Much Ado About ⌘Z

Alessandro Warth
Viewpoints Research

STEPS

... toward the reinvention of programming

The STEPS Project

- **Goal** - To create a highly useful end-user system including:
  - operating system
  - programming environment
  - “applications”
  - graphics, sound, ...

The STEPS Project

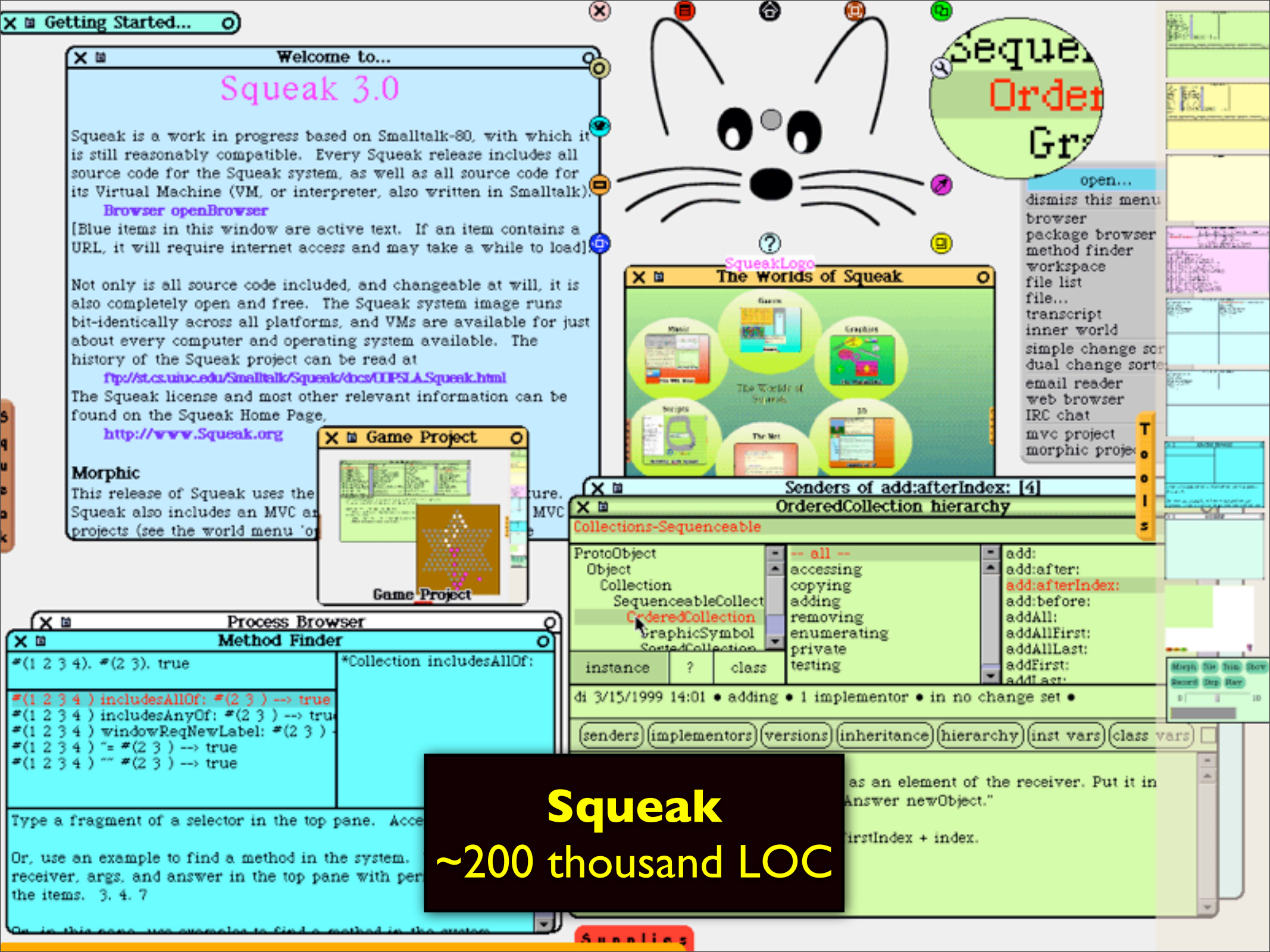
- **Goal** - To create a highly useful end-user system including:
  - operating system
  - programming environment
  - "applications"
  - graphics, sound, ...

} *personal computing*


... in
under
20,000
LOC!


... in
under
20,000
LOC!


Picture Tasks
View as a slide show
Order prints online
Print pictures
Copy all items to CD
Shop for pictures online
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
Shared Pictures
My Pictures
My Computer
My Network Places
Details
Sample Pictures
File Folder
Date Modified: Friday, January 24, 2003, 3:06 AM
Sample Music
Music Tasks
Play all
Beethoven's Symphony No. 9 (Scherzo)
Ludwig van Beethoven, comp...
New Stories (Highway Blues)
Marc Seales, composer. New ...
Speakin' Out
Control Panel
Switch to Classic View
See Also
Windows Update
Help and Support
Other Control Panel Options
Pick a category
Appearance and Themes
Printers and Other Hardware
Network and Internet Connections
User Accounts
Add or Remove Programs
Date, Time, Language, and Regional Options
Sounds, Speech, and Audio Devices
Accessibility Options
Performance and Maintenance
Security Center
Windows XP
~40 million LOC
start
Sample Pictures
Sample Music
Control Panel
11:28 AM


Getting Started...
Welcome to...
Squeak 3.0
Squeak is a work in progress based on Smalltalk-80, with which it is still reasonably compatible. Every Squeak release includes all source code for the Squeak system, as well as all source code for its Virtual Machine (VM, or interpreter, also written in Smalltalk).
Browser openBrowser
[Blue items in this window are active text. If an item contains a URL, it will require internet access and may take a while to load]
Not only is all source code included, and changeable at will, it is also completely open and free. The Squeak system image runs bit-identically across all platforms, and VMs are available for just about every computer and operating system available. The history of the Squeak project can be read at
ftp://st.cs.uiuc.edu/Smalltalk/Squeak/docs/OOPSLA.Squeak.html
The Squeak license and most other relevant information can be found on the Squeak Home Page,
http://www.Squeak.org
Morphic
SqueakLogo
The Worlds of Squeak
open...
dismiss this menu
browser
package browser
method finder
workspace
file list
file...
transcript
inner world
simple change sor
dual change sorte
email reader
web browser
IRC chat
mvc project
morphic proje
Tools
Game Project
Game Project
Senders of add:afterIndex: [4]
OrderedCollection hierarchy
Collections-Sequenceable
ProtoObject
Object
Collection
SequenceableCollect
OrderedCollection
GraphicSymbol
SortedCollection
-- all --
accessing
copying
adding
removing
enumerating
private
testing
add:
add:after:
add:afterIndex:
add:before:
addAll:
addAllFirst:
addAllLast:
addFirst:
addLast:
instance
?
class
di 3/15/1999 14:01 • adding • 1 implementor • in no change set •
senders
implementors
versions
inheritance
hierarchy
inst vars
class vars
Process Browser
Method Finder
#(1 2 3 4). #(2 3). true
*Collection includesAllOf:
#(1 2 3 4) includesAllOf: #(2 3) --> true
#(1 2 3 4) includesAnyOf: #(2 3) --> tru
#(1 2 3 4) windowReqNewLabel: #(2 3)
#(1 2 3 4) = #(2 3) --> true
#(1 2 3 4) ~= #(2 3) --> true
Type a fragment of a selector in the top pane. Acce
Or, use an example to find a method in the system.
receiver, args, and answer in the top pane with per
the items. 3. 4. 7
as an element of the receiver. Put it in
Answer newObject."
firstIndex + index.
Squeak
~200 thousand LOC

Why?

- “Put people in charge of their own software destinies”
  - Can’t understand 40,000,000 LOC (an entire library!)
  - Can “own” 20,000 LOC (one 400-page book)

3 C 006-04766

STEREO

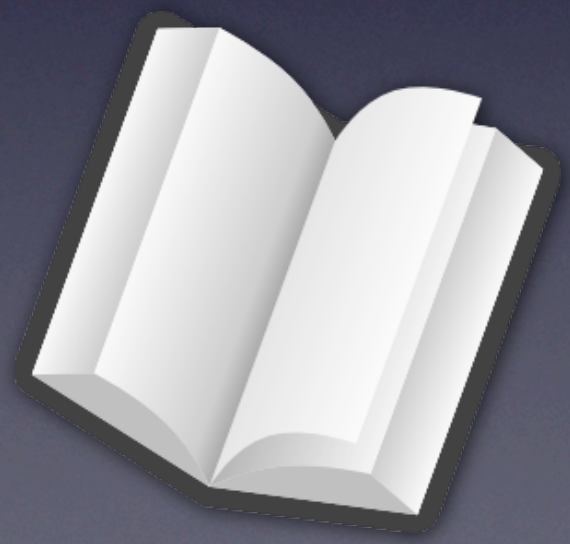
JOHN LENNON
PLASTIC ONO BAND

POWER TO THE PEOPLE

APPLE

Why? (cont'd)

- Didactic value!
- Curriculum for univ. students to learn about powerful ideas, building complex systems...
  - May even be useful at high-school level



The Path to 20K LOC

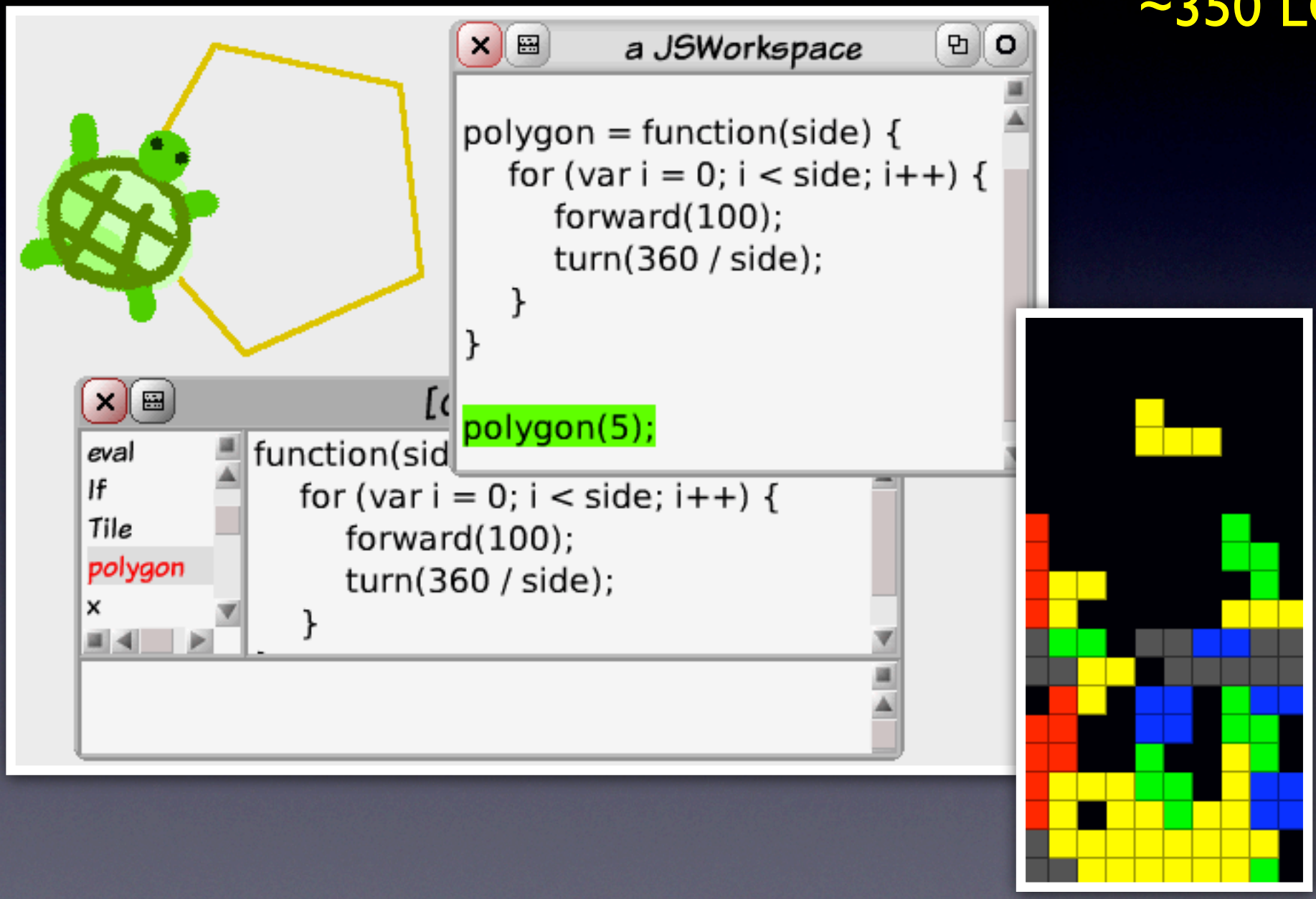
- Experimenting w/ new...
  - abstractions
  - PLs
  - DSLs

OMeta

Experimenting w/
Programing Languages

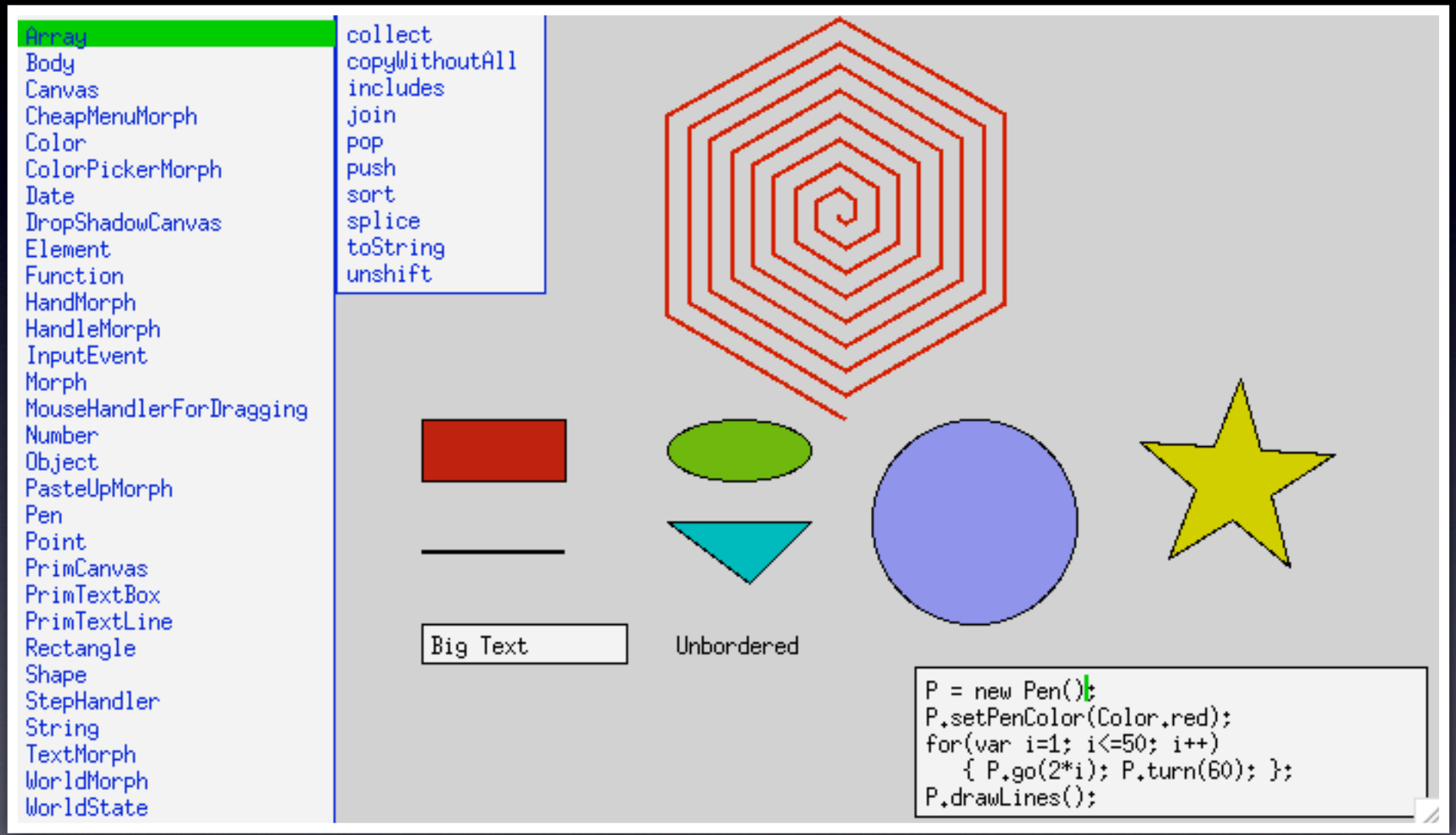
JavaScript (OMeta/Squeak)

~350 LOC



Sun's Lively Kernel (OMeta/COLA)

~300 LOC



Toylog (OMeta/Squeak)

- Get children interested in logic!
- Front-end to Prolog, runs on Squeak
- ~70 LOC

```
Homer is Bart's father.
Marge is Bart's mother.
x is y's parent if x is y's father or
                   or x is y's mother.
Homer is not bart's parent.
Marge is bart's parent.
x is Bart's parent?
```

Toylog (OMeta/Squeak)

- Get children interested in logic!
- Front-end to Prolog, runs on Squeak
- ~70 LOC

```
Homer is Bart's father.
Marge is Bart's mother.
x is y's parent if x is y's father or
                   or x is y's mother.
Homer is not bart's parent.
Marge is bart's parent.
x is Bart's parent?
```

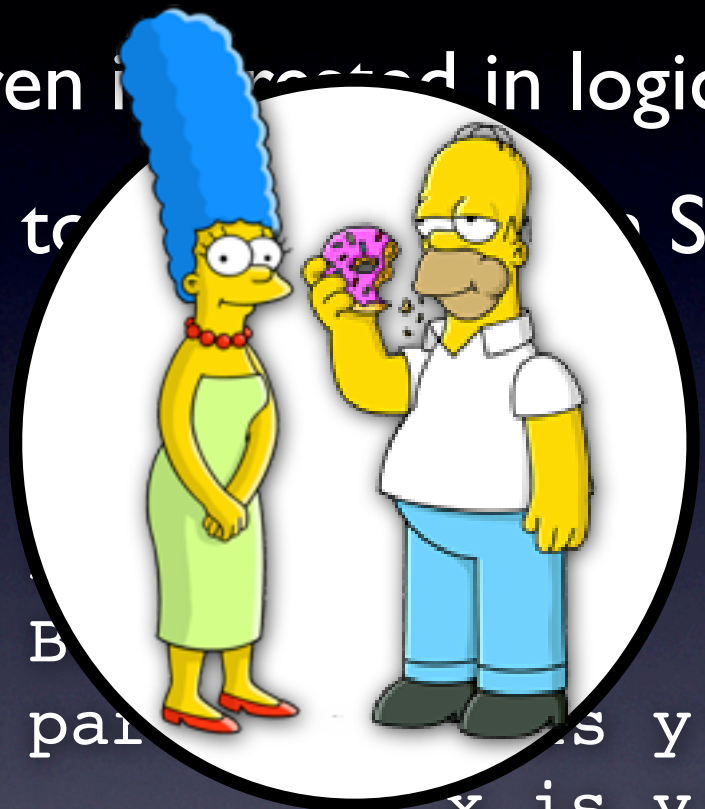
Toylog (OMeta/Squeak)

- Get children interested in logic!
- Front-end to Prolog, runs on Squeak
- ~70 LOC

```
Homer is Bart's father.
Marge is Bart's mother.
x is y's parent if x is y's father or
                   or x is y's mother.
Homer is not bart's parent.
Marge is bart's parent.
x is Bart's parent?
```

Toylog (OMeta/Squeak)

- Get children i[illegible]rested in logic!
- Front-end to[illegible] Squeak
- ~70 LOC



```
Homer is [illegible]
Marge is B[illegible]
x is y's par[illegible]s y's father or
                    [illegible] x is y's mother.

Homer is not bart's parent.
Marge is bart's parent.
x is Bart's parent?
```

Prolog - OMeta/JS 2.0 Workspace

important info: (nevermind)

go to project: (nevermind)

previous versions of this project

Instructions +/-

Play Area +/-

Source +/-

```
ometa PrologTranslator <: Parser {
  variable = spaces firstAndRest(`upper, `letterOrDigit):name  -> new Var(name.join('')),
  symbol   = spaces firstAndRest(`lower, `letterOrDigit):name  -> new Sym(name.join('')),
  clause   = symbol:sym "(" listOf(`expr, ','):args ")"        -> new Clause(sym, args),
  expr     = clause | variable | symbol,
  clauses  = listOf(#clause, ','),
  rule     = clause:head ":-" clauses:body      "."            -> new Rule(head, body)
           | clause:head                        "."            -> new Rule(head, []),
  prog     = (rule:r &clause -> r)*:rs clause:q "." spaces end -> {rules: rs, query: q}
}

translateCode = function(x) {
  var prog = PrologTranslator.matchAll(x, "prog")
  solve(prog.query, prog.rules)
}

nat(z).
nat(s(X)) :- nat(X).
nat(X).
```

~90 LOC

print it (ctrl+p) do it (ctrl+d) save it (ctrl + s)

Portable Programming Language Prototypes!


Wii
Wii


Forget Guitar Hero... I could be the next Dan Ingalls!
OMeta/JS


I
Z

Undo

- An important feature in most applications
- Not just about fixing mistakes:
  enables *exploration* w/o fear
  - learn by trying things out
    (errors not a big deal)
  - tool for experimenting w/ different
    choices

SURE_

You're right in liking MEAT

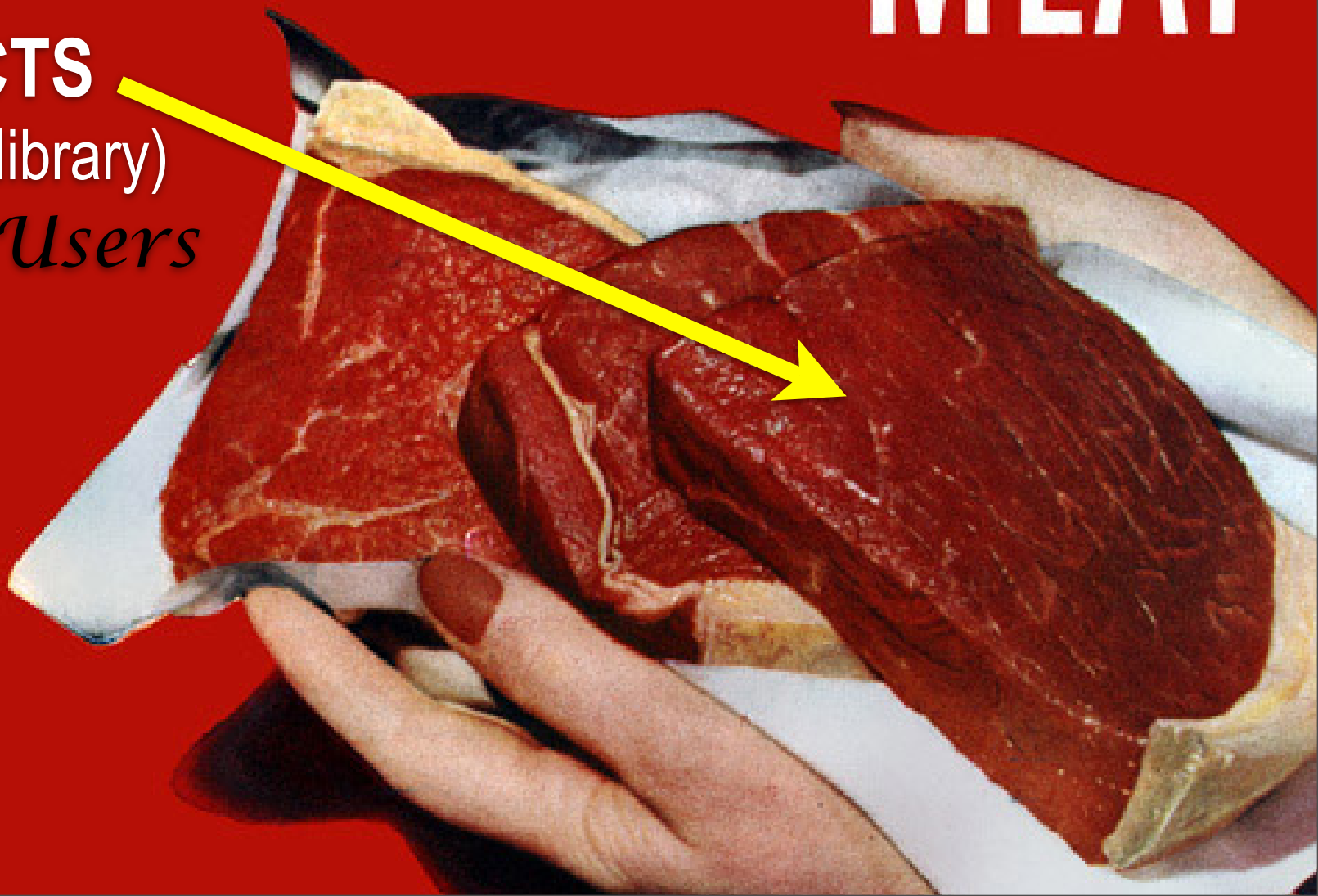
SURE_

You're right in liking MEAT

UOBJECTS

(framework / library)

Undo for Users

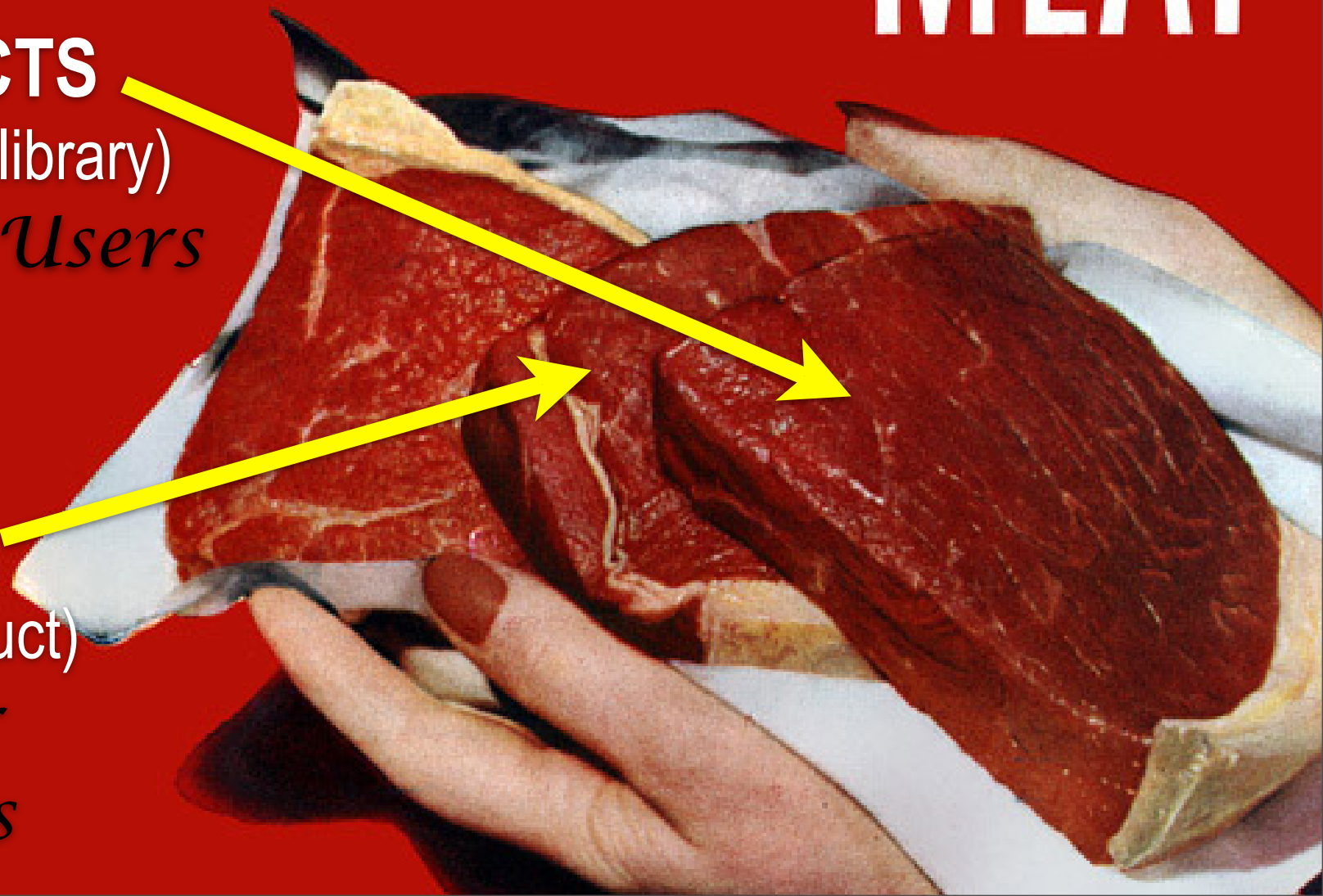


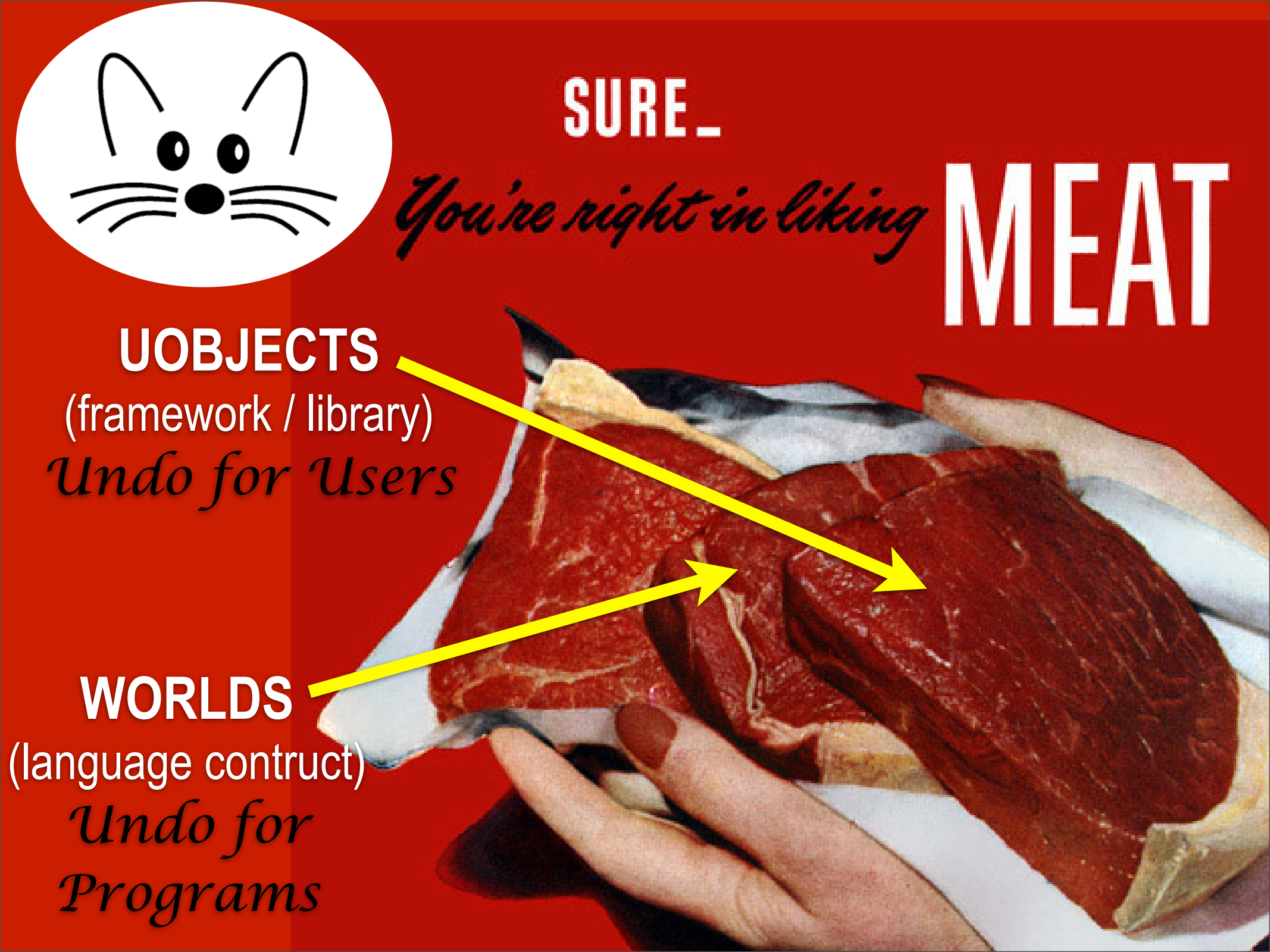
SURE_

You're right in liking MEAT

UOBJECTS
(framework / library)
Undo for Users

WORLDS
(language contruct)
Undo for Programs



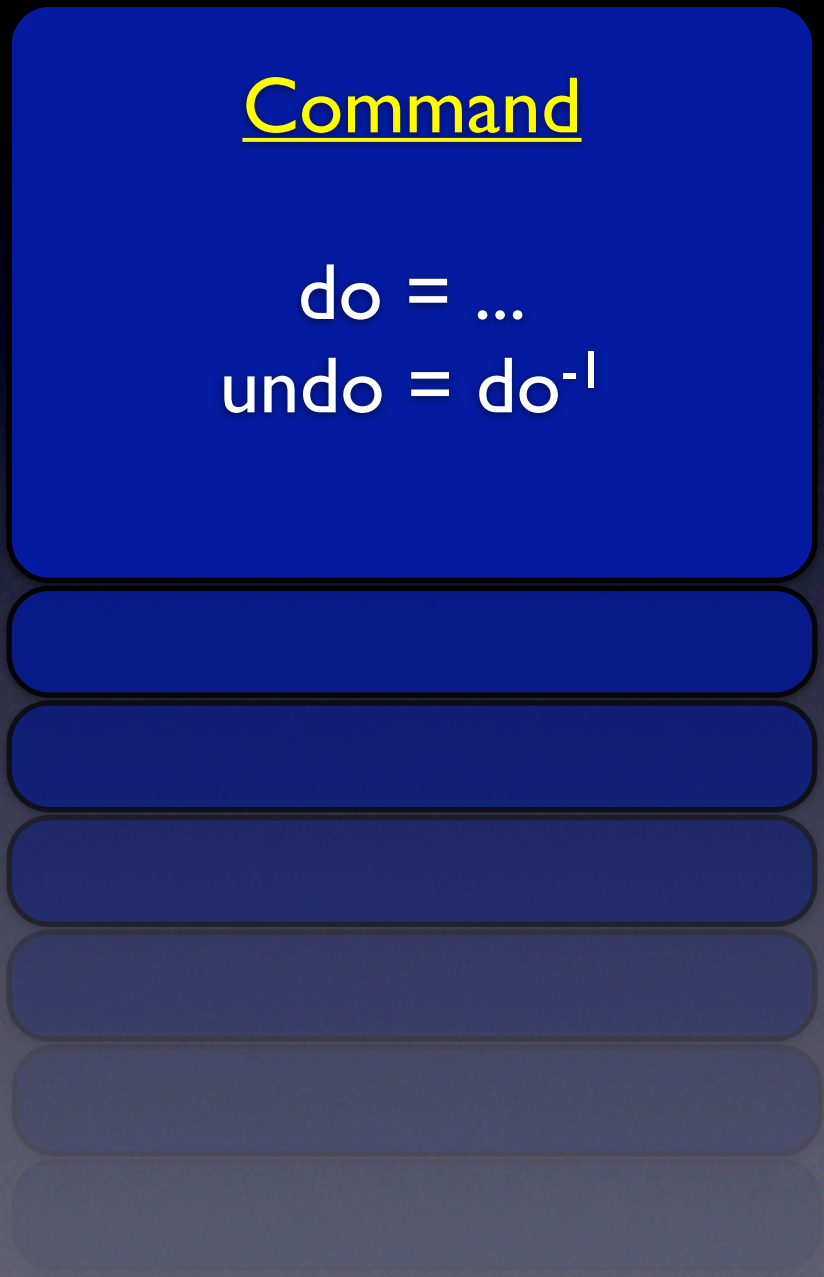

SURE_
You're right in liking
MEAT
UOBJECTS
(framework / library)
Undo for Users
WORLDS
(language contruct)
Undo for
Programs

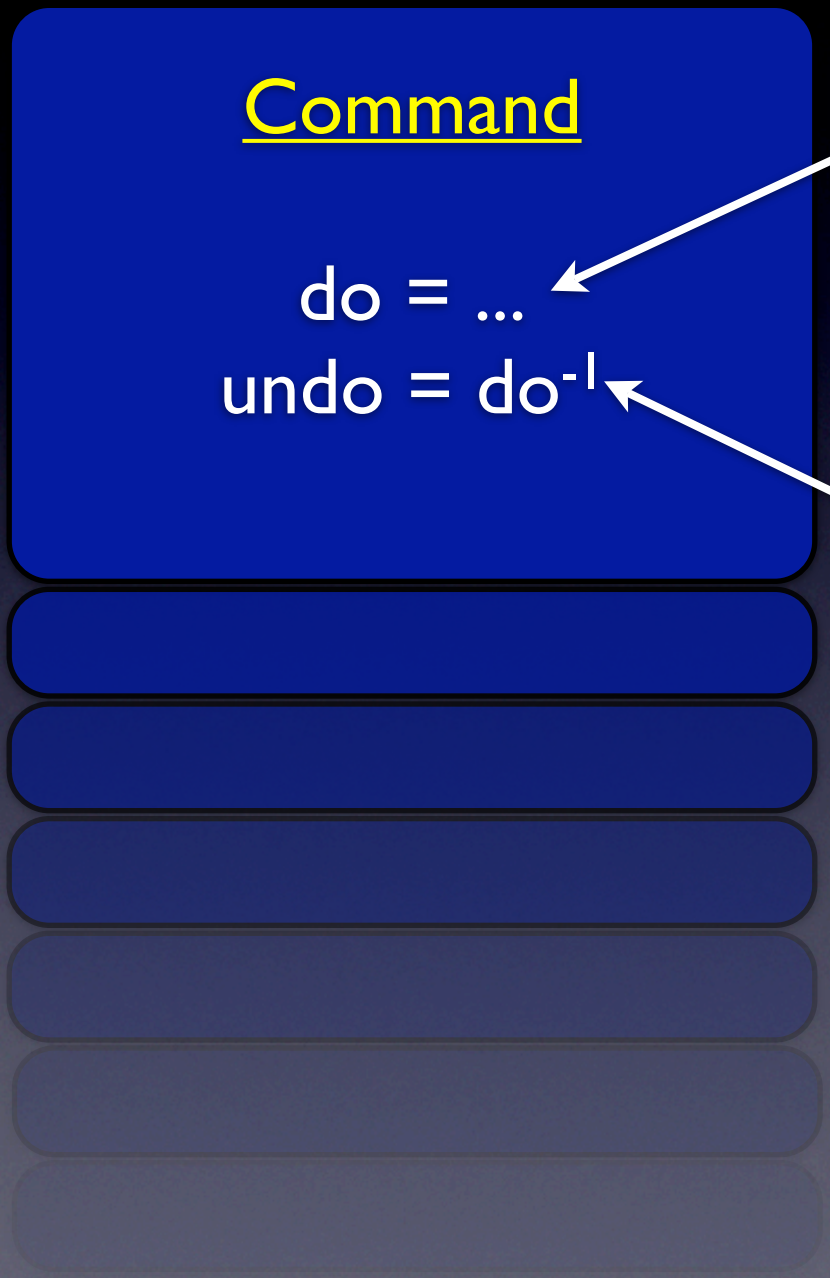
Part I

UObjects:
Undo for Users

The Status Quo

- Most apps support *linear undo*
- ... which is implemented using:
  - *command* design pattern
  - *memento* design pattern


Command
do = ...
undo = do-1


Command
do = ...
undo = do-1
may throw away info
... need to keep it for undo

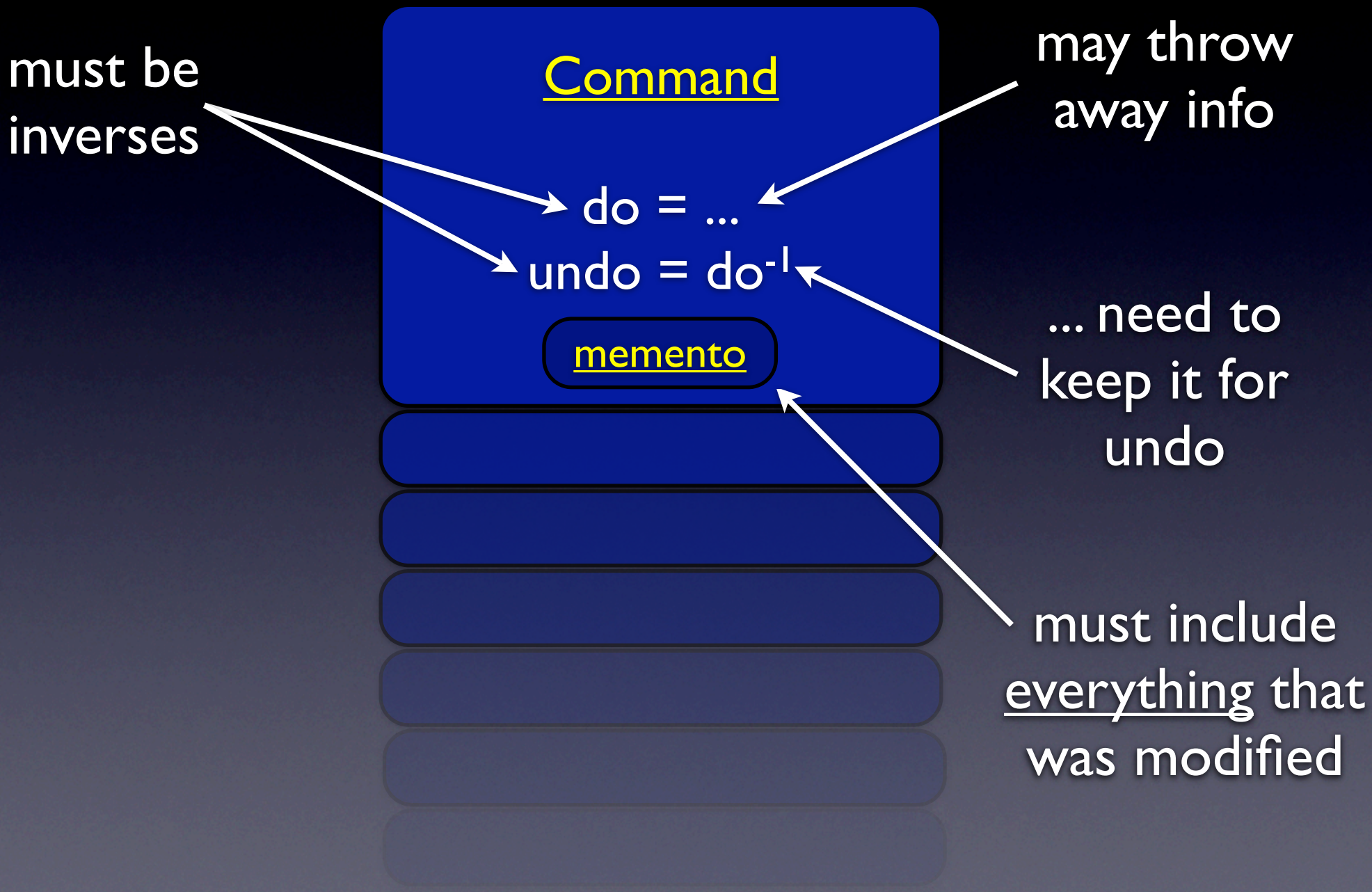
may throw away info

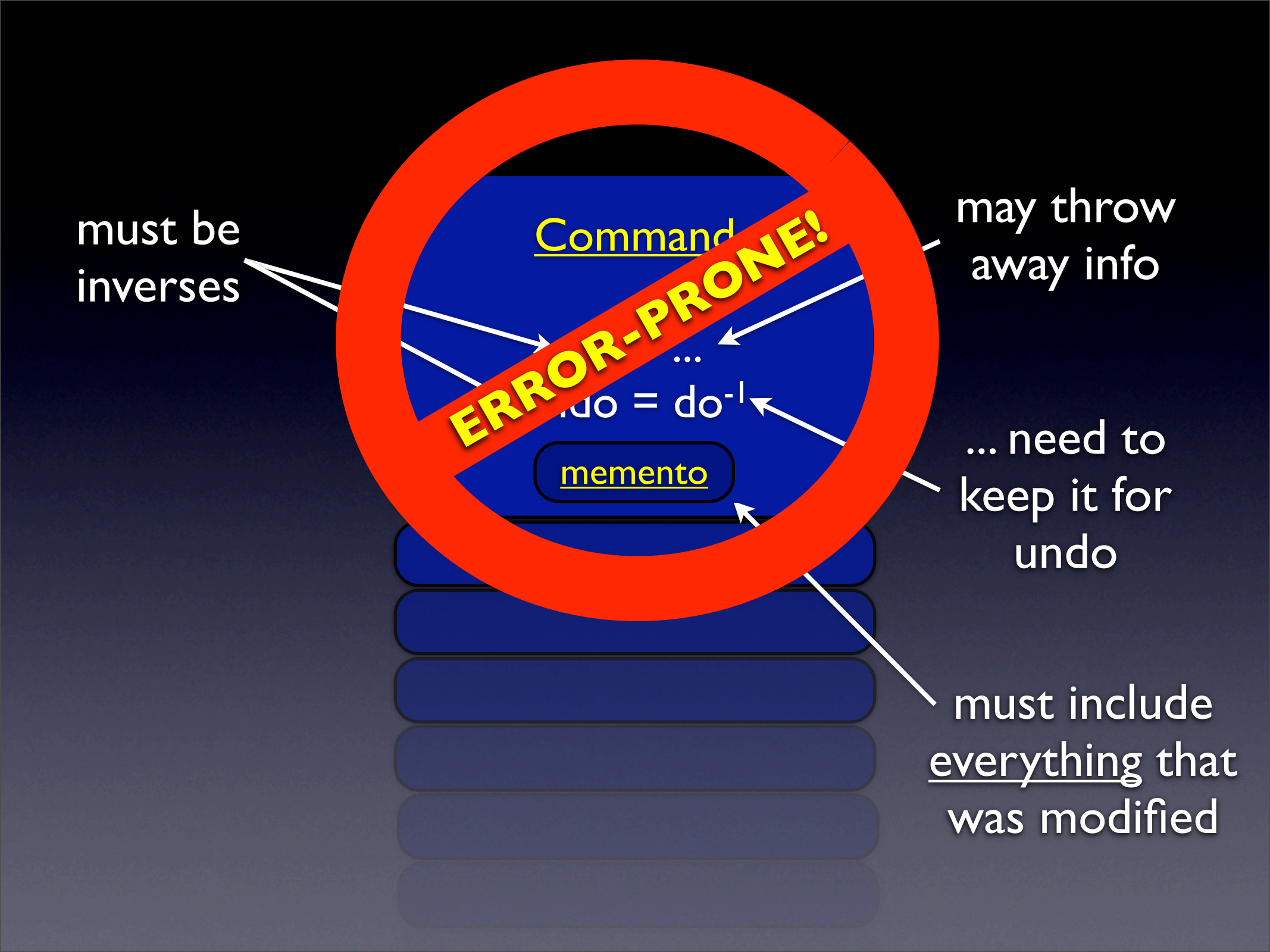
... need to keep it for undo

may throw away info

... need to keep it for undo

must include everything that was modified


must be inverses
Command
do = ...
undo = do-1
memento
may throw away info
... need to keep it for undo
must include everything that was modified


must be inverses
Command
ERROR-PRONE!
...
do = do-1
memento
may throw away info
... need to keep it for undo
must include everything that was modified

Proposed Approach

- Why not generate memento on the fly?
  - i.e., record original values of all variables modified
  - (which may belong to mutliple objects)
- Undo writes old values back into object(s)
- No need for error-prone idiom

Programming Model

- UObject — Undoable Object
  - operations: {#at, #at:put:, ...}
  - may only be modified inside...
- UTransaction — Undoable Transaction
  - may modify any no. of UObjects
  - operations: {#undo}

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'old'

obj2
bar is 5

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'old'

obj2
bar is 5

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'old'

obj2
bar is 5

t1

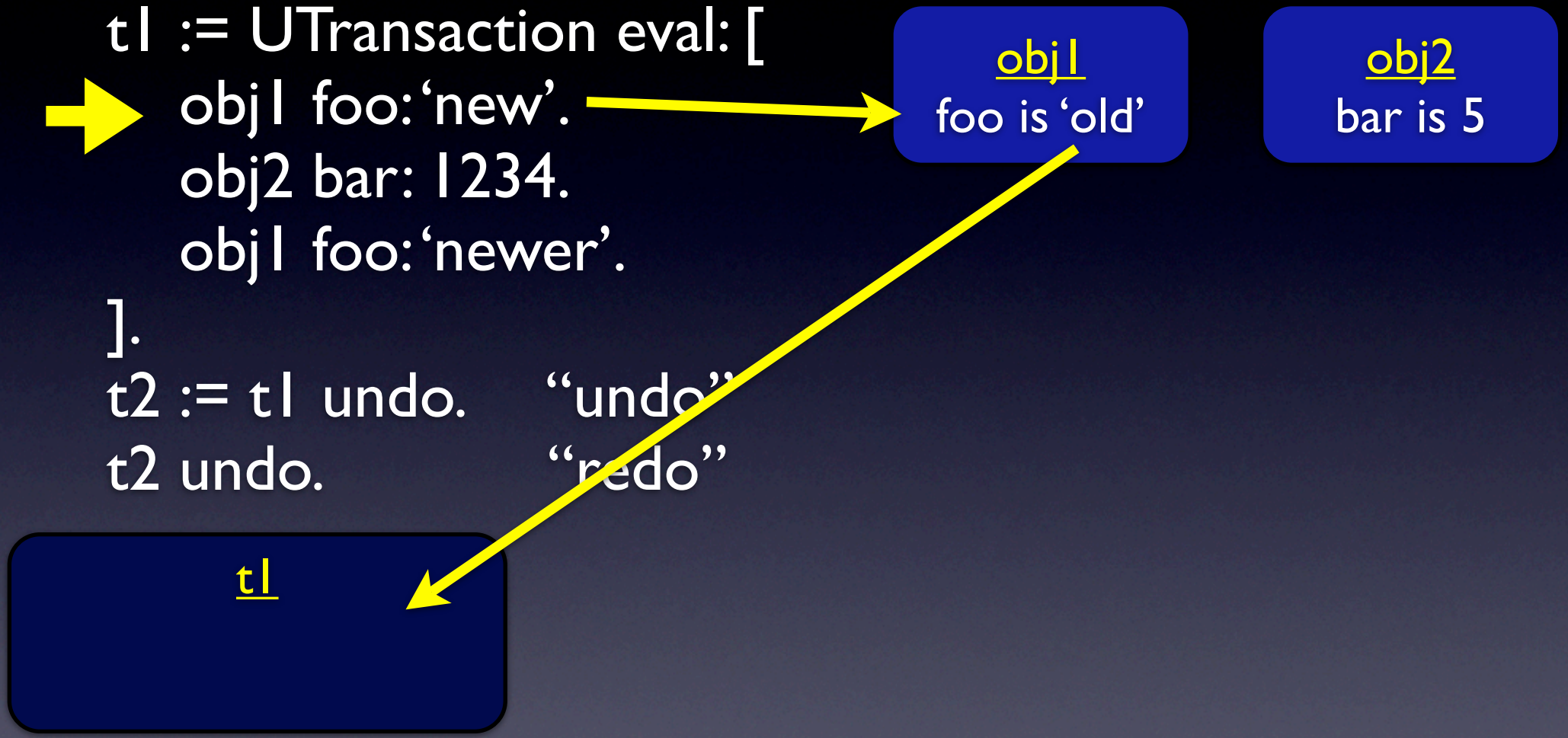
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'old'

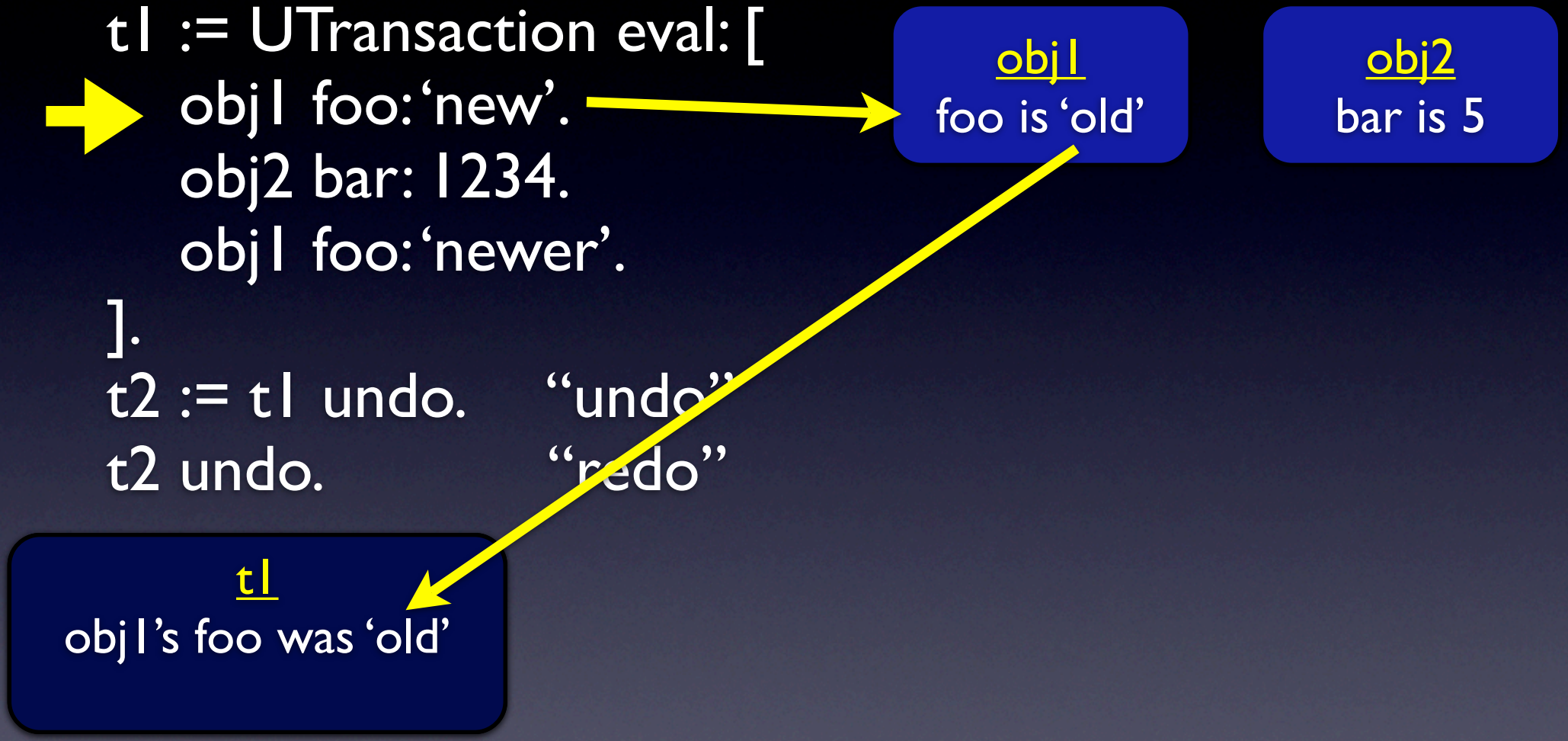
obj2
bar is 5

t1

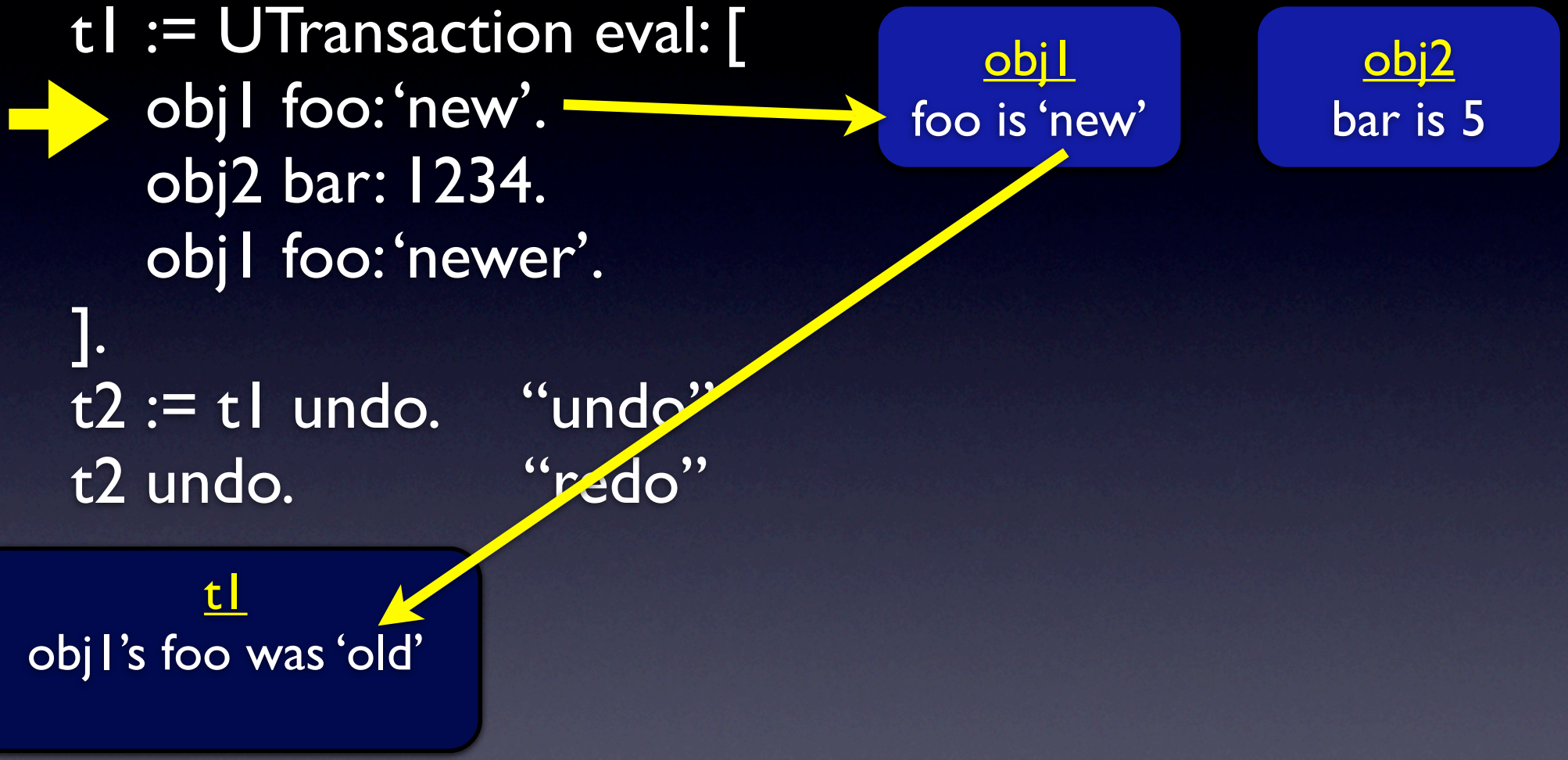
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



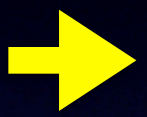
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

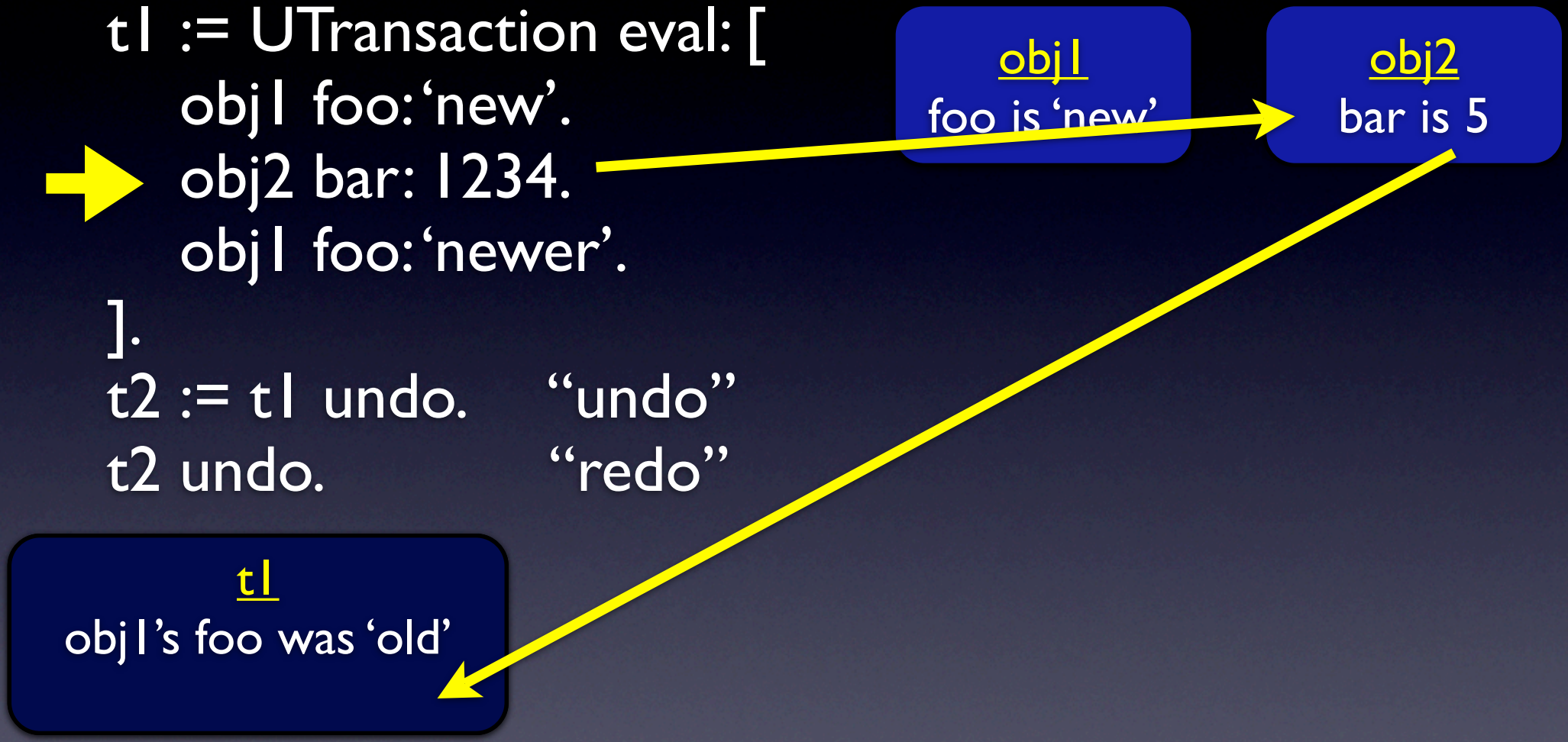


obj1
foo is 'new'

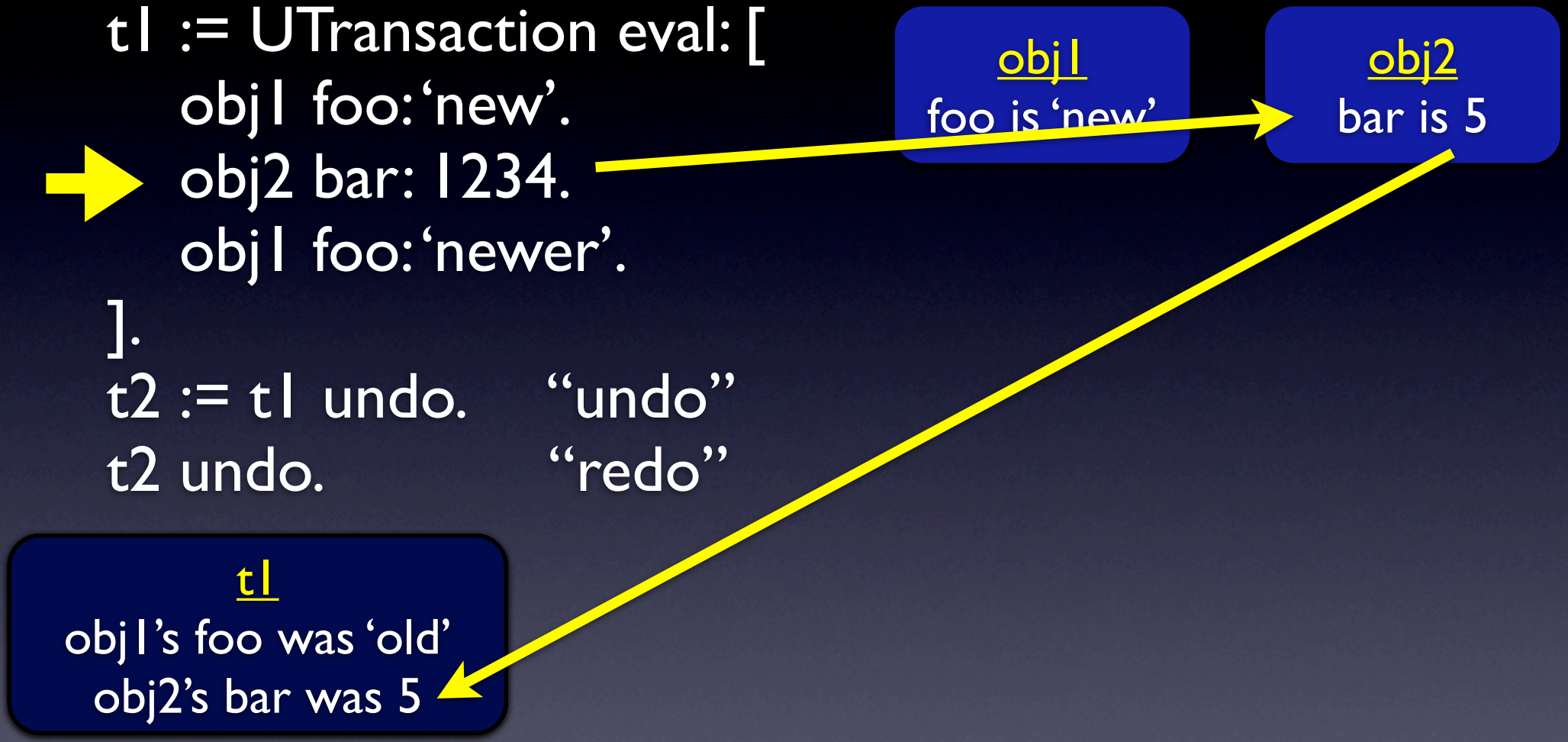
obj2
bar is 5

t1
obj1's foo was 'old'

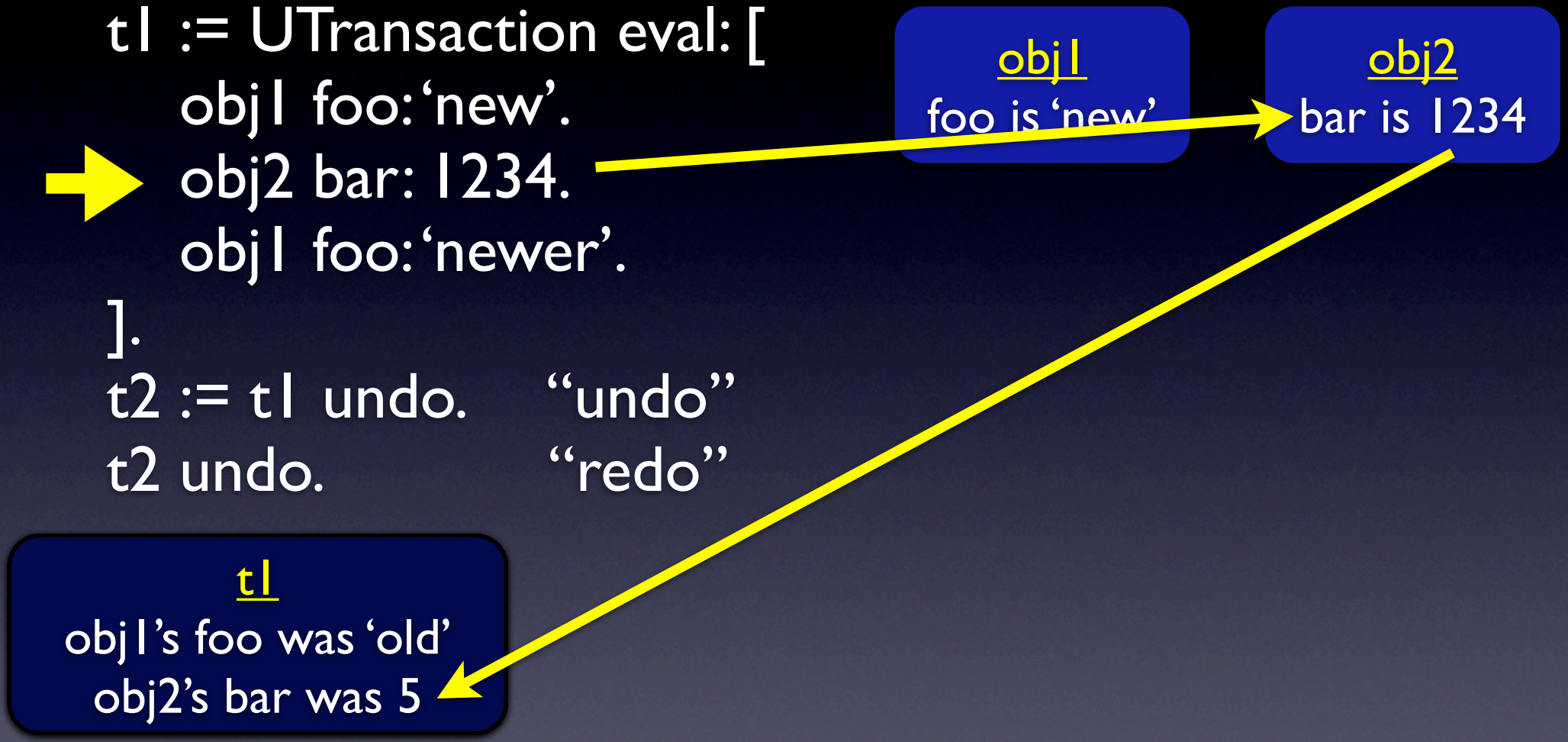
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



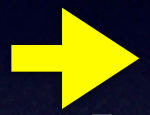
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

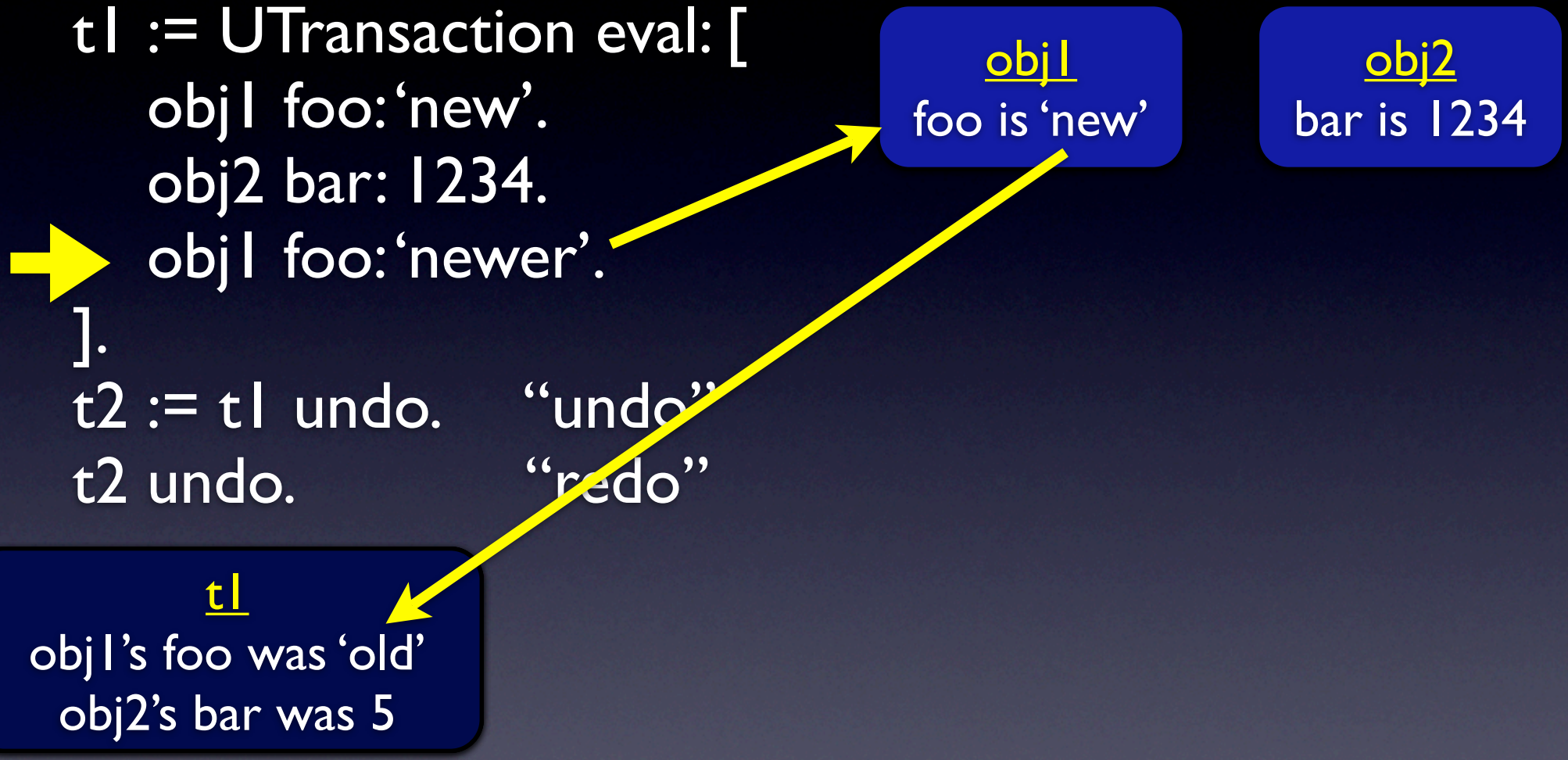


obj1
foo is 'new'

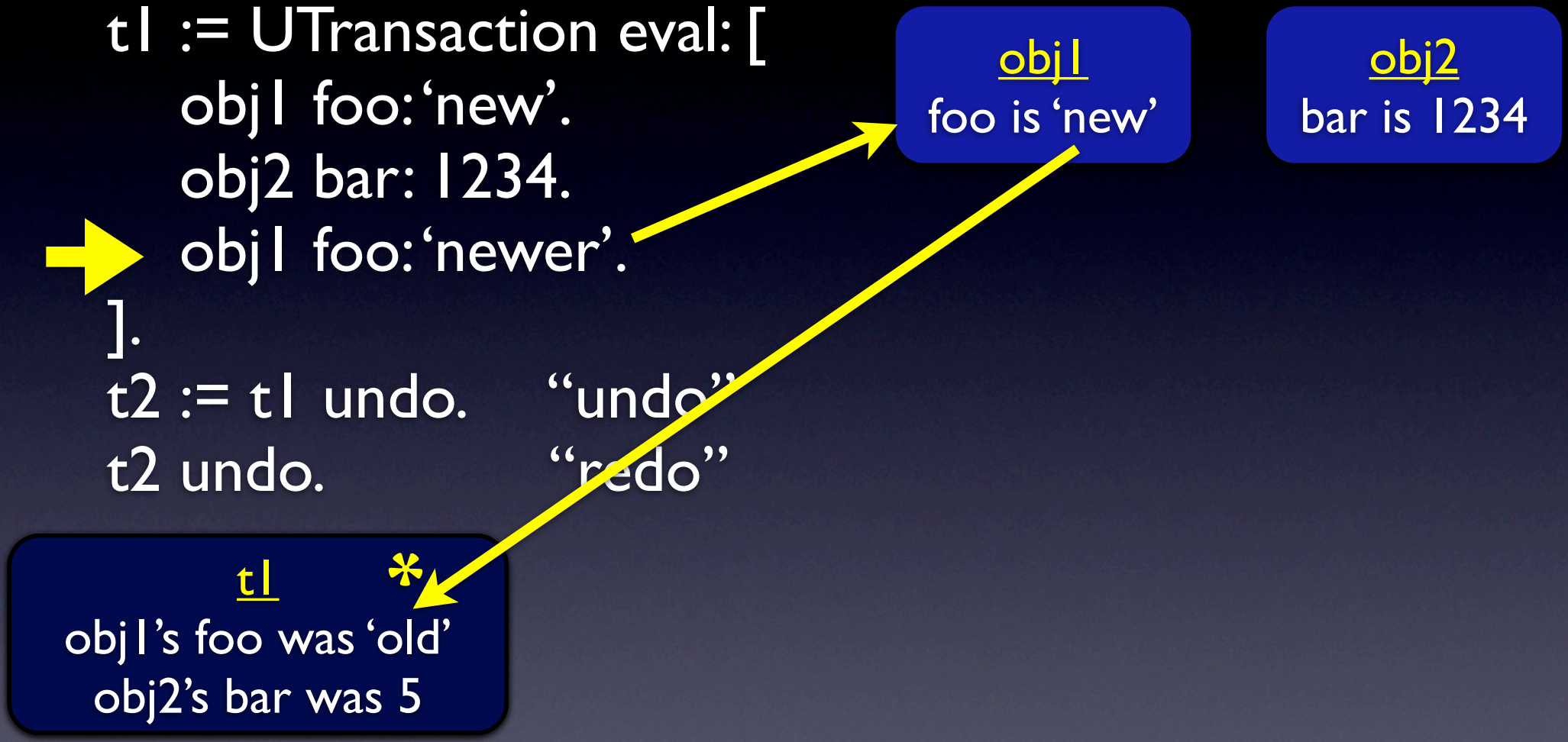
obj2
bar is 1234

t1
obj1's foo was 'old'
obj2's bar was 5

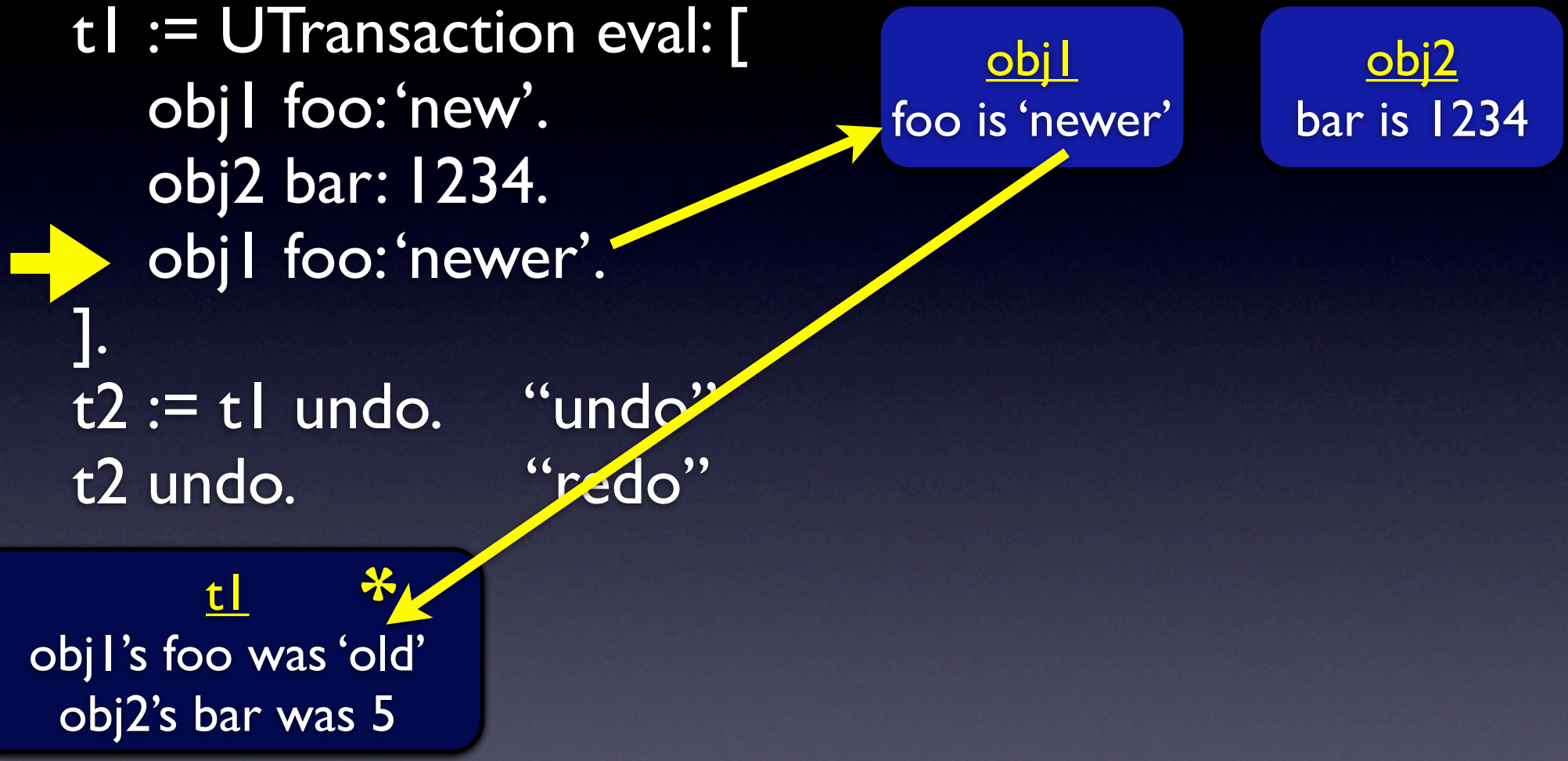
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



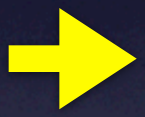
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

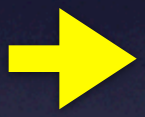


obj1
foo is 'newer'

obj2
bar is 1234

t1
obj1's foo was 'old'
obj2's bar was 5

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



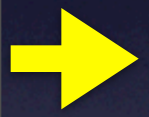
obj1
foo is 'newer'

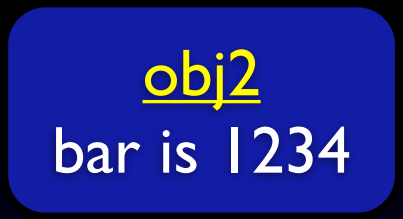
obj2
bar is 1234

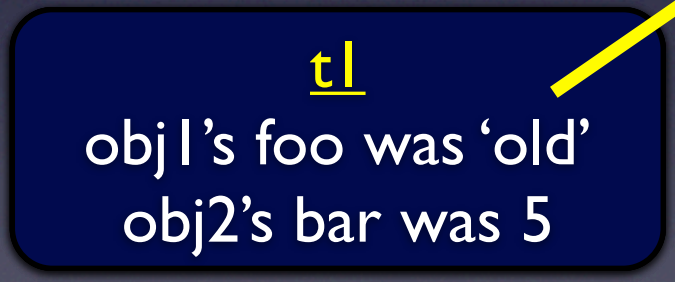
t1
obj1's foo was 'old'
obj2's bar was 5

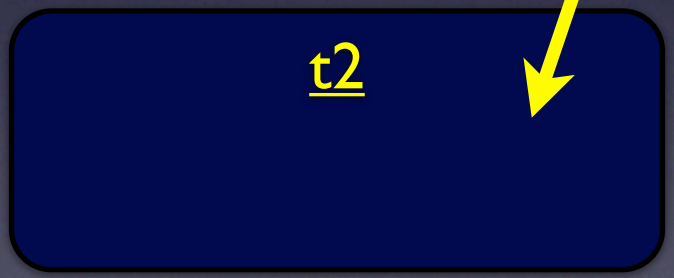
t2

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

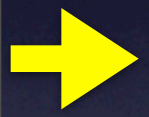




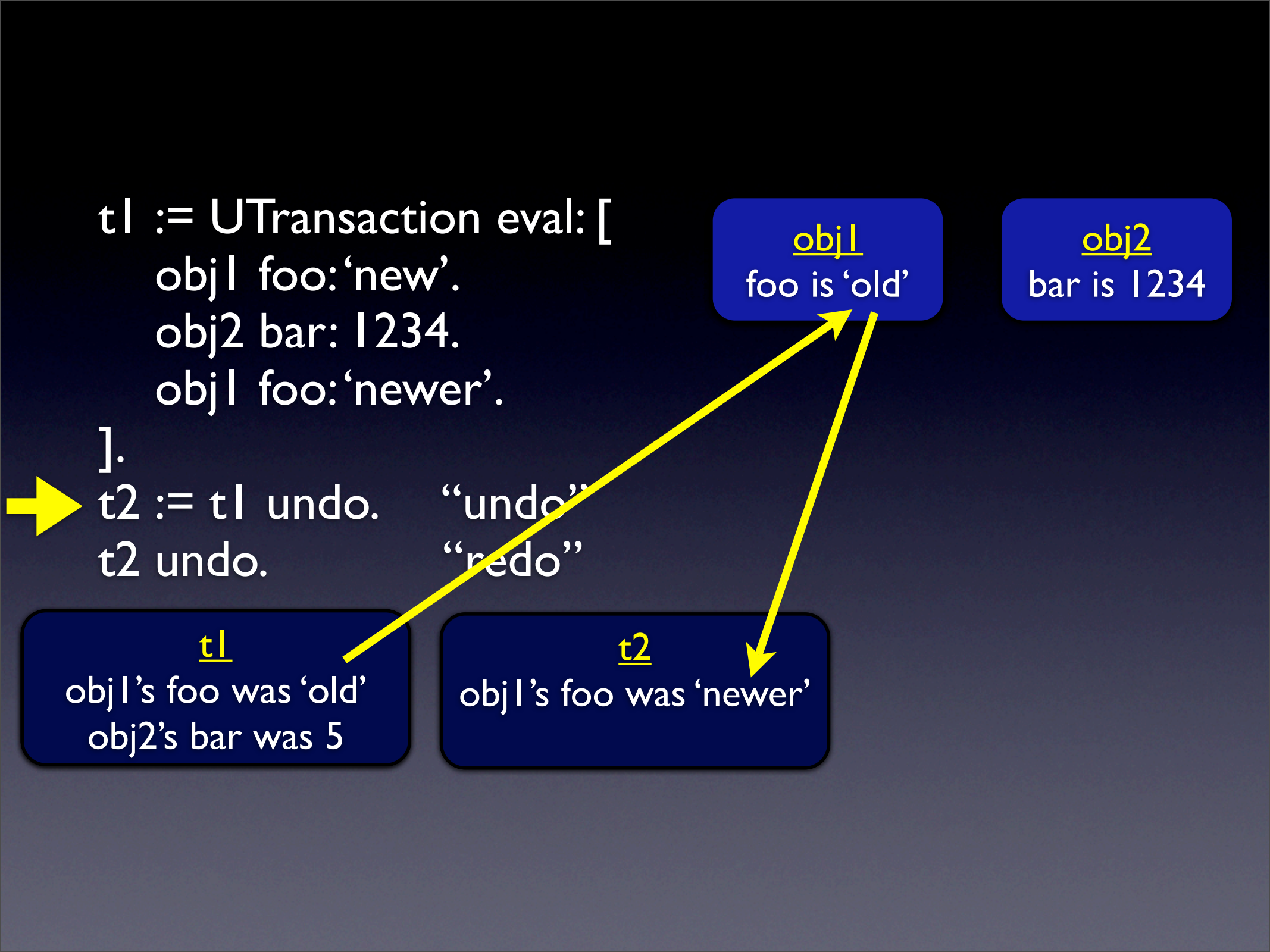




```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

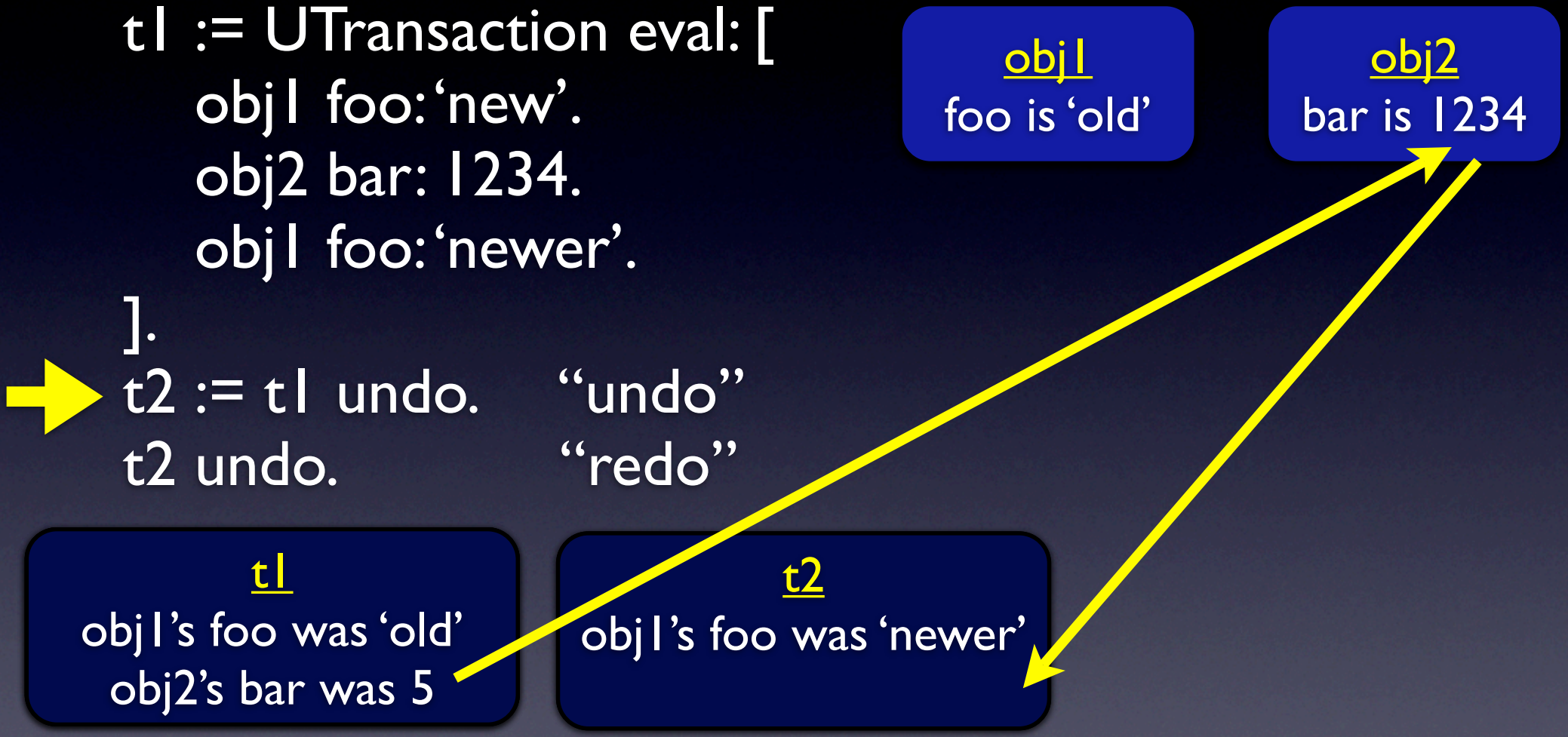
obj1
foo is 'old'

obj2
bar is 1234

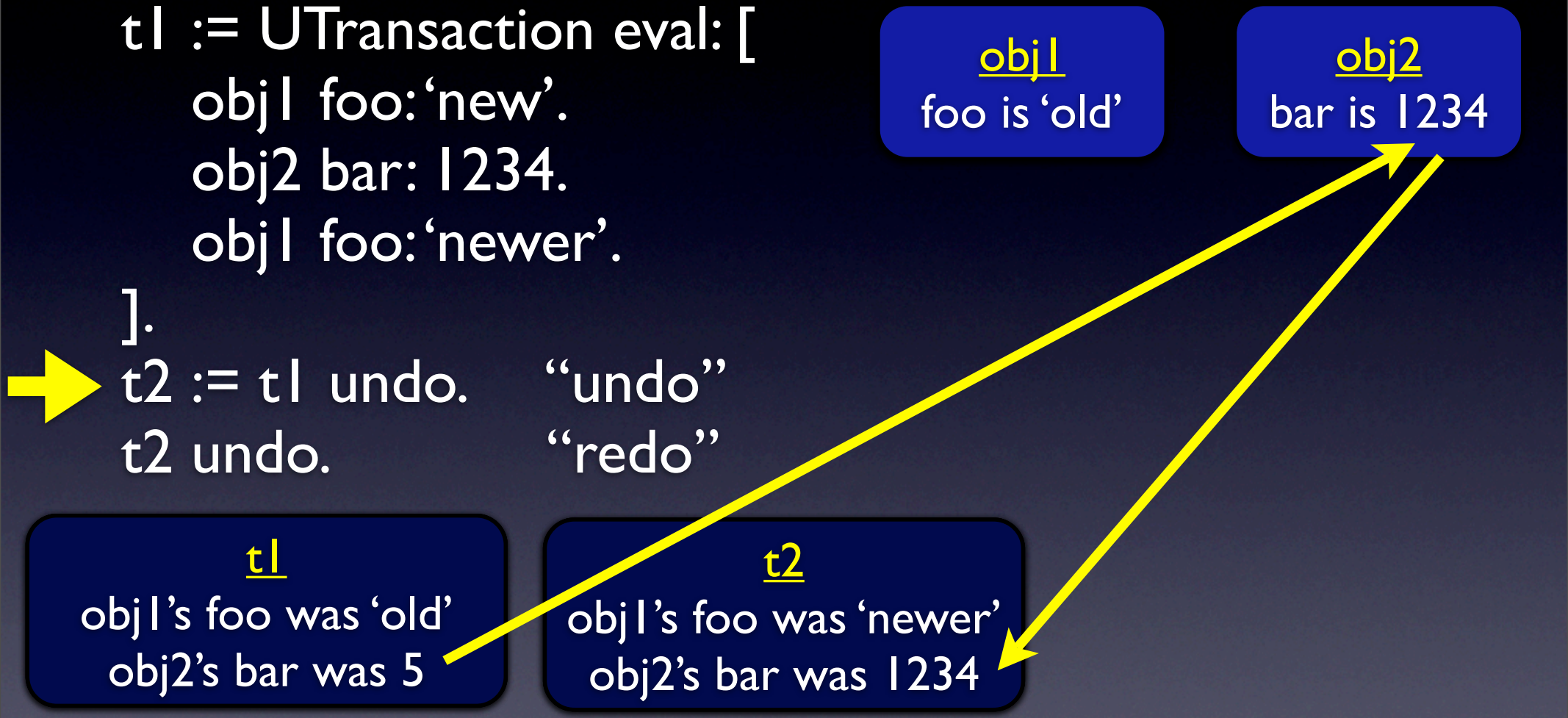
t1
obj1's foo was 'old'
obj2's bar was 5

t2
obj1's foo was 'newer'

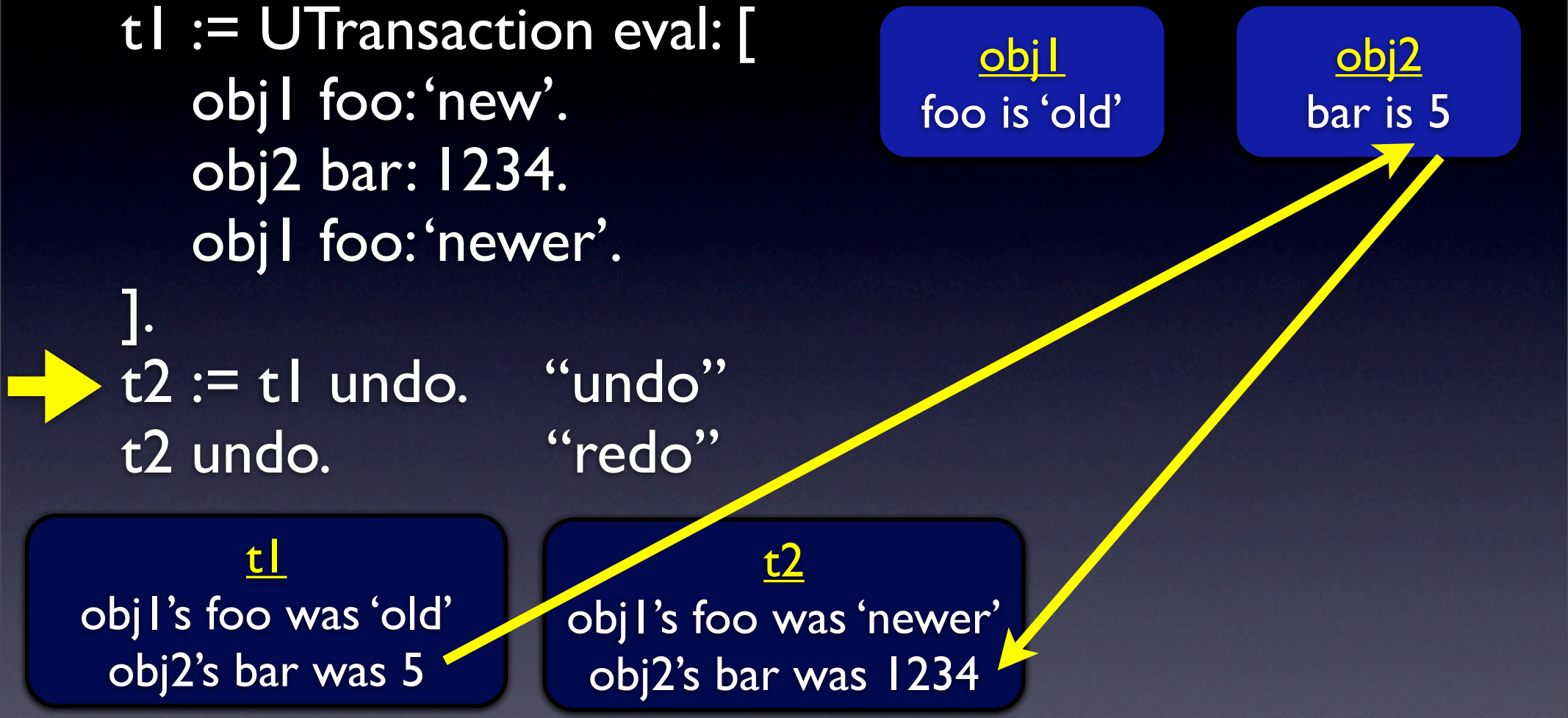
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



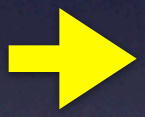
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



obj1
foo is 'old'

obj2
bar is 5

t1
obj1's foo was 'old'
obj2's bar was 5

t2
obj1's foo was 'newer'
obj2's bar was 1234

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'old'

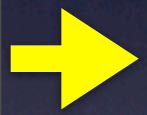
obj2
bar is 5

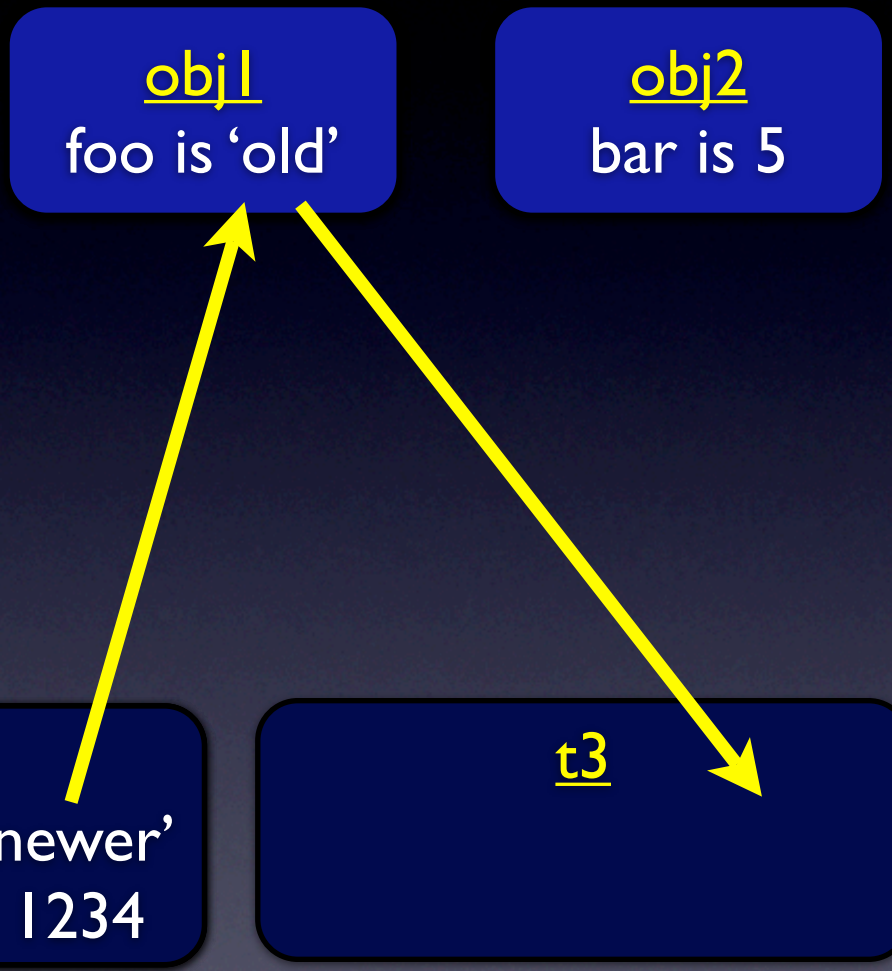
t1
obj1's foo was 'old'
obj2's bar was 5

t2
obj1's foo was 'newer'
obj2's bar was 1234

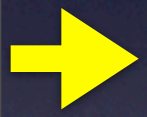
t3

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

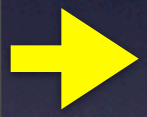


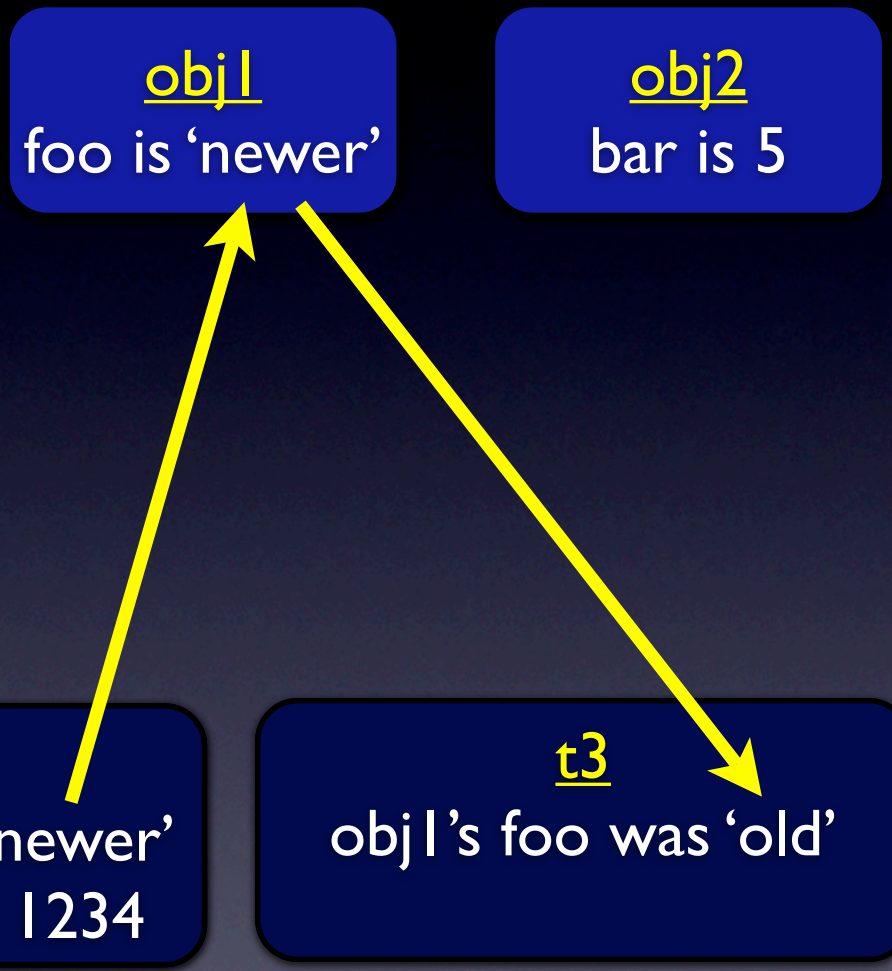


```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```





```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'newer'

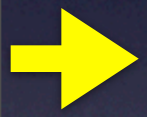
obj2
bar is 5

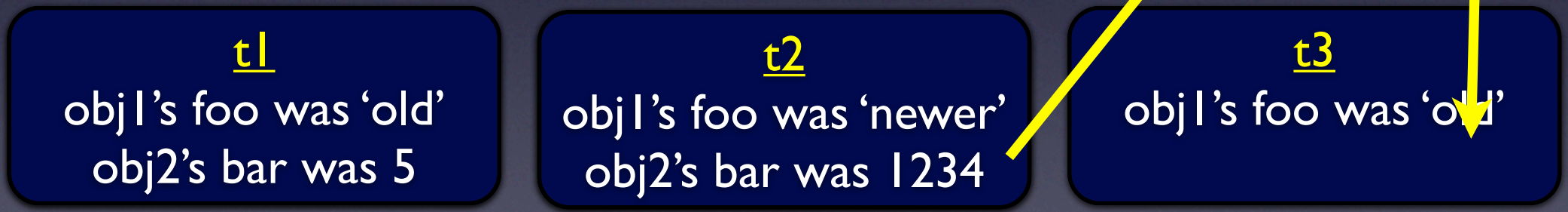
t1
obj1's foo was 'old'
obj2's bar was 5

t2
obj1's foo was 'newer'
obj2's bar was 1234

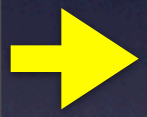
t3
obj1's foo was 'old'

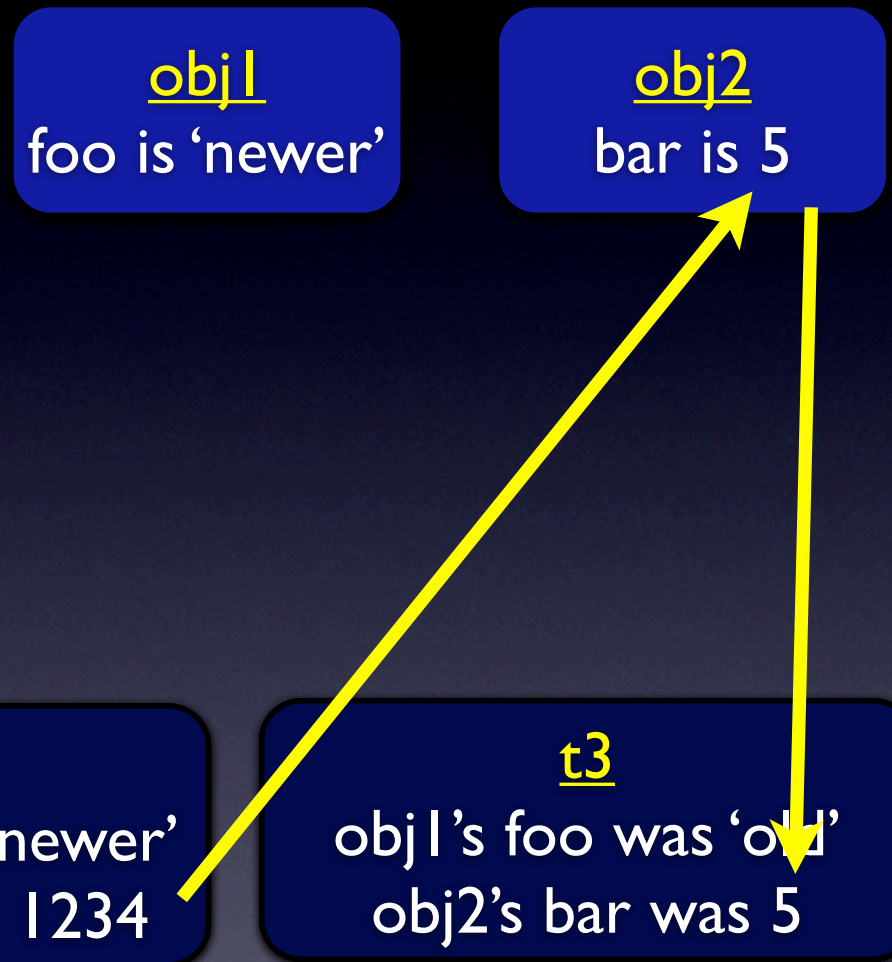
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```



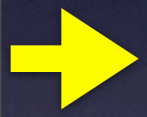


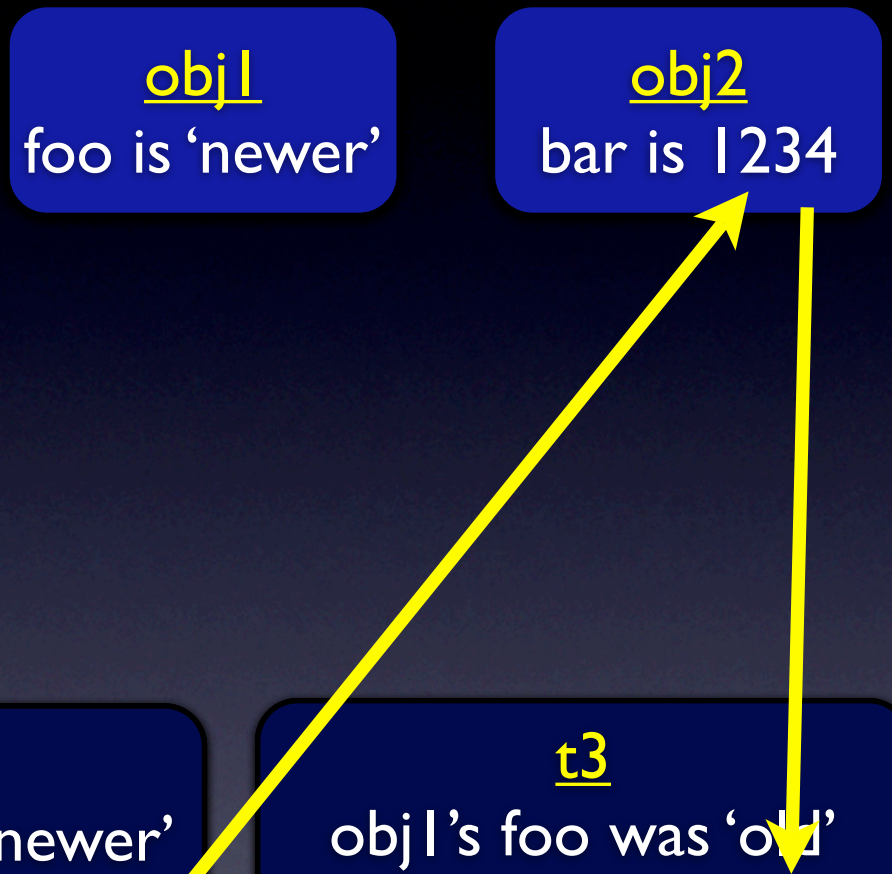
```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

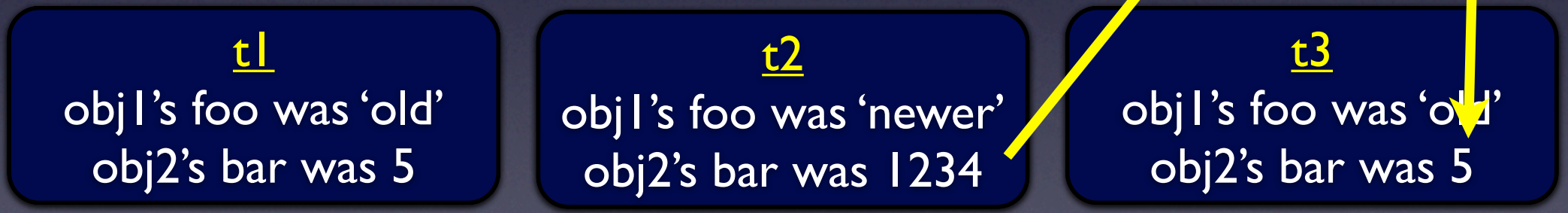




```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```







```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'newer'

obj2
bar is 1234

t1
obj1's foo was 'old'
obj2's bar was 5

t2
obj1's foo was 'newer'
obj2's bar was 1234

t3
obj1's foo was 'old'
obj2's bar was 5

```
t1 := UTransaction eval: [
    obj1 foo: 'new'.
    obj2 bar: 1234.
    obj1 foo: 'newer'.
].
t2 := t1 undo.      "undo"
t2 undo.            "redo"
```

obj1
foo is 'newer'

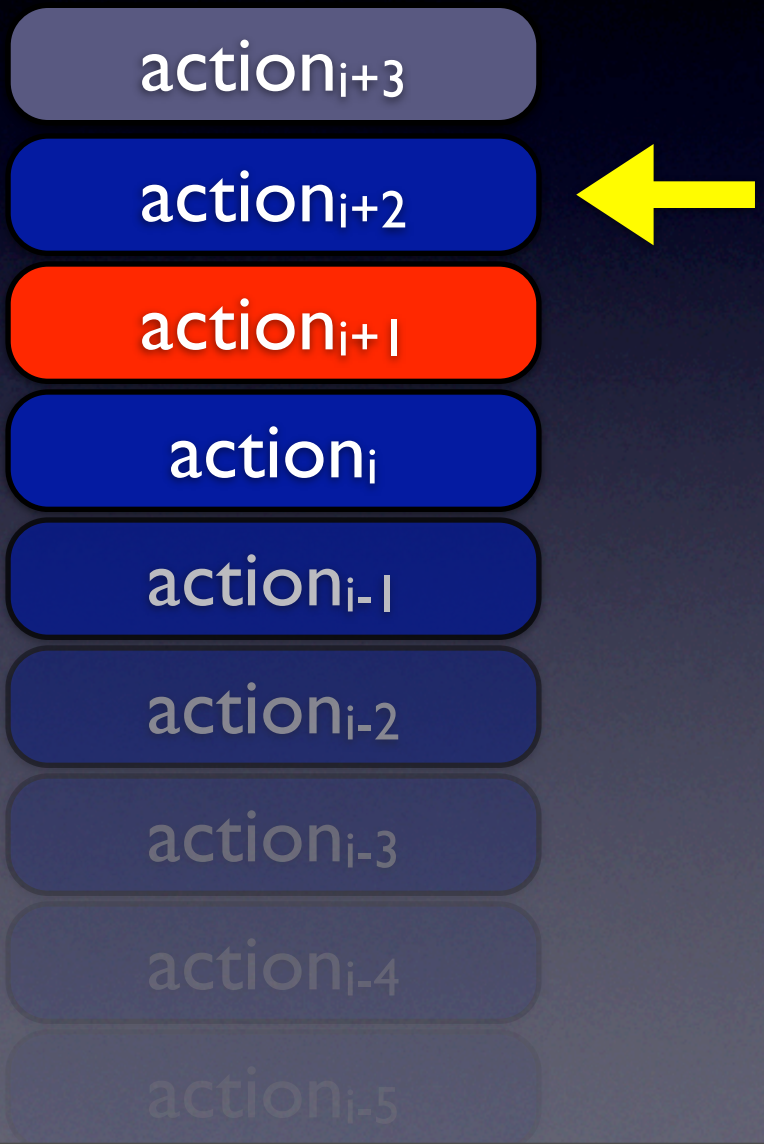
obj2
bar is 1234

t1
obj1's foo was 'old'
obj2's bar was 5

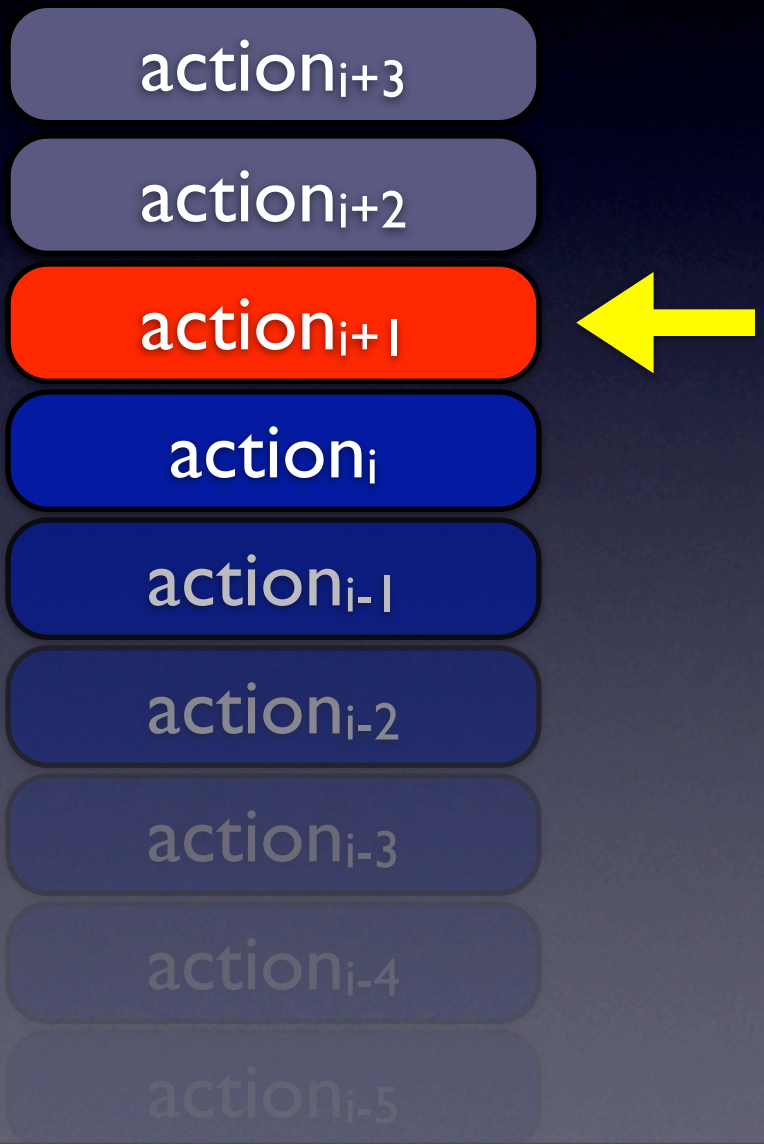
t2
obj1's foo was 'newer'
obj2's bar was 1234

Trouble w/ Linear Undo

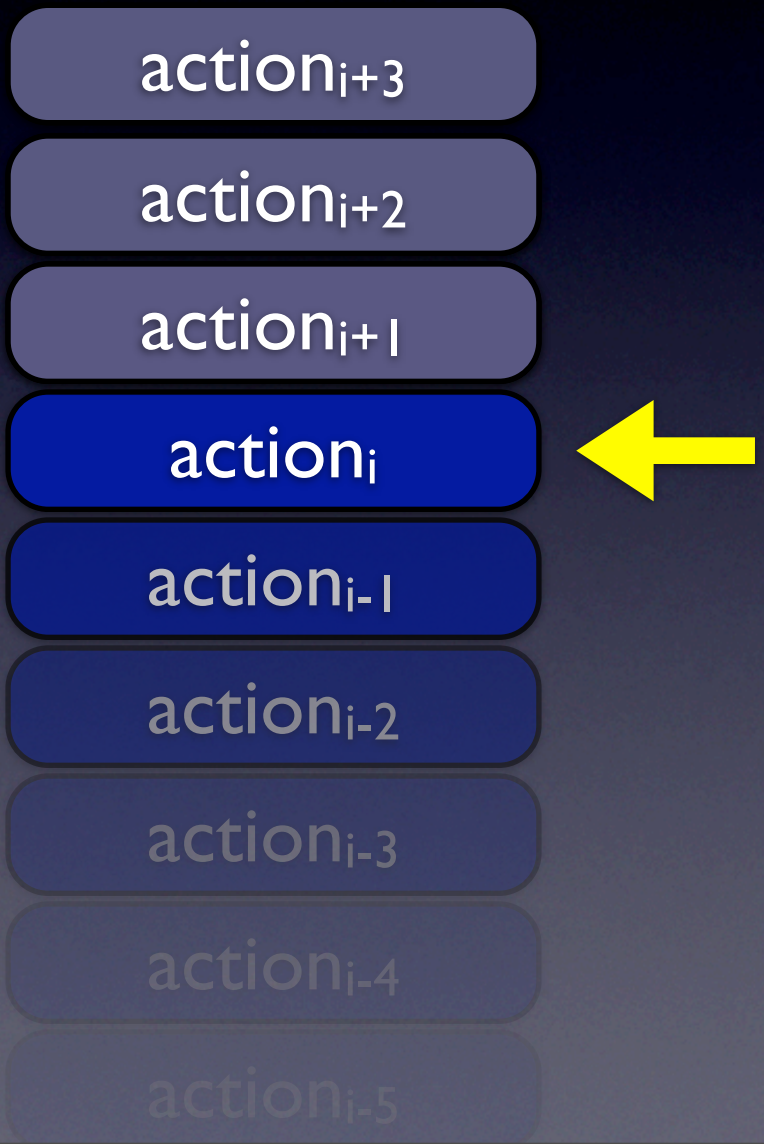
Trouble w/ Linear Undo



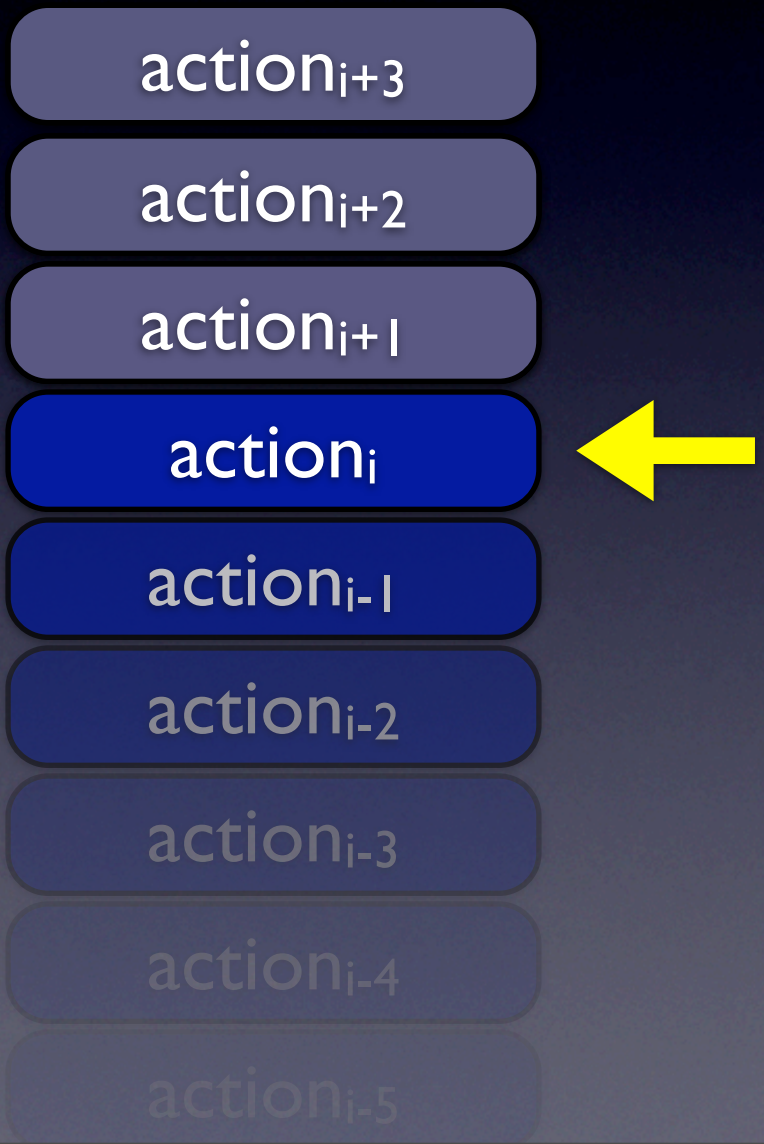
Trouble w/ Linear Undo



Trouble w/ Linear Undo



Trouble w/ Linear Undo



Problem:
can't redo $action_{i+2}$ and $action_{i+3}$ w/o redoing $action_{i+1}$

Want *Selective* Undo

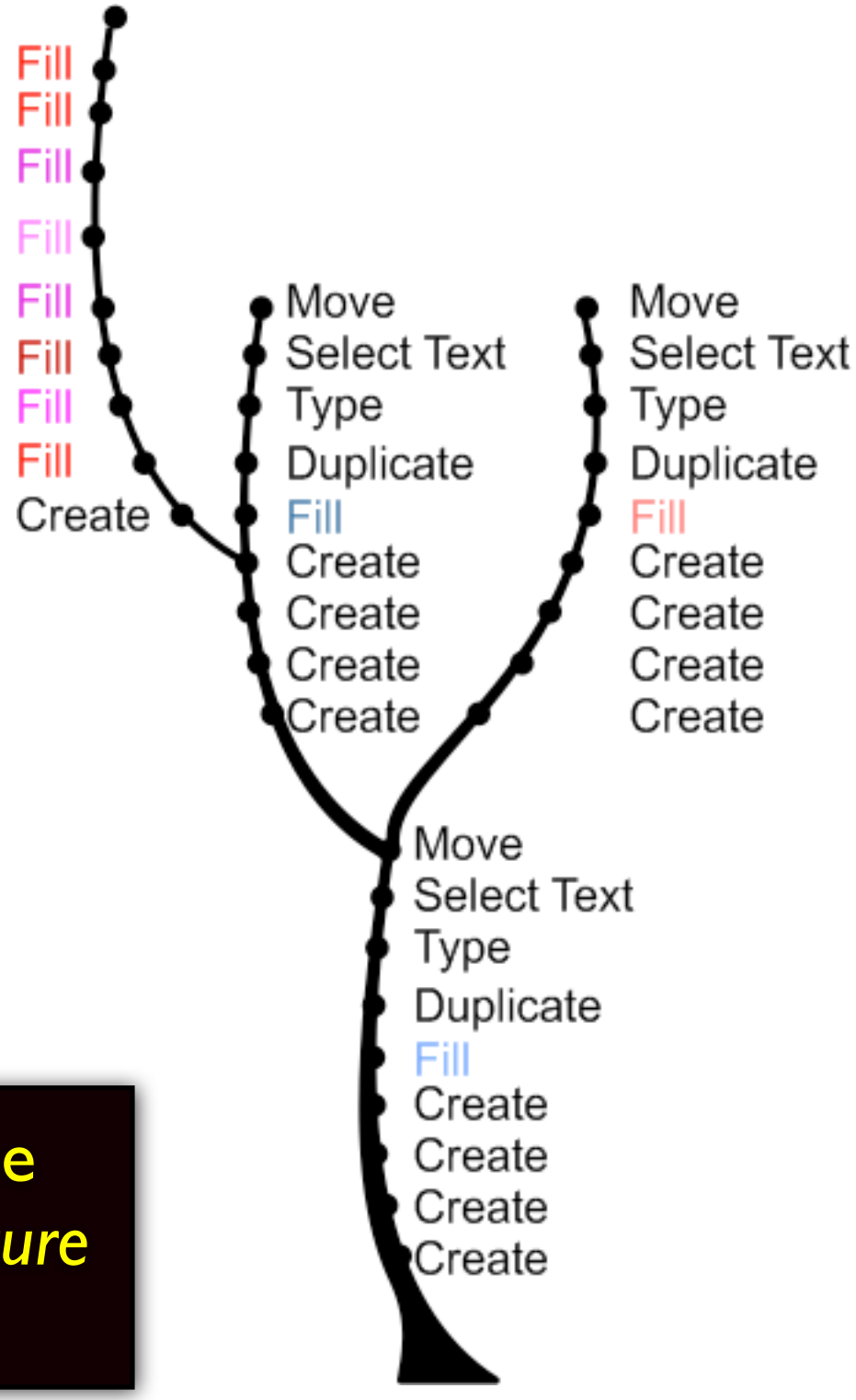
- Undo a command without first undoing commands that were issued afterwards
- **BUT** some commands are based on effects of earlier commands
  - gets tricky!

Selective Undo (Sort Of)

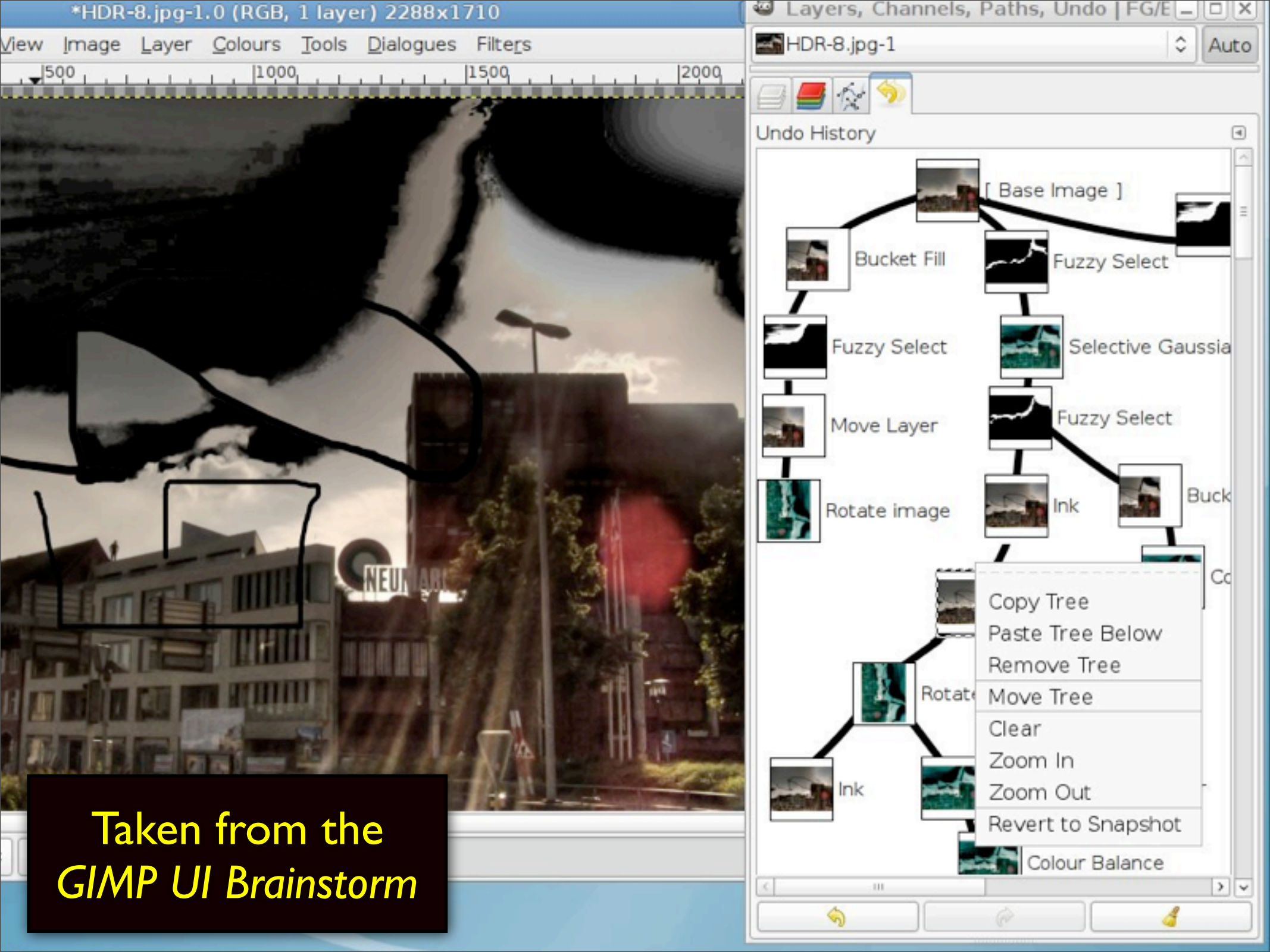
- Can undo $action_{i+1}$ directly (no stack required)
- ... but UTransaction's undo is *transitive*
  - *Undoing a transaction t will transitively undo all later transactions that modified one or more objects modified by t*
- Still stack-like, but *only related operations are undo'ed transitively*
  - A kind of "selective undo" that makes sense

(Too Big a Hammer?)

- It may be!
- ... but we could write the program so that different aspects of an object are stored in different "sub-objects"
  - keeps mechanism easy to understand
- Another option: take into account what properties of what objects were modified
- No clear winner yet

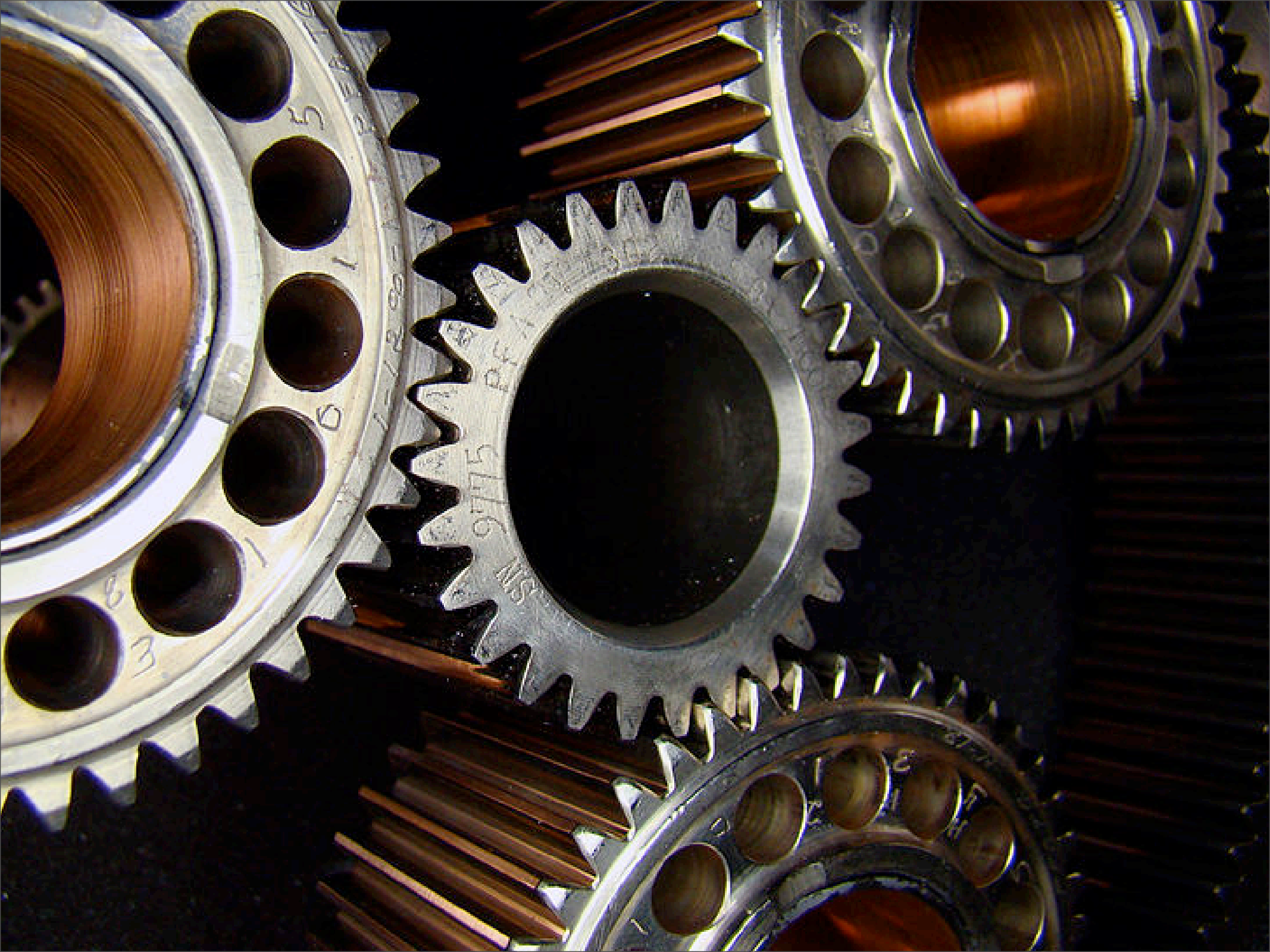


Taken from the *CorelDRAW Feature Request Site*

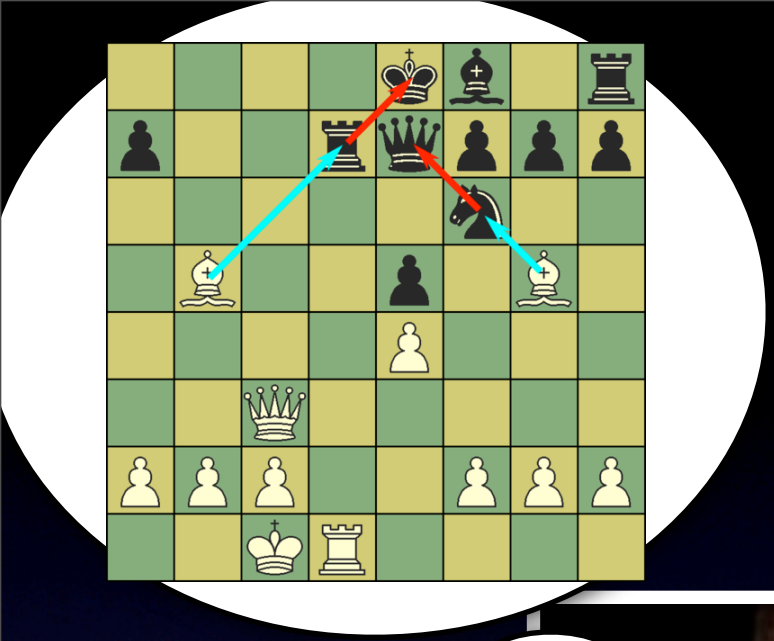

*HDR-8.jpg-1.0 (RGB, 1 layer) 2288x1710
View Image Layer Colours Tools Dialogues Filters
500 1000 1500 2000
Layers, Channels, Paths, Undo | FG/E
HDR-8.jpg-1
Auto
Undo History
[Base Image]
Bucket Fill
Fuzzy Select
Fuzzy Select
Selective Gaussia
Move Layer
Fuzzy Select
Rotate image
Ink
Buck
Copy Tree
Paste Tree Below
Remove Tree
Move Tree
Clear
Zoom In
Zoom Out
Revert to Snapshot
Ink
Colour Balance
Taken from the
GIMP UI Brainstorm

Part II

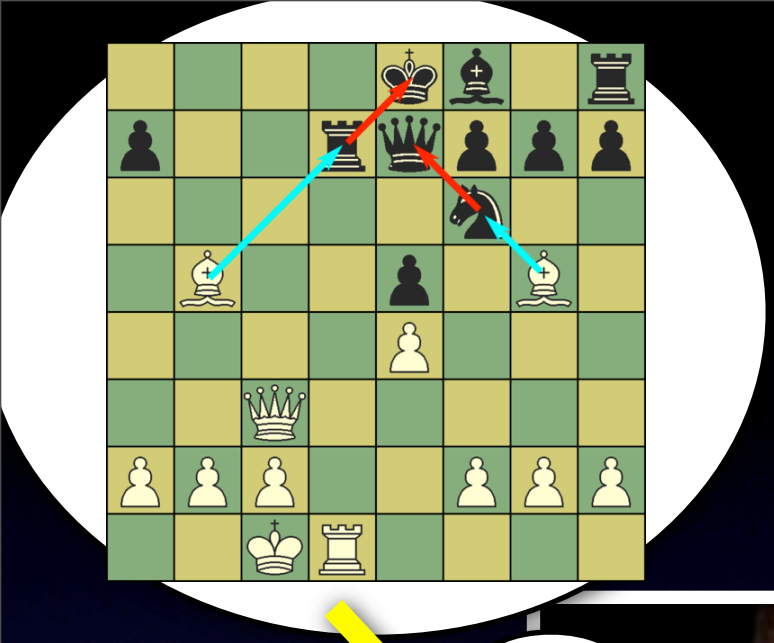
Worlds:
Undo for Programs



what if...
I take his knight with my bishop?



what if...
I take his knight with my bishop?



what if...
I take his knight with my bishop?

Part II

Worlds:
Undo for Programs

Part II

Worlds:
for Programs

I'm Talking About...

- Programming language support for
  - "thought experiments", a.k.a.,
  - "possible worlds reasoning"
- How? By enabling programmers to control the scope of side effects.

About Side Effects

- Not *all* side effects!
- Only changes to the program store, e.g.,
  - global, local, instance, and class variables
  - arrays
  - ...

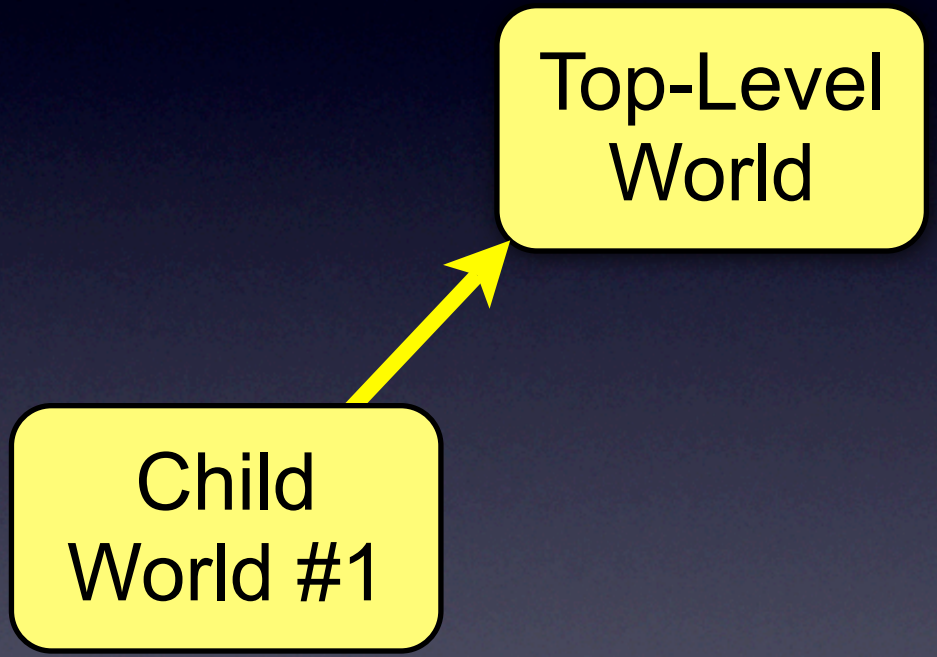
Worlds

- A simple and expressive model for controlling the scope of side effects
- Worlds: new kind of *first-class store*
  - allows multiple versions of the program store to co-exist
  - organized hierarchically
- Worlds/Squeak and Worlds/JS

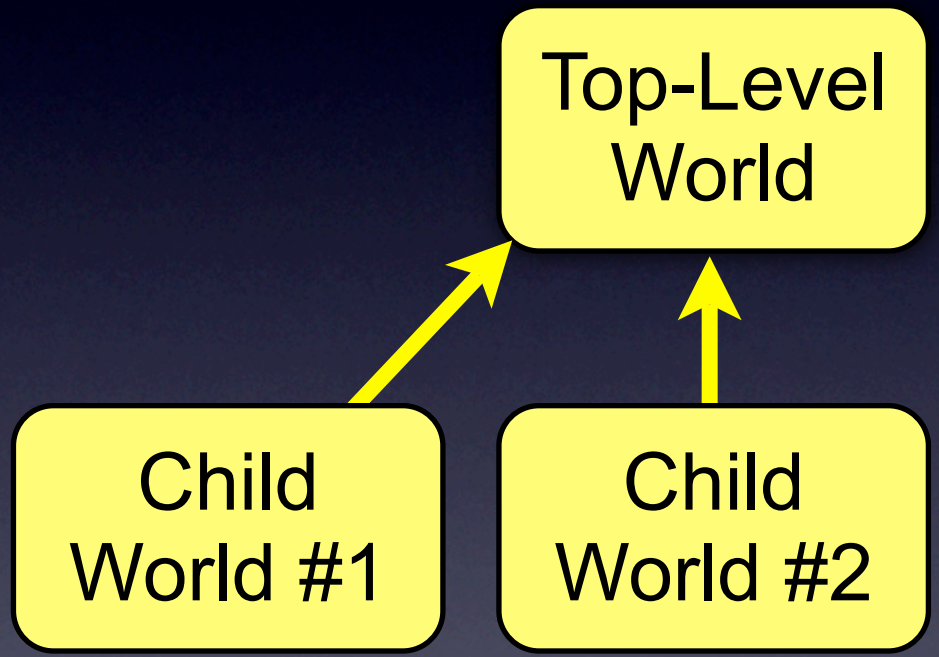
The Programming Model

The Programming Model

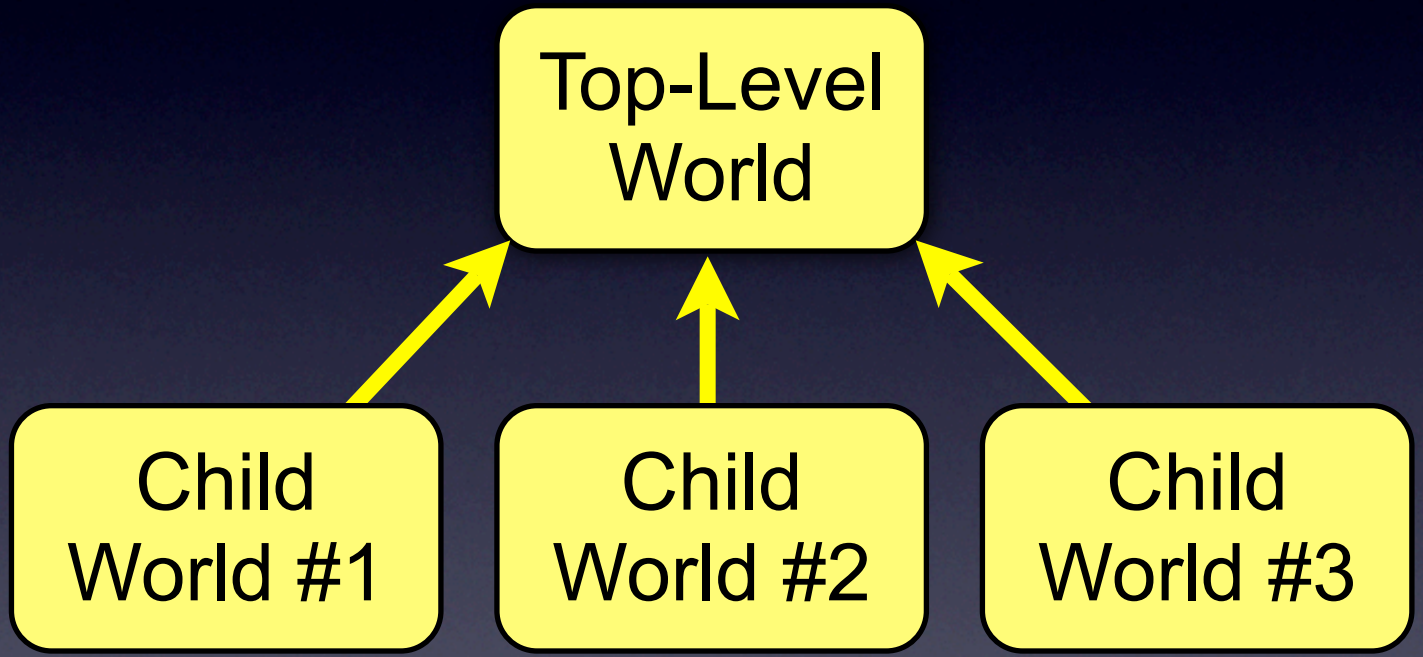
The Programming Model



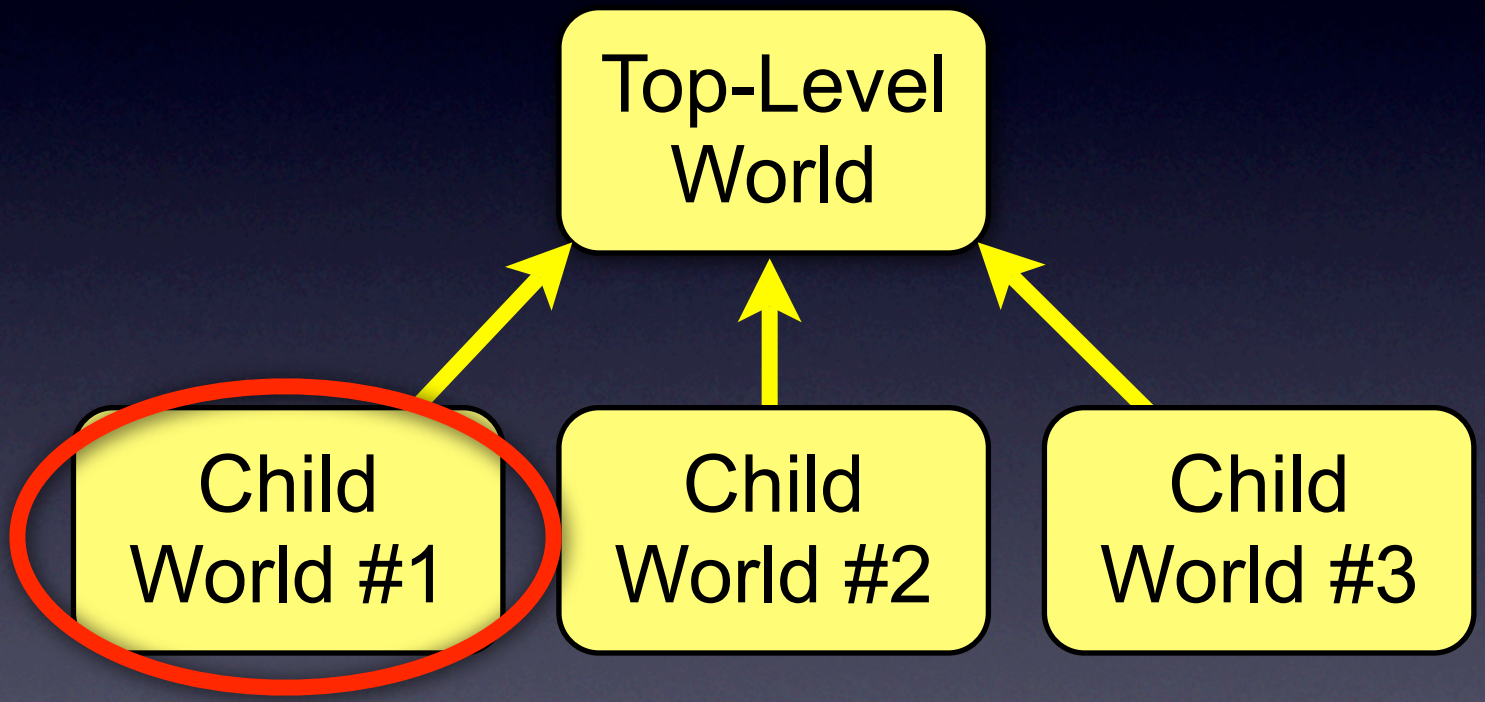
The Programming Model



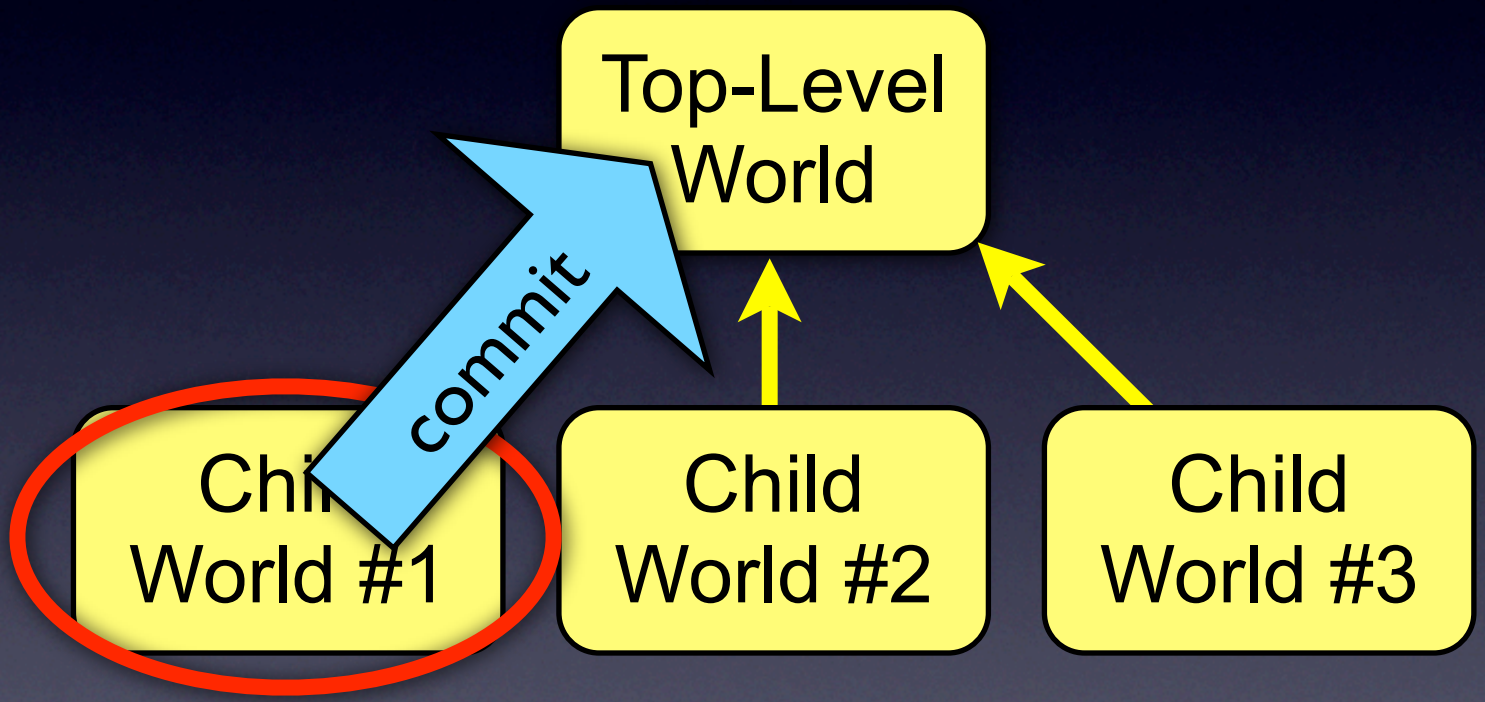
The Programming Model

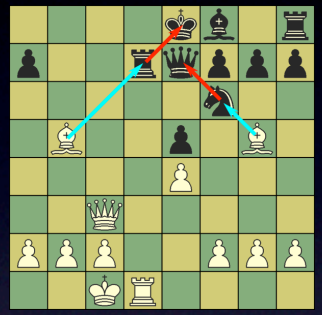


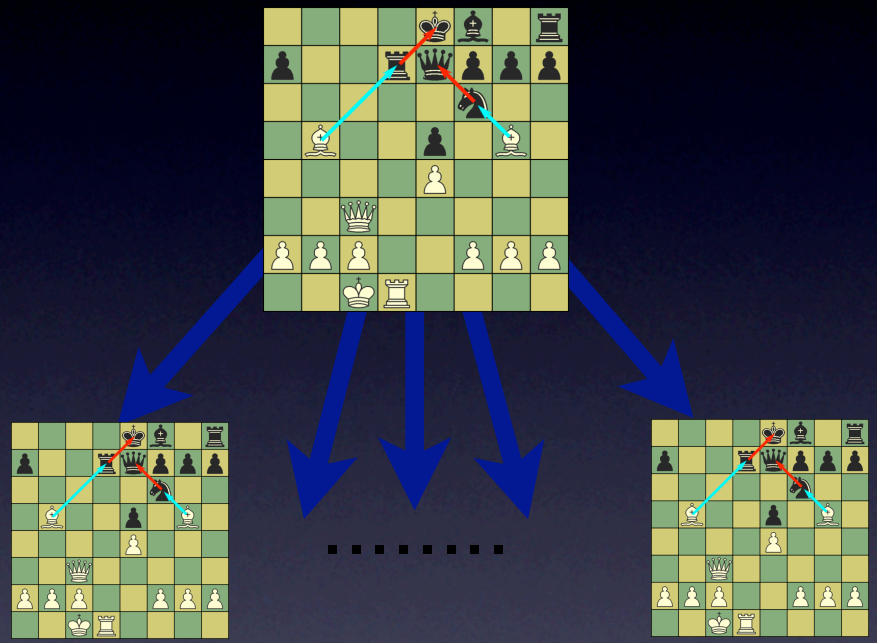
The Programming Model

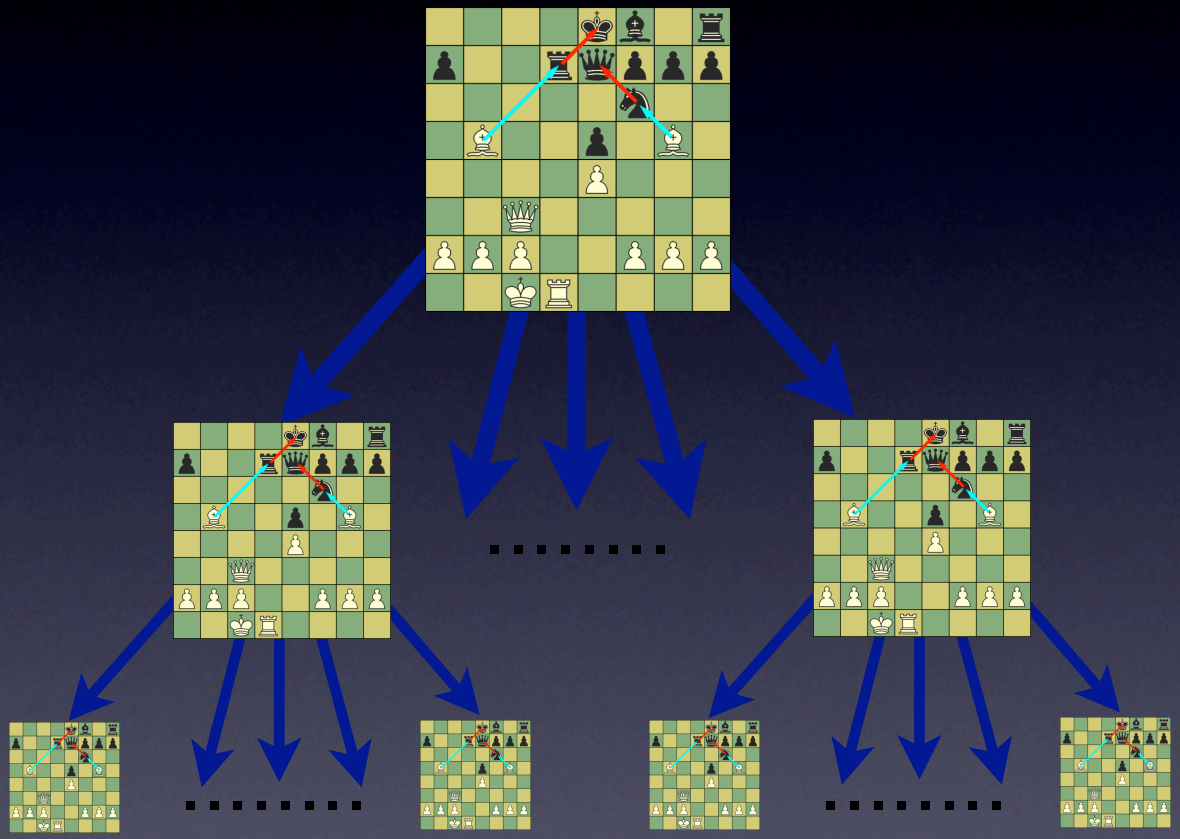


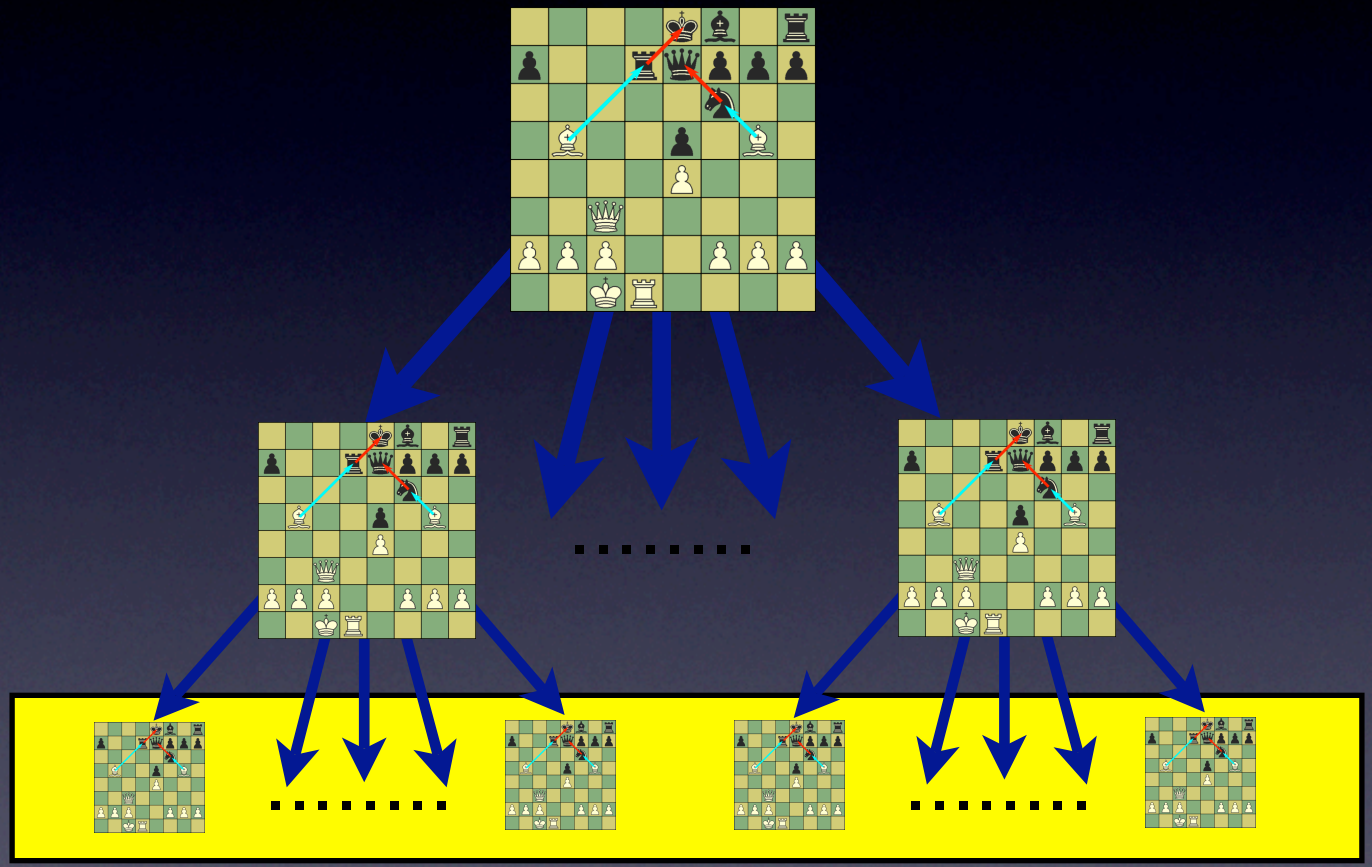
The Programming Model

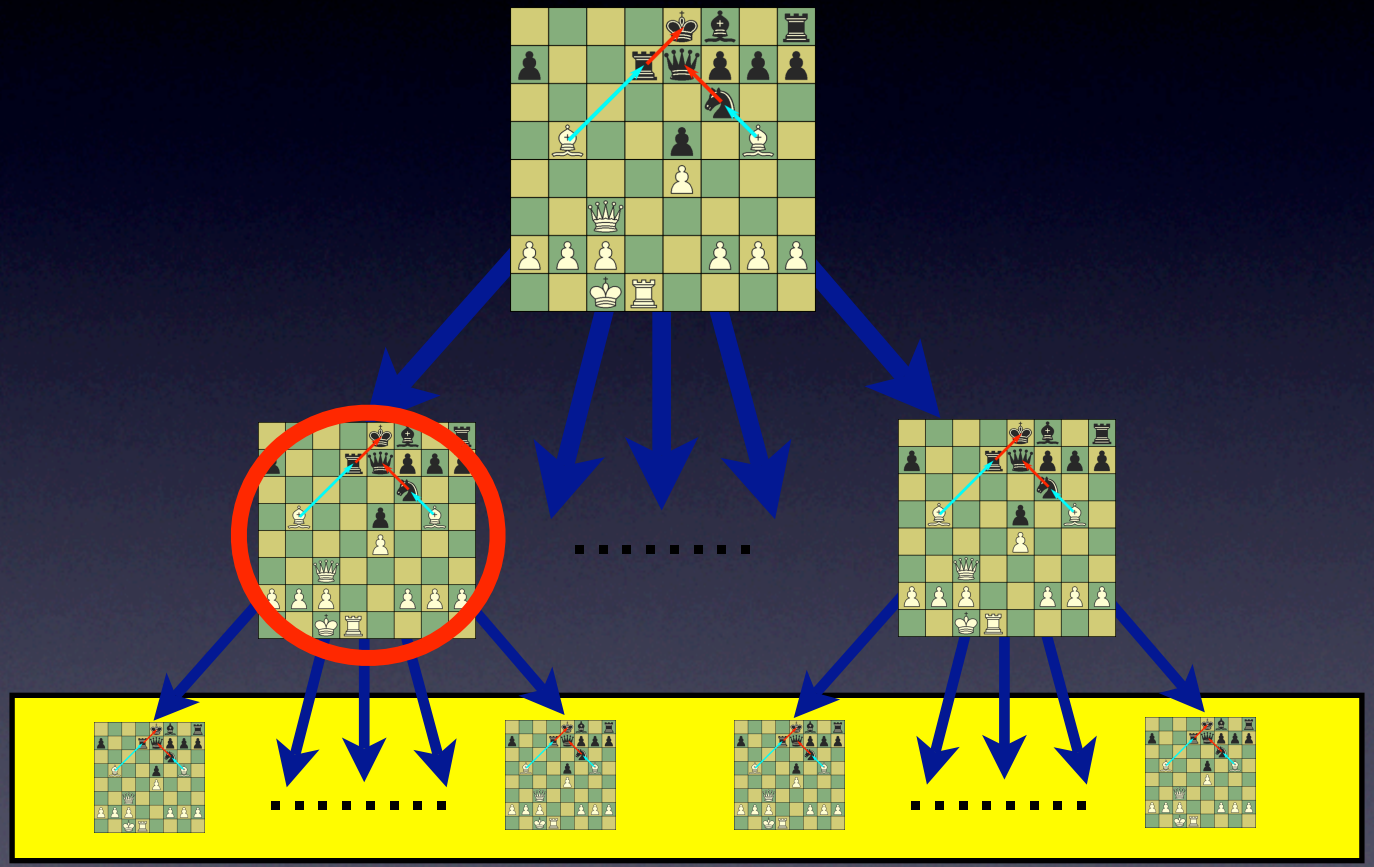


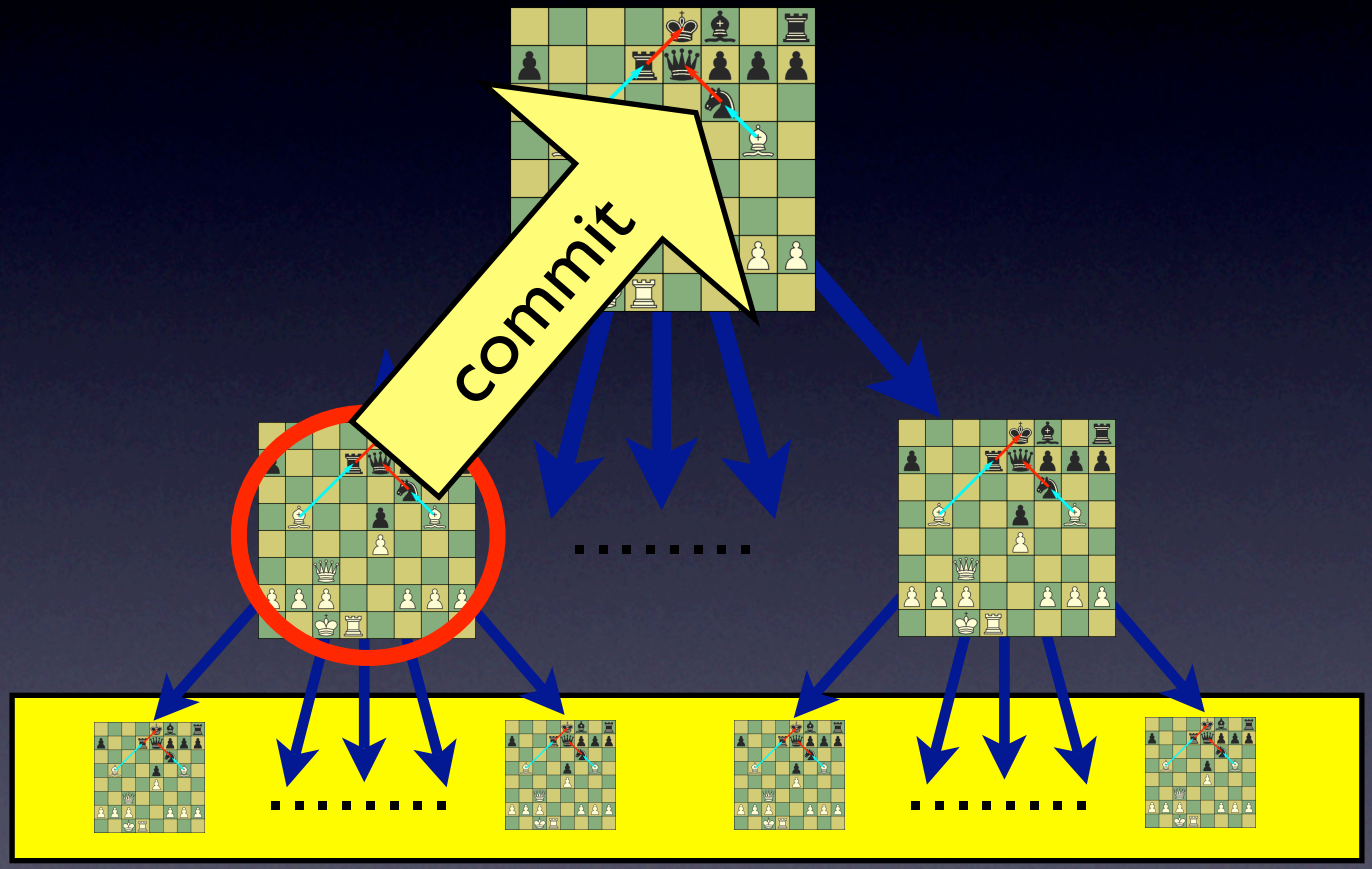


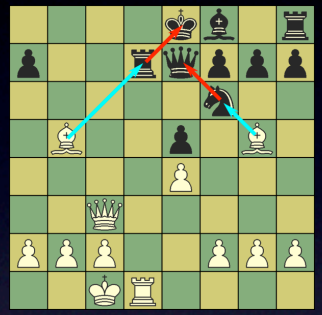









commit



Worlds/Squeak

- thisWorld
- w sprout
- w commit
- w eval: [...]

Exception Handling

```
[
    xs do: [:x |
        x update
    ]
] on: Exception do: [
    ...
]
```

Exception Handling


```
[
    xs do: [:x |
        x update
    ]
] on: Exception do: [
    ...
]
```

Exception Handling

```
[
    xs do: [:x |
        x update
    ]
] on: Exception do: [
    ...
]
```

Exception Handling

```
[
    xs do: [:x |
        x update
    ]
] on: Exception do: [
    ...
]
```

Exception Handling

```
[
    thisWorld sprout eval: [
        xs do: [:x |
            x update
        ].
        thisWorld commit
    ]
] on: Exception do: [

]
```

Exception Handling

```
[
    thisWorld sprout eval: [
        xs do: [:x |
            x update
        ].
        thisWorld commit
    ]
] on: Exception do: [

]
```

Sandboxing

```
sandbox = thisWorld.sprout();
in sandbox {
  eval(untrustedCode);
}
```

Sandboxing

```
disableDangerousStuff = function() {
  alert = null;
  Object.prototype.forbiddenMethod = null;
  ...
}
```

```
sandbox = thisWorld.sprout();
in sandbox {
  disableDangerousStuff();
  eval(untrustedCode);
}
```

Extension Methods in JS

```
Number.prototype.fact = function() {
  if (this == 0)
    return 1;
  else
    return this * (this - 1).fact();
};

print(5.fact());
```

Extension Methods in JS

^ scoped

```
myModule = thisWorld.sprout();
in myModule {
  Number.prototype.fact = function() { ... };
}

in myModule {
  print(5.fact());
}
```

Back to OMeta

```
rhyme = fee fie foe fum
      | fiddle dee dee
```

Back to OMeta

```
rhyme = fee fie foe fum
      | fiddle dee dee

fee   = ... -> ...
fie   = ... -> ...
foe   = ... -> ...
fum   = ... -> ...
```

Back to OMeta

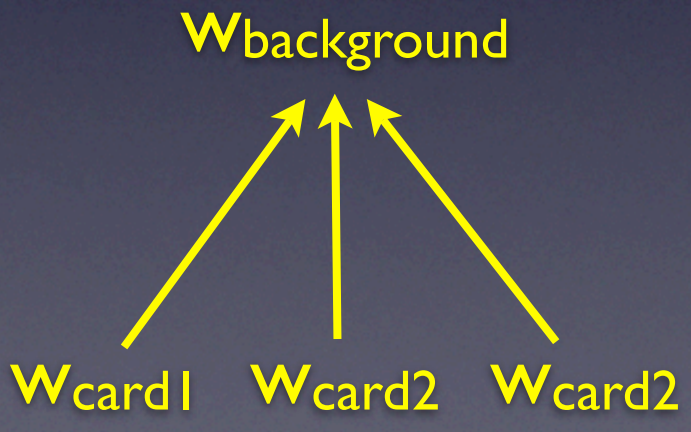
```
rhyme = fee fie foe fum
      | fiddle dee dee

fee   = ... -> ...
fie   = ... -> ...
foe   = ... -> ...
fum   = ... -> ...
```

Case Study I

- Variant of OMeta/JS in which backtracking rolls back the side effects of rules' semantic actions
- OMeta implemented in JS, and Worlds/JS is a proper superset of JS
  - Re-implemented OR, kleene-*, etc. using worlds
  - very, **very** difficult to do w/o (something like) Worlds

Case Study II

- Hypercard-like system implemented w/ Worlds (w/ Ted Kaeher and Yoshiki Ohshima)
- All backgrounds and cards in a stack are really *just one card*, viewed through different worlds



$W_{background}$ contains the default state of the card, which is shared by all cards

W_{cardi} overrides the state of the card, as it appears in $W_{background}$

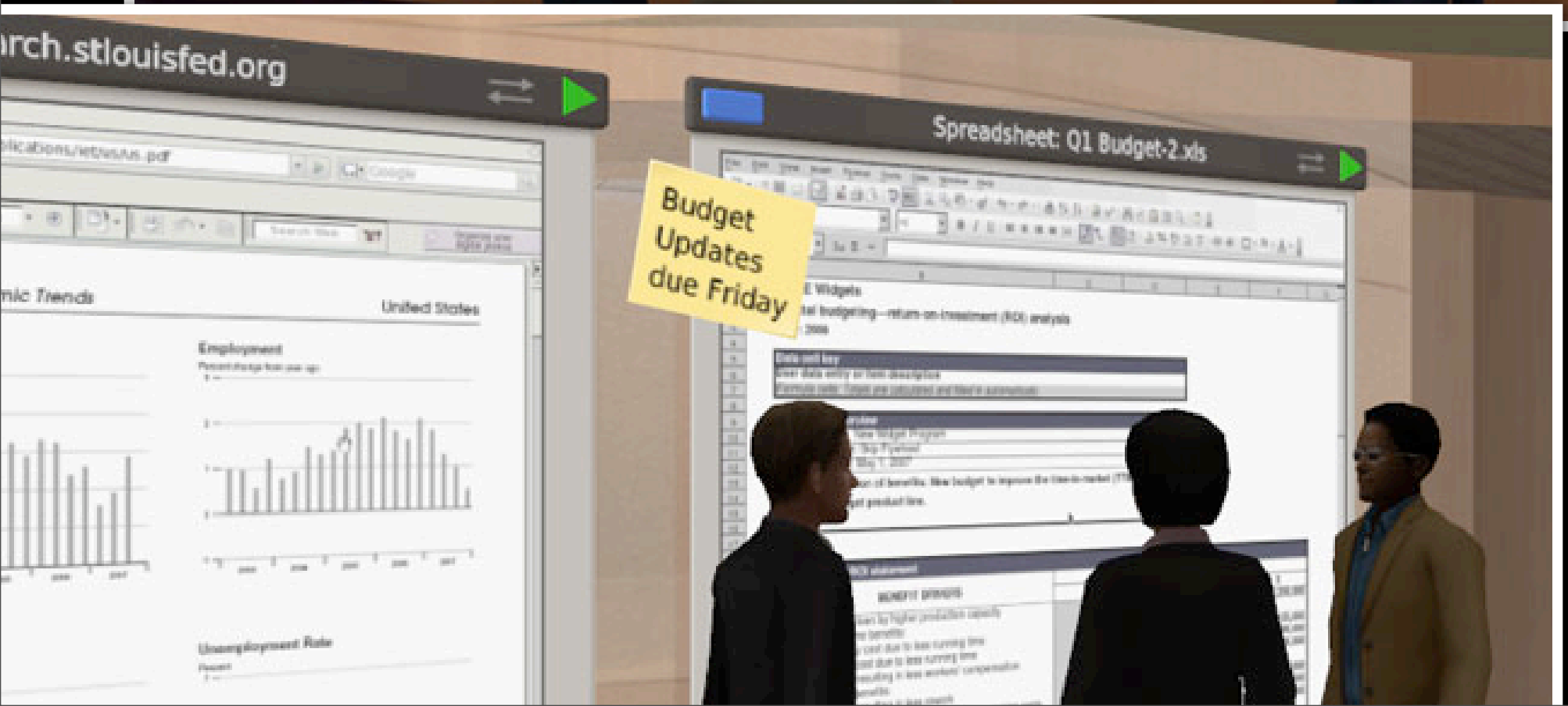
THE card

Future Work

Future Work

- Invariants!
  - register inter- *and* intra-object inv's dynamically
  - modify objects in transactions
  - all relevant invariants checked at end of transaction
  - only commit transaction if all inv's hold


SPC/NSSL WINTER WEATHER TRAINING
(WWT) HOME PAGE
Bob
Mic Off


rch.stlouisfed.org
Spreadsheet: Q1 Budget-2.xls
Budget
Updates
due Friday

Future Work (cont'd)

- Mechanisms for synchronizing distributed, decentralized systems like
  - TeaTime [Reed '78]
  - Virtual Time / Time Warp [Jefferson '85]
- ... rely on support for speculative execution
- (May be able to do even better w/ Worlds)

Future Work (cont'd)

- Worlds: a model for programming multi-core architectures?
  - e.g., choosing among optimizations
  - will need efficient, HW-based impl.

Part III

Churrasco!

Worlds vs. UObjects

support for spec. execution, possible worlds reasoning	transitive undo
very general, b/c it works on every object in the system	only affects objects designed to work with it
very dangerous, b/c it works on every object in the system	only affects objects designed to work with it

pro +

con -

Questions?

For more info...

http://tinlizzie.org/~awarth