RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

VLSI Design:

SKILL

Prof. Dr. P. Fischer

Lehrstuhl für Schaltungstechnik und Simulation
Technische Informatik der Uni Heidelberg

What is Skill and what can it do ?

- SKILL is the shell / control language of cadence
- It is used for
  - Configuration of the environment
    - Definition of library path
    - …
  - Configuration of tools
    - Definition of ShortCuts
    - Definitions of new commands / menu entries
    - …
- Skill allows, for instance, direct access to objects in an open view for
  - Scripted creation of shapes / labels / …
  - Automated creation of symbols
  - Extraction of pad positions, …

How does SKILL look like?

- SKILL – in its 'natural' form – is very similar to LISP ('LISt Processing')
  - Commands have the form `( cmd arg1 arg2 ... )`
  - Data is mostly stored as *lists*
- Operators are possible as well, i.e.

```
•3 + 5                    (equivalent to (plus 3 5))
•x = 6
```

- A 'C-like' form is possible as well: `cmd( args..)`
  - Note that the `(` must *DIRECTLY* follow `cmd`, i.e. with ***NO*** blank!
- SKILL is caseSENsitTive!
- Comments are started by `//` or enclosed in `/*...*/` (as C)
- SKILL is – normally – interpreted
  - it can also be *compiled* (→ `*.cxt`) end *encrypted*

Where to find Help & Documentation ?

- At http://en.wikipedia.org/wiki/Cadence_SKILL

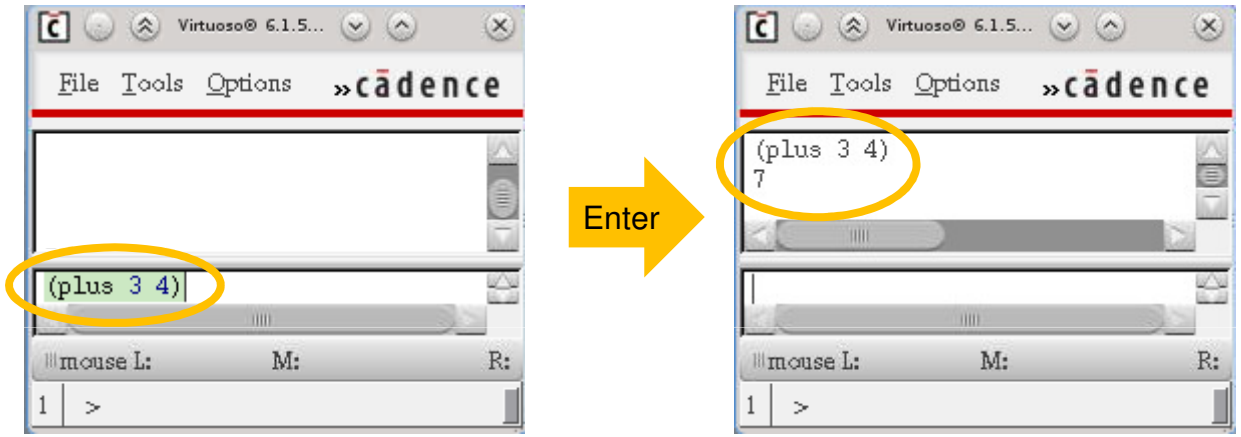
- On our Linux machines using a Web browser at /opt/eda/IC615/doc/...
  There you find for instance

Path	Purpose
sklangref/sklangrefTOC.html	Structure, Basic Commands
sklanguser/sklanguserTOC.html	Data structures
skdevref/skdevrefTOC.html	Routines
skdfrefTOC.html	Data objects
sklayoutref/sklayoutrefTOC.html	Layout specific stuff

- Best save some links in your browser!
  - If you use ssh, you can start for instance firefox

How to execute SKILL commands

- You can type commands directly in the Main CIW (Command Interpreter Window):



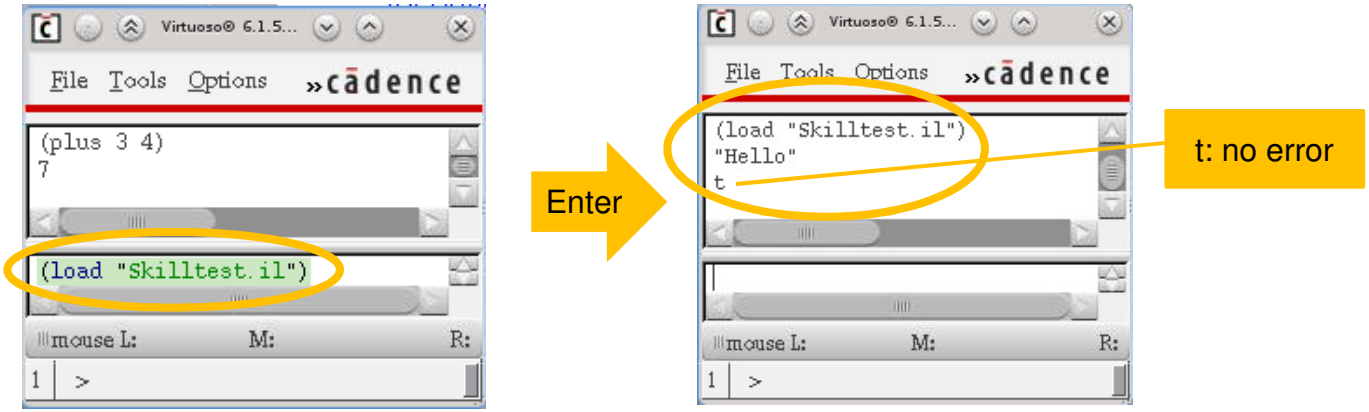
- You get back old entries with the **arrow up** key
- You can *select output* with the mouse and *paste it back* to the entry line with the **middle mouse button**

Automatic Execution of SKILL

- You can put code in a file (extension *.il) and load the file with **`( load "filename.il" )`**

In file “Skilltest.il”:

```
(print "Hello")
```



- Code in the file **`.cdsinit`** (in the directory from where you start cadence) is executed at startup of cadence

- In this file, you can
  - Define bindkeys (see exercise 4)
  - Define your own commands
  - Call other skill files

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

BASIC OBJECTS: ATOMS & LISTS

Objects: Atoms and Lists

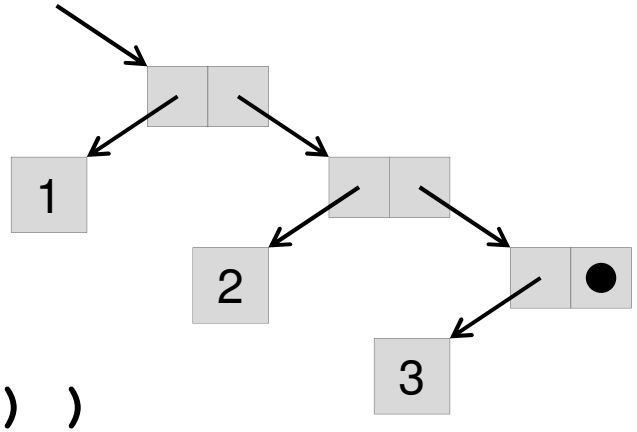
- An ***atom*** is a simple object:
  - numbers (integers, floats)
  - The boolean values `t` (true) or `nil` (false)
  - pointers (see later)
  - The function `atom` checks if the argument is indeed atomar:
    `(atom 5)` → `t`

- A ***list*** is a sequence of elements
  - Lists are created by: `( list obj obj ... )` → a list
  - Equivalent: `list( obj obj ... )`
  - Short hand notation: `` `( obj obj ...) ``
    (objects are *not* evaluated, works mostly only in top level!)

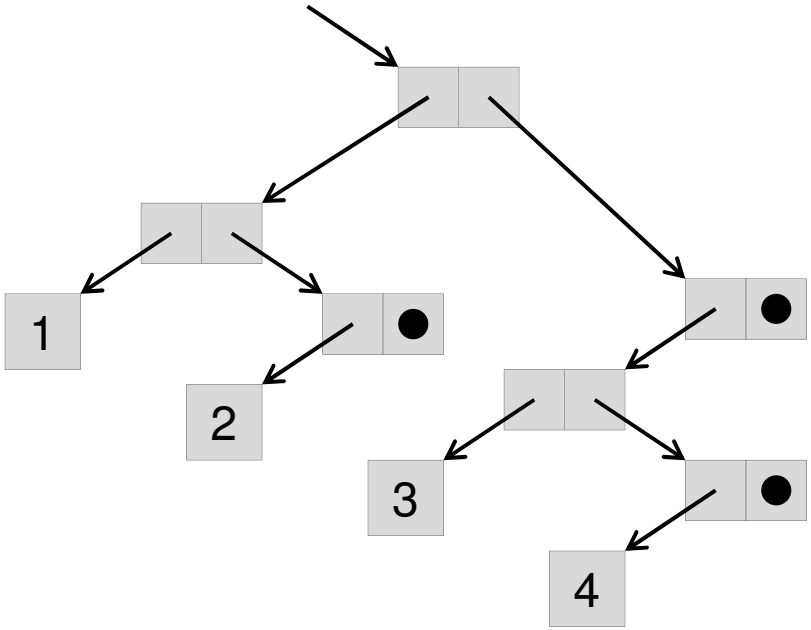
  - They are displayed as `(obj obj ...)`
  - Each element can be an atom or another list: `` `( (list 1 2) 3) ``
  - `(listp obj)` checks if an object is a list

Examples for lists

- `(list 1 2 3)`

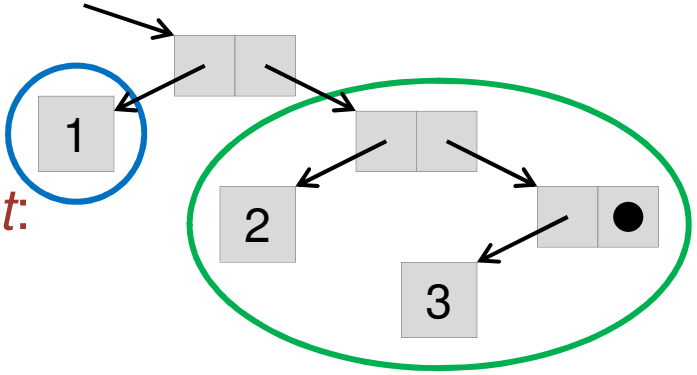


- `(list (list 1 2) (list 3 4) )`



Accessing Parts of lists

- The first element of a list x is `(car x)`, the rest is `(cdr x)`:
  - `(car '(1 2 3))` → `1`
  - `(cdr '(1 2 3))` → `(2 3)`

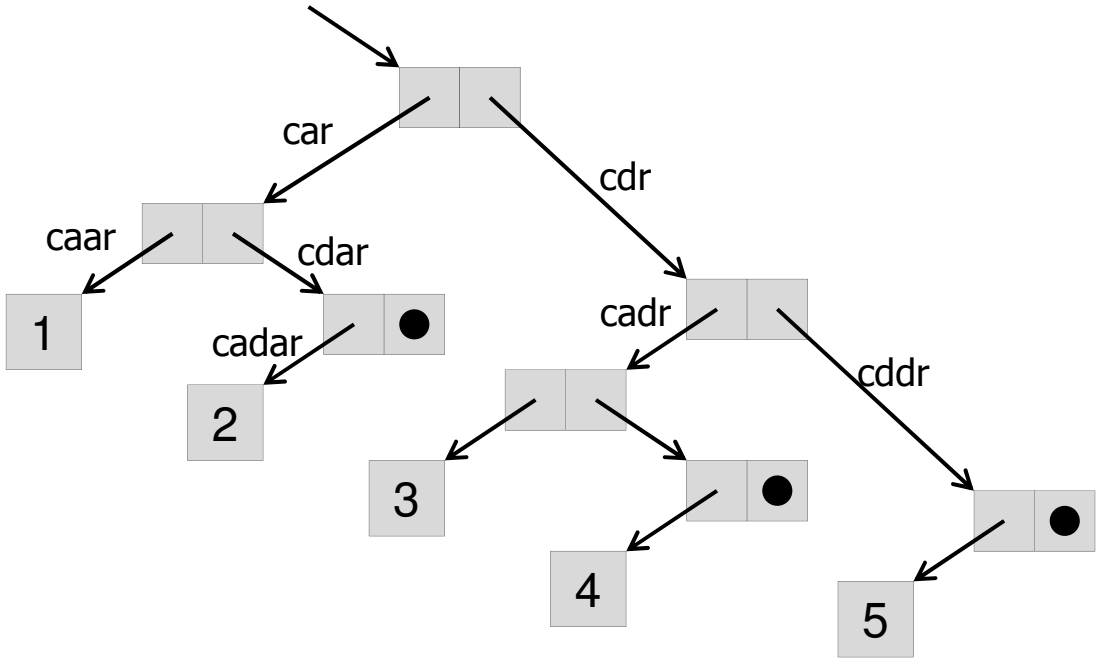


- Note that `cdr` always returns a *list*:
  - `(car '(1 2))` → `1`
  - `(cdr '(1 2))` → `(2)`

- Extensions for nested lists are `caar`, `cadr`, `cdar`, `cddr`,... (starting evaluation 'at the back'):
  With x = `'( (1 2) (3 4) 5 )`: (see also next page)
  - `(car x)` → `'(1 2)`
  - `(cdr x)` → `'(3 4)`
  - `(caar x)` → `1`
  - `(cdar x)` → `(2)`
  - `(cadar x)` → `2`
  - `(caadr x)` → `3` (note two 'a' !)

Accessing Parts of a List

- `(list (list 1 2) (list 3 4) 5 )`

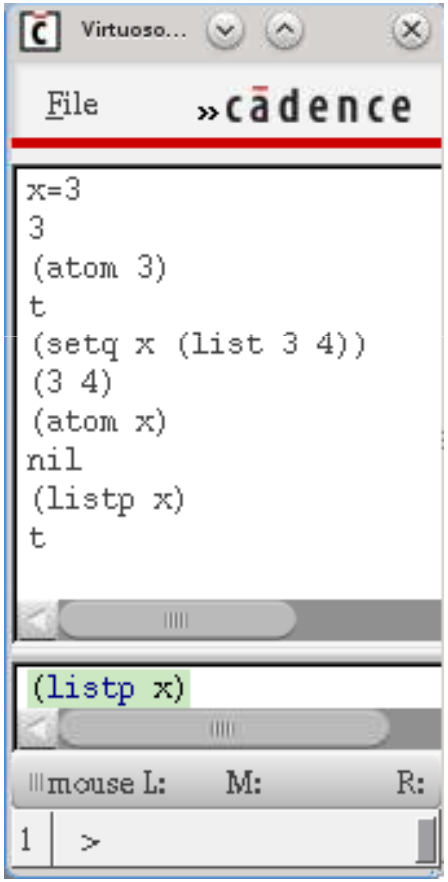


More List Commands

- Get the length of a list (or array / table / ...) (top level!):
  - `(length object)`
  - `(length '(a b c d))` → `4`
- Pick the n-th element (first element has index **0**):
  - `(nth index list)`
  - `(nth 2 '(a b c d))` → `c`
- Add an element to (the front of) a list:
  - `(cons element list)`
  - `(cons 5 '(a b c d))` → `(5 a b c d)`
  - Note: `list` is not changed! To change it, re-assign it:
  - `ll = (cons 5 ll)`
  - You can also `append` at the end, but this is slower!

More List Commands

- The check if an object is a list:
  - **`(listp object)`** → **`t`** or **`nil`**



Points and Rectangles

- A *point* is a *list* of two (float) values
- There is a short hand notation to enter such a list
  - `3:4` → `(3 4)`
- To extract the coordinates, one can use
  - `(xCoord p)` equivalent to `(car p)`
  - `(yCoord p)` equivalent to `(cadr p)`

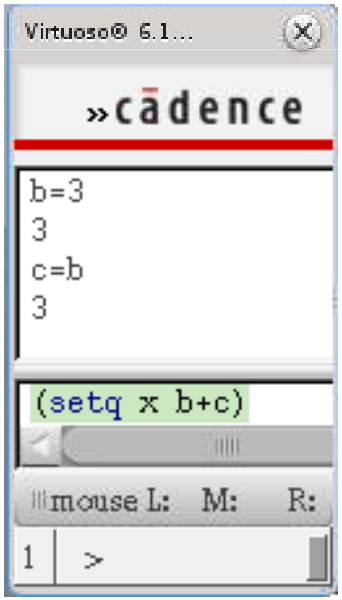
  (note capital 'C'!)

- A *rectangle* is a list of two points
  - `list(3:4 10:12)` → `((3 4) (10 12))`

Variables

- Variables do *not* need to be *declared*, they are just used
- Assignment can be done with
  - **`var = expression`**

  or
  - **`(setq var expression)`**
- Note that expression are evaluated:



RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Control Structures

Conditional Execution - if

- Readable version:
  - `if(` condition `then` expression1 `else` expression2 `)`
- More compact 'lisp' version:
  - `( if condition expression1 expression2 )`

- Examples:
  - `(if t 4 6)` → `4`
  - `(if (greaterp 6 7) 4 6)` → `6`
  - `(if 3+4>3*4 then print("yes") else (print "no"))`
    → „no"

Logical Expressions

- Boolean values can be true (**t**) or false (**nil**)
- Normal operators work: **<**, **==**, **<=**, ..
  - The function equivalents have mostly a 'p' at the end:
  - **(greaterp 5 4) (leqp 6 7)**
- Several functions return a Boolean value:
  - **(oddp 7)** → **t**
  - **(plusp -3)** → **nil**
  - **(zerop 0)** → **t**
- WATCH OUT: There are several versions of **eq, equal**,… which check content or addresses – see documentation:

```
p1 = `(1 2)     p2 = `(1 2)
(equal p1 p2)       →     t     // same values
(eq    p1 p2)       →     nil   // different objects!
```

Loops

- `(for var initial_value final_value expressions)`
  (loop variable is incremented by one!)
- `(while condition expressions)`

- Examples:
  - `(for i 1 9 (print i))` → `123456789`
  - `(setq i 1)`
    `(while i<100 i=2*i (printf "%d " i))`
    → `2 4 8 16 32 64 128`

- Also:
  - `(when …)`
  - `(unless …)`
  - `(case …)`

Very Useful: Foreach

- All elements of a list can be processed with 'foreach':

  `(foreach name list expression)`

  - Variable **`name`** is assigned an element of **`list`** and **`expression`** is executed. This is repeated for all elements of **`list`**.

- Example:

  - **`(foreach x '(1 2 3 5) (println x*x))`**

    →

    ```
    1
    4
    9
    25
    ```

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

PROCEDURES

Procedures

- A procedure can be declared with

```
procedure(
   name( arg1 arg2 …)
   commands
   …
   result of last command is return value
)
```

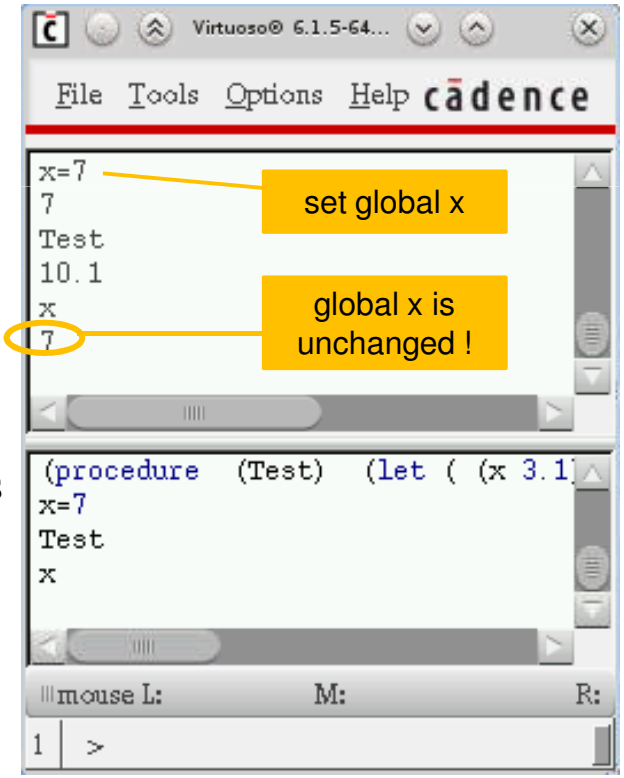
- Example:
  - `procedure( square(x) x*x)`
  - `square(4)` → `16`
- Alternative syntax:
  - `( procedure ( square x ) x*x )`
  - `( square 4 )`

(Local Variables)

- When defining procedures, it is recommended to declare variables *locally*. This can be done using a **`let`** – block:
- **`( let ( list of local variables ) commands )`**
- The local variables in the list can be
  - Declared by just naming them
  - Initialized using **`(name value)`**

- Example:

```
( procedure
    ( TestProcedure ) ;no args
    ( let
      ( (x 3.1) (y 3+4) z ) ;vars
      z = x + y
    ) ; end of let
) ; end of procedure
```



RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

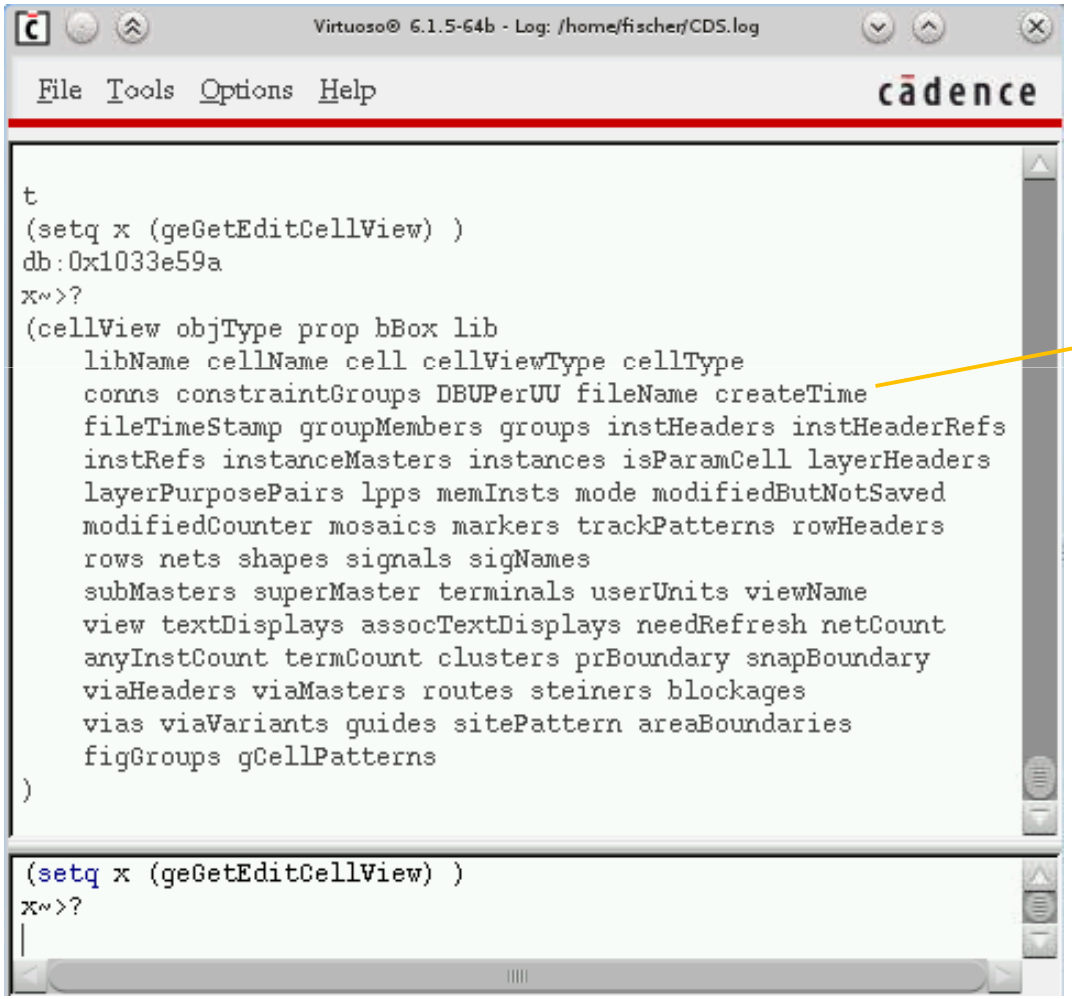
THE CADENCE DATABASE

Objects in the DataBase

- All objects used in cell views (wires, pins, labels, shapes, contacts,..) are stored in a ***data base***.
- Access to objects is via their *unique data base object identifier*, or *ID*
- Objects have properties (or 'attributes')
- The access operator to the properties is **`~>`**
- A list of all attributes can be shown with **`ID~>?`**
- Attributes & their values are listed with **`ID~>??`**

Getting access to an object (get the ID)

- With an open cell view (layout or schematic), the command **`(geGetEditCellView)`** gets the ID

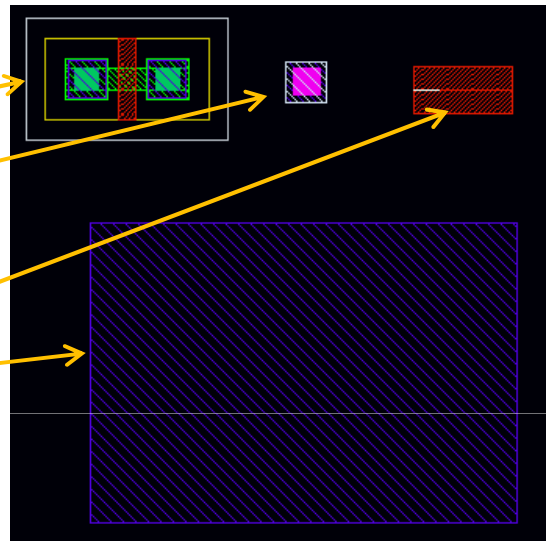


These are all the properties of the cell view

Looking at cell view properties

- Once we have a view ID, we can access the properties:

```
x~>cellName
"SKILLView"
x~>viewName
"layout"
x~>instances
(db:0x1033c91a)
x~>vias
(db:0x1033c61a)
x~>shapes
(db:0x1033e09a db:0x1033e09b)
```

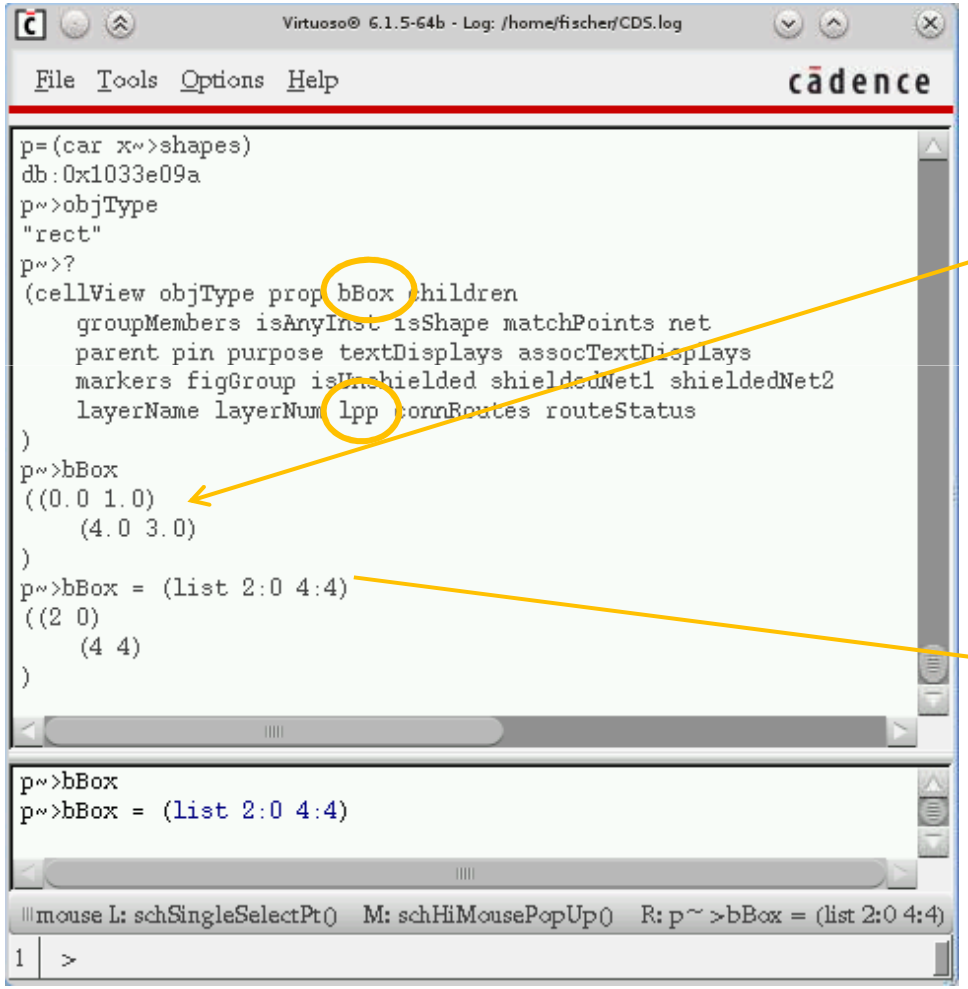


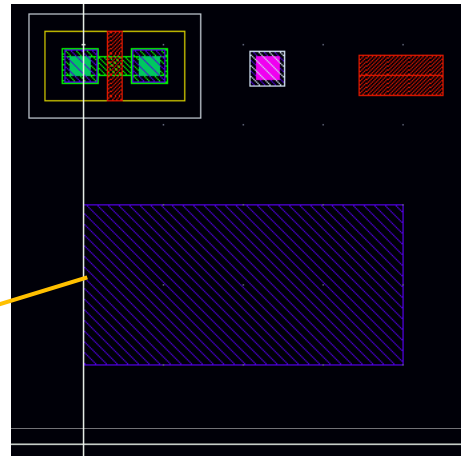
- The properties
  `instances,vias,shapes,layerPurposePairs (= lpp),..`
  are again *lists* of object **`Id`**s
- They can be studied further:

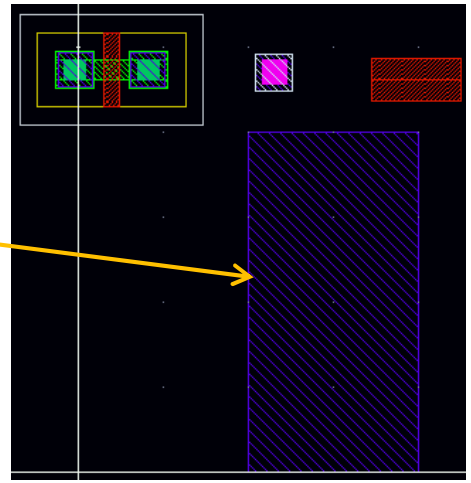
```
(car x~>shapes)~>objType
"rect"
(cadr x~>shapes)~>objType
"path"
```

Modifying Objects

- The properties can be modified and affect the open view immediately:







Modifying Objects - 2

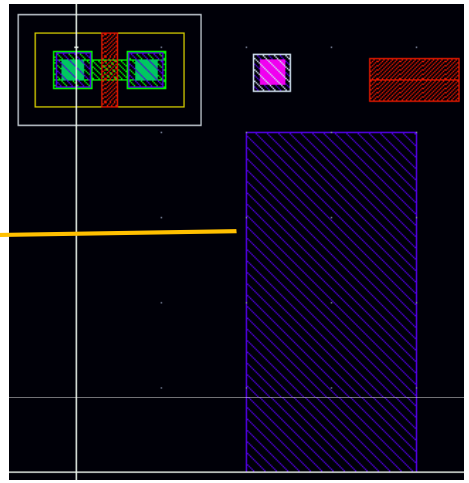
- The **`layerPurposePairs`** tells on which layer an object is
- It can be modified...

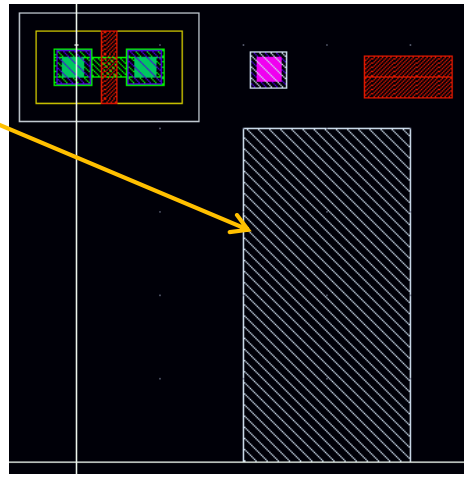
```
p~>lpp
("ME1" "drawing")
p~>lpp = '("ME2" "drawing")
("ME2" "drawing")
```

```
p~>lpp
p~>lpp = '("ME2" "drawing")
```

mouse L: mouseSingleSelectPt p~>lpp = '

1 >





Creating New Objects

- There are many commands to create objects, see *skdfref*
- For instance, create a new rectangle with

  `(dbCreateRect ID layer list(x:y x:y))`: