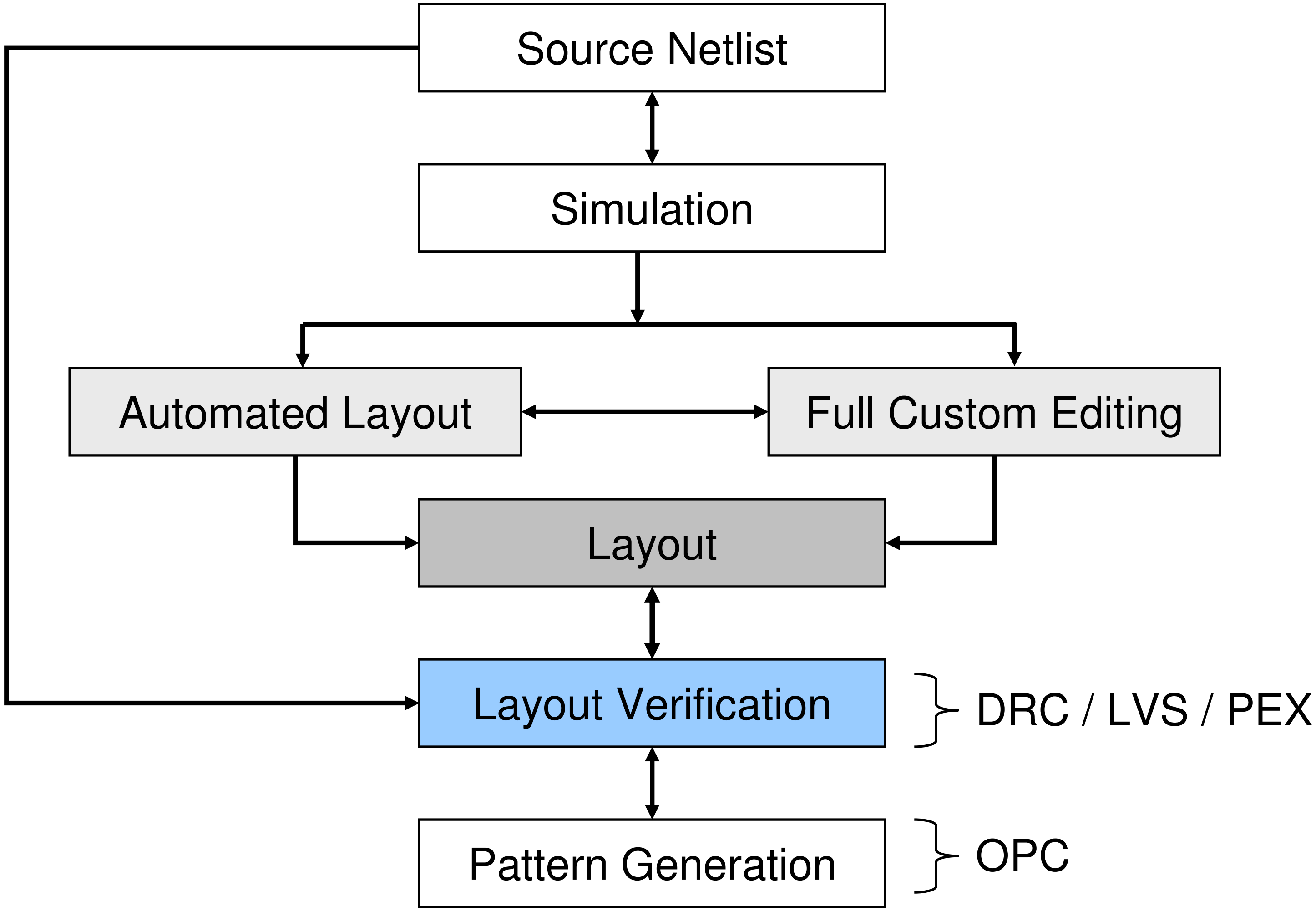
Calibre® Rule Writing

Module 1

Basic Concepts

Typical IC Design and Verification Flow



Source Netlist

Simulation

Automated Layout

Full Custom Editing

Layout

Layout Verification

Pattern Generation

DRC / LVS / PEX

OPC

What Is a SVRF File?

- Standard Verification Rule Format (SVRF) file—rule file
  - Used by Calibre and ICverify physical verification tools
  - A language standard that controls tool functionality
- The rule file has two main elements:
  - Operations
  - Specification statements

What Are Operations?

Operations work on the layout data:

- **Layer derivation**
  - **Generates polygons**
  - **Generates edges**
  - **Generates edge segments**
- **Connectivity extraction**
  - **Recognizes electrically-connected regions (nets) in the layout**
- **Device recognition**
  - **Identifies devices from layout geometry**
- **Text attachment**
  - **Assigns label names to nets establishing initial correspondence points between the source and the layout**

What Are Specification Statements?

- Specification statements control the environment
- Examples:
  - Layer definition
  - Cell exclusion
  - Results
    - Specifies the filename and type of results database
    - Controls the report file
    - Controls the output of DRC
  - File
    - Controls where to find the input and output files

How Do I Create a Rule File?

- **From scratch using an ASCII text editor**
- **Copy and modify an existing Calibre rule file**
- **Convert a Dracula® rule file**
  - **From the Command Line:**

    `$MGC_HOME/bin/drac_cvt` *`sourcefile destpath`*

    - ***`sourcefile`* Dracula command file pathname**
    - ***`destpath`* rule file pathname you want created**
- **Use the Calibre GUI**
  - **Does not write a complete rule file**
  - **Adds `INCLUDE` to the rule file to append “golden rules”**

Rule File Compilation

- **The rule file must be compiled before use.**
  - **Automatic when you invoke Calibre from the command line.**
  - **Occurs when you Load the rule file in the GUI.**
- **Compilation involves checking for:**
  - **Correct syntax**
  - **Correct layers for a particular operation**
- **Compilation resolves all dependencies between statements and operations.**

If you have a compilation failure, the error is reported. Fix the error and run Calibre again.

Repeat this process until you get a successful run.

SVRF Statement Syntax Conventions

- The next slides preview several selected SVRF statements.
- They illustrate the following syntactic conventions:
  - Parameter Order
  - Case sensitivity
  - Literal keywords versus variable parameters
  - White space considerations
  - Reserved keywords
  - Reserved symbols

STATEMENT OVERVIEW

Statement Syntax

DRC Maximum Results

CLASS: SECONDARY KEYWORD

Purpose: Specifies the maximum number of results per RuleCheck written to the DRC results database

Syntax: `DRC MAXIMUM RESULTS {`*`maxresults`*` | ALL}`

Parameters: *maxresults* — non-negative integer that specifies the ... n nu ... cou

Default: 1000

Example: `DRC MAXIMUM RESULTS all`

Primary keywords in a statement may not be interchanged.

Enclosed within the { } braces is a list of **required** parameters.

The vertical bar indicates an either / or choice between items.

All statements are case insensitive*.

* Except for cell names, filenames, and possibly net names

STATEMENT OVERVIEW

Statement Syntax (Cont.)

CLASS: SECONDARY KEYWORD

DRC Summary Report

Purpose: Specifies the DRC summary report filename and method in which it is written report

Syntax: `DRC SUMMARY REPORT *filename* [REPLACE | APPEND] [HIER]`

Parameters: t file that is created or appended to during

The italic font indicates a user-supplied argument—a variable.

REPLACE APPEND establishes the writing meth

ed;

Underlined secondary keyword indicates default behavior.

Enclosed in [ ] brackets are **optional** parameters.

REPLACE —

Separate words with at least one white space character.

ry

APPEND — appends to an existing summary report file

ck statistics by layout database cell

This pathname must be enclosed in quotation marks because it contains special characters.

Default: REPLA

Example: `DRC SUMMARY REPORT "../drc_report" HIER`

STATEMENT OVERVIEW

Statement Syntax (Cont.)

CLASS: SECONDARY KEYWORD

AND

Purpose: Selects all polygon regions common to one or more polygons

Syntax:

```
AND layer0 [constraint] //single-layer AND
AND layer1 layer2       //two-layer AND
```

You can write this statement as a single-layer operation or a two-layer operation.

al layer

– origi

optional

range of

The ordering of parameters in certain operations is very flexible.

For example, the following four statements generate the same geometric output:

```
AND metal poly
metal AND poly
poly AND metal
metal poly AND
```

The ordering of layers in this operation does, however, affect connectivity.

- Single-layer AND operates on pre-me
  - Single-layer AND selects polygon regi polygons corresponding to the constra
  - Constraint ==0 results in empty outp
- Two-layer AND operates on merged d
  - A layer derived from a two-layer AND

Statement Syntax (Cont.)

STATEMENT OVERVIEW

CLASS: SECONDARY KEYWORD

Perpendicular

Purpose: Measures perpendicular edges

Syntax: **INTERNAL *layer1 layer2 constraint***
[NOT] PERPENDICULAR [ONLY|ALSO]

Parameter: ONLY – Measures only perpendicular edges

dges

Default:

You must order the words within a secondary keyword as shown in the syntax for the statement. For example, the following three statements are valid:

```
INTERNAL metal < 5 PERPENDICULAR ONLY

INTERNAL metal < 5 PERPENDICULAR ALSO

INTERNAL metal poly < 5 NOT PERP
```

However, the following statement is invalid:

```
INTERNAL metal < 5 ONLY PERPENDICULAR
```

Reserved Keywords

- In general, the name of any specification statement, operation, or secondary keyword is considered to be a reserved keyword.
- Reserved keywords may not be used for the following:
  - Variables
  - Cell names
  - RuleChecks
  - Layer Names
- Questions about the status of a word?
  - Perform an automated search of the online *SVRF Manual*.

Reserved Symbols

- **Calibre recognizes all of the following symbols regardless of the absence of surrounding white space.**
- **Calibre reserved symbols:**

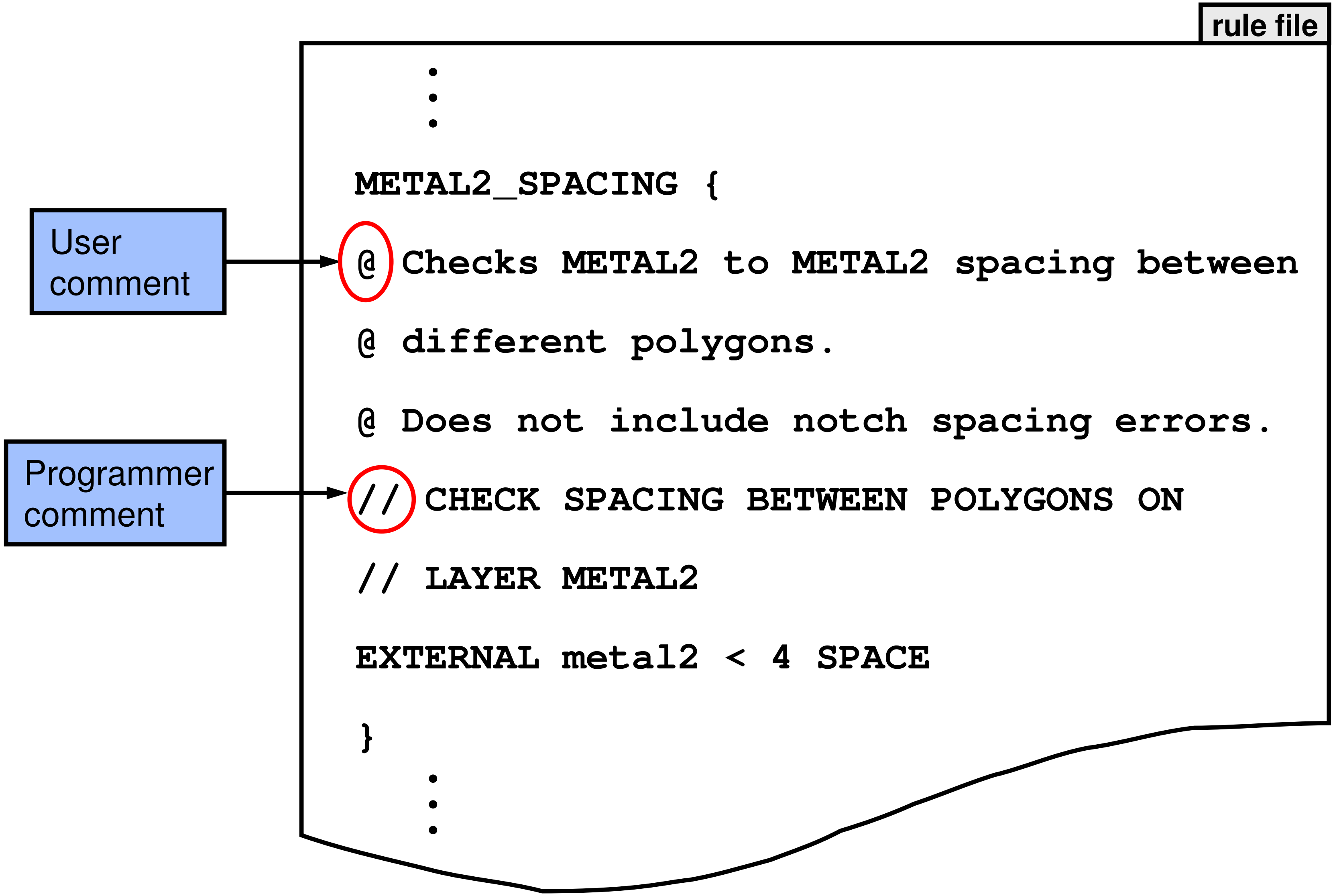
```
//   @   /*   */   {   }   "   `   (   )   [   ]   <   ==   >

<=   >=   !=   -   +   *   /   !   %   =   &&   ||   ::   ,   ?
```

Commenting the Rule File

- “ // ” C++ style comments
  - Begin anywhere on a line
  - Terminate at the end of the line where they occur
- “ /* … */ ” C-style comments
  - Begin anywhere on a line
  - May span multiple lines
  - Terminate at the “ */ ” combination
  - May not be nested
- “ @ ” DRC User comments
  - May begin anywhere on a line within a RuleCheck
  - Terminate at the end of the line
  - Use the RVE tool to view violations—DRC user comments

Example #1 of Using Rule File Comments



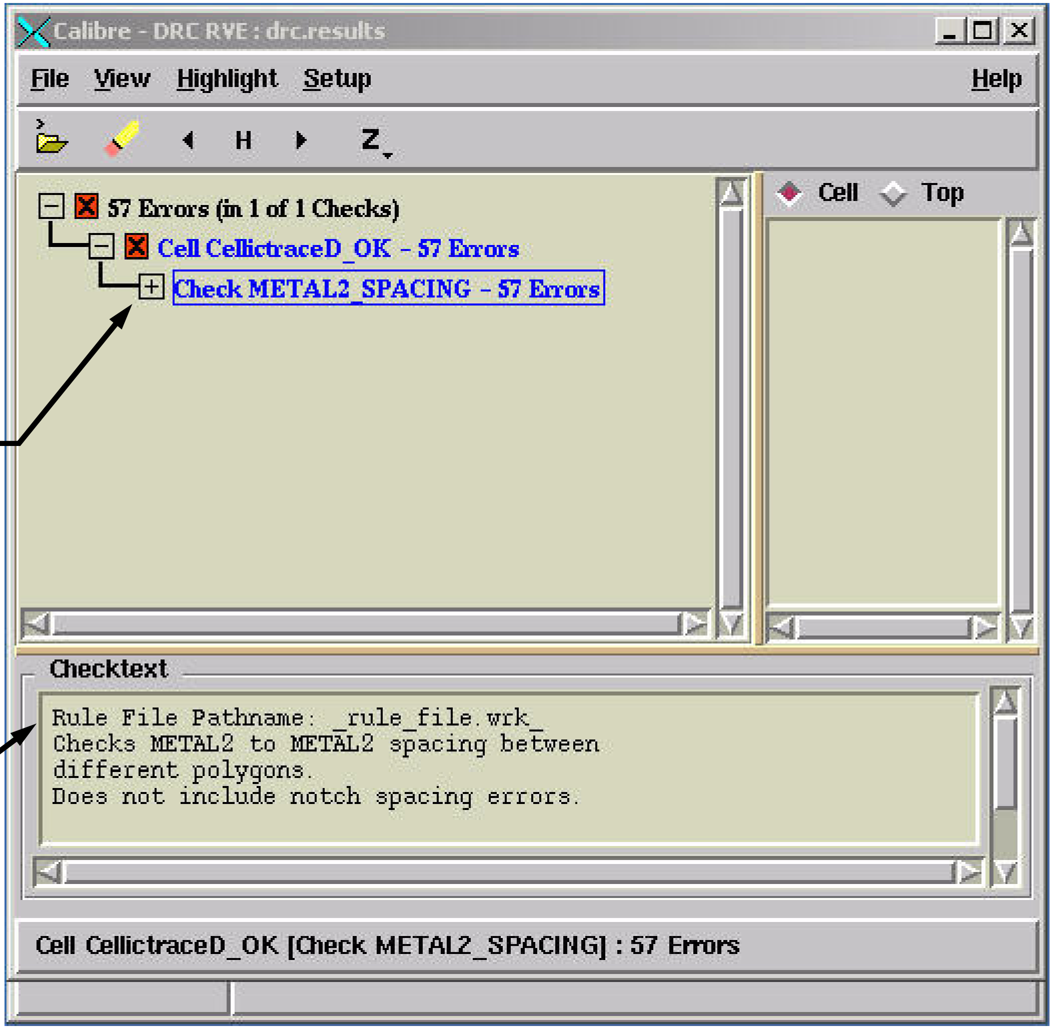
rule file

User comment

Programmer comment

```
    ⋮
METAL2_SPACING {
@ Checks METAL2 to METAL2 spacing between
@ different polygons.
@ Does not include notch spacing errors.
// CHECK SPACING BETWEEN POLYGONS ON
// LAYER METAL2
EXTERNAL metal2 < 4 SPACE
}
    ⋮
```

Example #1 of Using Rule File Comments (Cont.)



```
Rule File Pathname: _rule_file.wrk_
Checks METAL2 to METAL2 spacing between
different polygons.
Does not include notch spacing errors.
```

Cell CellictraceD_OK [Check METAL2_SPACING] : 57 Errors

DRC RuleCheck name

DRC User comments appear in the Calibre results viewer to aid the layout designer.

Example #2 of Using Rule File Comments

Comments out everything between the asterisks.

Be careful with these!

rule file

```
    .
    .
    .
/*

METAL2_SPACING {

@ Checks METAL2 to METAL2 spacing between

@ different polygons

// CHECK SPACING BETWEEN POLYGONS ON

// LAYER METAL2

metal2 EXTERNAL < 4 space

}

*/
```

Rule File Variables

- **Variables can be used in rule files as statement parameters.**
- **Rule file variables can be defined in two ways:**
  - **“Inside” the rule file via the `VARIABLE` statement**
  - **“Outside” the rule file as Unix environment variables**
- **To use a variable defined inside the rule file:**

```
VARIABLE pspace 3.0
poly_spacing {EXT poly < pspace}
```

- **To use a variable defined outside of the rule file:**

```
setenv pspace 3.0
```

C Shell

```
VARIABLE pspace ENVIRONMENT
poly_spacing {EXT poly < pspace}
```

SVRF file

- **To use an environment variable in a file path (note “$”):**

```
LAYOUT PATH “$my_chip/layout/$version/chip.gds”
```

NOTE: Environment variables used only in paths do not need to appear in a **`VARIABLE`** statement.

Using a Rule File Variable in a Comment

- **Variable values can be seen in user comments viewed in RVE.**
- **To do this, precede the variable name with the “^” character:**

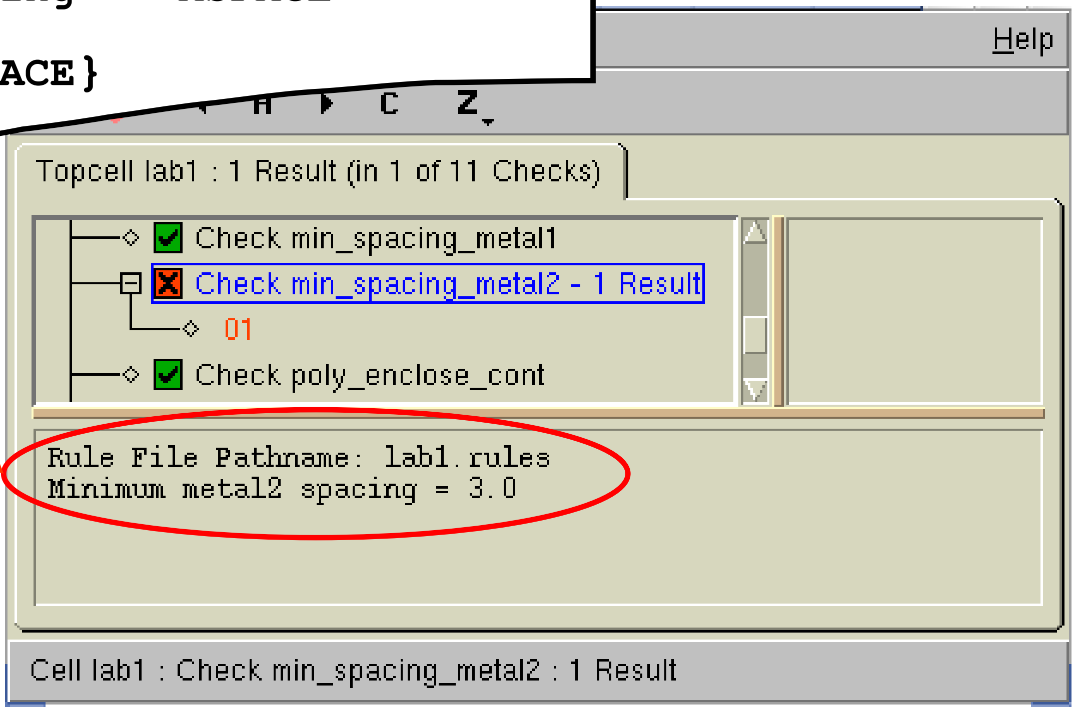
```
VARIABLE MSPACE 3.0  // declare variable

min_spacing_metal2 {

@ Minimum metal2 spacing = ^MSPACE

EXTERNAL metal2 < MSPACE}
```

Enables Calibre to resolve a variable inside the RuleCheck comment.



Topcell lab1 : 1 Result (in 1 of 11 Checks)

☑ Check min_spacing_metal1
☒ Check min_spacing_metal2 - 1 Result
01
☑ Check poly_enclose_cont

```
Rule File Pathname: lab1.rules
Minimum metal2 spacing = 3.0
```

Cell lab1 : Check min_spacing_metal2 : 1 Result

Including a Rule File

- Use the SVRF specification statement:
  - `INCLUDE` *`filename`*
- Uses the entire text of the included file as if it were in the parent file.
- The `INCLUDE` statement may appear anywhere in a rule file.
- Calibre processes all `INCLUDE` statements first.
- Allows you to control which statements are write-protected and which statements may be modified during layout debug.
- Nesting include files is allowed; recursion is not.
- Make sure to archive/save rule files or you may lose information.
- Example:
  - `INCLUDE “/user/joe/work/rulefile”`

Using Wildcard Characters

- **The question mark (?) wildcard character**
  - **Matches zero or more characters (unlike UNIX or NT).**
  - **Several SVRF statements allow this wildcard when referring to names other than cell names.**
  - **Example:**
    - `GROUP tapeout_checks “level?”`
- **The asterisk (*) wildcard character**
  - **Matches zero or more characters.**
  - **Several SVRF statements allow this wildcard when referring to cell names.**
  - **Example:**
    - `EXCLUDE CELL “ADDER*”`

Pre-Processor Directives (Conditionals) — `#DEFINE`

- **Pre-Processor Directives are structures permitting conditional compilation of rule file text.**
- **`#DEFINE` and `#UNDEFINE` keywords within the rule file or variables defined in the shell environment control conditional compilation.**
- **Syntax :**

  `#DEFINE` ***`name`*** **`[`*`value`*`]`**

  `#UNDEFINE` ***`name`***

  - ***`name`*** **is a mandatory string**
  - ***`value`*** **is an optional string**
- **If a *`name`* is defined in the shell environment, then it is considered defined in the pre-processor if it is dereferenced as `$`*`name`*.**
  - ***`name`*** **does not need to appear in a `VARIABLE` statement within the rule file.**
  - **If *`value`* is specified, it supercedes the value stated in the shell.**

Pre-Processor Directives (Conditionals) — #IFDEF

- **Conditionals have the following form:**

```
#IFDEF name [value] rule_file_text
    [#ELSE rule_file_text ]
#ENDIF
```

or

```
#IFNDEF name [value] rule_file_text
    [#ELSE rule_file_text]
#ENDIF
```

- **Precede *`name`* with "`$`" if *`name`* is defined as an environment variable (in this case, *`name`* does not need to appear in a `#DEFINE` statement).**

`rule_file_text` is executed when using:

`#IFDEF` if the ***`name`*** is defined (and equals ***`value`***)

`#IFNDEF` if the ***`name`*** is not defined (or does not equal ***`value`***)

#IFDEF Example

```
LAYER metal4 23
LAYER metal5 26
LAYER metal6 14

#IFDEF $P1
LAYER top_metal metal6
  #ELSE
     #IFDEF $P2
     LAYER top_metal metal5
          #ELSE
               LAYER top_metal metal4
     #ENDIF
#ENDIF
```

In this rule file example, process **`P1`** states that **`metal6`** is the top metal layer, process **`P2`** states that **`metal5`** is the top metal layer, and in all other processes **`metal4`** is the top metal layer. The desired process is specified by defining the appropriate environment variable (**`P1`**, **`P2`**, or neither).

Layout Input Statements

The next three statements specify the target layout:

- **`LAYOUT SYSTEM` — type of layout file**
- **`LAYOUT PATH` — path to file**
- **`LAYOUT PRIMARY` — top cell**

Layout System

Purpose:	Specifies the layout database type
Syntax:	**`LAYOUT SYSTEM`** ***`type`***
Parameters:	***`type`*** — keyword examples: GDSII, OASIS, LEFDEF, OpenAccess, Milkway
Default:	**none**
Example:	**`LAYOUT SYSTEM GDSII`**

- **You must specify this statement once in the rule file.**

Layout Path

Purpose:	Specifies the layout database pathname(s)
Syntax:	`LAYOUT PATH {filename [ …filename]|STDIN}`
Parameters:	***`filename`*** — pathname of the layout database
	`STDIN` — layout comes from standard input
Default:	**none**
Example:	`LAYOUT PATH "/tmp/work/mydesign.gds"`

- **Calibre merges multiple layout files before verification.**
- **You may specify this statement multiple times to load multiple databases.**
- **You must specify this statement at least once in the rule file.**
- **Layout file can be compressed (.gz or .Z).**
  - **Compressed file is limited to 2GB in some systems.**
  - **Size is limited by the uncompress utility.**

Layout Primary

Purpose:	Specifies a layout circuit, subcircuit, cell or symbol to verify
Syntax:	`LAYOUT PRIMARY` ***`name`***
Parameters:	***`name`*** — specifies the target design
Default:	**none**
Example:	`LAYOUT PRIMARY “cpu_topcell”`

- **Identifies the cell from which you want to start checking.**
- **Typically a top-level cell name.**
- **You must specify this statement once in the rule file for database types GDSII, OASIS, and OpenAccess.**
- **You may use “*” to match the cell name.**
  - **If more than one match, Calibre will use the first in the list.**
  - **Warning issued in this case.**

A Simple Rule File

rule file

```
//-----------------------------------

// OPTIONAL HEADER INFORMATION

//-----------------------------------

// REQUIRED DRC SPECIFICATION STATEMENTS

LAYOUT SYSTEM    GDSII

LAYOUT PATH      “./mydesign.gds”

LAYOUT PRIMARY   top_cell

DRC RESULTS DATABASE  “../drc_results”

// OPTIONAL INCLUDED RULE FILES

INCLUDE “/home/process/drc/golden_rules”
```

Data type

Data path

Where the DRC results go

A Simple Rule File (Cont.)

rule file

```
// ONE OR MORE DRAWN LAYER DEFINITIONS

LAYER diff   24            // DIFFUSION

LAYER poly   5             // POLY

LAYER metal2 9             // METAL2

LAYER via    12            // VIA

// ONE OR MORE DERIVED LAYER DEFINITIONS

gate =  poly AND diff      // GATE

sd =  diff NOT gate        // SOURCE-DRAIN
```

A Simple Rule File (Cont.)

rule file

```
// ONE OR MORE DRC RULECHECKS

min_gate_length {

       @ Gate length along POLY must be >= 3 microns.

       x = INSIDE EDGE poly diff

       INTERNAL x < 3

}
```

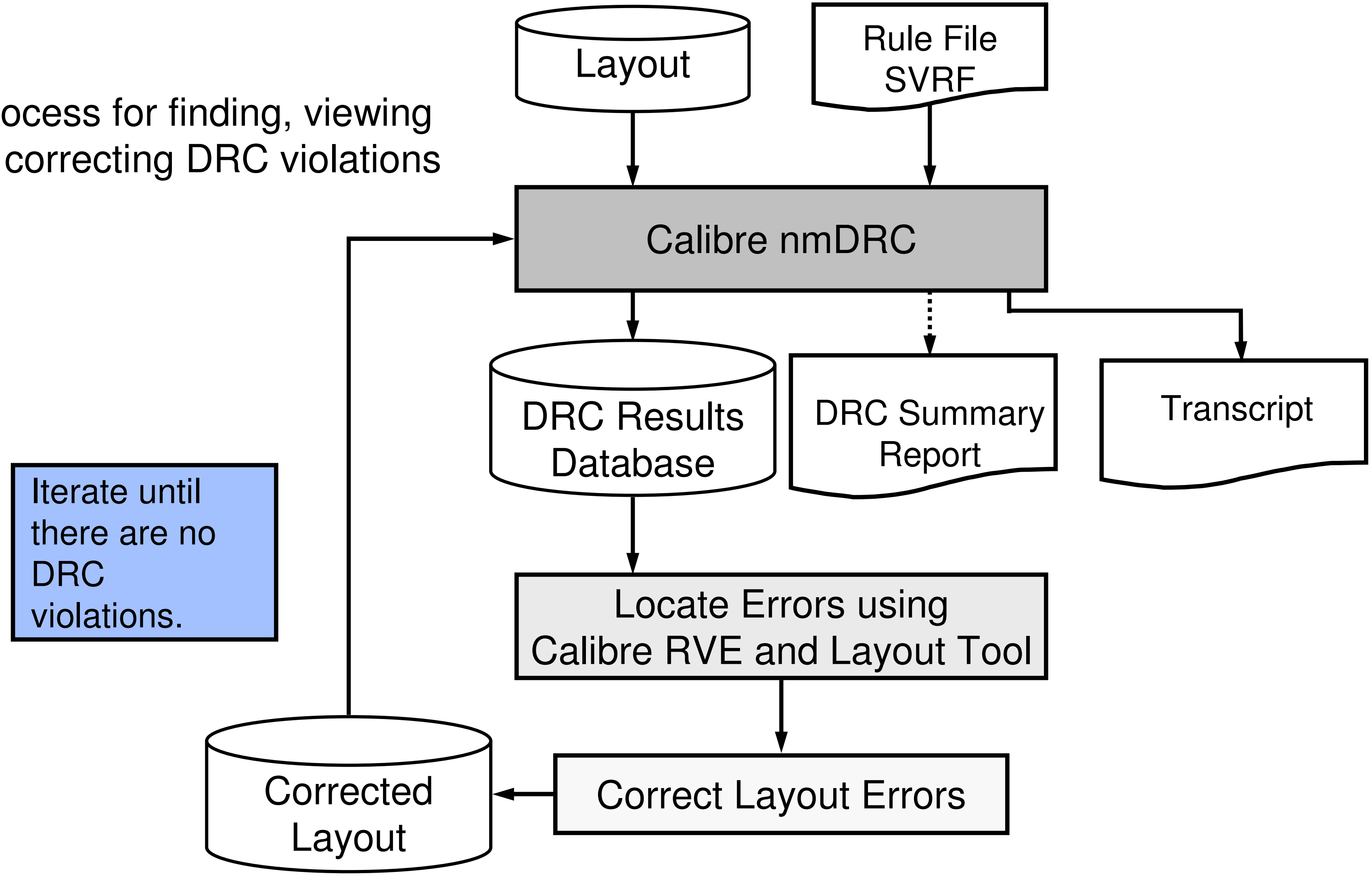
Calibre Rule Writing

Module 2

DRC Basics

The Calibre nmDRC Process

A process for finding, viewing and correcting DRC violations



Layout

Rule File SVRF

Calibre nmDRC

DRC Results Database

DRC Summary Report

Transcript

Locate Errors using Calibre RVE and Layout Tool

Correct Layout Errors

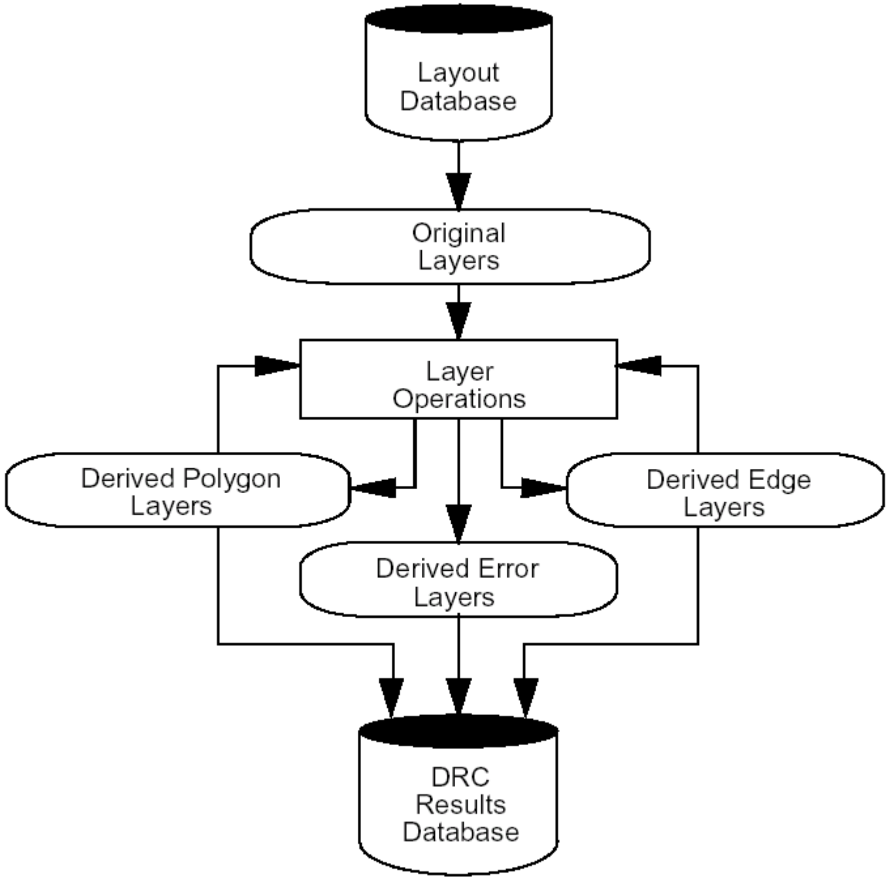
Corrected Layout

Iterate until there are no DRC violations.

Layer Types

A rule file creates or uses data from four types of layers:

- Original (drawn)
- Derived polygon
- Derived edge
- Derived error

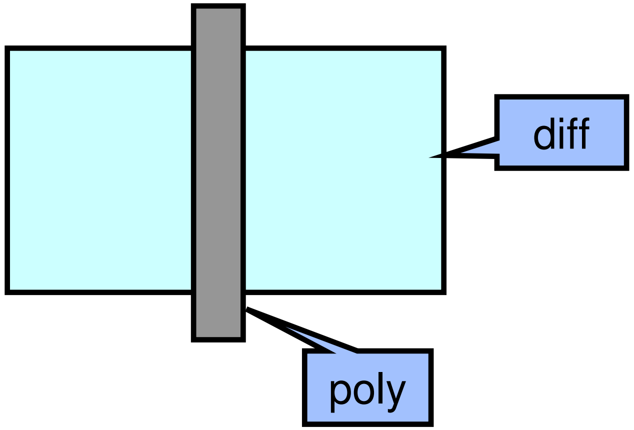


Layer Types and Data Flow in the DRC System

Layer Types — Drawn Layers

Drawn layer—also known as original layer:

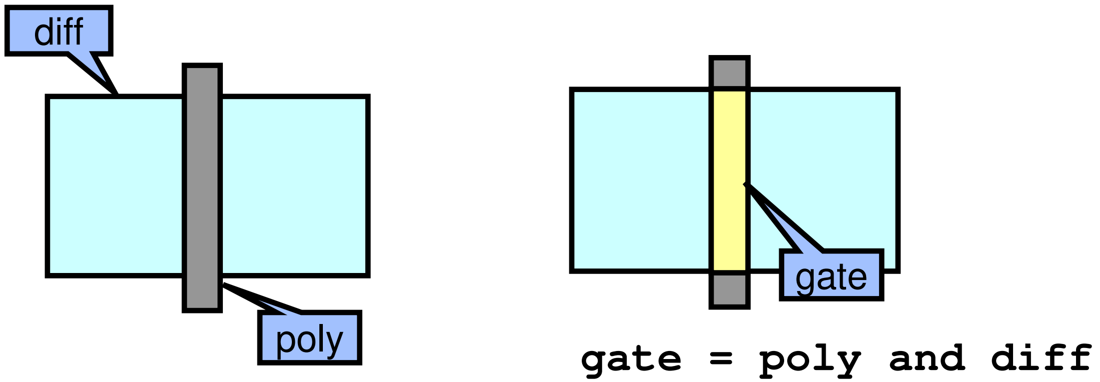
- **Original layout data**
- **Defined via SVRF `LAYER` statement:**
  - `LAYER diff 2`
  - `LAYER poly 4`
- **SVRF statements can refer to layers by name or number**



Layer Types — Derived Polygon Layers

Derived polygon layers—represent polygons generated as the output of layer operations:

- **Boolean operations**
- **Polygon-directed dimensional check operations**



`gate = poly and diff`

Layer Types — Derived Edge and Derived Error Layers

- **Derived edge layers—represent edges or edge segments of polygons generated as the output of layer operations:**
  - **Topological edge operations**
  - **Edge-directed dimensional check operations**
- **Derived error layers**
  - **Contain output of error-directed dimensional check operations**
  - **Cannot be manipulated by other operations**

DRC RuleChecks

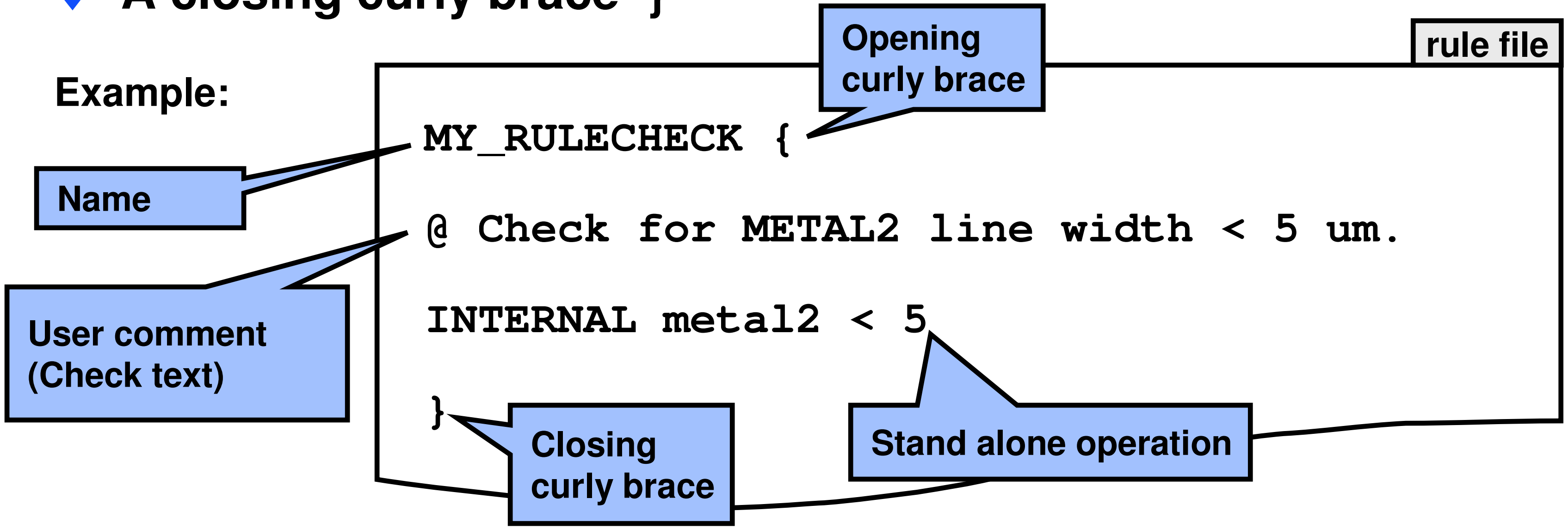
- A *RuleCheck* is a procedural statement structure added to the rule file to check one or more design rules.
- The rule file specifies which RuleChecks Calibre executes.
- Calibre RuleCheck sequence:
  - Evaluate statements
  - Output resulting data—DRC results database
- Calibre only keeps layer data in memory until it is no longer needed by another RuleCheck.
- Improve memory resource management and run time:
  - Group all RuleChecks together for a given derived layer immediately after layer derivation.

DRC RuleCheck Syntax

RuleChecks consist of:

- A name
- An opening curly brace {
- One or more (optional) layer definitions
- At least one stand-alone operation
- Optional comment text
- A closing curly brace }

Example:



```
MY_RULECHECK {

@ Check for METAL2 line width < 5 um.

INTERNAL metal2 < 5

}
```

DRC Constraints

- Certain layer operations depend on the evaluation of mathematical expressions:
  - Dimensional measurements
  - Edge or polygon counts
- Constraints are user-specified intervals of non-negative numbers.
- Calibre selects the data set meeting the constraint.
- Write rules so the constraint catches the problem geometry.

The DRC Constraint Table

Mathematical Interpretation	Calibre Constraint Notation	Calibre Alternate Notation
$X < A$	`< A`	
$X > A$	`> A`	
$X \leq A$	`<= A`	
$A \leq X$	`>= A`	
$X = A$	`== A`	
$X \neq A$	`!= A`	
$A < X < B$	`> A < B`	`< B > A`
$A \leq X < B$	`>= A < B`	`< B >= A`
$A < X \leq B$	`> A <= B`	`<= B > A`
$A \leq X \leq B$	`>= A <= B`	`<= B >= A`

Example Using a DRC Constraint

- **Example:**
  - **Process requires minimum metal2 width of 4.00 microns**
  - **The corresponding Calibre SVRF statement:**

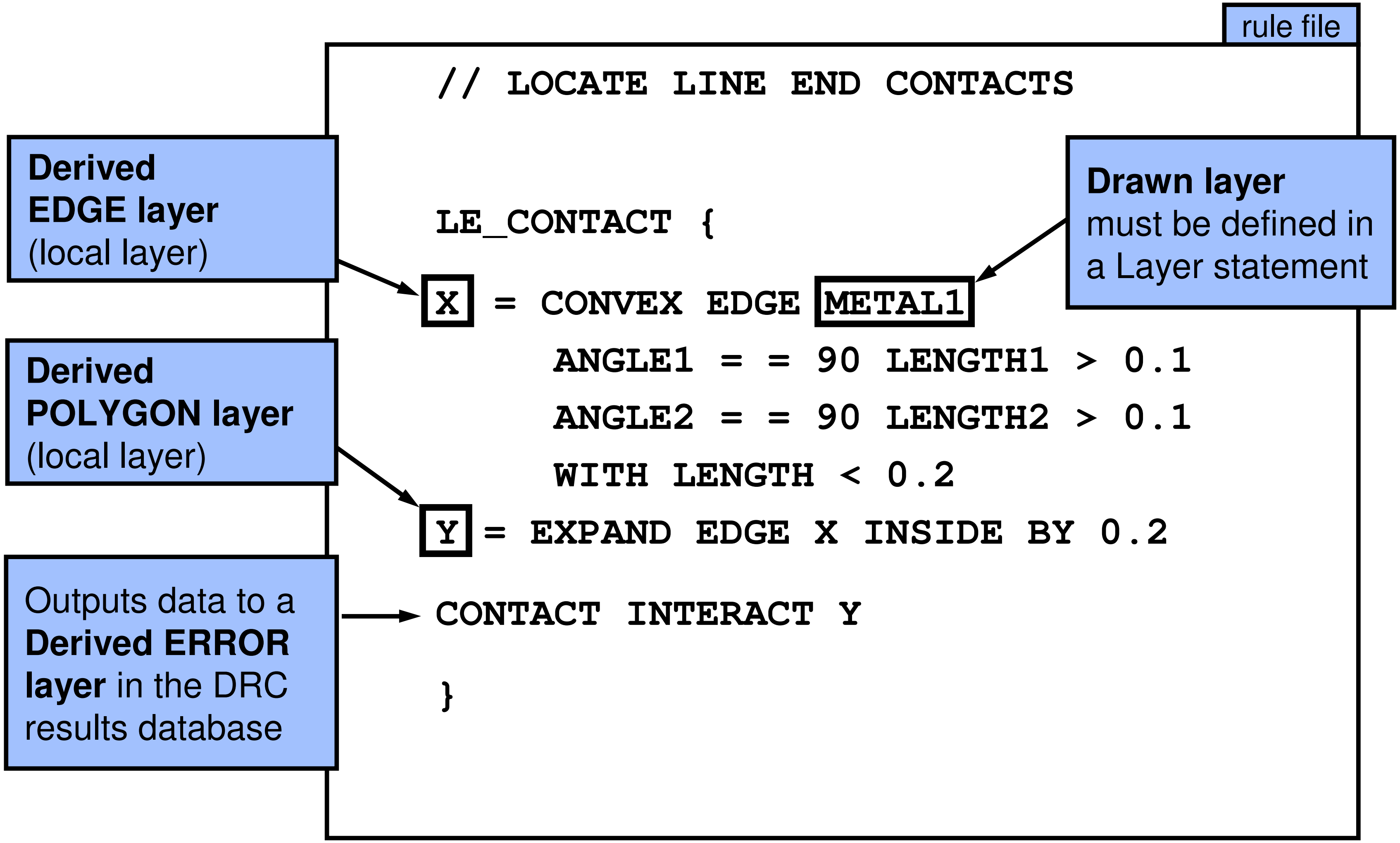
```
INTERNAL metal2 < 4
```

- **What statement does:**

  Instructs Calibre nmDRC to output to the DRC results database all layer metal2 edge pair segments having an internal spacing less than 4 microns

Note: Write statements to catch failing cases.

Example RuleCheck Using Different Layer Types



Derived EDGE layer
(local layer)

Derived POLYGON layer
(local layer)

Outputs data to a **Derived ERROR layer** in the DRC results database

Drawn layer must be defined in a Layer statement

rule file

```
// LOCATE LINE END CONTACTS


LE_CONTACT {

X = CONVEX EDGE METAL1
     ANGLE1 = = 90 LENGTH1 > 0.1
     ANGLE2 = = 90 LENGTH2 > 0.1
     WITH LENGTH < 0.2
Y = EXPAND EDGE X INSIDE BY 0.2

CONTACT INTERACT Y

}
```

Global Versus Local Layers

- **A global layer is a layer defined outside of a RuleCheck—available to all RuleChecks.**
- **A local layer is a layer defined within a RuleCheck.**
  - **Available only to the defining RuleCheck**
  - **Overrides a global layer of the same name within a RuleCheck**

Layer Operations

- A layer operation creates a derived layer from input consisting of original layers or derived layers.
- Generally, operations fall into three broad categories:
  - Edge-directed
  - Polygon-directed
  - Error-directed

Layer Operations — Classifications

Layer operations can be further classified as constructors or selectors:

- **Layer Constructors**:
  - Create new polygon data
  - Some Layer Constructors pass on node IDs depending on the operation
  - Include operations such as the Boolean operations, the `SIZE` operation and the `DENSITY` operation—These will be covered later
- **Layer Selectors**:
  - Select existing polygon or edge data from the appropriate layer selector
  - All layer select operations preserve node IDs
  - Connectivity is passed from the input layer to the derived layer
  - Include operations that have constraints (`AREA` < 4)

Layer of Origin

- Determining the layer of origin is important in dimensional check operations and net-preserving operations.
- For derived layer `y`, the layer of origin of `y` is the last layer produced by a layer constructor operation in the `y`-layer derivation chain.
- If there are no layer constructors in the `y`-layer derivation chain, then the layer of origin of `y` is the initial layer in the chain.
- The layer of origin of a drawn layer is itself.

Example:

```
A = metal AND contact
B = AREA A < 4
C = RECTANGLE B
```

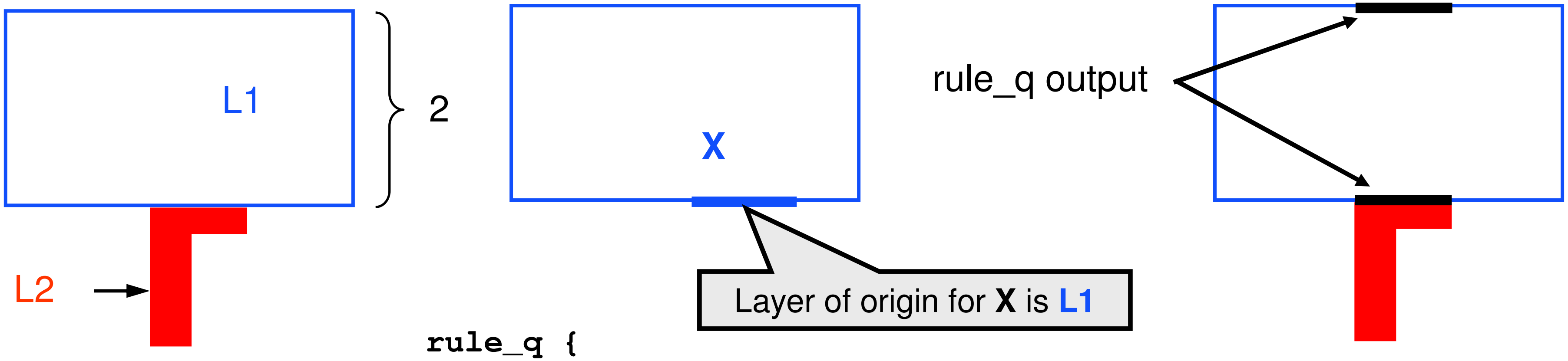
Constructor operation

Selector operation

The layer-of-origin of layer **A** is **metal** (initial layer).

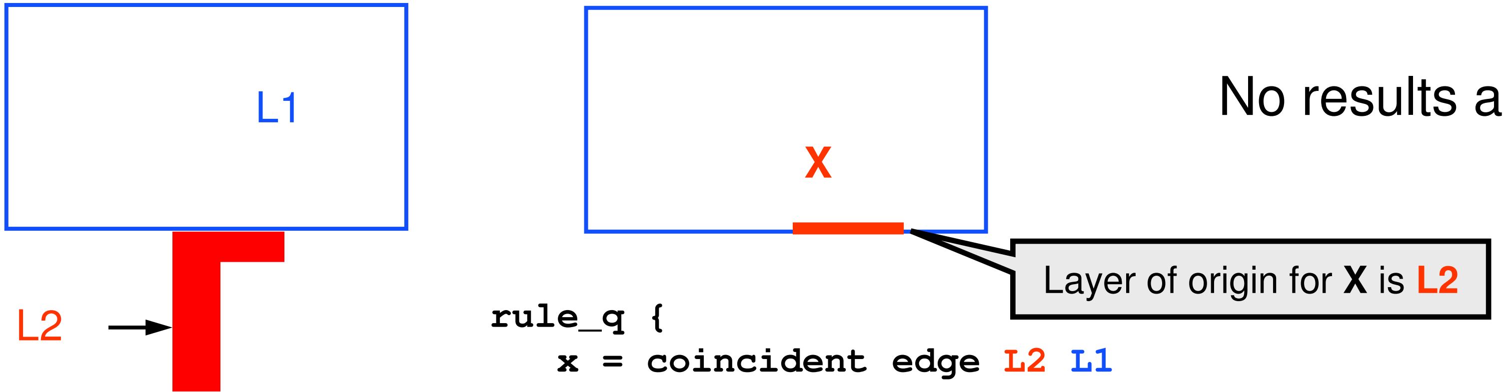
The layer-of-origin for layers **B** and **C** is layer **A** because it was the last layer that a layer constructor operation generated.

Layer of Origin Examples



Layer of origin for **X** is **L1**

```
rule_q {
   x = coincident edge L1 L2
   internal x L1 < 3 opposite
}
```



No results are generated.

Layer of origin for **X** is **L2**

```
rule_q {
   x = coincident edge L2 L1
   internal x L1 < 3 opposite
}
```

Layer Specification Statements

The next two statements control how Calibre defines layers:

- **`LAYER`**
- **`LAYER MAP`**

Layer

Purpose: Defines the name of an original layer or layer set

Syntax: `LAYER name original_layer [original_layer…]`

Parameters:

name — name of an original layer or layer set

original_layer — layer number of an original layer or the name of a layer (set) defined by another **LAYER** statement

Default: none

- **Use a LAYER statement to declare each drawn layer or layer set you reference by name in your rule file.**
- **You may reference original layers by name or number.**
- **You must reference layer sets by name.**

Layer (Cont.)

- **You may not redefine an original layer.**
- **You may not assign the same name to different layer numbers in separate layer statements.**
- **You may assign different names to the same layer number.**

Example:

rule file

```
// DEFINE A SIMPLE ORIGINAL LAYER
LAYER     METAL1          10
// DEFINE A LAYER SET CONSISTING OF TWO SIMPLE LAYERS
LAYER     METAL2          20  30
// DEFINE A LAYER SET USING PRE-DEFINED LAYERS
LAYER     ALL_METAL       METAL1  METAL2

LAYER     METAL1   50
```

This will cause an error!

Layer Map

Purpose: Maps OASIS databases and GDSII layer numbers, DATATYPEs, and TEXTTYPEs to Calibre layer numbers.

Syntax:

```
LAYER MAP source_layer {DATATYPE |TEXTTYPE}
        source_type target_layer
```

Parameters:

source_layer — number or constraint defining the source layer(s) to map

DATATYPE — instructs the tool to create a DATATYPE map

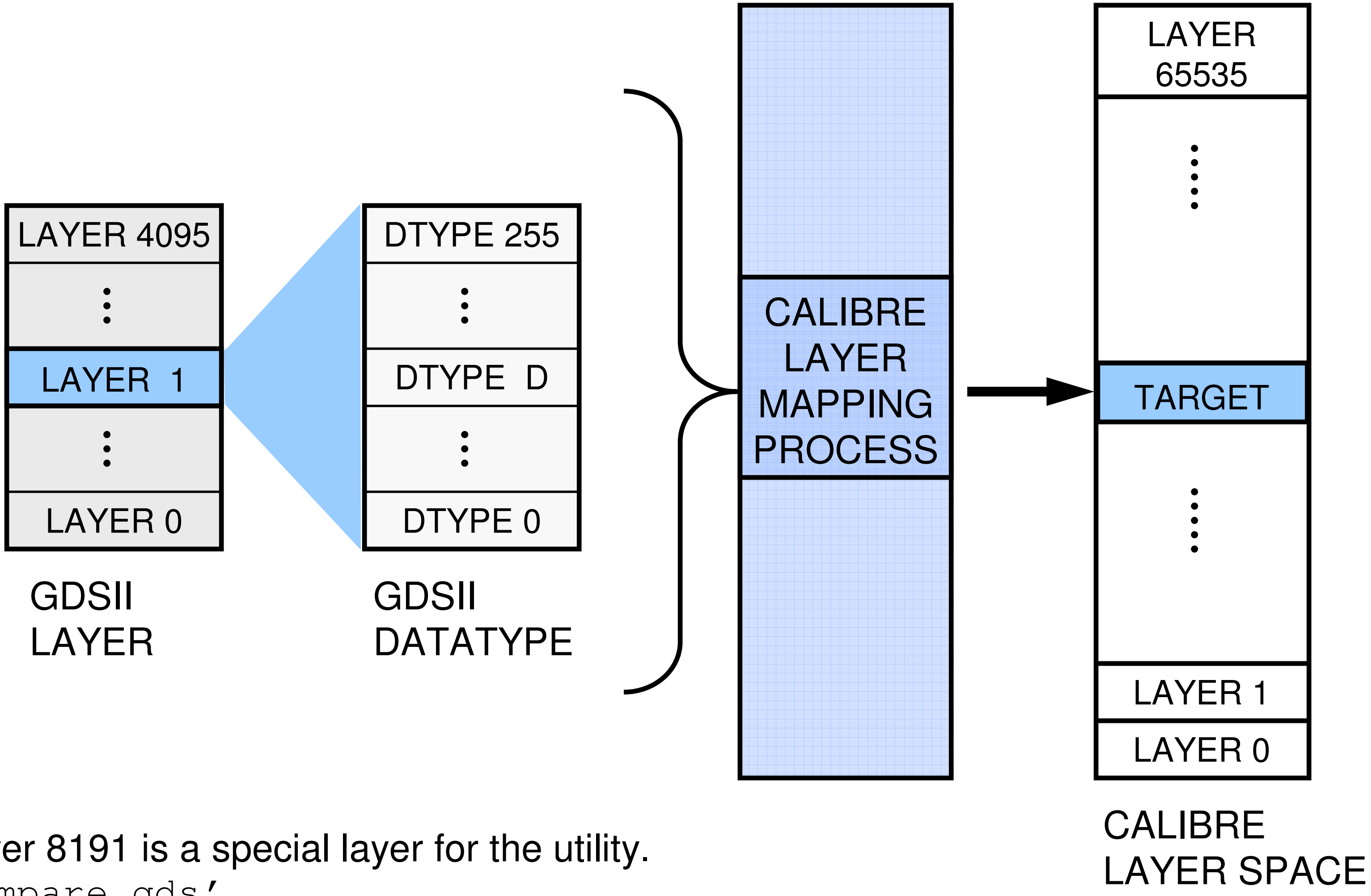
TEXTTYPE — instructs the tool to create a TEXTTYPE map

source_type — number or constraint for the source type

target_layer — specifies the number for the Calibre target layer

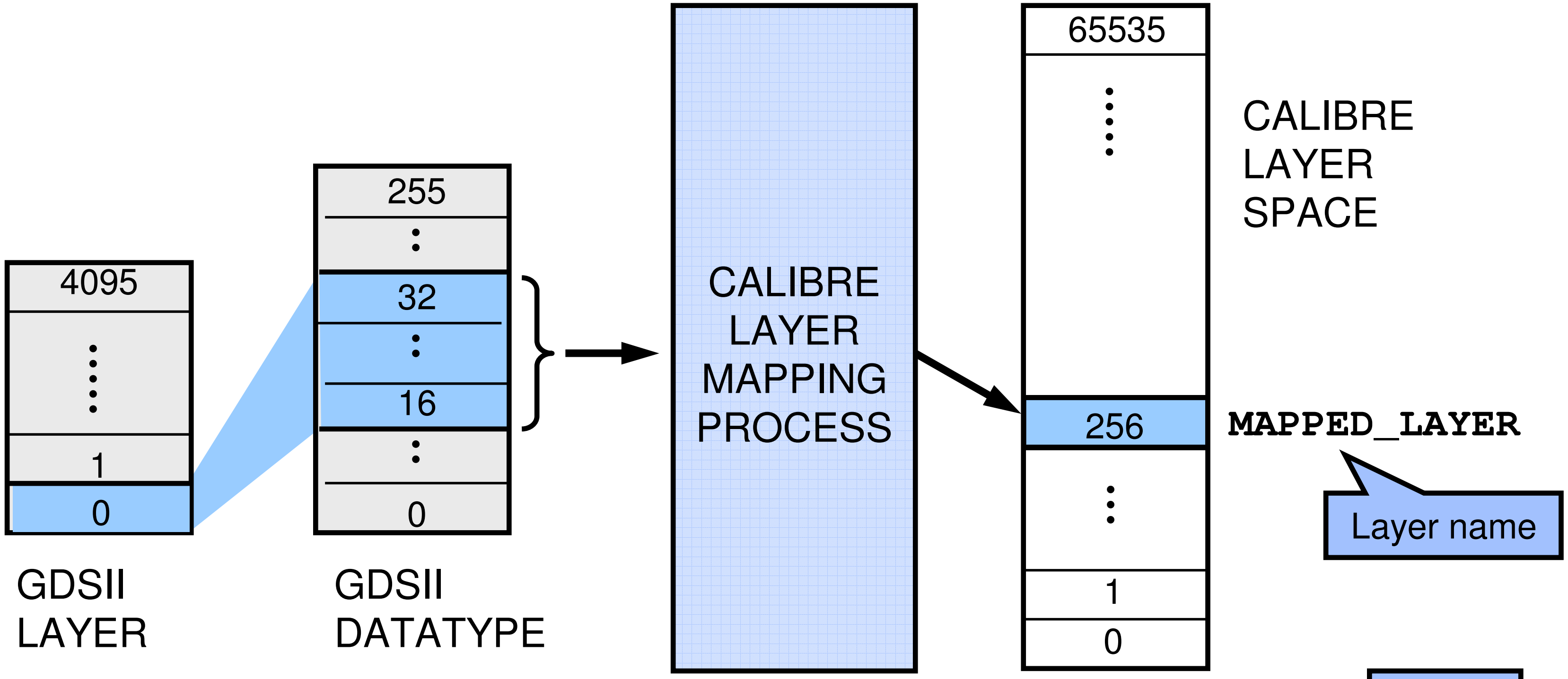
By default, Calibre ignores GDSII datatypes and texttypes.

GDSII to Calibre Layer Map Diagram



*Layer 8191 is a special layer for the utility.
'`compare_gds`'

Example of Using the Layer Map Statement



rule file

```
// MAP DATATYPES 16-32 ON LAYER 0 INTO LAYER 256
LAYER MAP  0  DATATYPE  >=16 <=32  256

// DEFINE THE NAME OF THE TARGET LAYER
LAYER  MAPPED_LAYER  256
```

DRC Output Control Statements

The following statements control the output of Calibre nmDRC:

- `DRC RESULTS DATABASE`
- `DRC MAXIMUM VERTEX`
- `DRC CHECK MAP`
- `DRC MAP TEXT`
- `DRC MAP TEXT DEPTH`
- `DRC SUMMARY REPORT`
- `DRC MAXIMUM RESULTS`

STATEMENT OVERVIEW

DRC Results Database

CLASS: SPECIFICATION

Purpose: Specifies the filename and type of the results database for Calibre nmDRC

Syntax:

```
DRC RESULTS DATABASE filename [type]
        [PSEUDO|USER MERGED|USER]
```

Parameters:

filename — pathname of the DRC results database

type — keyword from the set: **ASCII**, **BINARY**, **GDSII**, **OASIS**

PSEUDO—Instructs Calibre to include pseudocells generated during hierarchical processes.

USER MERGED—Suppresses the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell and then merged.

USER—Suppresses the output of pseudocells in a hierarchical results database. Geometry in pseudocells is transformed up the hierarchy to the first user cell but is not merged.

Default: **ASCII**

Example:

```
DRC RESULTS DATABASE “./drc.out” ASCII
```

STATEMENT
OVERVIEW

DRC Maximum Vertex

CLASS:
SPECIFICATION

Purpose:	Specifies the maximum vertex count of any polygon DRC result to be written to the DRC Results Database
Syntax:	`DRC MAXIMUM VERTEX {number | ALL}`
Parameters:	***number*** — must be an integer greater than or equal to 4 **ALL** — specifies that there is no maximum vertex count
Default:	**4096**
Example:	`DRC MAXIMUM VERTEX 1024`

Output polygons containing more vertices than the value specified will be broken up into multiple polygons.

DRC Check Map

Purpose: Controls the database output structure for DRC RuleChecks

Syntax:

```
DRC CHECK MAP rule_check
{{GDSII|OASIS}[layer[datatype]]}|ASCII|
BINARY[filename][MAXIMUM RESULTS{max|ALL}]
[MAXIMUM VERTICES{maxvertex|ALL}]
[TEXTTAG name][PSEUDO|USER|USER MERGED]
{[{AREF cell_name width length
[minimum_element_count]
[SUBSTITUTE x1 y1 … xn yn ]}...]|
[AUTOREF]}
```

Parameters:

rule_check — specifies a RuleCheck name or group name

GDSII, OASIS, ASCII, BINARY — specifies the format of the DRC results database; GDSII is the default

layer — optional layer number that receives ***rule_check*** results; default is 0

Practical use: Can generate multiple DRC results databases with different data formats and using different RuleChecks from a single Calibre nmDRC run.

DRC Check Map (Cont.)

- **Multiple DRC Check Map statements are permitted, allowing multiple databases.**
- **If results database is GDSII, Calibre issues a warning for each RuleCheck missing a DRC CHECK MAP statement.**
- **Example:**

```
LAYER gate_layer 100
gate = poly and diff
gates {
  copy gate
}
// Output all gates to layer 100 in file gates.gds
// All DRC results are output
DRC CHECK MAP gates GDSII 100 `./outfile.gds'
        MAXIMUM RESULTS ALL
```

STATEMENT
OVERVIEW

DRC Map Text

CLASS:
SPECIFICATION

Purpose: Specifies whether to transfer all text objects in the input database to the DRC results database

Syntax: `DRC MAP TEXT {NO|YES}`

Parameters:

- **`NO`** – Calibre outputs DRC results database with no text
- **`YES`** – Calibre transfers text to the DRC results database

Default: `NO`

Example: `DRC MAP TEXT YES`

- ♦ **Applies only to hierarchical DRC**
- ♦ **Applies only if the input and output are GDSII or OASIS**
- ♦ **Obeys `LAYER MAP` specification statements**

STATEMENT
OVERVIEW

DRC Map Text Depth

CLASS:
SPECIFICATION

Purpose: Controls the depth for reading text objects for the **`DRC MAP TEXT YES`** specification statement

Syntax: **`DRC MAP TEXT DEPTH {ALL|PRIMARY|`*`depth`*`}`**

Parameters:

`ALL` — instructs DRC-H to read text objects from all levels of layout hierarchy

`PRIMARY` — instructs DRC-H to read text objects from the top-level cell only (Same as ***`depth`*** = 0)

`depth` — instructs DRC-H to read text objects down to the hierarchical level of depth; the top level is zero

Default: **`ALL`**

Example:

```
DRC MAP TEXT DEPTH 1
      //read text objects from the top cell
      //and one level below
```

DRC Summary Report

Purpose: Specifies the DRC summary report file name and how it is written

Syntax:

```
DRC SUMMARY REPORT filename
      [REPLACE | APPEND][HIER]
```

Parameters:

- ***`filename`*** — the report file
- **`REPLACE`** — overwrite previous summary report file
- **`APPEND`** — appends to an existing summary report file
- **`HIER`** — lists non-empty RuleCheck statistics by layout database cell

Default: **`REPLACE`**

Example:

```
DRC SUMMARY REPORT "../drc_report" HIER
```

DRC Maximum Results

Purpose: Specifies the maximum number of results per RuleCheck written to the DRC results database

Syntax: `DRC MAXIMUM RESULTS {`***`maxresults`***` | ALL}`

Parameters:

`maxresults` — non-negative integer specifying the maximum number of DRC results

`ALL` — specifies unlimited count of DRC results

Default: 1000

Example: `DRC MAXIMUM RESULTS 50`

- **When Calibre reaches the maximum result count for a RuleCheck, it issues a warning and suspends output.**
- **Choose `ALL` when you are doing database manipulation.**
- **Specify this statement only once.**

DRC RuleCheck Control Specification Statements

The next three statements control DRC RuleCheck execution:

- `GROUP`
- `DRC SELECT CHECK`
- `DRC UNSELECT CHECK`

Group

Purpose: Names a collection of RuleChecks

Syntax: `GROUP` ***`name rule_check`*** `[…`***`rule_check`***`]`

Parameters:

`name` — name of a RuleCheck group

`rule_check` — name of a RuleCheck or RuleCheck group

Default: none

Example:

```
// GROUP LEVEL1 AND LEVEL2 CHECKS FOR TAPEOUT
GROUP tapeout_checks “level?”
```

- **Use RuleCheck groups in `DRC SELECT CHECK` and `DRC UNSELECT CHECK` statements.**
- **You may specify this statement multiple times.**
- **You may define RuleCheck groups with unlimited hierarchy.**
- ***`rule_check`*** **parameters may include “?” wildcard.**

STATEMENT
OVERVIEW

DRC Select Check

CLASS:
SPECIFICATION

Purpose: Specifies which RuleChecks to execute

Syntax: `DRC SELECT CHECK` ***`rule_check`*** `[…`***`rule_check`***`]`

Parameters:

`rule_check` — name of a RuleCheck or RuleCheck group

Default: Execute all RuleChecks in the rule file

Example:

```
DRC SELECT CHECK met1_checks
//only run checks in the met1_checks group
```

Note: There is a similar statement not covered in this class, **`DRC SELECT CHECK BY LAYER`**.

STATEMENT OVERVIEW

DRC Unselect Check

CLASS: SPECIFICATION

Purpose: Specifies which RuleChecks not to execute

Syntax: `DRC UNSELECT CHECK` ***`rule_check`*** `[…`***`rule_check`***`]`

Parameters:

`rule_check` — A RuleCheck name or group

Default: Execute all RuleChecks in the rule file

Example:

```
DRC SELECT CHECK  poly_width all_met_spacing
DRC UNSELECT CHECK metal4_spacing
//metal4_spacing RuleChecks are not executed
```

◆ **Calibre uses the following selection procedure:**

- **Selects all RuleChecks, otherwise selects only those RuleChecks specified in `DRC SELECT CHECK` statements**
- **Unselects all RuleChecks specified in `DRC UNSELECT CHECK` statements**

DRC Area Specification Statements

The following statements allow you to specify a region in the layout where DRC RuleChecks are performed:

- **`LAYOUT WINDOW`**
- **`LAYOUT WINDOW CLIP`**

There are other commands available to limit the layout regions checked. Please see the *SVRF Manual* for more information.

STATEMENT
OVERVIEW

Layout Window

CLASS:
SPECIFICATION

Purpose: Specifies a polygon window that defines the inclusion of input polygons and text for DRC RuleChecks

Syntax: `LAYOUT WINDOW {x1 y1 x2 y2} [{x y}…]`

Parameters:

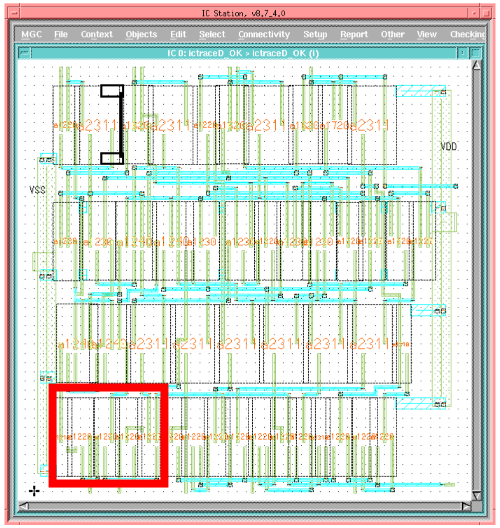
x1 y1 x2 y2 — a set of floating-point numbers specifying the coordinates of vertices of a polygon in user units (specifying ***x1 y1 x2 y2*** defines opposite corners of a rectangle)

Default: none

Example:

```
// DEFINE RECTANGLE AT POINTS (10,3)(30,25)
LAYOUT WINDOW 10 3 30 25
```

- **Calibre processes all database objects totally inside or intersecting the polygon window.**
- **You may specify this multiple times.**



Layout Window Clip

Purpose:	Specifies whether area-based filtering will be exclusive or inclusive
Syntax:	`LAYOUT WINDOW CLIP {NO |YES}`
Parameters:	**`NO`** — inclusive filtering **`YES`** — exclusive filtering
Default:	**`NO`**
Example:	`LAYOUT WINDOW CLIP YES`

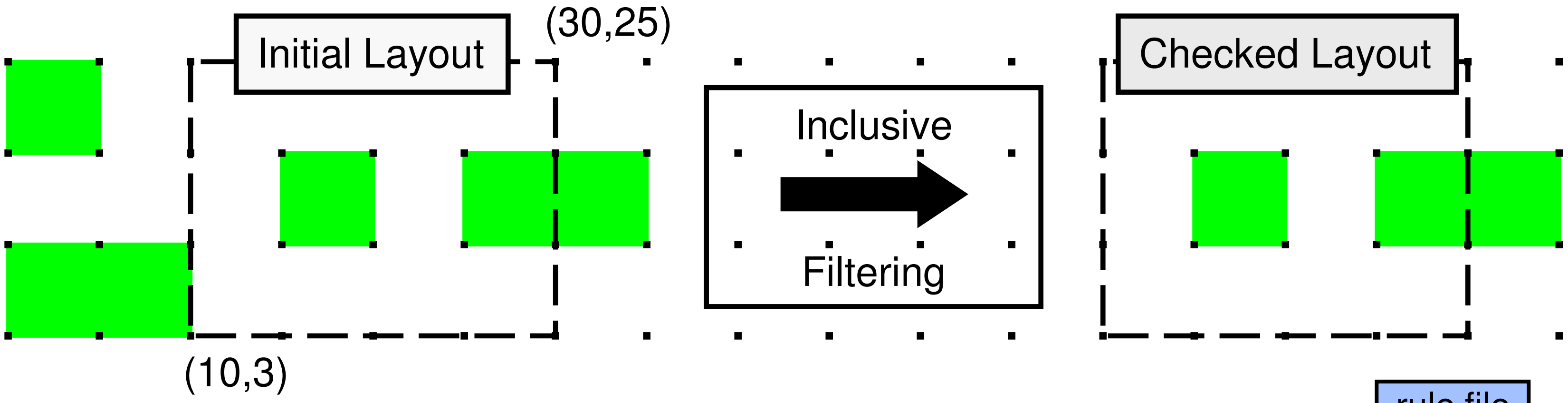
- **Inclusive filtering selects layout polygons that overlap the clipping region.**
- **Exclusive filtering performs a Boolean AND operation of the clipping region with the layout polygons.**

Note:
`LAYOUT WINDOW CLIP YES` is automatically invoked if you use the GUI.
This operation will modify the behavior of **`LAYOUT WINDOW`** and **`LAYOUT WINDEL`**.

Example of Using Layout Window Clip

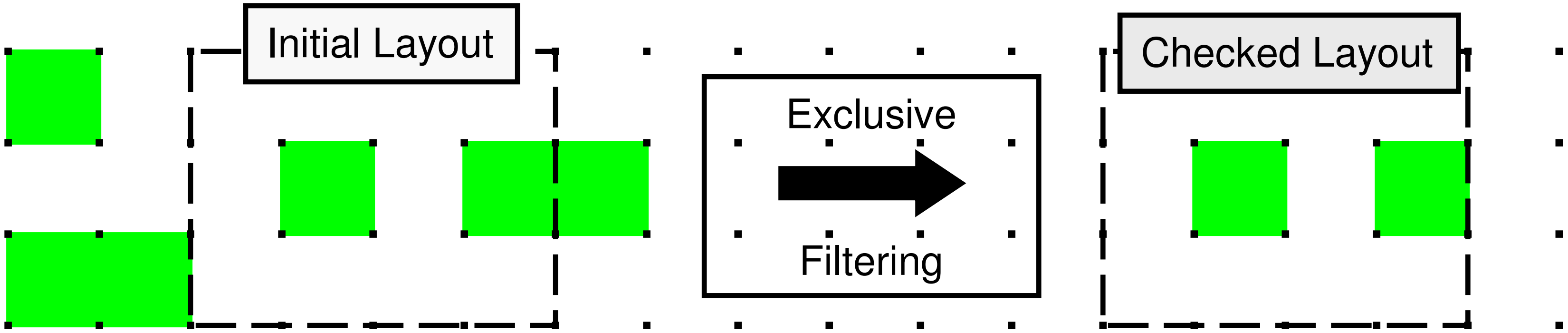
rule file

```
LAYOUT WINDOW 10 3 30 25     LAYOUT WINDOW CLIP NO
```



rule file

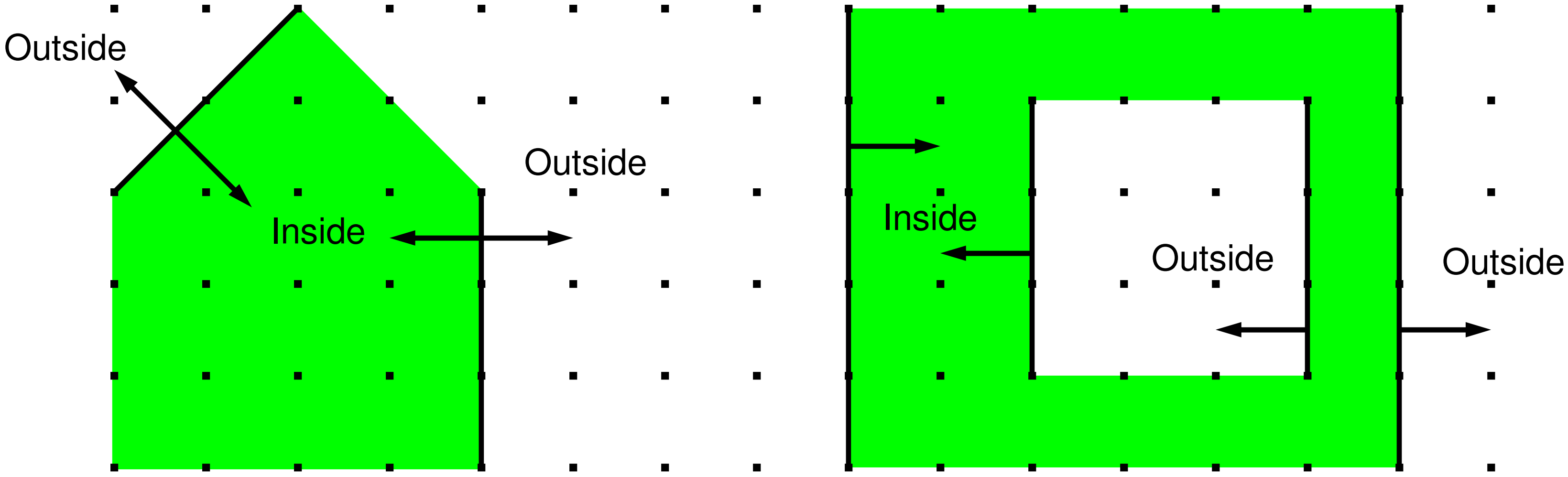
```
LAYOUT WINDOW 10 3 30 25    LAYOUT WINDOW CLIP YES
```



Geometric Data Types

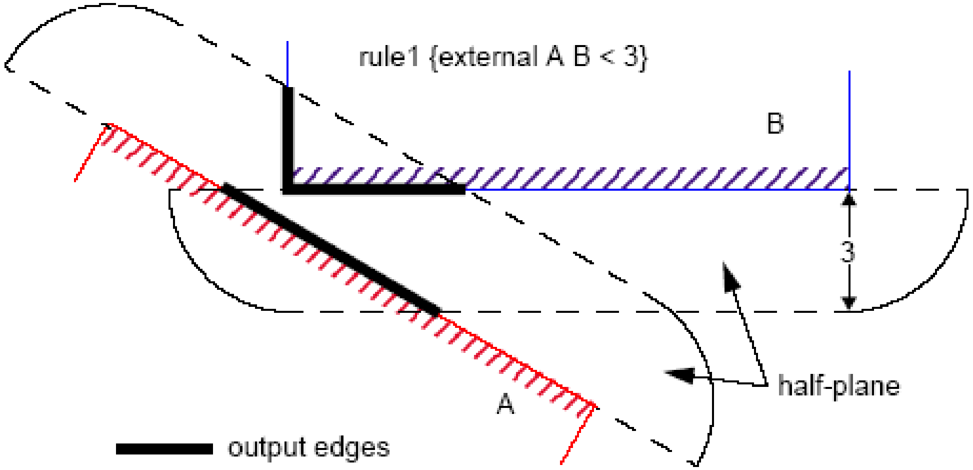
Calibre nmDRC processes two types of geometric data:

- **Polygons**
- **Edges (of polygons)**
  - **Always have a reference back to their source polygon**
  - **May have a reference to an electrical net**
  - **Always have an interior facing side and an exterior facing side**



Calibre Edge-Based DRC System

- When considering an edge pair in a dimensional check, Calibre constructs a region for each edge consisting of the half-plane of all points that fall within the specified distance of the edge.
- Calibre outputs any portion of one edge that intersects the region associated with the other edge.
- For example:

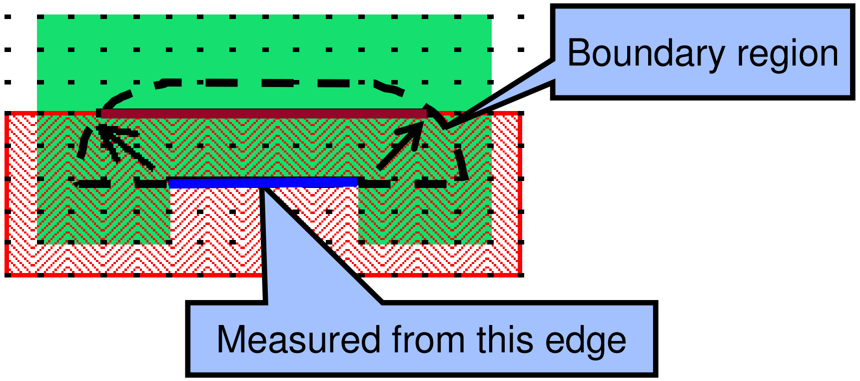


- The construction of the half-plane is controlled by the dimensional check metric (Euclidian, Square, or Opposite).

Edge Checking Metrics Options — Euclidean

Euclidean metric:

- **Forms a region with quarter-circle boundaries that extend past the endpoints of the selected edges**



Boundary region

Measured from this edge

```
INT L1 L2 < 3
```

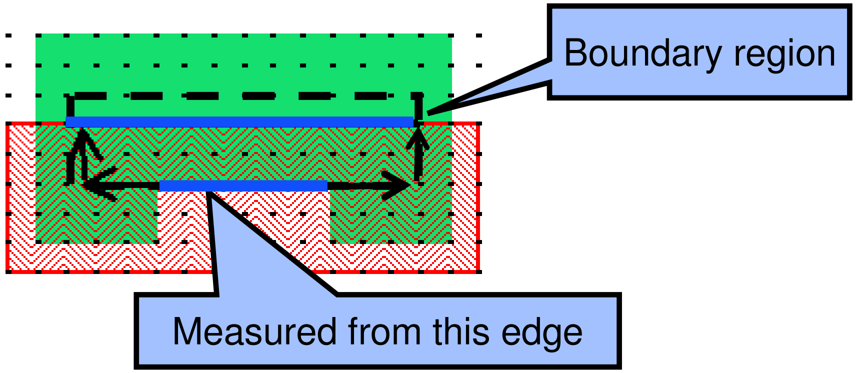
🟩 **L1**

▨ **L2**

Edge Checking Metrics Options — Square

Square metric:

Forms a region with right-angle boundaries that extend past the endpoints of the selected edges



Boundary region

Measured from this edge

```
INT L1 L2 < 3 SQUARE
```

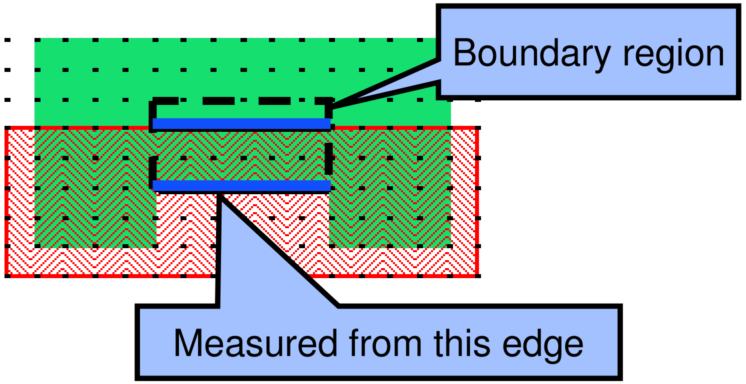
🟩 **L1**

▨ **L2**

Edge Checking Metrics Options — Opposite

Opposite metric:

- **Forms a region with right-angle boundaries that do not extend past the endpoints of the selected edges**



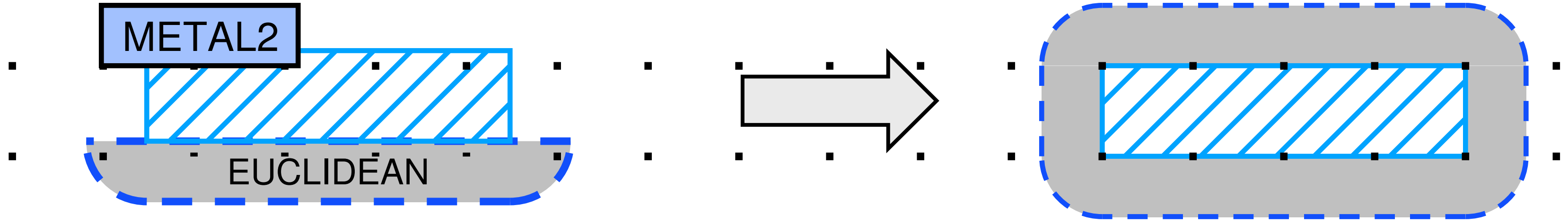
```
INT L1 L2 < 3 OPPOSITE
```

🟩 **L1**

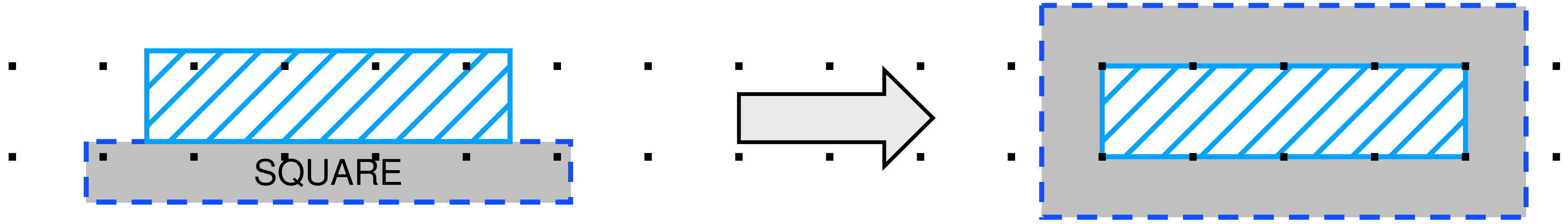
▨ **L2**

Measurement Regions for External Operations

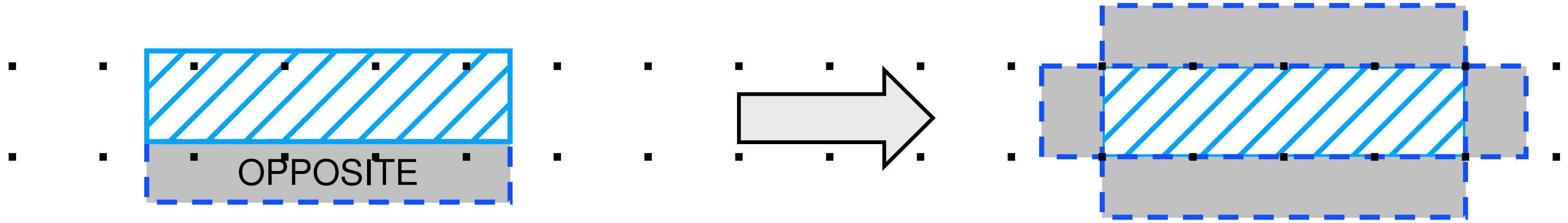
`EXTERNAL metal2 < 0.75`



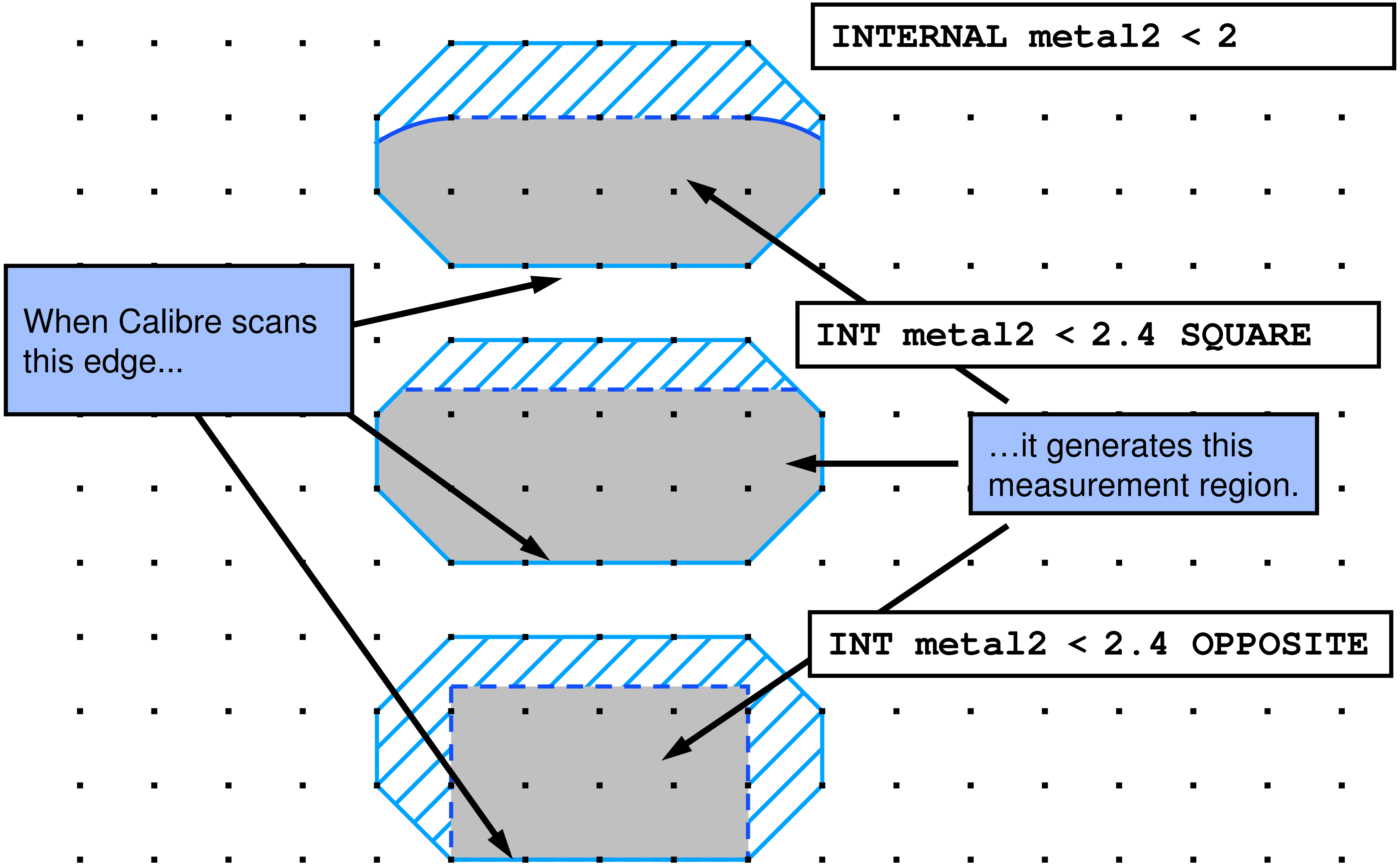
`EXT metal2 < 0.75 SQUARE`



`EXT metal2 < 0.75 OPPOSITE`



Measurement Regions for Internal Operations



`INTERNAL metal2 < 2`

`INT metal2 < 2.4 SQUARE`

`INT metal2 < 2.4 OPPOSITE`

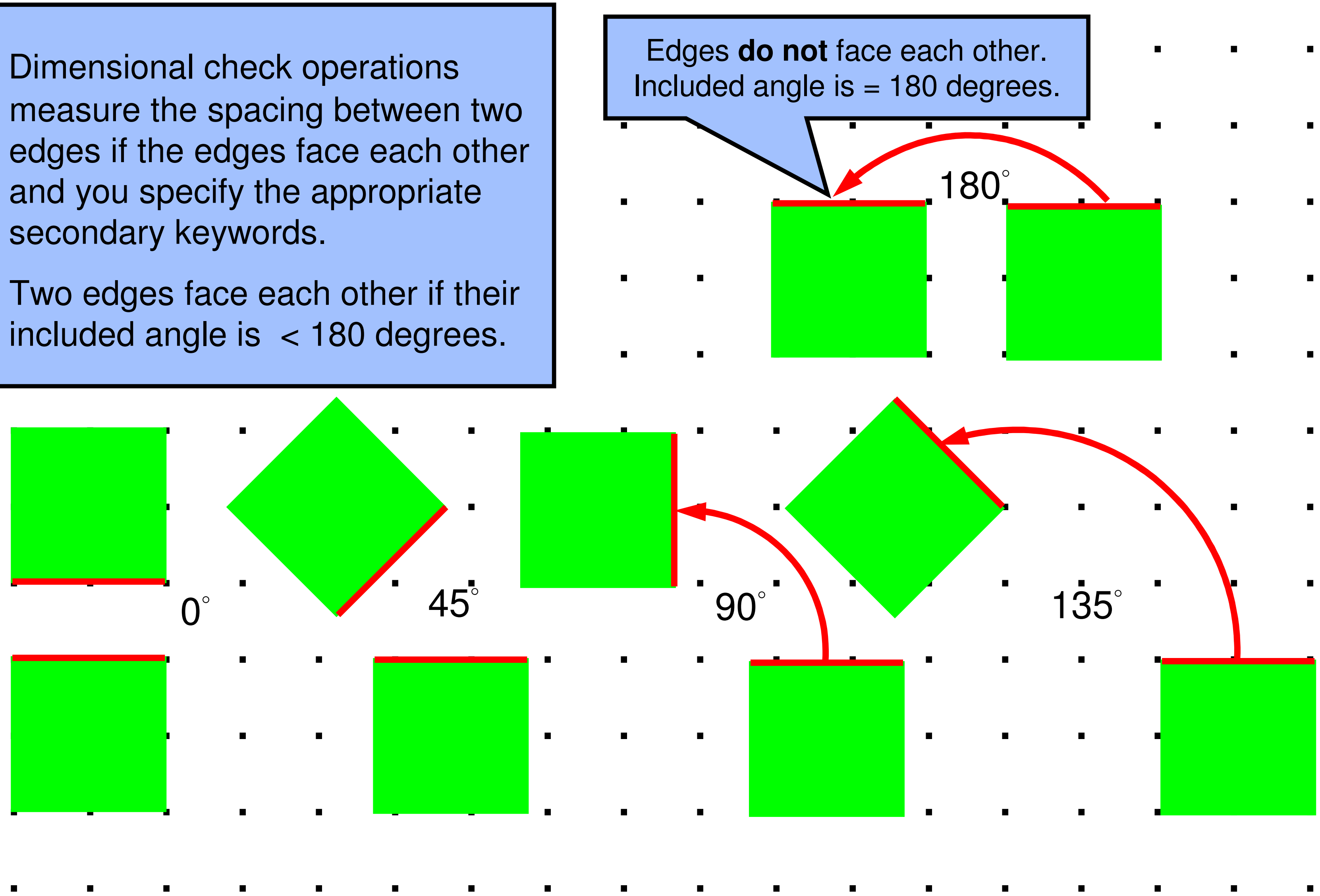
When Calibre scans this edge...

…it generates this measurement region.

Edge Measurement Criteria

Dimensional check operations measure the spacing between two edges if the edges face each other and you specify the appropriate secondary keywords.

Two edges face each other if their included angle is < 180 degrees.



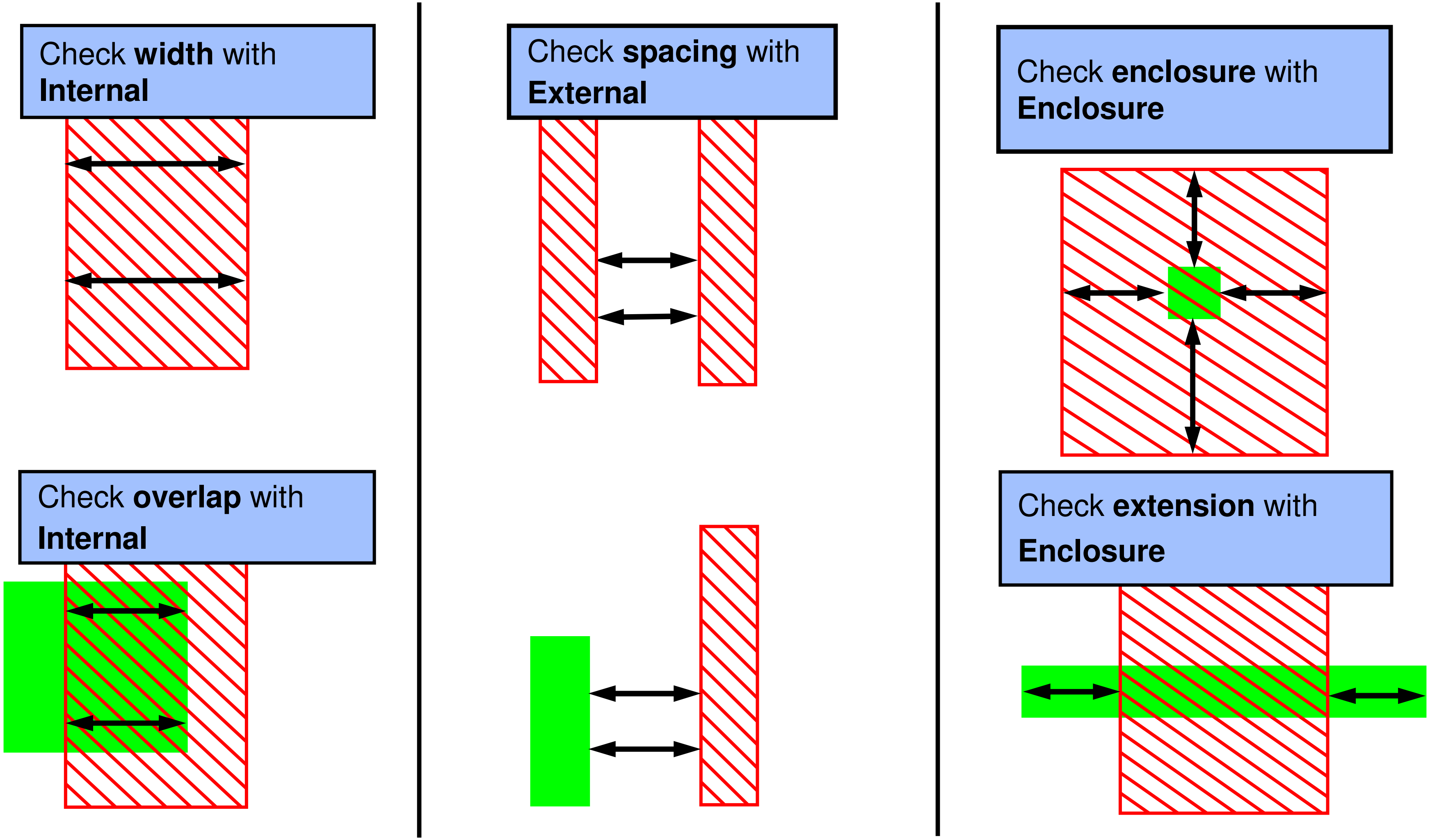
Edges **do not** face each other. Included angle is = 180 degrees.

Calibre Rule Writing

Module 3

Dimensional Rule Checks

DRC RuleCheck Operations



Internal

Check **width** with **Internal**

Check **overlap** with **Internal**

External

Check **spacing** with **External**

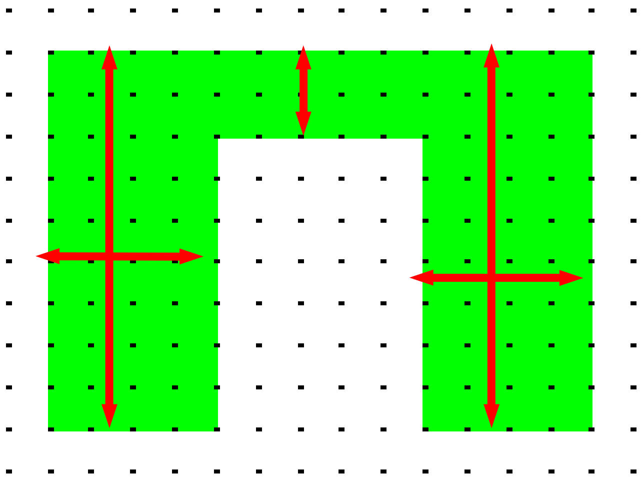
Enclosure

Check **enclosure** with **Enclosure**

Check **extension** with **Enclosure**

Width Checks

- Width checks are internal checks on polygons on a single input layer.
- Width checks are measured between interior-facing edge pairs on the same polygon.
- Intersecting edge pairs are not measured by default.
- Measured edge pairs satisfying the given constraints are output to the DRC results database.



STATEMENT OVERVIEW

Internal (Width Check) Statement Overview

CLASS: OPERATION

Purpose: Measures distances between interior-facing edge pairs of polygons on a single layer

Syntax:

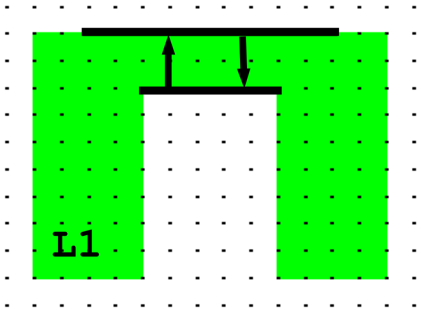
```
INTERNAL layer constraint
       [secondary_keywords]
```

Parameters:

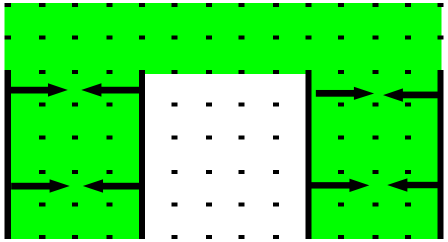
- ***layer*** — original layer,derived polygon or edge layer
- ***constraint*** — non-negative real value or range
- ***secondary_keywords*** — will be covered later

Defaults : `PARALLEL ALSO, ACUTE ALSO, NOT PERPENDICULAR` and `NOT OBTUSE`

Examples:



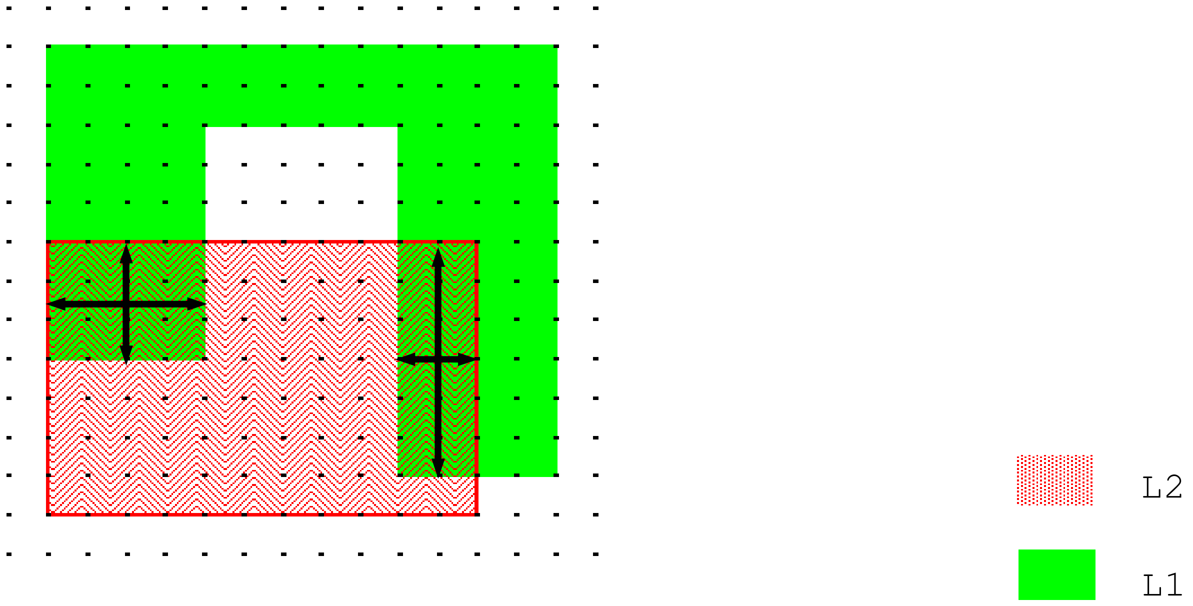
`INT L1 < 3`



`INT L1 == 4`

Overlap Checks

- **Checks between interior-facing edge pairs of polygons on two different layers.**
- **Intersecting edge pairs are not measured by default.**
- **Measured edge pairs satisfying the given constraints are output to the DRC results database.**



Internal (Overlap Check) Statement Overview

Purpose: Measures distances between interior-facing edge pairs of overlapping polygons on two layers

Syntax:

```
INTERNAL layer1 layer2 constraint
          [secondary_keywords]
```

Parameters:

layer1, layer2 — original layers, derived polygon or edge layers

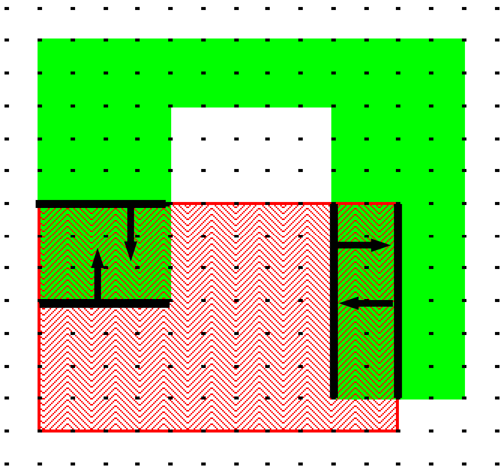
constraint — non-negative real value or range

secondary_keywords — will be covered later

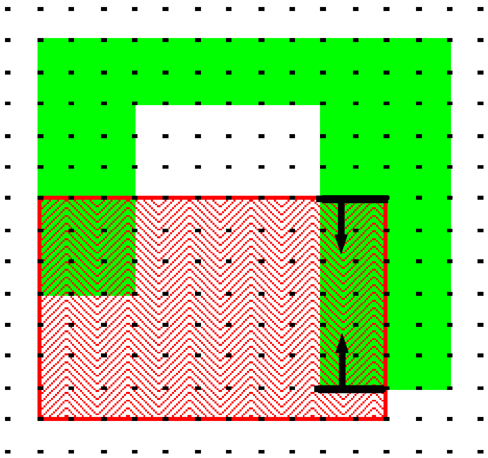
Defaults: **PARALLEL ALSO, ACUTE ALSO,**
NOT PERPENDICULAR and **NOT OBTUSE**

- **Layers are not order-dependent for this operation.**

Overlap Checks — Examples



`INT L1 L2 <=3`



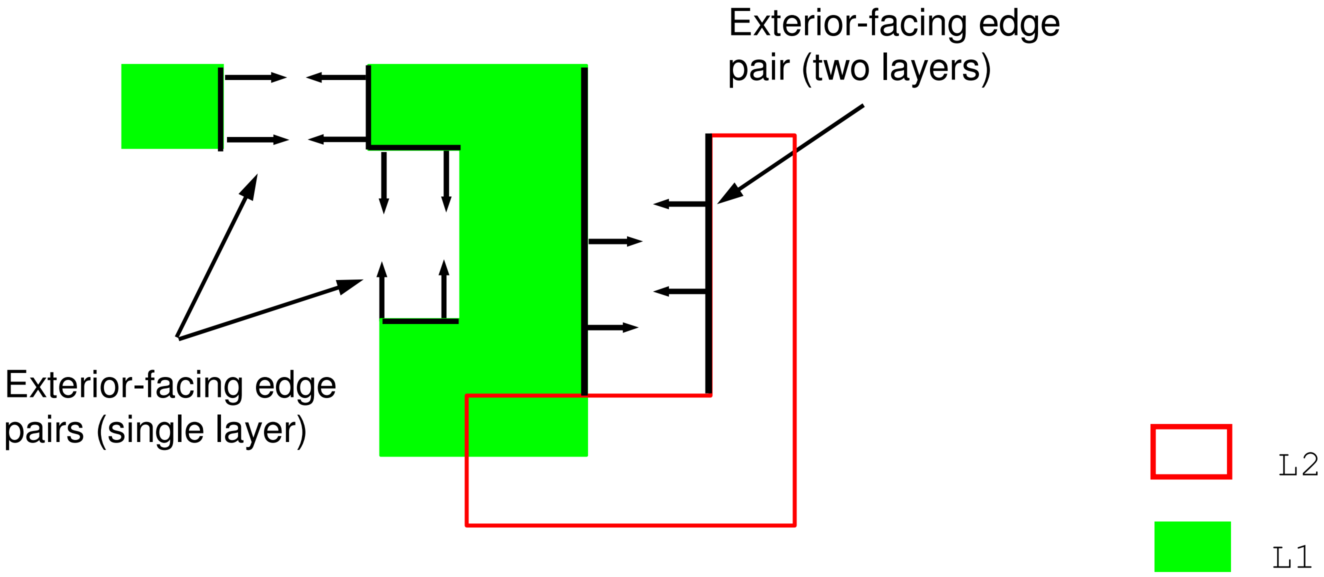
`INT L1 L2 >4.5 <=6`

▨ L2

🟩 L1

External Checks

- External checks on polygons are called spacing checks.
- Apply only to exterior-facing edge pairs.
- Intersecting edge pairs are not measured by default.
- Measured edge pairs satisfying the given constraints are output to the DRC results database.



External Statement Overview (Spacing Check)

Purpose: Measures the distance between exterior edge pairs of polygons on one or two layers

Syntax: **`EXTERNAL`** ***`layer1`*** **`[`*****`layer2`*****`]`** ***`constraint`***
`[`***`secondary_keywords`*****`]`**

Parameters:

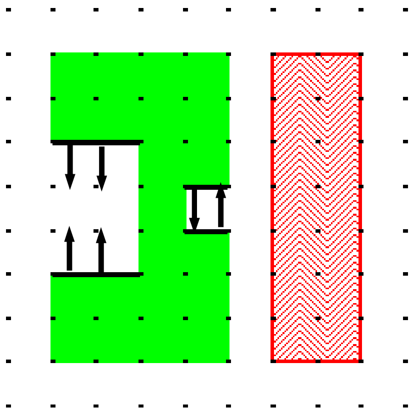
`layer1`, ***`layer2`*** — original layers or derived polygon or edge layers

`constraint` — non-negative real value or range

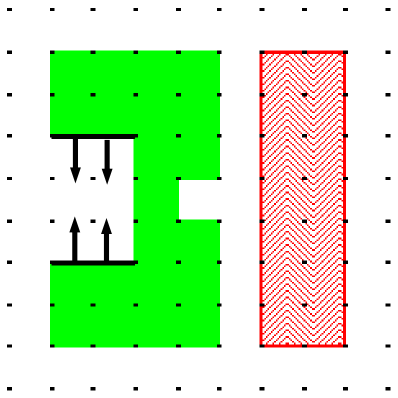
`secondary_keywords` — covered later in module

Default: **`PARALLEL ALSO, ACUTE ALSO,`**
`NOT PERPENDICULAR,` and **`NOT OBTUSE`**

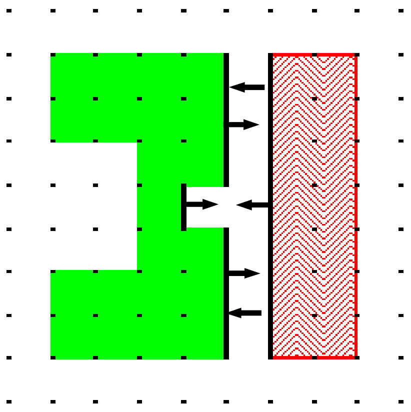
External (Spacing) Checks — Examples



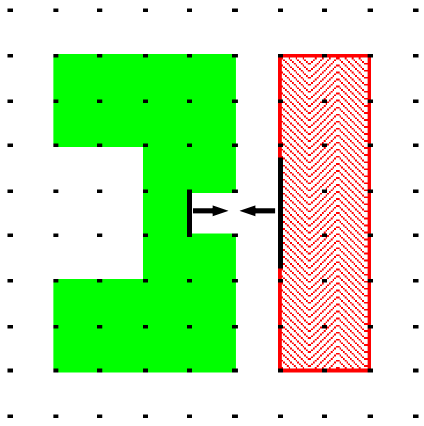
`EXT L1 <= 3`



`EXT L1 > 1 <= 3`



`EXT L1 L2 <= 2`



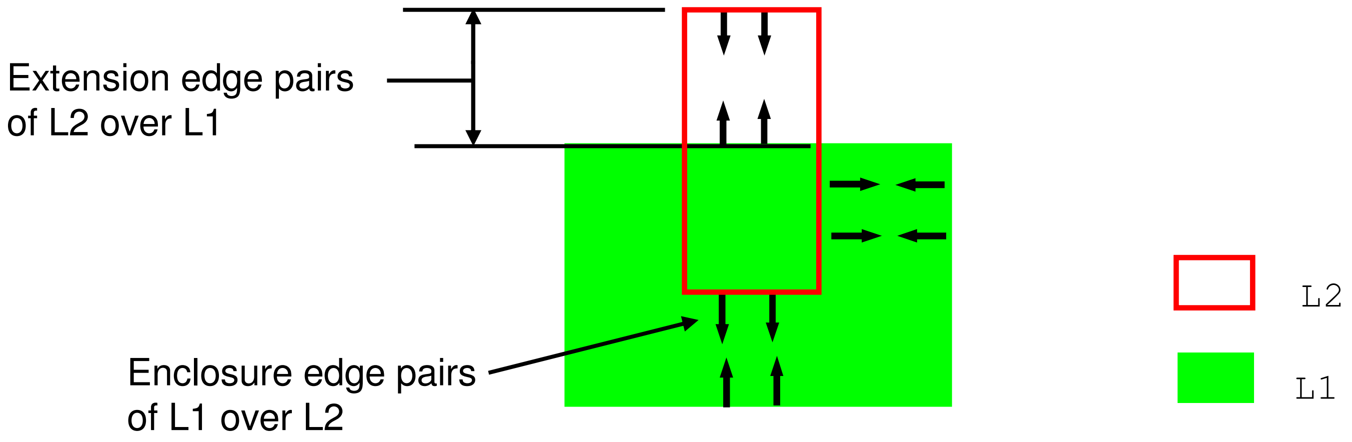
`EXT L1 L2 >1 <= 2`

🟥 L2

🟩 L1

Enclosure and Extension Checks

- Use Enclosure RuleChecks for both enclosure and extension checks.
- Enclosure checks from the the external edges of the first layer to the internal edges of the second layer.
- Edge pairs must face each other.
- Intersecting edge pairs are not measured by default.
- Measured edge pairs satisfying the given constraints are output to the DRC results database.



Extension edge pairs of L2 over L1

Enclosure edge pairs of L1 over L2

🟥 L2

🟩 L1

Enclosure Statement Overview (Enclosure and Extension Checks)

Purpose: Measures distances between the external edges of the first layer and the internal edges of the second layer

Syntax:

```
ENCLOSURE layer1 layer2 constraint
        [secondary_keywords]
```

Parameters:

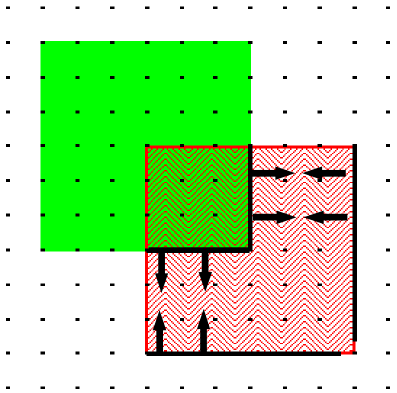
`layer1`, ***`layer2`*** — original layers or derived polygon or edge layers

`constraint` — non-negative real value or range

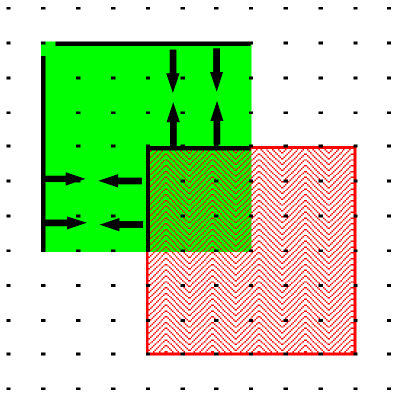
`secondary_keywords` — covered later in module

Defaults: **`PARALLEL ALSO, ACUTE ALSO,`**
`NOT PERPENDICULAR,` and **`NOT OBTUSE`**

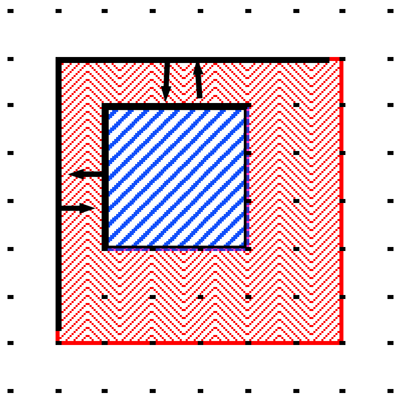
Enclosure Checks — Examples



ENC L1 L2 < 4



ENC L2 L1 < 4



ENC CONT L2 < 2

Notice the significance of switching the layer order.

▨ CONT

▦ L2

🟩 L1

Secondary Keywords

DRC dimensional RuleChecks use secondary keywords belonging to the following sets:

- Intersection
- Polygon Containment
- Connectivity Filters
- Angle Filters
- Orientation Filters
- Projection Filters
- Corner Filters
- Output

Intersecting Edge Pairs

- Intersecting edge pairs are not measured by default
- Can be overridden by secondary keywords

In both cases, edges A and B form an intersecting pair.

The distance between them is not measured by default.

🟥 L2

🟩 L1

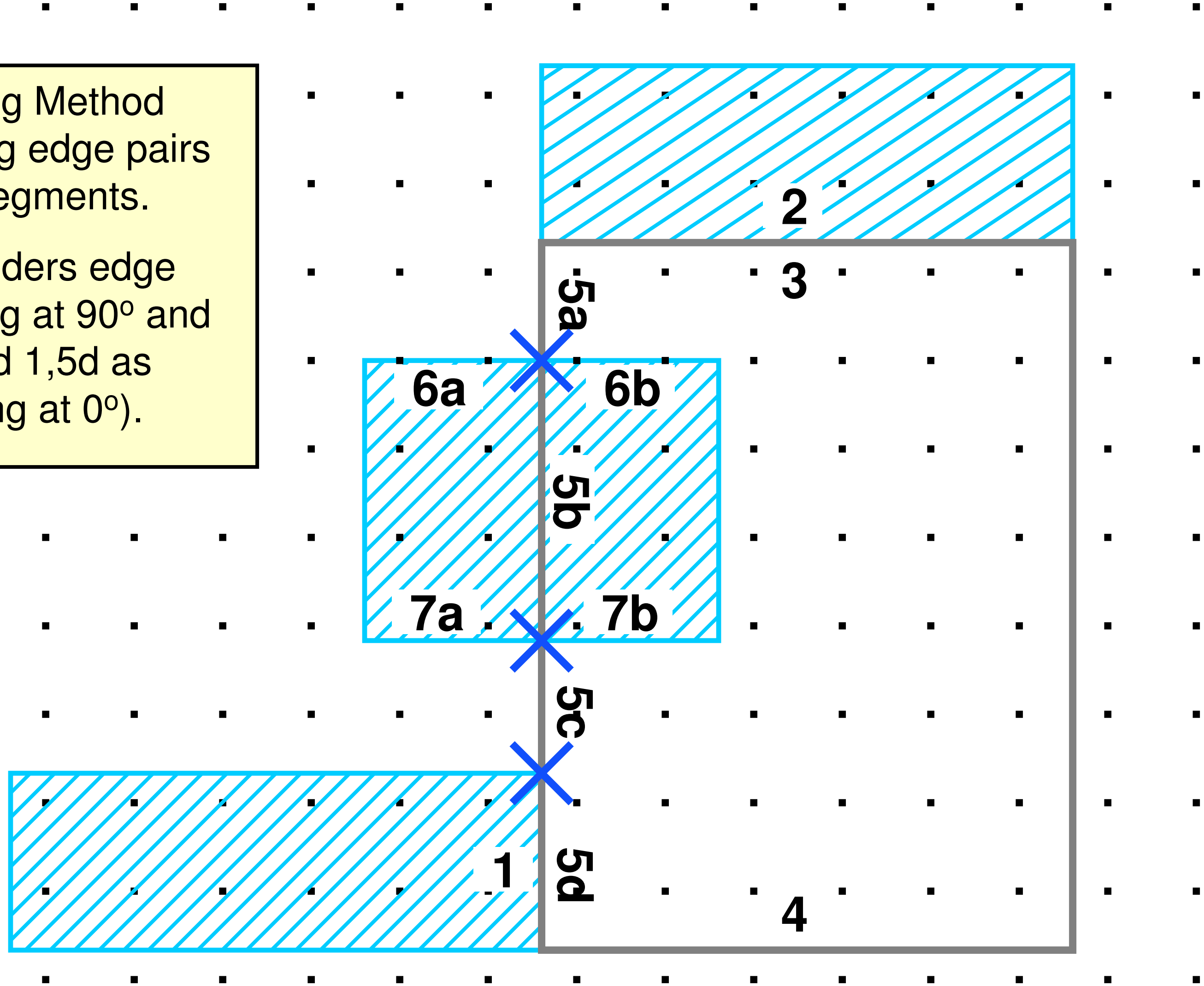
Edge Breaking

- Occurs during the evaluation of a two-layer dimensional check operation.
- Calibre DRC breaks edges from the input layer that cross polygon boundaries of the other input layer into edge segments:
  - Eliminates many false errors
  - Makes the output more precise

Two-Layer Edge Interactions

The Edge Breaking Method breaks intersecting edge pairs 5,6 and 5,7 into segments.

Calibre then considers edge pair 1,4 as abutting at 90º and edge pairs 2,3 and 1,5d as coincident (abutting at 0º).



Intersection

- The secondary keywords for intersection instruct dimensional RuleChecks to measure the separation between intersecting edge pairs.
- Secondary Keywords:
  - Additive filters:
    - `ABUT`
    - `SINGULAR`
    - `OVERLAP`
  - Restrictive filter:
    - `INTERSECTING ONLY`

NOTE:
For efficient rule writing you can combine **`ABUT`**, **`SINGULAR`**, and **`OVERLAP`** in one operation.

STATEMENT
OVERVIEW

Abut

CLASS:
SECONDARY
KEYWORD

Purpose: Measures separation between abutting edge pairs whose angular separation conforms to the specified parameter

Syntax:

`INT layer1 [layer2] constraint ABUT [parameter]`

`EXT layer1 [layer2] constraint ABUT [parameter]`

`ENC layer1 layer2 constraint ABUT [parameter]`

Parameter:

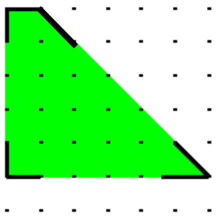
parameter — optional real value or range of values >=0 <180

Default: `ABUT >= 0 < 180`
if **ABUT** is specified without a constraint

- **Finds edges in addition to the default behavior for each operation.**
- **You will nearly always use `ABUT < 90`.**

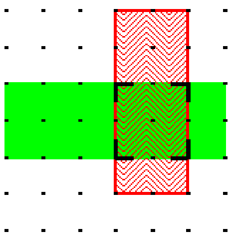
Abut — Examples

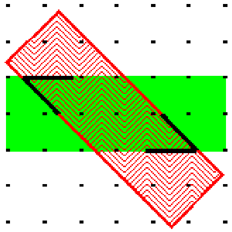
INT:



`L1 < 1 ABUT`

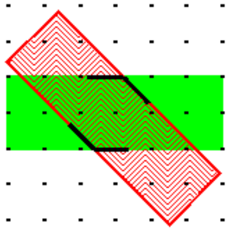
`L1 L2 < .5 ABUT == 90`



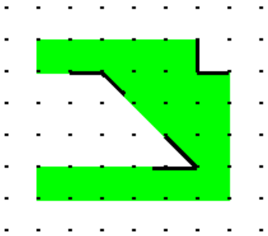


`L1 L2 < 1 ABUT < 90`

`L1 L2 < 1 ABUT > 90`

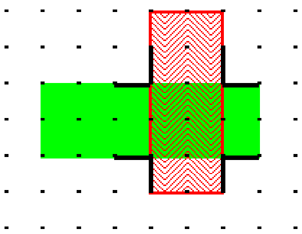


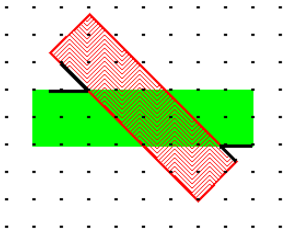
EXT:



`L1 < 1 ABUT`

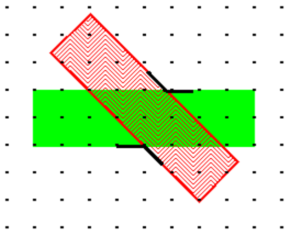
`L1 L2 < 1 ABUT == 90`



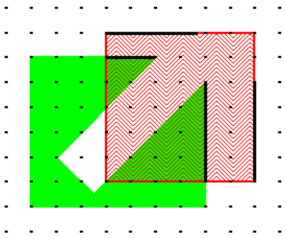


`L1 L2 < 1 ABUT < 90`

`L1 L2 < 1 ABUT > 90`

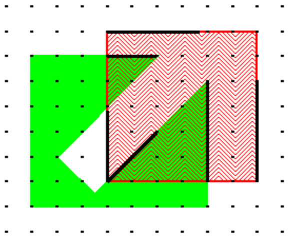


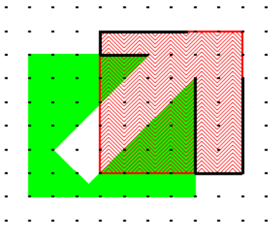
ENC:



`L1 L2 <= 2`

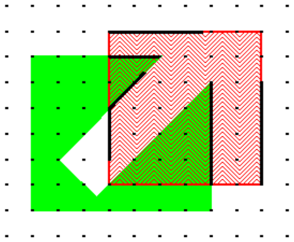
`L1 L2 <= 2 ABUT < 90`





`L1 L2 <= 2 ABUT == 90`

`L1 L2 <= 2 ABUT > 90`



🟩 L1 ▨ L2

STATEMENT OVERVIEW

Singular

CLASS: SECONDARY KEYWORD

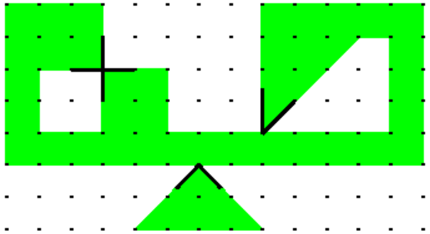
Purpose: Measures the separation between intersecting edge pairs at points of polygon singularity

Syntax:

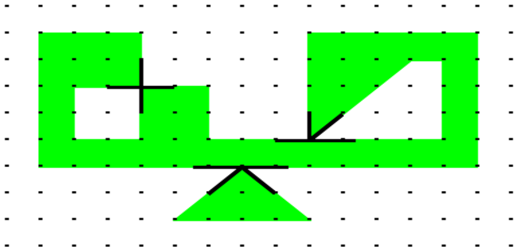
```
INT layer1 [layer2] constraint SINGULAR
EXT layer1 [layer2] constraint SINGULAR
ENC layer1 layer2 constraint SINGULAR
```

- **Finds results in addition to the default behavior.**
- **You will nearly always use `SINGULAR`.**

Example:



`INT L1 < 1 SINGULAR`

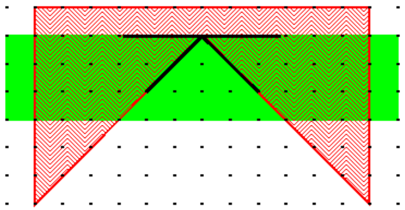


`EXT L1 < 1 SINGULAR`

🟩 L1

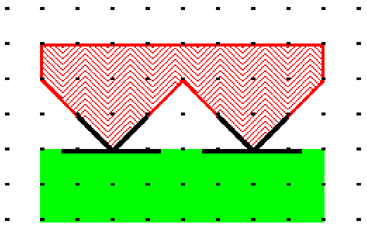
Singular — Examples

INT:



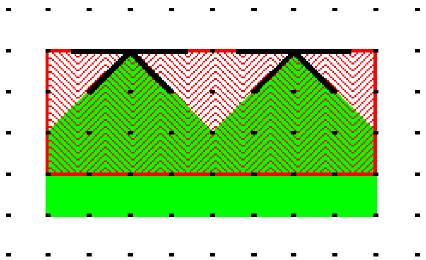
```
L1 L2 < 2 SINGULAR
```

EXT:



```
L1 L2 < 1 SINGULAR
```

ENC:



```
L1 L2 < 1 SINGULAR
```

🟩 L1 ▨ L2

STATEMENT
OVERVIEW

Overlap

CLASS:
SECONDARY
KEYWORD

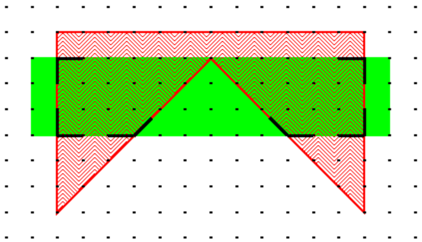
Purpose: Measures the separation between intersecting edge pairs at points where a polygon from one input layer crosses a polygon from the other

Syntax:

```
INT layer1 layer2 constraint OVERLAP
EXT layer1 layer2 constraint OVERLAP
ENC layer1 layer2 constraint OVERLAP
```

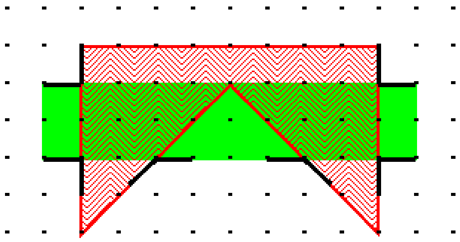
Examples:

INT:



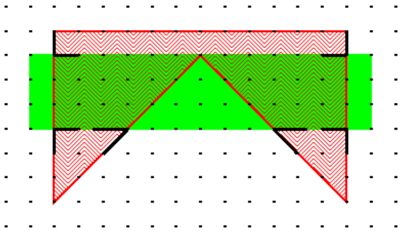
`L1 L2 <= 1 OVERLAP`

EXT:



`L1 L2 <= 1 OVERLAP`

ENC:



`L1 L2 < 1 OVERLAP`

🟩 L1 ▨ L2

STATEMENT
OVERVIEW

Intersecting Only

CLASS:
SECONDARY
KEYWORD

Purpose: Limits the number of edge pairs to be measured to intersecting edge pairs

Syntax:

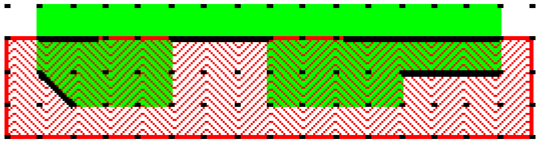
```
INT layer1 layer2 constraint ABUT|SINGULAR|OVERLAP
        INTERSECTING ONLY
EXT layer1 layer2 constraint ABUT|SINGULAR|OVERLAP
        INTERSECTING ONLY
ENC layer1 layer2 constraint ABUT|SINGULAR|OVERLAP
        INTERSECTING ONLY
```

`INTERSECTING ONLY` may only be used with the following secondary keywords:

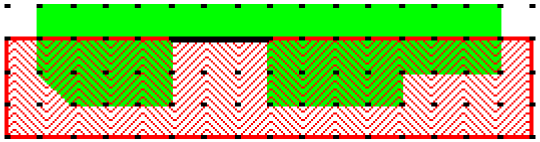
- `ABUT`
- `SINGULAR`
- `OVERLAP`

Intersecting Only — Examples

INT:

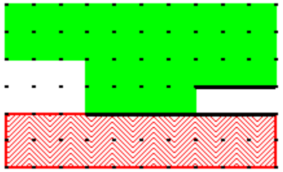


```
L1 L2 < 2 ABUT < 90
```

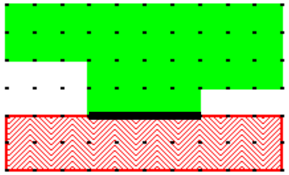


```
L1 L2 < 2 ABUT < 90
INTERSECTING ONLY
```

EXT:

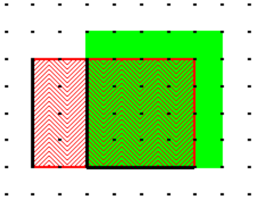


```
L1 L2 < 2 ABUT < 90
```

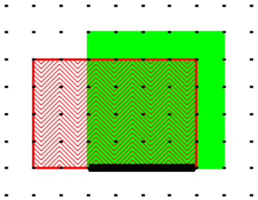


```
L1 L2 < 2 ABUT < 90
INTERSECTING ONLY
```

ENC:



```
L1 L2 < 3 ABUT < 90
```



```
L1 L2 < 3 ABUT < 90
INTERSECTING ONLY
```

🟩 L1

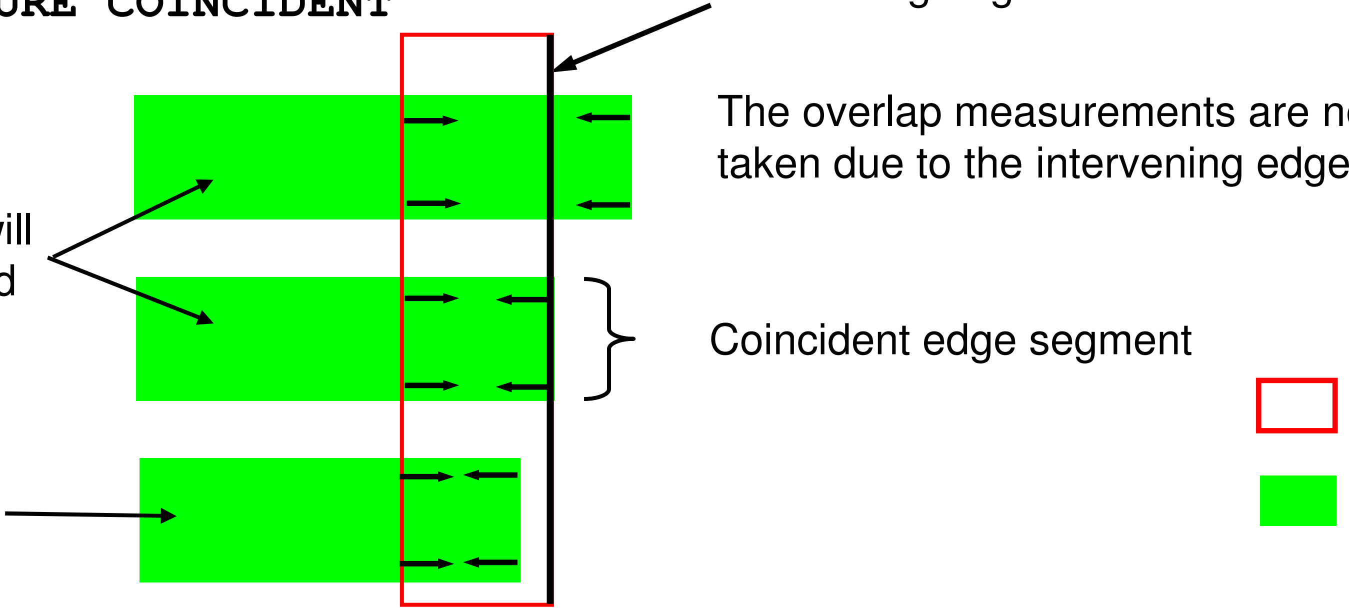
▨ L2

Polygon Containment

- **The secondary keywords for polygon containment instruct the two-layer dimensional check operations to ignore or relax the polygon containment criteria when measuring the separation between edge pairs.**
- **Secondary Keywords:**
  - **`MEASURE ALL`**
  - **`MEASURE COINCIDENT`**

Containment Criteria

- **Internal checks do not apply to interior-facing edges if one of the related polygons has an intervening edge coincident with or between the two measured edges.**
- **Use the following secondary keywords to override containment:**
  - `MEASURE ALL`
  - `MEASURE COINCIDENT`



Intervening edge

The overlap measurements are not taken due to the intervening edge

These cases will **not** be checked

Coincident edge segment

This case will be checked

STATEMENT
OVERVIEW

Measure All

CLASS:
SECONDARY
KEYWORD

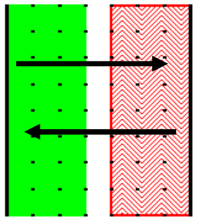
Purpose: Ignores polygon containment criteria when measuring the separation between edge pairs

Syntax:

```
INT layer1 layer2 constraint MEASURE ALL
EXT layer1 layer2 constraint MEASURE ALL
ENC layer1 layer2 constraint MEASURE ALL
```

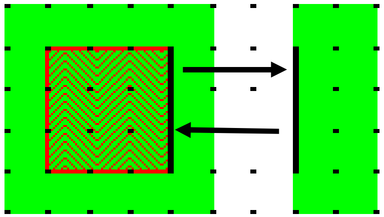
Examples:

INT:



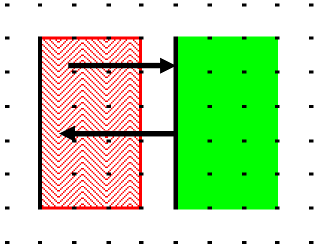
```
L1 L2 <= 7
MEASURE ALL
```

EXT:



```
L1 L2 <= 3
MEASURE ALL
```

ENC:



```
L1 L2 < 8
MEASURE ALL
```

🟩 L1 ▨ L2

Measure Coincident

CLASS:
SECONDARY
KEYWORD

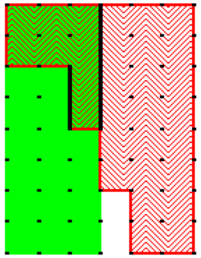
Purpose: Relaxes polygon containment criteria to measure coincident edge pairs

Syntax:

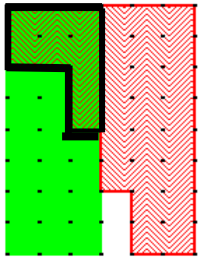
```
INT layer1 layer2 constraint MEASURE COINCIDENT
ENC layer1 layer2 constraint MEASURE COINCIDENT
```

Examples:

INT:

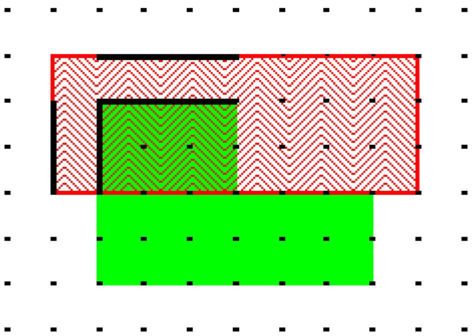


`L1 L2 <= 7`

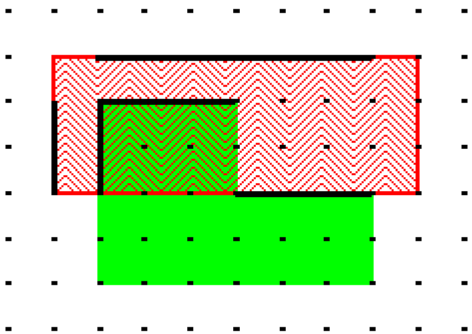


`L1 L2 <= 7 MEASURE COINCIDENT`

ENC:



`L1 L2 <= 3 OPPOSITE`



`L1 L2 <= 3 MEASURE COINCIDENT OPPOSITE`

🟩 L1

▨ L2

Connectivity Filter

- **The secondary keywords for connectivity instruct the dimensional RuleChecks to measure the separation between edges from polygons belonging to the same net.**
- **Secondary Keywords:**
  - `[NOT] CONNECTED`
- **Connectivity is covered in later modules.**

A Word About NOT Statements

- Some secondary keywords have a converse.
- The converse is the same as the original secondary keyword, only preceded by `NOT`.
- This training only presents the positive operation—the one without the “`NOT`”.
- If a converse of a secondary keyword exists, the slide indicates it.
- The `NOT` in these operations is a Boolean set operation—it does not correspond to the SVRF operation of the same name.

STATEMENT OVERVIEW

Connected

CLASS: SECONDARY KEYWORD

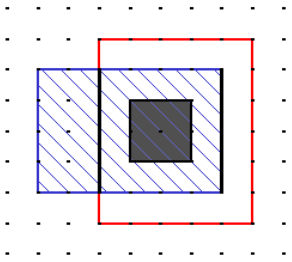
Purpose: Measures edge pairs only from polygons on the same net

Syntax:

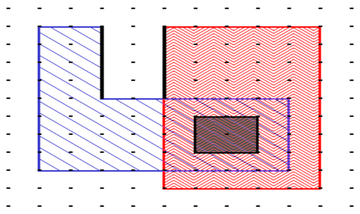
```
INT layer1 layer2 constraint [NOT] CONNECTED
EXT layer1 [layer2] constraint [NOT] CONNECTED
ENC layer1 layer2 constraint [NOT] CONNECTED
```

Default: Dimensional RuleChecks ignore connectivity

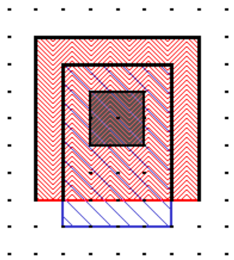
`layer1` and *`layer2`* must possess pre-established connectivity



```
CONNECT M1 POLY BY CONTACT
INT M1 POLY <= 4 CONNECTED
```



```
CONNECT M1 POLY BY CONTACT
EXT M1 POLY <= 4 CONNECTED
```



```
CONNECT M1 POLY BY CONTACT
ENC M1 POLY < 2 CONNECTED
```

■ CONTACT

▨ M1

▦ POLY

Orientation Filters

- **The secondary keywords for orientation instruct dimensional RuleChecks to measure the separation between edge pairs based on their appropriate angle or edge orientation.**
- **Secondary Keywords:**
  - `[NOT] ACUTE [ONLY | ALSO]`
  - `[NOT] PARALLEL [ONLY | ALSO]`
  - `[NOT] PERPENDICULAR [ONLY | ALSO]`
  - `[NOT] OBTUSE [ONLY | ALSO]`
- **Specify either `ONLY` or `ALSO`—unless `NOT` is used.**
- **If specifying `NOT`—may not use either `ONLY` or `ALSO`.**

Acute

Purpose: Measures between edge pairs forming appropriate angles > 0 and < 90 degrees

Syntax:

```
INT layer1 [layer2] constraint
        [NOT] ACUTE [ONLY | ALSO]
EXT layer1 [layer2] constraint
        [NOT] ACUTE [ONLY | ALSO]
ENC layer1 layer2 constraint
        [NOT] ACUTE [ONLY | ALSO]
```

Parameter:

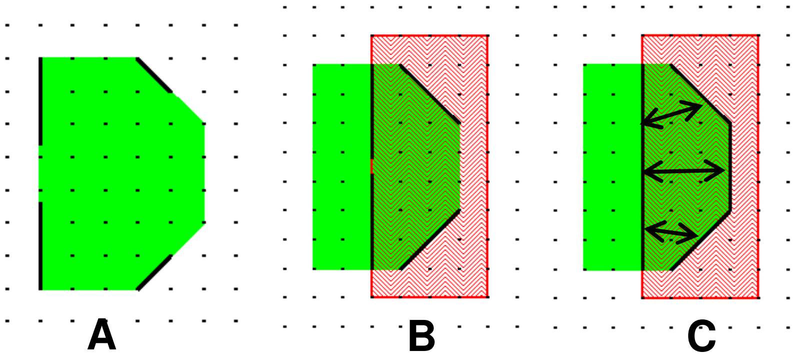
`ONLY` — measures only edge pairs with angular separation > 0 and < 90 degrees

`ALSO` — measures edge pairs with angular separation > 0 and < 90 degrees in addition to other angles

Default: **`ACUTE ALSO`**

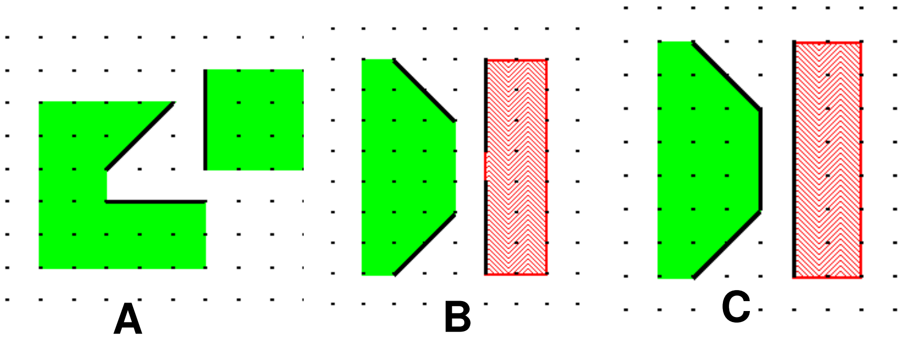
Acute — Examples

INT:



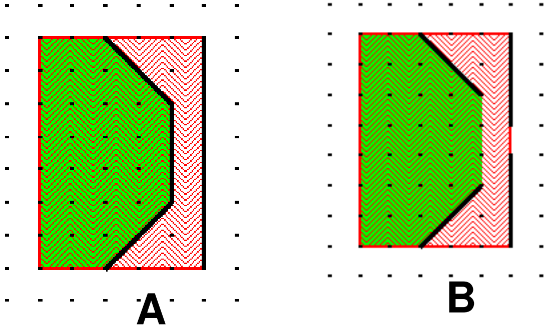
```
A: INT L1 <= 4 ACUTE ONLY
B: INT L1 L2 <= 3 ACUTE ONLY
C: INT L1 L2 <= 3 ACUTE ALSO
```

EXT:



```
A: EXT L1 <= 3 ACUTE ONLY
B: EXT L1 L2 <= 3 ACUTE ONLY
C: EXT L1 L2 <= 3 ACUTE ALSO
```

ENC:



```
A: ENC L1 L2 < 3 ACUTE ALSO
B: ENC L1 L2 < 3 ACUTE ONLY
```

🟩 L1

▨ L2

Parallel

Purpose: Measures parallel edge pairs

Syntax:

```
INT layer1 [layer2] constraint
        [NOT] PARALLEL [ONLY | ALSO]
EXT layer1 [layer2] constraint
        [NOT] PARALLEL [ONLY | ALSO]
ENC layer1 layer2 constraint
        [NOT] PARALLEL [ONLY | ALSO]
```

Parameter:

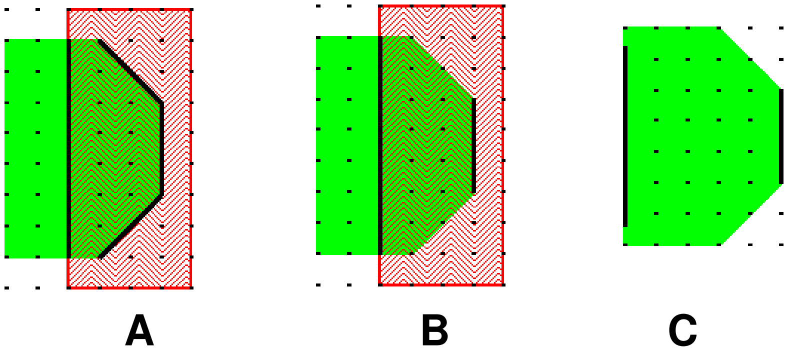
`ONLY` — measures only parallel edge pairs

`ALSO` — measures parallel edge pairs in addition to other edges

Default: `PARALLEL ALSO`

Parallel — Examples

INT:

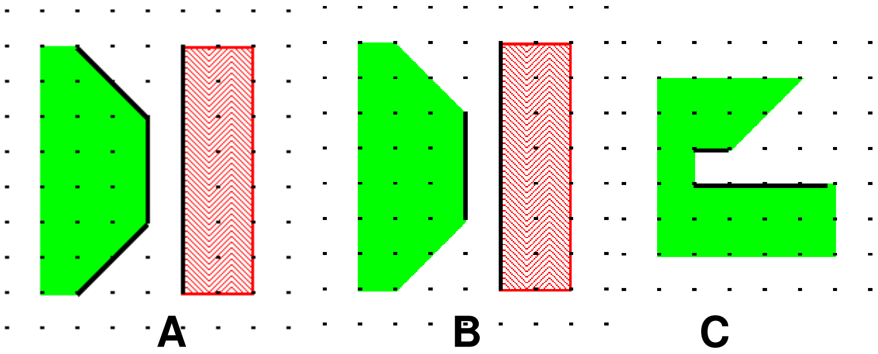


A: `INT L1 L2 <= 4 PARALLEL ALSO`

B: `INT L1 L2 <= 4 PARALLEL ONLY`

C: `INT L1 <= 5.2 PARALLEL ONLY`

EXT:

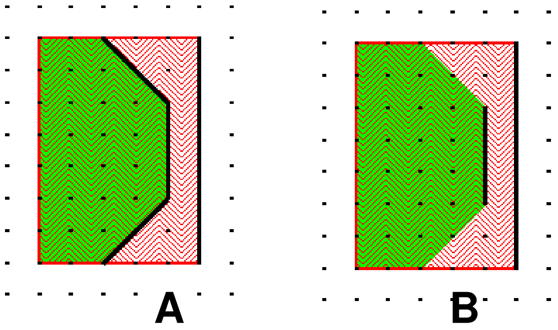


A: `EXT L1 L2 <= 3 PARALLEL ALSO`

B: `EXT L1 L2 <= 3 PARALLEL ONLY`

C: `EXT L1 <= 3 PARALLEL ONLY`

ENC:



A: `ENC L1 L2 < 3 PARALLEL ALSO`

B: `ENC L1 L2 < 3 PARALLEL ONLY`

🟩 L1

🟥 L2

Perpendicular

Purpose: Measures perpendicular edge pairs

Syntax:

```
INT layer1 [layer2] constraint
        [NOT] PERPENDICULAR [ONLY | ALSO]
EXT layer1 [layer2] constraint
        [NOT] PERPENDICULAR [ONLY | ALSO]
ENC layer1 layer2 constraint
        [NOT] PERPENDICULAR [ONLY | ALSO]
```

Parameter:

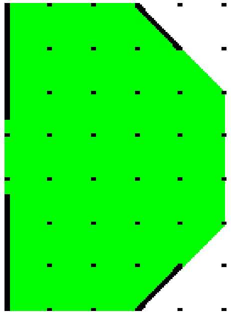
`ONLY` — measures only perpendicular edges

`ALSO` — measures perpendicular edges in addition to other edges

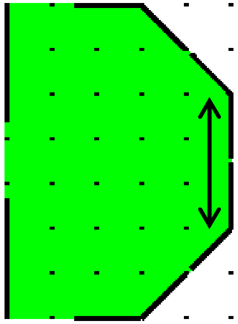
Default: `NOT PERPENDICULAR`

- **Edge pairs may be skew with respect to the coordinate axes.**

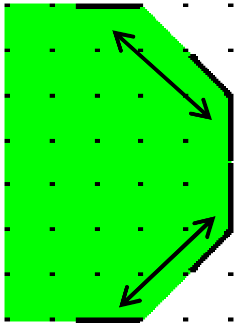
Perpendicular (Internal) — Examples



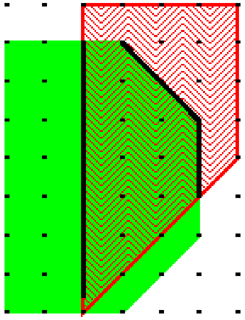
INT L1 <= 4



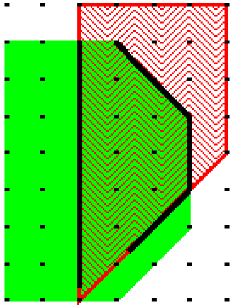
INT L1 <= 4 PERP ALSO



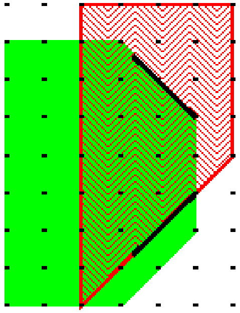
INT L1 <= 4 PERP ONLY



INT L1 L2 <= 4



INT L1 L2 <= 4
PERP ALSO

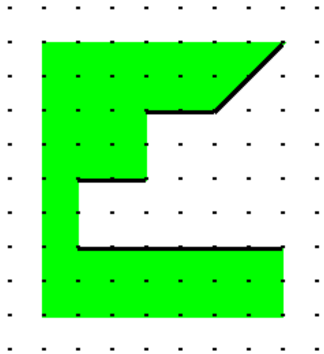


INT L1 L2 <= 4
PERP ONLY

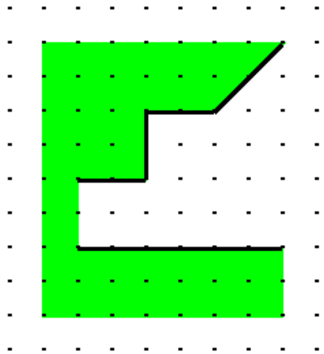
🟩 L1

▨ L2

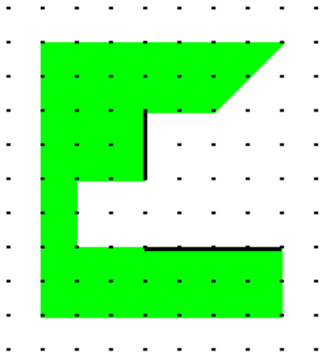
Perpendicular (External) — Examples



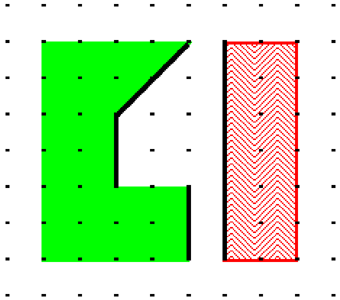
EXT L1 <= 6
NOT PERP



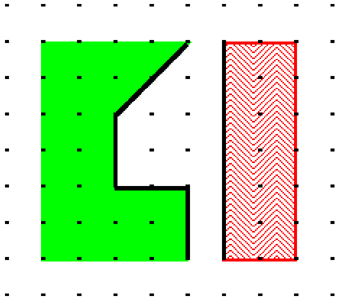
EXT L1 <= 6
PERP ALSO



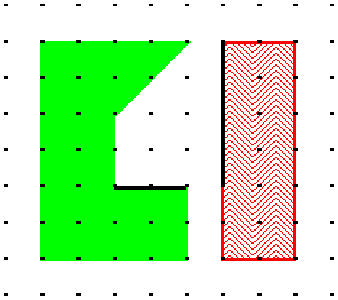
EXT L1 <= 6
PERP ONLY



EXT L1 L2 <= 3
NOT PERP



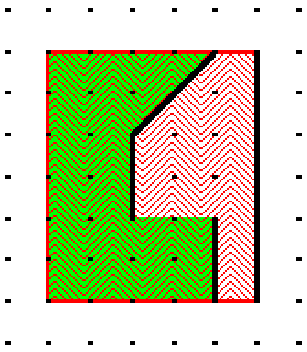
EXT L1 L2 <= 3
PERP ALSO



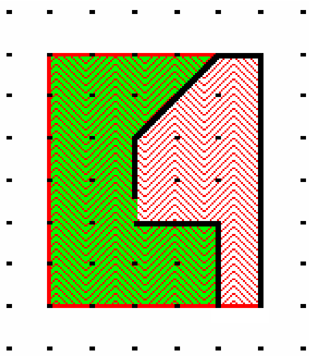
EXT L1 L2 <= 3
PERP ONLY

🟩 L1
▨ L2

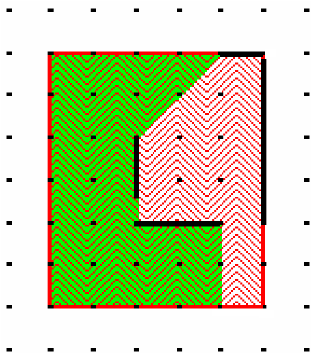
Perpendicular (Enclosure) — Examples



```
ENC L1 L2 < 4
NOT PERP
```



```
ENC L1 L2 < 4
PERP ALSO
```



```
ENC L1 L2 < 4
PERP ONLY
```

🟩 L1

▨ L2

Obtuse

Purpose: Measures between edge pairs forming appropriate angles > 90 and <180 degrees

Syntax:

```
INT layer1 [layer2] constraint
        [NOT] OBTUSE [ONLY | ALSO]
ENT layer1 [layer2] constraint
        [NOT] OBTUSE [ONLY | ALSO]
ENC layer1 [layer2] constraint
        [NOT] OBTUSE [ONLY | ALSO]
```

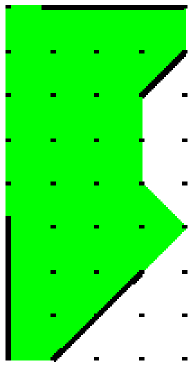
Parameter:

`ONLY` — measures only obtuse edge pairs

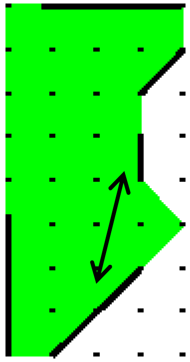
`ALSO` — measures obtuse edge pairs in addition to other edge pairs

Default: **`NOT OBTUSE`**

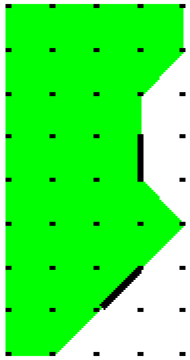
Obtuse (Internal) — Examples



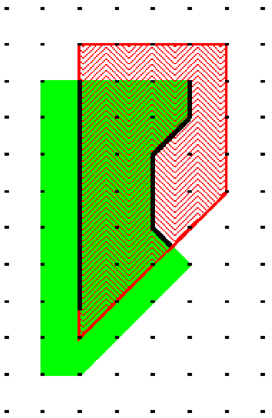
INT L1 < 3



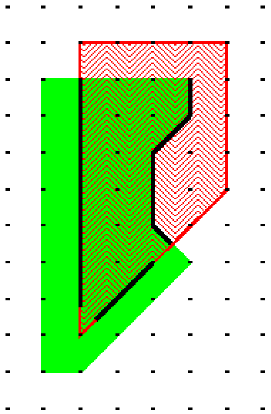
INT L1 < 3
OBTUSE ALSO



INT L1 < 3
OBTUSE ONLY

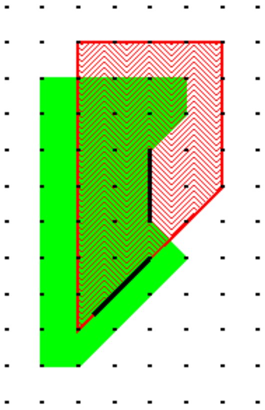


INT L1 L2 <= 3



INT L1 L2 <= 3

OBTUSE ALSO



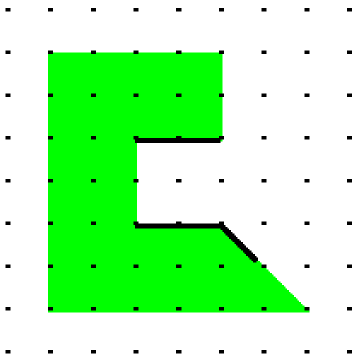
INT L1 L2 <= 3

OBTUSE ONLY

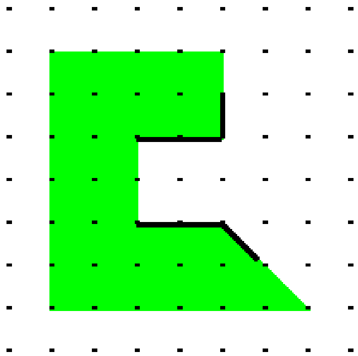
🟩 L1

▨ L2

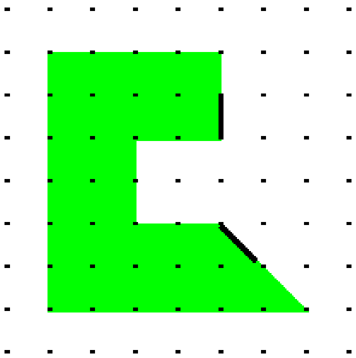
Obtuse (External) — Examples



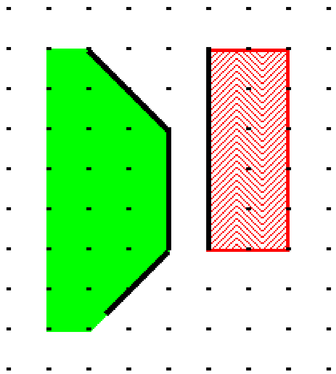
EXT L1 <= 3



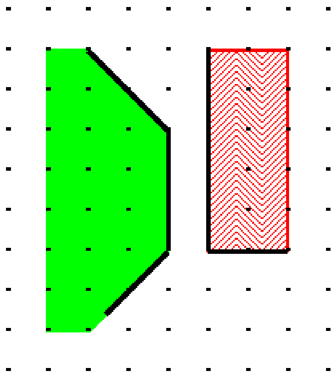
EXT L1 <= 3
OBTUSE ALSO



EXT L1 <=3
OBTUSE ONLY

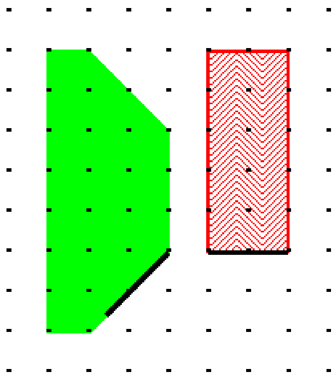


EXT L1 L2 <= 3



EXT L1 L2 <= 3

OBTUSE ALSO



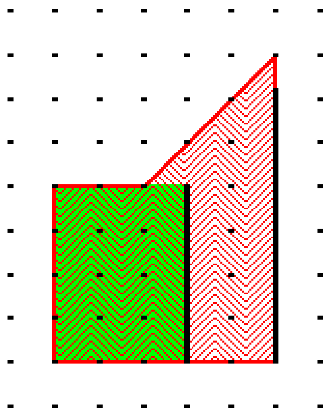
EXT L1 L2 <= 3

OBTUSE ONLY

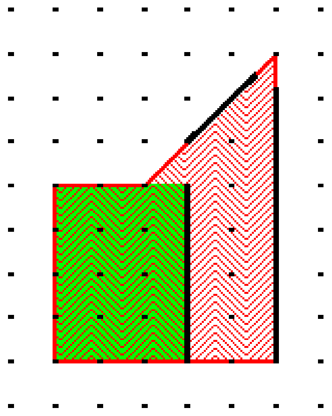
🟩 L1

▨ L2

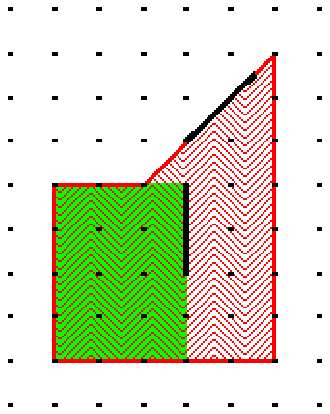
Obtuse (Enclosure) — Examples



ENC L1 L2 < 3



ENC L1 L2 < 3
OBTUSE ALSO



ENC L1 L2 < 3
OBTUSE ONLY

🟩 L1
▨ L2

Angle Filter

- The secondary keyword for the angle filter instructs dimensional RuleChecks to measure edge pairs based on orthogonality with respect to the coordinate axes.
- Secondary Keyword:
  - `ANGLED`

STATEMENT
OVERVIEW

Angled

CLASS:
SECONDARY
KEYWORD

Purpose: Measures edge pairs only when the number of edges which are non-orthogonal with respect to the coordinate axes satisfies the parameter

Syntax:

```
INT layer1 layer2 constraint ANGLED [parameter]
EXT layer1 layer2 constraint ANGLED [parameter]
ENC layer1 layer2 constraint ANGLED [parameter]
```

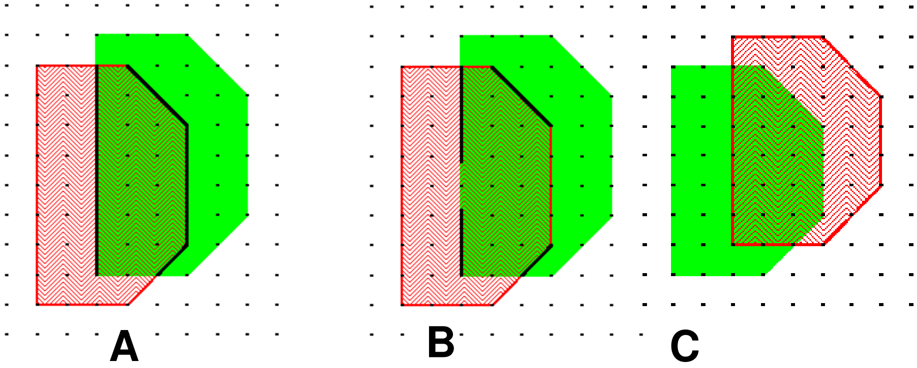
Parameter:

`parameter` – number or range of numbers from the set {0,1,2} specifying the number of edges in the pair which are non-orthogonal with respect to the coordinate axes

Default: `ANGLED > 0`

Angled — Examples

INT:

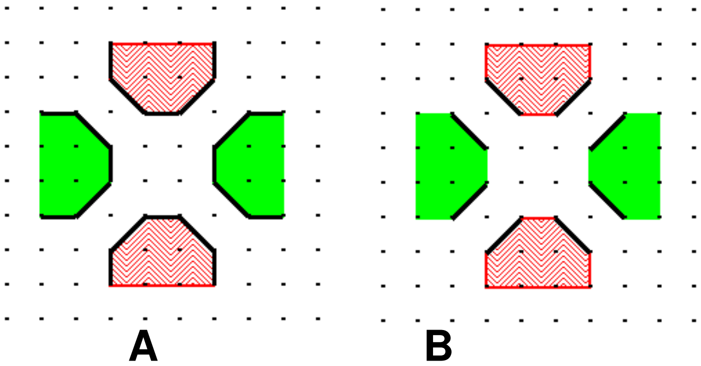


A: `INT L1 L2 <=3`

B: `INT L1 L2 <=3 ANGLED`

C: `INT L1 L2 <=3 ANGLED == 2`

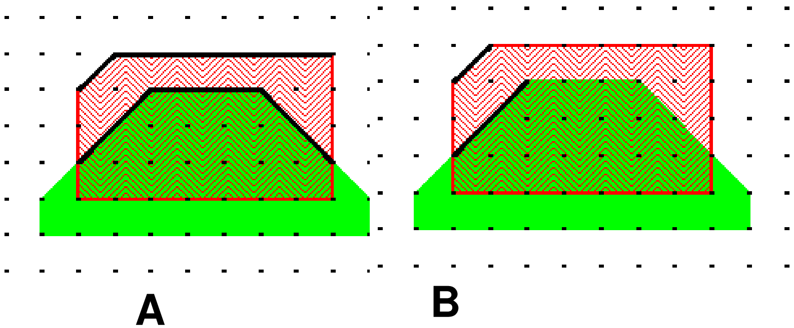
EXT:



A: `EXT L1 L2 <= 3`

B: `EXT L1 L2 <= 3 ANGLED == 2`

ENC:



A: `ENC L1 L2 < 3`

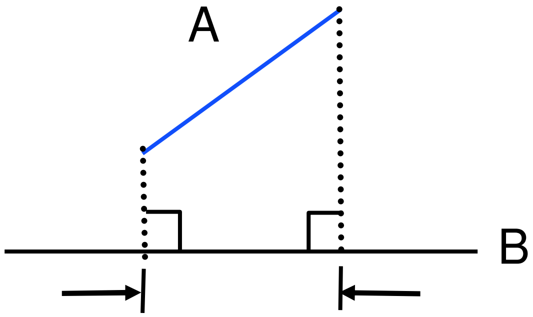
B: `ENC L1 L2 < 3 ANGLED == 2`

🟩 L1

▨ L2

Projection Filters

- The secondary keywords for projection instruct dimensional RuleChecks to measure the separation between edge pairs based on their mutual edge projection.
- Secondary Keywords:
  - `[NOT] PROJECTING`



Projection of A onto B

STATEMENT OVERVIEW

Projecting

CLASS: SECONDARY KEYWORD

Purpose: Measures the separation between two edges only when one edge projects onto the other edge and the length of the projection conforms to the given parameter

Syntax:

```
INT layer1 [layer2] constraint PROJECTING [parameter]
EXT layer1 [layer2] constraint PROJECTING [parameter]
ENC layer1 layer2 constraint PROJECTING [parameter]
```

Parameter:

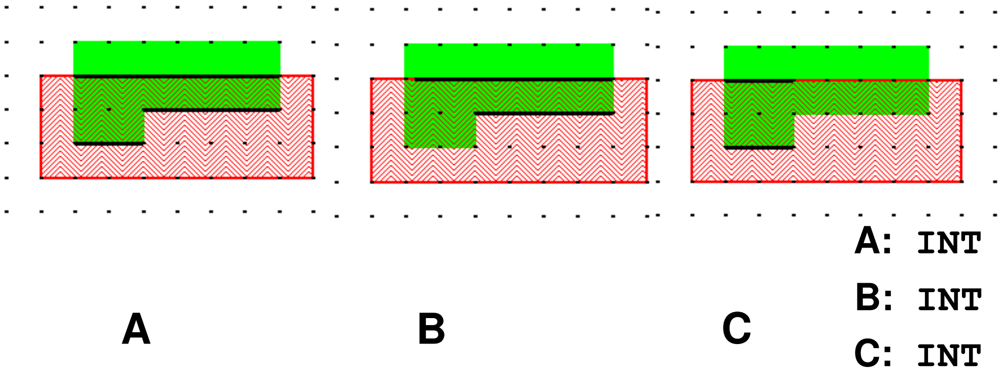
parameter — non-negative real value or range of values

Default: `PROJECTING >= 0`

> **NOTE:**
> **If a constraint is specified then `PARALLEL ONLY` will be set automatically.**

Projecting — Examples

INT:

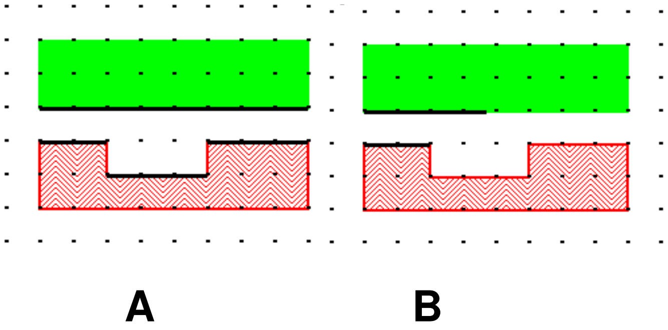


A: `INT L1 L2 <= 2`

B: `INT L1 L2 <= 2 PROJ > 2`

C: `INT L1 L2 <= 2 PROJ == 2`

EXT:



A: `EXT L1 L2 <= 2`

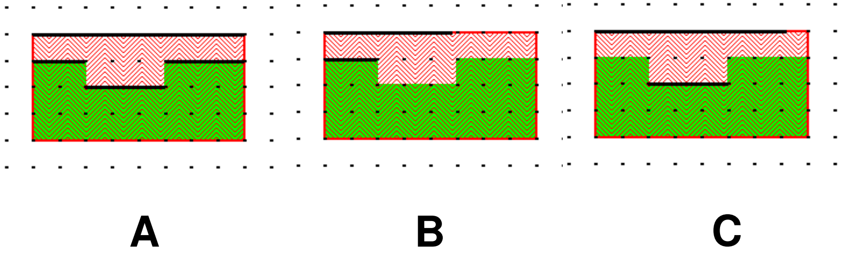
B: `EXT L1 L2 <=2 PROJ < 3`

ENC:

A: `ENC L1 L2 < 3`

B: `ENC L1 L2 < 3 PROJ < 3`

C: `ENC L1 L2 > 1 < 3 PROJ == 3`



🟩 L1

▨ L2

Output

- **Output modifiers instruct dimensional RuleChecks to generate a derived edge layer or derived polygon layer instead of a derived error layer.**
- **Output may be sent to either an intermediate layer or directly to the DRC results database.**
- **Modifiers:**
  - `[ ]`
  - `( )`
  - `REGION`

Output with [ ]

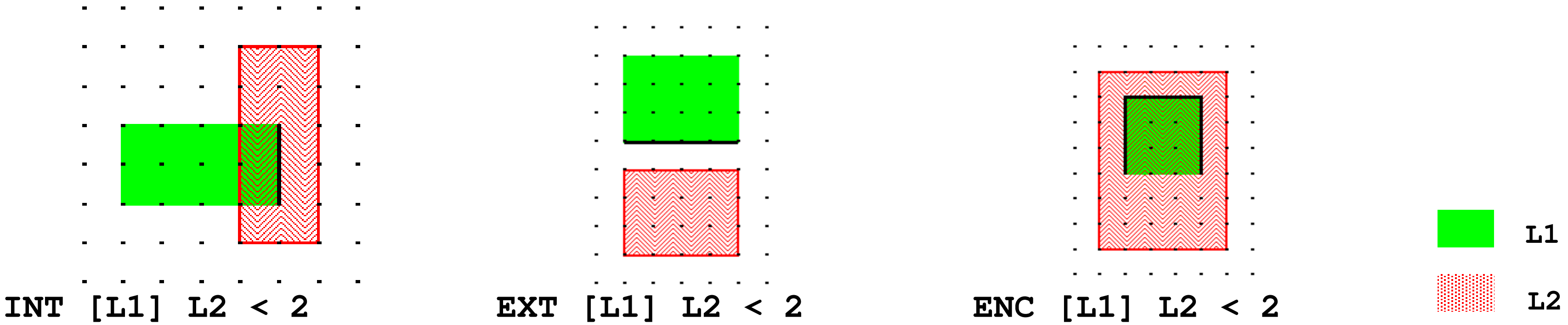
CLASS:
SECONDARY
KEYWORD

Purpose: Outputs error data from the specified input layer as positive output edge data

Syntax:

```
INT [layer1] layer2 constraint
INT layer1 [layer2] constraint
EXT [layer1] layer2 constraint
EXT layer1 [layer2] constraint
ENC [layer1] layer2 constraint
ENC layer1 [layer2] constraint
```

- **Only the edges satisfying the constraint for the bracketed layer are sent to output.**



`INT [L1] L2 < 2`

`EXT [L1] L2 < 2`

`ENC [L1] L2 < 2`

STATEMENT
OVERVIEW

Output with ( )

CLASS:
SECONDARY
KEYWORD

Purpose: Outputs error data from the specified input layer as negative output edge data

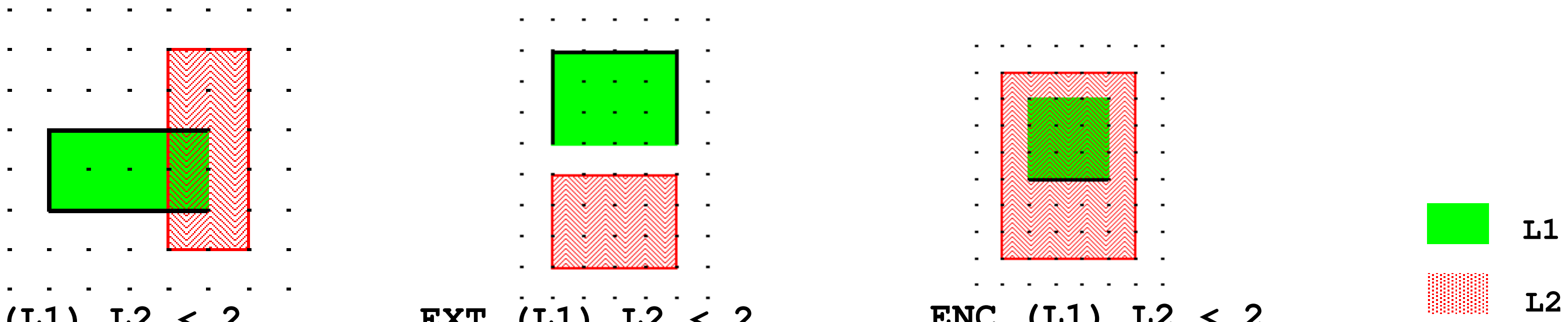
Syntax:

```
INT (layer1) layer2 constraint
INT layer1 (layer2) constraint

EXT (layer1) layer2 constraint
EXT layer1 (layer2) constraint

ENC (layer1) layer2 constraint
ENC layer1 (layer2) constraint
```

- **Only the edges not satisfying the constraint for the layer in parentheses are sent to output.**



`INT (L1) L2 < 2` `EXT (L1) L2 < 2` `ENC (L1) L2 < 2`

STATEMENT OVERVIEW

Region

CLASS: SECONDARY KEYWORD

Purpose: Outputs the error data from the specified input layers as a derived polygon layer

Syntax:

INT ***layer1*** **[*layer2*]** ***constraint***
REGION [EXTENTS| CENTERLINE [*value*]]
EXT ***layer1*** **[*layer2*]** ***constraint***
REGION [EXTENTS| CENTERLINE [*value*]]
ENC ***layer1 layer2 constraint***
REGION [EXTENTS| CENTERLINE [*value*]]

Parameters:

REGION— Constructs edge projections between the endpoints of selected edges to create polygonal regions

- This option may cause longer run times
- Use **REGION EXTENTS** to avoid creating non-Manhattan edges

Region (Cont.)

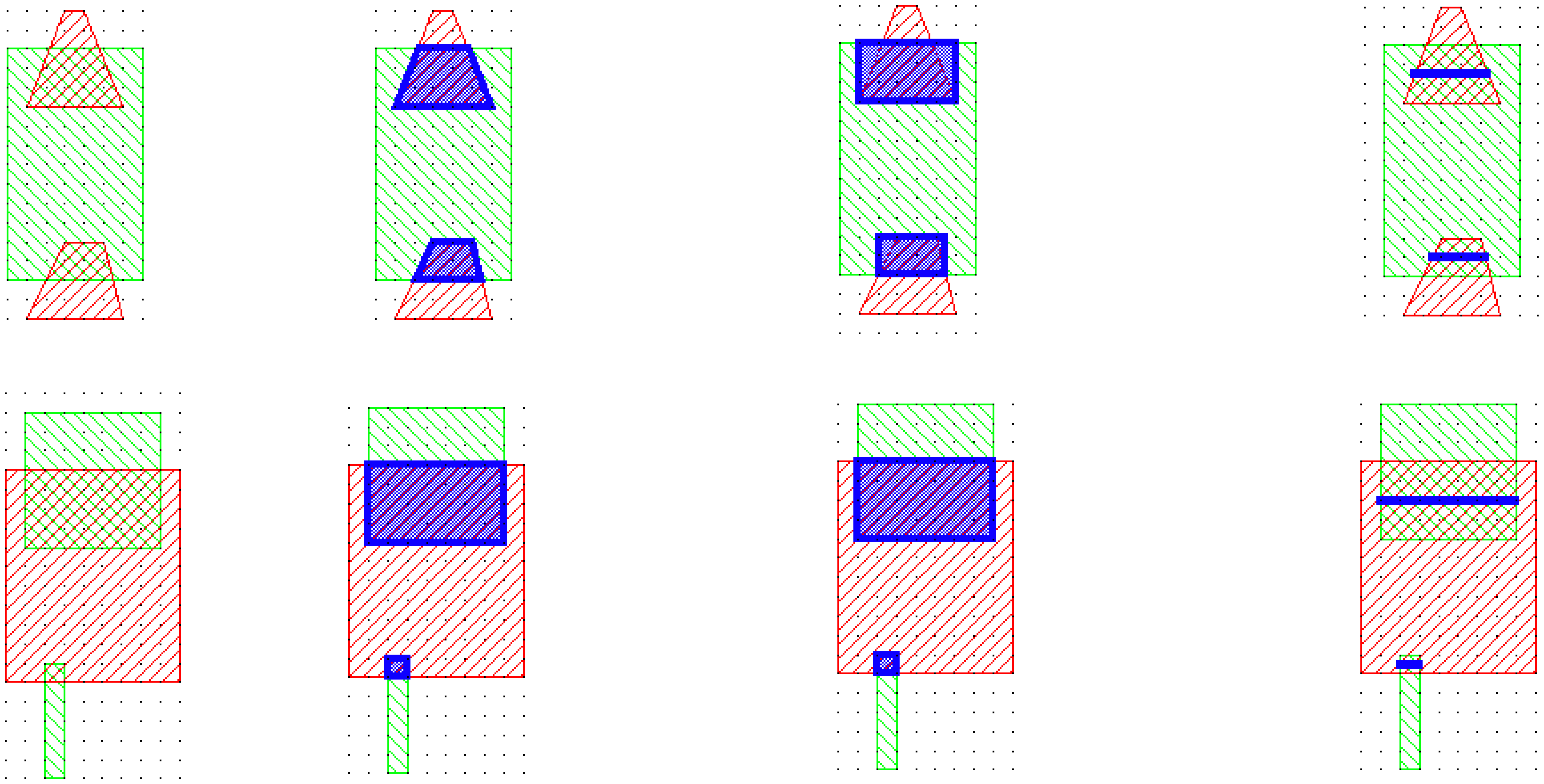
Parameters (Cont.):

`REGION EXTENTS`— Constructs derived polygon data as for `REGION`, but the output is the rectangular extents of the polygons output by `REGION`, rather than the polygons themselves.

`REGION CENTERLINE [`*`value`*`]`— Constructs derived polygon data as for `REGION`.

- The output consists of the centerlines of the polygonal regions, rather than the regions themselves.
- Centerlines are formed prior to the merging of the regions.
- Centerlines are along the direction of the edges whose measurement forms the region.
- Centerlines have a default width of eight database units.
- *`value`* allows you to specify the centerline width.
- *`value`* must be a floating-point number greater than or equal to two database units.

Internal Region Examples



`INT L1 L2 < 0.5`
`REGION`

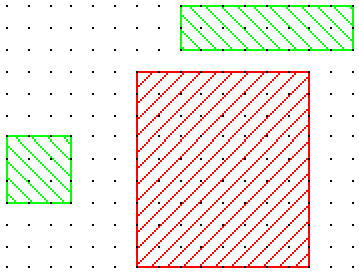
`INT L1 L2 < 0.5`
`REGION EXTENTS`

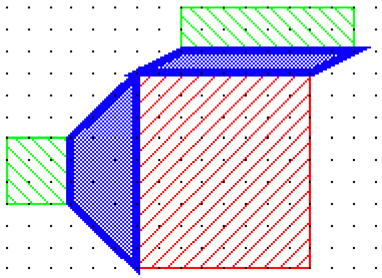
`INT L1 L2 < 0.5`
`REGION CENTERLINE`

🟩 L1
🟥 L2

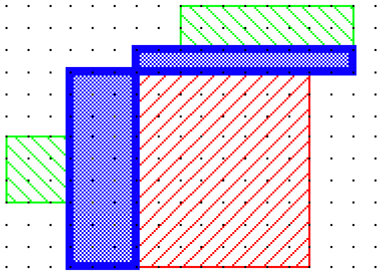
Grid spacing is 0.1 in these examples.

External Region Examples

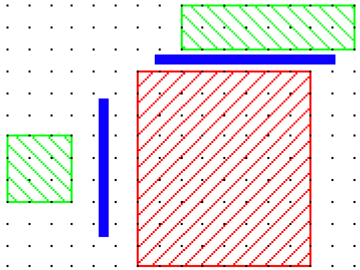




EXT L1 L2 < 0.5 REGION



EXT L1 L2 < 0.5 REGION EXTENTS



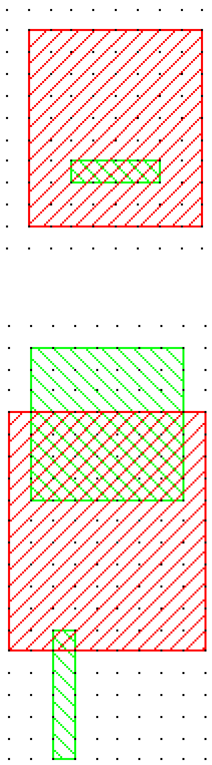
EXT L1 L2 < 0.5 REGION CENTERLINE

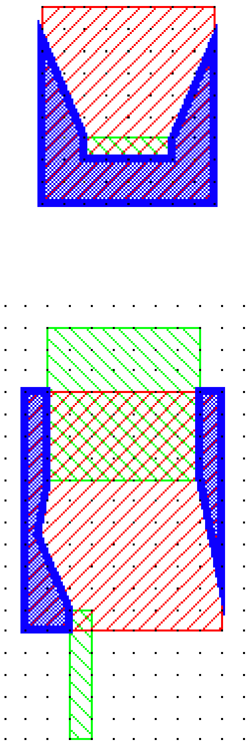
🟩 L1

🟥 L2

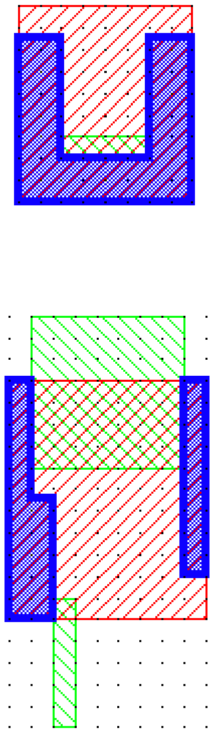
Grid spacing is 0.1 in these examples.

Enclosure Region Examples

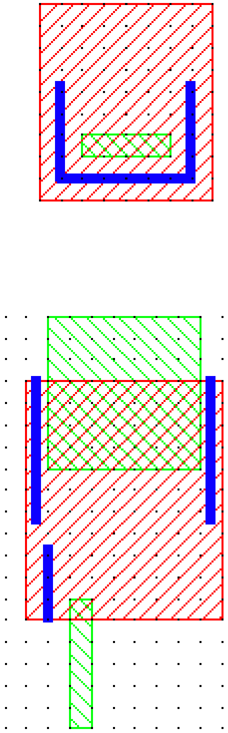




ENC L1 L2 <0.5
REGION



ENC L1 L2 <0.5
REGION EXTENTS



ENC L1 L2 <0.5
REGION CENTERLINE

▨ L1
▨ L2

Grid spacing is 0.1 in these examples.

Non-Universal Secondary Keywords

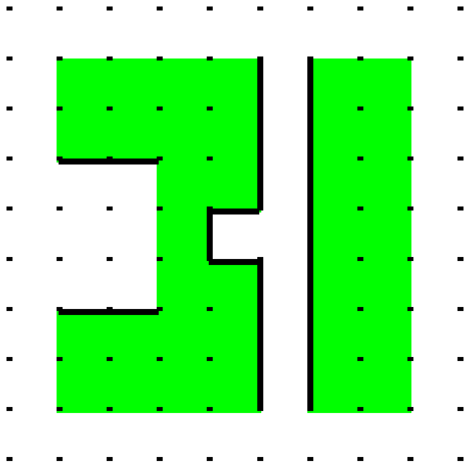
- **The previous set of secondary keywords is applicable to all dimensional checks (`INT`, `EXT` and `ENC`).**
- **Additional check-specific keywords which are not universal are also supported.**
- **Each of the following slides will indicate which dimensional checks are usable with the given keyword.**

Notch

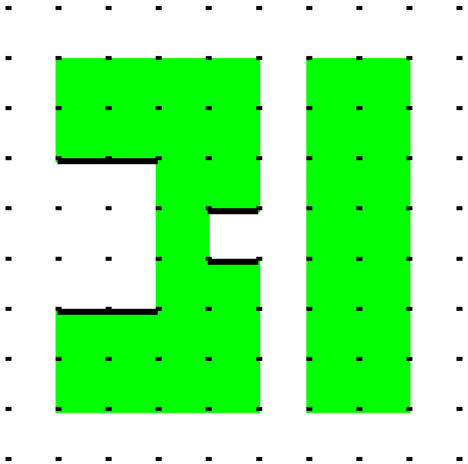
Purpose: Measures the separation between external edge pairs only from the same polygon

Syntax: `EXT layer constraint NOTCH`

Example:



`EXT L1 <= 3`



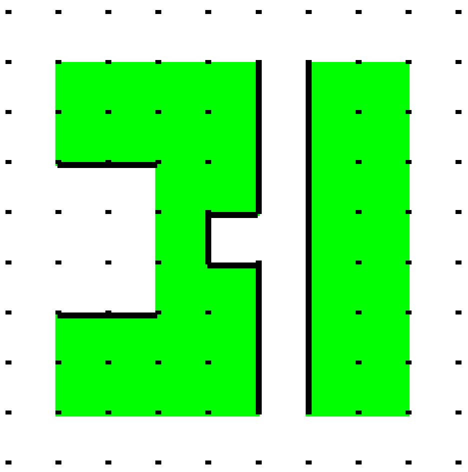
`EXT L1 <= 3 NOTCH`

Space

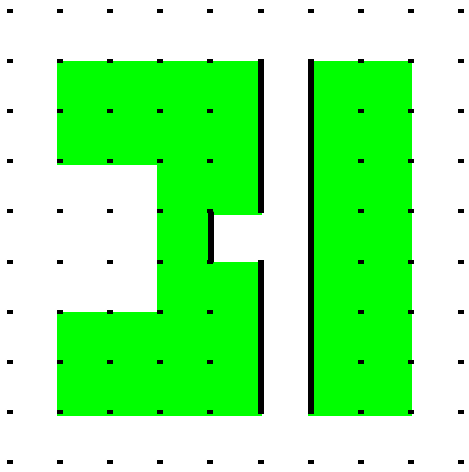
Purpose: Measures the separation between external edge pairs only from different polygons

Syntax: `EXT layer constraint SPACE`

Example:



`EXT L1 <= 3`



`EXT L1 <= 3 SPACE`

Inside Also

Purpose: For `EXTERNAL`, outputs the edges from **either** layer which are inside or coincident (but not **outside** coincident) to the other layer in addition to other edge pairs that meet the constraint. For `ENCLOSURE`, outputs edges from `layer2` which are inside (but not inside coincident) *`layer1`*.

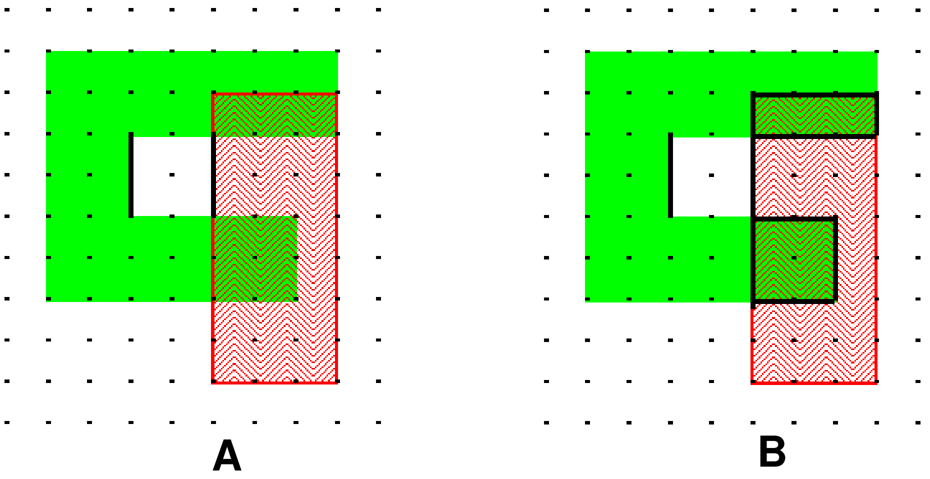
Syntax:

```
EXT layer1 layer2 constraint INSIDE ALSO
ENC layer1 layer2 constraint INSIDE ALSO
```

- **Edge output varies if either *`layer1`* or *`layer2`* are derived layers (consult the *SVRF Manual* for a complete description).**
- ***`Layer2`* edges which lie inside *`layer1`* do not need to meet the dimensional constraint.**

Inside Also — Examples

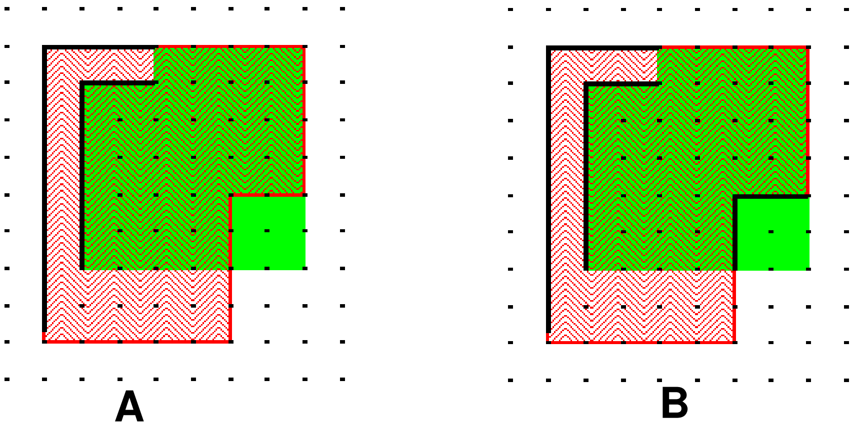
EXT:



A: `EXT L1 L2 < 3`

B: `EXT L1 L2 < 3`
`INSIDE ALSO`

ENC:



A: `ENC L1 L2 < 2`

B: `ENC L1 L2 < 2`
`INSIDE ALSO`

🟩 L1
▨ L2

Outside Also

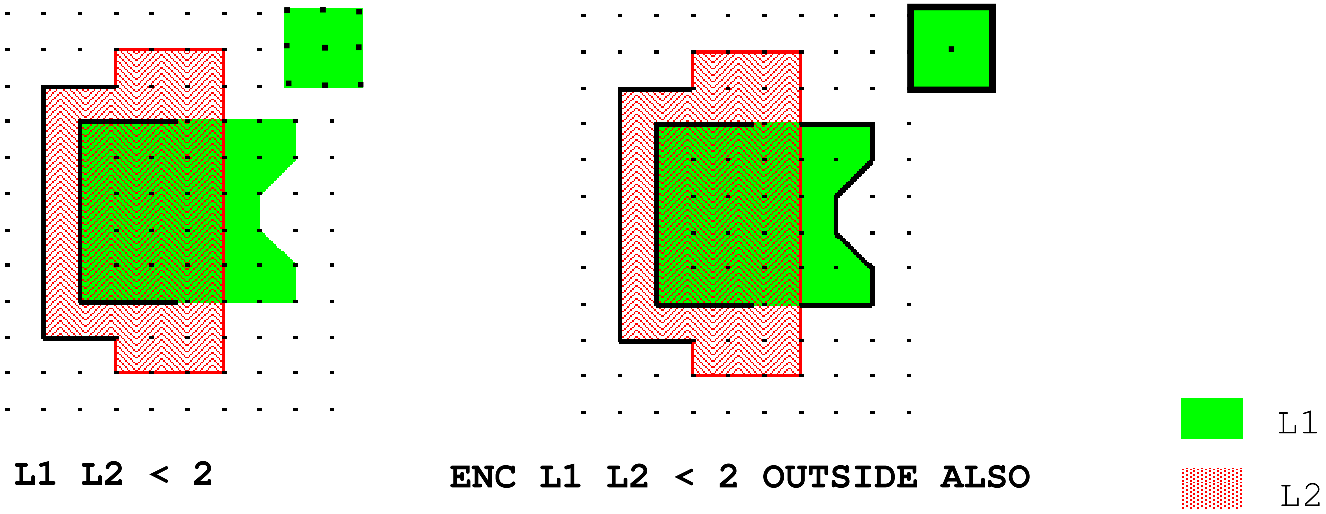
CLASS:
SECONDARY
KEYWORD

Purpose: Outputs the edges from the first layer which are outside or coincident to the second layer in addition to other edge pairs meeting the constraint

Syntax:

```
ENC layer1 layer2 constraint OUTSIDE ALSO
```

Example:



`ENC L1 L2 < 2`

`ENC L1 L2 < 2 OUTSIDE ALSO`

- **Edge output varies if either `layer1` or `layer2` are derived layers (consult the *SVRF Manual* for a complete description).**

Rectangle Enclosure

Purpose: Measures enclosure between enclosed rectangles when multiple rules may apply

Syntax:

```
RECTANGLE ENCLOSURE layer1 layer2
        [intersection_filter] [OUTSIDE ALSO]
        [ORTHOGONAL ONLY]
        {rectangle_rule[…rectangle_rule]}
```

Parameters:

layer1 — an original or derived polygon layer

layer2 — an original or derived polygon layer

intersection_filter — permits measurement of intersecting edge pairs — uses the format:

`[ABUT [constraint]] [SINGULAR]`

`OUTSIDE ALSO` — outputs edges from ***layer1*** not enclosed by ***layer2***

`ORTHOGONAL ONLY` — specifies processing only rectangles with edges parallel to the database coordinate axes

Rectangle Enclosure (Cont.)

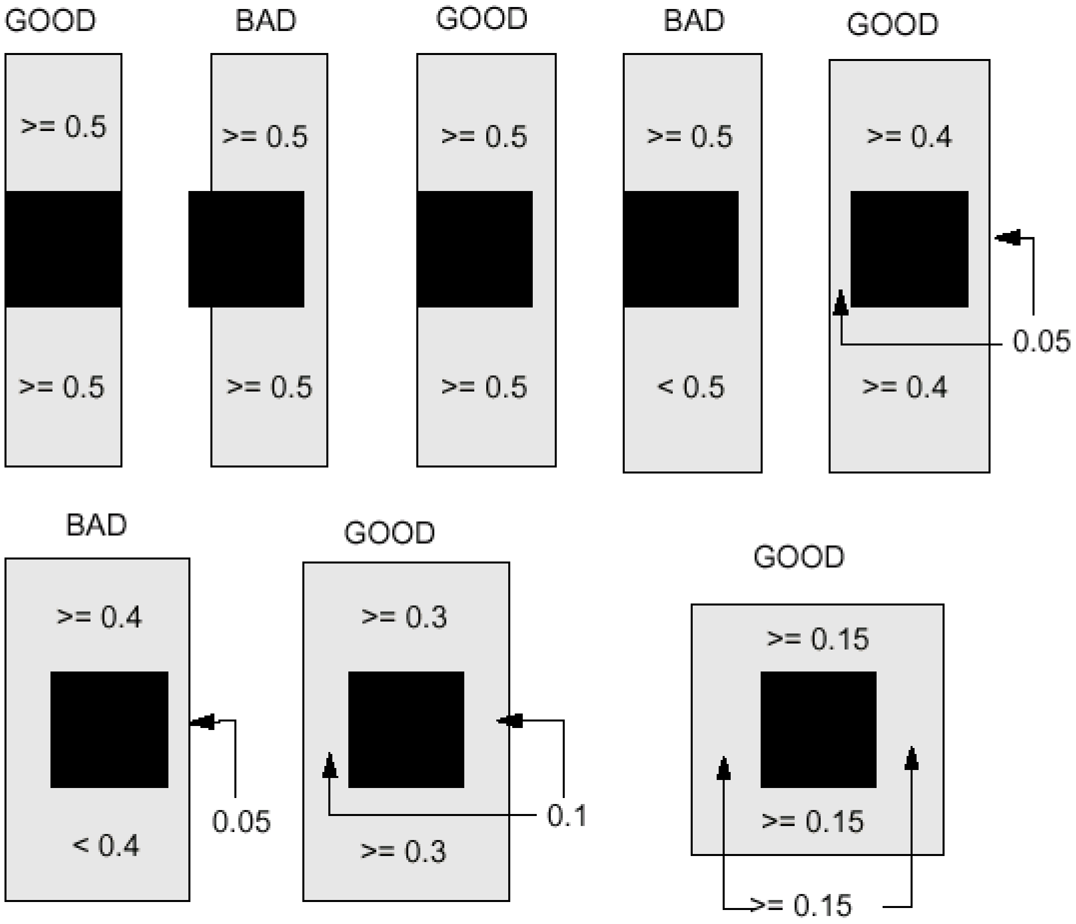
Example Specification:

- **Contacts must be enclosed by metal by 0.15.**
- **Exceptions:**
  - **Two opposite sides can each be as close as 0 if the other two sides are at least 0.5.**
  - **Two opposite sides can each be as close as .05 if the other two sides are at least 0.4.**
  - **Two opposite sides can be as close as 0.1 if the other two sides are at least 0.3.**
  - **All sides are at least 0.15.**
- **Syntax:**

```
Rule32 {RECTANGLE ENCLOSURE contact metal
  ABUT > 0 < 90
  SINGULAR
  OUTSIDE ALSO
  GOOD 0.00 0.50 0.00 0.50 // Anything not good is bad
  GOOD 0.05 0.40 0.05 0.40
  GOOD 0.10 0.30 0.10 0.30
  GOOD 0.15 0.15 0.15 0.15
  }
```

Rectangle Enclosure (Cont.)

Example Syntax Possible Results:



Calibre Rule Writing

Module 4

Polygon-Directed RuleChecks

Polygon-Directed Layer Operations

- Polygon-directed layer operations construct or select derived polygon layers from original layers or layer sets, or from derived layers.
- In most cases, an empty layer input to one of these operations will result in empty output.
- Unless otherwise stated, constraints specify polygon counts.
- For this module, original layers are assumed to include layer sets.

Boolean Operations

- Boolean operations include:
  - `AND`
  - `NOT`
  - `OR`
  - `XOR`
- These operations construct layers based upon Boolean logic as applied to sets of points belonging to specified layers.
- `AND` and `NOT` are net-preserving operations passing connectivity information between layers.
- Unmerged layers are presented to single-layer Boolean operations (Calibre will typically merge layers prior to performing operations).

AND

Purpose: Selects all polygon areas common to more than one polygon

Syntax:

```
AND layer [constraint] //single-layer AND
AND layer1 layer2 //two-layer AND
```

Parameters:

`layer` — original layer or layer set

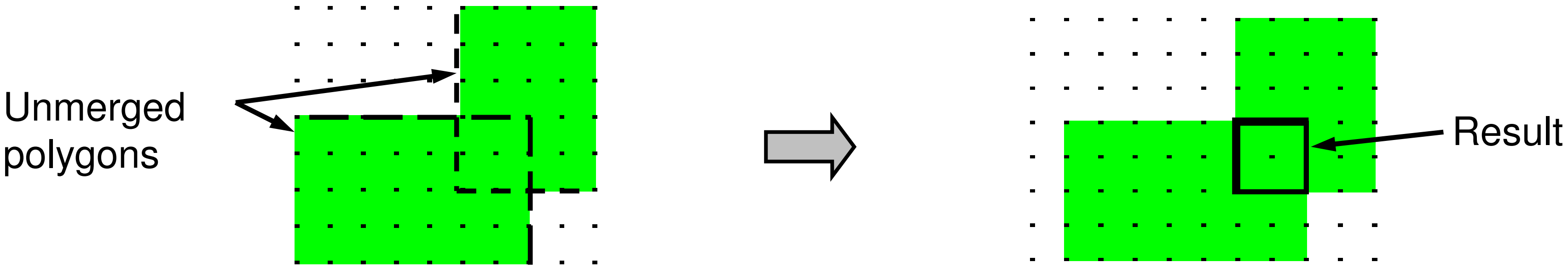
`layer1, layer2` — original or derived polygon layer

`constraint` — optional specification of integer value or range of values; default is >1

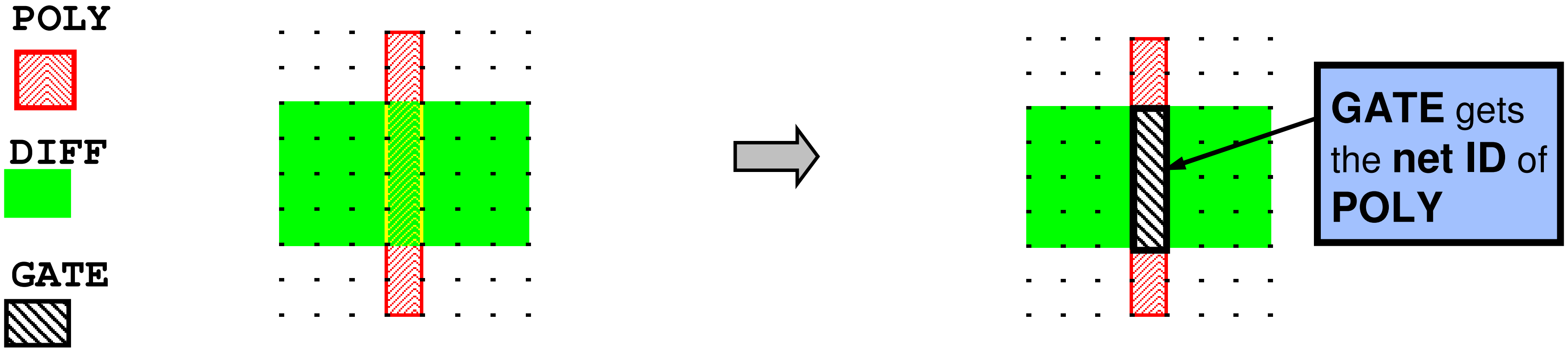
- **Single-layer `AND` operates on pre-merged original layers.**
  - ***`constraint`*** `==0` **results in empty output**
  - ***`constraint`*** `==1` **selects all non-overlapped areas of polygons**
- **Two-layer `AND` operates on merged original or derived layers.**
- **A layer derived from a two-layer `AND` operation receives the net ID of *`layer1`*.**

AND (Cont.)

Examples:



```
AND DIFF == 2
```



GATE gets the **net ID** of **POLY**

```
GATE = POLY AND DIFF
```

OR

Purpose: Merges overlapping polygons on the input layer into one polygon

Syntax:

```
OR layer                  //single-layer OR
OR layer1 layer2 //two-layer OR
```

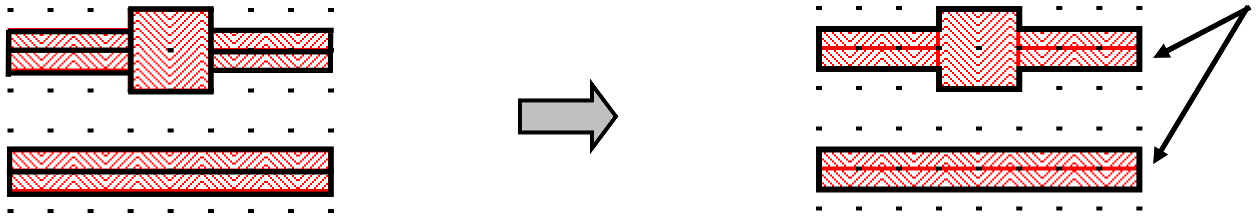
Parameters: ***layer*** — original layer
layer1, layer2 — original or derived polygon layers

- **Does NOT preserve connectivity.**
- **Single-layer `OR` operates on pre-merged original layers.**
  - **If *`layer`* is empty, output is empty.**
  - **Calibre automatically merges original layers before presenting them to most operations, so applicability of the single-layer `OR` is very limited.**
- **Two-layer `OR` operates on merged original or derived layers.**
  - **If *`layer1`* is empty and *`layer2`* is defined, only *`layer2`* polygons will be returned and vice-versa.**
  - **Interchanging *`layer1`* and *`layer2`* will not affect the output.**

OR (Cont.)

Examples:

Four
Unmerged
objects



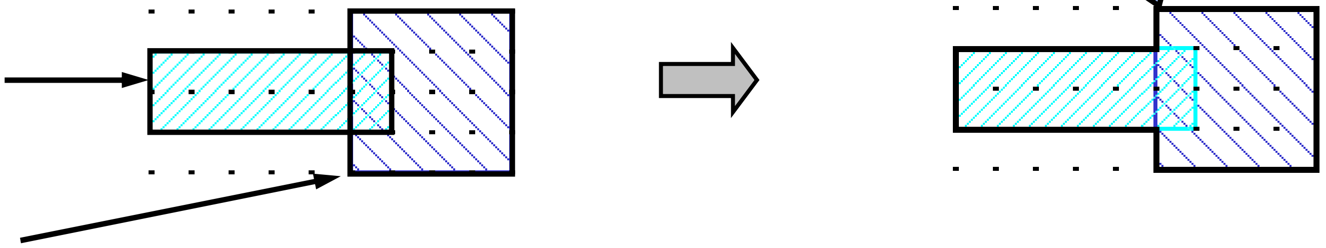
Results

`OR POLY`

Overlapping original polygons are merged but lose their original net connectivity

metal1

metal2



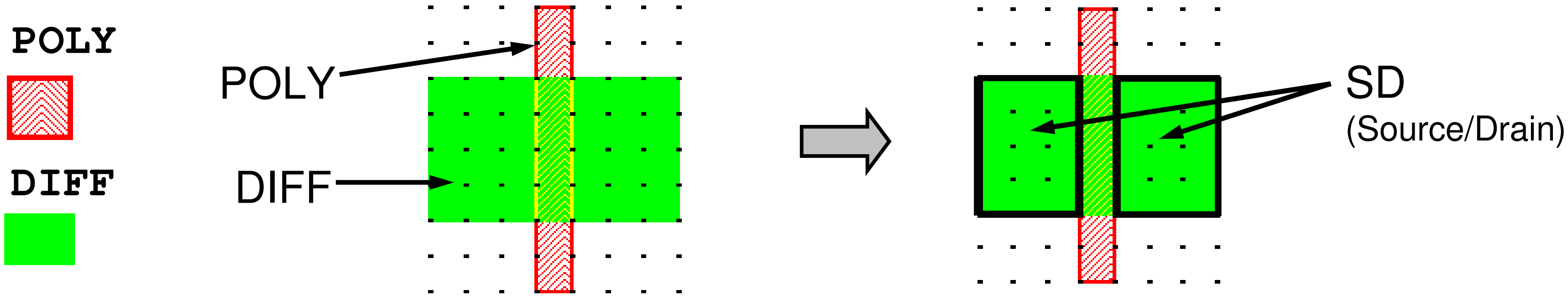
`METAL1 OR METAL2`

STATEMENT OVERVIEW

NOT

CLASS: OPERATION

Purpose:	Selects all ***layer1*** polygon areas not common to ***layer2*** polygon areas
Syntax:	`NOT` ***`layer1 layer2`***
Parameters:	***layer1, layer2*** — original or derived polygon layers
Example:	`SD = DIFF NOT POLY`



- **A layer derived from the `NOT` operation receives the net name of *layer1*.**
- **Interchanging *layer1* and *layer2* will give different geometric and connectivity results.**

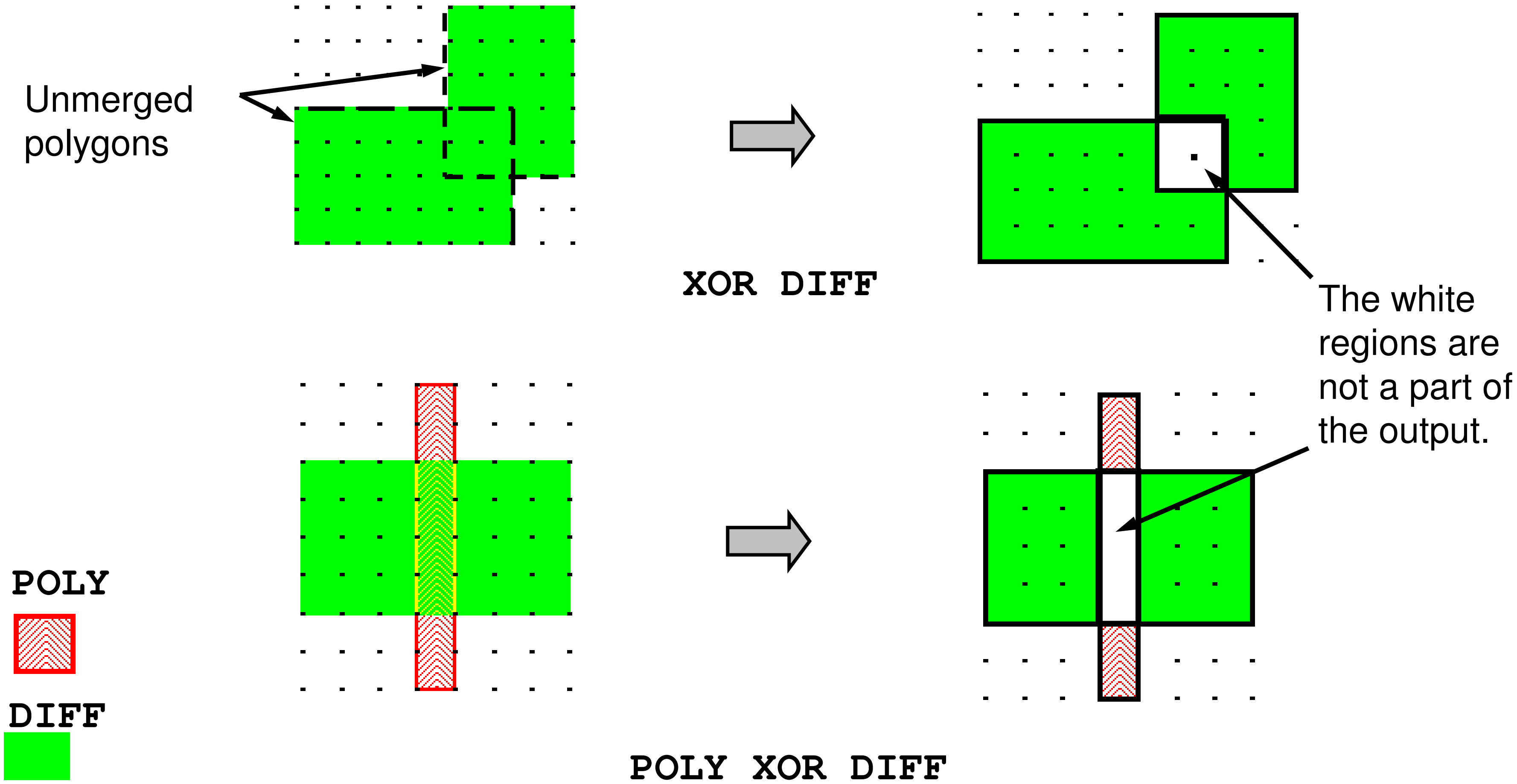
XOR

Purpose:	Selects all polygon areas common to exactly one polygon
Syntax:	`XOR layer   //single-layer XOR`
	`XOR layer1 layer2 //two-layer XOR`
Parameters:	*`layer`* — original layer
	`layer1, layer2` — original or derived polygon layers

- **Does NOT preserve connectivity.**
- **Single-layer `XOR` operates on pre-merged original layers.**
  - **If layer is empty, output is empty.**
  - **Single-layer `XOR` is equivalent to the operation: `AND layer ==1`.**
- **Two-layer `XOR` operates on merged original or derived layers.**
  - **If *`layer1`* is empty and *`layer2`* is not, only *`layer2`* polygons will be returned and vice-versa.**
  - **Interchanging *`layer1`* and *`layer2`* will not affect the output.**

XOR (Cont.)

Examples:



Unmerged polygons

XOR DIFF

The white regions are not a part of the output.

POLY

▨

DIFF

🟩

POLY XOR DIFF

Topological Operations

- Topological operations construct or select layers based upon inherent topological or geometric properties of polygons.
- Some operations have a converse operation as a counterpart.
  - Example: `contact TOUCH metal1`
    `contact NOT TOUCH metal1`
  - These represent a pair of converse operations.
  - When such a pair exists, this training will only present the positive operation (the one without the NOT); if a converse of an operation exists, the slide will indicate so,
  - The `NOT` in this type of operation is a Boolean set operator and does not correspond to the SVRF statement of the same name.

STATEMENT OVERVIEW

Donut

CLASS: OPERATION

Purpose: Selects all input polygons having interior cycles (holes) conforming to the constraint

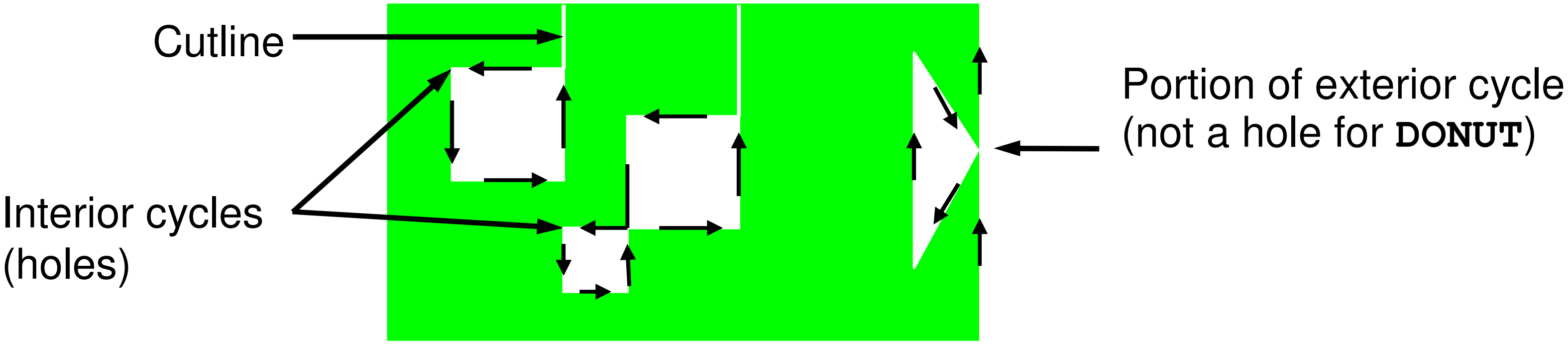
Syntax: `[NOT] DONUT layer [constraint]`

Parameters:

layer — original or derived polygon layers

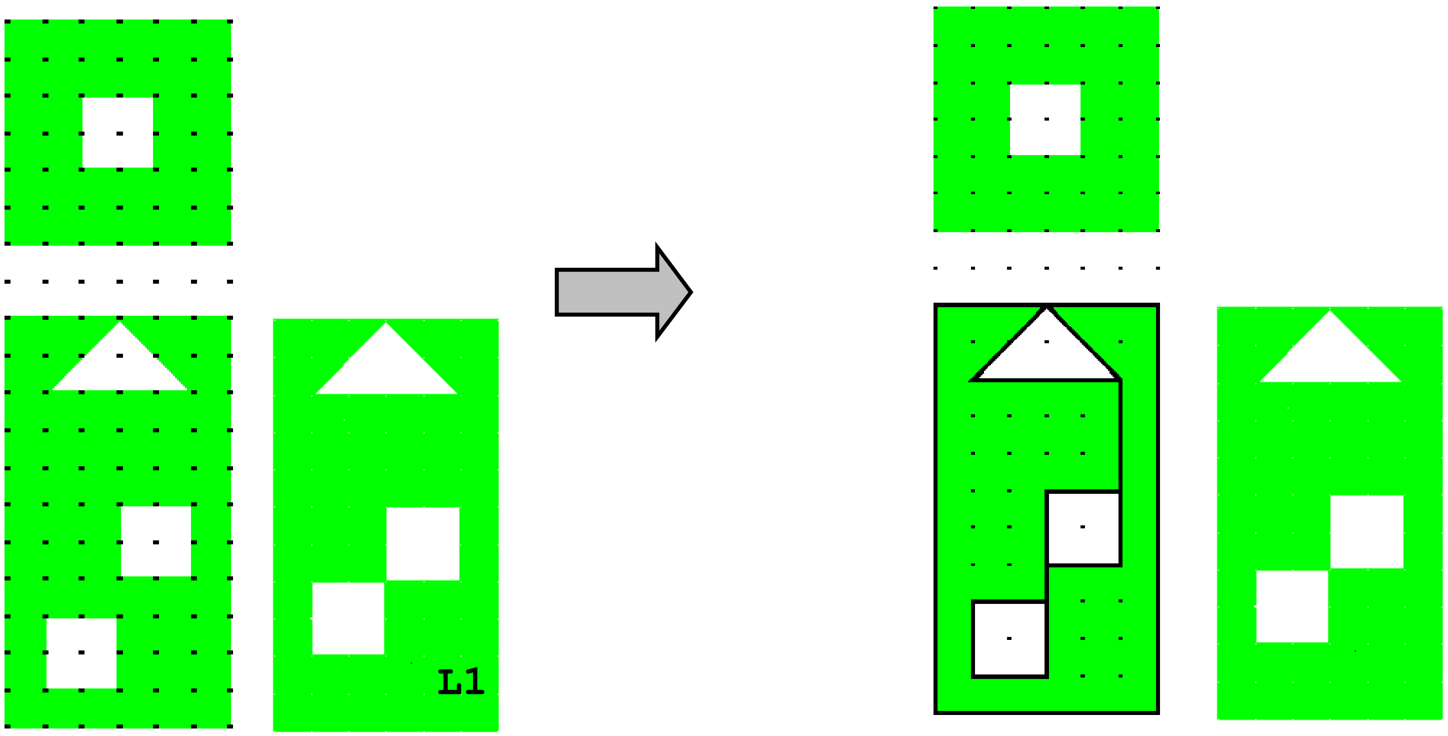
constraint — specifies the number of interior cycles a layer polygon must have to be selected; default is >= 1

An interior cycle is a set of vertices which, when connected, form a hole in the interior of a polygon.



Donut (Cont.)

Example:



`DONUT L1 == 2`

STATEMENT
OVERVIEW

CLASS:
OPERATION

Holes

Purpose: Forms a merged layer of polygons which fit exactly inside holes within the specified layer

Syntax: `HOLES layer [constraint] [INNER] [EMPTY]`

Parameters:

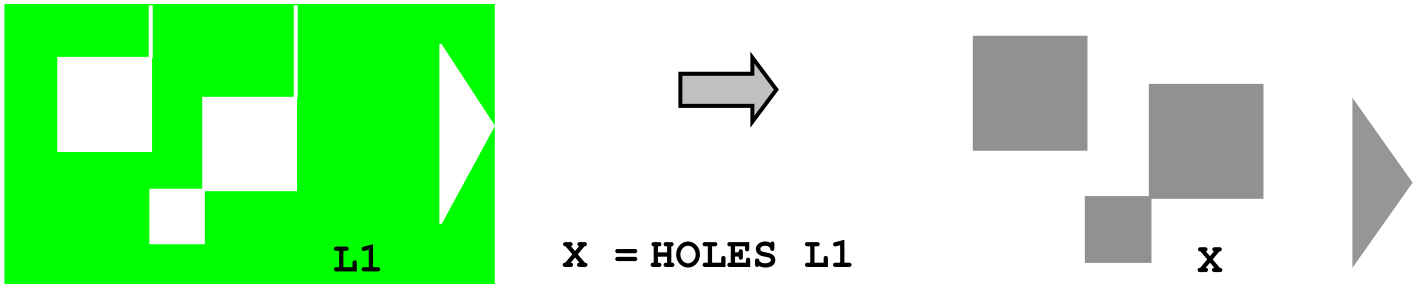
layer — original or derived polygon layers

constraint — limits hole selection to those whose area satisfies the constraint; integer value or range; default is >= 1

INNER — prevents holes containing other holes being output

EMPTY — prevents output of holes not outside layer

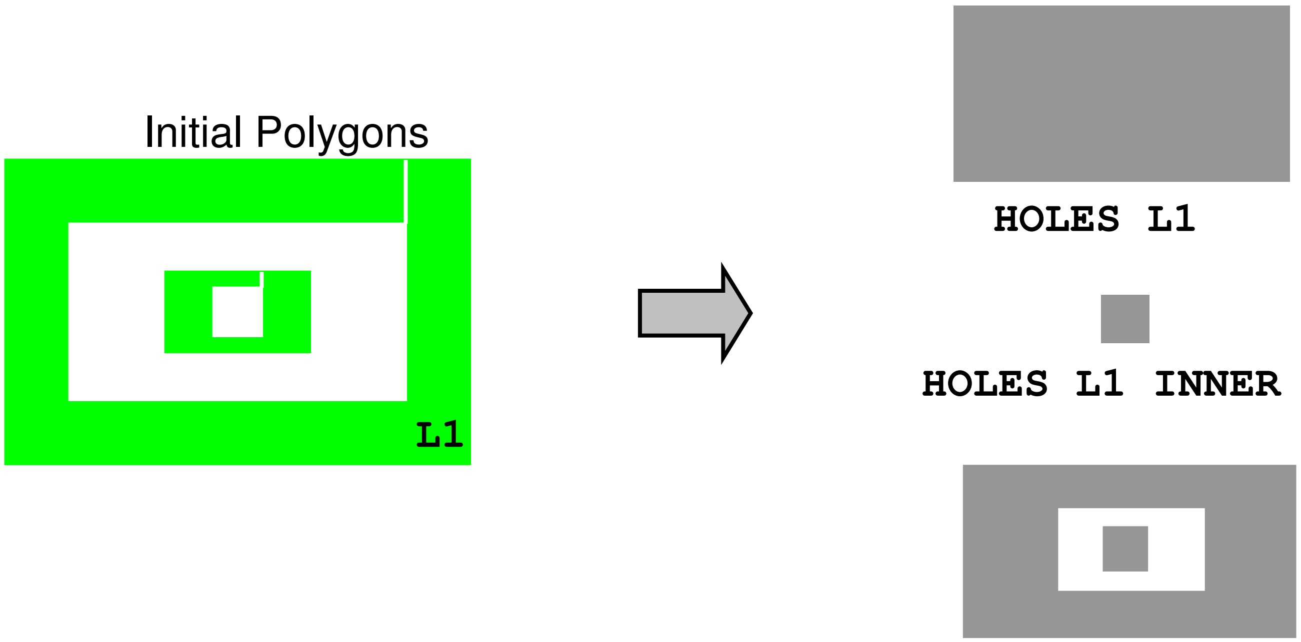
Example:



`X = HOLES L1`

Holes (Cont.)

- Polygonal holes inside other polygonal holes can produce unexpected results with this operation.
- Holes inside other holes are merged away rather than producing separate or overlapping polygons
- Example:



Initial Polygons

`L1`

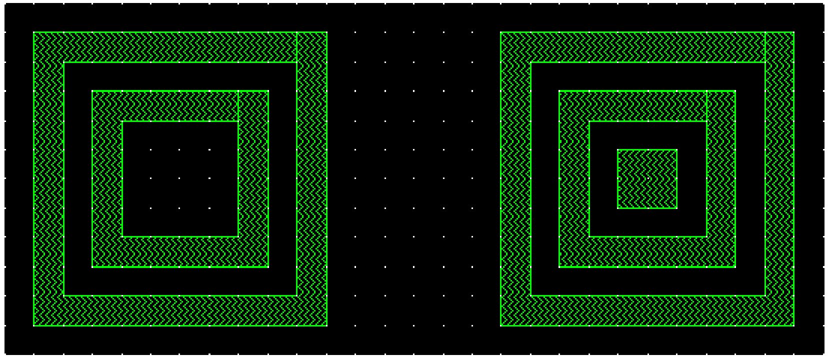
Results

`HOLES L1`

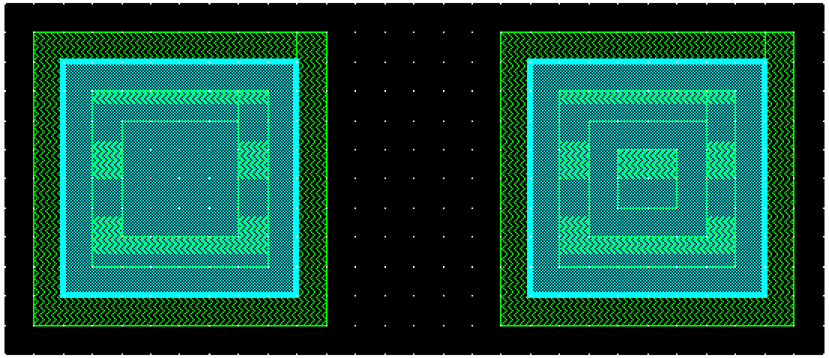
`HOLES L1 INNER`

`(HOLES L1)NOT L1`

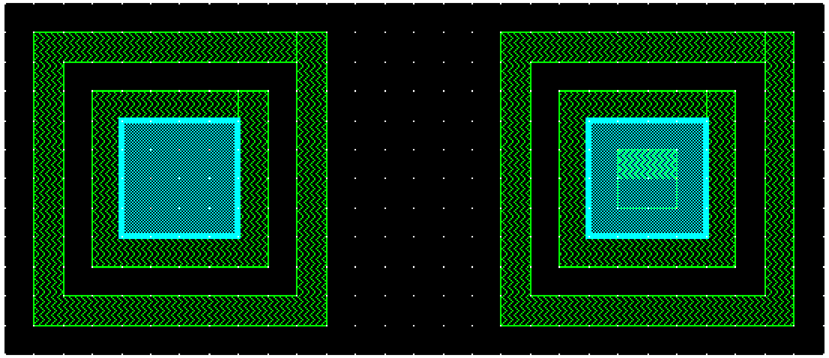
Holes (Cont.)



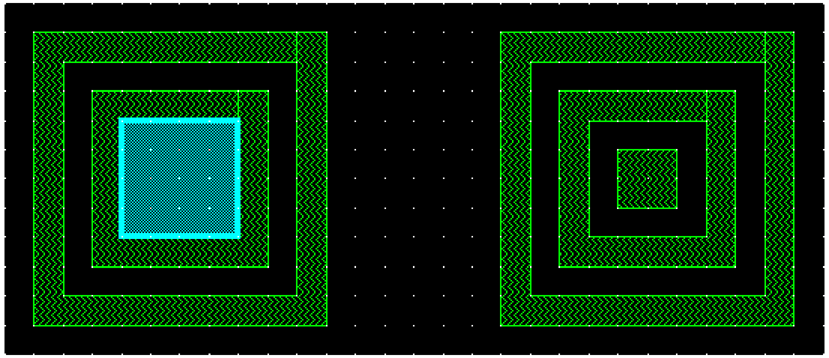
Original GDS



`HOLES L1`



`HOLES L1 INNER`



`HOLES L1 INNER EMPTY`

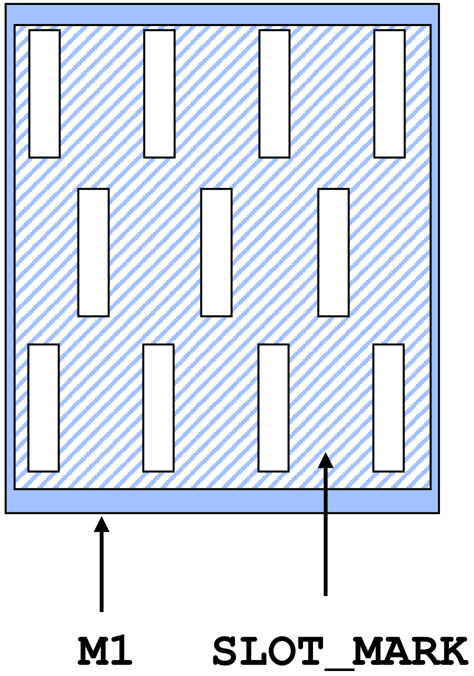
HOLES Practical Example

- Identifying metal slots can be time consuming.
- Use the HOLES command to correctly identify M1 holes:

```
X = HOLES M1 INNER EMPTY
```

- Use marker layer to distinguish slots:

```
LAYER SLOT_MARK 77
SLOTS = X AND SLOT_MARK
```



Enclose

Purpose: Selects all ***layer1*** polygons that contain ***layer2*** polygons

Syntax:

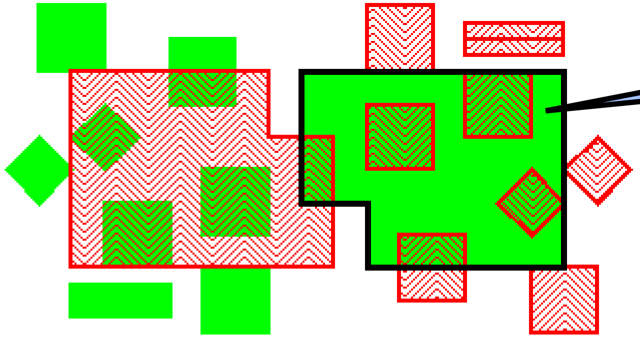
[NOT] ENCLOSE *layer1 layer2* [*constraint*[BY NET]]

Parameters:

layer1, layer2 — original or derived polygon layers

constraint — integer value or range; default is >= 1 applies to ***layer2*** polygon count

BY NET — specifies a ***layer1*** polygon is selected when the number of distinct nets in the set of ***layer2*** polygons, enclosed by the ***layer1*** polygon, meets the specified constraint; the connectivity of ***layer2*** is required



Selected ***layer1*** polygon with a count of 3

```
L1 ENCLOSE L2 == 3
//ENCLOSED L2 SHAPES SHARING EDGE
//SEGMENTS OR SINGULARITES WITH L1
//ARE INCLUDED
```

L1 🟩

L2 ▨

STATEMENT OVERVIEW

Inside

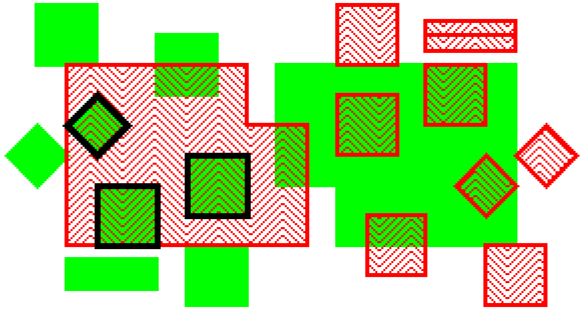
CLASS: OPERATION

Purpose: Selects all ***layer1*** polygons lying inside ***layer2*** polygons

Syntax: `[NOT] INSIDE layer1 layer2`

Parameters: ***layer1, layer2*** — original or derived polygon layers

Example:



L1 🟩

L2 🟥

```
L1 INSIDE L2
//ENCLOSED L1 SHAPES SHARING EDGE
//SEGMENTS OR SINGULARITES WITH L2
//ARE INCLUDED
```

Outside

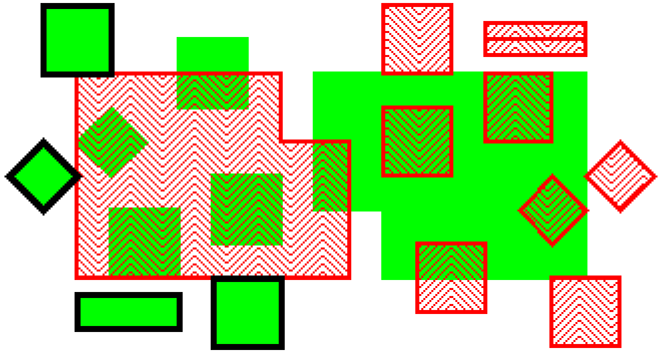
CLASS:
OPERATION

Purpose: Selects all ***`layer1`*** polygons lying outside ***`layer2`*** polygons

Syntax: `[NOT] OUTSIDE` ***`layer1 layer2`***

Parameters: ***`layer1, layer2`*** — original or derived polygon layers

Example:



L1

🟩

L2

🟥

```
L1 OUTSIDE L2
//EXTERIOR L1 SHAPES SHARING EDGE
//SEGMENTS OR SINGULARITES WITH
//L2 ARE INCLUDED
```

Cut

Purpose: Selects all ***layer1*** polygons that share only a portion of their area with ***layer2*** polygons

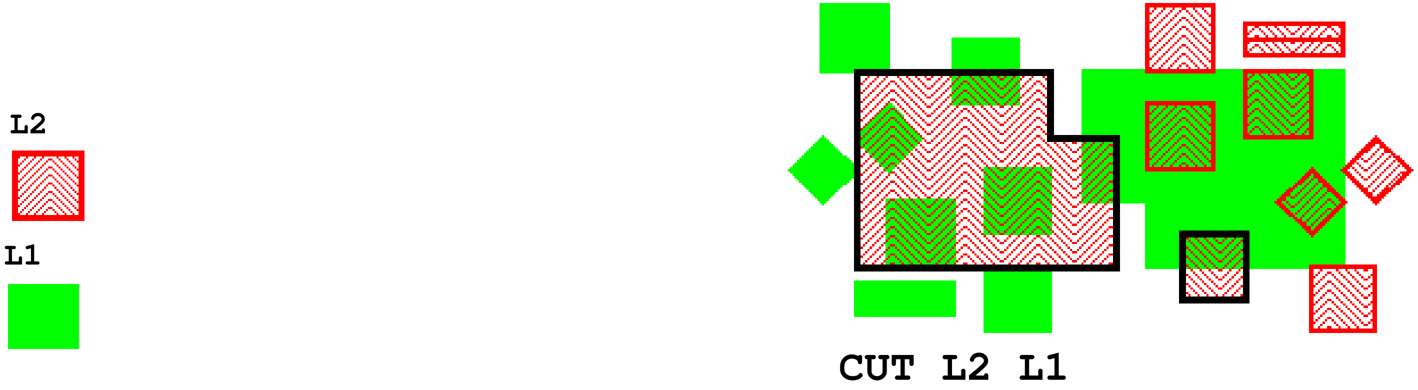
Syntax: `[NOT] CUT layer1 layer2 [constraint[BY NET]]`

Parameters:

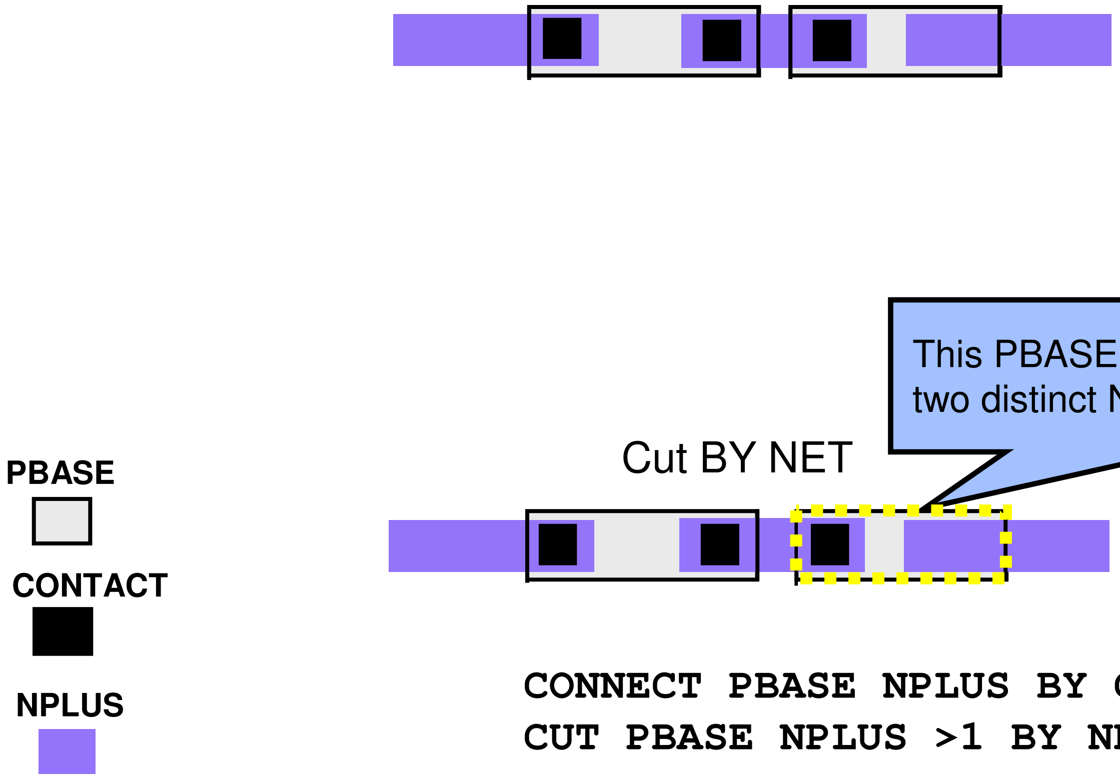
layer1, layer2 — original or derived polygon layers

constraint — specifies the number of ***layer2*** polygons or nets that a ***layer1*** polygon must share some (but not all), of its area with to be selected by the **CUT** operation; must be non-negative integers

BY NET — selects a ***layer1*** polygon when the number of distinct nets in ***layer2*** sharing only a portion of their area with ***layer1*** meets the ***constraint***



Cut BY NET — Example



Cut BY NET

This PBASE is selected because it cuts two distinct NPLUS nets.

```
CONNECT PBASE NPLUS BY CONTACT
CUT PBASE NPLUS >1 BY NET
```

Note: The syntax and functionality of the `‘CONNECT’` statement will be covered in Module 6.

Interact

Purpose: Selects all ***`layer1`*** polygons that share more than one point in common with ***`layer2`*** polygons

Syntax:

```
[NOT] INTERACT layer1 layer2
          [constraint[BY NET]]
          [SINGULAR {ALSO|ONLY}]
```

Parameters:

`layer1, layer2` — original or derived polygon layers

`constraint` — limits the selection of ***`layer1`*** polygons according to the number of ***`layer2`*** polygons or nets with which the interaction occurs; the ***`constraint`*** should contain positive integers

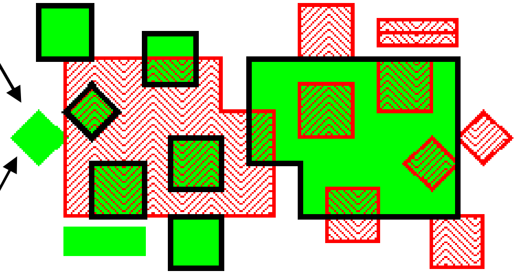
`BY NET` — specifies the selection of a ***`layer1`*** polygon is based upon the number of ***`layer2`*** nets, not polygons that interact with the ***`layer1`*** polygon

`SINGULAR ALSO` — include points of singularity

`SINGULAR ONLY` — only report points of singularity

Interact Example

Point of singularity included with **`SINGULAR ALSO`**



`INTERACT L1 L2`

Point of singularity will be only polygon selected with **`SINGULAR ONLY`**

L1 🟩

L2 🟥

Touch

Purpose: Selects all ***layer1*** polygons that lie outside ***layer2*** polygons and share a complete or partial edge

Syntax:

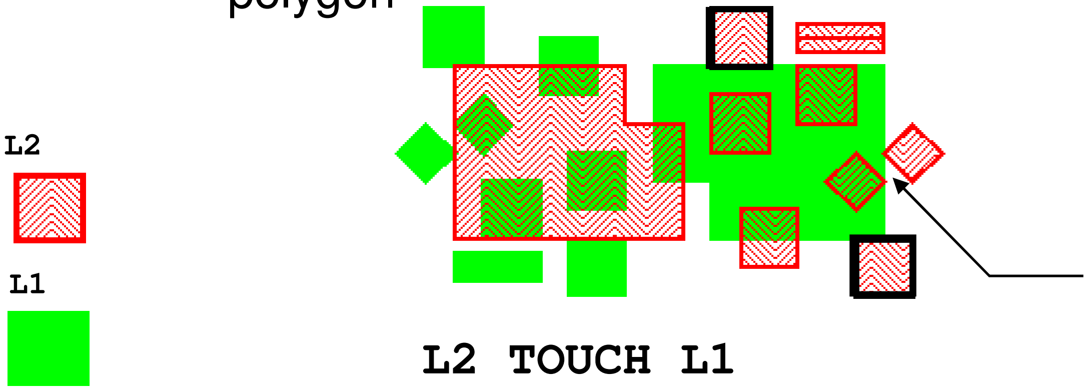
`[NOT] TOUCH layer1 layer2 [constraint][BY NET]`

Parameters:

layer1, layer2 — original or derived polygon layer

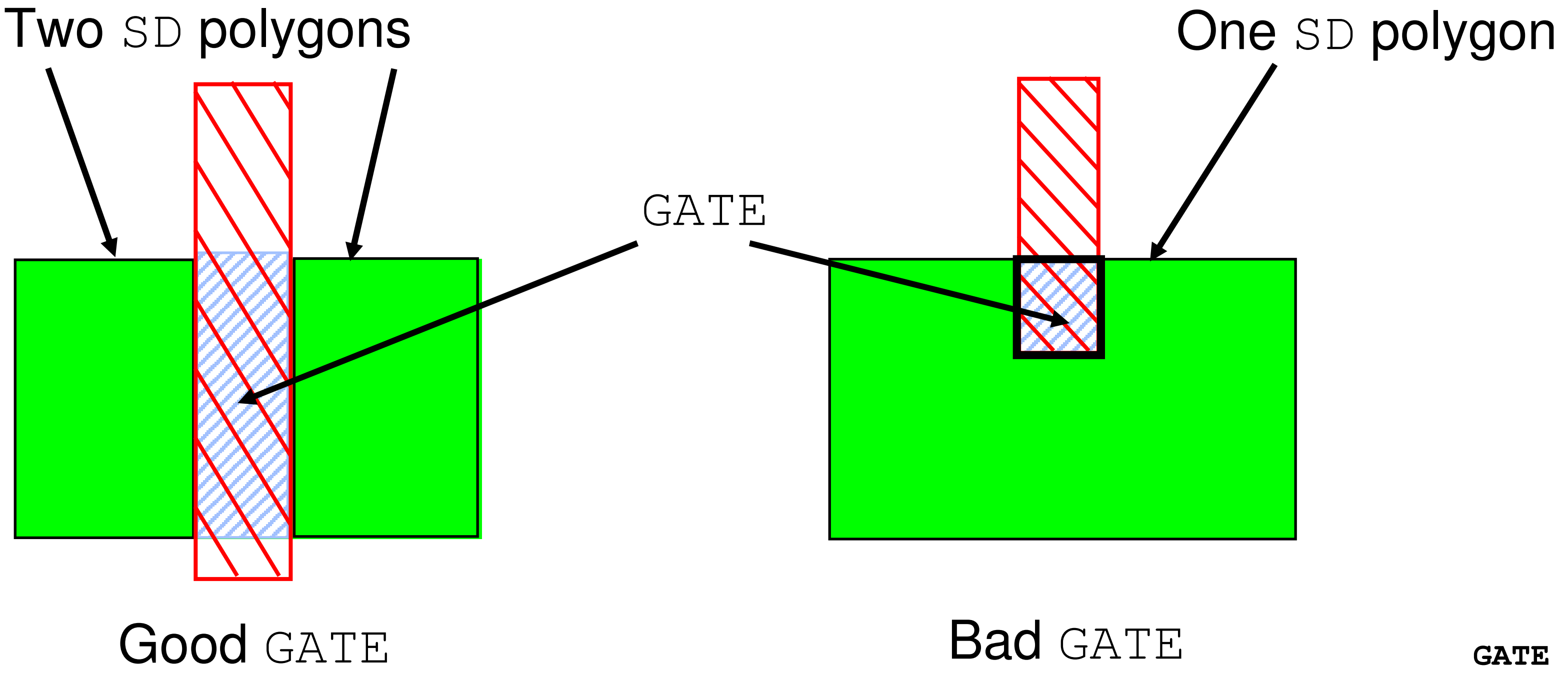
constraint — specifies the number of ***layer2*** polygons or nets a ***layer1*** polygon must touch in order to be selected

BY NET — specifies the selection of a ***layer1*** polygon is based upon the number of ***layer2*** nets, not polygons, that touch the ***layer1*** polygon



Points of singularity do not satisfy the **TOUCH** criteria.

Touch (Cont.)



```
SD = DIFF NOT POLY
GATE = POLY AND DIFF

SHORT_GATE{
       @GATE NOT CROSSING DIFF
       GATE TOUCH SD < 2
}
```

GATE ▨

POLY ▨

DIFF 🟩

STATEMENT OVERVIEW

With Edge

CLASS: OPERATION

Purpose: Selects all ***layer1*** polygons having complete or partial edges coincident with edges on ***layer2***.

Syntax: `[NOT] WITH EDGE layer1 layer2 [constraint]`

Parameters:

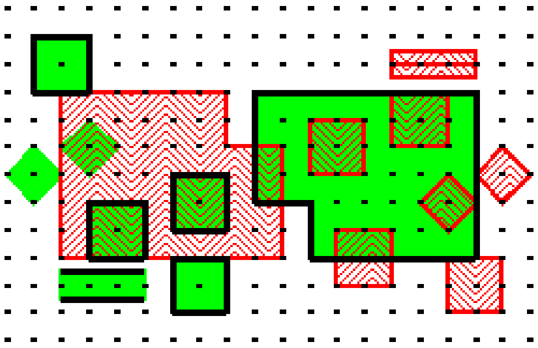
layer1 — original or derived polygon layers

layer2 — derived edge layer

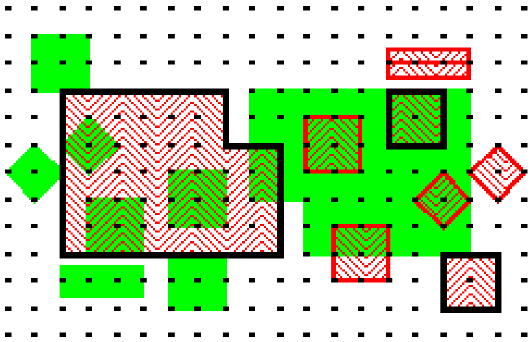
constraint — ***layer2*** edge or edge segment count; default is >= 1

L2 ▨

L1 🟩



X = L1 LENGTH >= 2

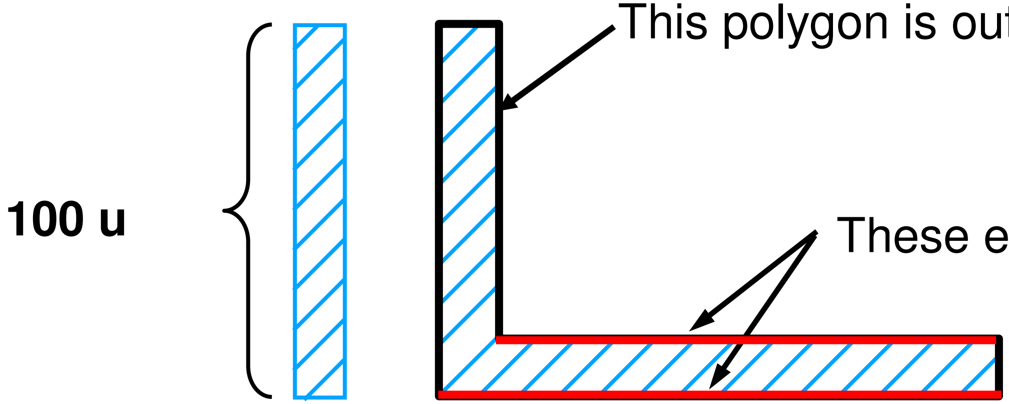


L2 WITH EDGE X

With Edge (Cont.)

Example:

```
//FIND ALL METAL1 HAVING AT LEAST ONE EDGE > 100u
LONG_METAL1{
    X = METAL1 LENGTH > 100
    METAL1 WITH EDGE X      //LONG METAL1
}
```



This polygon is output by rule **`LONG_METAL1`**

These edges belong to **`X`**

Area

CLASS:
OPERATION

Purpose: Selects all input polygons having areas satisfying the specified constraint

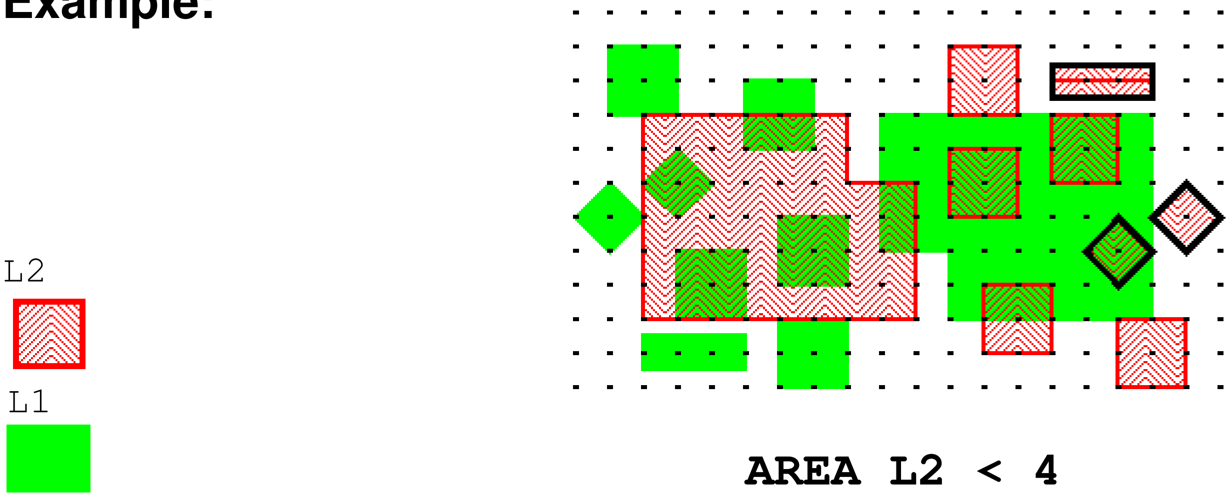
Syntax: `[NOT] AREA` *`layer1 constraint`*

Parameters:

`layer1` — original or derived polygon layers

`constraint` — real value or range of area

Example:



`AREA L2 < 4`

STATEMENT OVERVIEW

Perimeter

CLASS: OPERATION

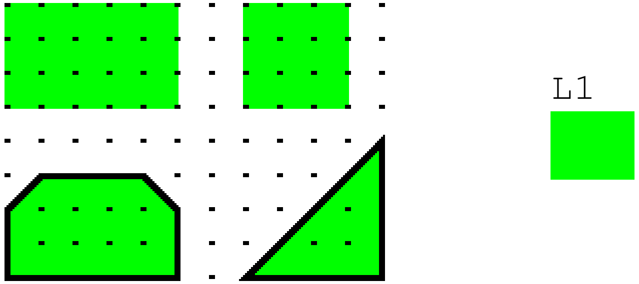
Purpose: Selects all input polygons having perimeters conforming to the constraint

Syntax: `PERIMETER` *`layer constraint`*

Parameters: *`layer`* — original or derived polygon layer

`constraint` — real value or range for perimeter

Example:



`PERIMETER L1 >12 <16`

◆ **It is generally good practice to specify a range for the *`constraint`* when dealing with non-rectangular polygons.**

STATEMENT OVERVIEW

Rectangle

CLASS: OPERATION

Purpose: Selects all input polygons that are rectangles having sides satisfying the constraints.

Syntax:

```
[NOT] RECTANGLE layer [constraint1 [BY constraint2]]
     [ASPECT constraint3]
     [ORTHOGONAL ONLY|MEASURE EXTENTS]
```

Parameters:

`layer` — original layer or a derived polygon layer

`constraint1` — one edge pair must satisfy the ***`constraint`*** (edge length)

`BY` ***`constraint2`*** — pertains to the pair of sides not handled by ***`constraint1`***

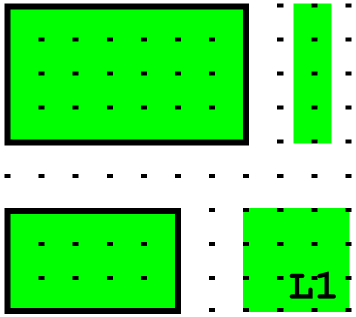
`ASPECT` ***`constraint3`*** — specifies the ratio of the longer side to the shorter side

`ORTHOGONAL ONLY` — sides must be parallel to the coordinate axes

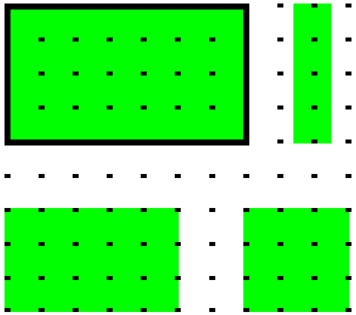
`MEASURE EXTENTS` — selects polygons that fit within a specific rectangular extent

Rectangle (Cont.)

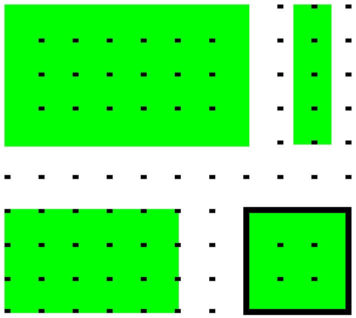
Examples:



`RECTANGLE L1 > 4`



`RECTANGLE L1 >=4 BY >=7`



`RECTANGLE L1 ==3 ASPECT ==1`

Rectangles

Purpose: Creates an array of rectangles of specified dimensions and spacing; often used in planarization and layer fill applications

Syntax:

```
RECTANGLES width length
    {spacing | {width_spacing length_spacing}}
    [OFFSET {offset | {width_offset length_offset}}]
    [{INSIDE OF x1 y1 x2 y2}|{INSIDE OF LAYER layer}]
    [MAINTAIN SPACING]
```

Parameters:

width length — a pair of numbers that indicate the ***width*** (x-axis) and ***length*** (y-axis) of a rectangle, in user units

spacing — a number that indicates the spacing in user units, in both the x- and y- directions, between rectangles

width_spacing length_spacing — a pair of numbers that indicate the width spacing (x-axis) and length spacing (y-axis) between rectangles

Rectangles (Cont.)

Parameters (Cont.):

`OFFSET` — specifies the horizontal and vertical offsets between adjacent rectangles

`offset` — a number specifying both the x-axis and y-axis offsets between rectangles

`width_offset length_offset` — a pair of numbers indicating the x-axis and y-axis offsets between rectangles

`INSIDE OF` ***`x1 y1 x2 y2`*** — specifies an area to be filled with rectangles; indicating the lower-left (***`x1`***, ***`y1`***) and upper right (***`x2`***, ***`y2`***) corners of the extent to be filled

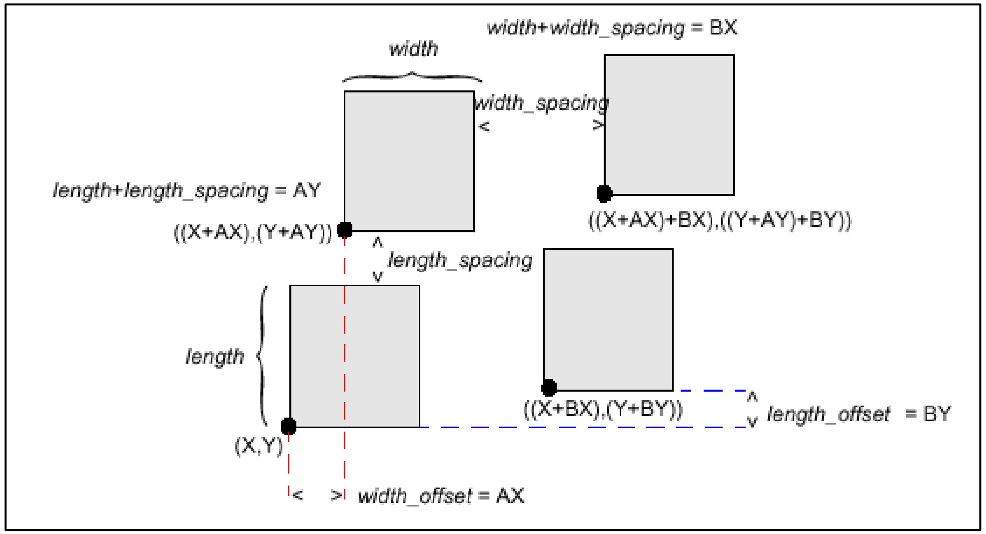
`INSIDE OF LAYER` ***`layer`*** — fill the extent of the specified layer with rectangles

`MAINTAIN SPACING` — controls the spacing of rectangles, so a halo area is constructed around each rectangle, where no other rectangle may fall within the ***`spacing`***, or ***`width_spacing`*** and ***`length_spacing`***

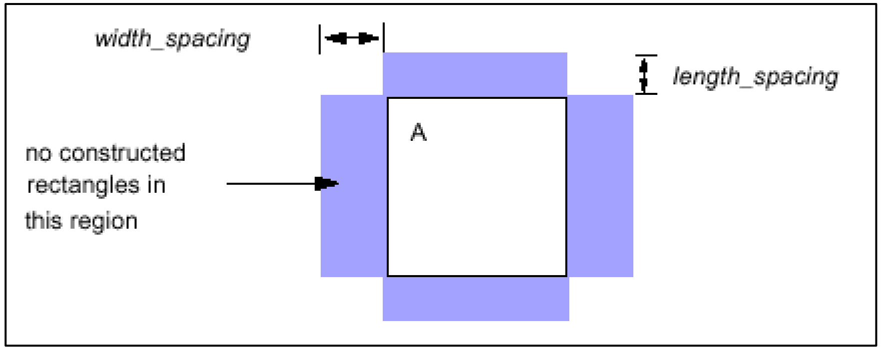
Rectangles (Cont.)

- Rectangles will begin the fill pattern from the lower left corner of the database extent by default.
  - If using `INSIDE` or `INSIDE OF LAYER`, the fill pattern begins at the lower left corner of the specified box in the former case and at the lower left corner of the layer extent in the latter.
  - Partial rectangles will not be output.
- Spacing and offset proceed from bottom left to top right of the area to be filled.
  - All spacing and offset calculations are based upon the location of the lower left corner of the previous rectangle in the placement sequence.

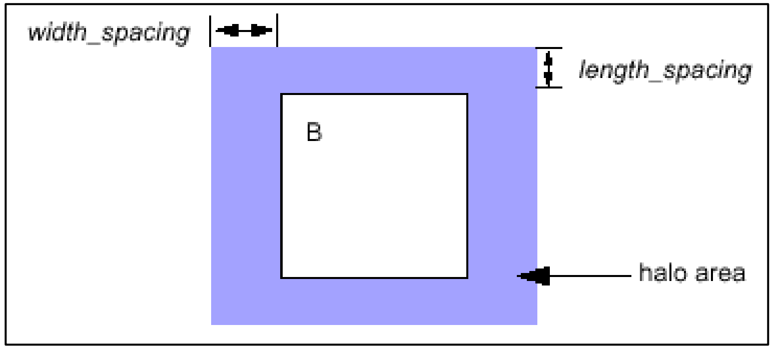
Rectangles (Cont.)



Rectangle Placement

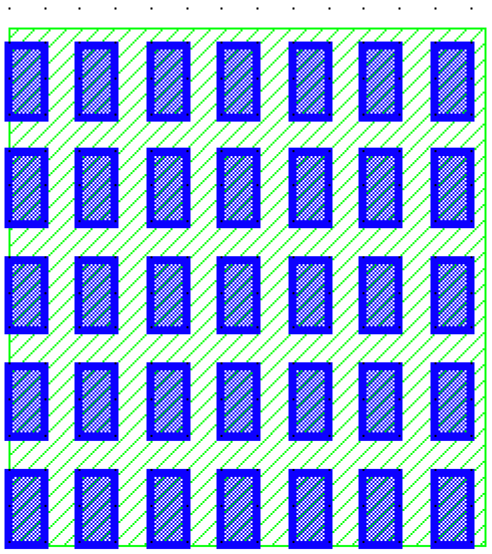


Default Spacing

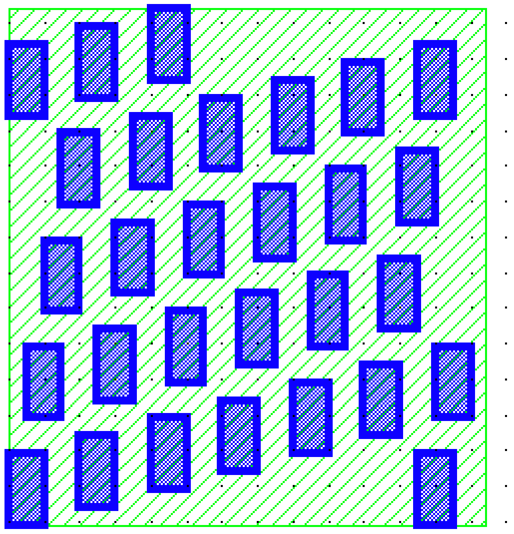


Maintain Spacing

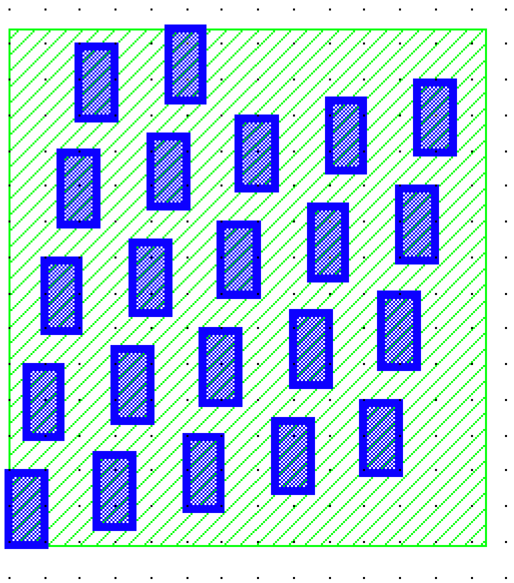
Examples of Rectangles



```
PLANAR_FILL {
  RECTANGLES 1 2 1
  INSIDE OF LAYER L1
}
```



```
PLANAR_FILL_OFFSET {
  RECTANGLES 1 2 1
  OFFSET 0.5
  INSIDE OF LAYER L1
}
```



```
PLANAR_FILL_MAINTAIN {
  RECTANGLES 1 2 1
  OFFSET 0.5
  INSIDE OF LAYER L1
  MAINTAIN SPACING
}
```

NOTE: Use DRC CHECK MAP to output the results

Vertex

Purpose: Selects all layer polygons having vertex (or edge) counts conforming to the constraint

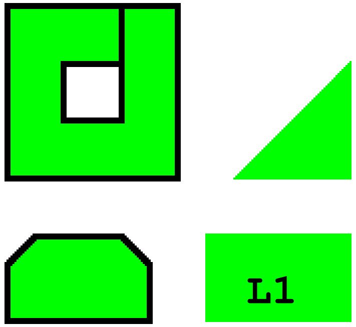
Syntax: **`VERTEX`** ***`layer constraint`***

Parameters:

`layer` — original or derived polygon layer

`constraint` — specification of integer value or range of values

Example:



`L1 VERTEX > 4`

- **For any polygon, vertex count = edge count.**

Layer Modifiers

The following operations construct new layers based upon existing geometry by modifying their size, location and orientation and so on:

- `EXPAND EDGE`
- `SIZE`
- `WITH WIDTH`
- `EXPAND TEXT`
- `GROW`
- `SHRINK`

Expand Edge

Purpose: Expands input polygon edges into rectangles

Syntax: `EXPAND EDGE` ***`layer expansion_set`***
`[EXTEND BY [FACTOR]` ***`number`***`] [CORNER FILL]`

Parameters:

`layer` — original or a derived polygon or edge layer

`expansion_set` — Converts all edges of ***`layer`*** into rectangles by expanding the edges in the direction specified.

Keywords:

- **`INSIDE BY`** ***`value`*** — expands edges toward the inside of input polygons by ***`value`*** user units
- **`INSIDE BY FACTOR`** ***`factor`*** — expands edges toward the inside of input polygons by ***`factor`*** multiplied by edge length
- **`OUTSIDE BY`** ***`value`*** — similar to **`INSIDE BY`** ***`value`*** except toward the outside
- **`OUTSIDE BY FACTOR`** ***`factor`*** — similar to **`INSIDE BY FACTOR`** ***`factor`*** except outside
- **`BY`** ***`value`*** — does both **`INSIDE BY`** ***`value`*** and **`OUTSIDE BY`** ***`value`***
- **`BY FACTOR`** ***`factor`*** — does both **`INSIDE BY FACTOR`** ***`factor`*** and **`OUTSIDE BY FACTOR`** ***`factor`***

Expand Edge (Cont.)

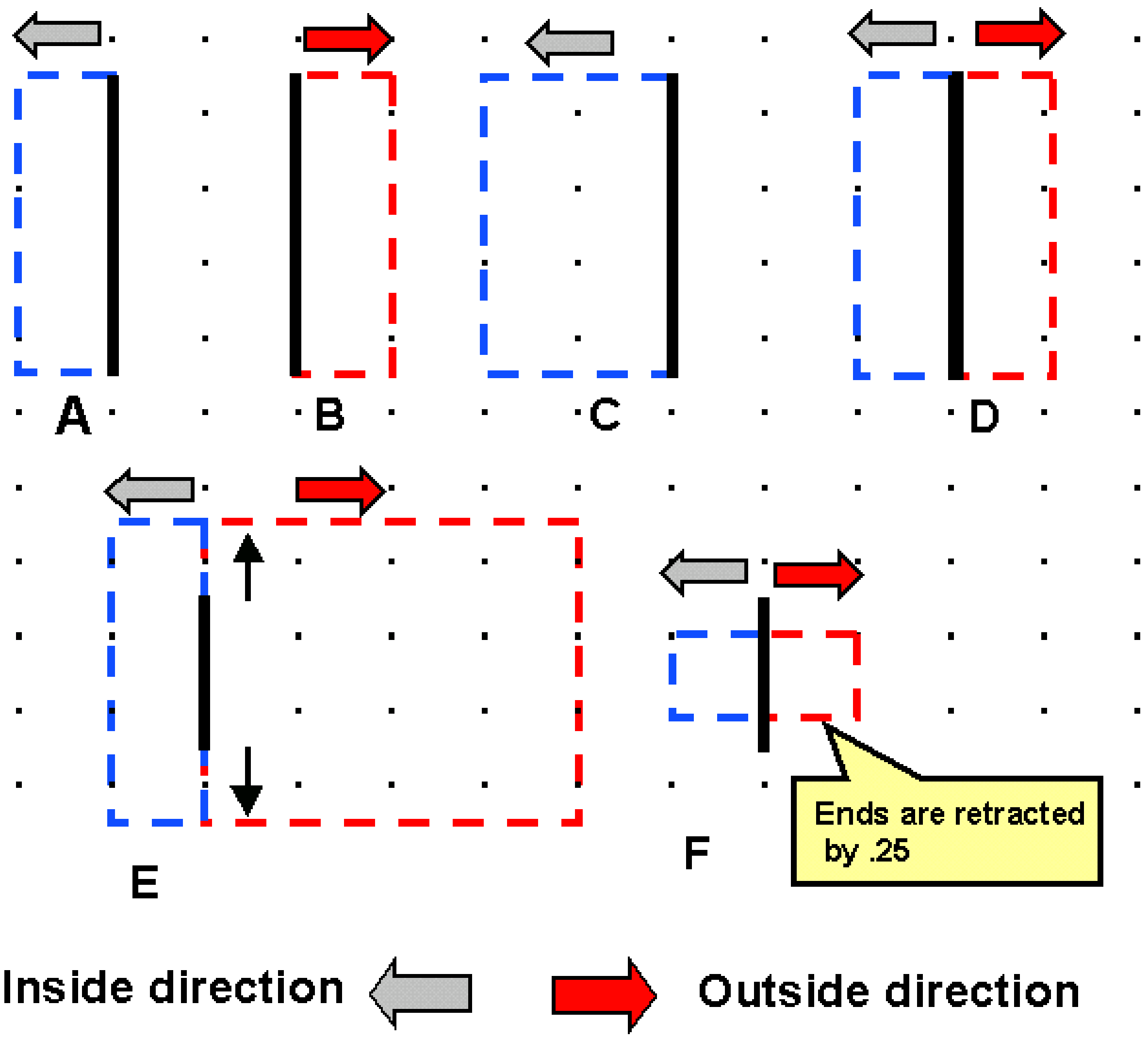
Parameters (Cont.):

`EXTEND BY` ***`number`*** — extends or retracts edges by ***`number`*** in user units before expanding them (***`number`*** < 0 retracts)

`EXTEND BY FACTOR` ***`number`*** — extends or retracts edges by ***`number`*** times edge length before expanding them

`CORNER FILL` — directs an **`EXPAND EDGE`** operation to fill gaps between rectangles formed by the operation at the corners of the input layer (Only fills corners pointing in the direction of the expansion)

Expand Edge (Cont.)



Ends are retracted by .25

```
EXPAND EDGE
E_LAYER ...
```

A: `INSIDE BY 1`

B: `OUTSIDE BY 1`

C: `INSIDE BY FACTOR .5`

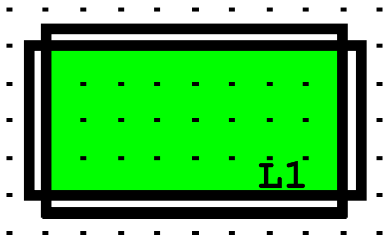
D: `BY FACTOR .25`

E: `INSIDE BY 1 OUTSIDE BY FACTOR 1 EXTEND BY 1`

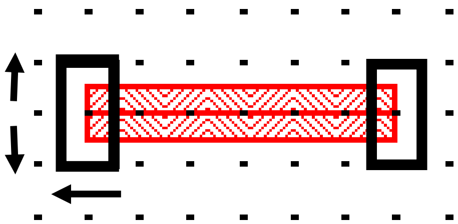
F: `BY FACTOR 1 EXTEND BY FACTOR -.25`

Expand Edge (Cont.)

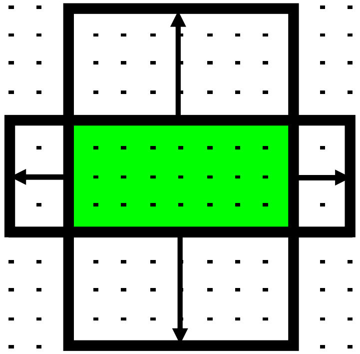
Examples:



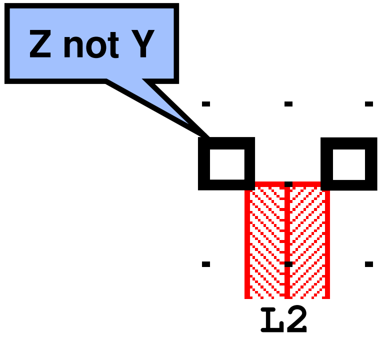
```
EXPAND EDGE L1 OUTSIDE
BY .5
```



```
SIMPLE_HAMMERHEAD{
   X = LENGTH L2 == 1
   EXPAND EDGE X BY .5
   EXTEND BY .5
}
```



```
EXPAND EDGE L1 OUTSIDE
BY FACTOR .5
```

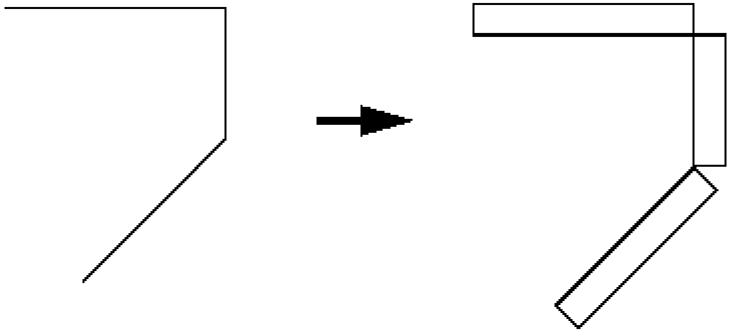


```
MICKEY_MOUSE{
  X = LENGTH L2 == 1
  Y = EXPAND EDGE X
    OUTSIDE BY .5
  Z = EXPAND EDGE X
    OUTSIDE BY .5
    EXTEND BY .5
  Z NOT Y
}
```

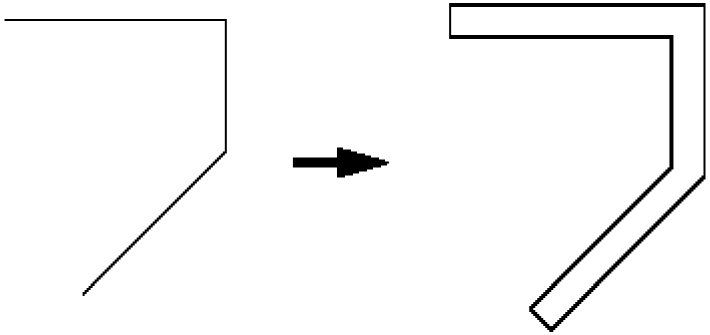
Expand Edge (Cont.)

Corner Fill Example:

```
EXPAND EDGE layer OUTSIDE BY 1
```



```
EXPAND EDGE layer OUTSIDE BY 1 CORNER FILL
```



Size

Purpose: Expands or shrinks input polygons by a specified value

Syntax:

```
SIZE layer1 BY size_value [TRUNCATE distance]
     [OVERLAP ONLY] | {[INSIDE OF | OUTSIDE OF] layer2}
     [STEP step_value]
SIZE layer1 BY size_value [TRUNCATE distance]
     [OVERUNDER | UNDEROVER]
```

Parameters:

`layer1` — original layer or derived polygon layer

`BY` ***`size_value`*** — specify how much to expand or shrink polygons

`TRUNCATE` ***`distance`*** — specifies the spike truncation distance; the default value of distance is 1/cos 67.5 (approximately 2.61)

Size (Cont.)

Parameters (Cont.):

`OVERLAP ONLY` — specifying that the output consists only of regions where the oversized polygons overlap (not the oversized polygons themselves); ***`size_value`*** must be greater than zero

`INSIDE OF` ***`layer2`*** — causes ***`layer1`*** to expand inside of ***`layer2`***

`OUTSIDE OF` ***`layer2`*** — causes ***`layer1`*** to expand outside of ***`layer2`***

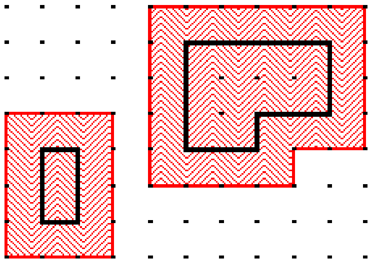
`STEP` ***`step_value`*** — specifies the incremental bloating or shrinking; polygons are grown or shrunk by ***`step_value`*** repeatedly until the ***`size_value`*** is met.

`OVERUNDER` — instructs Calibre to perform two `SIZE` operations; ***`layer1`*** is first increased in size, then decreased in size, based on ***`size_value`***.

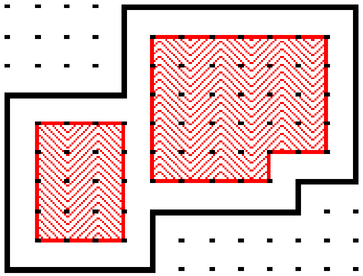
`UNDEROVER` — instructs Calibre to perform two `SIZE` operations; ***`layer1`*** is first decreased in size, then increased in size, based on ***`size_value.`***

Size (Cont.)

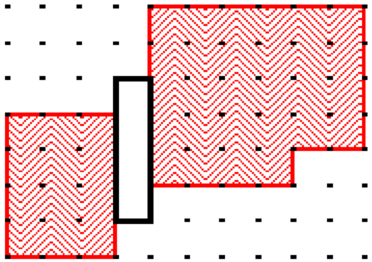
Examples:



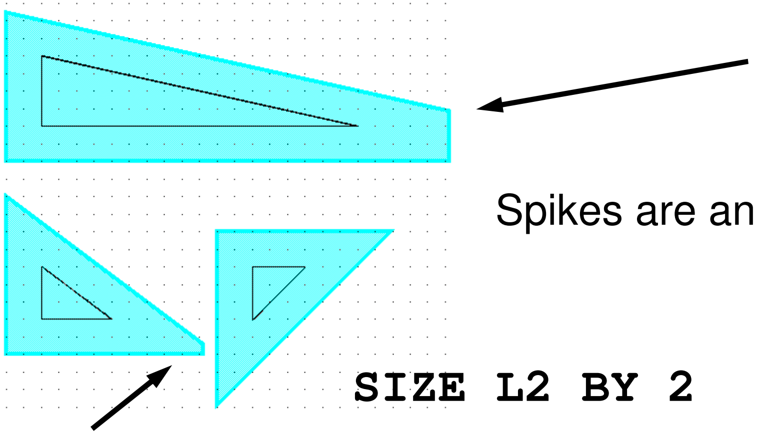
`SIZE L2 BY -1`



`SIZE L2 BY 1`



`SIZE L2 BY 1`
`OVERLAP ONLY`



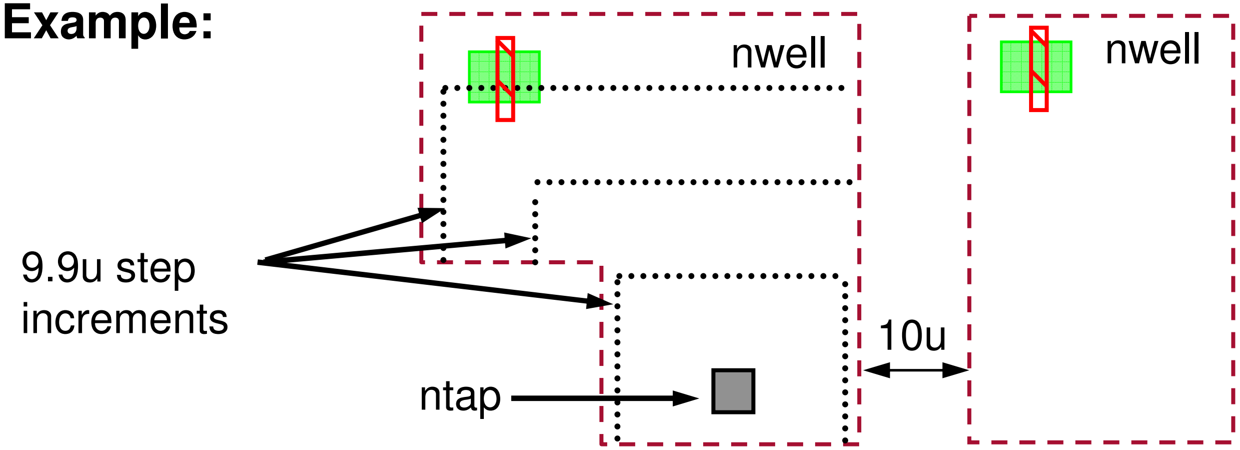
Spikes are angles <45º

`SIZE L2 BY 2`

By default, polygon spikes are truncated by Size at a distance of 1/cosine 67.5° * ***`size_value`***

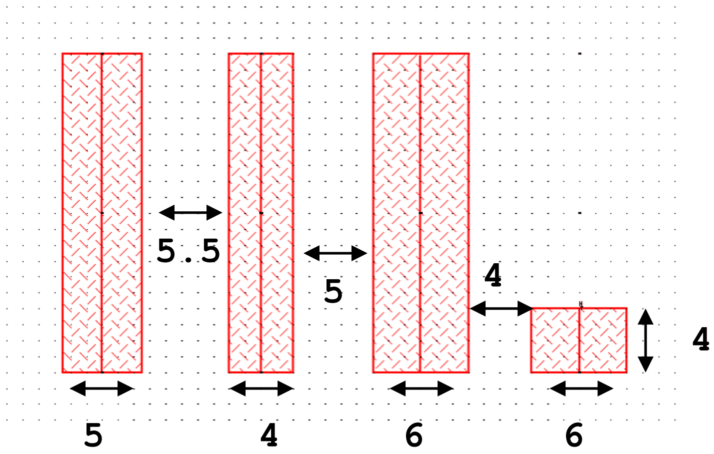
Size (Cont.)

Example:

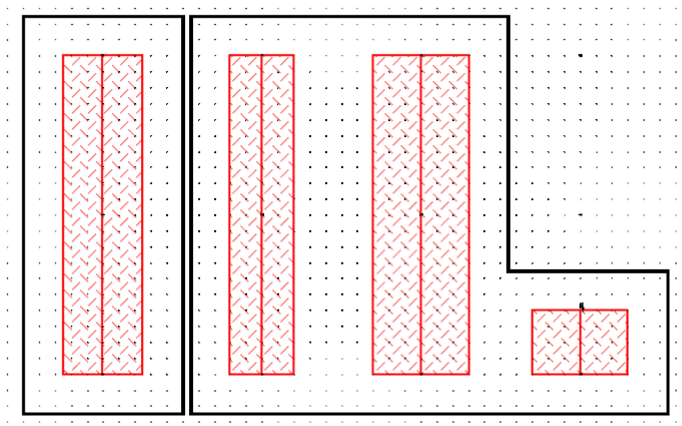


```
//VERIFY MAX DISTANCE BETWEEN NTAP AND GATES IS 100u
//MIN NWELL SPACING IS 10u, STEP MUST BE <10u
WORM_RULE{
   X = SIZE NTAP BY 100 INSIDE OF NWELL STEP 9.9
   //GROW 100u REGIONS INSIDE NWELLS IN 9.9u INCREMENTS
   GATE NOT X
   //FLAG GATES OUTSIDE 100u FROM NTAPS
}
```

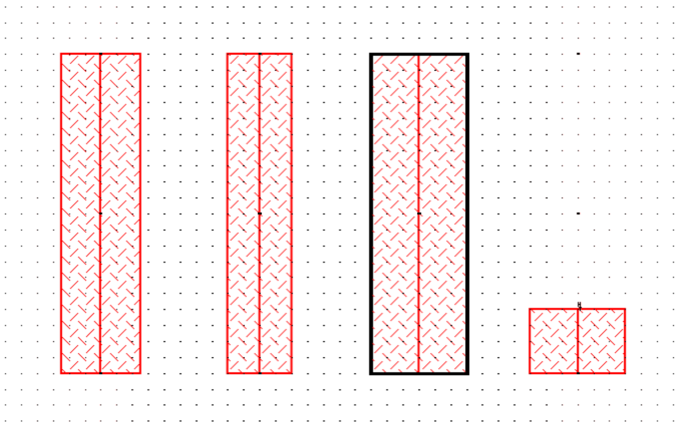
Size (Cont.)



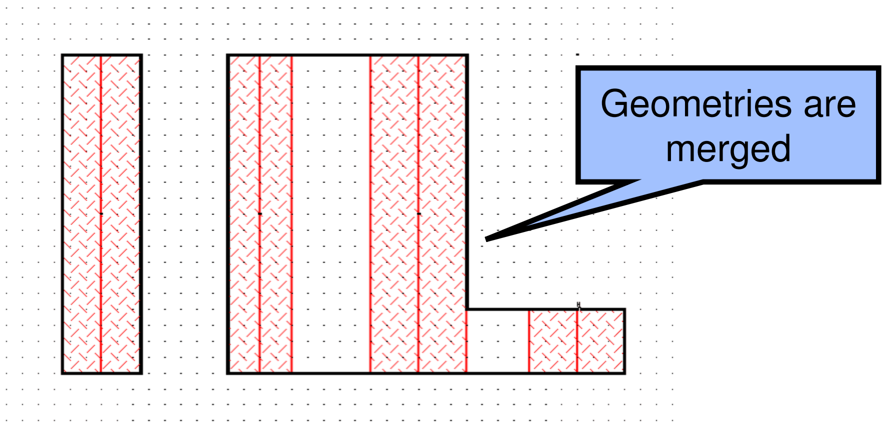
Original Data



SIZE L2 BY 2.5



SIZE L2 BY 2.5 UNDEROVER



Geometries are merged

SIZE L2 BY 2.5 OVERUNDER

With Width

Purpose: Selects just portions of polygons that satisfy width constraints; returns shapes

Syntax: `[NOT] WITH WIDTH` ***`layer constraint`***

Parameters:

`layer` — original layer or a derived polygon layer

`constraint` — interpreted as width in user units (>=0)

Example:

```
//derive polysilicon having width <= .10
narrow_poly = poly with width <= .10
```

Grow

Purpose: Enables expansion of polygon edges in the specified directions

Syntax:

```
GROW layer [RIGHT BY value][TOP BY value]
     [LEFT BY value][BOTTOM BY value]
```

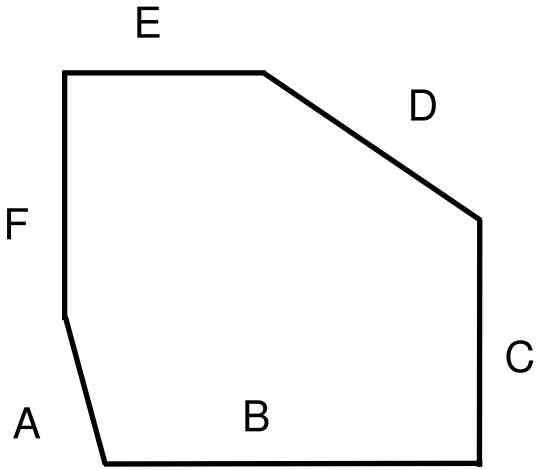
Parameters:

`layer` — a required original, derived polygon, or derived edge layer

`RIGHT, TOP, LEFT, BOTTOM` — orthogonal edge of input layer

`BY` ***`value`*** — amount of outside expansion

Use this operation with care in hierarchical designs.



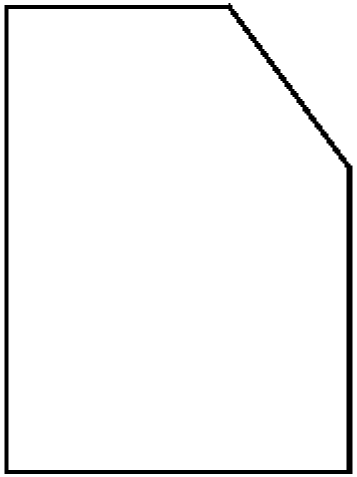
Edge Classification:

A = Left and Bottom
B = Bottom
C = Right
D = Top and Right
E = Top
F = Left

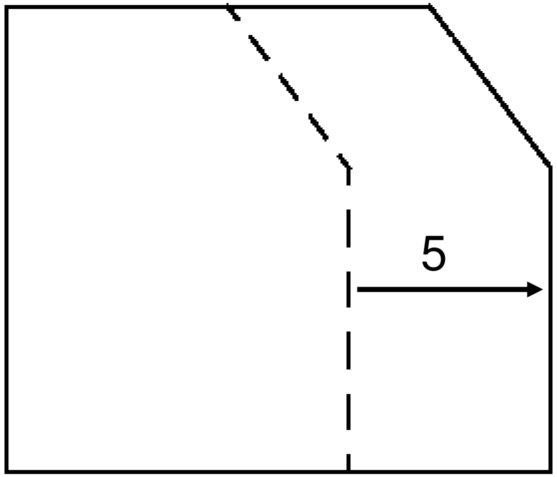
Grow (Cont.)

Grow Example:

`GROW layerA RIGHT BY 5`



Original polygon
on layerA



Derived polygon after
Grow operation

Shrink

Purpose: Contracts edges toward a polygon’s interior in the specified directions

Syntax:

```
SHRINK layer [RIGHT BY value] [TOP BY value]
      [LEFT BY value] [BOTTOM BY value]
```

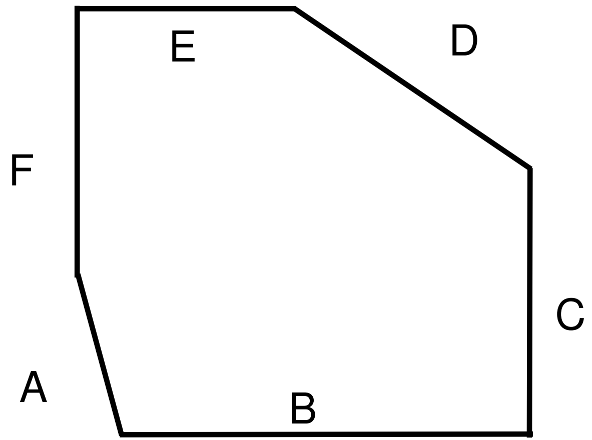
Parameters:

`layer` — original polygon layer

`RIGHT, TOP, LEFT, BOTTOM` — orthogonal edge of input layer

`BY` ***`value`*** — amount of inside contraction

Use this operation with care when you are also rotating the polygons. (Same as **`GROW`**.)



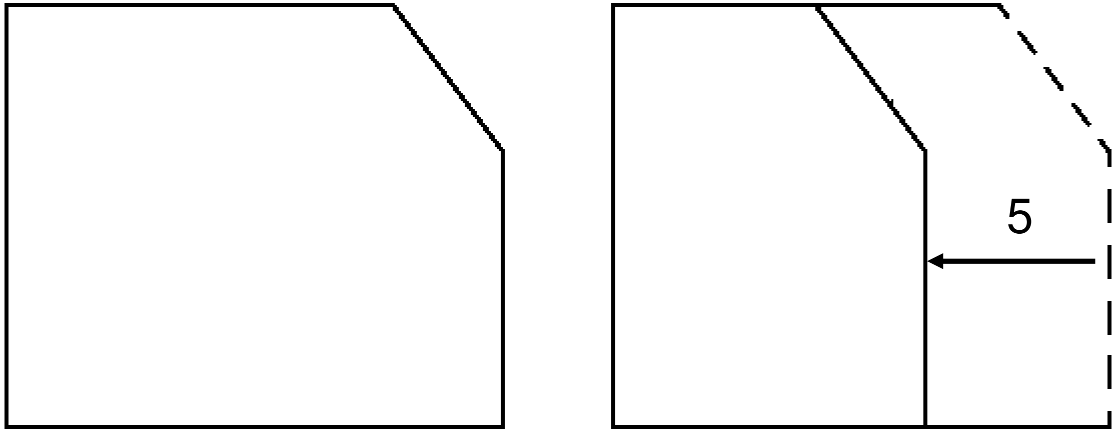
Edge Classification:

A = Left and Bottom
B = Bottom
C = Right
D = Top and Right
E = Top
F = Left

Shrink (Cont.)

Shrink Example:

```
SHRINK layerA RIGHT BY 5
```



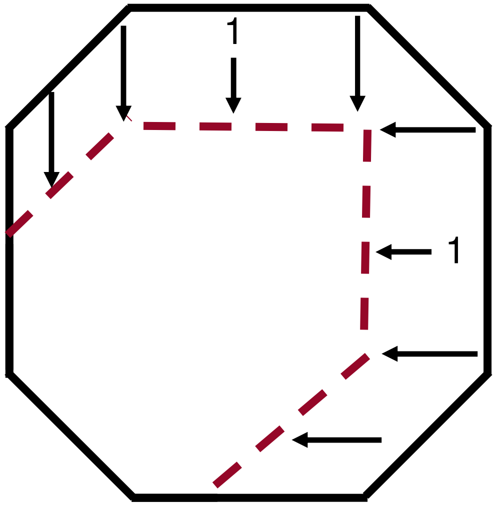
Original polygon
on layerA

Derived polygon after
Shrink operation

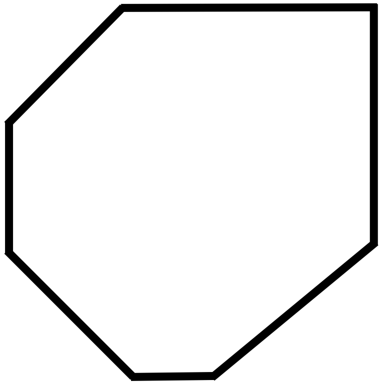
Shrink (Cont.)

Shrink Example:

```
SHRINK layerA RIGHT BY 1 TOP BY 1
```



Original polygon on layerA



Derived polygon layer after Shrink operation

STATEMENT OVERVIEW

Expand Text

CLASS: OPERATION

Purpose: Creates a derived polygon layer consisting of merged squares centered on the positions of text objects having the specified ***`text_name`***; the squares have edge length of ***`number`*****.**

Syntax:

```
EXPAND TEXT text_name [text_layer] BY number
    [PRIMARY ONLY]
```

Parameters:

- ***`text_name`*** — name of a text object; can contain one or more question mark (?) wildcard characters, where the (?) matches zero or more characters
- ***`text_layer`*** — original layer containing the ***`text_name`***; use to prevent ambiguity in selecting text objects having the same name but appearing in different layers
- **`BY`** ***`number`*** — specifies the size of marker squares

Expand Text (Cont.)

Parameters (Cont.):

`PRIMARY ONLY` — specifies that the operation only uses top-cell text

Example:

```
Rule1{
   x = EXPAND TEXT VDD text_layer BY 2
   metal1 INTERACT x}

//Place 2 x 2 markers on VDD text locations on text_layer
//Find all metal1 shapes that interact with VDD text
```

Miscellaneous Layer Operations

The following set of operations have varied applications including: copying, checking density, and finding the boundaries of specified layers:

- `COPY`
- `EXTENT`
- `EXTENTS`
- `EXTENT CELL`
- `DENSITY`
- `WITH TEXT`
- `NET`

Copy

Purpose: Copies *`layer1`* polygons to a derived layer

Syntax: `COPY` *`layer1`*

Parameters: *`layer1`* — original or derived polygon or edge layer

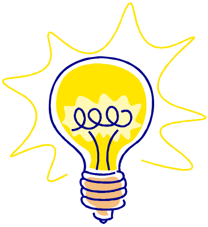
Example:

```
METAL_LONG_SPACE{
@Spacing between metal edges longer
@than 100 um must be at least 4 um
    LONG_METAL = LENGTH metal > 100
    COPY LONG_METAL // creates a derived layer
    //EXTERNAL LONG_METAL < 4

}
```

Display LONG_METAL edges for debugging

- **Creates a derived layer that can be viewed in RVE.**
- **The `COPY` operation is useful in debugging RuleCheck statements and layer derivation.**



Extent

Purpose: Generates a derived polygon layer consisting of one rectangle equivalent to the boundary of the database

Syntax: **`EXTENT [layer]`**

Parameters: ***`layer`*** — original or derived polygon or edge layer

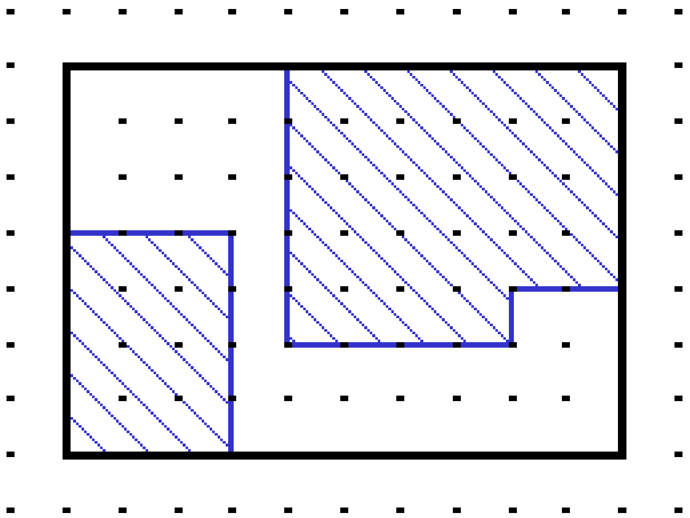
Example:

```
//FORM PWELL
BULK = EXTENT
PWELL = BULK NOT NWELL
```

- **Can be used in a layer derivation statement.**
- **When used with *`layer`*, Extent generates a layer that is the minimum bounding box of all polygons on *`layer`*.**

Extent (Cont.)

Example:



`EXTENT METAL2`

METAL2

▨

Extents

Purpose: Generates a derived polygon layer consisting of the merged minimum bounding boxes (with edges parallel to the coordinate axes) of each polygon on the input layer

Syntax: **`EXTENTS`** ***`layer`*** **`[CENTERS [`*`number`*`]]`**

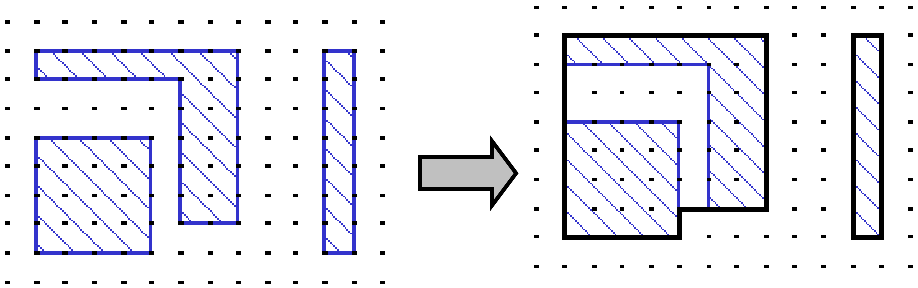
Parameters:

`layer` **— original or derived polygon layer**

`CENTERS [` ***`number`*****`]`** — generates marker squares of size ***`number`*** at the center of each bounding box instead of the boxes themselves; default marker size is 1 user unit. (Generates centers before merging extents.)

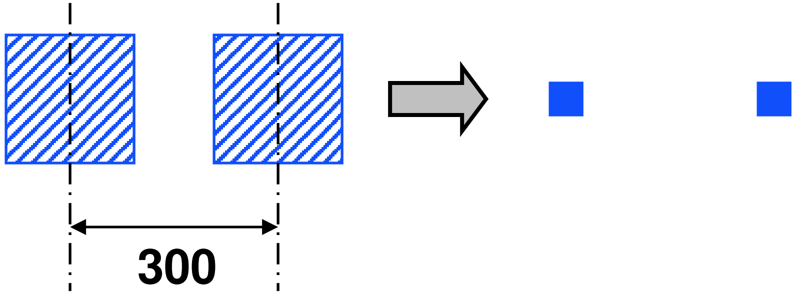
Extents Examples

Example 1:



```
m2_extents {EXTENTS metal2}
```

Example 2:



```
//Center-to-center pad distance
// must be 300 microns:

cp = EXTENTS pad CENTERS

rule {EXT cp < 299 }
 // Use 299 because centers are 1x1
```

Extent Cell

Purpose: Generates a derived polygon layer consisting of rectangles that represent the extents of cells in the given list; by default, Calibre only uses the extents of objects actually required for the run

Syntax:

`EXTENT CELL name […name] [ORIGINAL [OCCUPIED]]`

Parameters:

`name` — cell name, can be a string variable; the “ * ” wildcard is permitted with cell names in quotes

`ORIGINAL` — specifies that all objects in the layout database are used to compute the specified cell extents

`OCCUPIED` — specifies that only the cells containing geometries required in the Calibre run (including subhierarchy) have their original extents returned; all other cells are ignored

Example: **`EXTENT CELL “ALU*”`**

STATEMENT OVERVIEW

Density

CLASS: OPERATION

Purpose:

The **`DENSITY`** operation is typically used to check the area of an input layer versus the area of a data capture window moved through a user-defined grid. This operation has numerous features that control how the data capture window scans the layout, as well as the mathematical expression the operation is supposed to check. Outputs window that meets the constraint.

Syntax:

```
DENSITY layer1[…layerN][[density_expression]]constraint
    [INSIDE OF{EXTENT|x1 y1 x2 y2|LAYER layerB
          [BY EXTENT|BY POLYGON|BY RECTANGLE|
          CENTERED value]}]
    [WINDOW {wxy|wx wy}][STEP {sxy|sx sy}]]
    [TRUNCATE|BACKUP|IGNORE|WRAP]
```

… many more options

See the *SVRF Manual* for all the additional options and Secondary Keywords.

Density (Cont.)

Example 1

The density of `metal2` in every 5 × 5 area of the layout must exceed 25%:

```
met2_check {
       @ The density of metal2 in every 5x5 area of the
       @ layout must exceed 25%
       DENSITY metal2 < 0.25 WINDOW 5.0
}
```

Example 2

This example specifies a 2 user unit step size because “`3 -1`” is viewed as the arithmetic operation 3 minus 1:

```
DENSITY metal2 < 0.25 WINDOW 10.0 STEP 3 -1
```

This example results in a compilation error due to the negative y-value:

```
DENSITY metal2 < 0.25 WINDOW 10.0 STEP 3 (-1)
```

Density (Cont.)

Example 3

`Metal` density in any 100 × 100 window (stepped 50 × 50) must exceed 0.25. However, if there is `poly` present in the window, then there is no requirement on metal density.

```
density_rule_a {
        DENSITY metal poly <= 0.25 WINDOW 100 STEP 50
                [ AREA(metal) / ( !AREA(poly) * AREA() ) ]
}
```

Example 4

Same as Example 3 except, if there is `poly` present in the window, then the area of the `poly` must first be subtracted from the window area.

```
density_rule_b{
        DENSITY metal poly <= 0.25 WINDOW 100 STEP 50
        [ AREA(metal) / ( AREA() - AREA(poly) ) ]
}
```

With Text

Purpose: Selects all layer polygons intersecting the positions of the text objects having the specified name

Syntax:

`[NOT] WITH TEXT layer name [text_layer] [PRIMARY ONLY]`

Parameters:

`layer` — original layer or derived layer

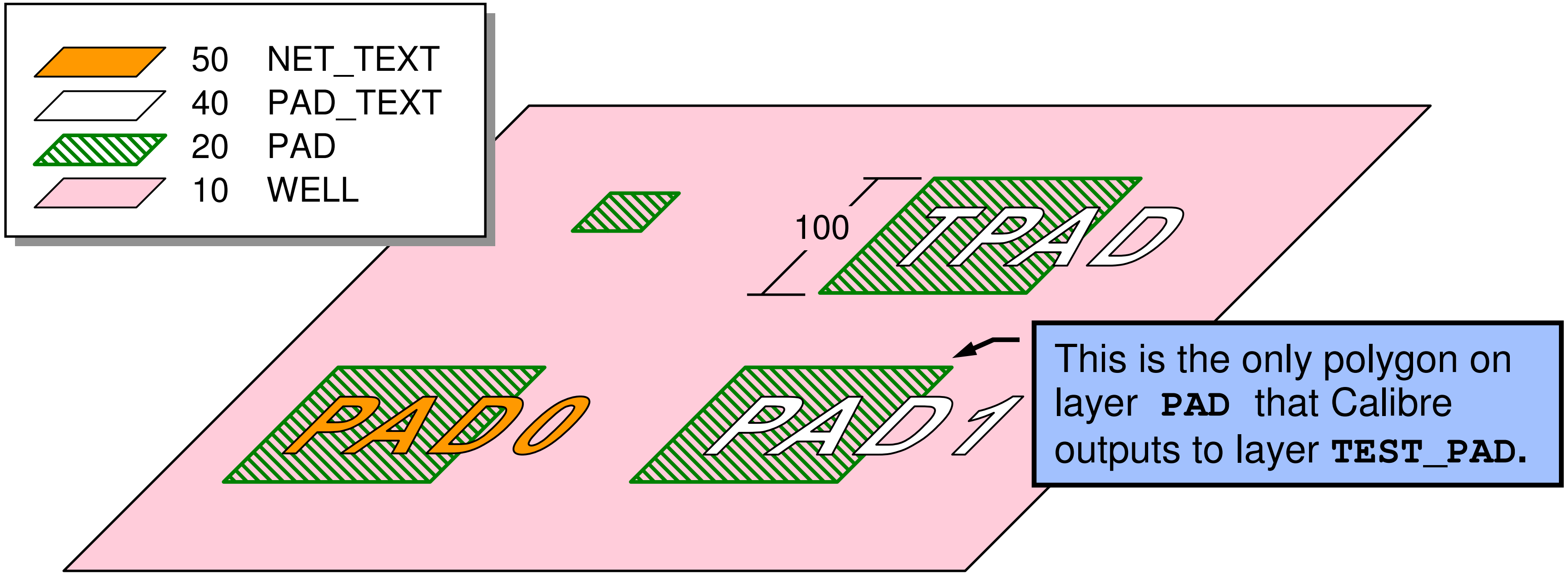
`name` — name of a free-standing text object; name can contain one or more wild card characters (“?”) and can be a string

`text_layer` — original layer where text objects are found; if not specified, text objects from all layers will be considered

`PRIMARY ONLY`— specifies that only top-cell text is used in the operation

- **Does not check connectivity**
- **Does not see text placed with `TEXT` statement**
- **Is not impacted by `TEXT LAYER` specification**

Example of With Text Operation



This is the only polygon on layer **`PAD`** that Calibre outputs to layer **`TEST_PAD`**.

rule file

```
// GENERATE DERIVED LAYER test_pad FROM ALL POLYGONS
// ON LAYER pad WITH AREA > 5000 AND INTERSECTED BY
// TEXT “pad?” ON LAYER 40
large_pad = pad  AREA > 5000
test_pad  = large_pad WITH TEXT “pad?” 40
```

Net

Purpose: Selects all layer polygons on the electrical node having the specified net name

Syntax: `[NOT] NET layer name […name]`

Connectivity on *layer* must be pre-established.

Parameters:

- ***layer*** — original layer or a derived polygon layer
- ***name*** — name of a net, which can contain one or more question mark (?) characters; name can be a string variable

Example:

```
METAL_SP {
@ VDD metal must be spaced at 4.5 microns
@ VCC metal must be spaced at 4 microns
    vdd_metal = metal NET vdd
    vcc_metal = metal NET vcc
    EXTERNAL vdd_metal < 4.5
    EXTERNAL vcc_metal < 4.0
}
```

Not a good tool to use to look for shorts!

Mentor Graphics®

Calibre Rule Writing

Module 5

Edge and Error-Directed Checks

Edge-Directed Operations

- **Edge-directed operations generate derived edge layers from original layers, layer sets or derived layers.**
- **An empty input layer presented to one of these operations will result in empty output.**
- **Edge operations operate on polygon and edge layers—they generate only edge layers.**
- **For this module, original layers are assumed to include layer sets as a sub-category.**

Inside Edge

Purpose: Selects all ***`layer1`*** edge segments that lie completely inside ***`layer2`*** polygons

Coincident edges not included

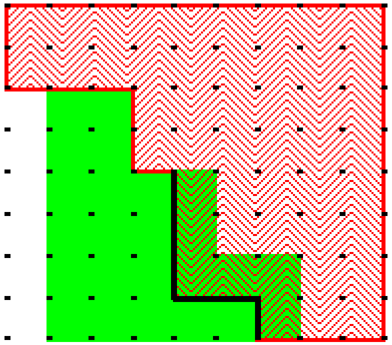
Syntax: `[NOT] INSIDE EDGE` ***`layer1 layer2`***

Parameters:

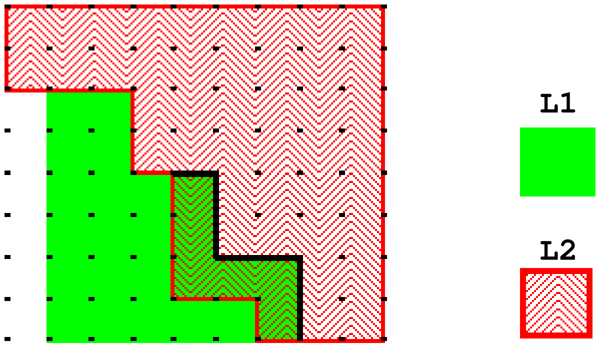
`layer1` — original or derived layer

`layer2` — original or derived polygon layer

Examples:



`L2 INSIDE EDGE L1`



`L1 INSIDE EDGE L2`

Outside Edge

Purpose: Selects all *`layer1`* edge segments that lie completely outside *`layer2`* polygons

Coincident edges not included

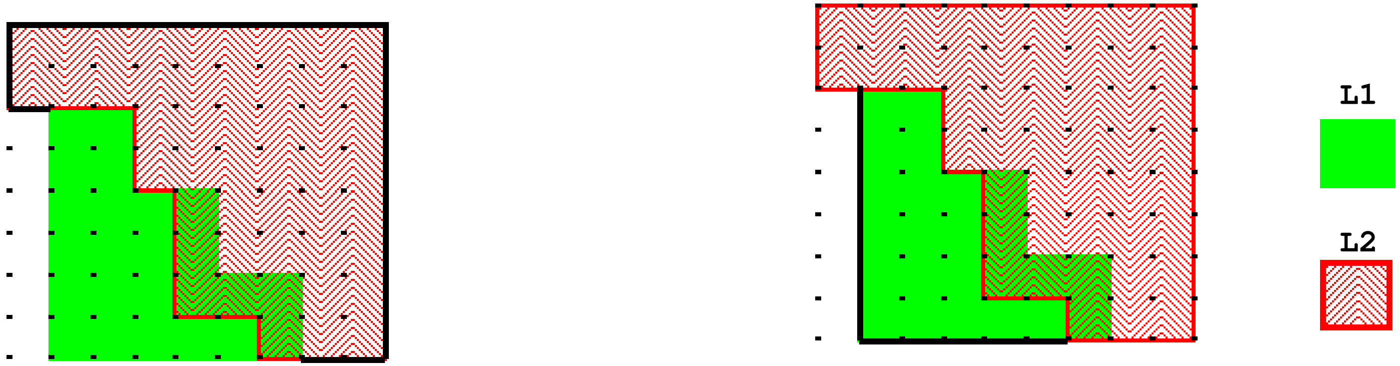
Syntax: `[NOT] OUTSIDE EDGE` *`layer1 layer2`*

Parameters:

`layer1` — original or derived layer

`layer2` — original or derived polygon layer

Examples:



`L2 OUTSIDE EDGE L1`

`L1 OUTSIDE EDGE L2`

Coincident Edge

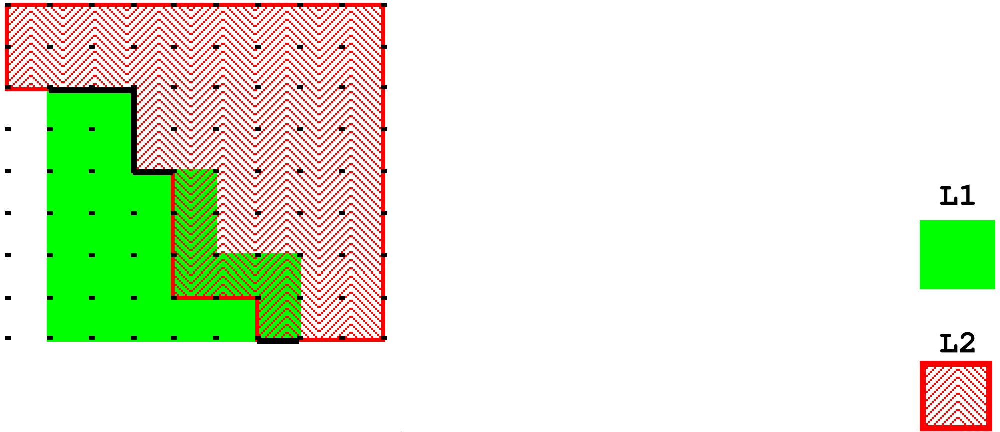
Purpose: Selects all ***layer1*** edge segments that coincide with ***layer2*** edges

Syntax: `[NOT] COINCIDENT EDGE layer1 layer2`

Parameters: ***layer1, layer2*** — original or derived layers

Example:

Although the results appear identical when the layers are interchanged, they have a different layer of origin. This may be important if you use the results in another operation.



```
L2 COINCIDENT EDGE L1
//INTERCHANGING LAYERS PROVIDES
//IDENTICAL RESULT
```

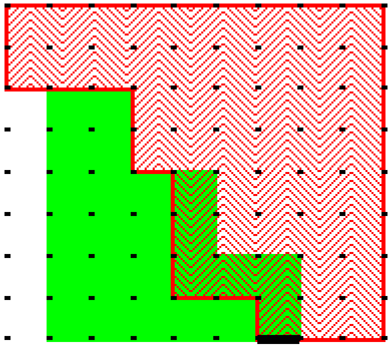
Coincident Inside Edge

Purpose: Selects all ***`layer1`*** edge segments that are inside-coincident with ***`layer2`*** edges

Syntax: **`[NOT] COINCIDENT INSIDE EDGE`** ***`layer1 layer2`***

Parameters: ***`layer1, layer2`*** — original or derived layers

Example:



L1 🟩

L2 ▨

```
L2 COINCIDENT INSIDE EDGE L1
//INTERCHANGING LAYERS PROVIDES
//IDENTICAL RESULT
```

Coincident Outside Edge

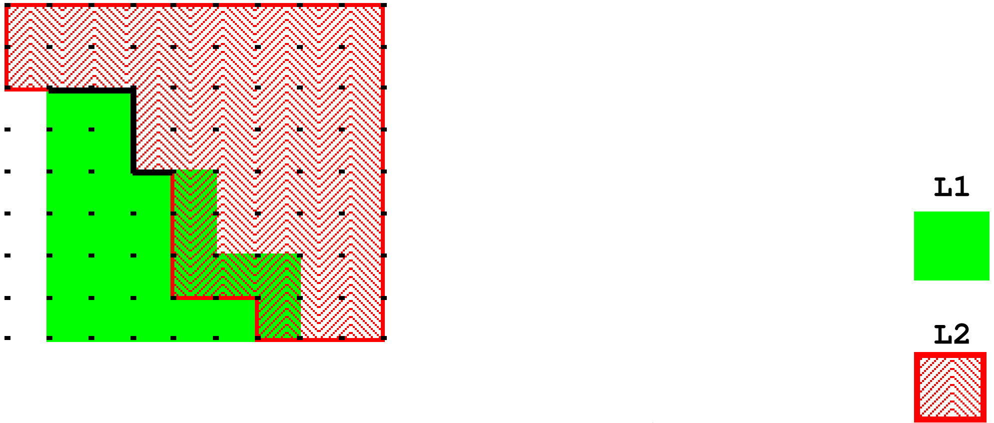
Purpose: Selects all ***`layer1`*** edge segments that are outside-coincident with ***`layer2`*** edges

Syntax: `[NOT] COINCIDENT OUTSIDE EDGE layer1 layer2`

Parameters: ***`layer1`***, ***`layer2`*** — original or derived layers

\Example:

Although the results appear identical when the layers are interchanged, they have a different layer of origin. This may be important if you use the results in another operation.



```
L2 COINCIDENT OUTSIDE EDGE L1
//INTERCHANGING LAYERS PROVIDES
//IDENTICAL RESULT
```

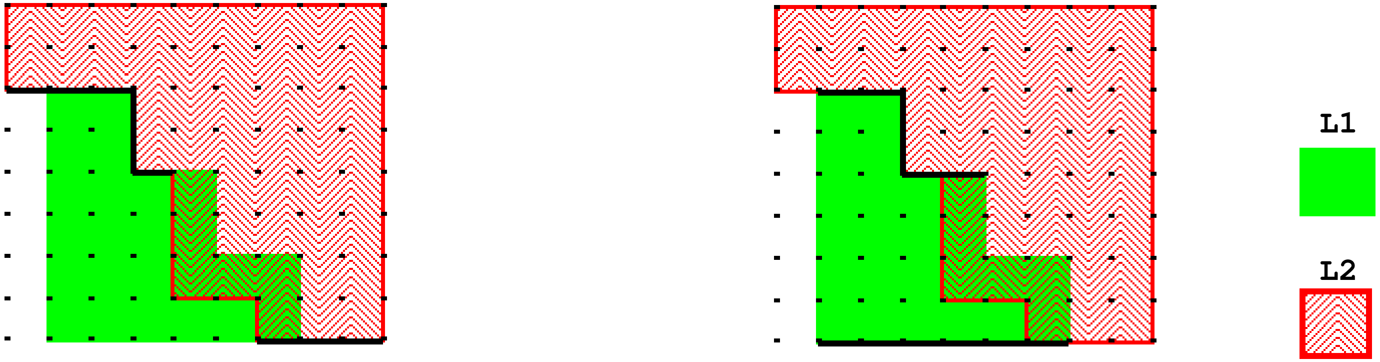
Touch Edge

Purpose: Selects complete ***layer1*** edges that touch ***layer2*** edges at more than 1 point

Syntax: `[NOT] TOUCH EDGE layer1 layer2`

Parameters: ***layer1***, ***layer2*** — original or derived layers

Example:



`L2 TOUCH EDGE L1`

`L1 TOUCH EDGE L2`

STATEMENT
OVERVIEW

Touch Inside Edge

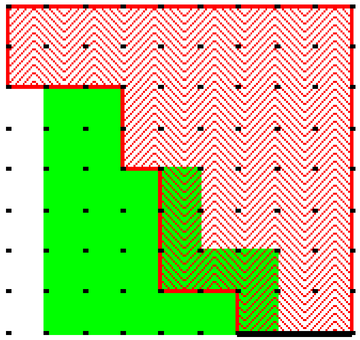
CLASS:
OPERATION

Purpose: Selects complete ***layer1*** edges that touch ***layer2*** edges on an inside edge of ***layer1***; the inside edge of ***layer1*** must face the interior of ***layer2***

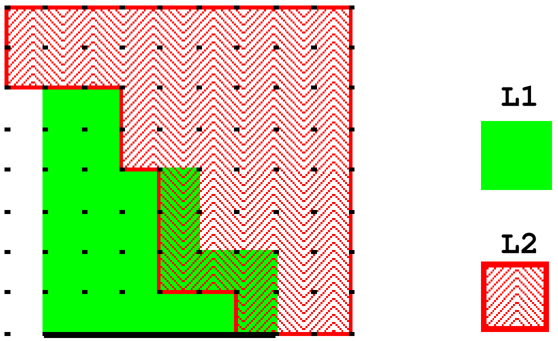
Syntax: `[NOT] TOUCH INSIDE EDGE` ***`layer1 layer2`***

Parameters: ***layer1***, ***layer2*** — original or derived layers

Example:



`L2 TOUCH INSIDE EDGE L1`



`L1 TOUCH INSIDE EDGE L2`

Touch Outside Edge

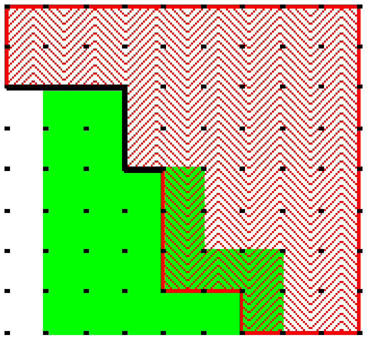
Purpose: Selects complete ***layer1*** edges that touch ***layer2*** edges on an outside edge of ***layer1***; the outside edge of ***layer1*** must face the interior of ***layer2***

Syntax:

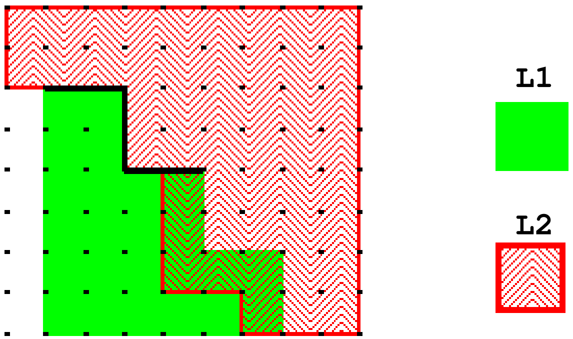
```
[NOT] TOUCH OUTSIDE EDGE layer1 layer2
```

Parameters: ***layer1***, ***layer2*** — original or derived layers

Example:



L2 TOUCH OUTSIDE EDGE L1



L1 TOUCH OUTSIDE EDGE L2

Length

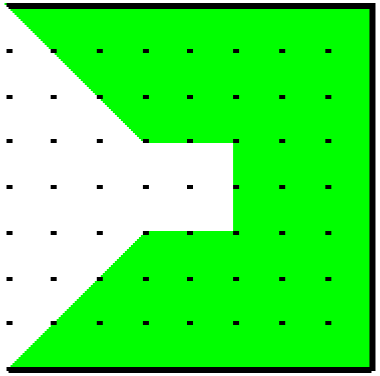
Purpose: Selects all input edges whose length satisfies the constraint

Syntax: `[NOT] LENGTH` ***`layer constraint`***

Parameters:

- ***`layer`*** — original or derived layer
- ***`constraint`*** — real number or range specifying length in user units

Example:



`LENGTH L1 > 5`

Path Length

Purpose: Selects all input edges from individual polygons which form continuous chains whose total length satisfies the constraint

Syntax: **`PATH LENGTH`** ***`edge_layer constraint`***

Parameters:

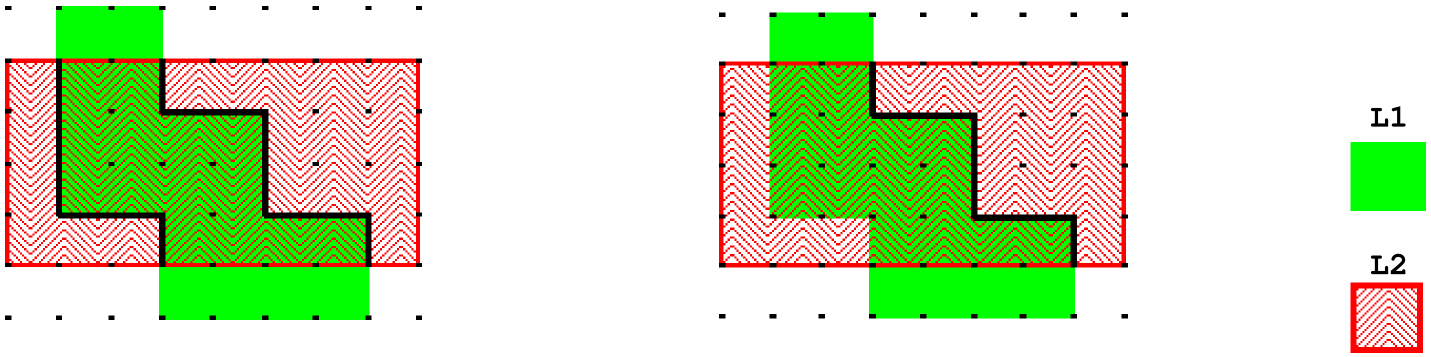
`edge_layer` — derived edge layer

`constraint` — real number or range specifying path length in user units

- **No selected edge can be a part of more than one selected path.**
- **If oblique edges exist on layer, it is best to use a range for constraint.**

Using Path Length — Example

```
LONG_PATH {
        @ SHOW PATH LENGTHS > 6u
           X = L1 INSIDE EDGE L2    // STEP 1
           PATH LENGTH X > 6        // STEP 2
    }
```



STEP 1

STEP 2

Error-Directed Checks

- The following set of specification statements and operations pertains to detecting types of error geometry including acute angles, skew lines and nonsimple polygons.
- Each of these statements operate on unmerged original geometries.
- In order for the Flag and Drawn statements to detect errors on specific layers, the layers must be read from the database.
  - Layers which appear in a RuleCheck get read.
  - Layers which do not appear in a RuleCheck may not get read, causing the Flag and Drawn statements to overlook them.

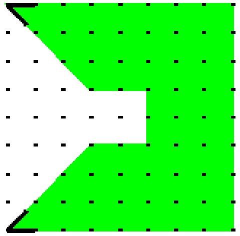
Drawn Acute

Purpose: Generates a derived error layer consisting of acute angle geometry markers

Syntax: `DRAWN ACUTE`

Example:

```
touch_L1 {AREA L1 == 0} //guarantee that L1 layer
                        // is used in RuleCheck
ACUTE_CHECK {DRAWN ACUTE}
```



- Acts on unmerged original geometries.
- Output is a two-edge cluster, each edge being 1 user unit in length, corresponding to the vertex of each acute angle.
- Specified once in a rule file.
- Statement must occur in the context of a RuleCheck.
- Scanned layers must be read by a layer operation or connectivity statement.

Drawn Skew

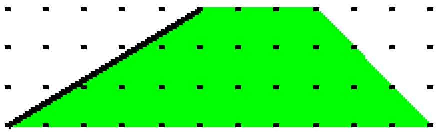
Purpose: Generates a derived error layer consisting of all skew edge geometry

Syntax: `DRAWN SKEW`

Example:

```
touch_L1{AREA L1 == 0} //guarantee L1 layer used in RuleCheck
SKEW_CHECK {DRAWN SKEW}
```

- **Skew edges are neither vertical nor horizontal and do not have slopes of +1 or -1 (non-45° multiples with respect to the X-axis).**
- **Acts on unmerged original geometries.**
- **Output is an edge corresponding to each original skew edge.**
- **Specified once in a rule file.**
- **Must occur in the context of a RuleCheck.**
- **Scanned layers must be read by a layer operation or connectivity operation.**



Layout Magnify

Purpose: Maps hierarchically (x, y) of all input polygon points to (x * factor, y * factor), as layout database is read into Calibre

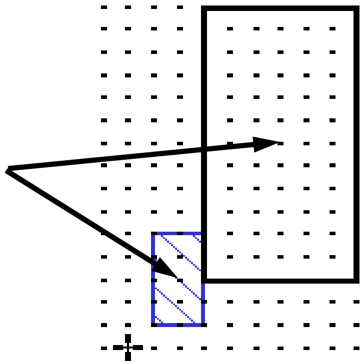
Syntax: `LAYOUT MAGNIFY` ***`factor`***

Parameter:

`factor`— required positive real value for magnification of layout database

- ***`factor`*** > 1 magnifies polygons, 0 < ***`factor`*** < 1 demagnifies.
- **Magnify does not check coordinate space overflow .**
- **Polygon centroids shift under this operation.**
- **Magnification is applied prior to DRC Rule checks.**

Layout Magnify (Cont.)



New vertices (3,3) and (9,15)

Centroid

`LAYOUT MAGNIFY 3`

Original vertices: (1,1) and (3,5)

Offgrid Checks

The following set of specification statements and operations handles resolution, offgrid checks and grid snapping:

- `RESOLUTION`
- `LAYER RESOLUTION`
- `FLAG OFFGRID`
- `DRAWN OFFGRID`
- `OFFGRID`
- `DRC TOLERANCE FACTOR`
- `SNAP`
- `SNAP OFFGRID`

Layers not read by other RuleChecks or connectivity statements are not checked!

Geometric Precision Specification Statements

The following statements specify the precision of Calibre nmDRC:

- User units—dimensioning units (for example, microns)
- Precision—ratio of database units to user units
  - Default value is 1000
  - For example:
    - `PRECISION 1000 // 1000 database units to 1 user unit`
- Resolution—the layout grid step size:
  - Allows off-grid flagging of original polygons

```
PRECISION 1000 // 1000 database units per user-unit
RESOLUTION 250
// Alignment points of original polygons must be every .25
// user units
```

Original geometries align at 250, 500, 750, or 1000 database units

Resolution

Purpose: Defines layout grid step size

Syntax: `RESOLUTION {`*`grid_size`*` | `*`x_grid y_grid`*`}`

Parameters:

grid_size — positive integer specifying both x and y layout grid step sizes

x_grid y_grid — positive integers specifying x and y grid step sizes respectively

Default: One database unit in x and y directions

Example:

```
PRECISION 1000
RESOLUTION 250 //POLYGON ALIGNMENT EVERY.25 USER UNITS
```

- **Primary use is to enable offgrid polygon checking**
- **Specified once in a rule file**

Layer Resolution

Purpose: Defines layout grid step size for specified original layers

Syntax: `LAYER RESOLUTION` ***`layer`*** `[`***`layer…`***`]`
`{`***`grid_size`***`|`***`x_grid y_grid`***`}`

Parameters:

`layer` — original layer (Must use name not number!)

`grid_size` — positive integer specifying both x and y layout grid step sizes

`x_grid y_grid` — positive integers specifying x and y layout grid step sizes respectively

Default: `RESOLUTION`

Example: `LAYER RESOLUTION POLY 50`

- **Overrides `RESOLUTION` value for the specified layer.**
- **May be specified once for each original layer.**

STATEMENT OVERVIEW

Flag Offgrid

CLASS: SPECIFICATION

Purpose:	Issues a warning upon detection of offgrid geometry; operates on unmerged original geometries
Syntax:	`FLAG OFFGRID YES	NO`
Parameters:	`YES` — enables offgrid warning `NO` — disables offgrid warning
Default:	`NO`
Example:	`FLAG OFFGRID YES`

- **Report lists a maximum of 100 warnings.**
- **Output includes the coordinates, layers and cell names of original offgrid vertices to the Summary Report and the transcript.**
- **Does not output the results to the DRC Database.**
- **Specified once in a rule file.**

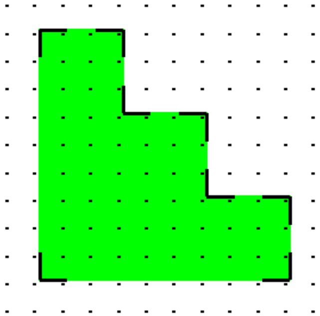
Drawn Offgrid

Purpose: Generates a derived error layer consisting of offgrid geometry markers

Syntax: `DRAWN OFFGRID`

Example:

```
touch_L1{AREA L1 == 0} // guarantee L1 used
OFFGRID_CHECK { // Check for off grid
   DRAWN OFFGRID}
```



- Acts on unmerged original geometries.
- Uses grid specified in resolution statements.
- Generates two-edge error clusters which correspond to adjacent edges sharing a common endpoint which is an offgrid vertex.
- Specified once in a rule file.
- Must occur in the context of a RuleCheck.
- Scanned layers must be read by a layer operation.

Offgrid

Purpose: Generates a derived error layer consisting of offgrid geometry markers for the specified layer

Syntax: `OFFGRID` ***`layer {grid_size | x_grid y_grid}`***

Parameters:

`layer` — original or derived polygon layer

`grid_size` — positive integer specifying both x and y snap grid step sizes

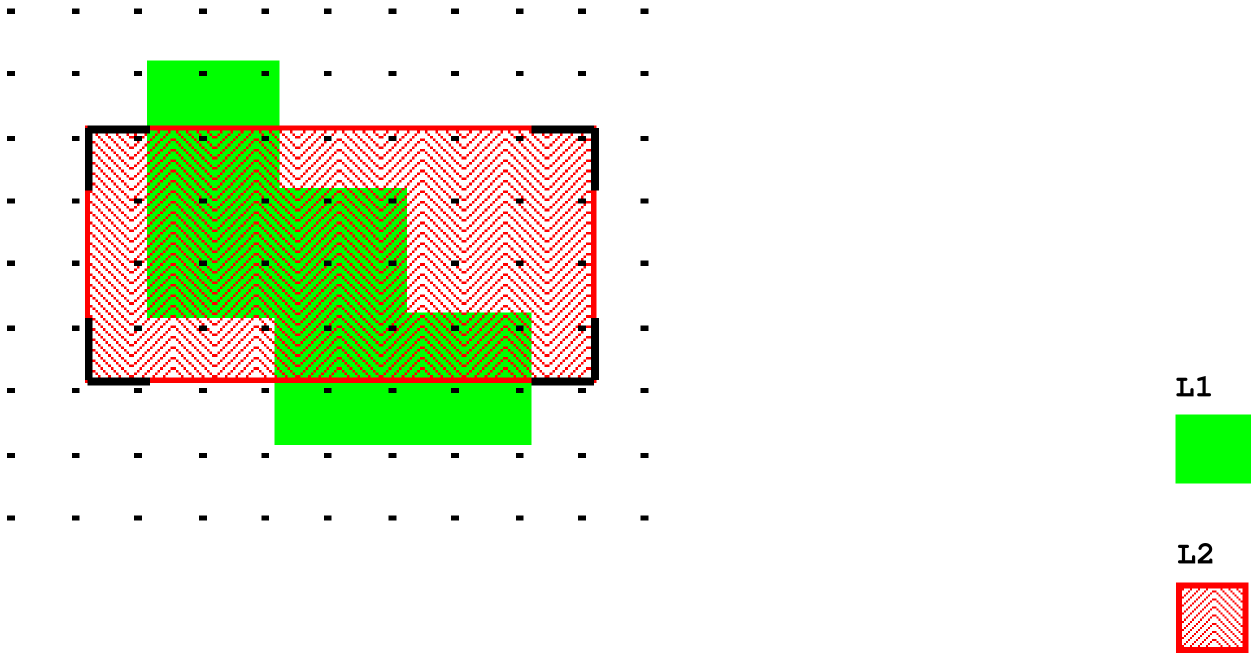
`x_grid y_grid` — positive integers specifying x and y snap grid step sizes respectively

- **Generates two-edge error clusters which correspond to adjacent edges sharing a common endpoint which is also an offgrid vertex**
- **Must occur in the context of a RuleCheck**

Offgrid — Example

Example:

```
L2_OFFGRID{
   @L2 MUST BE ON .250u GRID
   OFFGRID L2 250
 }
```



DRC Tolerance Factor

Purpose: Suppress false errors on non-Manhattan geometries (such as 45° paths and circular structures) where, due to round off errors, distances between edges can be slightly less than the process-specified value

Syntax: **`DRC TOLERANCE FACTOR`** ***`tolerance`***

Parameters: ***`tolerance`*** — a positive real number or numeric expression in user units

Note:
Numeric expressions can normally contain variables.

Defaults: none

Example: **`DRC TOLERANCE FACTOR .003`**

STATEMENT OVERVIEW

Snap

CLASS: OPERATION

Purpose: Snaps input layer vertices to the specified grid

Syntax: `SNAP layer {snap_grid | x y}`

Parameters:

`layer` — original or derived polygon layer

`snap_grid` — positive integer specifying both x and y snap grid step sizes

`x y` — positive integers specifying x and y snap grid step sizes respectively

Example:

```
PRECISION 1000
SNAP_DIFF = SNAP DIFF 10 //DIFF ON .01 USER UNIT GRID
```

- **Preserves 45° bends if *`x_grid`* = *`y_grid`***
- **May be specified once for each original layer**

For hierarchical Calibre applications, snapping to unequal ***x*** and ***y*** resolutions is not permissible and the least common multiple of the two resolutions is used instead.

Snap Offgrid

Purpose: Snaps all unmerged offgrid vertices on original layers to the grid specified in the `RESOLUTION` statement or appropriate `LAYER RESOLUTION` statements

Syntax: `SNAP OFFGRID YES | NO`

Parameters: `YES` — enables offgrid vertex snapping

`NO` — disables offgrid vertex snapping (default)

Example: `SNAP OFFGRID NO`

- **Occurs before offgrid, acute or skew checks.**
- **Preserves 45° bends if *`x_grid`* = *`y_grid`* in `SNAP` statement.**
- **For hierarchical DRC, initial cell placements are snapped followed by shape snapping on a per-cell basis.**
    - **Resolution for placement snapping is the least common multiple of all grid values in the `RESOLUTION` and `LAYER RESOLUTION` statements.**
    - **If x and y grids are unequal, resolution becomes their least common multiple.**

Calibre Rule Writing

Module 6

Other Topics

Hierarchical DRC Applications

The following set of statements are primarily used in hierarchical DRC applications:

- `LAYOUT BASE LAYER`
- `EXCLUDE CELL`
- `LAYOUT RENAME CELL`
- `LAYOUT ALLOW DUPLICATE CELL`
- `INSIDE CELL`
- `EXPAND CELL`

Layout Base Layer

Purpose: Specifies device-level layers for performance tuning of hierarchical applications.

Syntax: **`LAYOUT BASE LAYER`** ***`layer`*** **`[…`*****`layer`*****`]`**

Parameters: ***`layer`*** — original layer name

Default: **None**

Example:

```
LAYOUT BASE LAYER POLY DIFF CONTACT NPLUS PPLUS
```

- **NOTE:** Do not include substrate or wells layers in **`LAYOUT BASE LAYER.`**
- This statement should be in any hierarchical rule file.

Layout Base Layer (Cont.)

- Recommended layers to include:
  - all device-forming layers like poly and diffusion
  - implant layers (nplus and pplus specifically)
  - contact (not via) layers
- Do not include these layers:
  - Metal
  - Via (not contact)
  - Solder bump
  - Pad
  - Fuse
  - Artificial cell boundary
  - Well
  - Substrate
- For improved hierarchical processing, a rule file needs to contain either `LAYOUT BASE LAYER` or `LAYOUT TOP LAYER` statement.
- `LAYOUT BASE LAYER` is easier to use than `LAYOUT TOP LAYER.`
  - `LAYOUT TOP LAYER` is the inverse of `LAYOUT BASE LAYER`.
  - Calibre ignores `LAYOUT TOP LAYER` if in the same Rule file as `LAYOUT BASE LAYER`.

Exclude Cell

Purpose: Excludes specified cells from DRC and LVS processing

Syntax: `EXCLUDE CELL name […name]`

Parameters: ***name*** — name of a cell to be excluded

Default: No cells are excluded

Example: `EXCLUDE CELL "ADDER*"`

- **Cells in the cell list are excluded, including all instances within any hierarchy.**
- **May be specified more than once.**
- **Not supported for ASCII or binary databases.**
- **The " * " wildcard is permitted for cell names in quotes.**

Inside Cell

Purpose: Selects shapes on the specified layer inside specified cells

Syntax:

```
[NOT]INSIDE CELL layer name[…name]
[PRIMARY ONLY] [WITH MATCH]
[WITH LAYER layer2]
```

Parameters:

layer — original layer

name — cell name

PRIMARY ONLY — instructs the tool to output geometry only from the top level of the specified cells

WITH MATCH — allows a placed cell to be treated as a ***name*** parameter in the operation if this cell geometrically matches another cell (unplaced) that is specified as a ***name*** parameter

WITH LAYER ***layer2*** —limits selection to the specified cells having any geometry on ***layer2*** (must be an original layer) in their immediate hierarchy.

Inside Cell (Cont.)

Examples:

```
x = INSIDE CELL metal ramcell romcell
```

Select all **`metal`** from cells **`ramcell`** and **`romcell`**, including the subhierarchies.

```
x = INSIDE CELL metal ramcell romcell PRIMARY ONLY
```

Exclude **`metal`** from selection existing in the subhierarchies of **`ramcell`** and **`romcell`**. If **`romcell`** is instantiated in the subhierarchy of **`ramcell`**, then **`metal`** at the primary level in cell **`romcell`** is still selected by the operation.

```
metal1_sram = metal1 INSIDE CELL `*' WITH LAYER sram
```

Select all polygons from `metal1` that are inside any cell, including the subhierarchies of any cell limited to those cells having any geometry on layer `sram` in their immediate hierarchy.

- **Parameter order is important to avoid ambiguity.**
- **The “ * ” wildcard is permitted for cell names in quotes.**

STATEMENT OVERVIEW

Expand Cell

CLASS: SPECIFICATION

Purpose:	Expands instances of cells one level to fill the cells in which they are placed
Syntax:	**`EXPAND CELL`** ***`name`*** **`[…`*****`name`*****`]`**
Parameters:	***`name`*** — name of a cell to be expanded
Default:	None
Example:	**`EXPAND CELL “ADDER*” “MUX*”`**

- **Particularly useful for improving FPGA performance by expanding base cell containers down to the level of the base.**
- **May be specified more than once.**
- **The “ * ” wildcard is permitted for cell names in quotes.**

Dual Database Capabilities

- **Calibre has the capability to compare two separate layout databases.**
  - **Layout Versus Layout (LVL) comparison is the most common application.**
  - **Supported database types: GDSII and OASIS.**
  - **Used for comparing one database against another database without merging data.**

- **LVL comparison requires these specification statements:**
  - `LAYOUT SYSTEM2`
  - `LAYOUT PATH2`
  - `LAYOUT PRIMARY2`
  - `LAYOUT BUMP2`

LVL Comparison

- When comparing two layout databases:
  - Specify one of the databases using:
    - `LAYOUT SYSTEM`
    - `LAYOUT PATH`
    - `LAYOUT PRIMARY`
  - Specify the other database using:
    - `LAYOUT SYSTEM2`
    - `LAYOUT PATH2`
    - `LAYOUT PRIMARY2`
- You may specify `LAYOUT PATH` and `LAYOUT PATH2` more than once to input multiple databases.
- Each database is constructed by merging the individual files (all into one database or all into two separate databases) into their respective databases.
- The possibility of duplicate layer number assignments requires special consideration…

Next slide

Layout Bump2

Purpose:	Increments second layout database layer numbers by a specified value
Syntax:	**`LAYOUT BUMP2`** ***`value`***
Parameters:	***`value`*** – positive integer
Default:	None

- ***`value`*** **should be greater than the largest layer number found in the first layout database.**
- **Calibre ignores drawn layer objects from the first database whose numbers are greater than or equal to** ***`value`*****.**
- **Applies to geometry and text layers.**

Layout Bump2 Example

```
//DATABASE 1 Original LAYERS
      LAYER POLY    1
      LAYER OXIDE   2
      LAYER CONTACT 3
              .
              .
     LAYER VIA5    31
```

```
//DATABASE 2 Original LAYERS
      LAYER POLY    1
      LAYER OXIDE   2
      LAYER CONTACT 3
              .
              .
     LAYER VIA5    31
```

```
LAYOUT BUMP2 100
```

- **Causes Database 2 layers to be incremented by 100 (database 2 layer 1 is read in as layer 101)**
- **Use RuleChecks to compare layers in the two layout databases**

Example:

```
comp_poly {// find differences between POLY layers
           XOR 1 101}
```

STATEMENT OVERVIEW

Layout Rename Cell

CLASS: SPECIFICATION

Purpose:	Renames a cell as the GDSII or OASIS database is read
Syntax:	`LAYOUT RENAME CELL` ***`source_cell target_cell`***
Parameters:	***`source_cell`*** — cell to be renamed ***`target_cell`*** — new name of ***`source_cell`***
Default:	None
Example:	`LAYOUT RENAME CELL TOPCELL TOPCELL_1`

- **Particularly useful for dual database applications for the establishment of cell correspondence**
- **May be specified once per** ***`source_cell`***

Layout Allow Duplicate Cell

Purpose: Specifies whether multiple records for the same layout cell are allowed for the input layout database

Syntax: `LAYOUT ALLOW DUPLICATE CELL NO | YES`

Parameters: `NO` — instructs the tool not to allow multiple records for the same layout cell; all records after the first will be discarded *

`YES` — cells with the same name will be merged together

Default: `NO`

Example: `LAYOUT ALLOW DUPLICATE CELL YES`

Useful when the database is split into multiple files by layer

*** See also `LAYOUT INPUT EXCEPTION SEVERITY`**

Utilities Making Dual Database Comparison Easier

- **`create_compare_rules`**
  Utility to create a rule file for Calibre dual-database comparison. The rule file `XORs` all non-empty (layer, datatype) coordinates in the input gds file(s). LAYER MAPs and bumps are created automatically.
- **`compare_gds`**
  Allows you to compare two GDSII databases (flat). This utility produces an ASCII DRC results database based on a layer-by-layer analysis.
- **Both utilities have a 64-bit version available.**
  - **`create_compare_rules -64`**
  - **`compare_gds -64`**
  - **64-bit version allows input files larger than 2 Meg.**
  - **Requires 64-bit license.**

`create_compare_rules` Utility

- **This utility scans a database and outputs a rule file that can be used to compare the original layout with another layout.**
- **Syntax:**

```
$MGC_HOME/bin/create_compare_rules [-COPY]
     output_rule_file layout_database1 |
     output_rule_file layout_database1 layout_database2
```

  - `-COPY` **—An optional argument that causes the utility to use the** `Copy` **operation in the** *`output_rule_file`* **rather than the** `XOR` **operation. The RuleChecks in the** *`output_rule_file`* **generate copies of all the layers from the** *`layout_database1`***.**
  - *`output_rule_file`* **— pathname of the generated rule file**
  - *`layout_databaseN`* **— pathname of a layout database. When one pathname is provided, a generic comparison rule file is output (list of layers, etc.). When two pathnames are provided, the** *`output_rule_file`* **assumes the two databases are compared, and the appropriate statements appear in it.**

See the *Calibre Verification User's Manual* for more information on how to use this utility

`compare_gds` Utility Syntax

```
$MGC_HOME/bin/compare_gds database1 top_cell1
       [-RULES rule-file1]database2 top_cell2
       [-RULES rule-file2]output-database
       [-NOT|-XOR ] [-NOKEEPEMPTY]
```

`databaseN – GDSII database`

`top_cellN – top cell in the database`

`-RULES rule-fileN – Rule files for each database (looking for LAYER MAP statements)`

`-NOT` – changes the comparison from and **`XOR`** operation to a Boolean **`NOT`** of ***`database1`*** and ***`database2`***, in that order

`-XOR – default operation`

`-NOKEEPEMPTY` –If the **`XOR`** is empty, **`diff_L`** is an empty rule check unless the **`–NOKEEPEMPTY`** switch is specified; in that event, **`diff_L`** does not exist. (See next slide.)

`compare_gds` Utility

- **This utility compares two GDSII databases *`database1`* and *`database2`* with top-cells *`top_cell1`* and *`top_cell2`*.**
- **The comparison is between layers (from 0 to 8191) that have geometry in at least one of the databases.**
- **For each layer `L` with shapes in at least one of the input databases, the shapes are flattened and a Boolean `XOR` is done between the resulting two layers.**
- **Results of the `XOR` are written to the output DRC results database with the rule check name "`diff_L`" where `L` is the layer number.**
- **If the `XOR` is empty, `diff_L` is an empty rule check unless the `–NOKEEPEMPTY` switch is specified; in that event, `diff_L` does not exist.**
- **The program does not consider datatype nor does it compare text.**

Defining Macros

- Macros are functional templates that can be called multiple times in a rule file.
- A macro definition consists of the keyword `DMACRO` (define macro), followed by a name, followed by a list of zero or more arguments, followed by “{”, followed by a sequence of zero or more SVRF statements or operations, followed by “}”. For example:

```
DMACRO WIDTH_CHECK lay val {
R1 = INT lay < val ABUT < 90 SINGULAR REGION
R2 = INT lay < val ANGLED == 2 PARALLEL OPPOSITE REGION
R1 OR R2
}
```

- `DMACRO` names must be unique, each argument must be a name, and an argument may not be duplicated in the same `DMACRO` argument list.
- Macro definitions cannot be nested.

Calling Macros

- A macro is invoked by the keyword `CMACRO` (call macro) followed by a macro name and a list of zero or more arguments.
- Each argument may be either a name or a numeric constant.
- The macro name referenced in a `CMACRO` statement must match that of some `DMACRO` definition and a sufficient number of arguments must be present after the `CMACRO` name.
- For example, calling the previous `DMACRO`:

```
poly_width { CMACRO WIDTH_CHECK poly 0.5 }
metal_width { CMACRO WIDTH_CHECK metal 0.6 }
```

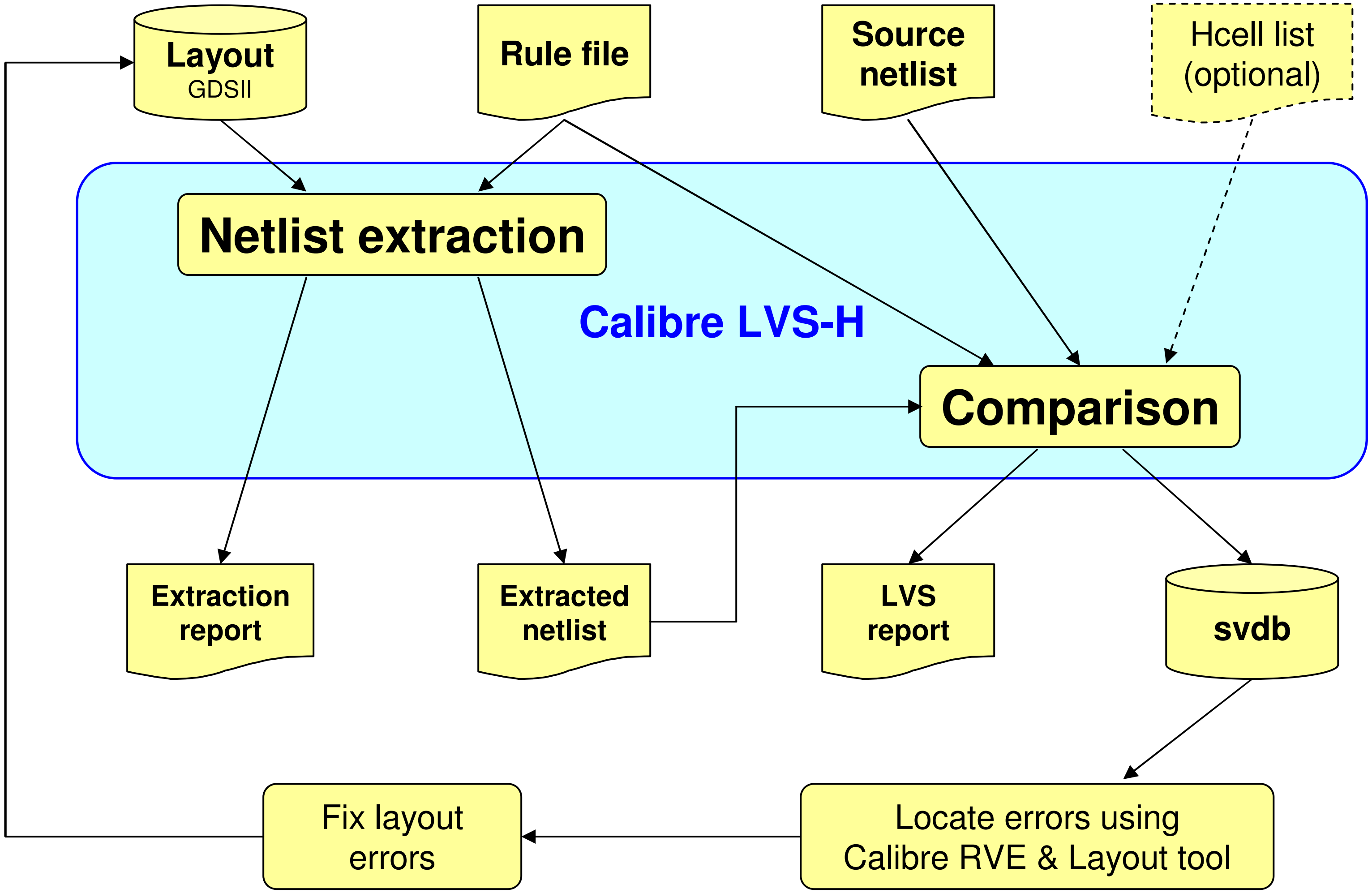
- The arguments `poly` and `0.5` are substituted into the `DMACRO` `WIDTH_CHECK` and this becomes the RuleCheck `poly_width`.
- `DMACRO` definitions may themselves contain `CMACRO`s.
- Recursive `DMACRO`s are not allowed.

Calibre Rule Writing

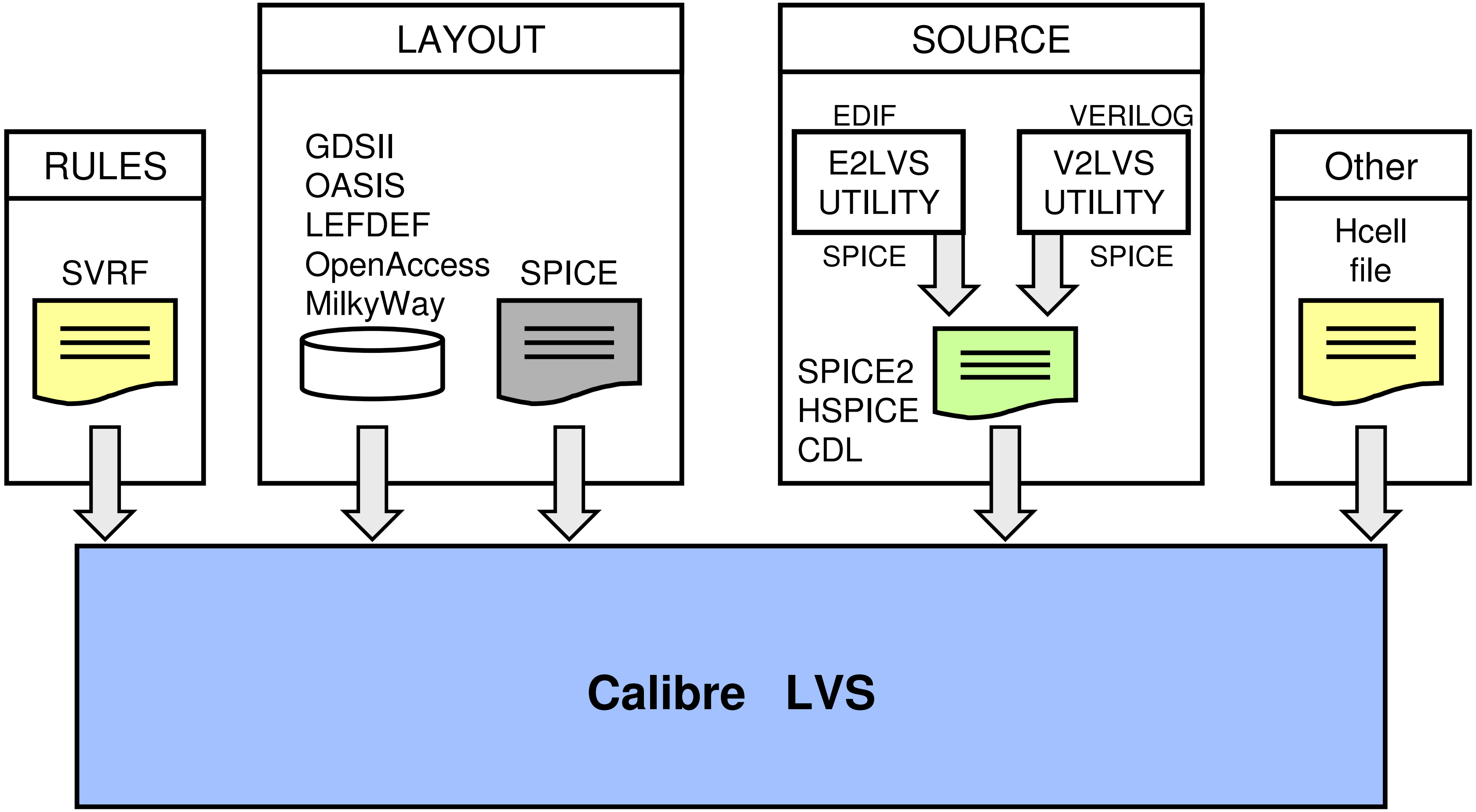
Module 7

LVS Basics

Layout Verification Process Flow for LVS



Calibre LVS Input File Formats



Basic Specification Statements

rule file

```
//--------------------------------

// OPTIONAL HEADER INFORMATION

//--------------------------------

// REQUIRED DRC SPECIFICATION STATEMENTS

LAYOUT SYSTEM         GDSII

LAYOUT PATH           “./mydesign.gds”

LAYOUT PRIMARY        top_cell

//OPTIONAL INCLUDED RULE FILES

INCLUDE “/home/process/drc/golden_rules”

//REQUIRED LVS SPECIFICATION STATEMENTS

SOURCE SYSTEM         SPICE

SOURCE PATH           “./mydesign.spi”

SOURCE PRIMARY        top_cell

LVS REPORT            “lvs_report”

MASK SVDB DIRECTORY   “svdb” QUERY

...
```

Layout Input Statements

The next three statements specify the target layout:

- **`LAYOUT SYSTEM`** **— type of layout file**
- **`LAYOUT PATH`** **— path to file**
- **`LAYOUT PRIMARY`** **— top cell**

Source Input Statements

The next three statements specify the target source:

- `SOURCE SYSTEM` — type of source file
- `SOURCE PATH` — path to file
- `SOURCE PRIMARY` — top cell

STATEMENT
OVERVIEW

Source System

CLASS:
SPECIFICATION

Purpose:	Specifies the source database type
Syntax:	**`SOURCE SYSTEM`** ***`type`***
Parameters:	***`type`*** — keyword example: **`SPICE`** or **`CNET`**
Default:	**none**
Example:	**`SOURCE SYSTEM SPICE`**

- **You must specify this statement once in the rule file.**

Source Path

CLASS:
SPECIFICATION

Purpose:	Specifies the source database pathname(s)
Syntax:	`SOURCE PATH` ***`filename`***
Parameters:	***`filename`*** — the pathname of the source database
Default:	**none**
Example:	`SOURCE PATH “/tmp/work/mydesign.spi”`

- **You can specify this statement only once in the rule file.**
- **The *`filename`* may contain environment variables.**

Source Primary

Purpose: Specifies a subcircuit, cellname for SPICE source systems

Syntax: `SOURCE PRIMARY` ***`name`***

Parameters: ***`name`*** — a required top-level cell or subcircuit name of the source database

Default: **none**

Example: `SOURCE PRIMARY “cpu_topcell”`

Mask SVDB Directory

Purpose: Specify the Standard Verification Database Directory and the types of files generated

Syntax:

`MASK SVDB DIRECTORY` ***`directory_path`*** **`[QUERY] [XRC] [CCI] [IXF] [NXF] [PHDB] [PINLOC] [NOPINLOC] [GDSII] [XDB] [DV] [SLPH] [NETLIST] [ANNOTATE DEVICES] [NOFLAT] [BY GATE]`**

Parameters:

`directory_path` — absolute or relative pathname

`QUERY` — create files needed for query server operation

`XRC` — creates all information necessary for the Calibre xRC flow

`CCI` — creates a file containing the same information as **`PHDB`**, **`GDSII`**, **`XDB`**, **`NETLIST`**, and **`ANNOTATE DEVICES`** options

`IXF` — creates an instance cross-reference file

`NXF` — creates a net cross-reference file

`PHDB` — creates a persistent hierarchical database

`PINLOC|NOPINLOC` — controls the generation of pin location information

Mask SVDB Directory (Cont.)

Parameters (cont.):

- **`GDSII`** — creates information sufficient for generating Annotated GDSII files
- **`XDB`** – creates a file containing the same information as IXF and NXF, but is not interchangeable with them
- **`DV`** — creates a Discrepancy Viewer database
- **`SLPH`** – creates layout and source placement hierarchy files
- **`NETLIST`** — creates information to generate layout netlists from the SVDB database
- **`ANNOTATE DEVICES`** — adds fully-merged device seed shapes annotated with device numbers to the PHDB database
- **`NOFLAT`** — instructs flat Calibre LVS not to create the SVDB directory
- **`BY GATE`** — instructs Calibre LVS applications to write information about logic gates

Default: None

Mask SVDB Directory (Cont.)

Examples:

```
MASK SVDB DIRECTORY “./results/svdb” QUERY
MASK SVDB DIRECTORY svdb CCI
MASK SVDB DIRECTORY “./results/svdb” IXF NXF SLPH
```

- You must specify the `QUERY` option to run Calibre-RVE.
- `PHDB` option allows LVS debugging in IC Station without creating cross-reference files.
- Mask SVDB outputs differently in Flat LVS than Hierarchical LVS.
- Also used by Calibre xRC.

LVS Report Control Statements

The following statements affect LVS report generation:

- `LVS REPORT`
- `LVS REPORT MAXIMUM`
- `LVS REPORT OPTION`

LVS Report

Purpose: Specifies the file name of the LVS report

Syntax: `LVS REPORT filename`

Parameters: ***filename*** — specifies the file name of the LVS report

Default: **None**

Example: `LVS REPORT “./lvs.rpt”`

- **You must include this statement to run Calibre LVS.**

LVS Report Maximum

Purpose: Specifies the maximum number of printed items per section in the LVS report

Syntax: `LVS REPORT MAXIMUM [number | ALL]`

Parameters: ***number*** — specifies max number of printed items

ALL — specifies no limit of printed items

Default: `50` (recommended for most cases)

Example: `LVS REPORT MAXIMUM 25`

- **Specifies max number of discrepancies and max number of items per discrepancy.**
- **Calibre lists the most critical discrepancies first.**
- **Setting number = `–1` also specifies no limit (same as `ALL`).**

STATEMENT OVERVIEW

LVS Report Option

CLASS: SPECIFICATION

Purpose: Controls the detail and verbosity of the LVS report file

Syntax: `LVS REPORT OPTION` ***`option1 …optionN`***

Parameters:

`option` – Large number of options available most commonly used are:

- `S` – Reports Sconnect conflicts
- `V` – Reports virtual connections

See the SVRF manual for the complete listing of options.

Default: None of the keywords are specified

Example: `LVS REPORT OPTION S V`

Many LVS REPORT options generate extremely large amounts of data. Take care when using them.

LVS Power and Ground Specification Statements

The next three statements affect LVS power and ground specification statements:

- `LVS POWER NAME`
- `LVS GROUND NAME`

LVS Power Name

Purpose:	Specifies a list of power net names
Syntax:	`LVS POWER NAME name […name]`
Parameters:	*name* — name of a power net
Default:	**No names specified**
Example:	`LVS POWER NAME VDD VDDA VDDB ?VCC?`

- **Required for logic gate recognition and certain device filtering operations.**
- **You may specify this statement multiple times.**

STATEMENT OVERVIEW

LVS Ground Name

CLASS: SPECIFICATION

Purpose: Specifies a list of ground net names

Syntax: `LVS GROUND NAME name […name]`

Parameters: ***name*** — name of a ground net

Default: **No names specified**

Example: `LVS GROUND NAME VSS AGND DGND`

- **Required for logic gate recognition and device filtering.**
- **You may specify this statement multiple times.**

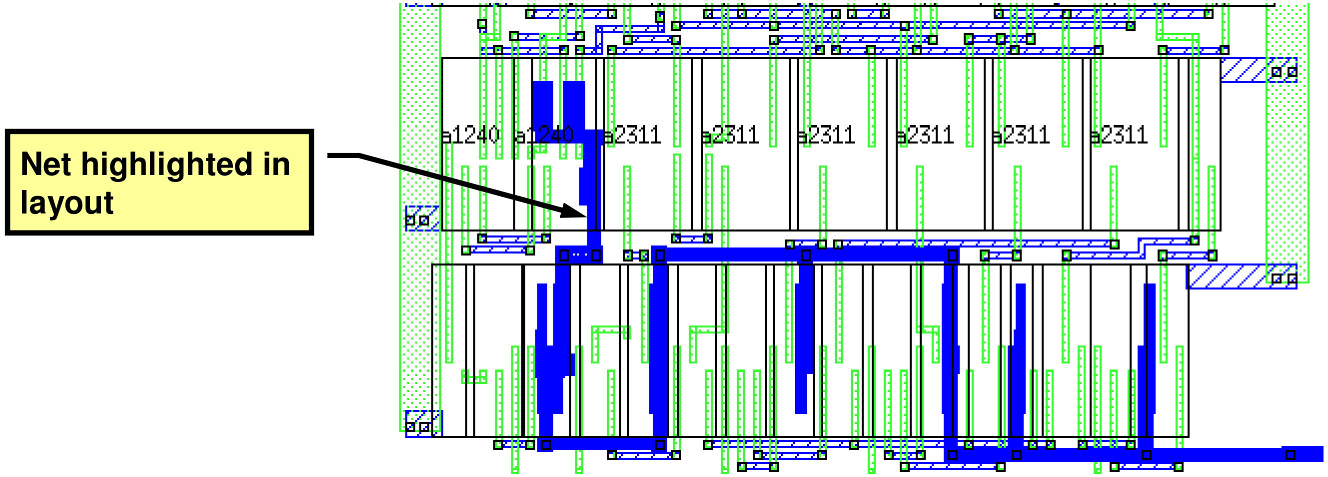
Calibre Rule Writing

Module 8

Establishing Connectivity

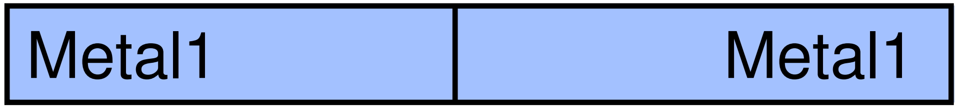
Nets

- A net is a set of objects that are electrically connected.
- A net could include a connection between several layout geometries on several different layers.
- Each net is given an unique number for identification after connectivity extraction is run.

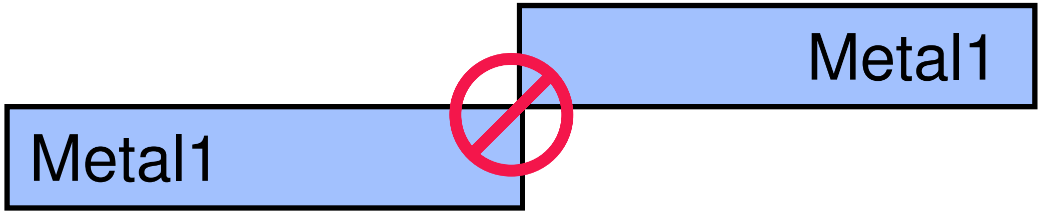


How Calibre Establishes Connectivity

- Shapes on a single layer that abut or overlap are considered part of a single net.



- Single point connections (singularities) do NOT give connectivity.



Connectivity Extraction Operators

The following slides describe connectivity extraction operators:

- **`CONNECT`**
- **`CONNECT BY`**
- **`SCONNECT`**
- **`LVS SOFTCHK`**
- **`LVS ABORT ON SOFTCHK`**

Connect

Purpose: Specifies connection between abutting or overlapping polygons

Syntax: `CONNECT layer1 …layerN`

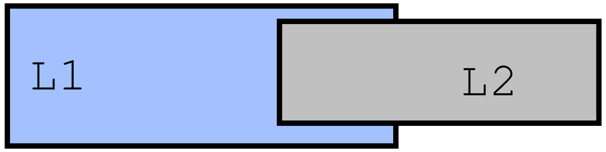
`CONNECT layer1 …layerN BY layerC`

Parameters:

- ***`layer1 …layerN`*** — original layer, derived layer or layer set
- **`BY`** ***`layerC`*** — specifies mutual connection layer

Default: Uses both mask and direct

Example:



`CONNECT L1 L2`

- **Use the `CONNECT` operation when establishing connectivity on one or more layers.**
- **All layers are order independent.**

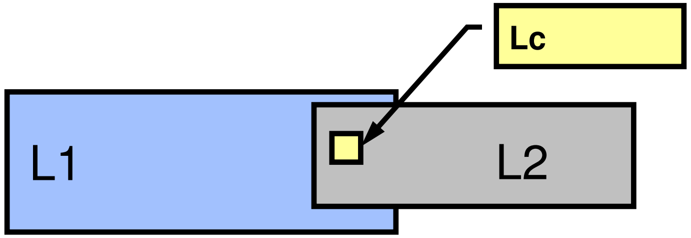
Connect (Cont.)

- **Calibre always treats abutting or overlapping polygons on the same interconnect layer as part of the same net.**
- **You may specify up to 32 layers in a `CONNECT` operation.**
- **Connectivity transfer for this operator is always bidirectional.**

Connect Example Using ‘By *LayerC*’

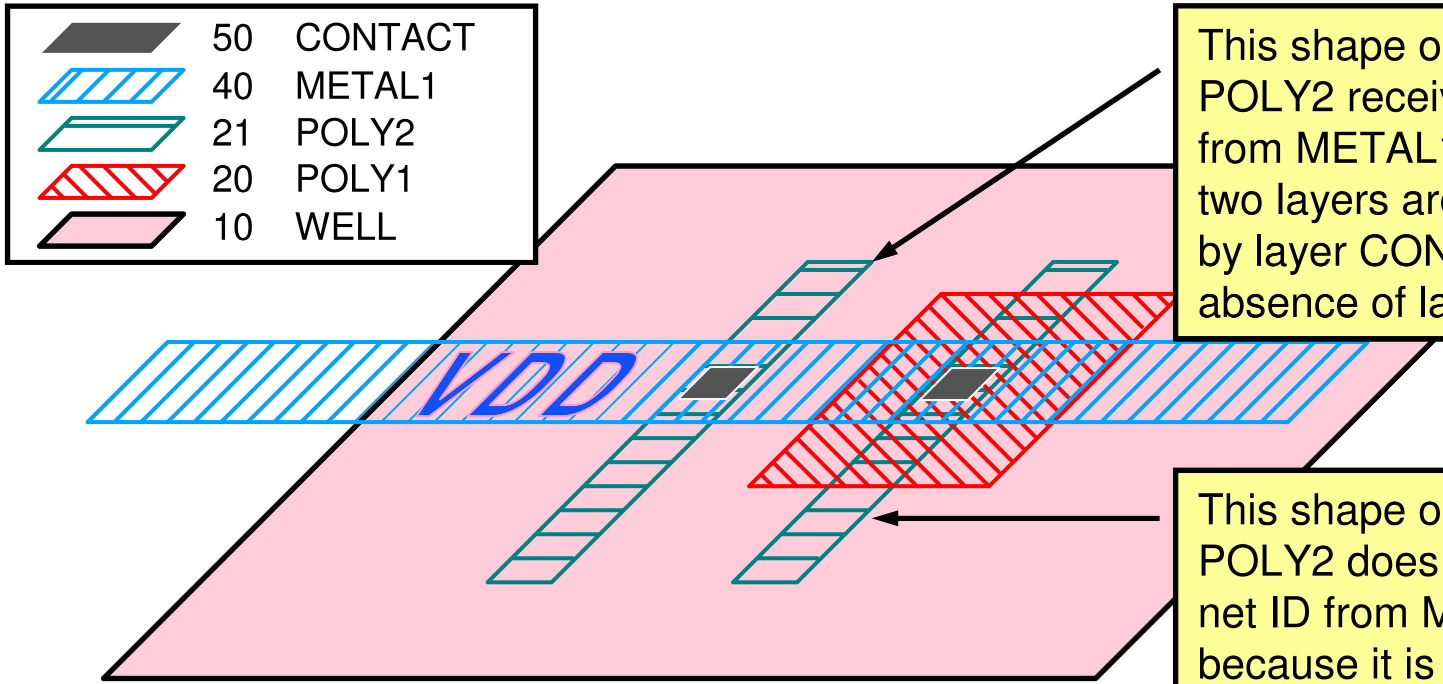
`CONNECT` ***`layer1 …layerN`*** `BY` ***`layerC`***

- **Polygons on two layers can be connected to each other by mutual intersection with a third polygon on a “contact” layer specified in a `CONNECT BY` operation.**
- **Only *`layerC`* and the first mutally-intersecting shape found on layers *`layer2`* through *`layerN`* are connected to the *`layer1`* shape. This is shielding.**
- **Shielding only applies if you specify `BY` *`layerC`*.**
- **Example:**



`CONNECT L1 L2 BY Lc`

Example #1 of Connect Operation



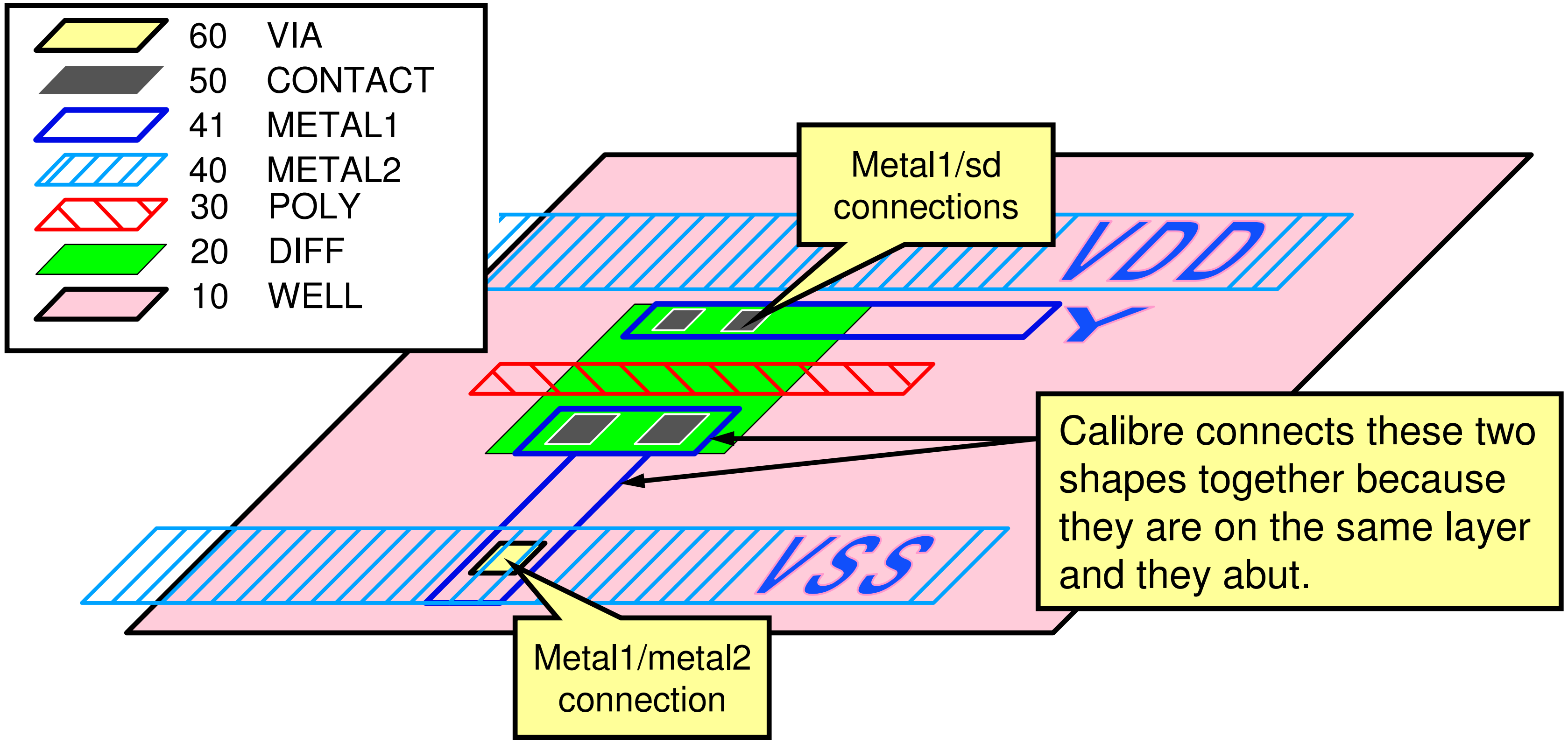
This shape on layer POLY2 receives a net ID from METAL1 because the two layers are overlapped by layer CONTACT in the absence of layer POLY1.

This shape on layer POLY2 does not receive a net ID from METAL1 because it is shielded by POLY1.

rule file

```
CONNECT metal1 poly1 poly2 BY contact
```

Example #2 of Connect Operation



Metal1/sd connections

Calibre connects these two shapes together because they are on the same layer and they abut.

Metal1/metal2 connection

rule file

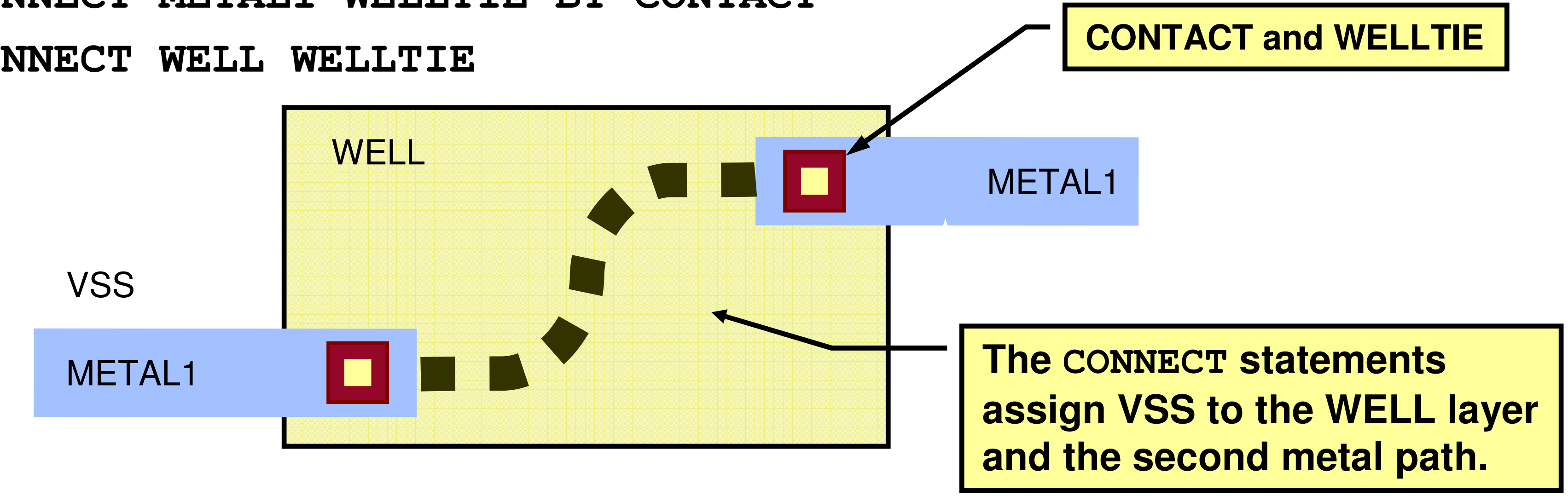
```
sd = diff not poly //derived layer — covered in module 4
CONNECT metal1 metal2 BY via
CONNECT metal1 poly sd BY contact
```

What Are Soft Connections?

- The use of a high-resistivity layer to connect two conductors creates a soft connection.
- Soft connections are usually undesirable for electrical performance reasons.
- Soft connections satisfy LVS requirements for network connectivity but can lead to unsatisfactory circuit performance.

Soft Connection Example

```
CONNECT METAL1 WELLTIE BY CONTACT
CONNECT WELL WELLTIE
```



CONTACT and WELLTIE

The `CONNECT` statements assign VSS to the WELL layer and the second metal path.

- **Calibre sees a connection between the two metal paths through the high resistance WELL.**
- **The missing hardwire connection between the two metal paths is not detected—circuit fails.**

Sconnect

Purpose: Specifies a one-way connection between an upper layer and a lower layer

Syntax:

```
SCONNECT upper_layer lower_layer [LINK name][ABUT ALSO]
SCONNECT upper_layer lower_layer…lower_layerN
     BY  contact_layer  [LINK name]
```

Parameters:

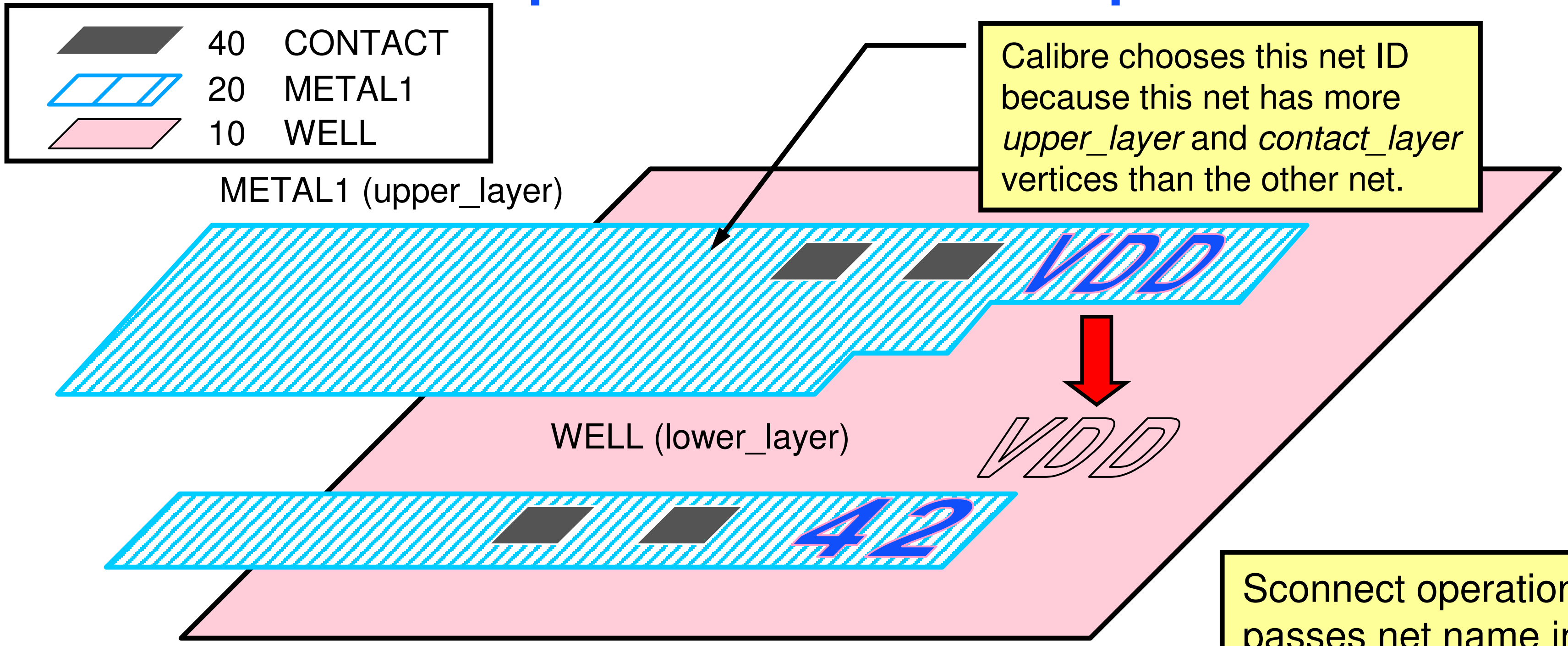
- ***upper_layer*** — original layer, derived layer or layer set
- ***lower_layer*** — original layer, derived layer or layer set
- ***lower_layerN*** — original layer, derived layer or layer set
- **LINK *name*** — specifies node id for floating polygons
- **ABUT ALSO** — allows abutment to constitute overlap
- **BY *contact_layer*** — specifies mutual connection original layer, derived layer or layer set

LINK *name* must already exist in the layout.

Sconnect (Cont.)

- **Connections established by `SCONNECT` are unidirectional—net identification is passed from the upper layer to the lower layer only.**
- **Use the `SCONNECT` operation when you need to specify connection to a high-resistivity layer (e.g. a well) and you want to identify soft connection attempts involving that layer.**
- **Shielding applies if you specify more than one lower layer.**
- **Layer *`upper_layer`* must have previously-assigned connectivity.**
- **Layers *`lower_layer …lower_layerN`* must not have previously-assigned connectivity.**
- **See also `LVS SOFTCHK` and `LVS REPORT OPTION S`.**
- **Use of `SCONNECT` rather than `STAMP` is encouraged.**

Example #1 of Sconnect Operation



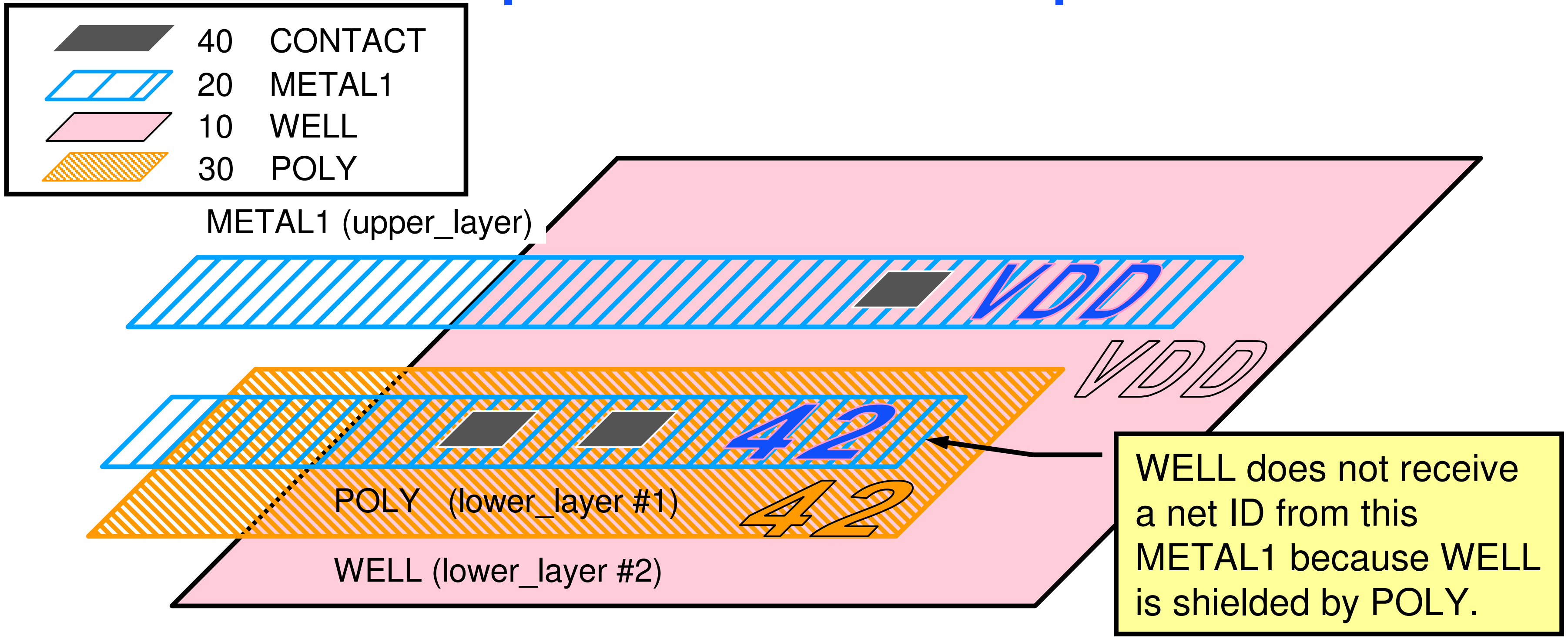
Calibre chooses this net ID because this net has more *upper_layer* and *contact_layer* vertices than the other net.

Sconnect operation passes net name in one direction only.

rule file

```
// LAYER METAL1 MUST HAVE ITS CONNECTIVITY PREVIOUSLY
// ESTABLISHED
// ESTABLISH CONNECTION BETWEEN METAL1 AND WELL
// IF BOTH LAYERS ARE INTERSECTED BY CONTACT LAYER
SCONNECT metal1 well BY contact
```

Example #2 of Sconnect Operation



WELL does not receive a net ID from this METAL1 because WELL is shielded by POLY.

rule file

```
// LAYER METAL1 MUST HAVE ITS CONNECTIVITY PREVIOUSLY
// ESTABLISHED
// ESTABLISH CONNECTION BETWEEN METAL1 AND POLY
// OR BETWEEN METAL1 AND WELL
SCONNECT metal1 poly well BY contact
```

STATEMENT OVERVIEW

LVS Softchk

CLASS: SPECIFICATION

Purpose: Finds and reports conflicting connections resulting from **`SCONNECT`** operations. Creates a DRC results database for viewing in DRC-RVE.

Syntax:

```
LVS SOFTCHK lower_layer {CONTACT|UPPER|LOWER} [ALL]
```

Parameters:

`lower_layer` — original layer, derived layer or layer set

`CONTACT` — selects ***`contact_layer`*** polygons from an **`SCONNECT`** operation

`UPPER` — selects ***`upper_layer`*** polygons from an **`SCONNECT`** operation

`LOWER` — selects ***`lower_layer`*** polygons from an **`SCONNECT`** operation

`ALL` — All electrical nodes involved in conflicting connections are eligible for reporting. Not effect if you specify LOWER.

the net selected by **`SCONNECT`** for establishing the net ID of the ***`lower_layer`*** along with all other ***`upper_layer`*** nets for error reporting

Default: **`LOWER`**

LVS Softchk (Cont.)

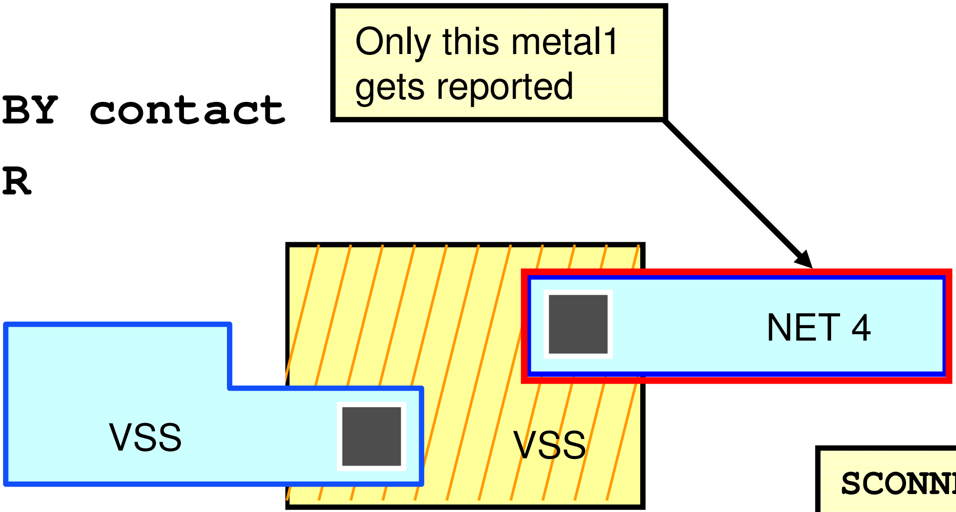
- Provides soft connection checking for Calibre LVS.
- Reports soft connection attempts to the specified *`lower_layer`* generated by `SCONNECT` operations .
- Keyword `CONTACT|UPPER|LOWER` specifies a layer on which to report polygons involved in soft connections.
- Parameter *`lower_layer`* must appear in an `SCONNECT` operation.
- Calibre writes LVS Softchk results to `primary_cell.softchk` in the SVDB directory. This file is viewable in Calibre DRC-RVE.
- Takes full advantage of hierarchical processing and reporting.

Locating Soft Connections With the SCONNECT Operator

Rule file #1:

```
SCONNECT metal1 well BY contact
LVS SOFTCHK WELL UPPER
```

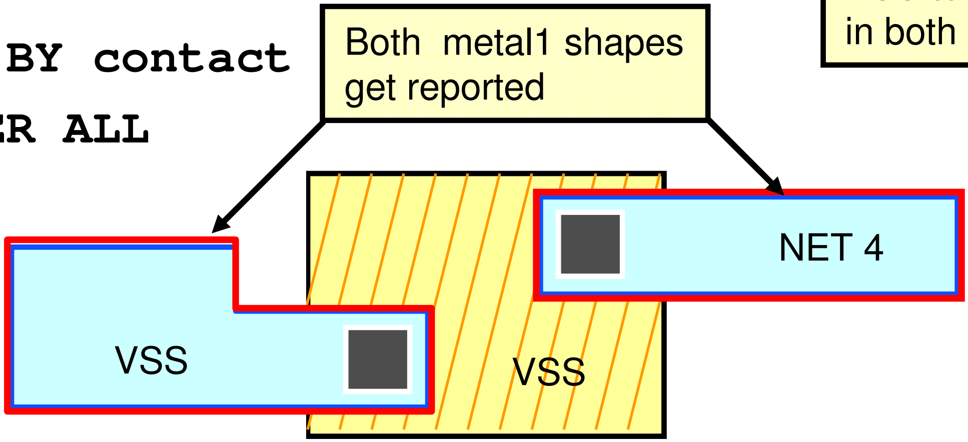
■	40	CONTACT
□	20	METAL1
▨	10	WELL


Only this metal1 gets reported

SCONNECT heuristic assigns net ID VSS to the well in both cases.

Rule file #2:

```
SCONNECT metal1 well BY contact
LVS SOFTCHK WELL UPPER ALL
```


Both metal1 shapes get reported

LVS Abort On Softchk

Purpose: Specifies whether to abort LVS processing if Calibre detects a soft connection conflict resulting from an **`SCONNECT`** operation

Syntax: **`LVS ABORT ON SOFTCHK {YES|NO}`**

Parameters:

`YES` — Calibre aborts processing on **`SCONNECT`** soft connection conflict

`NO` — Calibre continues processing on **`SCONNECT`** soft connection conflict

Default: **`NO`**

Example: **`LVS ABORT ON SOFTCHK YES`**

Initial Correspondence Points

- **Pairs of nets or ports which have identical user-given names in the source and the layout.**
- **Good practice to name the ports of the top-level cell and the major nets in the design.**
- **"Information and Warnings" section of the LVS report lists the Initial Correspondence Points.**
- **Use the report to resolve circuit ambiguities between the source and the layout.**
- **Improves processing performance.**
- **Do not confuse with matching cell names.**

Texting

Prerequisites for successful texting of nets and ports:

- **Specify which layers are valid text layers.**
- **Establish connectivity of target object layers.**
- **Attach the text labels to target objects.**

LVS Cpoint

Purpose: Specifies a correspondence point between a layout net and a source net. With this information, an LVS application can match the layout and source databases through the use of the specified net names.

Syntax: **`LVS CPOINT`** ***`layout_net_name source_net_name`***

Parameters:

`layout_net_name` — required name of a net the database specified in the **`LAYOUT PATH`** specification statement

`source_net_name` — required name of a net in the database specified in the **`SOURCE PATH`** specification statement

Examples:

```
LVS CPOINT "AAA" "X1/X2/5"
LVS CPOINT "BBB" "CCC"
LVS CPOINT "DDD" "7"
LVS CPOINT "X3/X4/5" "X6/N1"
```

Texting Statements

The following statements control how Calibre recognizes and uses text:

- **`LAYOUT TEXT`**
- **`TEXT LAYER`**
- **`TEXT DEPTH`**
- **`LAYOUT RENAME TEXT`**
- **`ATTACH`**

Layout Text

Purpose: Treats a text object as if in a GDSII, OASIS, OpenAccess, or MilkyWay layout database

Syntax:

```
LAYOUT TEXT name location layer [texttype] cellname
```

Parameters:

name — name (label) of the text object

location — x,y coordinate in the space of the specified cell (in user units)

layer — original layer name or layer set

texttype — specifies the object’s GDSII texttype

cellname — specifies the destination cell for the text object

Default: **None**

Example: `LAYOUT TEXT clock 2000 3000 metal2 “alu”`

Layout Text (Cont.)

- Applies only to GDSII, OASIS, OpenAccess, or MilkyWay layout systems
- Applies to both LVS connectivity extraction and DRC `WITH TEXT` operations
- Overwritten by text placed with a `TEXT` statement at the same location
- Obeys `TEXT LAYER`, `TEXT DEPTH` and `LAYER MAP` statements
- Attaches text to the specified cell in the hierarchy using cell coordinates

Text Layer

Purpose:	Specifies the layers in the database from which Calibre reads free-floating text
Syntax:	**`TEXT LAYER`** ***`layer1 …layerN`***
Parameters:	***`layer`*** — original layer name or number
Default:	**None**
Example:	**`TEXT LAYER poly metal2 50`**

- **Calibre uses free-floating text to name nets during connectivity extraction.**
- **Statement does not apply to `WITH TEXT` operations.**

Text Depth

Purpose: Specifies hierarchical depth for reading text objects from the layout database

Syntax: **`TEXT DEPTH ALL|PRIMARY|`*`number`***

Parameters:

`ALL` — read text from all levels of the hierarchy

`PRIMARY` — read text at the level of the **`PRIMARY`** cell name only

`number` — read text from the top number+1 layers

Default: **`PRIMARY`**

Example:

```
TEXT DEPTH 1  // read text from the top
              // two hierarchical levels
```

- **Applies to text placed with the `LAYOUT TEXT` statement.**
- **Does not apply to text placed with the `TEXT` statement.**
- **Applies only to LVS connectivity extraction.**
- **For hierarchical LVS, text is used at the level where it is placed.**

Layout Rename Text

Purpose: Specifies text values to be edited or replaced

Syntax:

LAYOUT RENAME TEXT ***delimiter find_pattern delimiter replace_pattern delimiter*** **[n|g] [e|b] [i|m] [Mc]**

Parameters:

delimiter — any single character except space or new line

find_pattern — specifies regular expression to replace

replace_pattern — string that replaces ***find_pattern***

[n|g] — specifies which occurrences to replace (**n**= next **g** = all)

[e|b] — specifies which form of regular syntax to use
(**e**= extended **b** = basic)

[i|m] — specifies case sensitivity (**b** = ignore case **m** = match case)

Mc — specifies meta-character used in ***replace_pattern***

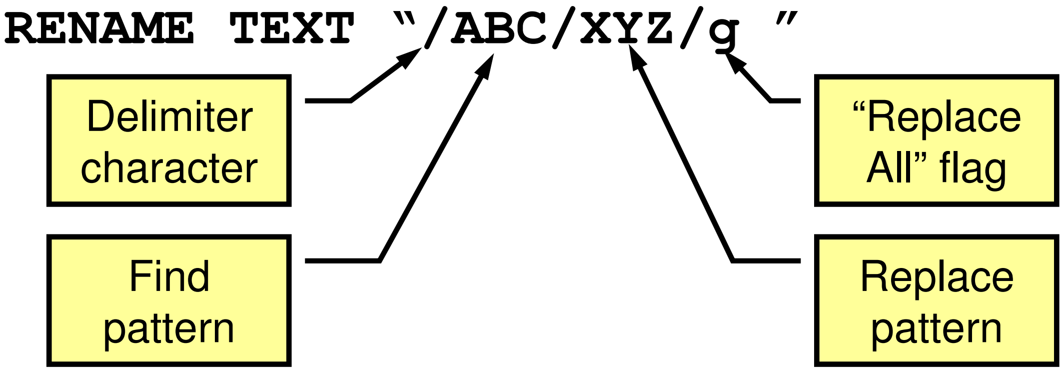
Default:

Replace nth occurrence, use extended syntax, ignore case

Layout Rename Text (Cont.)

- **Applies to both Calibre DRC and LVS.**
- **Applies to text read from GDSII or CIF databases, and text specified by `LAYOUT TEXT` or used by `WITH TEXT` statements.**
- **Does not apply to text specified by `TEXT` statements.**
- **The delimiter character must appear exactly three times.**
- **Enclose the parameters in quotes if using special characters.**
- **You may specify this statement more than once.**

- **Example:** `LAYOUT RENAME TEXT “/ABC/XYZ/g ”`



Text Label Attachment

- After you establish connectivity for layout polygons, then you may attach text labels to name nets and ports.
- There are three methods for attaching labels — (highest to lowest priority):
  - Explicit attachment
  - Implicit attachment
  - Free attachment

Make the attachment method consistent throughout the design.

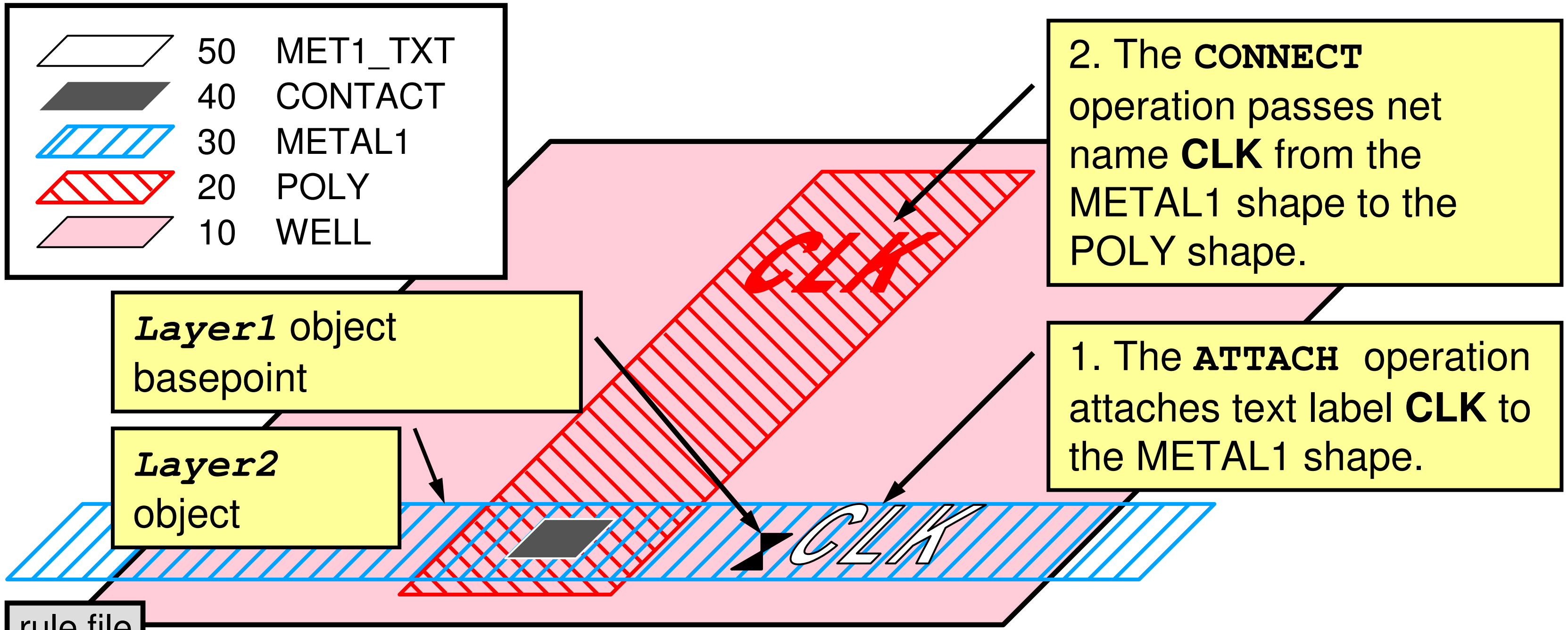
- Two rule file statements control how Calibre attaches text labels:
  - Attach (controls explicit attachment)
  - Label Order (controls free attachment)

Attach

Purpose:	Explicitly attaches connectivity information from a ***layer1*** object to a ***layer2*** object
Syntax:	`ATTACH layer1 layer2`
Parameters:	***layer1*** — original layer, derived layer or layer set ***layer2*** — original layer, derived layer or layer set. Must appear as an input layer to a **CONNECT** or **SCONNECT** operation.
Default:	Uses both mask and direct

- ***layer1* objects are typically text objects, shapes and paths.**
- **If *layer1* object is a polygon, it must be completely overlapped by the *layer2* object.**
- **Connectivity information can be net names or port names.**

Example #1 of Explicit Label Attachment



▱	50	MET1_TXT
▰	40	CONTACT
▨	30	METAL1
▨	20	POLY
▱	10	WELL

Layer1 object basepoint

Layer2 object

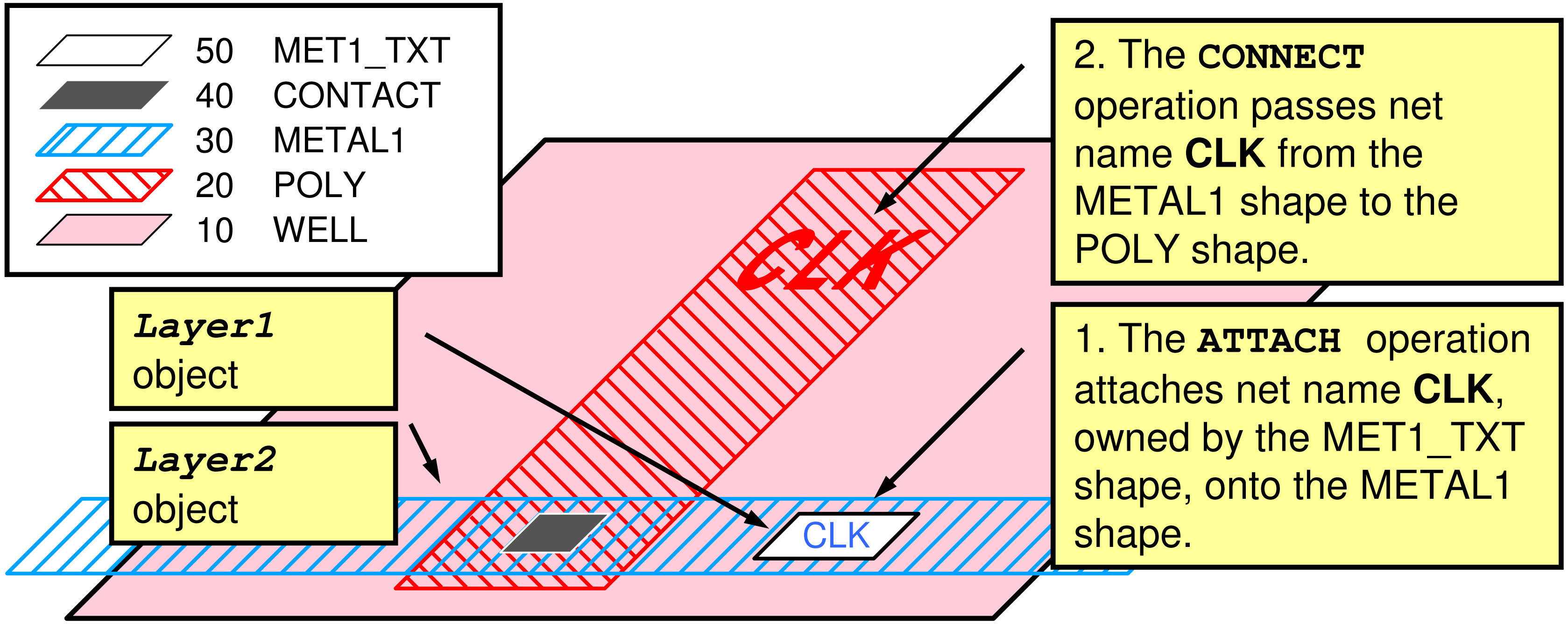
2. The **CONNECT** operation passes net name **CLK** from the METAL1 shape to the POLY shape.

1. The **ATTACH** operation attaches text label **CLK** to the METAL1 shape.

rule file

```
LAYER met1_txt 50
TEXT LAYER met1_txt
// ATTACH TEXT LABELS ON LAYER met1_txt to
// OVERLAPPING SHAPES ON LAYER metal1
ATTACH met1_txt metal1
// LAYER METAL1 MUST APPEAR IN A (S)CONNECT OPERATION
CONNECT metal1 poly BY contact
```

Example #2 of Explicit Label Attachment



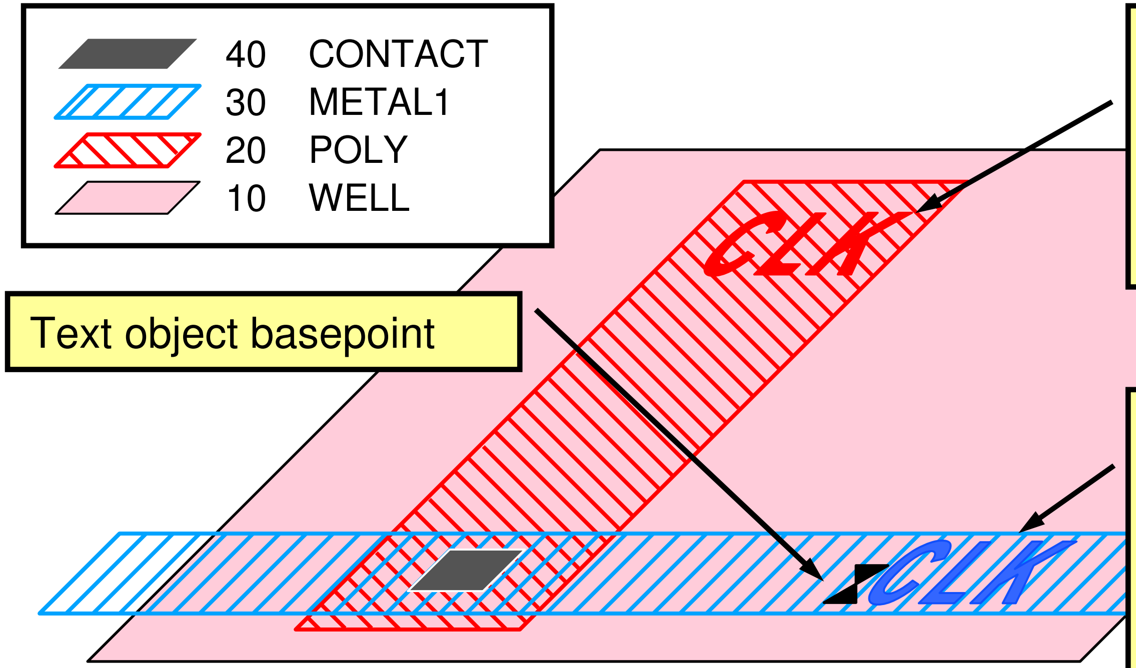
2. The **`CONNECT`** operation passes net name **CLK** from the METAL1 shape to the POLY shape.

1. The **`ATTACH`** operation attaches net name **CLK**, owned by the MET1_TXT shape, onto the METAL1 shape.

rule file

```
// ATTACH CONNECTIVITY INFORMATION FROM SHAPES ON
// LAYER met1_txt TO SHAPES ON LAYER metal1
ATTACH met1_txt metal1
// LAYER metal1 MUST APPEAR IN A (S)CONNECT OPERATION
CONNECT metal1 poly BY contact
```

Example of Implicit Label Attachment



Text object basepoint

2. The **`CONNECT`** operation passes net name CLK from the METAL1 shape to the POLY shape.

1. The **`CONNECT`** operation implicitly attaches text label CLK to the shape on layer METAL1 because the two objects are on the same layer.

rule file

```
LAYER poly    20      // ORIGINAL LAYER ASSIGNMENT
LAYER metal1  30      // ORIGINAL LAYER ASSIGNMENT
TEXT LAYER    30      // TEXT LAYER SPECIFICATION
CONNECT metal1 poly BY contact
```

STATEMENT
OVERVIEW

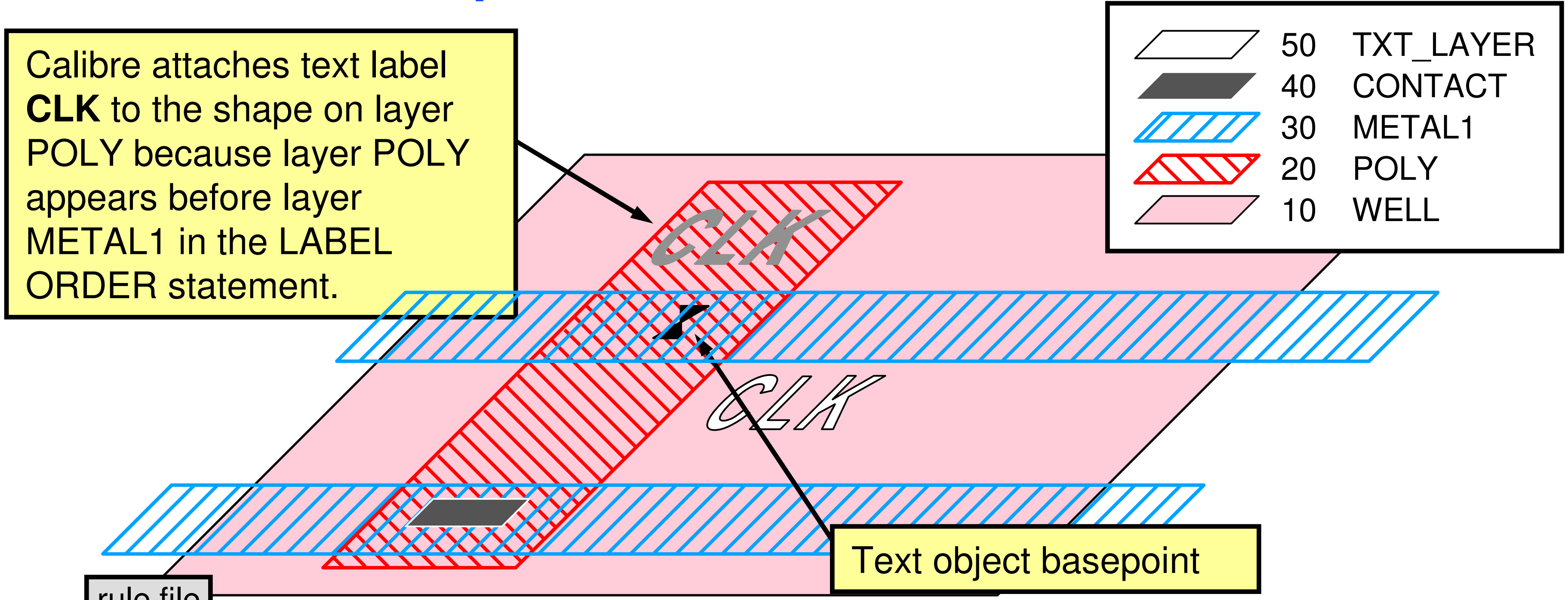
Label Order

CLASS:
SPECIFICATION

Purpose:	Defines free label attachment order during connectivity extraction
Syntax:	`LABEL ORDER` ***`layer`*** `[`***`layer`*** `…]`
Parameters:	***`layer`*** — original layer, derived layer or layer set
Default:	None

- **Defines the order in which connectivity extraction searches layers for an object that intersects a label location**
- **Applies to net names and port objects**
- **Input layers must appear in `(S)Connect` operations**
- **Controls free label attachment (lowest priority)**

Example of Free Label Attachment



Calibre attaches text label **CLK** to the shape on layer POLY because layer POLY appears before layer METAL1 in the LABEL ORDER statement.

▱	50	TXT_LAYER
▰	40	CONTACT
▨	30	METAL1
▨	20	POLY
▱	10	WELL

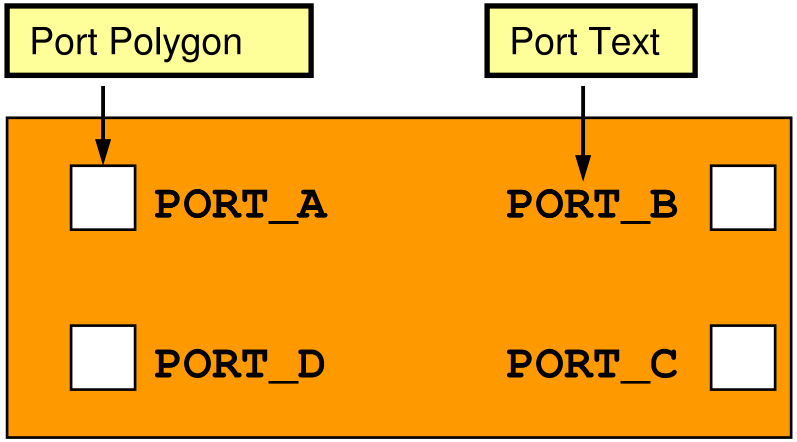
Text object basepoint

rule file

```
LAYER poly        20   // ORIGINAL LAYER ASSIGNMENTS
LAYER metal1      30
LAYER contact    40
LAYER txt_layer  50
TEXT LAYER        50   // TEXT LAYER SPECIFICATION
CONNECT metal1 poly BY contact
LABEL ORDER poly metal1
```

Port Terminology

Port objects:	Port polygons and port text
Port layer:	A layer where geometry or text are recognized as port polygons or text ports
Port naming:	Placing port text into the source or layout database

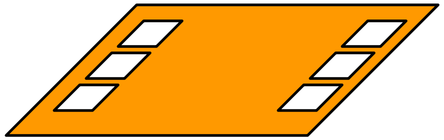


Placing Port Objects into the Source and Layout Databases

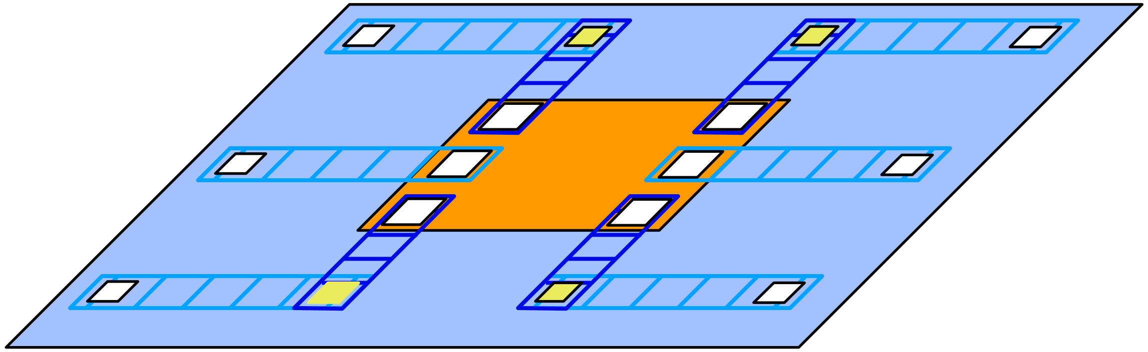
- **If the layout database is GDSII, then define port objects with the rule file.**
- **If the source database is SPICE, then name ports by:**
  - **Naming external nodes of the top-level subcircuit**
  - **Naming nodes specified by the `.GLOBAL` keyword**

How Calibre Distinguishes Between Ports and Pins

This cell has six ports...



until it is placed as ⇩ an instance in a larger cell.



Then the ports become pins.

Port Specification Statements

The following statements control how Calibre recognizes and uses ports:

- `PORT LAYER POLYGON`
- `PORT LAYER TEXT`
- `LVS IGNORE PORTS`
- `LVS CHECK PORT NAMES`

Port Layer Polygon

Purpose:	Treats GDSII and OASIS geometry on the specified layer(s) as port polygons for LVS
Syntax:	`PORT LAYER POLYGON` ***`layer1`*** `[`***`layerN`*** `…]`
Parameters:	***`layer`*** — layer on which Calibre treats polygons as ports
Default:	None
Example:	`PORT LAYER POLYGON 19`

- **Calibre does not require specified layers to be used by other operations.**
- **Calibre does not flag acute, skew or offgrid port objects.**
- **Flat LVS reads top-level port objects only.**

Port Layer Text

Purpose:	Treats GDSII and OASIS text objects on the specified layer(s) as port text for LVS
Syntax:	`PORT LAYER TEXT` ***`layer1`*** `[`***`layerN`*** `…]`
Parameters:	***`layer`*** — layer on which Calibre treats text as port names
Default:	None
Example:	`PORT LAYER TEXT 51`

- **Text objects specified in a `LAYOUT TEXT` statement apply.**
- **Text objects defined with a `TEXT` statement do not apply.**
- **`TEXT LAYER` and `TEXT DEPTH` statements do not apply.**
- **Flat LVS reads top-level port names only.**
- **Texting Hcell ports improves hierarchical LVS performance.**

LVS Ignore Ports

Purpose:	Specifies whether LVS ignores source and layout ports
Syntax:	`LVS IGNORE PORTS {YES|NO}`
Parameters:	`YES` — ports not included for LVS comparison
	`NO` — ports are included for LVS comparison
Default:	`NO`
Example:	`LVS IGNORE PORTS NO`

- **Controls whether ports are used as initial correspondence points.**
- **Only affects top level in hierarchical LVS as Hcell ports are then pins.**
- **Controls whether LVS reports discrepancies involving ports.**
- **Choosing `YES` avoids discrepancies caused by `.GLOBAL` declarations in the SPICE source netlist.**

LVS Check Port Names

Purpose: Specifies whether the tool checks the names of matched ports

Syntax: `LVS CHECK PORT NAMES {NO|YES}`

Parameters:

`NO` — instructs Calibre not to compare names of matched ports

`YES` — instructs Calibre to compare the names of matched ports

Default: `NO`

Example: `LVS CHECK PORT NAMES NO`

When you specify `YES`, the tool verifies that (in the top-level cell) the layout port name matches the corresponding source port name, and reports a discrepancy if no match is made.

Text Case Control Statements

The following statements control text case:

- `LAYOUT CASE`
- `SOURCE CASE`
- `LVS COMPARE CASE`
- `LAYOUT PRESERVE CASE`

Layout Case

Purpose:	Specifies case sensitivity while reading layout database
Syntax:	`LAYOUT CASE {YES|NO}`
Parameters:	`YES` — Calibre treats layout names as case-sensitive `NO` — Calibre treats layout names as case-insensitive
Default:	`NO`
Example:	`LAYOUT CASE YES`

- **Determines relationship between names in layout database when case-sensitivity is important**
- **Applies only to net names, subcircuit names, model names and user-defined names**
- **Does not apply during LVS circuit comparison**
- **Only applies if the `LAYOUT SYSTEM` is SPICE**

Source Case

Purpose:	Specifies case sensitivity while reading source netlist
Syntax:	`SOURCE CASE {YES|NO}`
Parameters:	`YES` — Calibre treats SPICE names as case-sensitive
	`NO` — Calibre treats SPICE names as case-insensitive
Default:	`NO`
Example:	`SOURCE CASE YES`

- **Determines relationship between names in source netlist when case-sensitivity is important**
- **Applies only to net names, subcircuit names, model names and user-defined names**
- **Does not apply during LVS circuit comparison**

STATEMENT OVERVIEW

LVS Compare Case

CLASS: SPECIFICATION

Purpose: Controls case sensitivity for LVS comparisons

Syntax:

```
LVS COMPARE CASE YES|NO[NAMES][TYPES][SUBTYPES][VALUES]
```

Parameters:

`YES` — all comparisons are case sensitive

`NO` — all comparisons are case insensitive

`NAMES` — case sensitive net, instance and port names

`TYPES` — case sensitive component types

`SUBTYPES` — case sensitive component subtypes

`VALUES` — case sensitive string property values

Default: **`NO`**

Example: **`LVS COMPARE CASE YES NAMES TYPES`**

`LAYOUT CASE` and **`SOURCE CASE`** should also be specified as **`YES`** when using **`LVS COMPARE CASE`** or the results could be unexpected.

Layout Preserve Case

Purpose: Specifies whether matching layout net names that differ only by case are treated as identical or different.

Syntax: `LAYOUT PRESERVE CASE {YES|NO}`

Parameters:

YES — layout names must match exactly, including case, to be considered the same.

NO — treats names that differ only by case as identical (default)

The **NET** and **NOT NET** layer operations are case-sensitive when used in conjunction with **LAYOUT PRESERVE CASE YES** specified.

Default: **NO**

Example:

```
LAYOUT PRESERVE CASE YES
// net "abc" and "ABC" are treated as
// two separate nets and both names appear
// in the SPICE netlist
```

What Is LVS Isolate Shorts?

- Finds the shortest path between two texts on the same net.
- A short is defined as one layout net with at least two different attached text names.
- Outputs a DRC-like database of the polygons making up the shortest path.
- Although you access this feature from LVS, it is really a DRC-type feature/function.
- Use this feature with any texted net. (Not limited to power/ground problems.)

LVS Isolate Shorts

Purpose: Specifies whether to perform short-circuit isolation

Syntax: `LVS ISOLATE SHORTS YES|NO [BY LAYER] [NO CONTACTS][{CELL PRIMARY|CELL ALL} [`*`operand`*` NAME `*`names`*`]] [FLAT]`

Parameters:

- `YES|NO` — specifies whether to isolate shorts
- `BY LAYER` — generates output in separate DRC check/short/layer
- `NO CONTACTS`— omits contact layers from the ouptut
- `CELL PRIMARY|CELL ALL` — specifies the hierarchy range
- *`operand`* — operator from the set: `&&` (AND), `||` (OR)
- `NAME` — specifies that a list of net names will follow
- *`names`* — list of text object names
- `FLAT` — forces flat mode execution

Default: `NO`

LVS Isolate Shorts (Cont.)

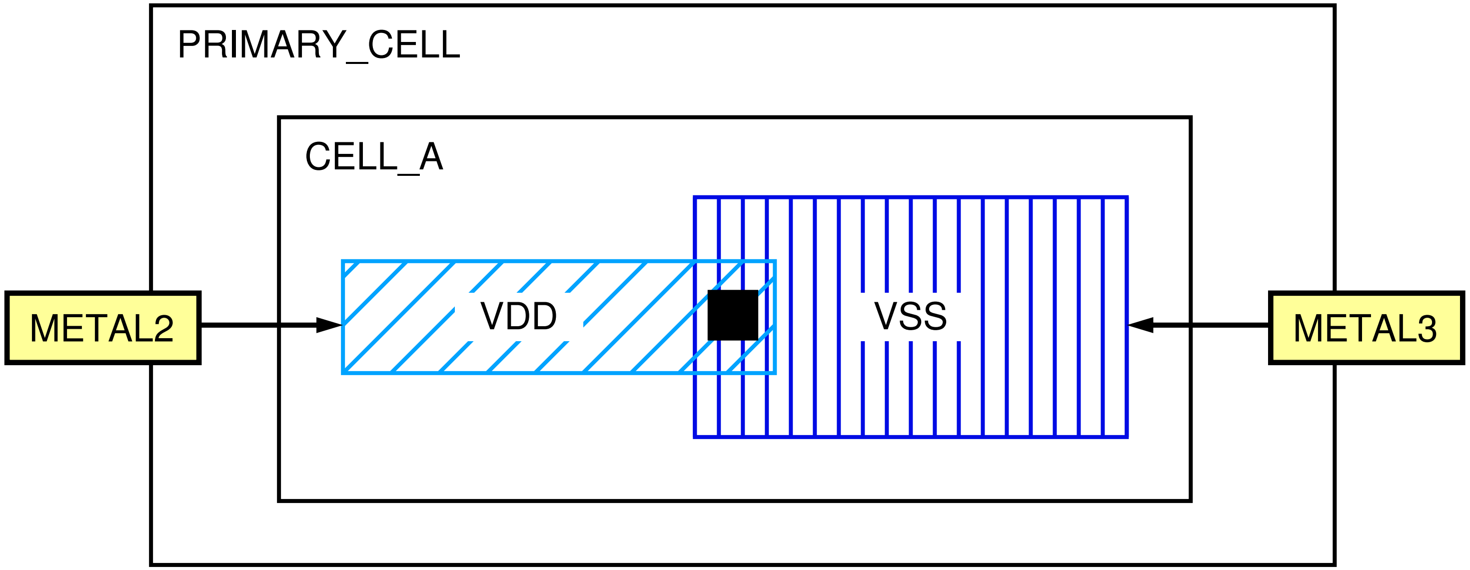
- Layout system must be GDSII, OASIS, ASCII or Binary.
- Hierarchy range subject to `TEXT DEPTH` specification.
- You may specify this statement only once.
- If you specify `BY LAYER`, then results are in the form:

```
SHORT <#>. <net> – <net> – …<net> in <cell> (<layer>)
```

Else:

```
SHORT <#>. <net> – <net> – …<net> in <cell>
```

Example of Using LVS Isolate Shorts



rule file

```
CONNECT metal2 metal3 BY via
TEXT DEPTH PRIMARY   // read text of current cell only
LVS ISOLATE SHORTS YES BY LAYER CELL ALL && NAME VDD VSS
```

short isolation results database

```
SHORT 1. VDD - VSS in CELL_A (METAL2)  // LVS results
```

Calibre Rule Writing

Module 9

Devices

Boolean Operations

- Boolean operations include:
  - `AND`
  - `NOT`
  - `OR`
  - `XOR`
  - `OR EDGE`
- These operations construct layers based upon Boolean logic as applied to sets of points belonging to specified layers.
- `AND` and `NOT` are net-preserving operations passing connectivity information between layers.
- Boolean operations are used to derive new layers used in DRC RuleChecks and device recognition operations.

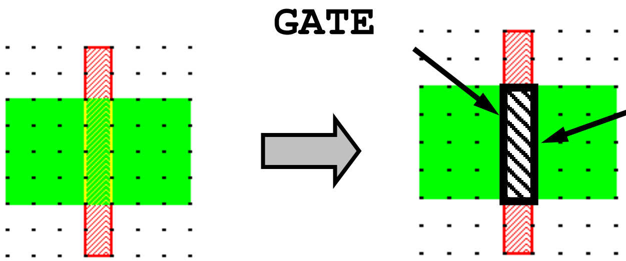
Boolean Operation Examples

POLY 🟥

OXIDE 🟩

METAL1 ▨

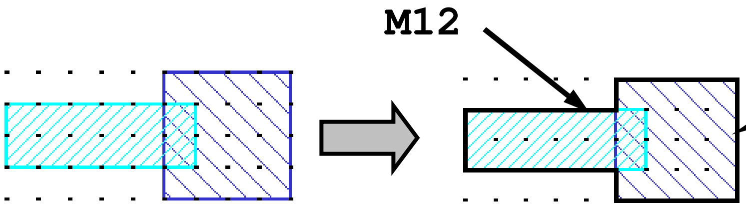
METAL2 ▨



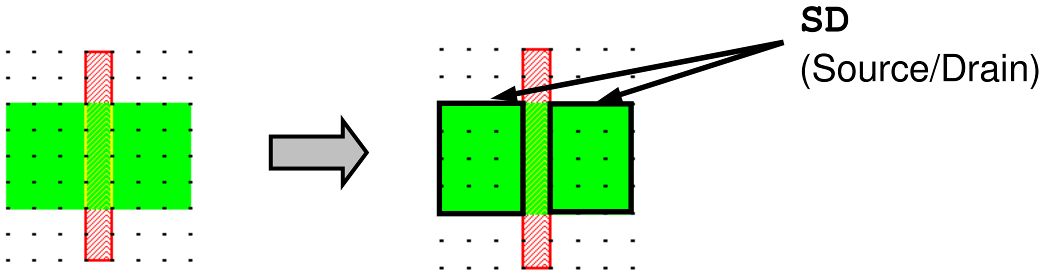
GATE gets the **net ID** of **POLY.**

GATE = POLY AND OXIDE

Overlapping original polygons are merged on derived layer M12 but lose their original net connectivity.

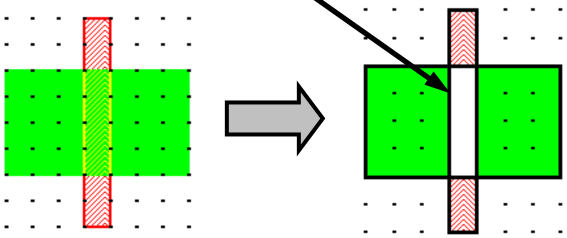


M12 = METAL1 OR METAL2



SD = DIFF NOT POLY

This region is not a part of the NG derived layer.



NG = POLY XOR OXIDE

STATEMENT OVERVIEW

Copy

CLASS: OPERATION

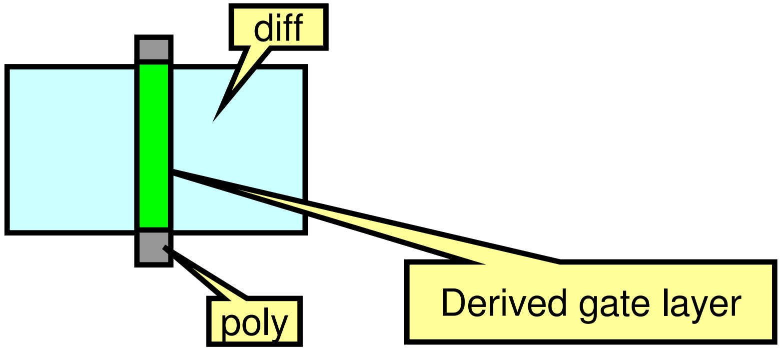
Purpose: Copies ***layer1*** polygons to a derived layer

Syntax: `COPY layer1`

Parameters: ***layer1*** — original or derived polygon or edge **layer**

Example:

```
LAYER DIFF 2
LAYER POLY 4
GATE = POLY AND DIFF
copy_gate{
@copies GATE layer for visual debugging
COPY GATE  // Just copies GATE to output
}
```



NOTE: Use the **COPY** statement in a DRC style RuleCheck and use the rule file in a Calibre DRC run to get output.

- **Creates output that can be viewed in DRC RVE.**
- **The Copy operation is useful in debugging layer derivation.**

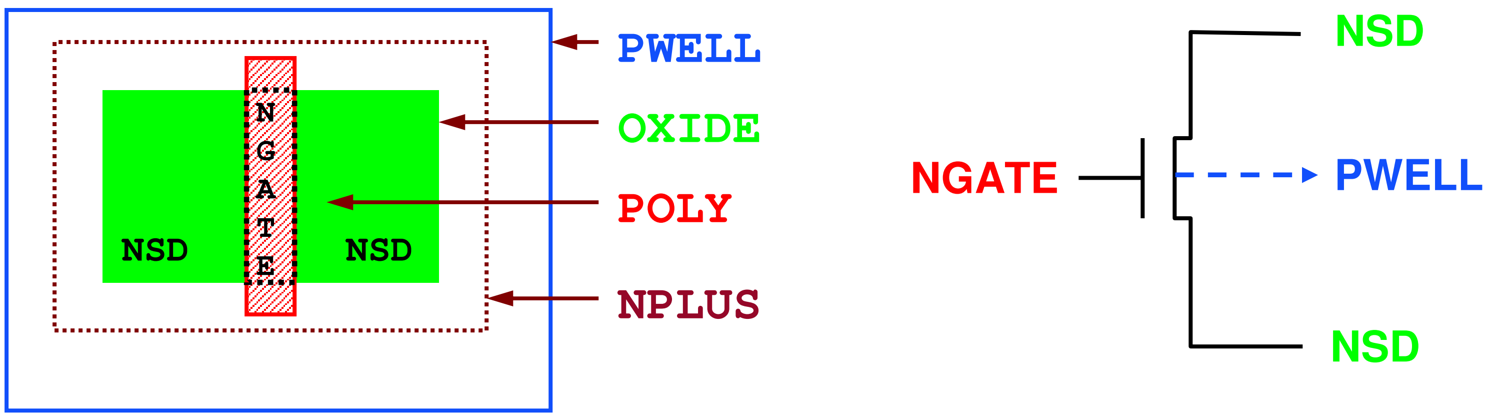
Device Layer Derivation — Example

```
LAYER PWELL                 1
LAYER OXIDE                 2
LAYER POLY                  4
LAYER NPLUS                 5
```

ORIGINAL LAYERS

```
PAREA  = OXIDE AND PWELL
NGATE   = POLY AND PAREA
NOX     = OXIDE AND NPLUS
NSD     = NOX NOT NGATE
```

LAYER CONSTRUCTORS



Device Statement

- Defines a device template for recognizing instances from a union of geometric shapes
- Names and classifies a device
- Specifies device layer, pin layers, and pin swap groups
  - Shapes on the device layer seed the recognition process.
  - Calibre recognizes devices if a shape on each pin layer touches (overlaps or abuts) the shape on the device layer.
  - Pin layer order determines pin name assignment (for built-in devices).
- Specifies parameters for device property calculations

STATEMENT OVERVIEW

Device

CLASS: Device Recognition

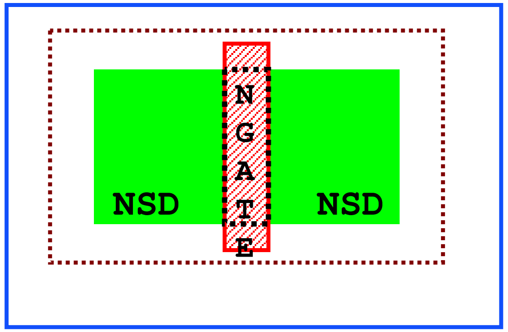
Purpose: Classifies device instances

Syntax:

```
DEVice element_name [(model_name)]
device_layer
{pin_layer [(pin_name)]…}
[<auxiliary_layer> …]
[(swap_list)…]
…
[[property_specification]]
```

Example:

```
DEV MN NGATE NGATE(G) NSD(S) NSD(D) NPLUS(B)
```



Device (Cont.)

Parameters:

`element_name` — specifies the component type

Element name	Definition	Pin names	Default Properties for Tracing	Parameters for Property Specification

`MN`
`MP`
`MD`
`ME`
	MOS Transistor	`G` (gate) 1st pin layer
		`S` (source) 2nd pin layer
		`D` (drain) 3rd pin layer
		`B` (bulk) 4th pin layer is optional
			Width
			Length
				effective_ width_factor (weffect)

`D`	Diode	`POS` (+ pin) 1st pin layer `NEG` (– pin) 2nd pin layer `SUB` (substrate) 3rd pin layer is optional	Area Perimeter	n/a
`C`	Capacitor	`POS` 1st pin layer `NEG` 2nd pin layer `SUB` 3rd pin layer is optional	Capacitance	area_cap, perim_cap
`R`	Resistor	`POS` 1st pin layer `NEG` 2nd pin layer `SUB` 3rd pin layer is optional	Resistance	resistivity
`Q`	Bipolar Transistor	`C` (coll.) 1st pin layer `B` (base) 2nd pin layer `E` (emit.) 3rd pin layer `SUB` 4th pin layer is optional	None	n/a

Device Pin Swapping

- **Pins swappable by default include:**
  - **Device pins with identical layer names (e.g. source and drain pins of MOS regular transistors)**
  - **Resistor pins**
- **Capacitor pins are NOT swappable by default.**
  - **To make pins of all capacitors swappable, rule file must contain statement**

```
LVS ALL CAPACITOR PINS SWAPPABLE YES
```

- **Use pin swap lists in device statements to specify other pin swap options.**

Swap Lists

- The *`swap_list`* parameter specifies groups of pin names that are interchangeable for device recognition purposes. Each pin swap group is of the following form:

  `(`*`pin_name1 pin_name2`*` … `*`pin_nameN`*`)`

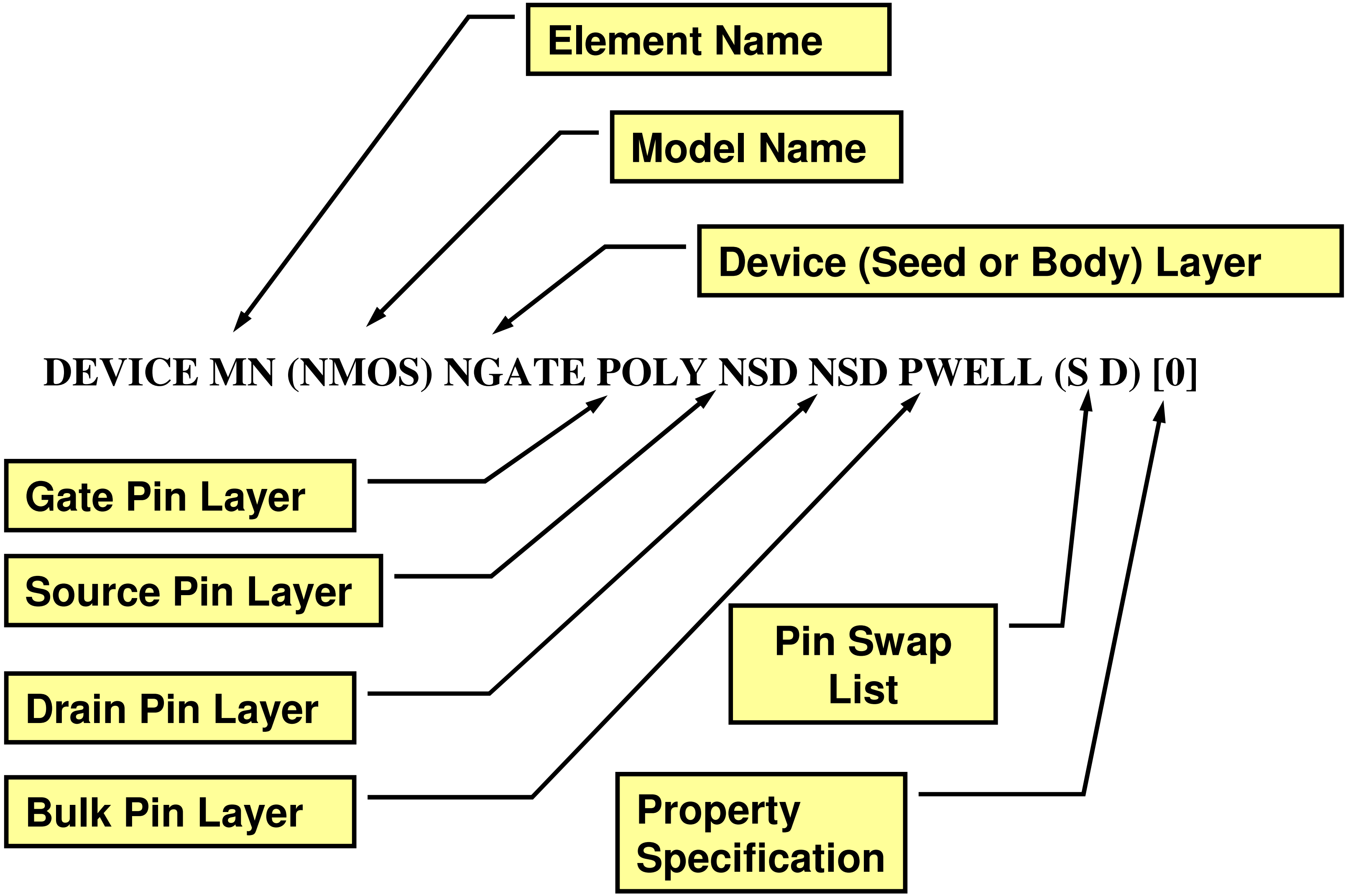
- The pin layers identified by the pin names in a *`swap_list`* are swappable for the purposes of device recognition.

Creating Swap Lists

Here are some rules for creating swap lists:

- By default, two pins are in the same swap group if, and only if, they appear on the same layer. In this case, you do not need to specify a swap list.
- You can specify more than one pin swap group, each in its own parenthesized list.
- If a pin swap group contains one pin name from a given pin layer, then it must contain all pin names from that layer.
- The order of pin names within a swap group is unimportant.

Device Statement — Example



DEVICE MN (NMOS) NGATE POLY NSD NSD PWELL (S D) [0]

Diode Device Example

```
DEV D diode_layer anode(POS) cathode(NEG) <active>
```

- Diode area and perimeter are calculated in square meters and meters respectively by default and are available as properties.
- The built-in algorithm for the calculation of diode area and perimeter is as follows:

```
    [
            property A,P
            A = area(diode_layer)
            P = perim(diode_layer)
    ]
```

- (Module 12 will provide further details on built-in language.)

Capacitor Device Example

```
DEV C (CP) cap_layer anode(POS) cathode(NEG)
       (POS NEG) [300 10]

//CAPACITOR OF MODEL CP WITH SWAPPABLE PINS
//AREA CAPACITANCE SPECIFIED AT 300 pF PER SQUARE um
//PERIMETER CAPACITANCE SPECIFIED AT 10 pF PER um
```

- **Default units:**
  - capacitance in picofarads
  - length in microns
  - area in um^2

- **Area capacitance and perimeter capacitance are user specified (if not specified, both default to 0).**

Bipolar Device Example

```
DEV Q (BJT) BASE COLL(C) BASE(B) EMIT(E)
                    //BIPOLAR TRANSISTOR OF MODEL BJT
```

- ♦ No properties are calculated for these devices by default.
- ♦ User-defined property specifications will be covered later.
- ♦ (Module 12 will provide further details on built-in language).

Resistor Device Example

```
DEVICE R res_layer pos_pin (POS) neg_pin (NEG) [1.1]
//RESISTOR WITH RESISTIVITY SPECIFIED AT 1.1 OHMS PER
//SQUARE
```

- **Ohms is the default unit of resistance in a property specification.**
- **Resistivity is user specified (if not specified, default is 0).**
- **Module 12 will provide further details on built-in language.**

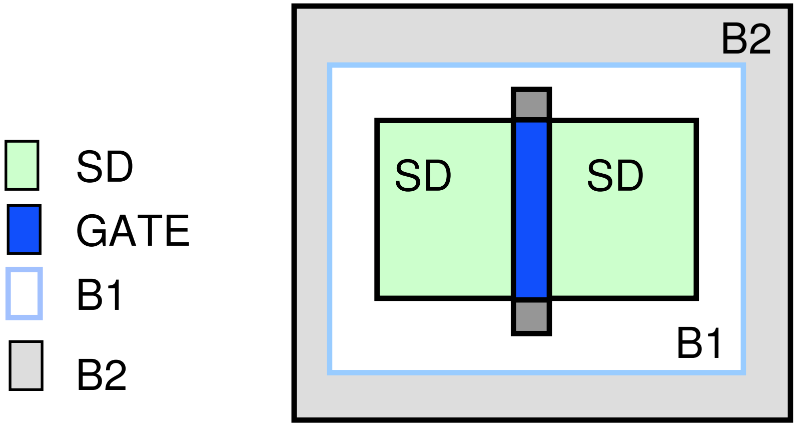
CMOS Device Example

```
DEV MN (NMOS) gate gate(G) diff(S) diff(D) s_pwell(B)
                          //N-TYPE TRANSISTOR OF MODEL NMOS
```

- **Length and width are properties computed for MOS transistors by default.**

User-Defined Devices

- In this example, you need to define pins for an NMOS device with two optional layers (B1 and B2).



- Problem: The built-in NMOS device does not specify all the needed pins.
- Solution: Add non-default pins to the DEVICE statement.

```
DEVICE MN (nmos) GATE GATE SD SD B1 B2 (BULK2)
```

- Notice that the B1 shape becomes the “B” pin as defined by the default MOS model while the B2 shape is a new pin named “BULK2”.

Useful Device SVRF Statements

Two statements helpful during LVS comparisons:

- `LVS FILTER UNUSED OPTION`
- `LVS MAP DEVICE`

LVS Filter Unused Option

Purpose: Controls the filtering process of unused devices.

Syntax: **`LVS FILTER UNUSED OPTION`** ***`option`***
`[`*`option…`*`] [SOURCE LAYOUT|SOURCE|LAYOUT]`

Parameters: ***`option`*** — A required, case-insensitive keyword that specifies various rules to follow for the filtering of unused devices.
(refer to the *SVRF Manual* for option choices)

`SOURCE LAYOUT|SOURCE|LAYOUT` — specifies if the filtering applies to the schematic, the layout, or both

Example: **`LVS FILTER UNUSED OPTION INV SOURCE LAYOUT`**