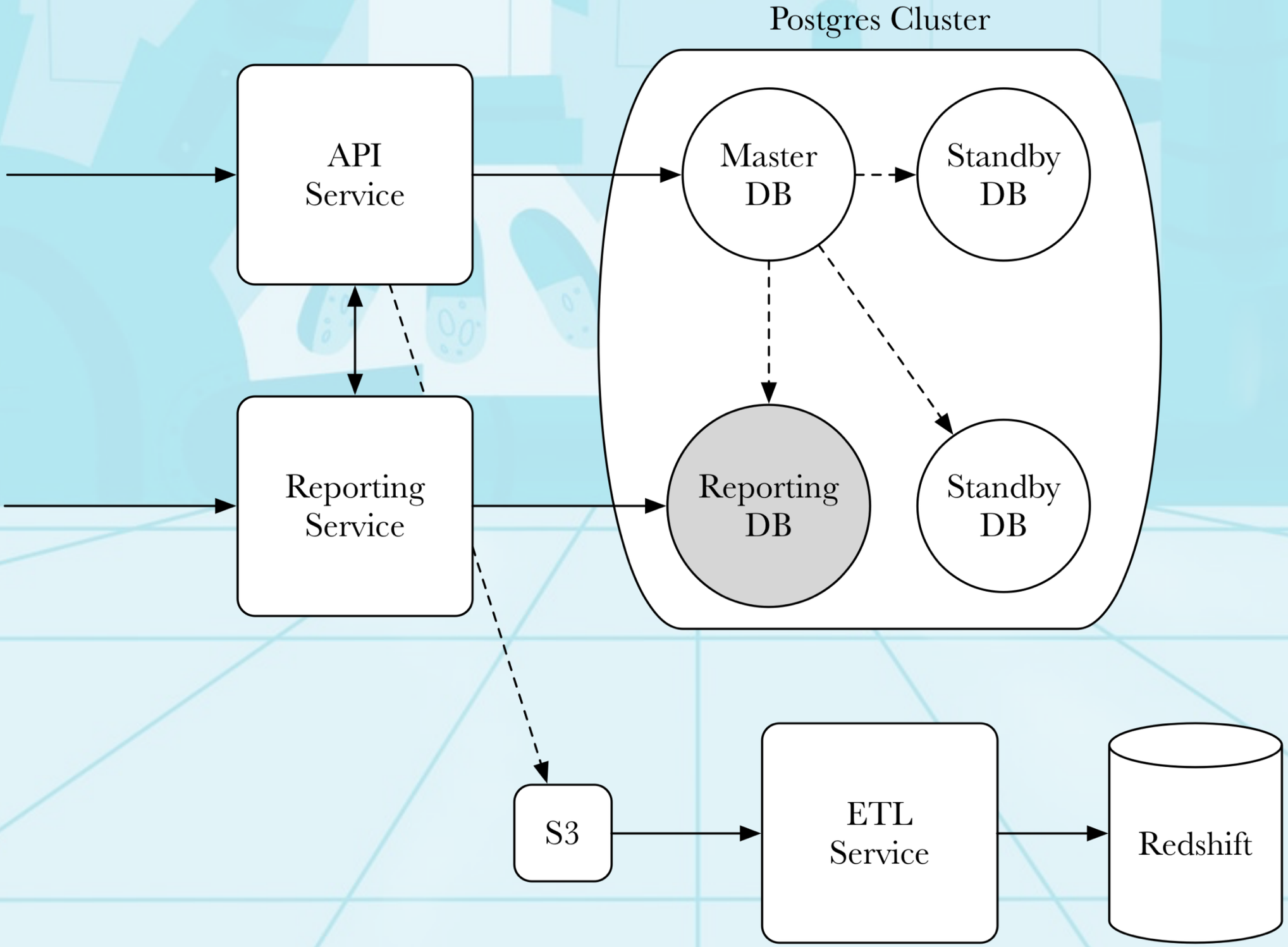
- capacity to run a lot of experiments in parallel
- eCommerce opinionated
- low latency (synchronous)
- real time reports
- controlled ramp-ups
- layered experiments
- statistically sound (needs to be auditable by data scientists, CxOs, etc.)
- deeper integration

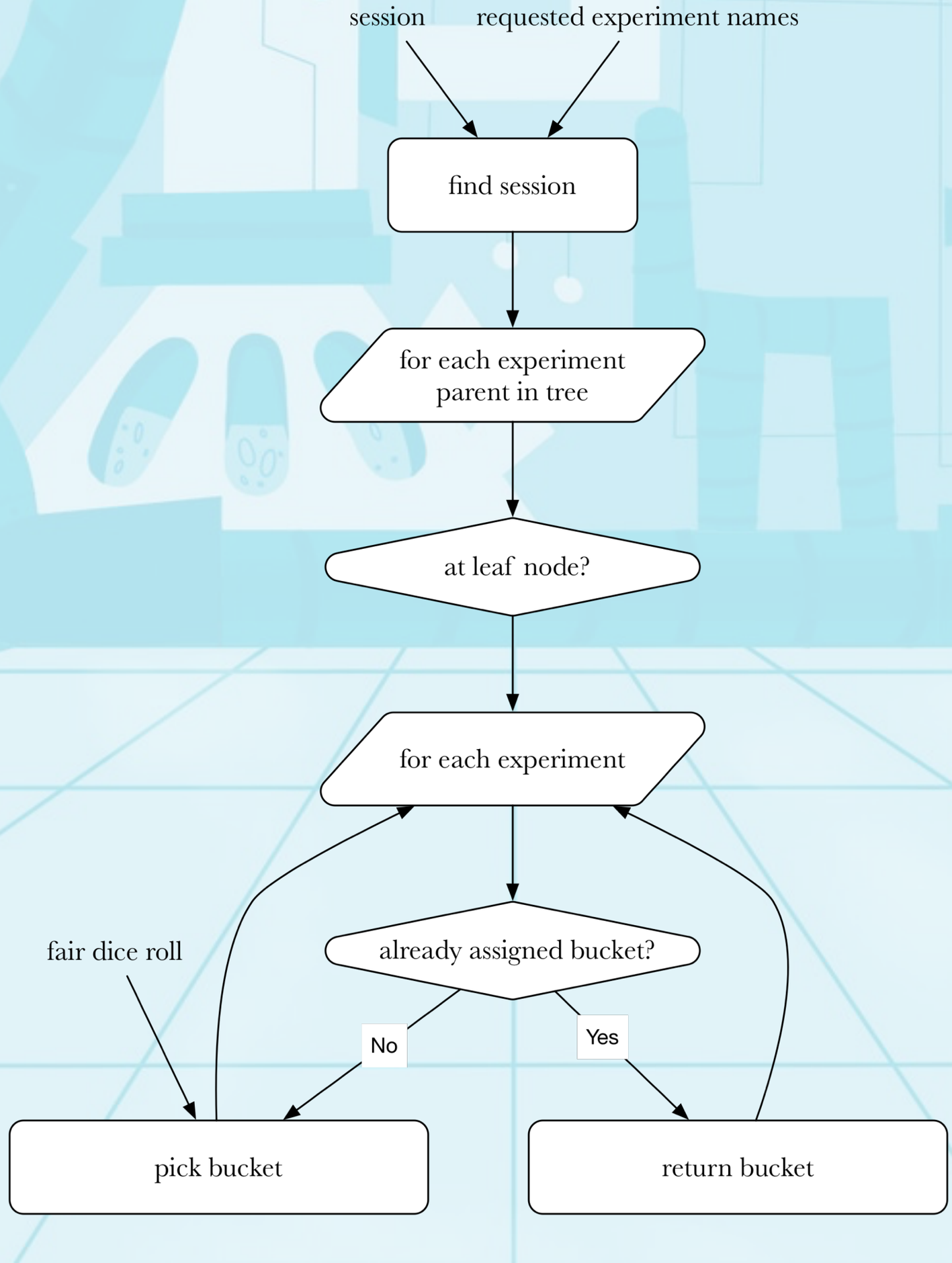
परन्तु

- the domain is quite complex
- significant investment of time, effort and maintenance (takes years to build correctly)
- you might not need to build this if your requirements can be met with existing 3rd party services.

implementation




Postgres Cluster
API Service
Master DB
Standby DB
Reporting Service
Reporting DB
Standby DB
S3
ETL Service
Redshift


session
requested experiment names
find session
for each experiment parent in tree
at leaf node?
for each experiment
already assigned bucket?
fair dice roll
No
Yes
pick bucket
return bucket

postgres cluster

- data centered domain
  - data integrity
  - quick failover mechanism
- no out of the box postgres cluster management solution
- built it ourselves using repmgr
- multiple lines of defense
  - repmgr pushes
  - applications poll
  - zfs - mirror and incremental snapshots

reporting on postgres

- sweet spot of a medium sized warehouse
- optimized for large reads
- streams data from master (real time reports)
- crazy postgres optimizations
- maintenance (size, bloat) is non trivial
- freenode#postgresql rocks!

real OLAP solution

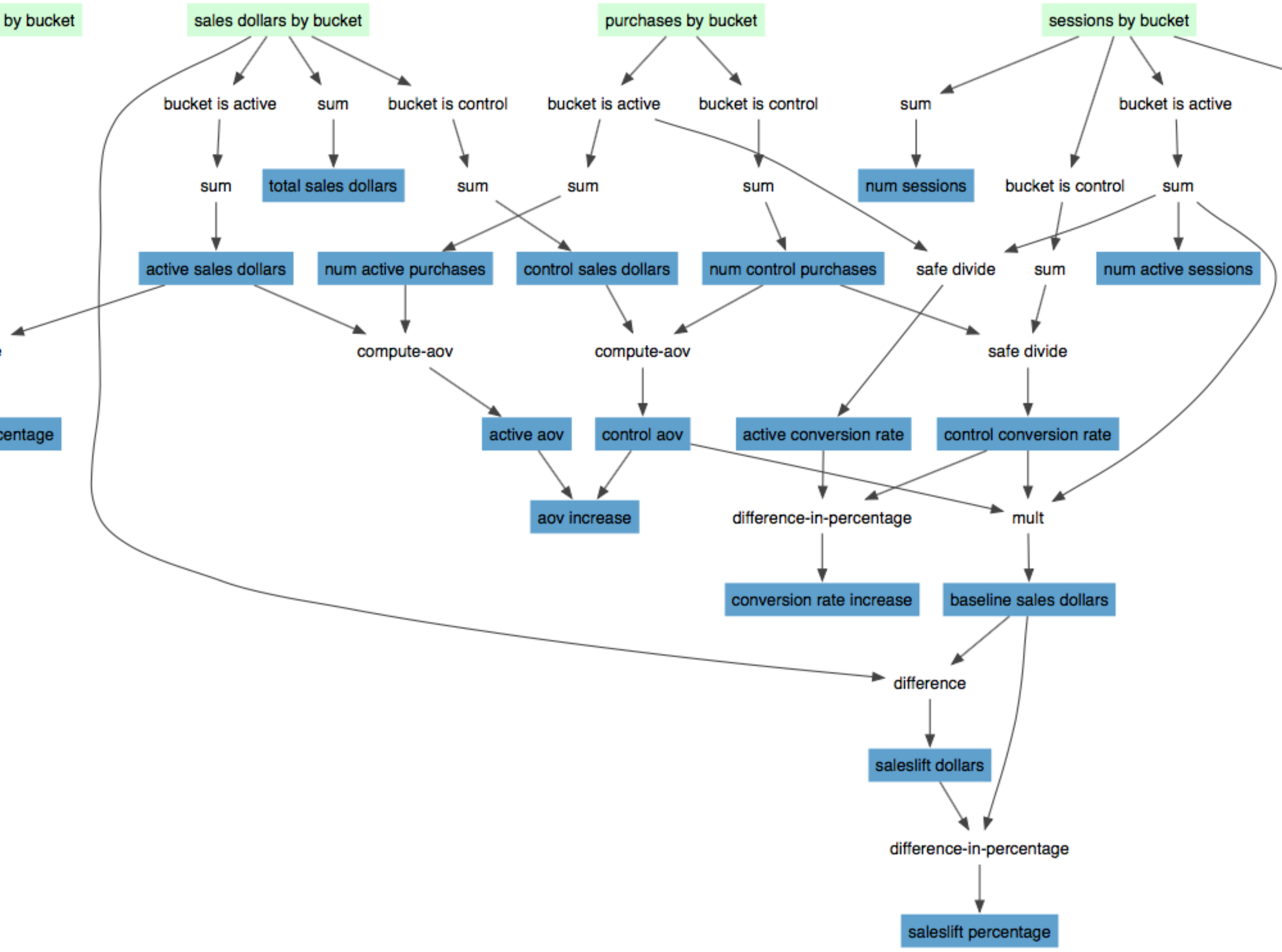
- reporting on historical data (older than 6 months)
- reporting across multiple systems' data
- tried greenplum
  - loading, reporting was pretty fast
  - has a 'merge'/upsert strategy for loading data
  - not hosted, high ops cost
- leveraged existing ETL service built for Redshift
  - assembly line built using core.async

```clojure
(defn start-processes []
  (let [cores (.availableProcessors (Runtime/getRuntime))
        srvrs (config/redshift-cluster-size)
        quota (config/redshift-quota-size)
        wrtrs (* 4 srvrs)
        xtract (* cores (config/extract-core-multiplier))]
    {:extract          (pipeline-blocking xtract pe/extracting)
     :transform        (pipeline          cores  pt/transforming)
     :csv-write        (pipeline-blocking wrtrs  pc/csv-writing)
     :manifest         (pipeline-blocking 1      pm/manifest-writing)
     :s3-upload        (pipeline-blocking 10     pu/s3-uploading)
     :sql-build        (pipeline          1      sb/sql-building)
     :redshift         (pipeline-blocking quota  pr/redshifting)
     :results-split    (pipeline-blocking quota  ps/results-splitting)
     :label-results    (pipeline          cores  pl/labelling)
     :analyze-results  (pipeline          cores  pa/analyzing)
     :s3-delete        (pipeline-blocking 10     pd/s3-deleting)}))
```

why clojure?

- lets us focus on the actual problem
- expressiveness (examples ahead)
- jvm: low latency, debugging, profiling
- established language of choice among the teams
- java, scala, go, haskell, rust, c++

```clojure
(def dispatch-table
  {[c/precise c/messy]        :default
   [c/precise c/precise]      :precise
   [c/precise c/unrestricted] :unrestricted
   [c/messy   c/messy]        :messy
   [c/messy   c/precise]      :default
   [c/messy   c/unrestricted] :unrestricted})
```


by bucket
sales dollars by bucket
purchases by bucket
sessions by bucket
bucket is active
sum
bucket is control
bucket is active
bucket is control
sum
bucket is active
sum
total sales dollars
sum
sum
sum
num sessions
bucket is control
sum
active sales dollars
num active purchases
control sales dollars
num control purchases
safe divide
sum
num active sessions
centage
compute-aov
compute-aov
safe divide
active aov
control aov
active conversion rate
control conversion rate
aov increase
difference-in-percentage
mult
conversion rate increase
baseline sales dollars
difference
saleslift dollars
difference-in-percentage
saleslift percentage

```clojure
(defn relative-graph [{:keys [active control]}]
  (m/let-map
   :active-bucket              (select-keys active bucket-stats-keys)
   :control-bucket             (select-keys control bucket-stats-keys)
   :aov-cents-increase         (-' (:aov-cents active-bucket) (:aov-cents control-bucket))
   :aov-increase-%             (u/increase-percentage (:aov-cents control-bucket)
                                                      (:aov-cents active-bucket))
   :baseline-cents             (*' (:num-managed active-bucket)
                                   (:conversion-% control-bucket)
                                   (:aov-cents control-bucket))
   :saleslift-cents            (-' (:sales-cents active-bucket) baseline-cents)
   :saleslift-%                (safe-divide saleslift-cents baseline-cents)
   :baseline-margin            (*' (safe-divide (:margin-cents control-bucket)
                                                (:num-managed control-bucket))
                                   (:num-managed active-bucket))
   :marginlift-cents           (-' baseline-margin (:margin-cents active-bucket))
   :marginlift-%               (safe-divide marginlift-cents baseline-margin)
   :conversion-confidence-%    (binary-confidence-level (:num-orders active-bucket)
                                                        (:num-managed active-bucket)
                                                        (:num-orders control-bucket)
                                                        (:num-managed control-bucket))))
```

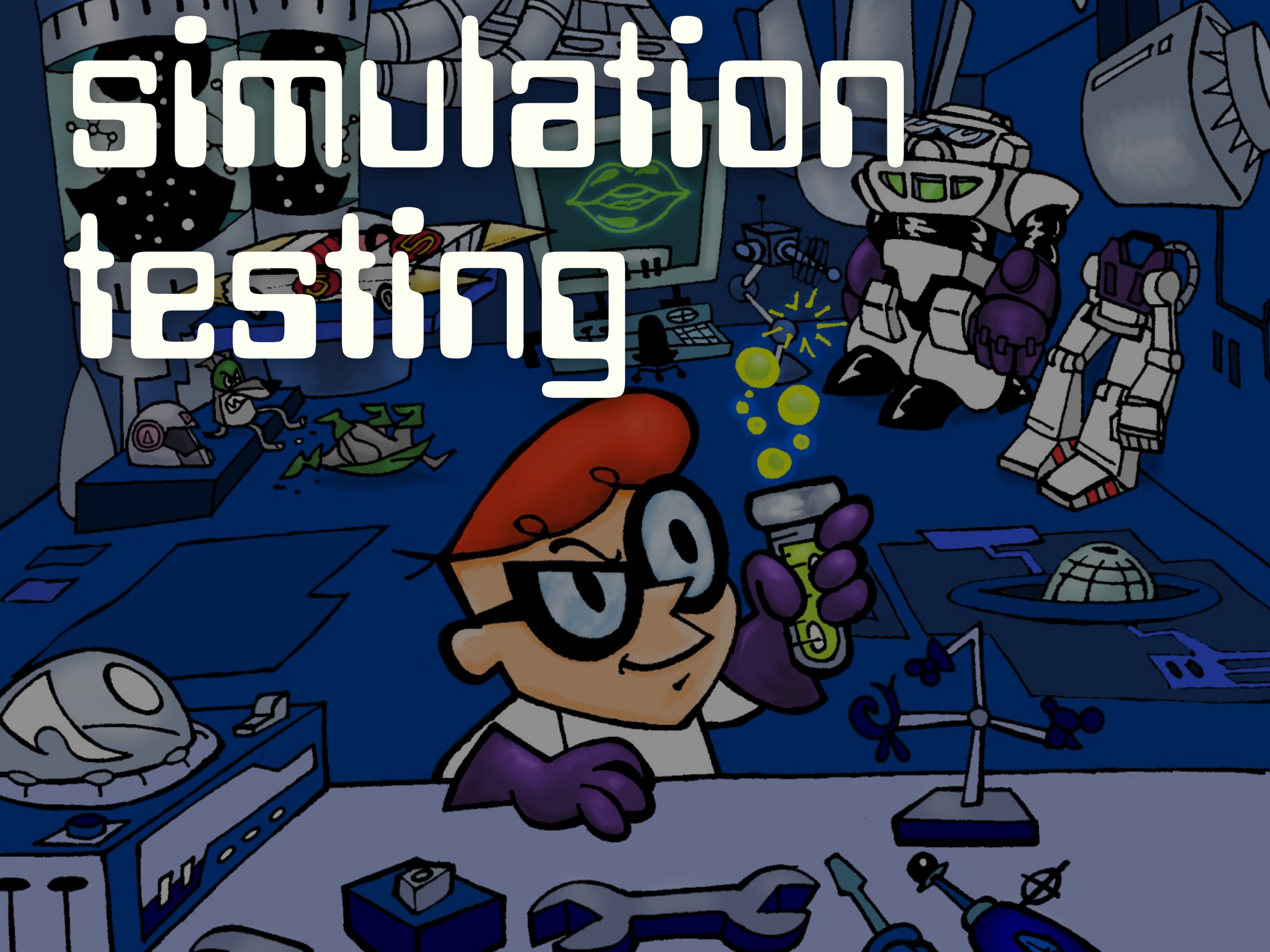
```clojure
(defn edges
  [& {:keys [multi-session-odds] :or {multi-session-odds 10}}]
  ;;start-state          end-state                 weight  min-d   max-d
  [[:start               :token-acquired              100  5         10]
   [:token-acquired      :idle                        100  200      300]
   [:idle                :user-info-provided           10  5       5000]
   [:idle                :collecting-info              10  5       5000]
   [:idle                :displaying-promo             10  5       5000]
   [:idle                :purchase-made                15  5       5000]
   [:idle                :buckets-identified           50  5       5000]
   [:idle                :session-end                  05  5       5000]
   [:user-info-provided  :idle                        90  5       5000]
   [:user-info-provided  :session-end                 10  5       5000]
   [:collecting-info     :idle                        90  5       5000]
   [:collecting-info     :session-end                 10  5       5000]
   [:displaying-promo    :idle                        90  5       5000]
   [:displaying-promo    :session-end                 10  5       5000]
   [:purchase-made       :session-end                100  5       5000]
   [:buckets-identified  :buckets-confirmed          100  5         50]
   [:buckets-confirmed   :idle                       100  5         50]
   [:session-end         :start        multi-session-odds  5       5000]
   [:session-end         :halt (- 100 multi-session-odds)  5       5000]])
```

परन्तु

realize your lazy seqs!

```
commit c5730986e64bd4c56cbe3b7c390e539bdcfb202b
diff --git a/src/eccentrica/.../store.clj b/src/eccentrica/.../store.clj
index f9ddb82..793dfc6 100644
--- a/src/eccentrica/.../store.clj
+++ b/src/eccentrica/.../store.clj
@@ -28,7 +28,7 @@
-                          :buckets (map
+                          :buckets (mapv
```

simulation testing

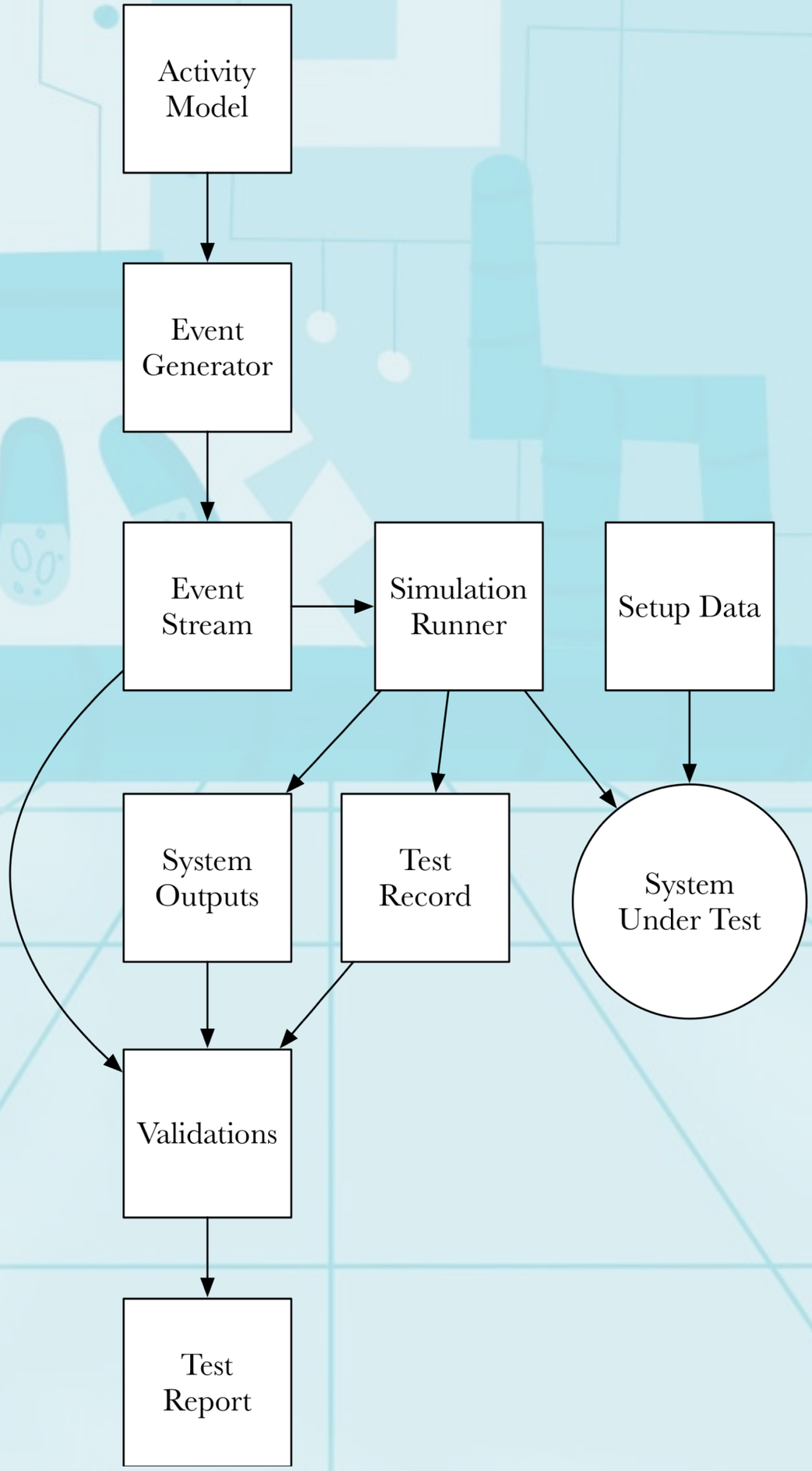


why

- top of the test pyramid
- generating confidence that your system will behave as expected during runtime
- humans can't possibly think of all the test cases
- simulation testing is the extension of property based testing to whole systems
- testing a system or a collection of systems as a whole

tools

- **simulant** - library and schema for developing simulation-based tests
- **causatum** - library designed to generate streams of timed events based on stochastic state machines
- **datomic** - data store


Activity Model
Event Generator
Event Stream
Simulation Runner
Setup Data
System Outputs
Test Record
System Under Test
Validations
Test Report

state machine to create streams of actions

```
(defn edges
  [& {:keys [multi-session-odds] :or {multi-session-odds 10}}]
  ;;start-state              end-state                  weight  min-d   max-d
  [[:start                   :token-acquired               100  5         10]
   [:token-acquired          :idle                         100  200      300]
   [:idle                    :user-info-provided            10  5       5000]
   [:idle                    :collecting-info               10  5       5000]
   [:idle                    :displaying-promo              10  5       5000]
   [:idle                    :purchase-made                 15  5       5000]
   [:idle                    :buckets-identified            50  5       5000]
   [:idle                    :session-end                   05  5       5000]
   [:user-info-provided      :idle                          90  5       5000]
   [:user-info-provided      :session-end                   10  5       5000]
   [:collecting-info         :idle                          90  5       5000]
   [:collecting-info         :session-end                   10  5       5000]
   [:displaying-promo        :idle                          90  5       5000]
   [:displaying-promo        :session-end                   10  5       5000]
   [:purchase-made           :session-end                  100  5       5000]
   [:buckets-identified      :buckets-confirmed            100  5         50]
   [:buckets-confirmed       :idle                         100  5         50]
   [:session-end             :start          multi-session-odds  5       5000]
   [:session-end             :halt (- 100 multi-session-odds)    5       5000]])
```

run the simulation, record the data

```
(defmethod sim/perform-action :action.type/getTreatments
  [action process]
  (let [sim              (-> process :sim/_processes only)
        agent            (-> action :agent/_actions solo)
        test             (-> agent :test/_agents solo)
        ...
        request-time (java.util.Date.)]
    (try
      (let [response (get-treatments target-host
                                     user-token
                                     experiment-logical-names)]
        (record request-time action process response))
      (catch Exception e
        (record-error request-time action process e)))))
```

setting up and teardown of target system

```clojure
(defn setup
  [ctx]
  (-> ctx
      generate-domain-entities
      record-domain-entities
      config/set-feature-bits
      config/set-rules))

(defn teardown
  [ctx]
  (-> ctx
      retrieve-reports
      finalize-merchants-in-ep
      finalize-runs-in-ep
      ep/reset-session))
```

validate the recorded data

```
(defn no-exceptions-were-thrown [db sim]
  (for [[log action extype exmessage]
        (d/q '[:find ?log ?action ?extype ?exmessage
               :in $ ?sim
               :where
               [?log :actionLog/sim ?sim]
               [?log :actionLog/action ?action]
               [?log :actionLog/exceptions ?ex]
               [?ex  :exception/type ?extype]
               [?ex  :exception/message ?exmessage]]
             db (su/e sim))]
    {:action-log-id log
     :validation-type :no-exceptions-were-thrown
     :action action
     :exmessage exmessage}))
```

examples of validations

- are all our requests are returning non-500 responses under the given SLA.
- invalidity checks for sessions, like no conflicting treatments were assigned
- traffic distribution
- the reports match

running diagnostics

- all the data is recorded
- you can create a timeline for a specific session from the data recorded for diagnostics purposes

```
(chesterfield.ep.diagnostics/print-timeline-for-token db sim token)
```

```
    action/clientRequestId: 536a358b-7832-4e5b-941f-5631963357e8
               action/type: action.type/getUserToken
             action/atTime: 1790
               action/type: action.type/getUserToken
  actionLog/contentType: application/x-protobuf
         actionLog/nsec: 8435000
 actionLog/responseCode: 200
actionLog/DeviceTrackerId: f26ea457-4fe6-820a-f7b9-3848cfef862f
  actionLog/apiResponse: SUCCESS
    actionLog/UserToken: 62344bd9-3f00-4360-9bf5-f8f4b284ba4c

               action/type: action.type/getBuckets
             action/atTime: 2223
               action/type: action.type/getBuckets
  actionLog/contentType: application/x-protobuf
         actionLog/nsec: 19597000
 actionLog/responseCode: 200
actionLog/DeviceTrackerId: f26ea457-4fe6-820a-f7b9-3848cfef862f
   :actionLog/experiments: 76b20010-c318-5754-c86c-400eff88a1e3/messy-5
                           76b20010-c318-5754-c86c-400eff88a1e3/precise-6
              merchant/id: 76b20010-c318-5754-c86c-400eff88a1e3
  actionLog/apiResponse: INVALID_REQUEST
    actionLog/UserToken: 62344bd9-3f00-4360-9bf5-f8f4b284ba4c

               action/type: action.type/confirmBucket
             action/atTime: 2246
  actionLog/exceptions: #{{:db/id 496979255773638}}
               action/type: action.type/confirmBucket
             actionLog/nsec: 0
```

परन्तु

- requires dedicated time and effort
- was difficult to for us to put into CI
- many moving parts

conclusions

- traffic is precious, take it account when you are designing your experiments
- ETL as assembly line work amazingly well
- test your system from the outside
- use simulation testing
- use clojure ;)

Great Material on Experiment Infrastructure

- Overlapping Experiment Infrastructure
  - More, Better, Faster Experimentation (Google)
- A/B testing @ Internet Scale
  - LinkedIn, Bing, Google
- Controlled experiments on the web
  - survey and practical guide
- D. Cox and N. Reid
  - The theory of the design of experiments, 2000
- Netflix Experimentation Platform
- Online Experimentation at Microsoft
- Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO (Microsoft)

nilenso

@nid90

@sriharisriraman