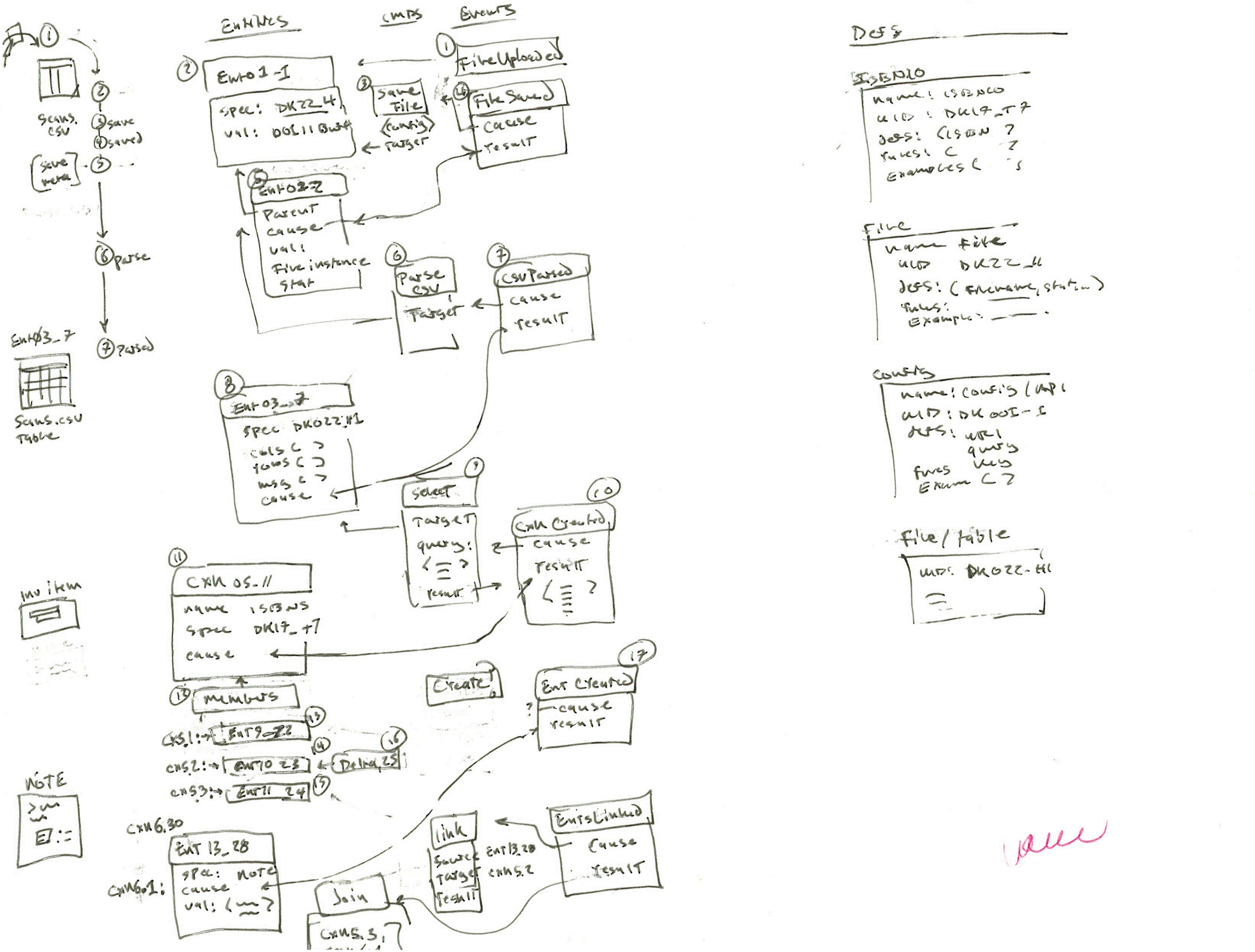
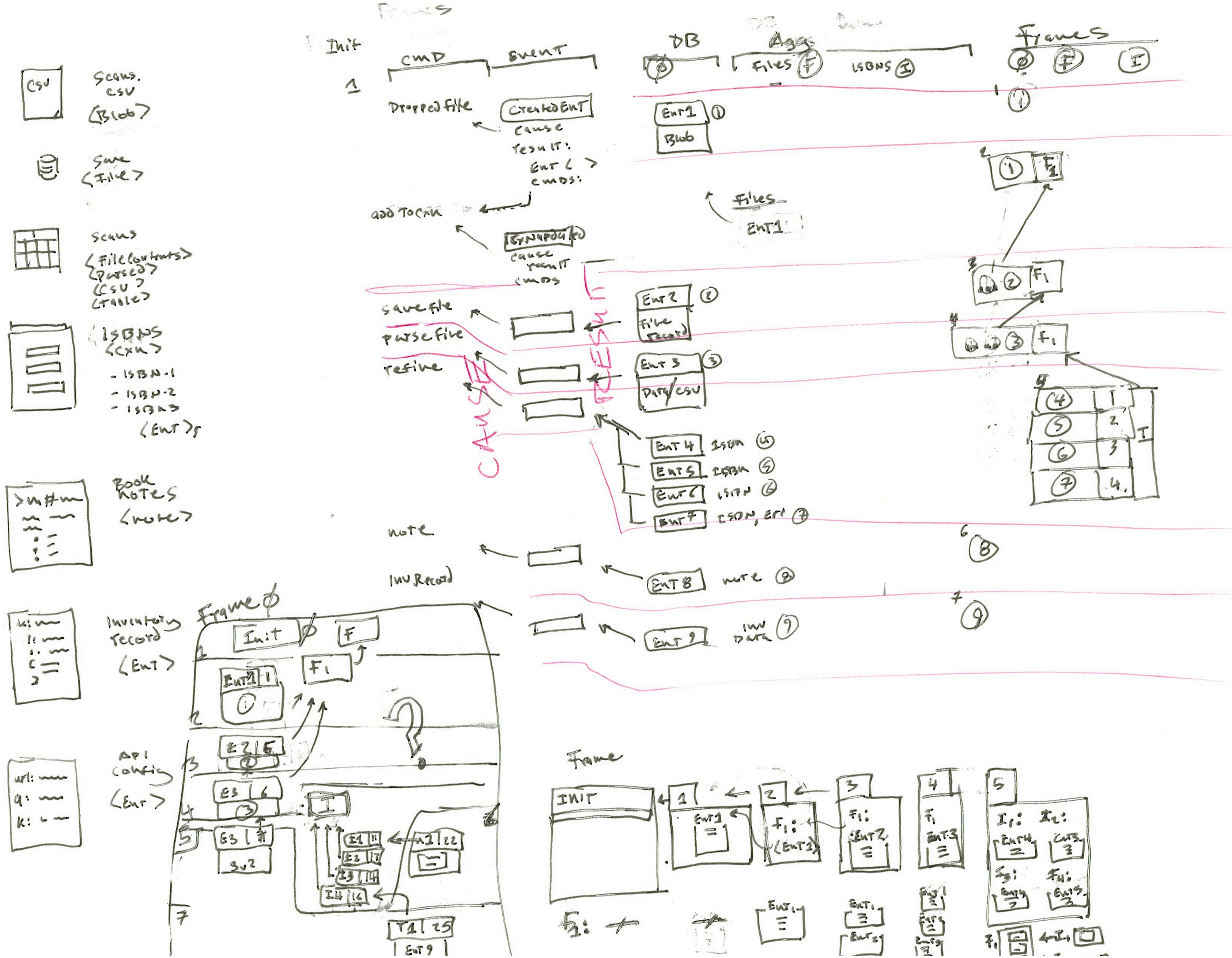


Entities
Cmds
Events
FileUploaded
Ent01_1
spec: DK22_4
val:
save File
config
target
FileSaved
cause
result
scans.csv
save
saved
save meta
Ent02_2
parent
cause
val:
File instance
stat
parse
parsed
Parse csv
target
CsvParsed
cause
result
Ent03_7
scans.csv table
Ent03_7
spec DK022#1
cols < >
rows < >
msg < >
cause
Select
target
query:
result
CxnCreated
cause
result
Cxn05_11
name ISBNs
spec DK17_t7
cause
Members
Ent9_22
Ent10_23
Ent11_24
Create
EntCreated
cause
result
NOTE
Ent 13_28
spec: NOTE
cause
val:
link
source
target
result
Ent13.28
cxn5.2
EntsLinked
cause
result
Join
Defs
ISBN10
name: ISBN10
uid: DK17_T7
defs:
rules:
examples
File
name file
uid DK22_4
defs: (filename, status)
rules:
Example:
Config
name: config
uid:
defs: url
query
keys
rules
Exam < >
File/table
uid: DK022-H1


Init
CMD
Event
DB
Aggs
Files
ISBNS
Frames
CSV
Scans.csv
<Blob>
Save
<File>
Scans
<FileContents>
<parsed>
<CSV>
<Table>
ISBNS
<CXN>
- ISBN-1
- ISBN-2
- ISBN-3
Book notes
<note>
Inventory record
<Ent>
API config
<Ent>
Dropped file
CreatedEnt
cause
result:
Ent < >
cmds:
add to CXN
cause
result
cmds
save file
parse file
refine
note
Inv Record
CAUSE
RESULT
Ent1
Blob
Ent2
File record
Ent3
Data/csv
Ent4
Ent5
Ent6
Ent7
Ent8
note
Ent9
Inv Data
Frame Ø
Init
Frame
INIT
T

Just filter all Entity messages out of the log...

Blocks are only way to create NB "sections" in narrative

Blocks are only way to define defs (via outs)

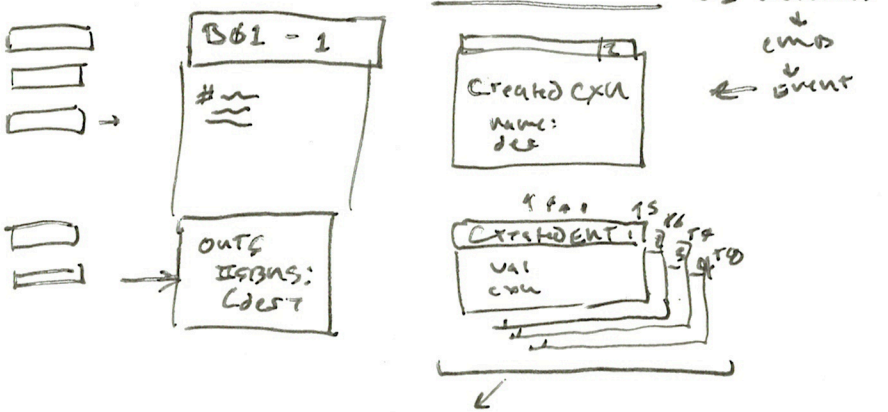
Blocks

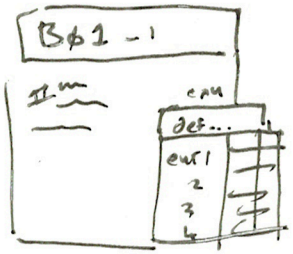
Design Events

main/default Branch

Instance Events

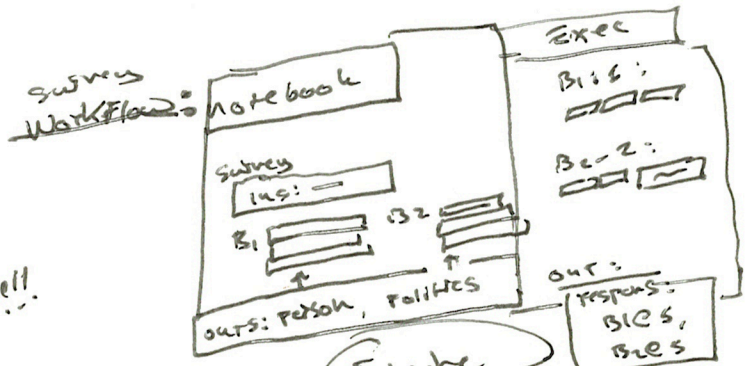
UI Actions → cmd → Event





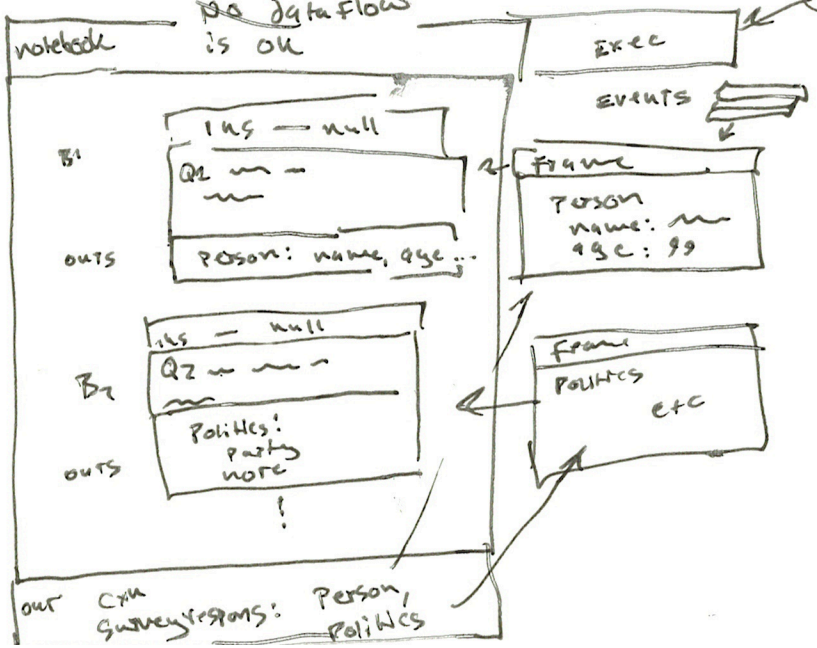
User Actions

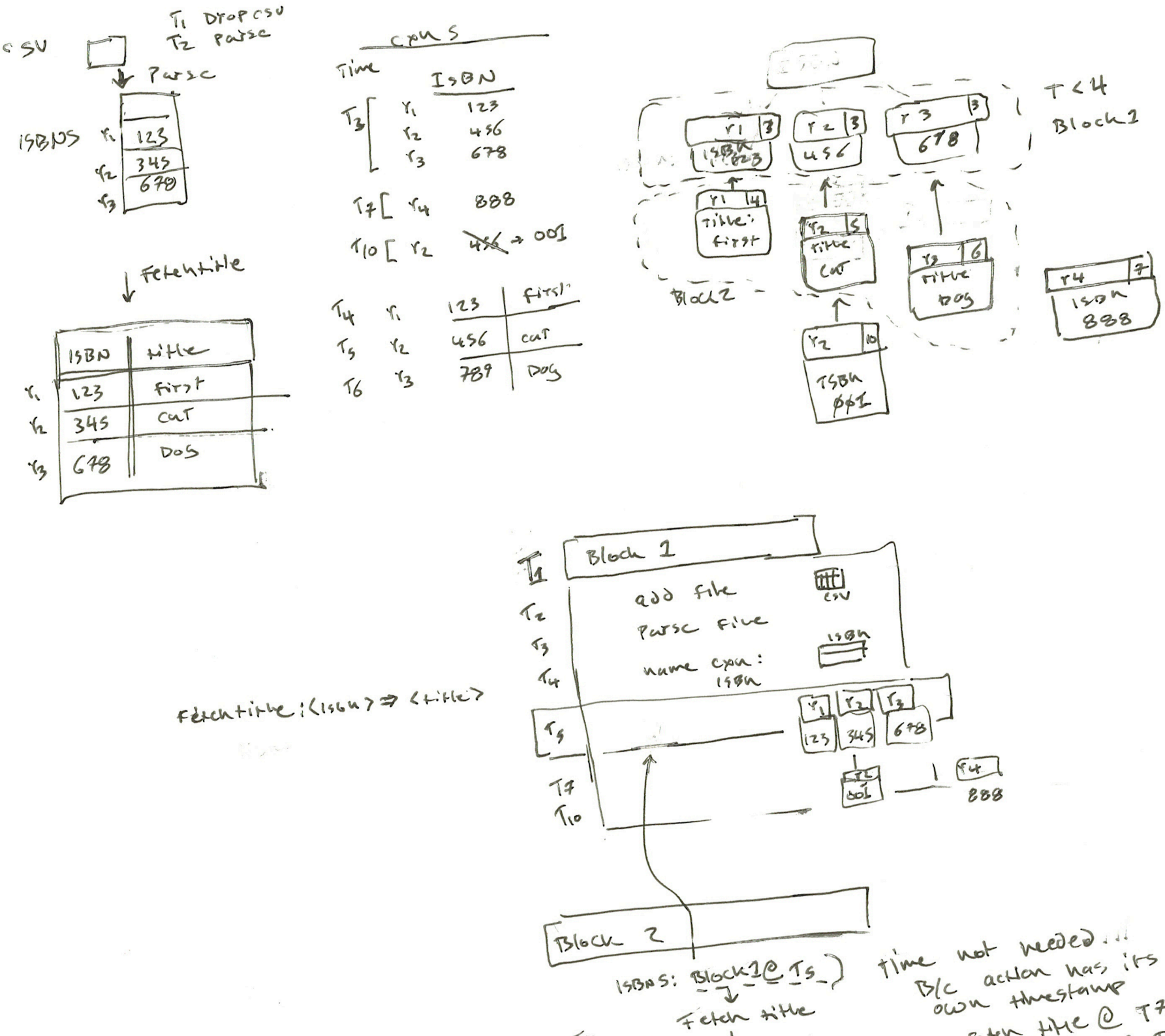
	Action	Result
- "Edit table" →	cmd →	Event
	cmd →	Event Event

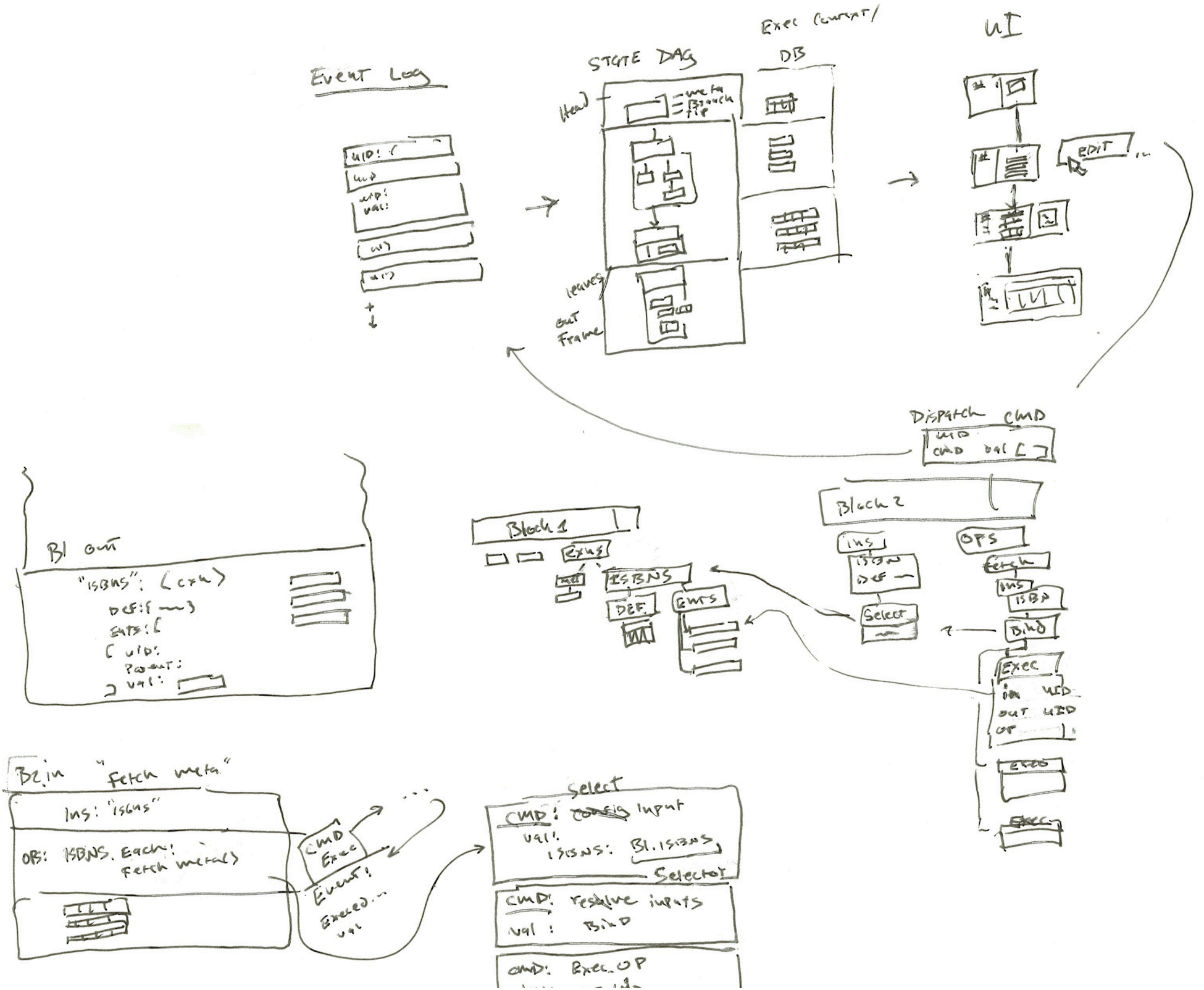


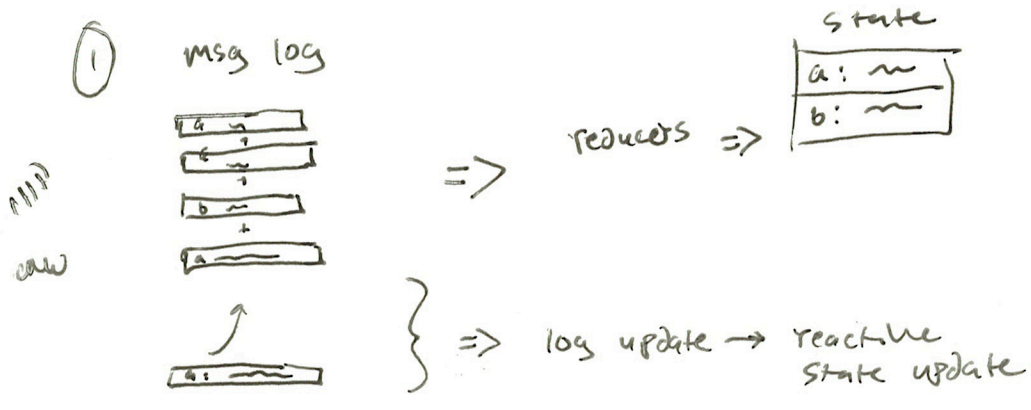
Survey NB Well!...

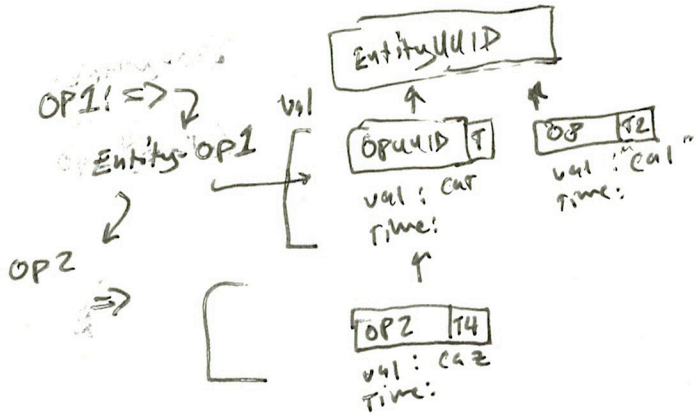
Example - no dataflow is ok

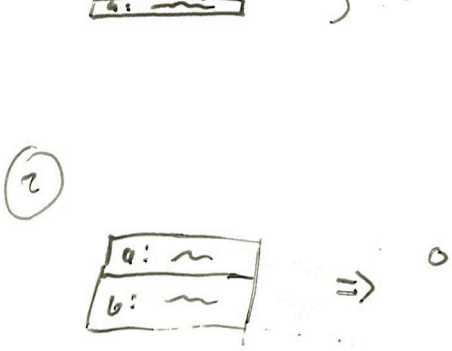


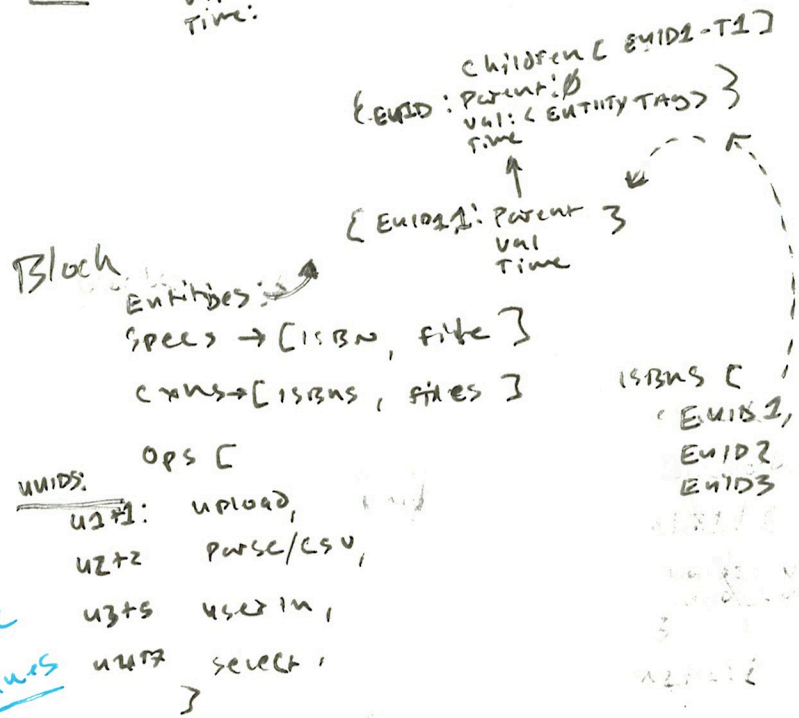

CSV
T1 Drop csv
T2 Parse
Parse
ISBNS
r1 123
r2 345
r3 678
Fetchtitle
ISBN title
r1 123 first
r2 345 cat
r3 678 dog
csvs
Time ISBN
T3 r1 123
r2 456
r3 678
T7 r4 888
T10 r2 456 → 001
T4 r1 123 first
T5 r2 456 cat
T6 r3 789 dog
ISBN
r1 3 ISBN 123
r2 3 456
r3 3 678
r1 4 title: first
r2 5 title cat
r3 6 title dog
r4 7 ISBN 888
r2 10 ISBN 001
T<4
Block1
Block2
Block 1
add file
Parse file
name csv: ISBN
csv
ISBN
r1 r2 r3
123 345 678
r4 888
T1
T2
T3
T4
T5
T7
T10
Block 2
Fetchtitle: <ISBN> ⇒ <title>
ISBNS: Block1@T5
Fetch title
time not needed...
B/c action has its own timestamp
fetch_title @ T7
means ISBNS<T7

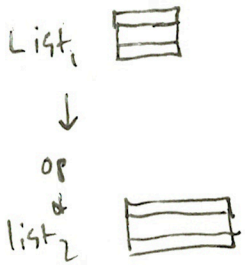

Event Log
State DAG
Exec Context/
DB
UI
meta
Branch
Head
Leaves
Out
Frame
EDIT
Dispatch CMD
Block 1
Block 2
Exns
ISBNS
DEF
Ins
OPs
Select
Fetch
Bind
Exec
Bl out
"ISBNS": <ctx>
DEF:{ ~ }
Exns:[
{ uid:
Parent:
val:
}
Bl.in "fetch meta"
Ins: "ISBNS"
OPs: ISBNS. Each!
Fetch meta()
CMD
Exec
Event!
Exec...
val
Select
CMD: Config Input
val:
ISBNS: Bl.ISBNS
Selector
CMD: resolve inputs
val: Bind
CMD: Exec_OP











Env: Sequence of Frames

Frames: Seq of Bindings

Bindings: associative name → value including Return Values

Environments

Frames:

BINDS var names to Values

Assignment statements: They change Bindings B/W names & values in Frames

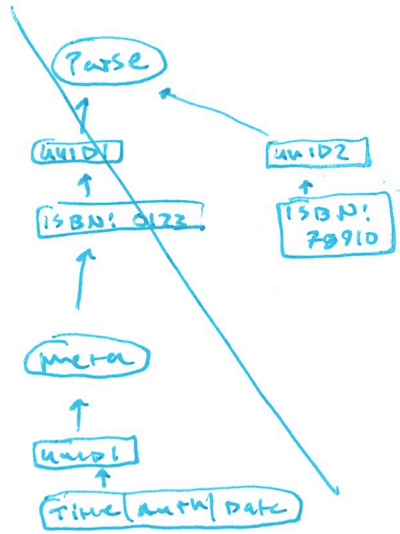
name can only be bound to 1 value in Frame!

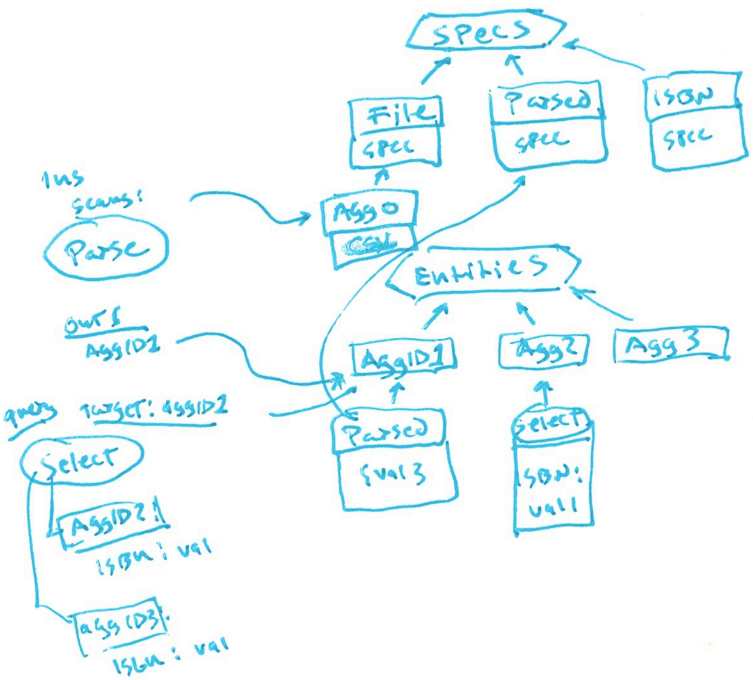
=> Function Definition is more Powerful: Binds Name to Expression

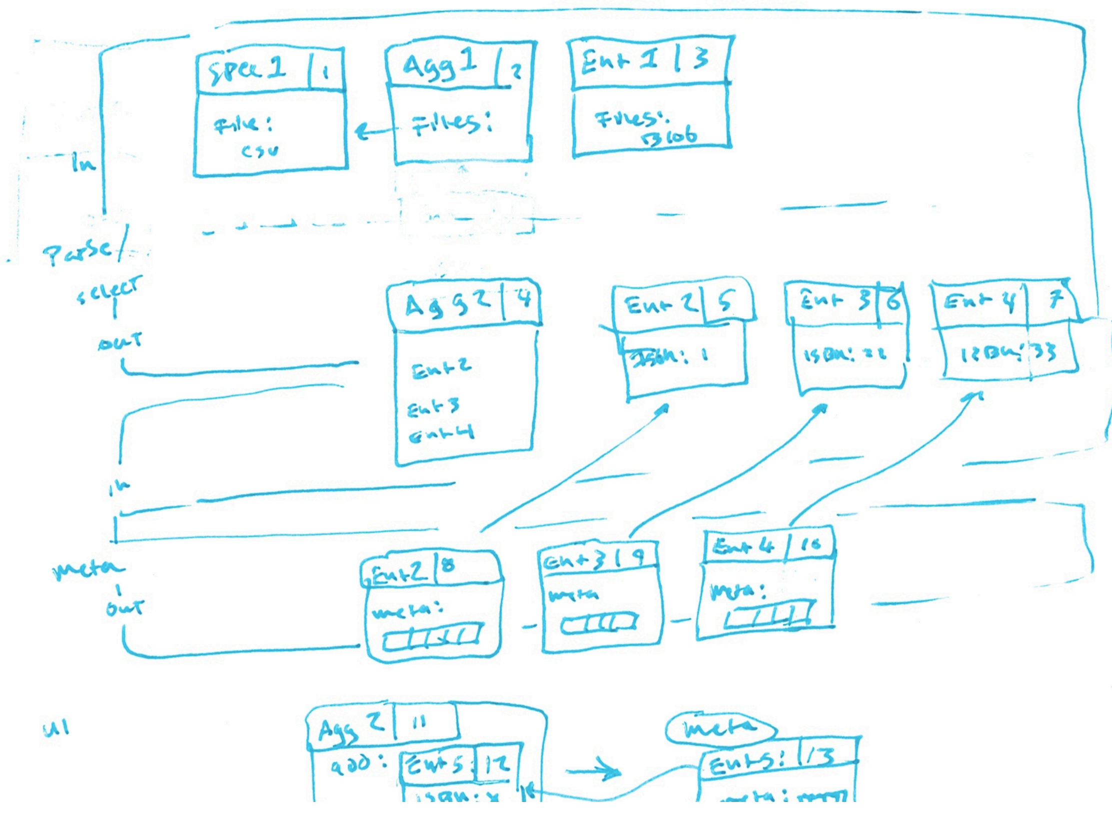
Func Signature: def # arguments

Func Body: def procedure

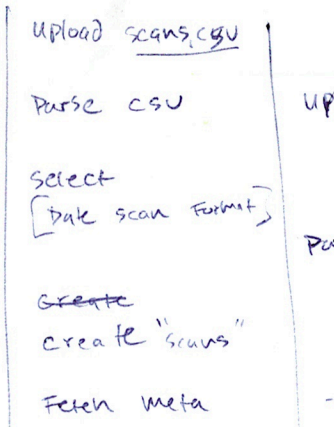
EntityID	clock	version 1	2
csv1	1	{ cols: [ ], rows: [ ] }	
uuid2	2	isbn1	
uuid3	3	isbn2	
uuid4	4	isbn3	
csv2	5	[uuid1, uuid2, uuid4]	

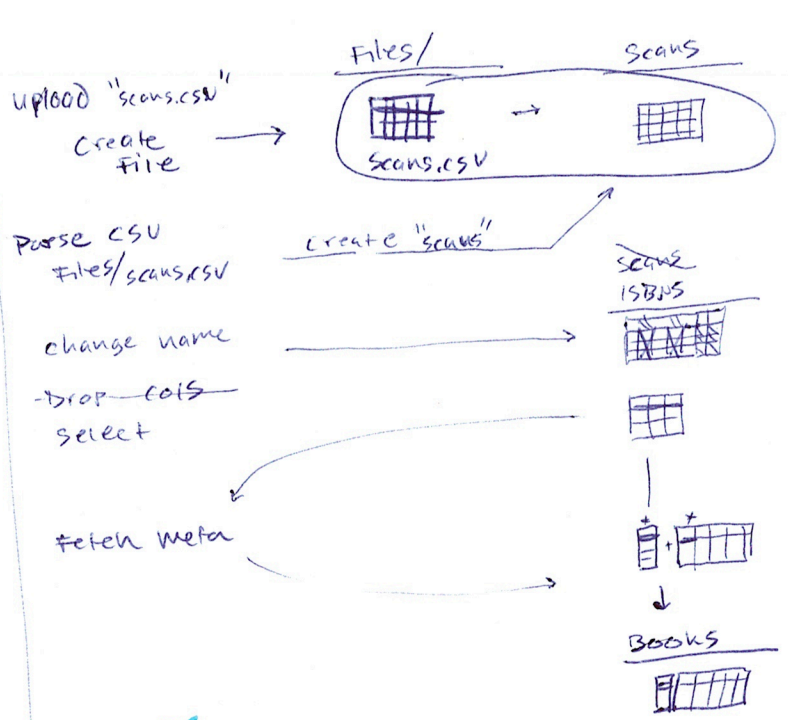


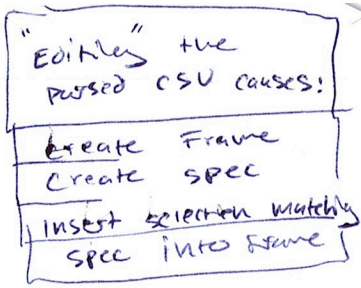


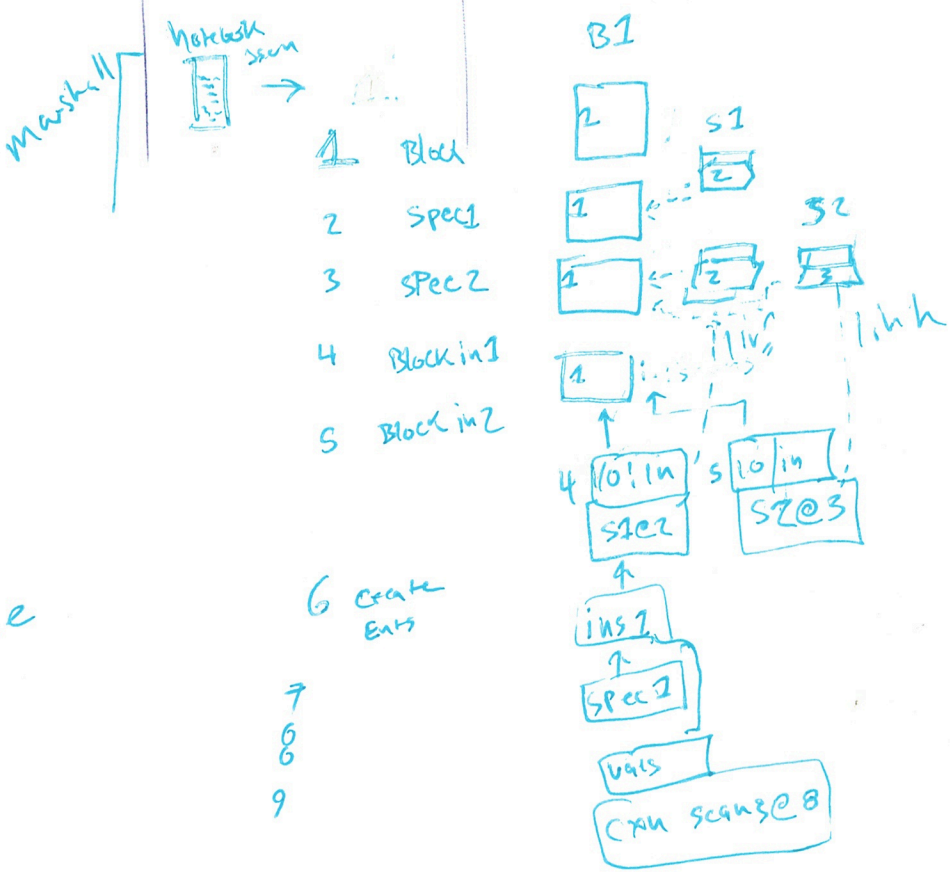


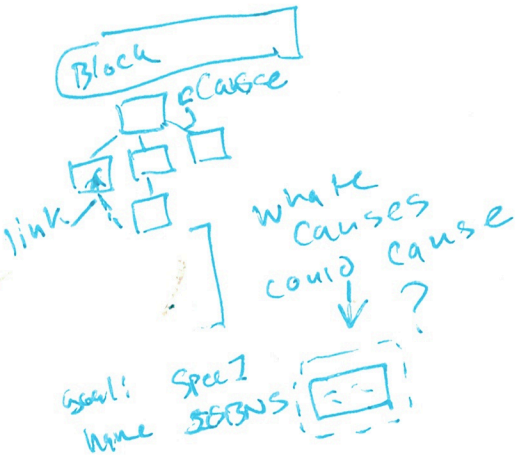

ROOT
S1@T1
~~Create~~ ~~Scans~~"
upload "scans.csv"
S1@T2
create cxn: "scans"
S1@T3
Parse CSV into "scans"
S1@T4
rename "ISBNS"
S1@T5
drop cols [qual, D3, D4]

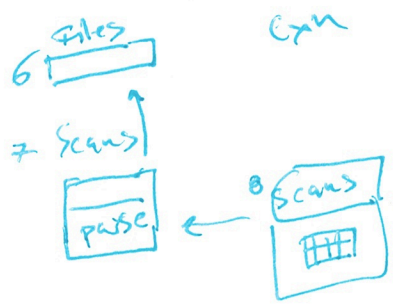

Upload scans.csv
Parse csv
Select [bak scan Format]
~~Create~~
create "scans"
Fetch meta

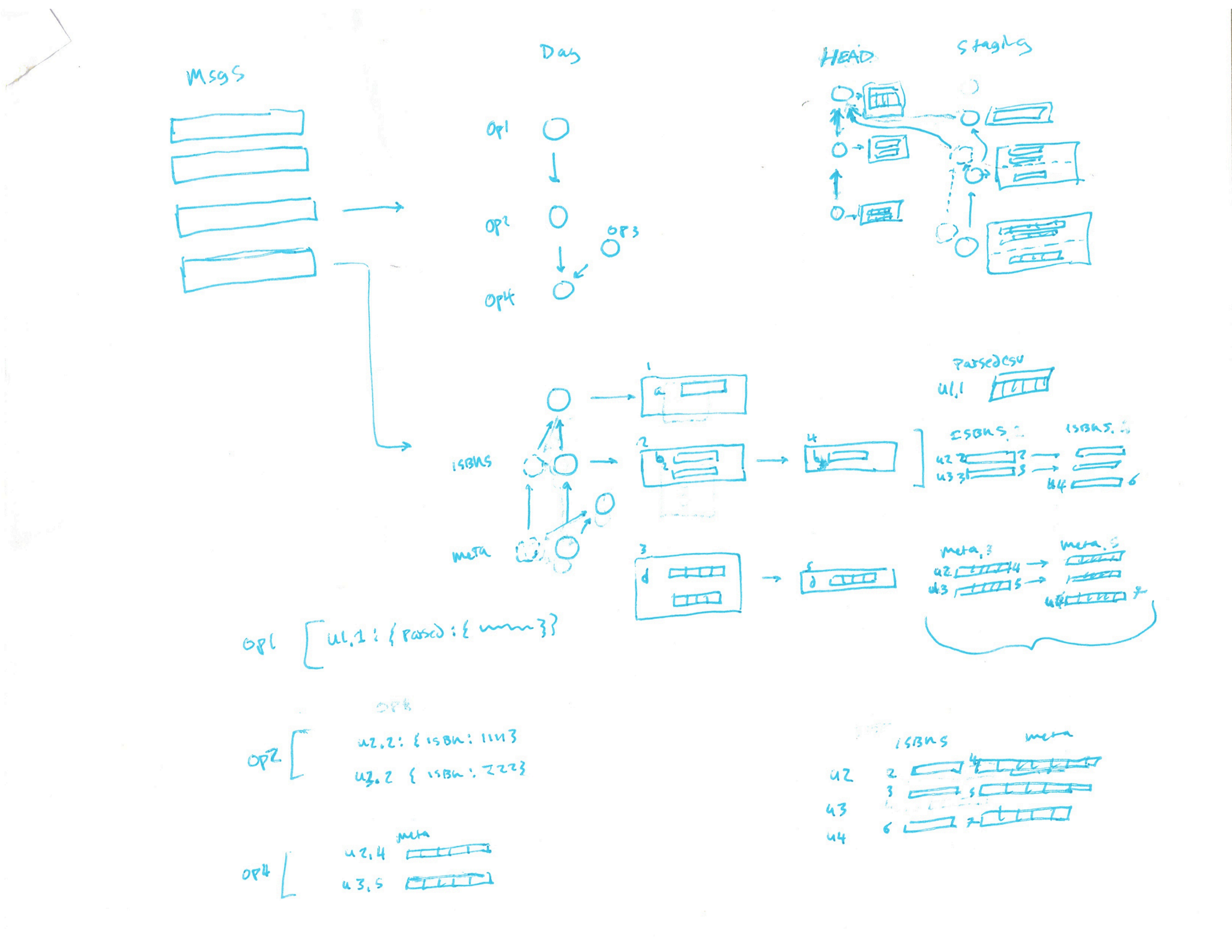

Files/
Scans
upload "scans.csv"
create file
scans.csv
Parse csv Files/scans.csv
create "scans"
change name
~~Drop cols~~
select
Fetch meta
~~Scans~~ ISBNS
Books

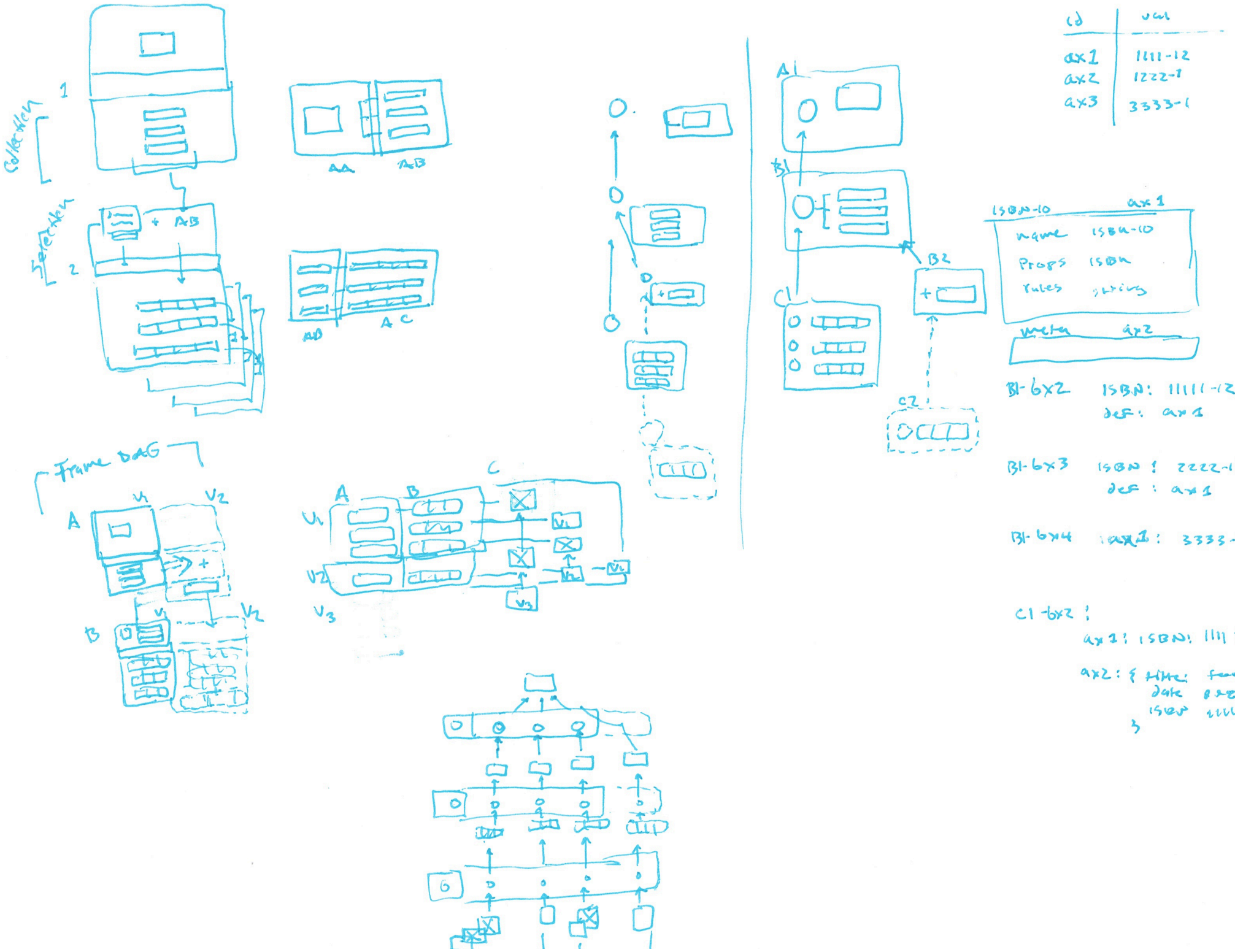

"Edit" the parsed csv causes!
~~Create~~ Frame
Create spec
Insert ~~selection~~ matching spec into frame

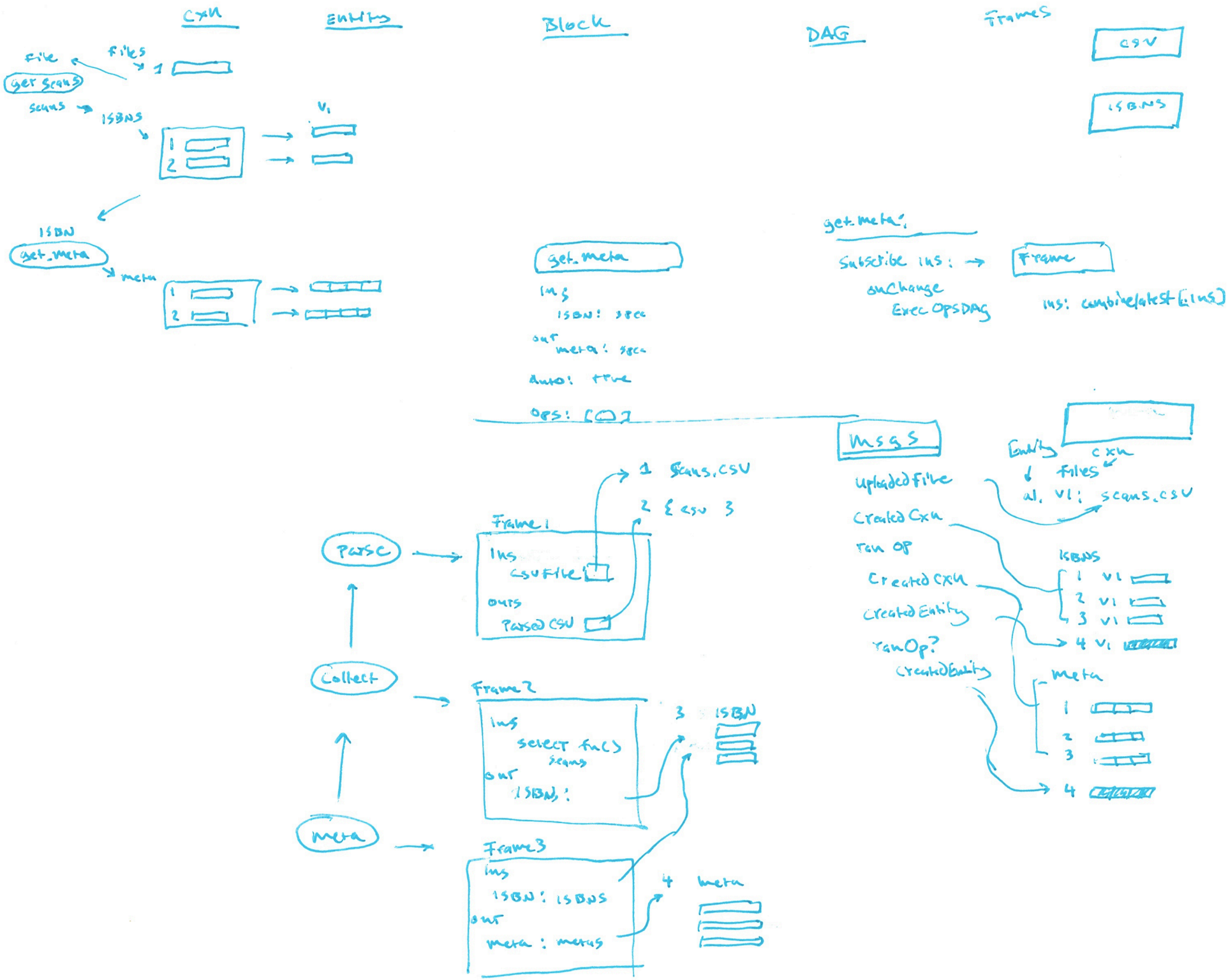

Notebook spec
1 Block
2 Spec1
3 Spec2
4 Block in 1
5 Block in 2
6 Create Exts
7
8
9
B1
S1
S2
link
4 IO! in
S1@2
5 IO in
S2@3
ins 1
Spec 1
vals
Cxn scans@8


Block
cCause
link
what causes could cause ?
Goal: Spec 1 name ISBNS


6 Files
Cxn
7 Scans
parse
8 Scans


Msgs
Dag
Op1
Op2
Op3
Op4
HEAD
Staging
ISBNs
Meta
parsedCSV
u1,1
u2
u3
u4
op1
op2
op4
u2,2: { isbn: 1113
u3,2 { isbn: 2223
u2,4
u3,5
meta
u2
u3
u4


Collection
Selection
1
2
+ AB
AA
AB
AD
A C
Frame DAG
A
B
V1
V2
V3
C
G
id
val
ax1
1111-12
ax2
1222-1
ax3
3333-1
A1
B1
B2
C1
C2
ISBN-10
ax1
name
ISBN-10
Props
ISBN
rules
syntax
ax2
B1-6x2
ISBN: 11111-12
def: ax1
B1-6x3
ISBN: 2222-11
def: ax1
B1-6x4
ax1: 3333-1
C1-6x2:
ax1: ISBN: 1111-2
ax2: { title: foo
date 0-28
isbn 1111-10
}


Cxn
Entity
Block
DAG
Frames
CSV
ISBNs
file
files
get scans
scans
ISBNs
V1
ISBN
get_meta
meta
get_meta
ins:
ISBN: spec
out
meta: spec
auto: true
ops: []
get_meta:
Subscribe ins: →
onChange
Exec OpsDAG
Frame
ins: combineLatest(ins)
Msgs
Entity
Cxn
files
a1, v1: scans.csv
UploadedFile
CreatedCxn
ranOp
CreatedCxn
CreatedEntity
RanOp?
CreatedEntity
ISBNs
1 v1
2 v1
3 v1
4 v1
meta
1
2
3
4
1 scans.csv
2 { csv }
Frame1
ins
csvFile
outs
ParsedCSV
Parse
Collect
Meta
Frame2
ins
select fn()
scans
out
ISBN:
3 ISBN
Frame3
ins
ISBN: ISBNs
out
meta: metas
4 meta

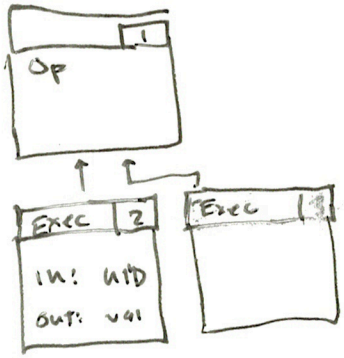
- Skim rxjs - redux docs
- Practical Microservices
- UI:

  Plan/outline flowchart

  MD / rbx blocks

  Frame of data instances

  - cmd history



Block1

make some ents

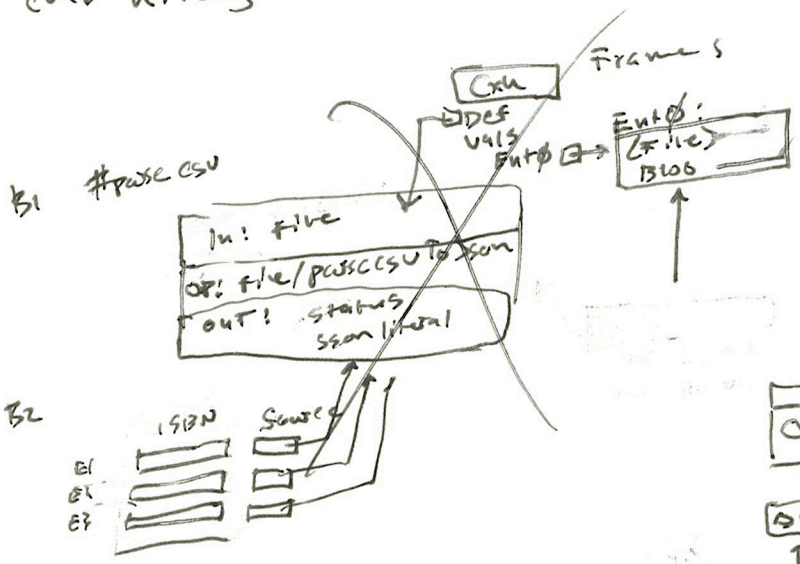
1 Desc name cols

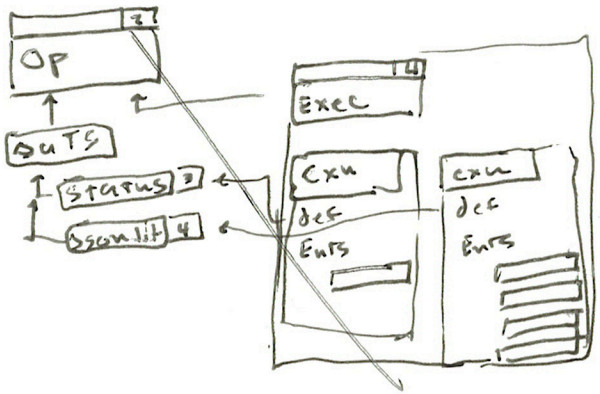
cxn

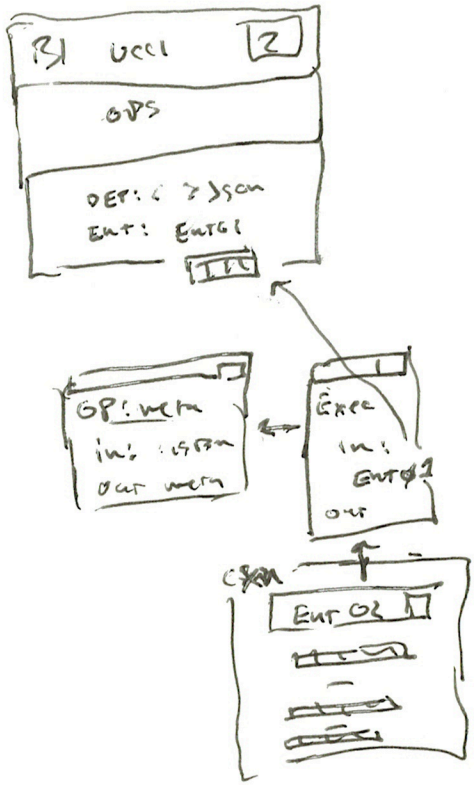
ents

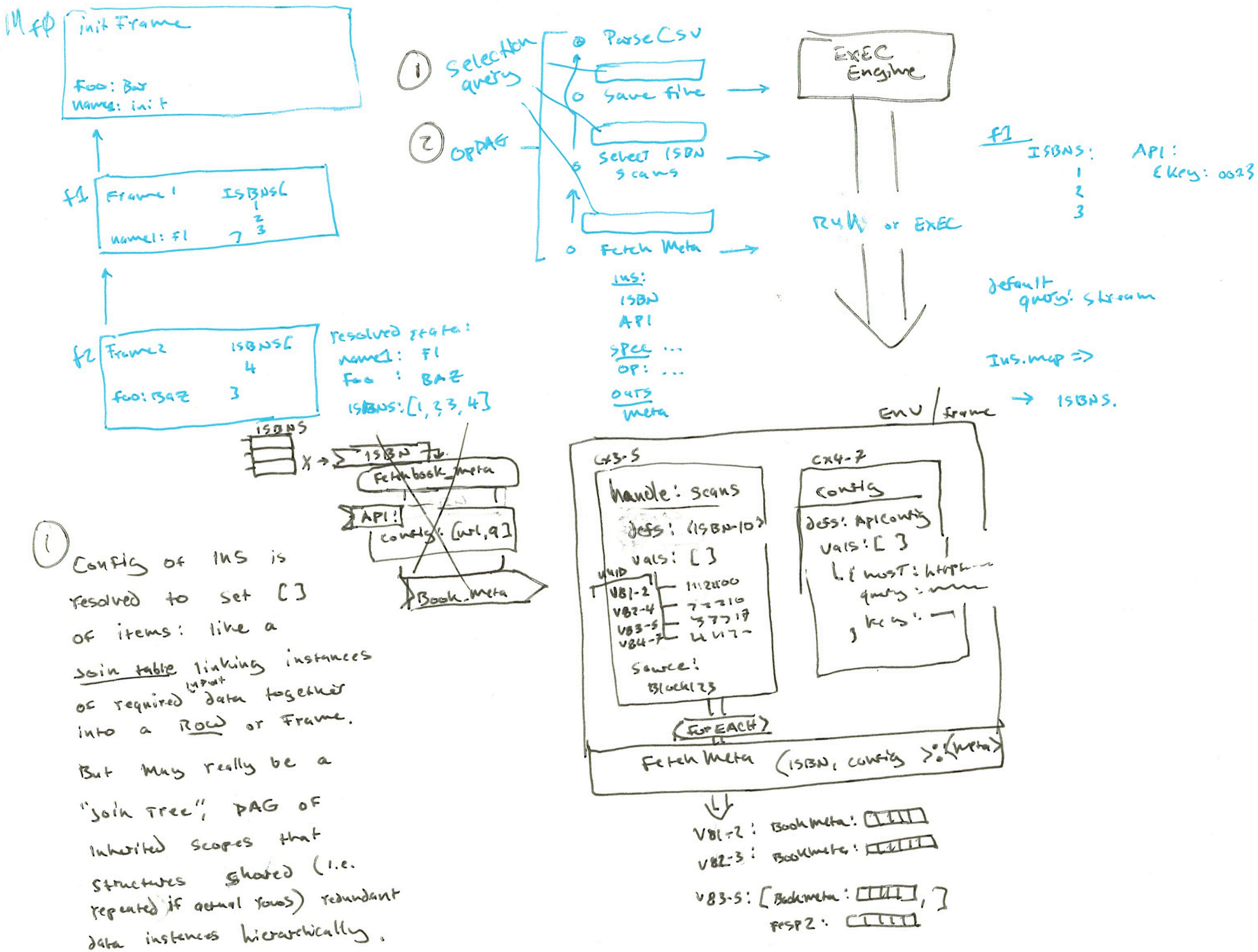
2 ⇒ add to cxn

3 Edit Ent in cxn





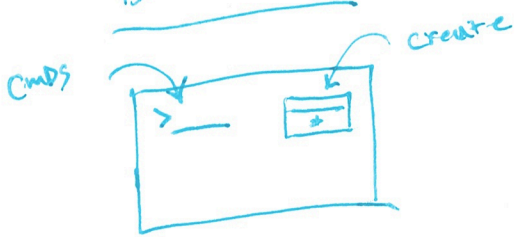




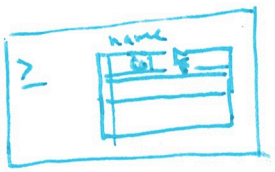
1) Config of ins is resolved to set [ ] of items: like a join table linking instances of required input data together into a Row or Frame.

But may really be a "join tree", DAG of inherited scopes that structures shared (i.e. repeated if actual rows) redundant data instances hierarchically.

Block



spec [] cxn [] Entities []



] ← CID1:
name: ?
attrib: @ {
CID2:
name: "a"
} spec: ...

CID3:
spec: CID1
meta: [UI...]
Entities: [
...
]

[⊖]

CID1:
name: "unnamed"
attrib:
CID2:
name: "tmpCol"

CID3:
Entities: [
CID4 →
]

Entities:

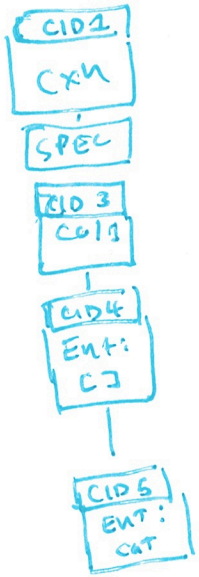
CID4:
value: {
CID1: {"CAT"},
}
Link: "CIDX"
}

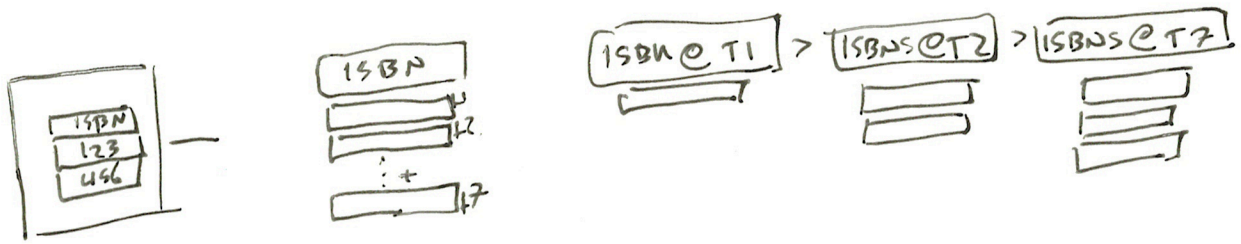
JSON type
Import

Messages [];

CMD-Block1-t1: create cxn
EVENT-Block1-t2: cxn created
[cxn; uuid: CID2]

cxn:
spec:
cols: [
CID: name
]
Ents: []





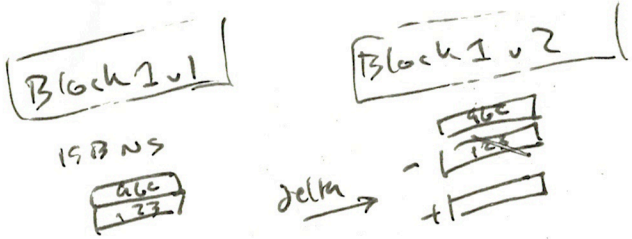
Append only

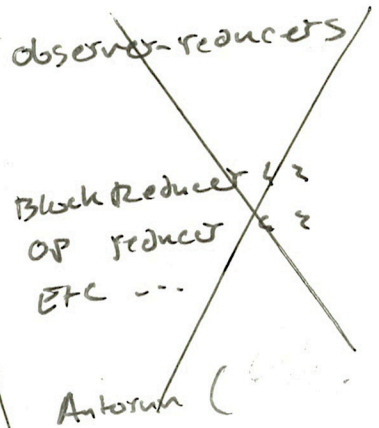
use mobx shallow or ref Atom

Observable. shallow → Log []

{key: payload}

S1-Block1-op2-T7 : {...}





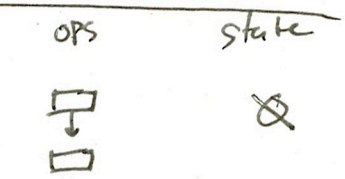
Just dispatch log msgs

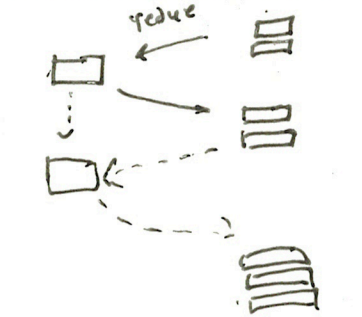
- Root reducer pushes into log store
- rest reducers process
  - Produce Aggregates (block etc
- Aggregates computed slice of log

OP Blocks linked in graph

ops reduce state graph → new state

- by reading / reducing log slice desired by inputs





ordering of ops ↔ ordering of update messages for state (slice)

Careful thought: how to make input reducer / watcher see post-op updateds

Blocks act like Transaction Boundaries around Oplogs

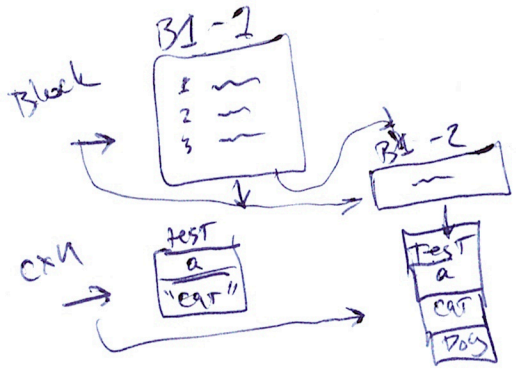
—

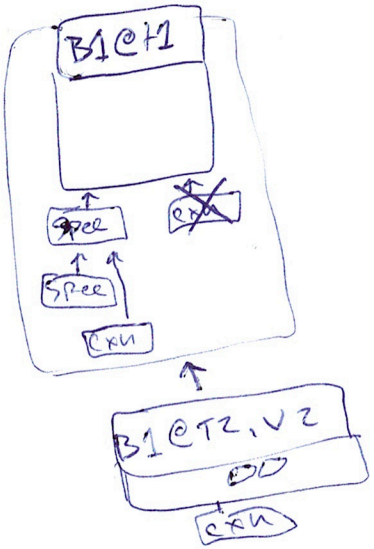
Blocks ~~are~~ produce state

Transaction ensures Blocks always can declare their ~~outs~~ spec

cxn get uuid
- Point to spec
- Point to Block
- Materialize updates
- Form future blocks

- Block uuid - user/site - time



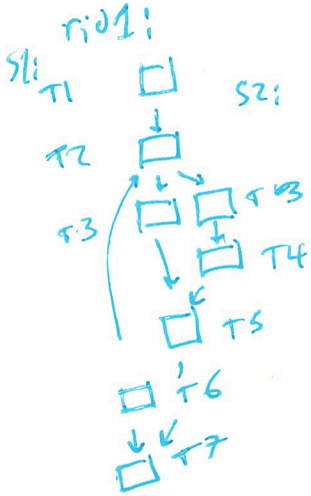


ExecDag

Tree API

- Stores CT-like DAG
- materializes set of values for all objs
- State Machine / reducer can ~~RR~~ rewind / advance / TIP(s) for materialized views & switch Branches
- Fork(at)
- merge(a, b)

cxn: nouns



Change in
~~Des~~
Data

Design

Everything gets UUID

CSV

Scans

ISBNs

DESIGN

EXECUTIONS

links

versions in time & space

specs

shape, name, UUID base for ins & outs

State

Books

Block

Spec

OP

Exec

Cxn

Data

101

Block

1-3

OP

1-5

OP

1-2

Spec

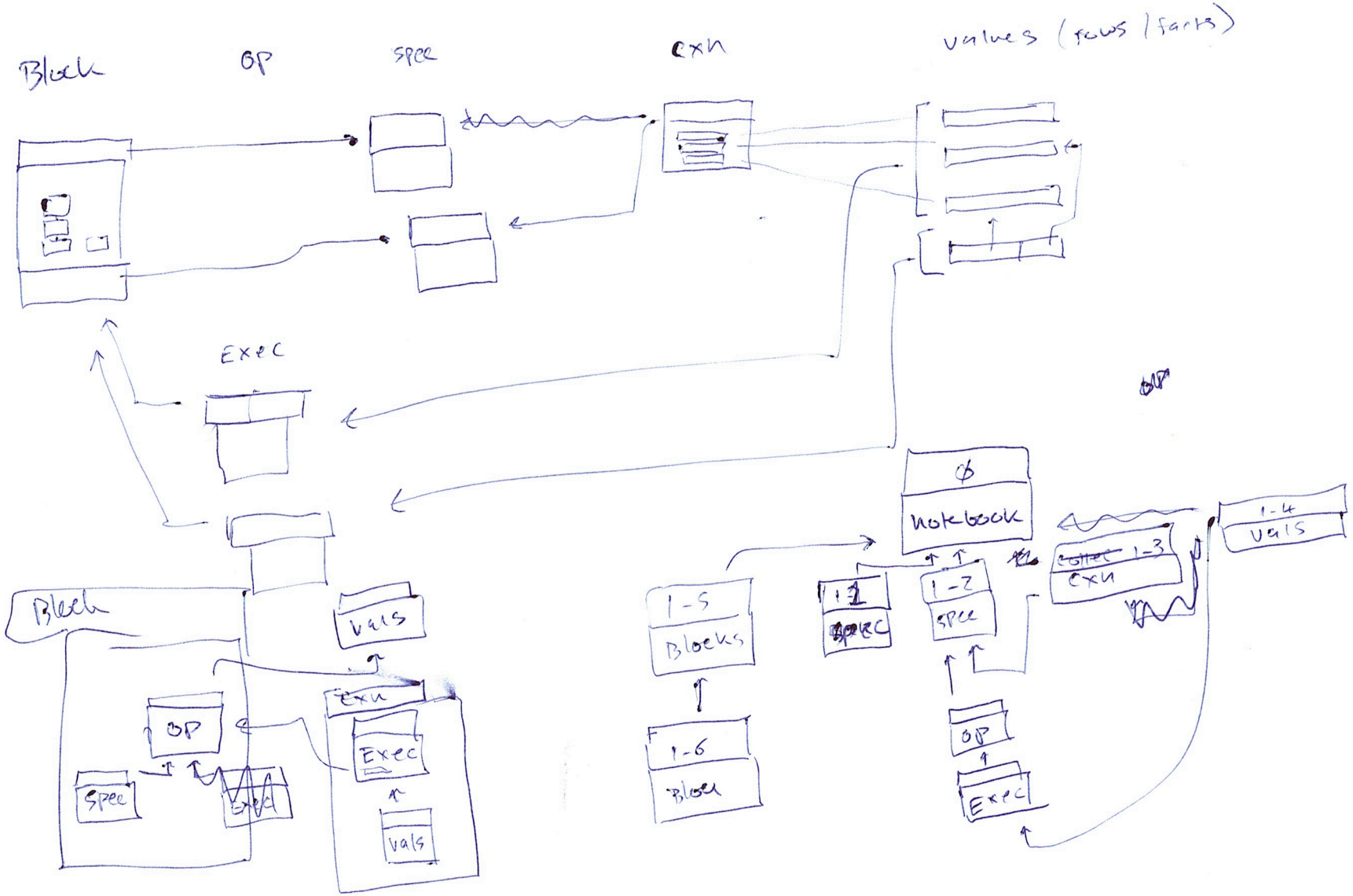
1-3

1-4

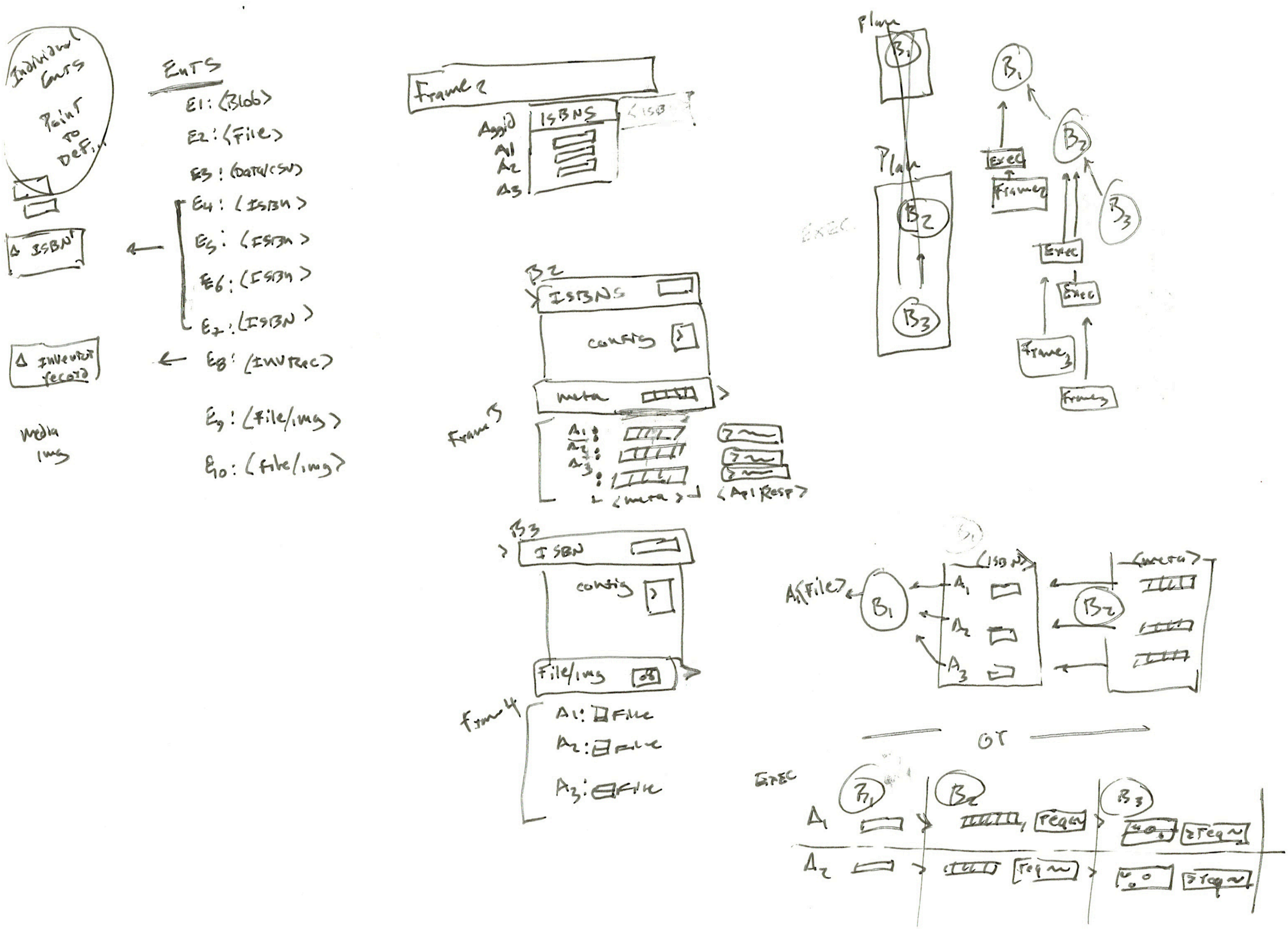
Spec

Exec

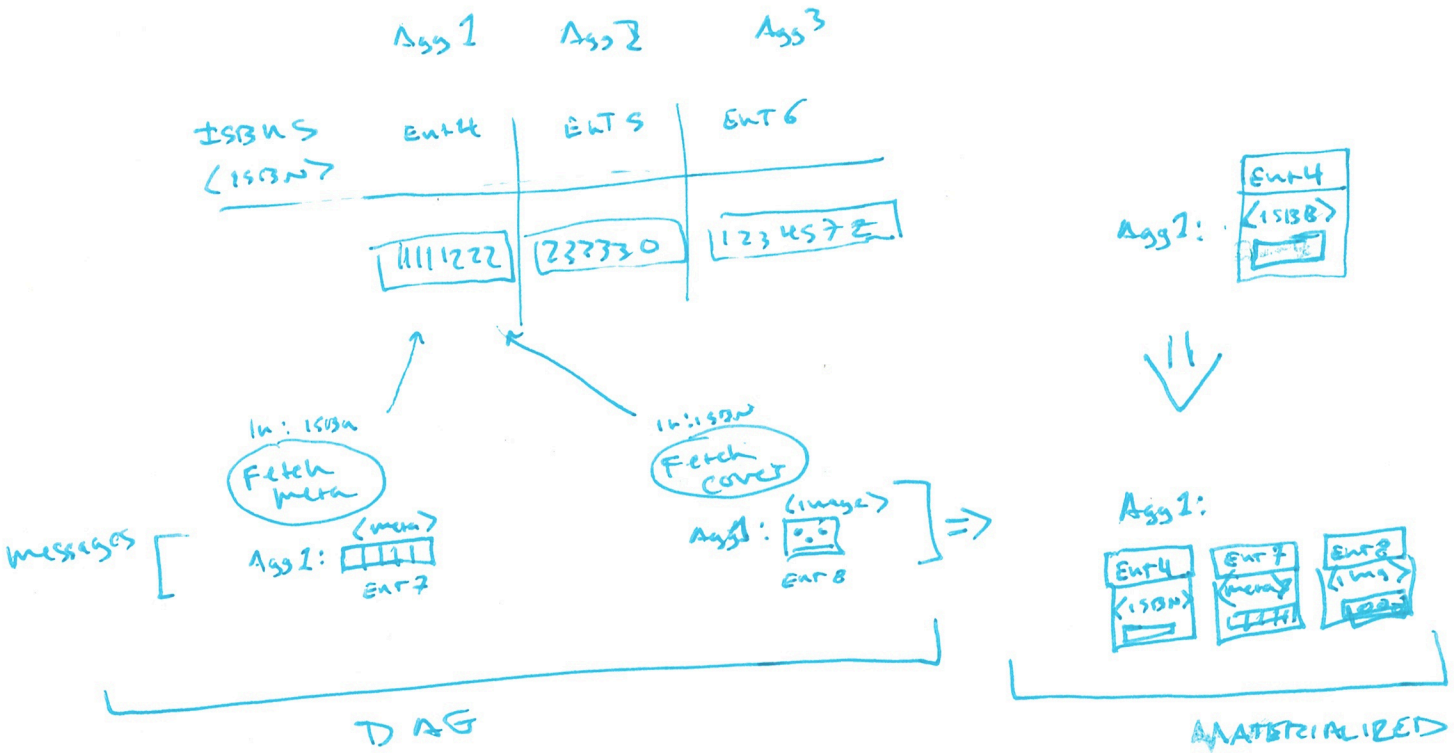
Exec

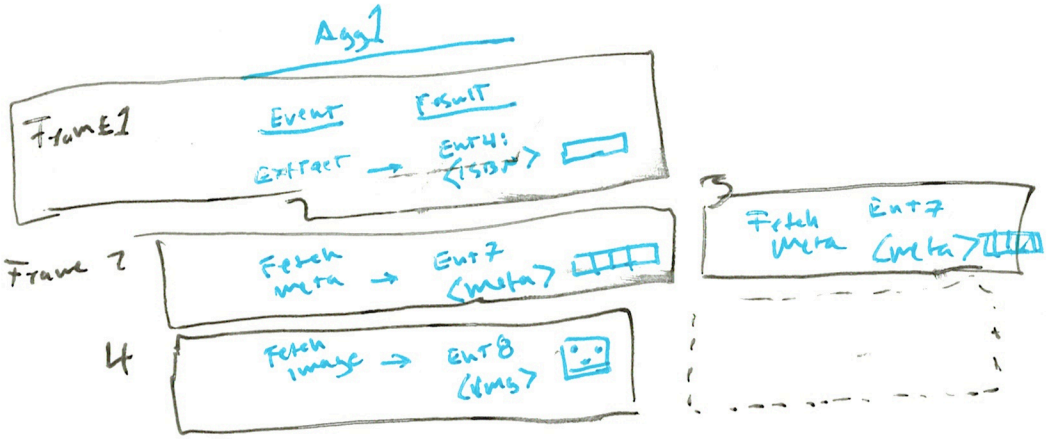


Command
vs
Event


Individual Ents Point to Def...
Ents
E1: <Blob>
E2: <File>
E3: <Data/CSV>
E4: <ISBN>
E5: <ISBN>
E6: <ISBN>
E7: <ISBN>
E8: <InvRec>
E9: <File/img>
E10: <File/img>
ISBN
Inventory record
Media img
Frame 2
ISBNS
AggId
A1
A2
A3
B2
ISBNS
config
meta
Frame 5
<meta>
<API Resp>
B3
ISBN
config
File/img
Frame 4
A1: File
A2: File
A3: File
Plan
B1
Plan
B2
B3
Exec
B1
B2
B3
Exec
Frame 2
Exec
Exec
Frame 3
Frame 2
B1
<ISBN>
<meta>
B2
or
Exec
B1
B2
B3
A1
A2

What happens - how is Ent or Agg Materialized - when it has parallel parents





Blocks have Specs that declare the handle (name), shape, rules of their inputs & outputs

Ops have spec

cxn have spec

Notebook has spec

User adds ops to Blocks and Blocks to notebook until their is a path from ~~notebook block foot to notebook OUT SPEC~~ in-spec to out-spec

Block out-specs create collections cxn in notebook

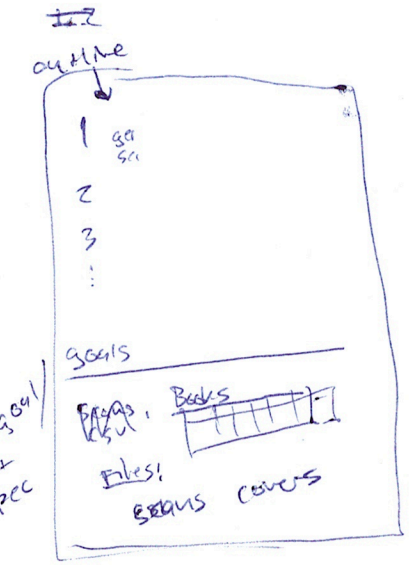
Exec run ops, put resulting values in cxn defined by spec

Exec specifies the instance values (their version) used for op. Values ~~must~~ obey specs.

→ implicit, b/c two ops can only be chained if specs compatible

→ UI hint, not strict

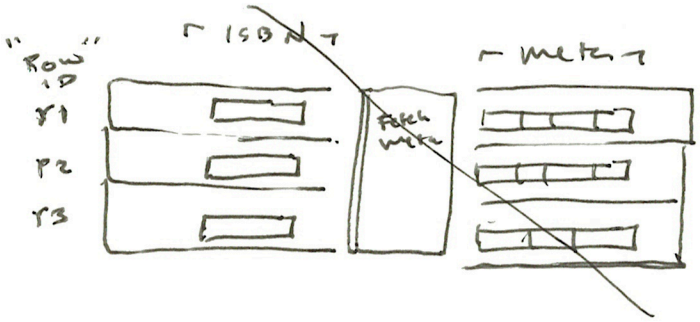
op Exec just returns [] if



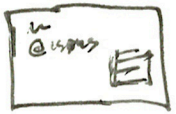
Weds
Container Storage

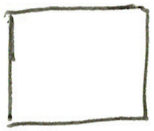
Affordable SCIF storage
10P 6P
Albert
415.822.444444
750 Ave E x 9th st

chris storage SF
415 314 7675

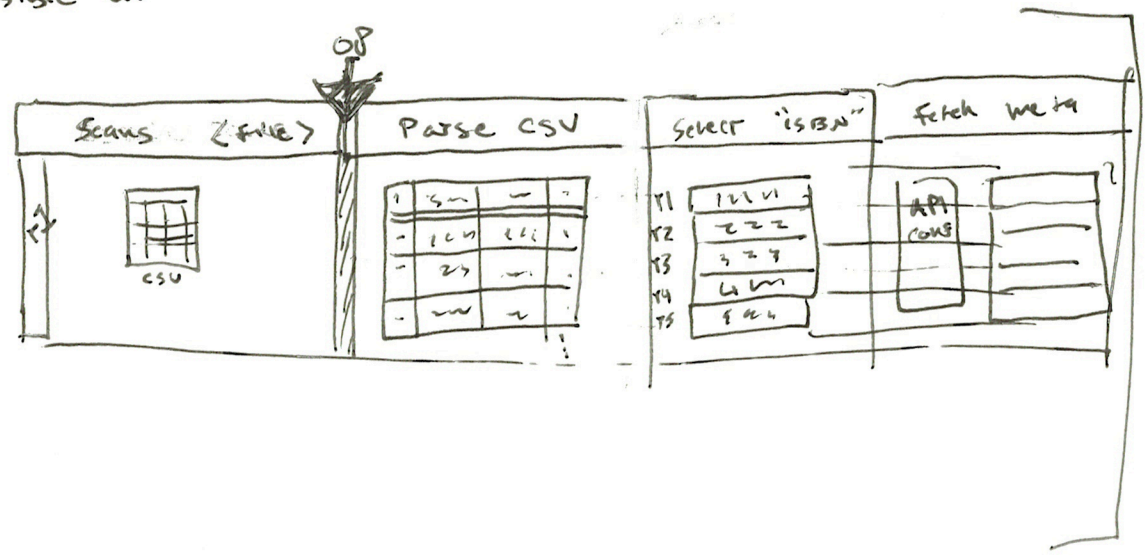


workflow
B1 (CSV)
↓
B2 ISBNs

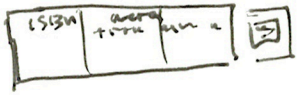


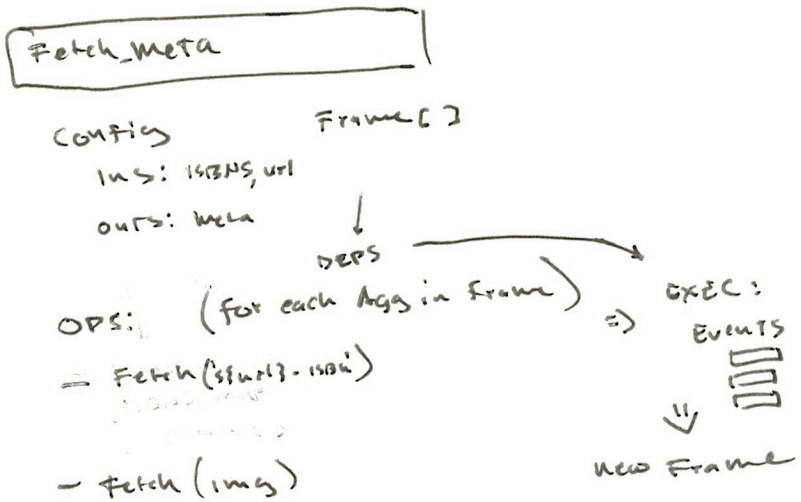


more of an "Excel Sheets" style ui



goal spec





Plan

- File → csv
- csv → ISBNS
- ISBN → meta
- ISBN → img

1. Scans (file)
2. ISBNs (ISBN)
3. meta (Book meta)
   - title
   - auth
   - Date
4. Cover (img)
5. Books (Book records)
   ISBN meta cover_img

adding Ents causes new Exec same "version Branch"

"version Branch" =

Blocks create relations

Steps/ops create Ents (tables)

---

def

Event: ~~CID~~
ins: [~~rID, CID~~]
resut: [~~eID~~]

CID:
Parent: ~~Exec~~ / Frame
OP:
result:

total state / Event DAG when developing / editing NB Flow

- diff from -

Final version of state / history ~ "snapshot"?

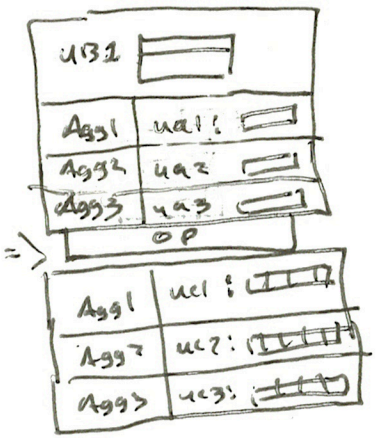
Unit of Organizat: ... : Exec

OP:

ins: [AggUID1, Ag2, Ag3]

[ua2, ub1] [ua3, ub1]

out: [AggUID1: val, ag2: val, ag3, val]



Unit of Organization: Aggregate:

DB views

DG it w/ state sson to start

msgLog =>

NB → OpDAG → stateDAG

(Engine.compute)

① UI

Entity 1

Entity 2

+

menu

Entity CxN ⚙

UI components

- EntList
- EntItem
- addEnt button
- Edit toggle home button

~3/10/2020

②

Ent 1

- Attrib 1
- Attrib 2 ← click to edit

+

Ent 2

+

Definition

Ent 1:

Attributes

~~UI config~~

Collection

- UI config
- definition
- values

- Attrib list
- Attrib item
- add attrib

③

Ent 1

Attrib 1

name
type
desc

Attrib 2

name
type
desc

④ Collection

Ent 1

vals

def

Value list

Attrib 1

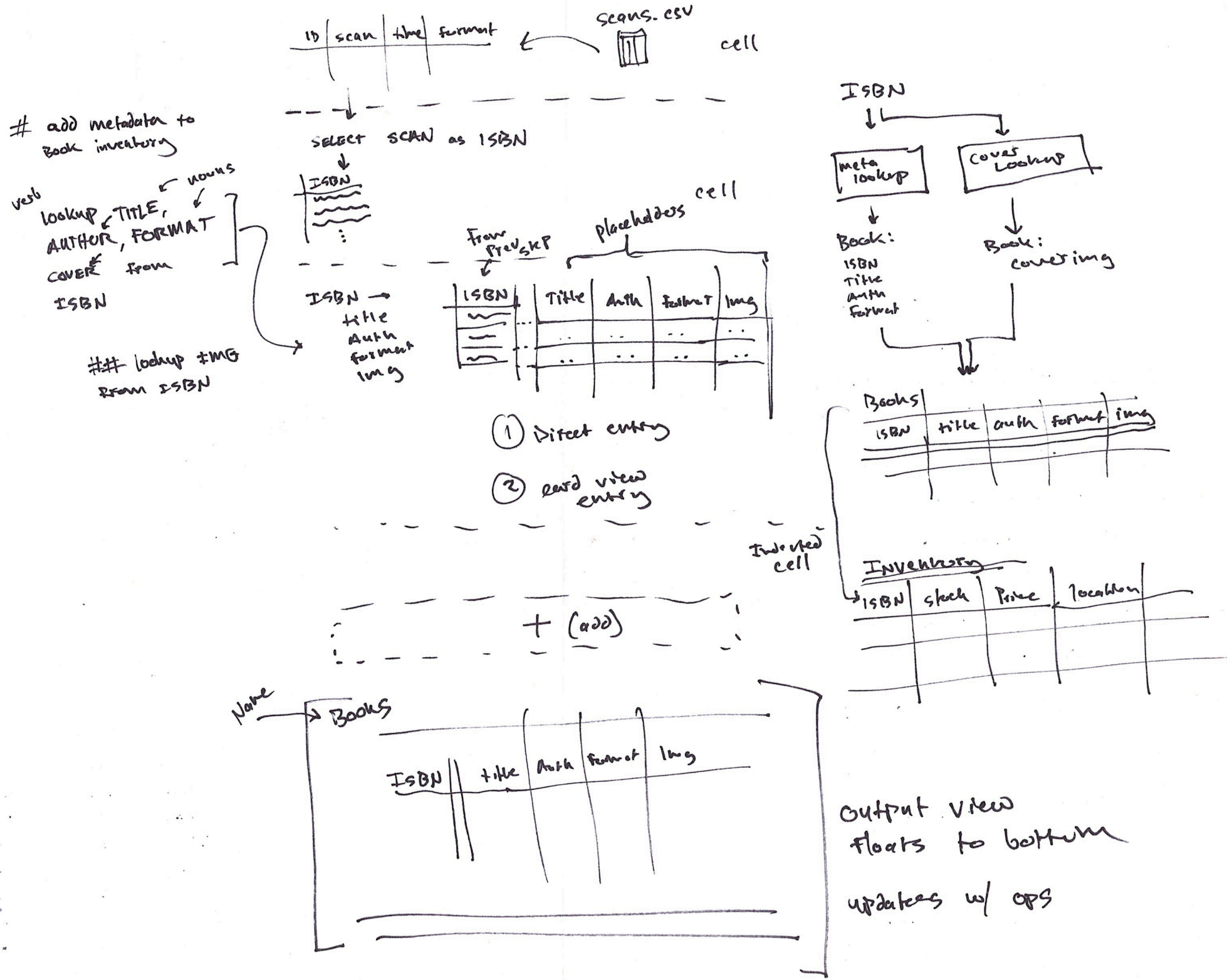
name
type
desc

Attrib 2

name
type
desc

Attrib 3

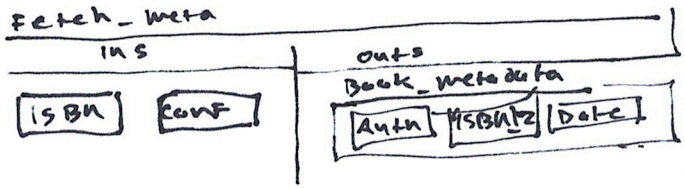
- AttribItemDetail
- AttribItem Edit


ID | scan | time | format
scans.csv
cell
add metadata to Book inventory
SELECT SCAN as ISBN
ISBN
verb
nouns
lookup TITLE, AUTHOR, FORMAT
COVER from ISBN
lookup IMG from ISBN
ISBN → title auth format img
from previous
placeholders
cell
ISBN | Title | Auth | format | img
(1) Direct entry
(2) card view entry
ISBN
meta lookup
cover lookup
Book: ISBN Title Auth Format
Book: cover img
Books
ISBN | title | auth | format | img
Inverted cell
Inventory
ISBN | stock | Price | location
+ (add)
Name
Books
ISBN | title | Auth | format | img
output view floats to bottom
updates w/ ops

Scan
↓
Scans csv Parse
↓ Scan — Extract
ISBNs, conf
↓ Fetch Meta

ISBN_12
Auth
cover_URL

↓ Fetch img



OUT:

Name: ISBNS

Def: →

rows: Aggregates

AggregateID : ⟨isbn⟩

⋮

Address to unique thing in time & space

(sikid) - BlockId - AggId — thingId - timestamp : val

nrf

~~2x1~~

2x1 JST power

4x1 led

5x1 Knob line

Dog Vet recc From M+M Bretz

31 → charlavoix

EntDefn (~~EntSchema~~) | EntInst | EntsStore

EntDefn

~~CreateField~~
UpdateField
delete Field
getField
getFieldS

Fields: set()

~~Entities:~~
instances: ()=>
EntityStore.Filter({@id: "name"})

EntInst

Frame!
refname: samples
defs: [
thinguuids
]
rows [
[thingid1, thingid2],
[{ literal
⋮
]
]

EntsStore

EntInst
EntInst
EntInst
⋮

but they are really → thing uuids

namespace

name?

UI shows

Context: [
["thingId",
"thingid/attrib1"
"thingid/attrib2"]
(all attrib)

literal → {namespace: {attribname: {defn}}}

Frame:
Context: ~~(defs)~~
~~[thinguuids]~~
row_uuid:

ThingStore

map
id: {
thingsinstance
}
id2: {
attrib1
attrib2
@thingid
@thingid

Thing: (Attribute class)

name: mutablevalue
id: uuid
datatype:
desc
rules
Example
usedby
Instances: (~~map~~) <set>
(by id)
ad11z1ph
11111
0000
id2
⋮
>

A1
⋮
Z
Z999

Block1 @ T1

Clock	Event	Payload
T2	Created spec	<UID18>
T3	created cxn	<UID88>
T4	Created Entity	<UID1A>
T5	updated entity	<UID1B>
T16	created Entity	<UID1C>

spec [
uuid: uid18
name: ISBNS
desc: "~~~ ~"
defn: [
uuid12:
name: isbn
rules: [ ]
desc: [ ]
etc:
]]

cxn [
uuid: uid88
name: isbns
spec: uid18
query / selector : ?
selection: [
]
]

T6 created Block <uuidB2>

T7? cmd: add op Fetch_title
→ Events[

T8 op added op:<opID>

T9 created spec

T10 updated spec ins: [ ] UID8, "ISBNS"

~~T11~~ cmd: Suggest ins
→ Evt

T11 set ins:
ISBNS: Block1 (@T5)?

⋮

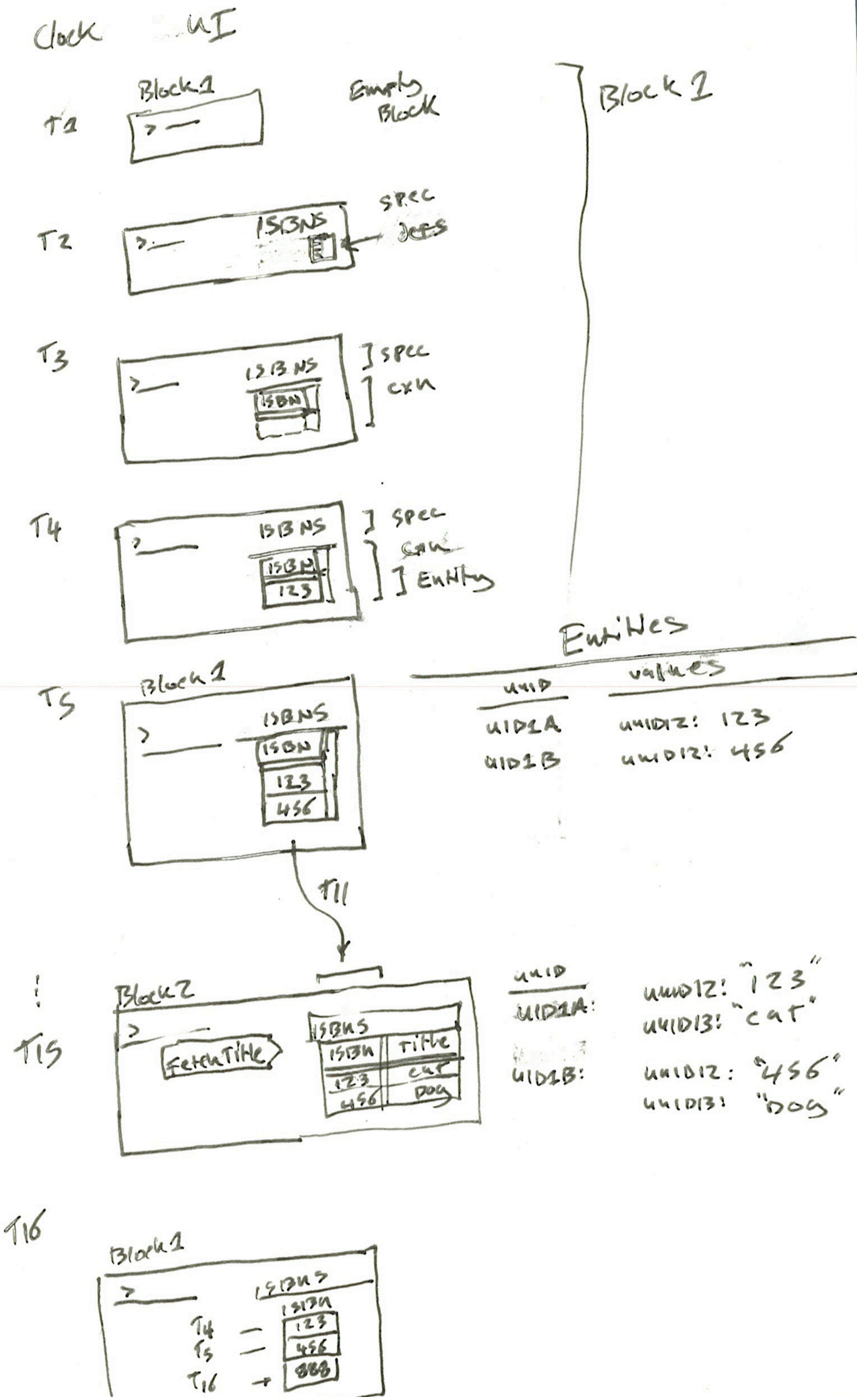
T15 opExecuted:
results [Events? values?]

T17 opExecuted: <UID1C → title>

K-V store? [UID1A: ["isbn": "123"]
UID1B: ["ISBN: "456"]]
or ← json literals

or

Entities [
Ent_02:
uid18/uuid12: "123" ← type references
Ent_02:
uuid18/uuid12: "456"
]



Block

ins A, B

outs ...

ops [ ]

CMDs

configure Block

results Emitted as Events

Causing changes to config/state of Block ...

→ frame

Engine watches Block config

resolves deps

(materializes) Frame for each node

sets up DAG of deps

when deps change, Emits CMDs that Create Branch

- new block node or Exec Events

- nodes Exec automatically when able

Deps: reactive query

Materializes Frame from DAG

→ ID Arrays interface Iterator

① Nodes created into notebook DAG from Event store

② Engine sets up Frame for state,

Deps handles Evts Args | materializes & reacts to changes to Deps

③ Engine Execs DAG

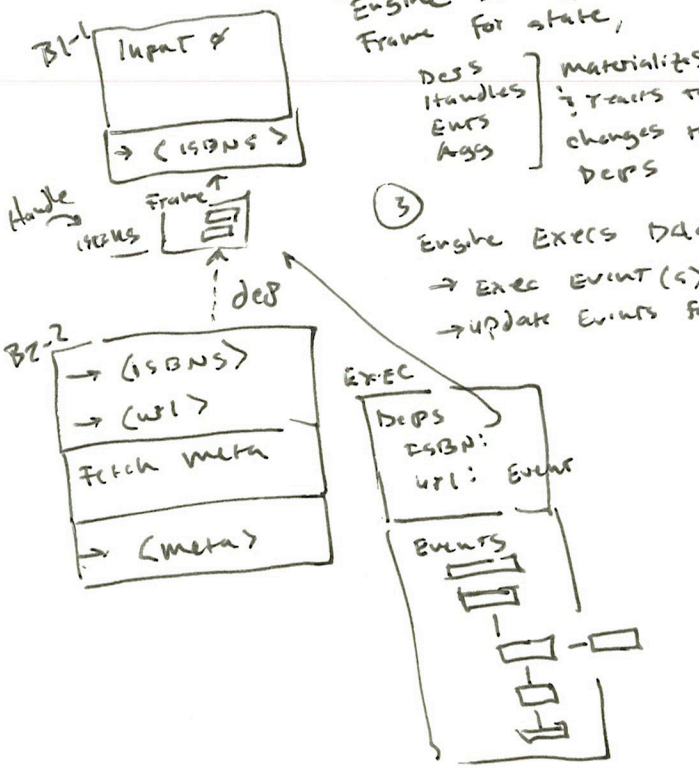
→ Exec Event(s) w/

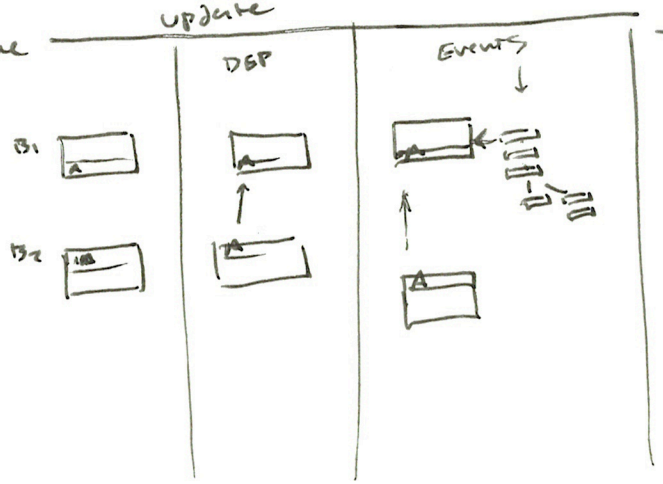
→ update Events for Frame

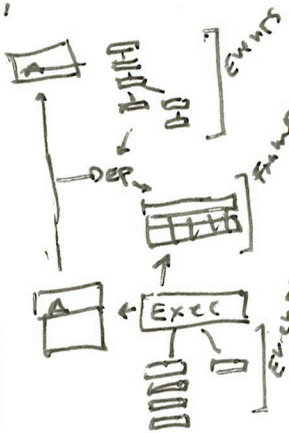
Deps are reactive queries that Trigger Exec Propagation on update

Exec:

observes Deps Array, runs 1 per Element, Emits Events





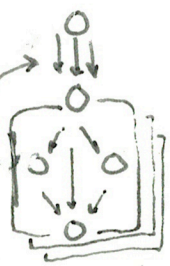


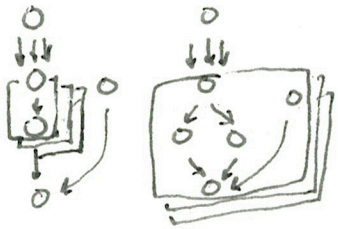

Files
ISBNs
books ISBN ~ meta
1
ISBNS
Bookmeta
123
456
Øfoo
title1
twain
foo
sense
null
+
click
disabled
2
Editing...
err
Selection: ISBN
3
title
Editing
> 888...
4
Date
> 880
preview
title2
self
ok
esc
flow
Note: new "isbn" val 888 used in step 3, 4 to generate Book meta values

Summer '20

cardinality (ordinality?)

Transducers?





① UA1
UA2
UA3

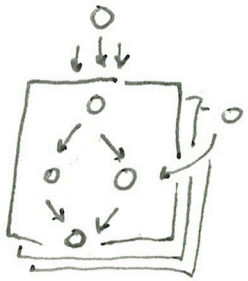
⑤ csv

↑in

SomeBlock | 3

Productions: ...

Productions are arrows in Process Tree ("workflow")



⑤.2 UX
Preferably Ent from Prior production implies wired b/w blocks

④ csv

Productions:

Extract @ r1 | 1
scanDate: 01/22
isbN: ....

Extract @ r2 | 2
scanDate: 2022
isbN: ....

or

ExtractRows@ 2
isbns:
r1: scanDate isbn
r2: scanDate isbn

③ "Ent"/"Out"/"Exec"/"Production"

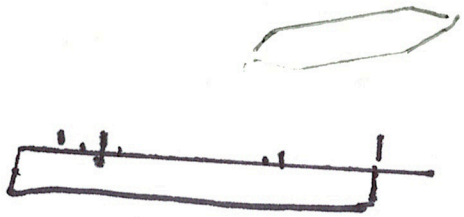
UE1: csv

↑ Parse

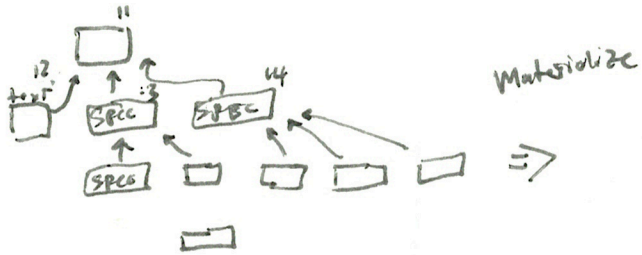
↑ structure/Extract

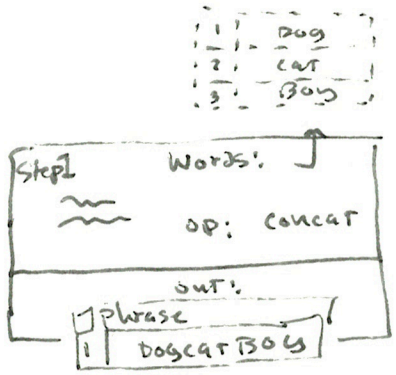
@ r1: q
@ r2: b
@ r3: a

(Extract CSV to Rows)

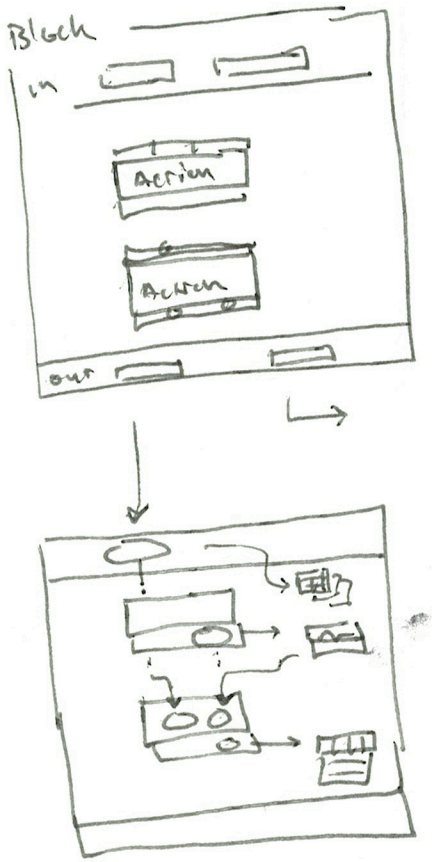


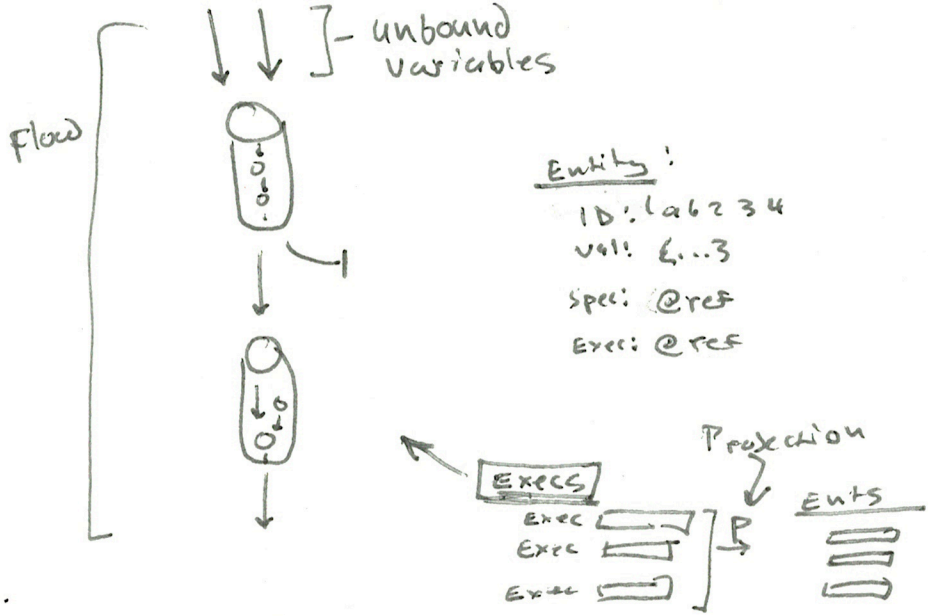
Step 1: Do cat
In: words Out: phrase





Step 2: Do count letters
In: string out: count

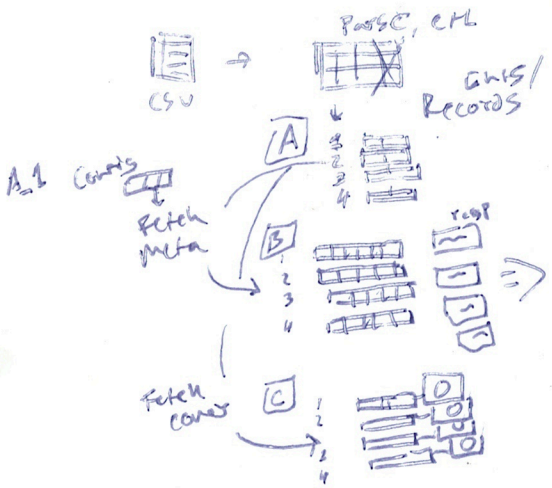




Execs are Bounded contexts

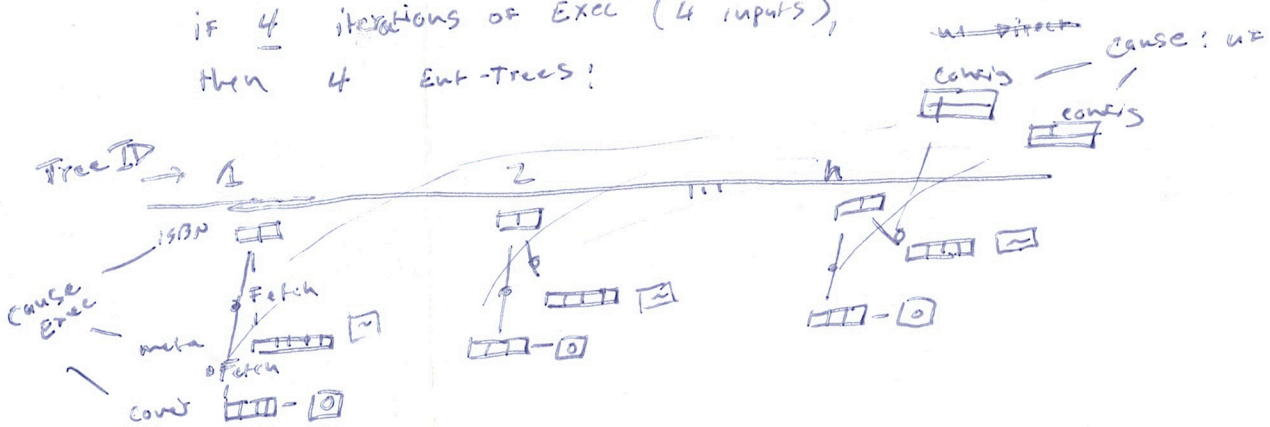
Ents are Projected Dynamically
- but serialized Elsewhere (to disk)

16 ~~14~~ Apr 20



Each Ent is potentially
linked into tree of relations
from transformations / Execs / edits

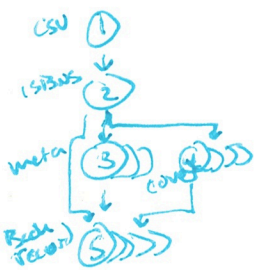
if 4 iterations of Exec (4 inputs),
then 4 Ent-Trees!



20 Apr 20

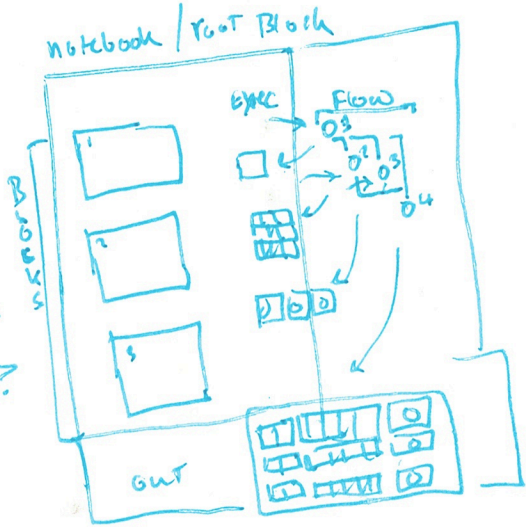
Blocks

1. Scan Books → upload CSV → CSV
2. Parse ISBNs from CSV → ISBNs
3. Fetch meta - using ISBN
4. ??? ... ?
5. Finished Book_inv record - save as CSV → IO



outline view scaffolds FLOW

So where is FLOW
Defined in notebook..?
- Empty ?
- 1 Block ?
- 2 Blocks, no link, ents ?
- 4 Blocks, some links, ents ?



- Blocks are Containers for user content
- NB Engine suggests / resolves Flow DAG based on Block IO + user
- Execs & Data instances are linked to ... which link to blocks

Prgm cmd opt ? R
Save shift ctl opt

Base Frame AKA snapshot

DB

Handles

ISBNS →

API →

Spec

vals

- 111
- 222
- 333

Spec

vals

aaa

bbb

"@/" builtin

ops [ ]

Baby EventLog

CMDS

Events

- EntityCreated:
  - Parent: spec1 ?
  - cause: ?

- Invalid out of order
  - EntityUpdated
    - id Ent12
    - val ~
  - Entity created
    - Parent: Spec1 ?
    - id: Ent12
    - val: ~

	ISBNS		Roadword
C11	11111	C44	cat
C22	12222	C55	foo
C33	3331	C77	Boobie
C66	4444		

⇓

C11, C44

C22, C55

C33, C77

C66, C44

Avg	ISBNS	Inventory		API
@1	A1.1 : 11111	B1.4	12	C1.7 { host "www..." key xyz112 }
@2	A2.2 2221	B2.5	∅	
@3	A3.3 3333	B3.6	8	

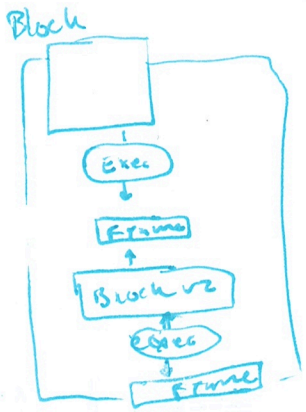
⇓

getAvgPrice

ISBN INV API

⇓

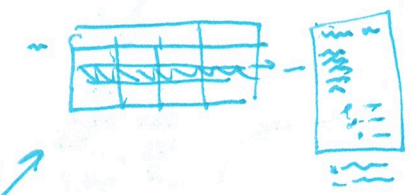
@1:
- OPID: G11-11
- ins: [A1.1, B1.4, C1.7]
- out:
  - CID: { price: $12
  - resp: { ~~~ } }



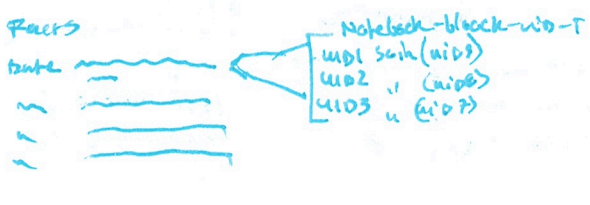
Dynamic DB slice from all notebook logs →

Primers

oligos

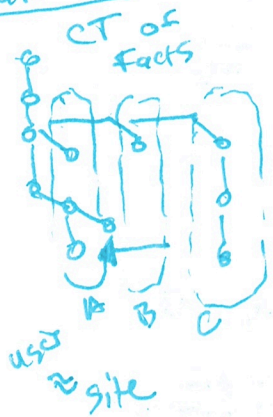


aggregates to ↗



step backward to see history

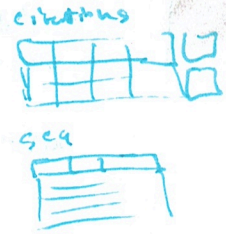
variations



used ≈ site

Ranked list for logs

→ citations, segs ---→



Order primers

segs → docs

→ oligos

Docs

- order conf
- invoice

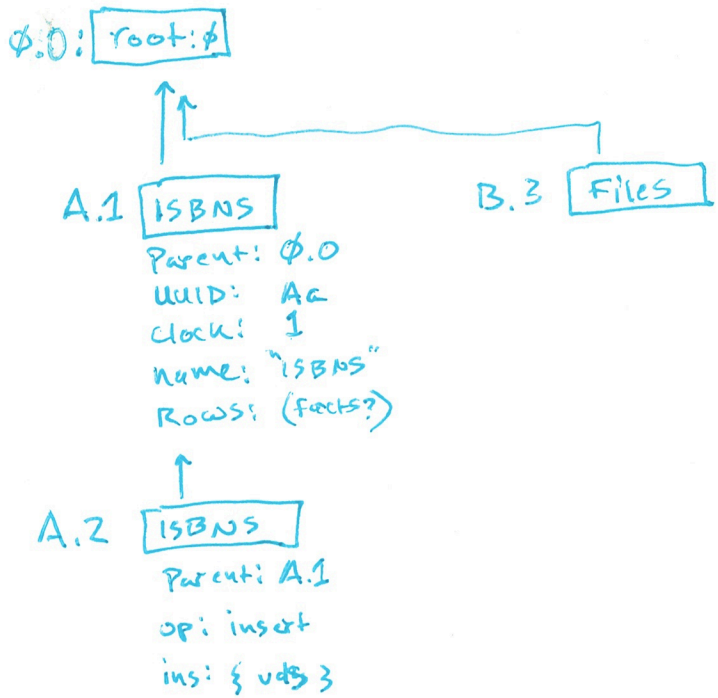
segs oligos

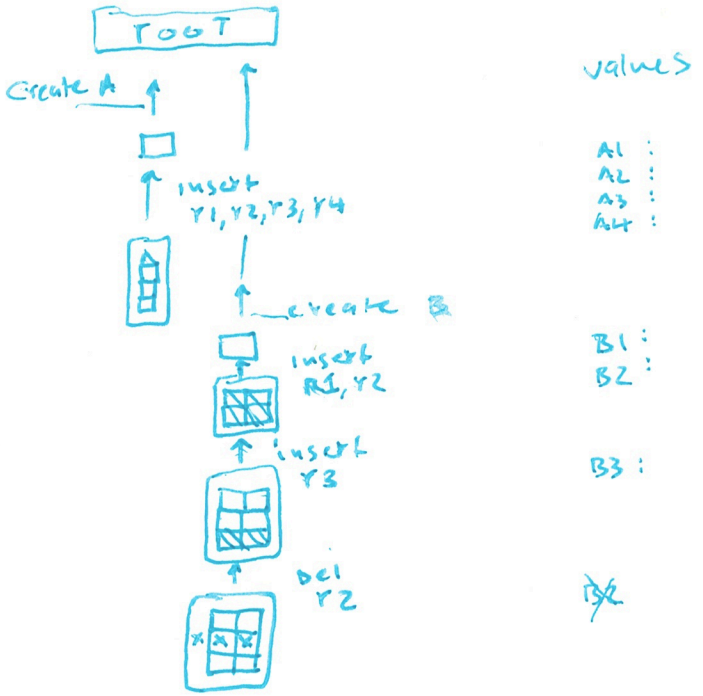
joinby

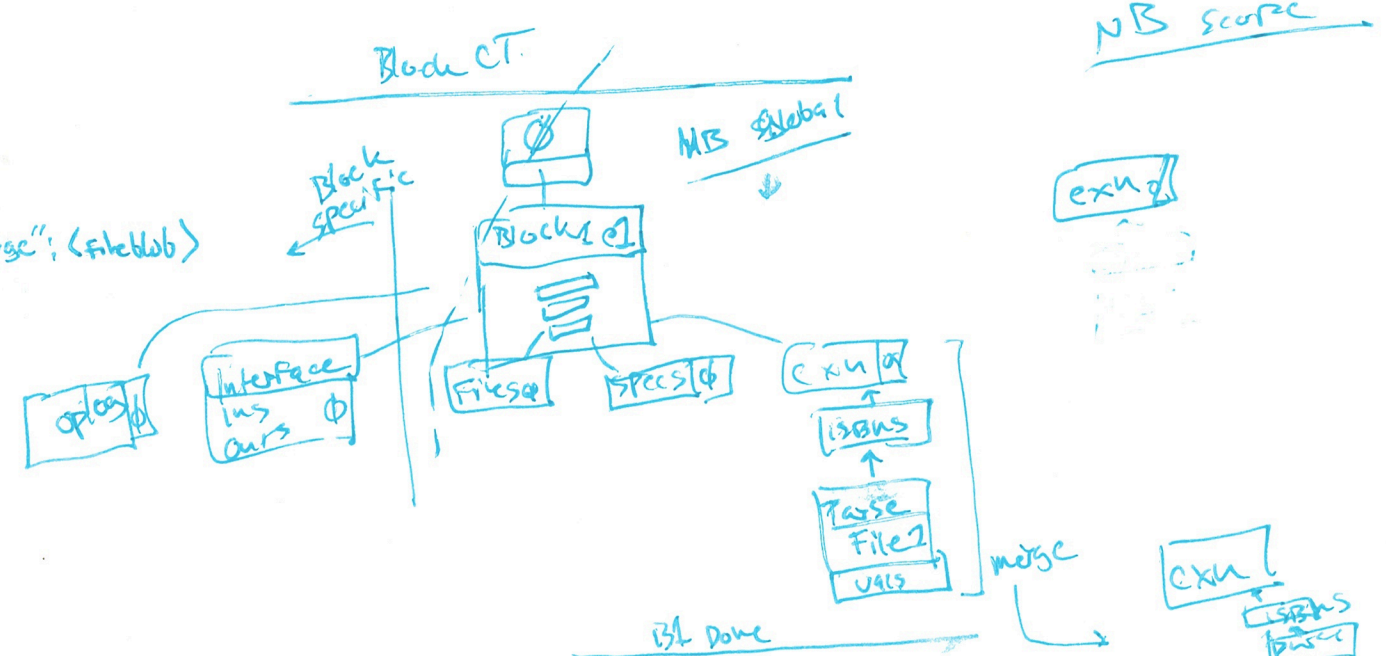
Make stocks

oligos → labstocks

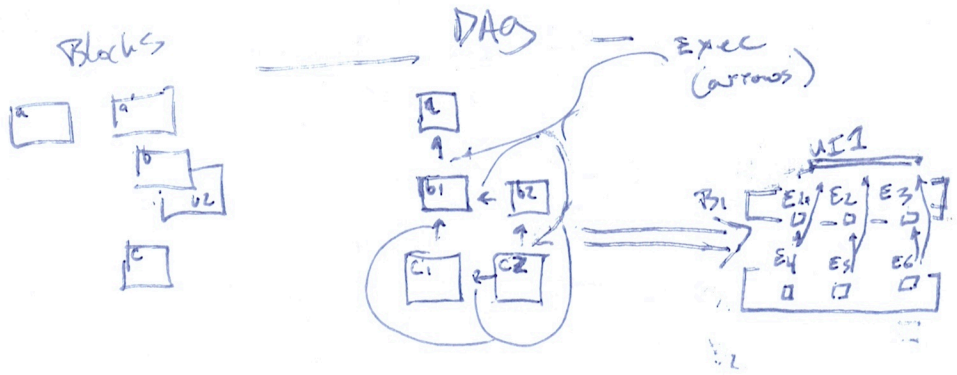
Data / Values / Collections / Frames Study







7 Apr 2020



8 Apr

7 Apr

Make Branch ; Recompute Study

Buttons

New Branch

Switch Branch

update inputs

Accept preview

Dataflow

op1

Interim data

op2

final result

How are results of multiple runs of Bookstore Inventory workflow/ notebook organized?

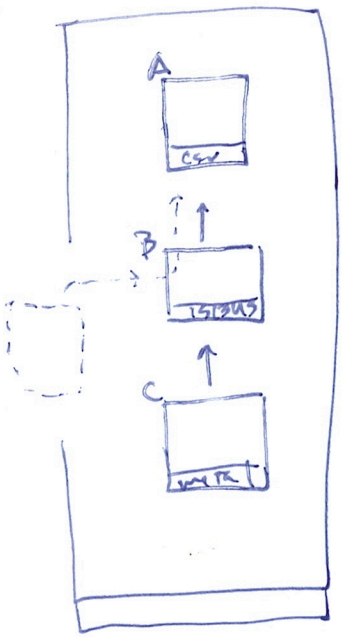
Process

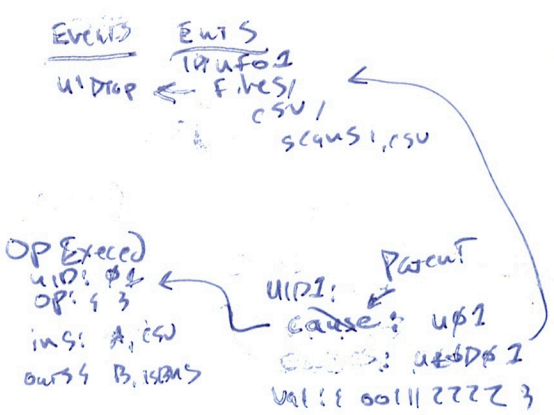
check out pipedream.com

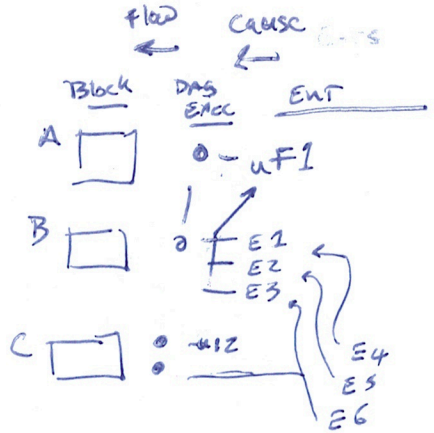
mdx

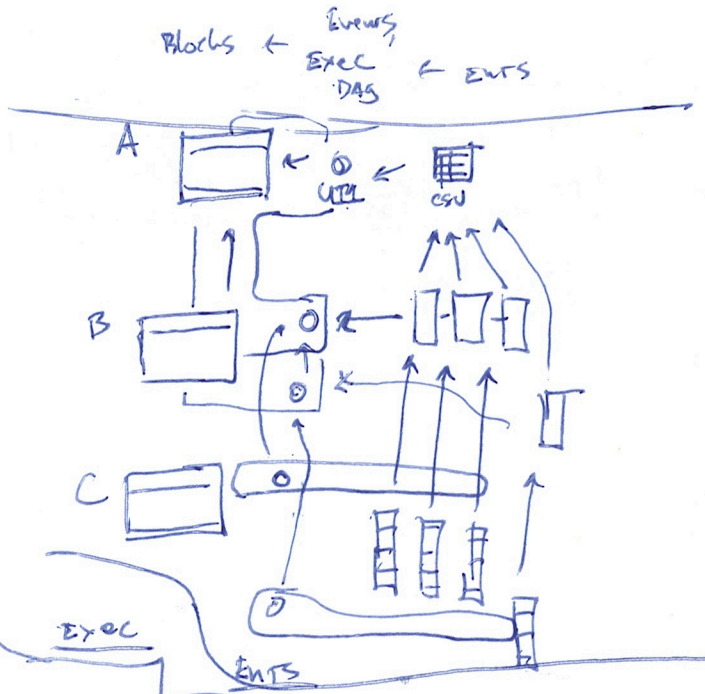
UI state!

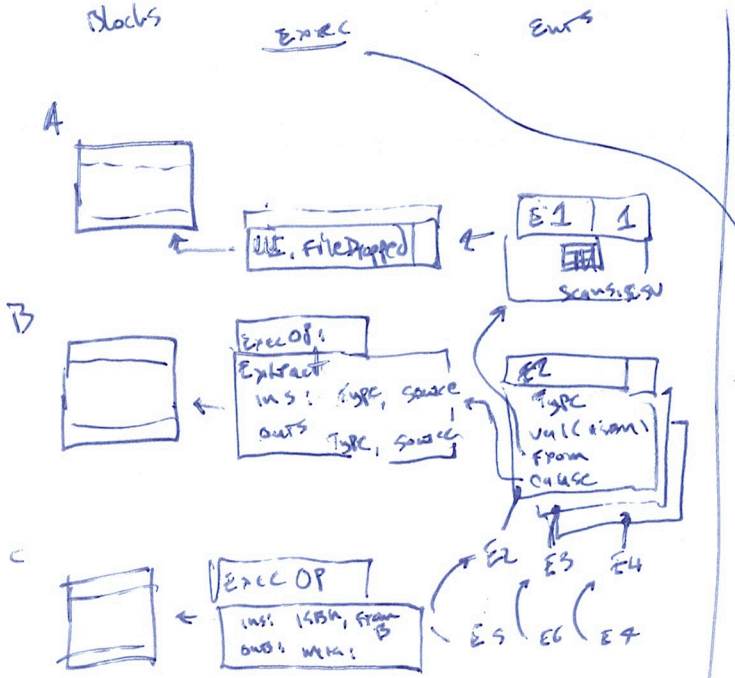
- Branch
- ? Branches
- ? Cur State

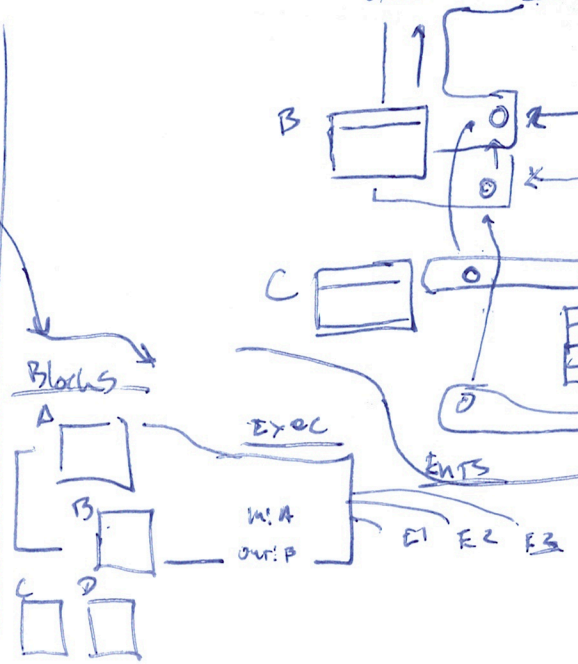

A
csv
B
isbns
C
write

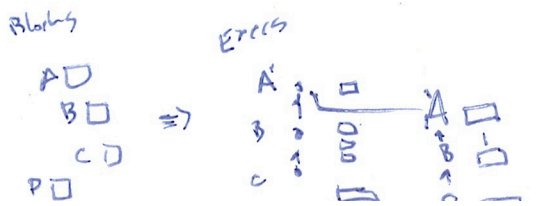

Events
Evnts
Info1
UI Drop
Files/
csv/
scans1.csv
OP Exec
uID: #1
OP: 1 3
ins: A.csv
outs: B.isbns
UID1:
Parent
Cause: up1
Val: 0011 2222 3

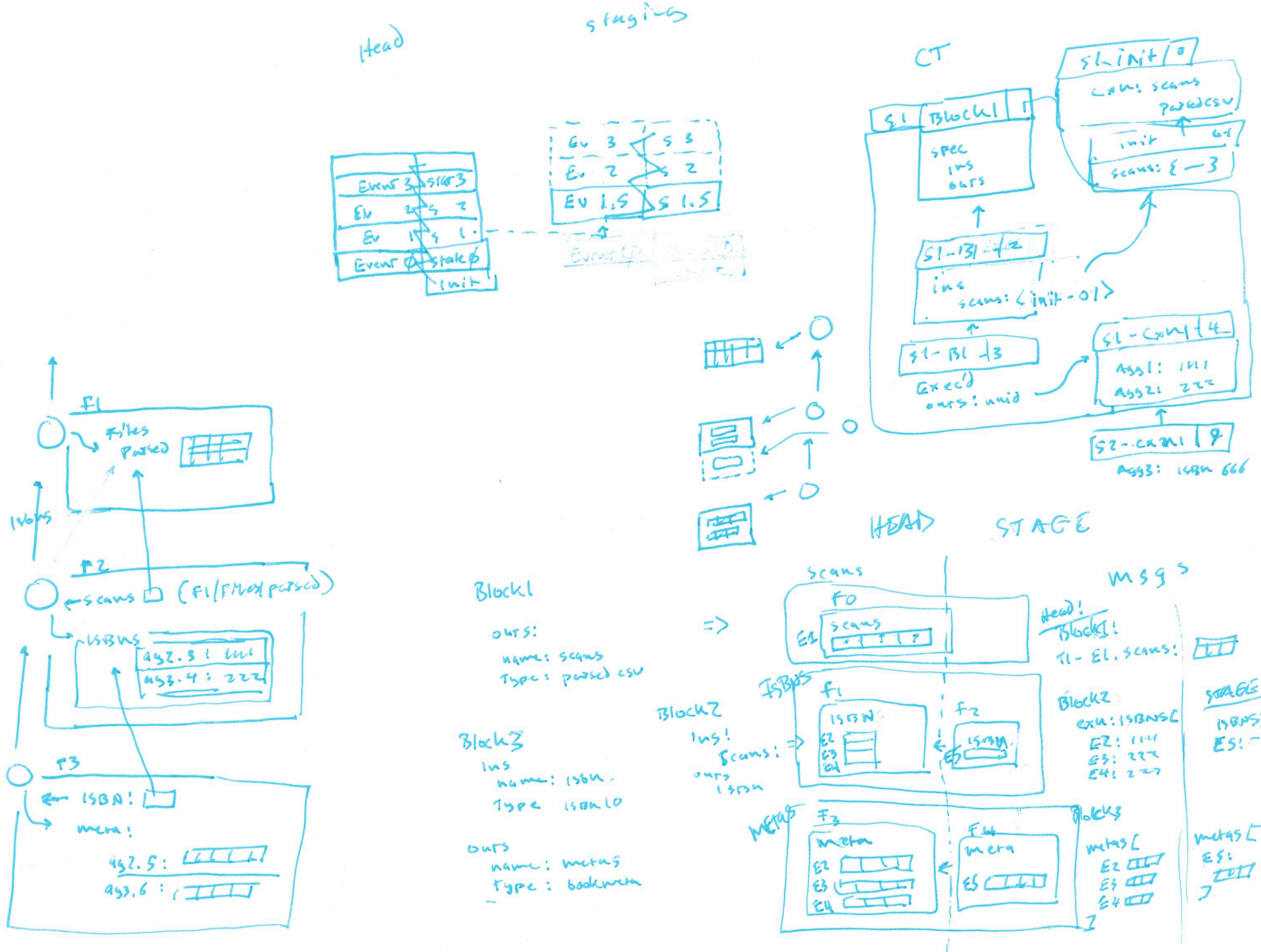

Flow
Cause
Block
DAG
Exec
Evnt
A
uF1
B
E1
E2
E3
C
E4
E5
E6


Blocks
Events
Exec
DAG
Evnts
A
csv
B
C


Blocks
Exec
Evnts
A
E1
1
scans.csv
B
Exec OP:
Extract
ins: type, source
outs type, source
E2
type
val(from)
from
cause
C
Exec OP
ins: isbn, from B
outs: info:
E2
E3
E4
E5
E6
E4


Blocks
A
B
C
D
Exec
in: A
out: B
E1
E2
E3
Evnts


Blocks
Execs
A
B
C
D



Head
staging
CT
Block1
spec
ins
outs
s1-init
scans
parsed csv
init
Event 3
Event Ø
state Ø
Ev 1.5
S 1.5
S1-B1
Exec'd
Agg1: 111
Agg2: 222
Agg3: isbn 666
F1
Files
Parsed
F2
F3
ISBNs
agg2.3 : 111
agg3.4 : 222
ISBN:
meta:
Block1
outs:
name: scans
Type: parsed csv
Block3
ins
name: isbn
Type isbn10
outs
name: metas
Type: bookmeta
HEAD
STAGE
Scans
msgs
F0
F1
F2
F3
F4
Block2
meta
metas

names [ ]

Pick 5 random

namelist

verblist

recombine XX

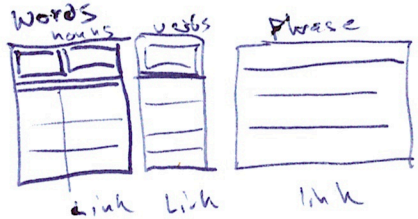
Phraselist

Specs

Vals

namelist

name	type
John	PP
Dog	N
Mill	N



VerbList

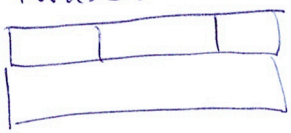
Fly
Run
Jump

add new name

↓

Phraselist updates

Phrases



- Check out tables (blueprint vs Autodesk)
- umbrella translucency
- show Block tree (mock) + specs + vals (empty)

clock

	add name
1	John
2	Jill
3	mary
	add verb
4	run
5	jump
6:	op
!	in names, verbs
12	add name
	New

op

ins [ ] I/O: spec vals

for ins values X:

let _news = X.~~map~~ Filter(X.clock > self.clock)

Action

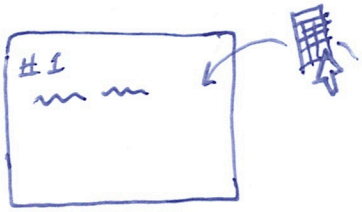
Block1

Block2

Block 3

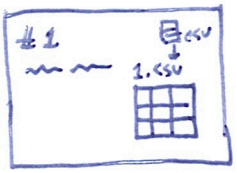
- add csv file

1.V1



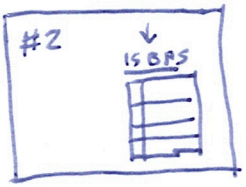
- file uploaded
- file read
- csv parsed
- "1.csv" collection created

1.V2



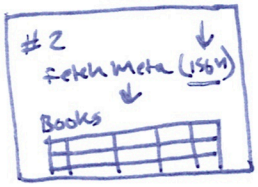
- Block 2 created
- Input: select "isbns" from "1.csv" collection

2.V1



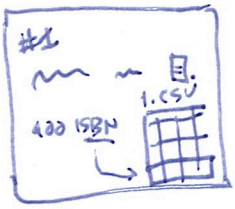
- run op "fetchMeta" on isbns
- op creates "Books" collection

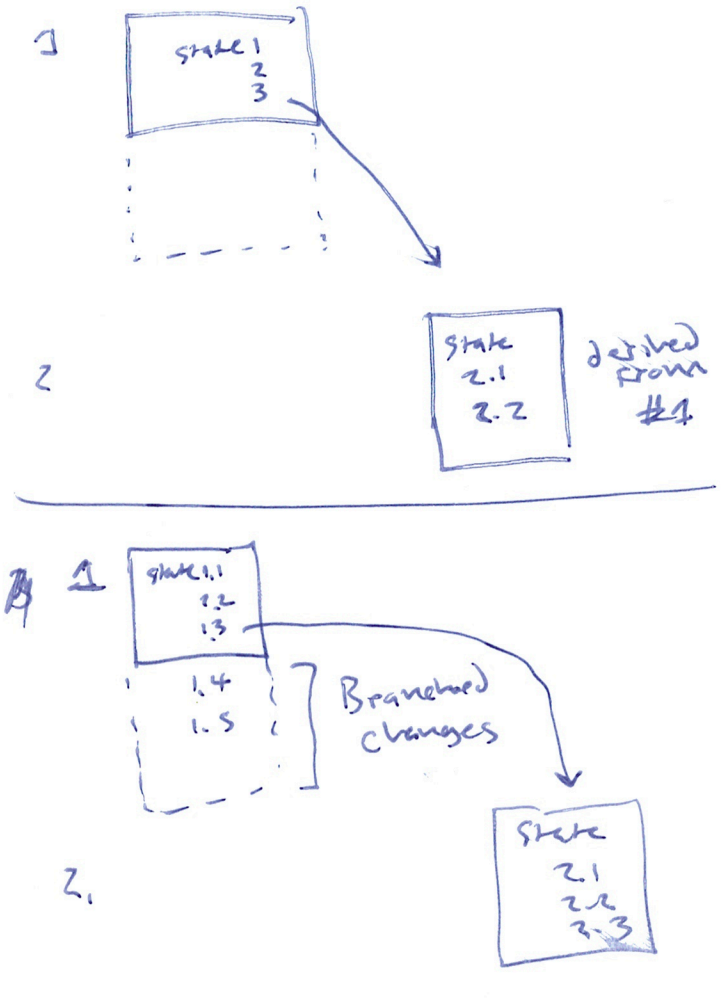
2.V2



- add row to "1.csv" collection

1.V3





ent1. Row1

cat	Dog	ISBN	Price
a	a	a	

ent2. Row2

cat	dog	ISBN	price
a	a	a	1

still at version #1

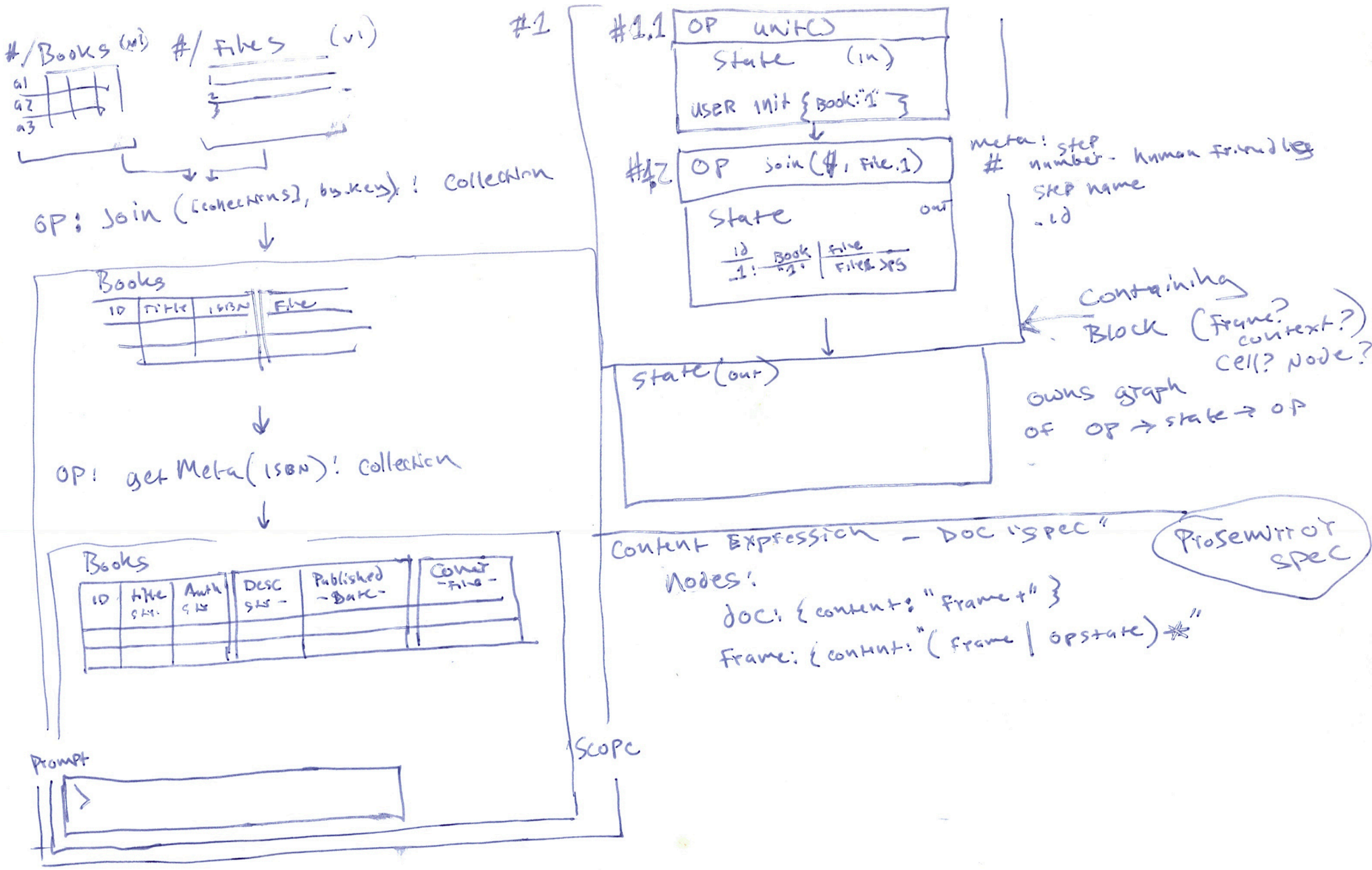
also version 1

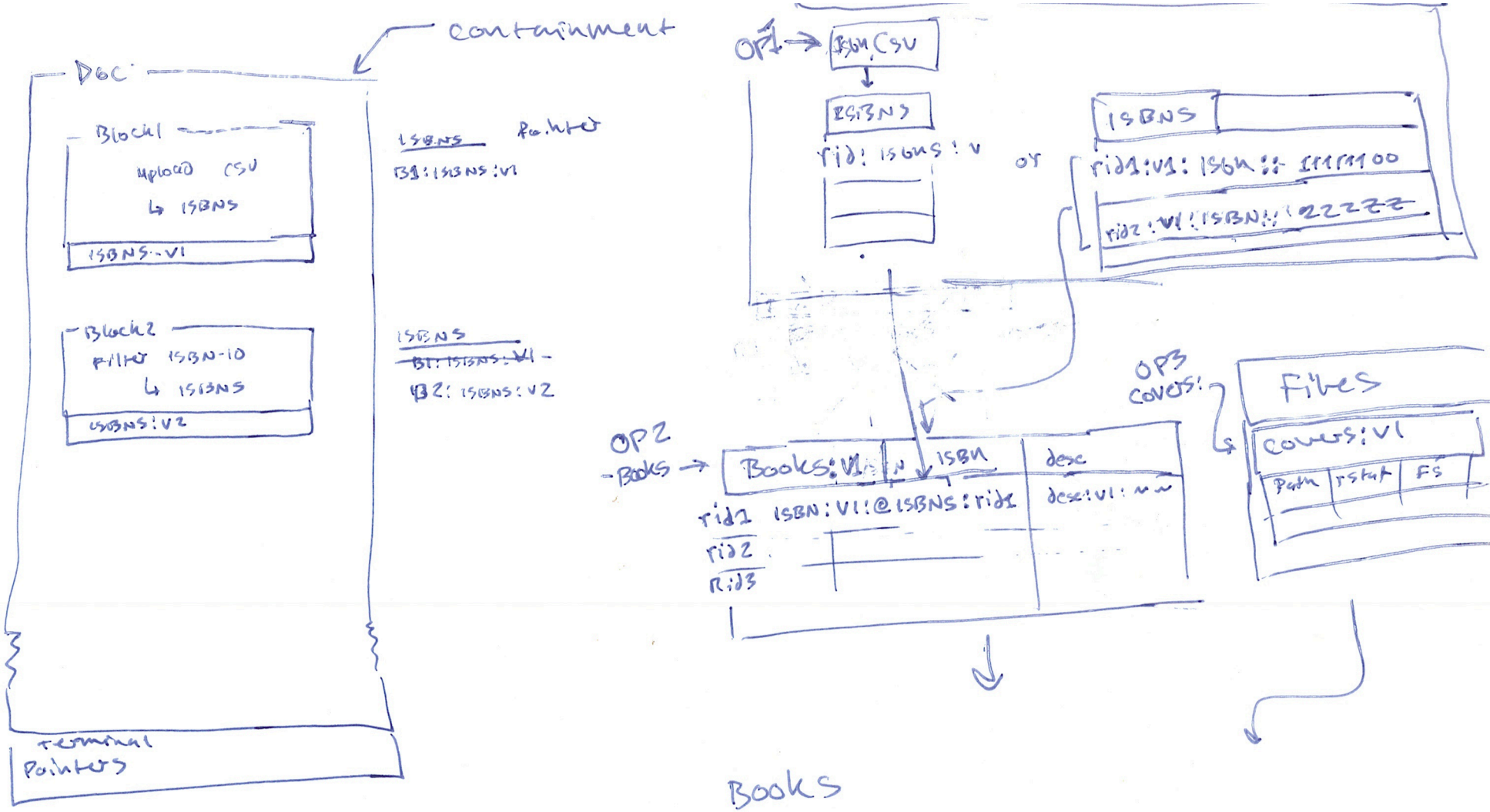
What is version # for ent1. Row1. price?

∅ null?

3 How to update #2?

- Blocks Aggregate event log
- Knobs computes dependant blocks state from log
- ~~Hell~~ Weird? blocks allow out-of-order aggregate insertion of new events → recompute view


#/Books (v2)
#/Files (v1)
a1
a2
a3
#1
#1.1
OP unit()
State (in)
USER init { Book: "1" }
#1.2
OP join(#, File.1)
State
out
id
Book "1"
File
File #5
meta: step
number - human friendly
step name
.id
OP: join ([collections], by.key): Collection
Books
ID
Title
ISBN
File
OP: getMeta(ISBN): Collection
Books
ID
title
Auth
Desc
Published -Date-
Cover -File-
Containing Block (Frame? context? cell? Node?
State (out)
owns graph of OP → state → OP
Content Expression - Doc "spec"
Prosemirror spec
Nodes:
doc: { content: "frame+" }
frame: { content: "(frame | opstate)*" }
Prompt
>
Scope



Zuck 628 206 4420
Mr Zuck
415.353.2757

Step 1

Build drast datamodel

+ UI

showing: use case

+

nonfunctional solution

Step 2

think hard about how to solve.

work backwards, incrementally

Step 3

Find solutions, or realize use-case + problem not important / replaced w/ something else

Problem statement / use-case

(1) Easy / literally- separation b/w

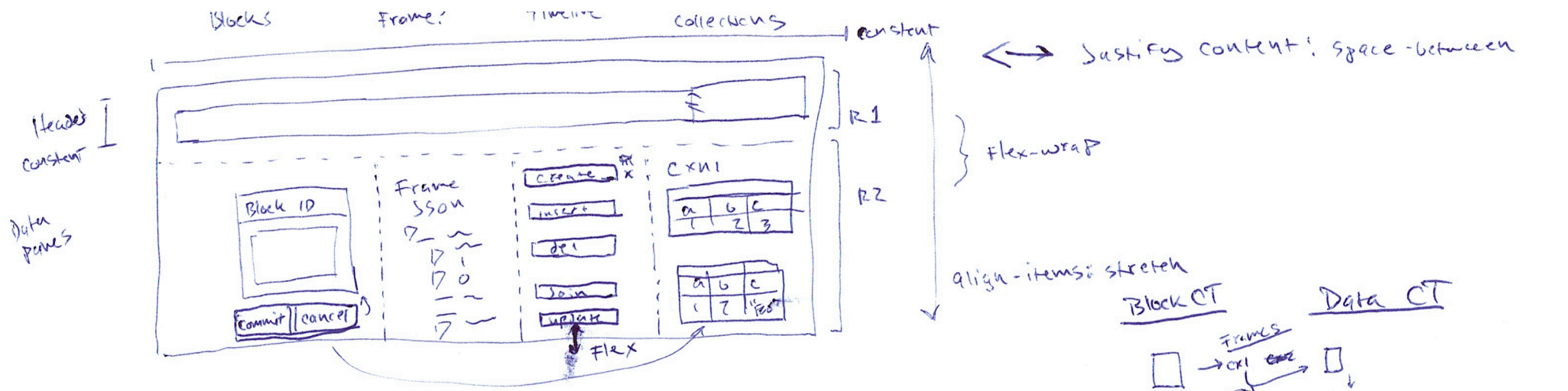
A) "Program" described by Blocktree

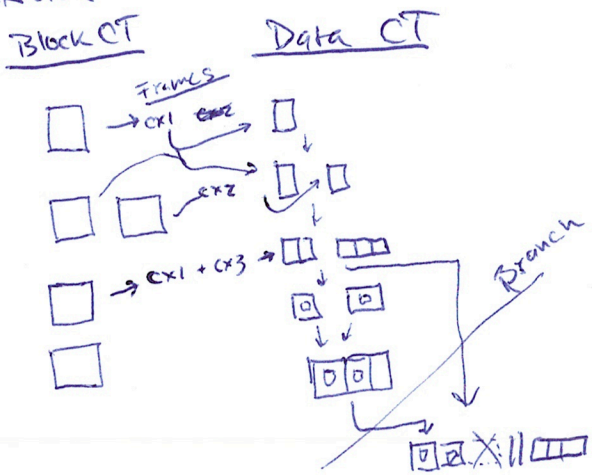
+

B) revisioned history of data (flowing in & out of "program")

requires INDIRECTION LAYER B/W Block "semantic" ops + ~~the~~ instances of Execution ins + outs

(2) Detect future-updates to data deps of Block, optionally return block w/ new data, w/o infinite loop. (causal spiral)





- debug-container (Grommet Diagram component)
  - ~~Grommet~~
  - row
    - header?
    - col 1
    - col 2
    - col 3
    - col 4

Miniblock -
- header
  - ID
- contents
  - ?
  - (ID)

Frame:
- ?
- json?

History:
- list
  - Li (ID)
  - button
    - ok
    - warn
    - select
    - fail

Data / cxn:
- cxnView (FrameView)
  - table (ID)
    - ? list
    - ? cwd

add Block +

Modal ~~confirm~~ Buttons
- commit! | cancel

undo | redo

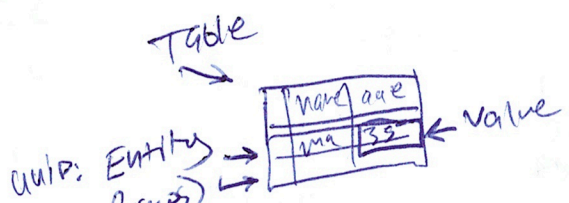
- show / hide / remove

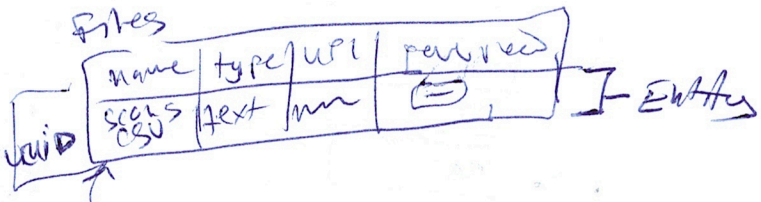
columns
~~Blocktree~~

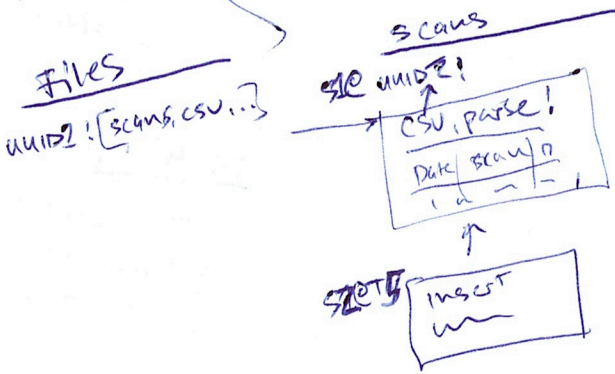
UI column enable/disable

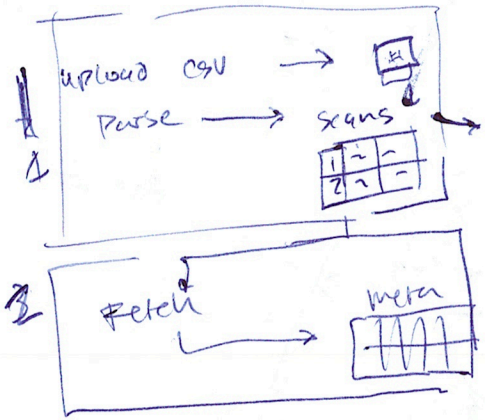
BlockTree | Frames | cmd timeline | cxns | UI history

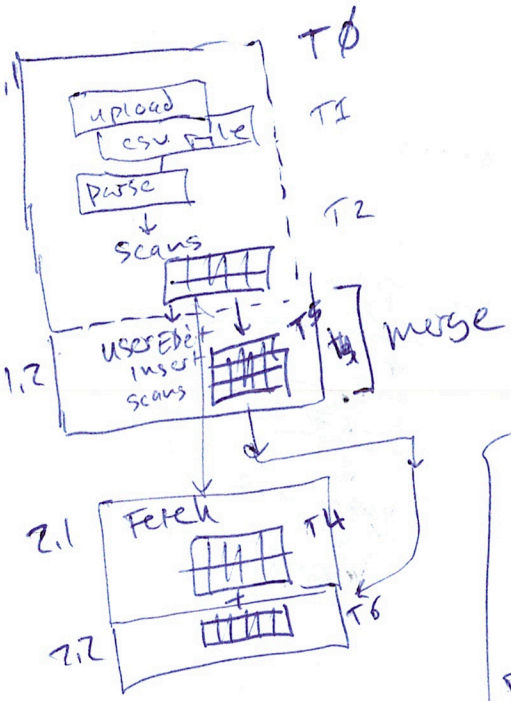
How does a consumer Block Discover there's a new upstream version of a value dep?

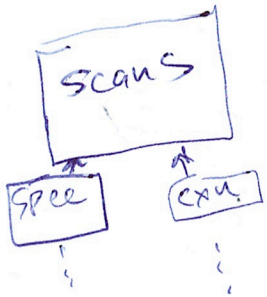










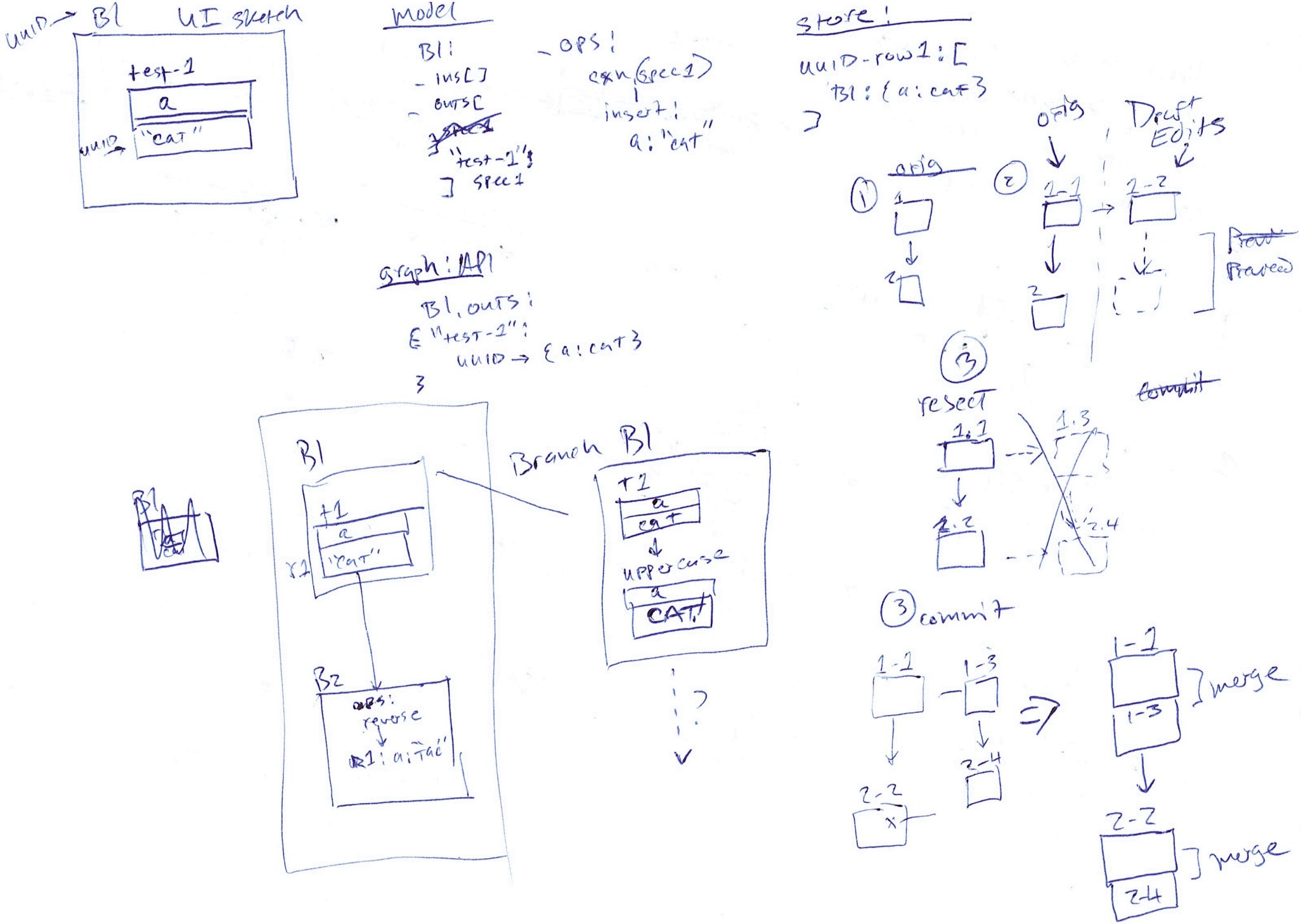


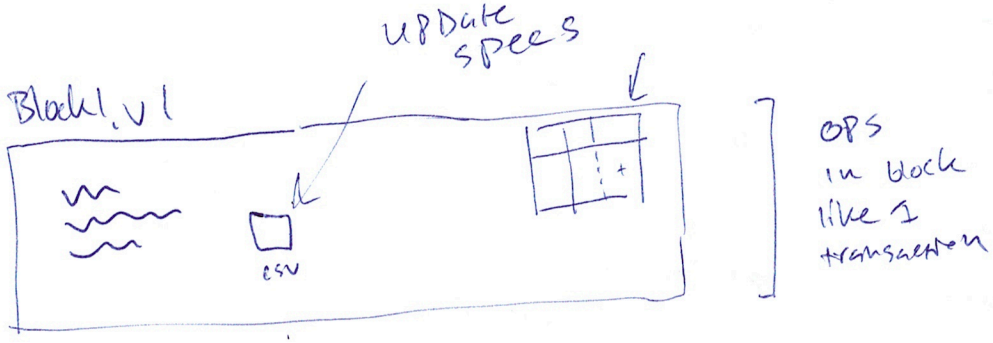
2.1 ins => ~~scans~~
B1.1 - T2, scans
{ Tmax = T2 }

B1.2: fork
scans Tmax = T5

2.1 ins = B.scans (Tmax = T2)

Tmax > cache?
recompute


uuid → B1 UI sketch
test-1
a
uuid → "cat"
model
B1:
- ins[]
- outs[
~~spec1~~
"test-1"
] spec1
- ops:
cxn(spec1)
insert:
a: "cat"
graph API
B1, outs:
{ "test-1":
uuid → {a: cat}
}
B1
t1
a
"cat"
r1
B2
ops:
reverse
r1: a: "tac"
Branch B1
t1
a
cat
uppercase
a
CAT
store:
uuid-row1: [
B1: {a: cat}
]
orig
Draft Edits
1
1-1
1-2
2
2
Proposed
3 reset
1.1
1.3
2.2
2.4
commit
3 commit
1-1
1-3
2-2
2-4
merge
1-3
2-2
merge
2-4



Ent 1

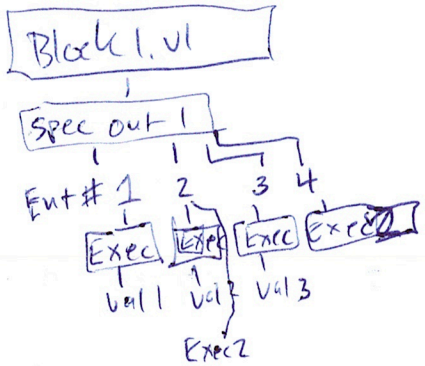
Exec 1

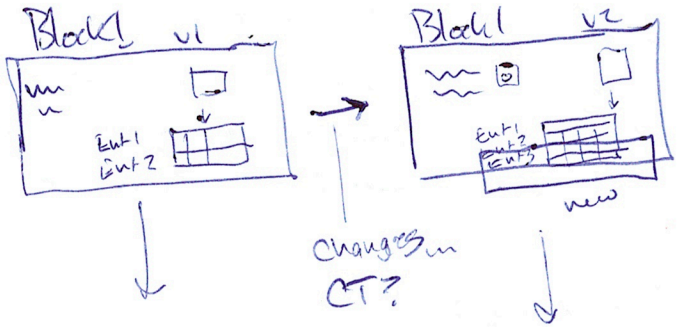
EntID1 → new val 1
EntID2 → new val 2
EntID3 → new val 3

Exec 2

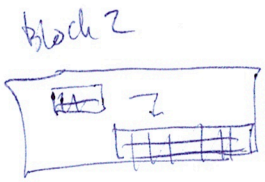
EntID1 → val 1.v1 no change
EntID2 → val 2.v2 ← update
EntID3 → val3.v1 no change
EntID4 → val4.v1 ← add

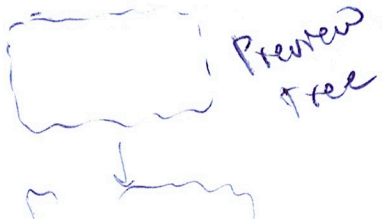
~~Exec~~





Changes stored like concurrent edits from diff site





1 B01

(1) 2 SpecA
↑
3 cxn1
↑
4 USERIO
insert msg:
rid1 { Hello
rid2 world!
↑
5 op1
Count
rid1: msglength: 5
rid2: " : 6

Block 1

1 spec1
name: msgs

disable

ops
☐ 3 usertext
☐ 4 count

Join
cxn1
cxn2
on: msg

1 cxns:
2 msgs: [ ]
11
⋮
22 msgs: [
rid1: "reversed"

	msg
r1	Hello
r2	world!

} 1

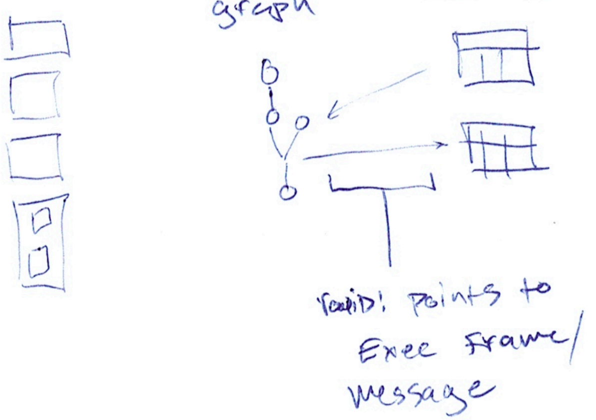
	msg	Length
r1	Hello	5
r2	world!	6

} 2

☐ 20 r1: msg: olleh

Design Principles

Design
Execution graph
Data values



Data outputs are immutable

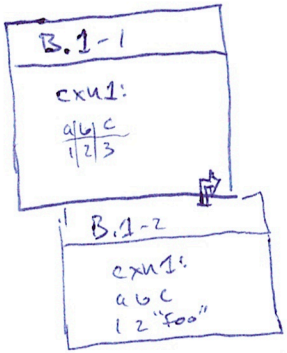
Blocks scope
& deltas to data

Therefore Data + Execution can be seen as a static snapshot at some time (+ for design) ⇒ can get stable UUIDS

r1 →
~~Hello~~
olleh
} 20

Would be nice if Exec context + output could be stored in Blockchain like thing

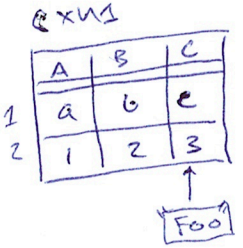
Block Tree



Frames

timeline

Collections



Sort By:

1: Timestamp + uuid (mergesort)
   1st by time, then by site

2 depth-First Preorder traversal
   (~~order~~ reverse-chronological)
   ∴ i.e. newer on leftish
   Branch of tree

State

[EntityID : Val]

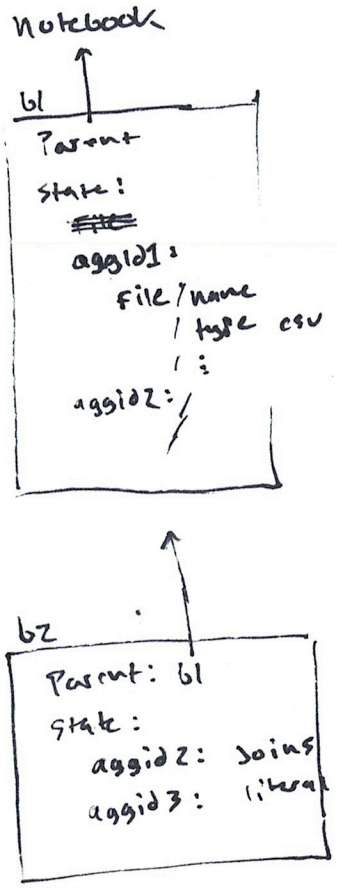
BLOCK

State: <state>

thing Concept Attribute

StateDAG

Blocks



Block1

ins:

name isbns

type file/csv

op

outs: file, isbns

_id: b1

block2

_id: b2

ins:

isbns: b1/isbns

APIconf: aggid23

outs

~~aggid~~

Books: aggid3, 4, 5

aggid2: delta

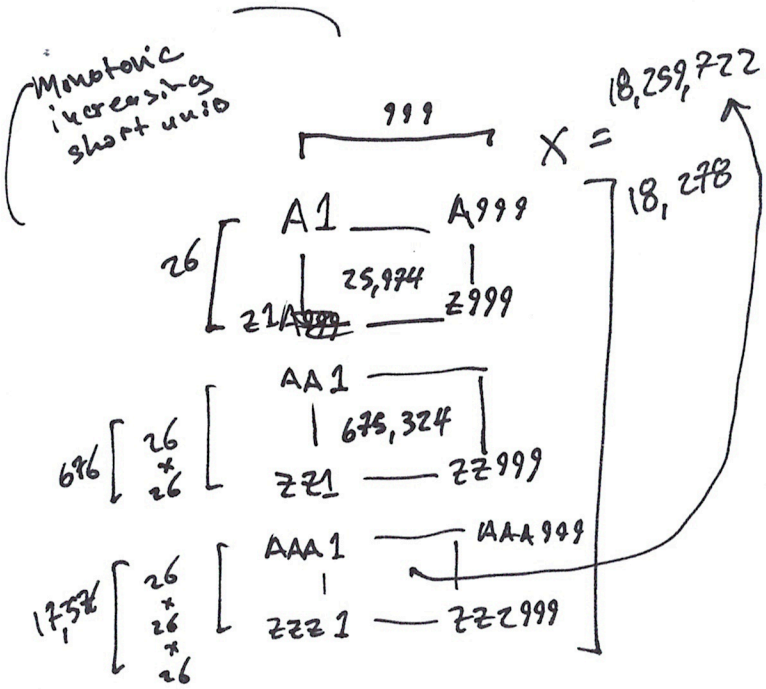
Books

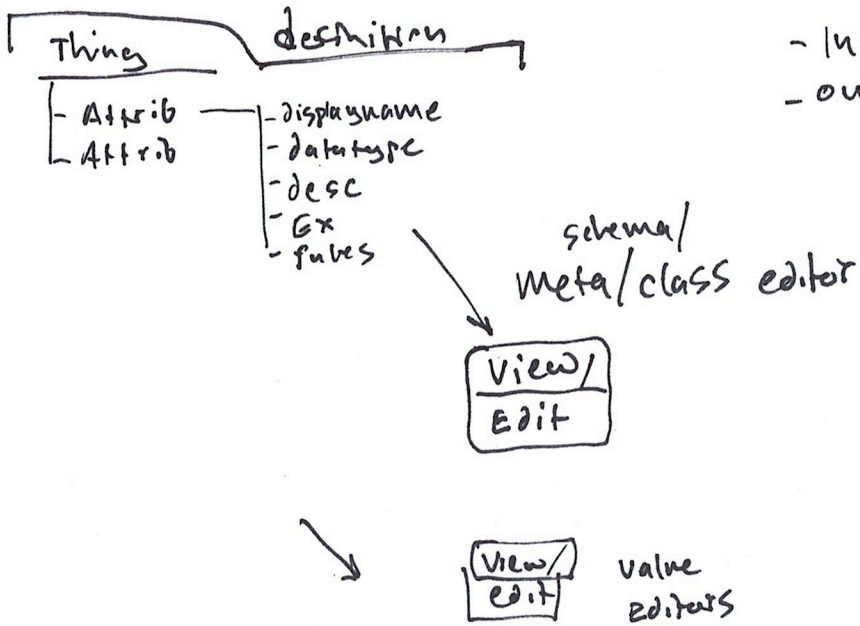
Book:

ISBN: ~~@isbn~~

Auth

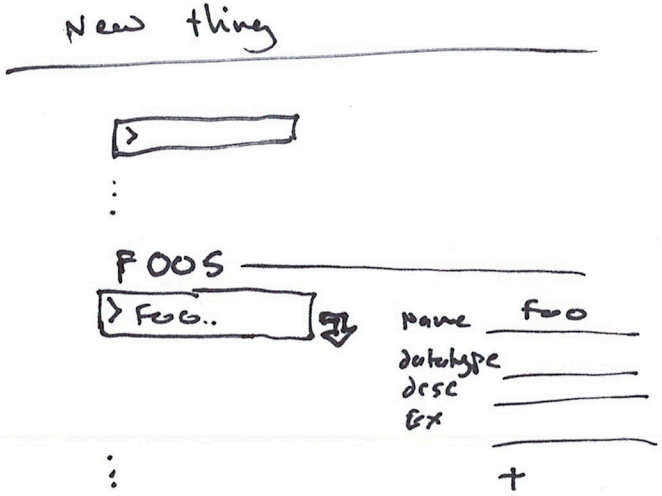
~~Cover~~ : @ File/image/

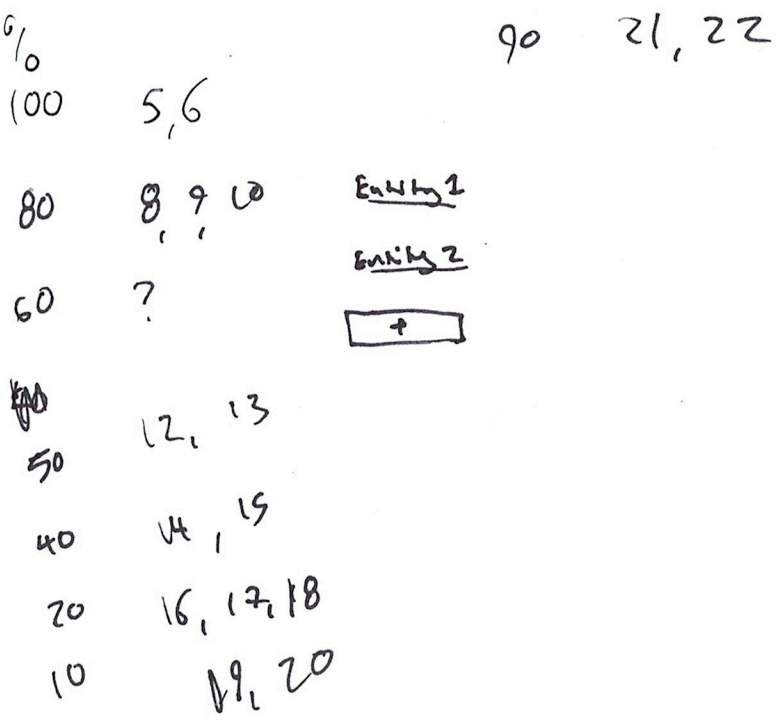


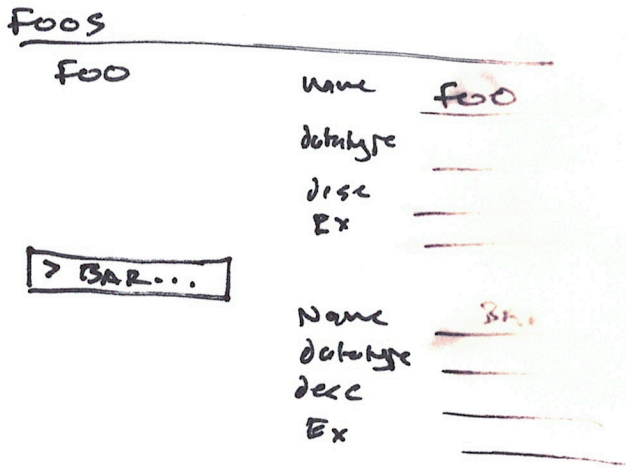


- ins
- outs

- collections
- Things







Social Resources

David: Story ~~Guru~~ Master

- "people need to see themselves in the 'fictional' story / path / map to future you tell"
- old man w/ map ~~Mask~~ more significant than Hero in Hero's journey
- What if Atometa helped people solve problems the way David does (by exciting & motivating ~~other~~ people who are Experts)
- use his butterfly pics as testbed for Atometa Expts

Read outdoor.com piece about David & the story of the gold

Nick Pinkston

Collection:

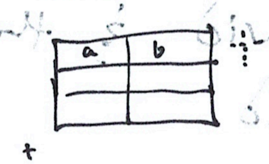
test1 test2 Big

von reese

plane tix

add col

Table



card

a

Name ____

desc ____

type ____

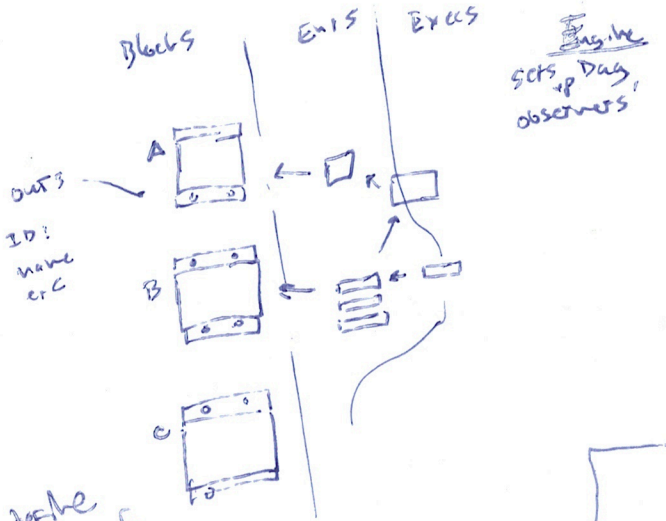
Example ____

Syns ____

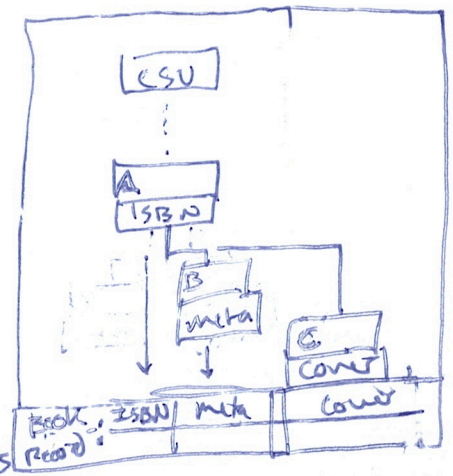
b

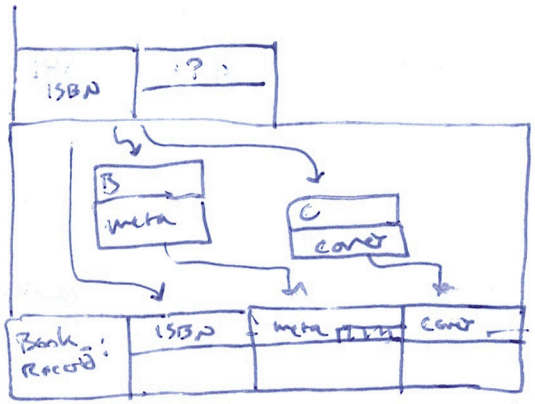
etc

6 Apr 2020



B
↑
C



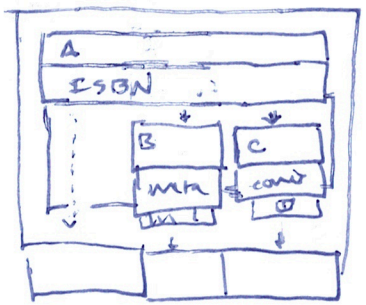


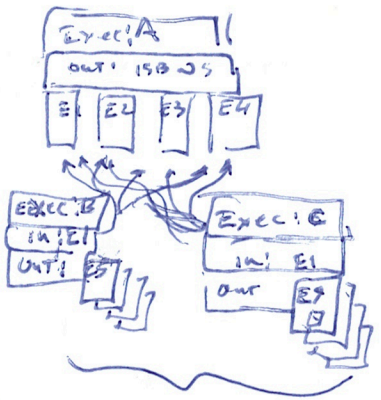
Book_Record:

ISBN

meta
- title
- auth

cover img
- image
- url

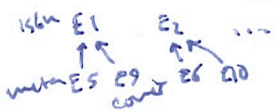




OUTS used to suggest "joins" from Flow

A: _ → ISBN
B: ISBN → meta
C: ISBN → img

IF ISBN from A changes, B, C should recompute



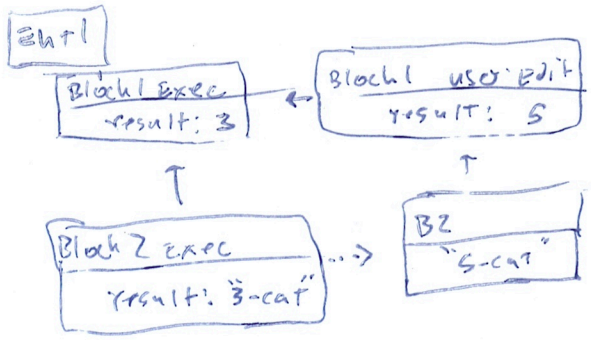
Goal: outline ES architecture for wakenhit

Blocks

- Ents
- specs
- Flow/dag
- Exec
- Branch
- ops / steps

Engine

- Exec
- undo



Start w/ static (json)

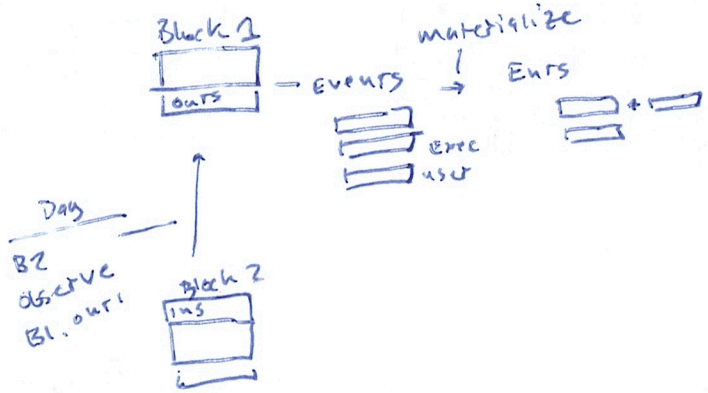
+ Block definitions
+ mock Ents

Engine can:

Exec
→ create/update Ents, Execs

Branch
→ main + draft/preview

undo
→ user cmds
→ roll back preview



101

31 Mar 20

- Sprotty is TS diagram tool (Elk.js?) - check out
  * Factory classes

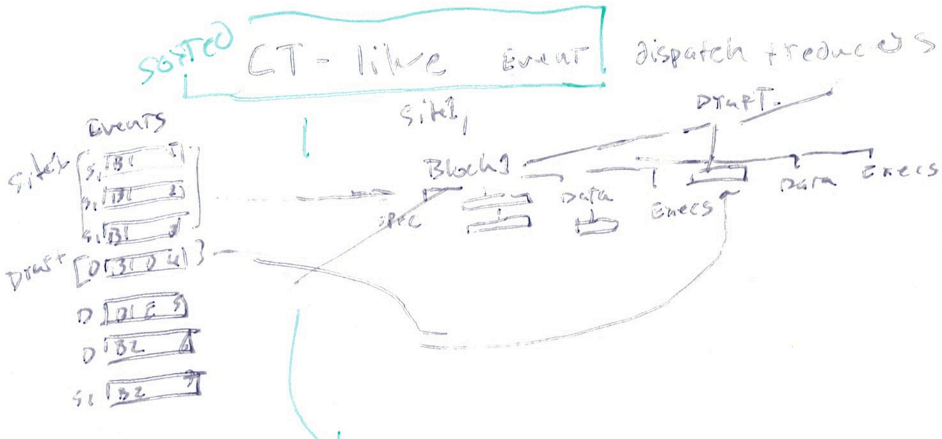
- Rete.js is graphical workflow tool (vue) w/ react plugin
  - workflow nodes define
    Worker (exec)
    +
    Builder (I/O spec)

- Intro to microservices Book
  has JS implementation of ES, projections, aggs

- *** ~~?, Polo swift-mobile~~ swift-client repo
  shows mobx + crdt.js

- Wolkenkit.io
  is node server framework for creating ES / CQRS apps
  - use it to prototype App, then port to custom client only redux / mobx

  or use resolve.js



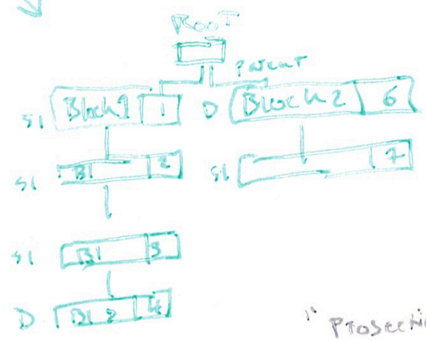


Figure out

mobx
- component
- aggregate
- projection

systems to

1. consume msg/event "stream" / "log"

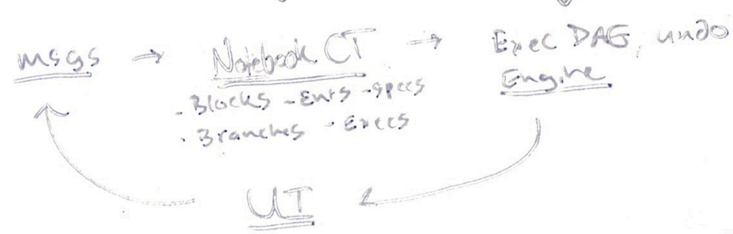
2. organize / dispatch into sorted causal-tree like structure

3. reduce reactively to update views, modulated by UI state

(Draft enabled?)

(Event sourcing terminology)

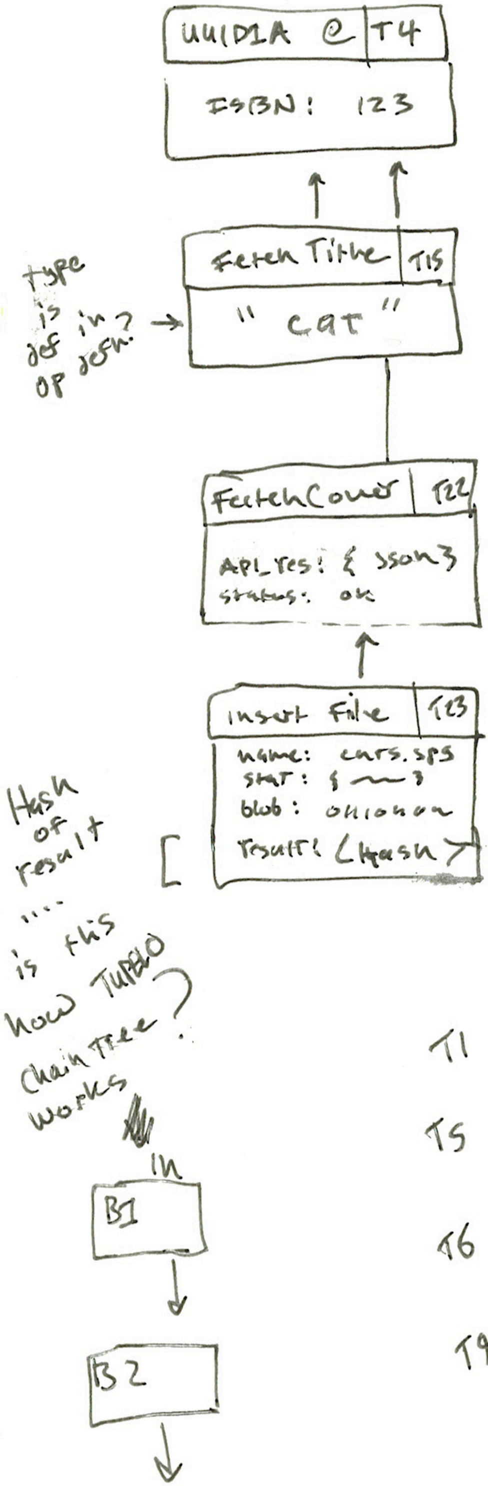
"Projections" "Components"



mobx, redux-saga or mobx-keystone ? or redux-box

Technologies map

Event Sourcing	Redux / redux Box
Handlers	dispatchers
Event	Action
Comp. Aggregate	mutation / reducer
[illegible]	selector

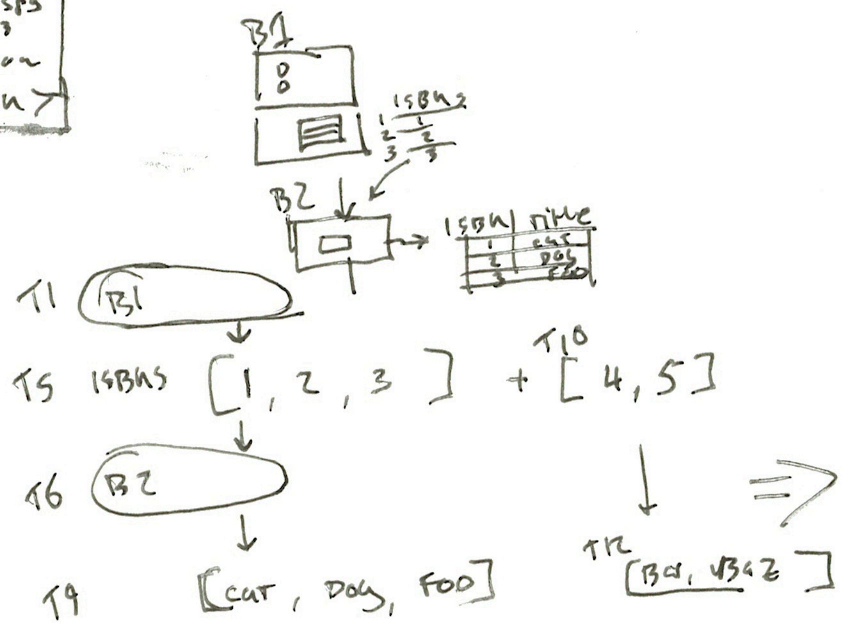


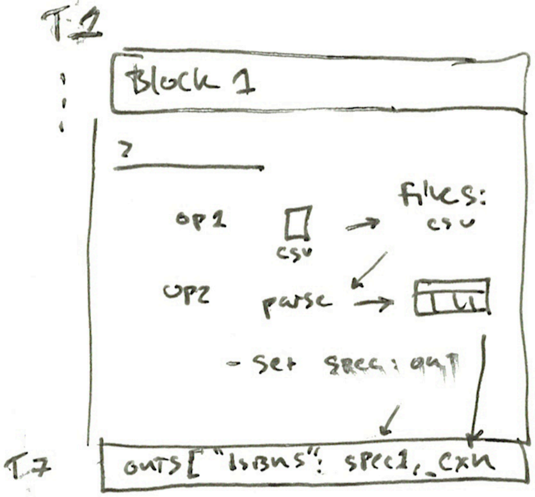
A) Event Sourcing to drive CT / RGA / ORSET CRDT

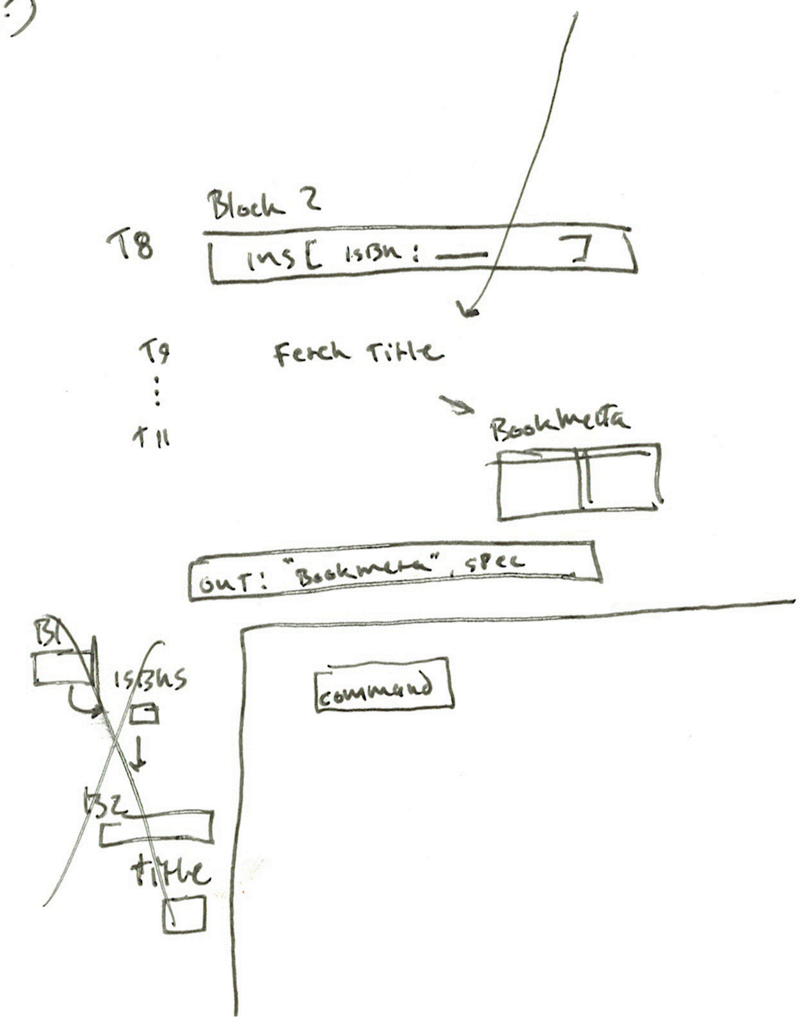
B) Custom Tree driven by ES
- not CRDT

Two interlinked state controllers:

- ES (+ redux) NB + graph API
- Reactive (shadow-Branch? staging?) node Executor + Dep tracker







Msg Store:

Cmds? Evts

Block Aggregate

materializes state from log

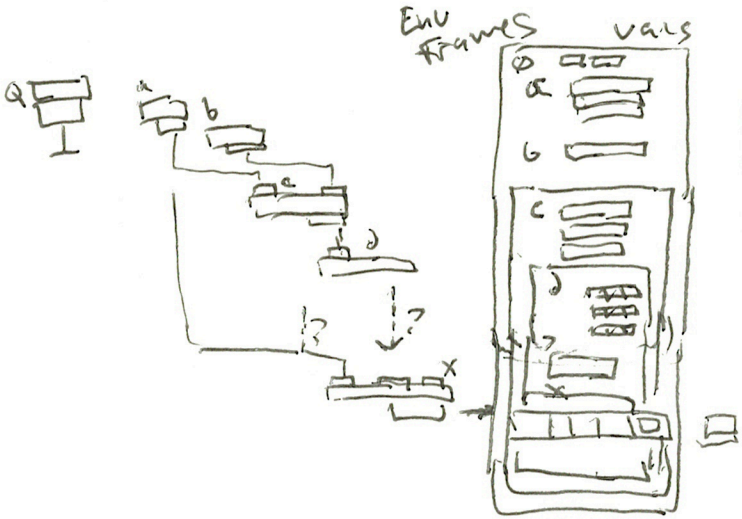
Provides runtime API for Block

Block must can have parent

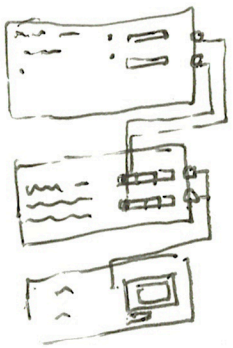
↳ Diff than sibling/ immediate / chain of ancestors providing inputs

The Notebook helps users compose "programs" aka Workflow
- Document
+
- Exec them

NB is interactively watching ins - outs of Blocks & maintains workflow Dag separate from any Execution with instances of Inputs



Linking resources



Workflow REPL

??? How to go from links to feasible workflow

Workflow Runner

Specs

Dag / Dataflow

Workflow Exec

Blocks

Data / Exec

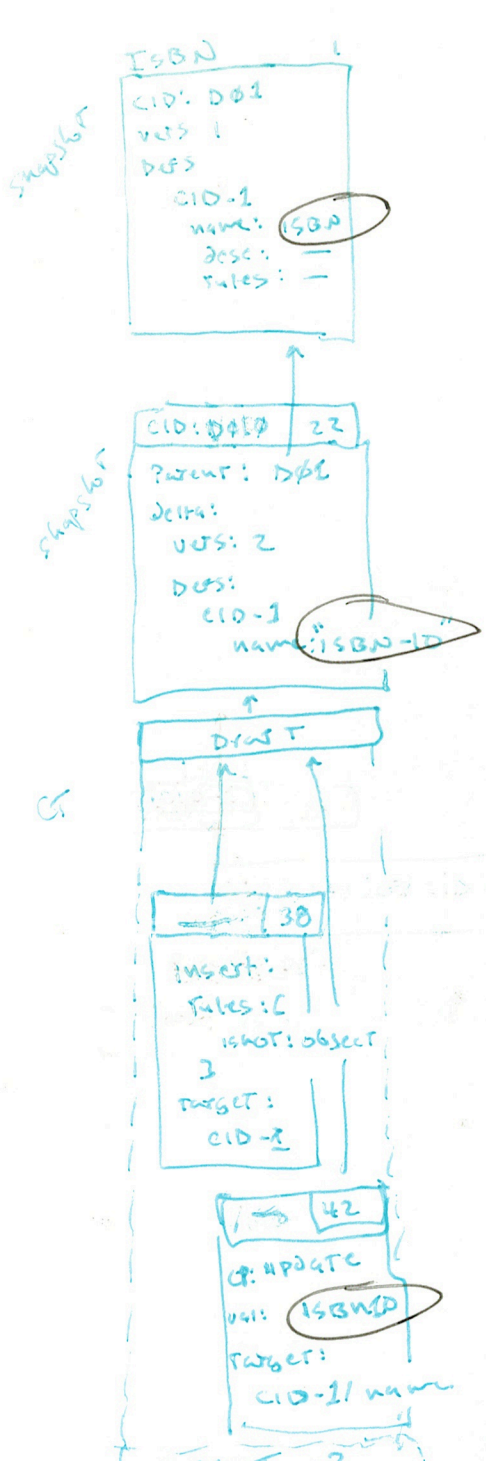
oplog

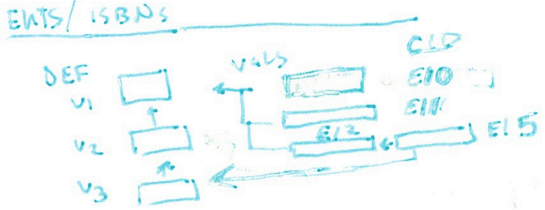
16 MAR 20

DEFS

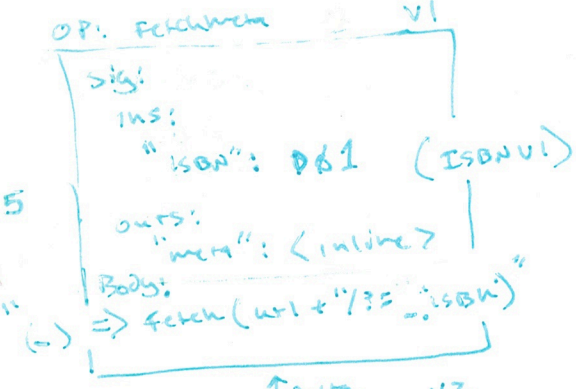
ENTS

OPS

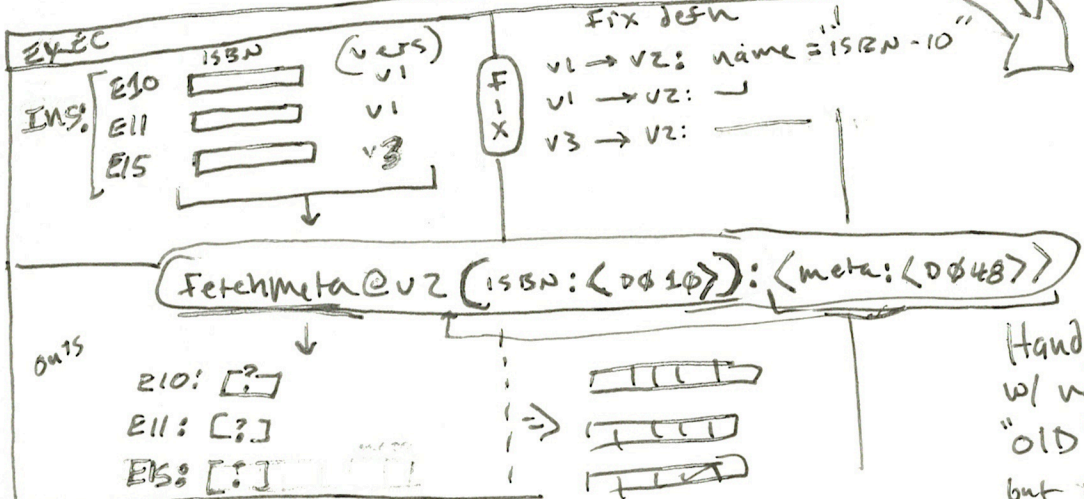




ENT - OP Defn Version Mismatch



ENT's OP use diff versions of DEF (ISBN)



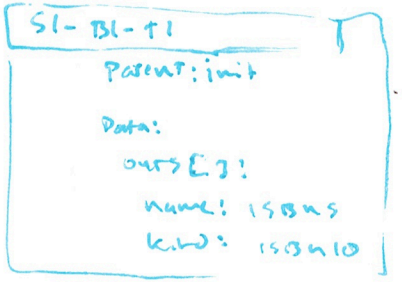
So compute patch BW version in ? version needed, apply on the fly!!

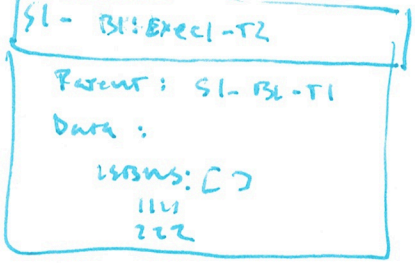
Handy for new data w/ new Defn using "OLD" OP (ancestor Defn)

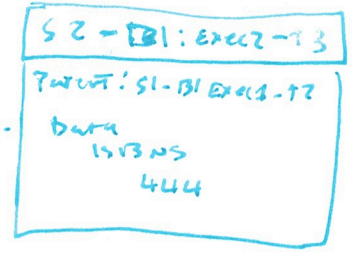
but new OP, old ENT won't TGO

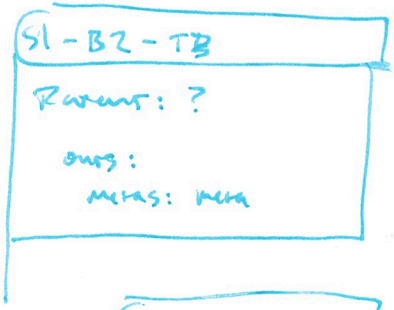
messages [ ]

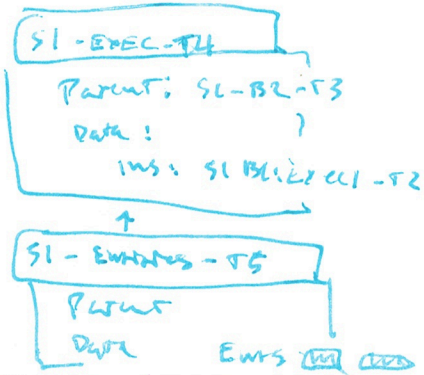
Frame

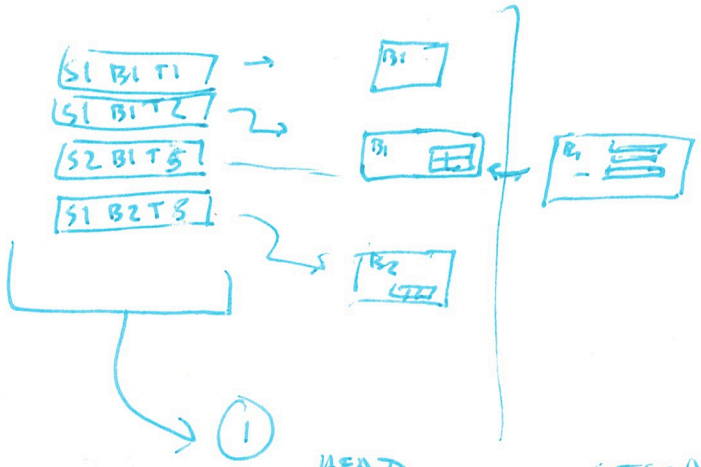












1

HEAD

msgs S: S1

2 collect-B

3 sort T

4 map reduce

Data

5 materialize

Block:
state

6 render

STAGES

S = S2

map B