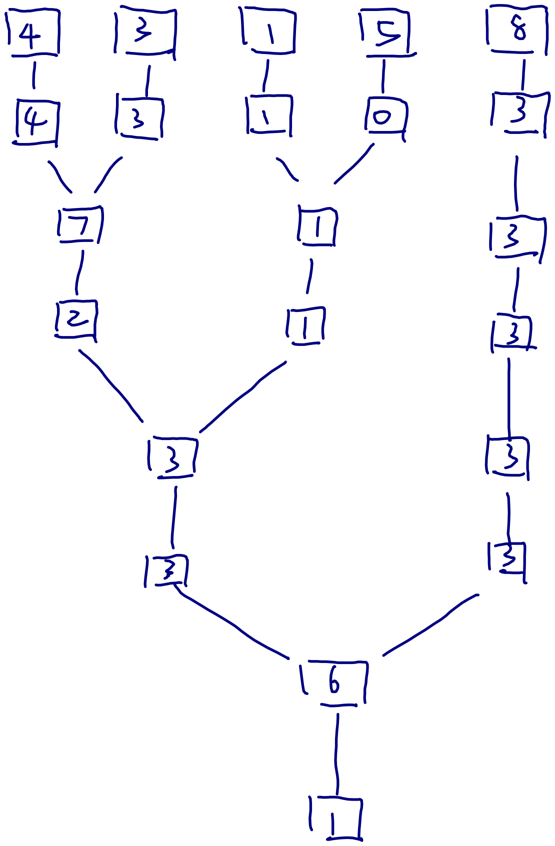
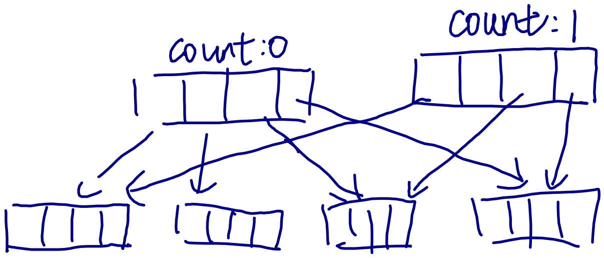
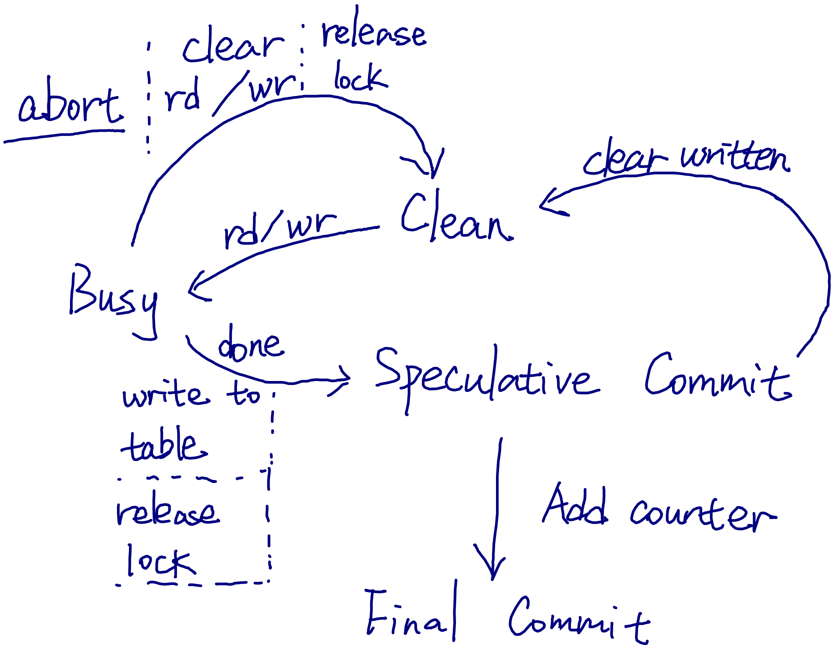

4
3
1
5
8
4
3
1
0
3
7
1
3
2
1
3
3
3
3
3
6
1


count:0
count:1



Tx
- write lock
- read set

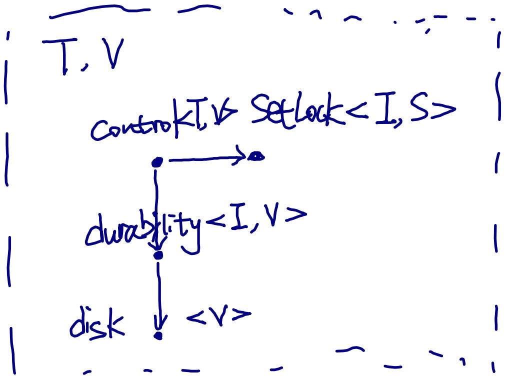
DataBase Managment System

Set <T>

Qry <T> Descriptor of T

Upd <T> Map from T::Id to T

Lock Interface



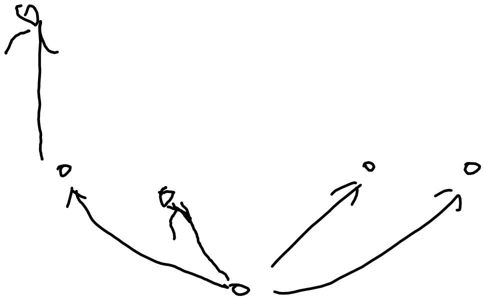
$$x_1 + x_2 + \cdots + x_n = 1$$

$$\hat{x}_i = \varepsilon$$

$$\hat{x}_i = (1 - m\varepsilon) \cdot \frac{x_i}{\sum_{j \in S} x_j}$$

$$1 - m\varepsilon = \sum_{i \in S} \hat{x}_i$$

$$m\varepsilon = \sum_{i \in \{1..n\} \setminus S} \hat{x}_i$$



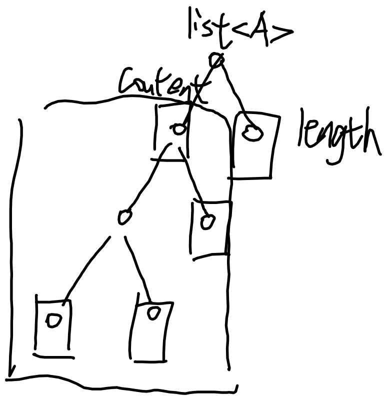
During compile: Tag only

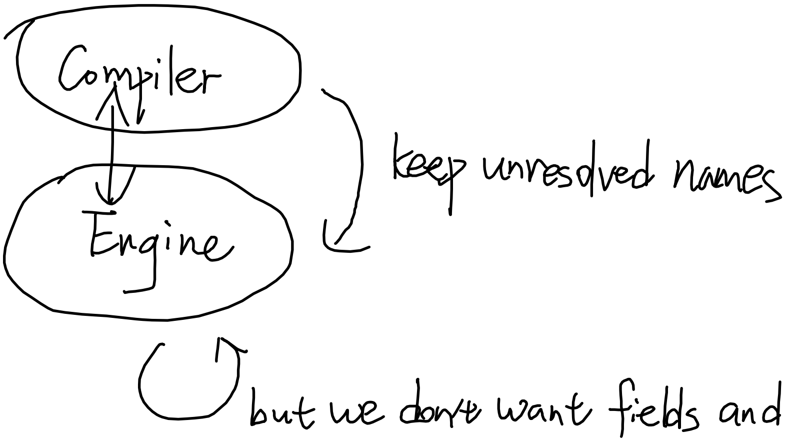
↓

After compile

Pipeline Executor 中怎么表达?

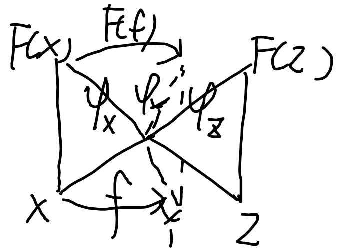
```
F<A, B>
            match {
                _ => ....

            }
```





but we don't want fields and names in engine.

$F : \mathcal{J} \to \mathcal{C}$



X_1

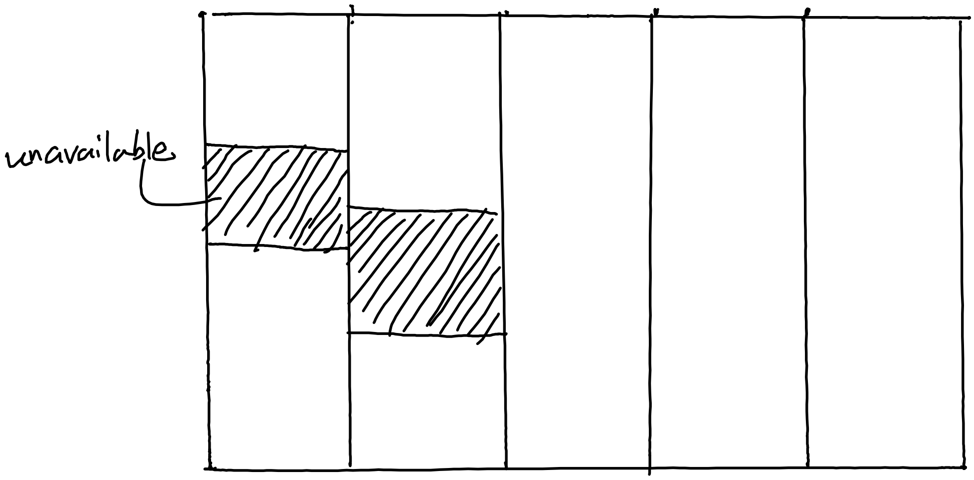
X_2

X_3

X_4

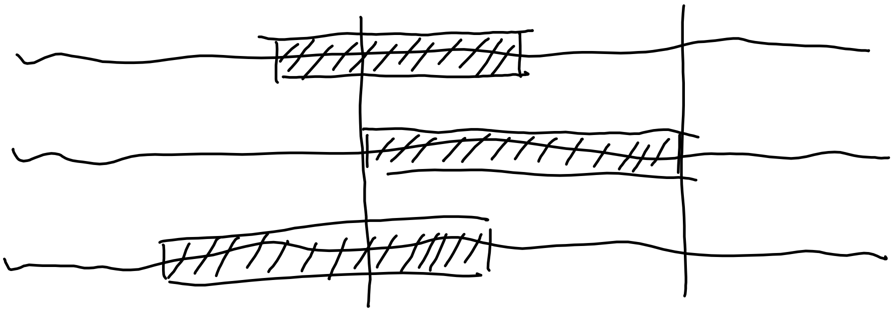
X_5

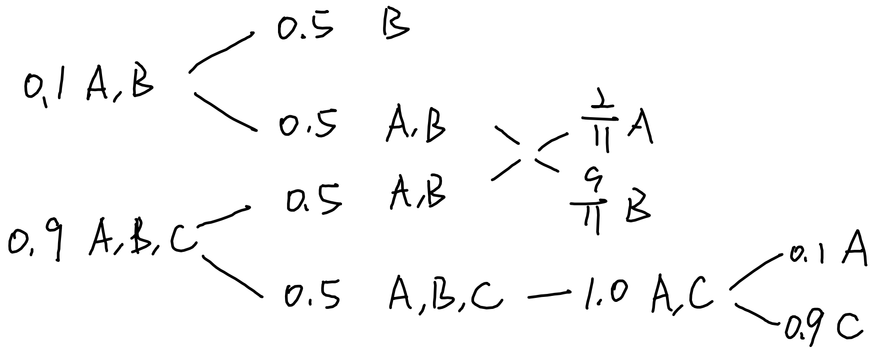
$$\begin{tikzcd}
 & A \arrow[dl] \\
X \arrow[rr] & & Y
\end{tikzcd}$$



① a timeline index for each resource

② a task acquire resources and release resources





$$(0.1 + 0.9) \times 0.5 \times 0.2 + 0.9 \times 0.5 \times 1.0 \times 0.1$$

$$= 0.1 + 0.045$$

$$= 0.145$$

$$P_A = 0.1 + 0.9 \times 0.5 \times P_A$$

$$P_A = \frac{2}{11}$$

The type theory for notiz language

① I need the evaluation to be very efficient.

i.e. For a function term $x \to \underline{\underline{y}}$. $\underline{\underline{\text{This}}}$ syntax tree is traversed only once for subst x.

which means no reduce happen unless y has an "endpoint" type.

To keep track of values, we use a "Stuck Variant" to store the value and removed abstraction layers.

↳ This is a bad idea. Because we have tuples.

② $\underline{\underline{\text{CPS}}}$ to rescue

↳ Continuation Pass Style

Idea: for each term, find a way to represent "the next step"

Goal: for each term M, convert it to a term $CPS[M]$,
such that $CPS[M, k]$ means when k is eventually called its argument $\equiv M$.

$$CPS[(M\ N), k] := CPS[N, CPS[M, k]]$$

$$CPS[\lambda x . M, k] := \lambda x .\ CPS[M, k]$$

$$CPS[N, \lambda x . CPS[M, k]] \equiv CPS[M[x/N], k]$$

This will cause an ever-growing stack.

So no ✗

③ De Brujin with move

Define "last usage" of a variable, use movement

Case 1:

$$\lambda x.(\underset{\underset{x}{\uparrow}}{\underset{?}{M}}\ \underset{\underset{x}{\uparrow}}{\underset{\sim}{N}})$$

then "last usage" is in N
(M is evaluated first)

Case 2:

$$\lambda x.(\underset{\underset{x}{\uparrow}}{\underset{\sim}{M}}\ N)$$

then "last usage" is in M

Case 3:

$$\lambda x.(\lambda y.\ \underset{\underset{x}{\uparrow}}{\underset{\sim}{M}})$$

then "last usage" is in M

④ Composition

1. Modules are tree shaped (like rust)
2. Module-level parameters: different params results in different views when compiling standalone modules (code analysis).
3. Modules are imported and included with file path.
4. Module attributes are constructed using export statement (top level, "only once", same for mod-params)
5. "top level" is defined during lowering (before evaluation)

⑤ Defining Types

Trouble: Consider evaluating function $f: A \times B \to C$

The problem is $\overset{\text{or } A}{\overset{\downarrow}{B}}$ is not necessarily binded in for example

$\lambda x.(f\ (a\ x))$, we shift x to stack,

but where to store a.

The evaluationshould be designed against it.

Allow enum/tuple/struct types.

Markups will be encoded into these types.

Build core terms from face sytax.