KLayout

How it came to be and what is under the hood

Matthias Köfferlein, www.klayout.org

Why talk about KLayout?

How do students benefit from KLayout?

Students benefit from KLayout in several ways:

- **Learning real industry tools and formats**: KLayout supports standard layout formats such as GDSII and OASIS, helping students gain experience with tools and data formats used in semiconductor and photonics industries. klayout +1
- **Understanding chip layout concepts**: Students can explore cells, hierarchy, layers, polygons, and physical design principles through hands-on experimentation. klayout +1
- **Developing programming skills**: The Python API allows students to automate layout generation, perform geometric analysis, and create parameterized designs, combining software development with engineering design. klayout +1
- **Supporting research projects**: KLayout can be used to generate custom structures for semiconductor, MEMS, and photonic research, enabling rapid prototyping and reproducible design workflows. klayout +1
- **Free access to professional capabilities**: As an open-source tool, KLayout gives students access to advanced layout editing, visualization, and scripting features without requiring expensive commercial EDA licenses. klayout +1

In short: KLayout helps students bridge the gap between theory and practice by allowing them to design, analyze, and automate real-world microelectronics and photonics layouts. klayout +1

About Me

- An age 59 engineer with ~30 years experience in the semiconductor business
- Spent most of my career in the Tapeout business at the technology/design interface

How did it happen?

Like many Open Source Projects it started with gripes and some spare time

Here is what Guido van Rossum writes about the origin of Python:

> During the 1989 Christmas holidays, I had a lot of time on my hand, so I decided to give it a try. During the next year, while still mostly working on it in my own time, Python was used in the Amoeba project with increasing success, and the feedback from colleagues made me add many early improvements.

Once Upon a Time ...

Late 90s

- GDS2 files were all around, but there was no really good solution to view or edit them
  - Tools were expensive with strange UIs and slow
- Most people would stream them into Virtuoso to view or edit them
  - Well, you get used to that

A GDS2 (GDSII)

A Calma Graphical Data System

Where “Tapeout” comes from

Prehistoric I/O

An actual “Tapeout” :)

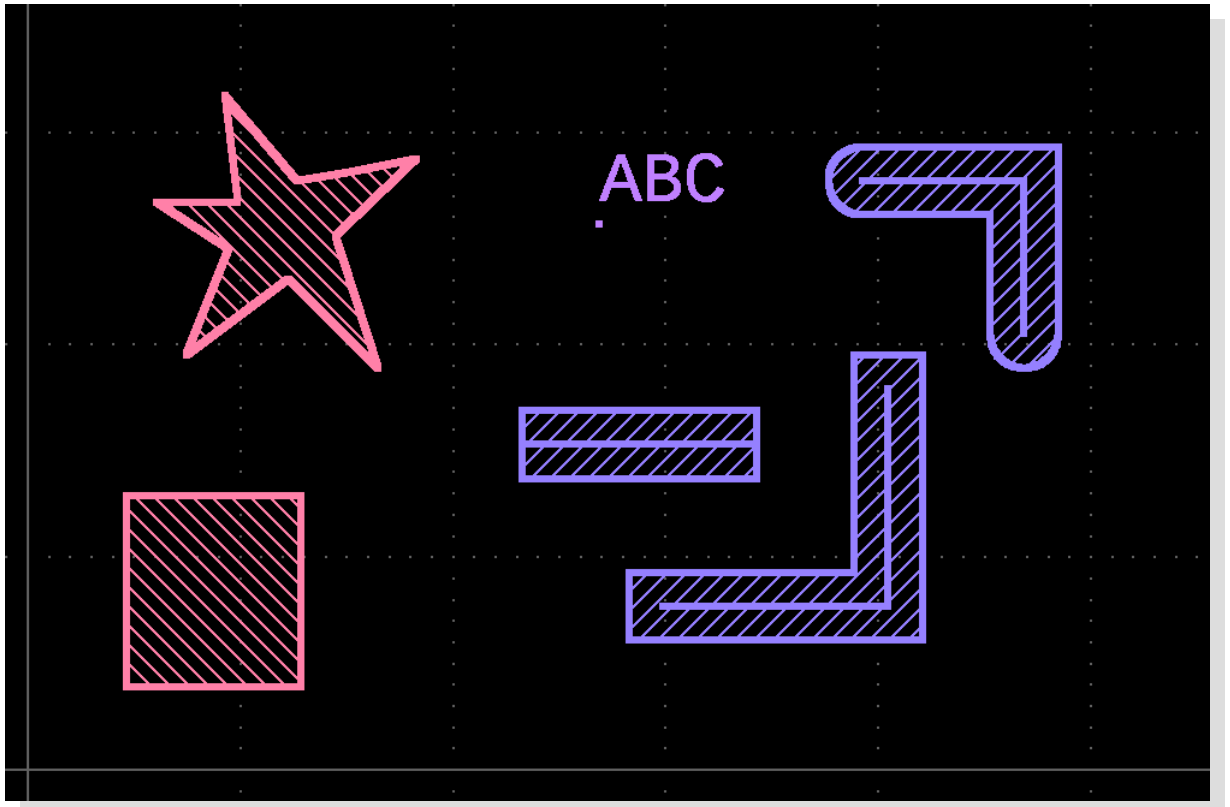
GDS2 – aged, but loved

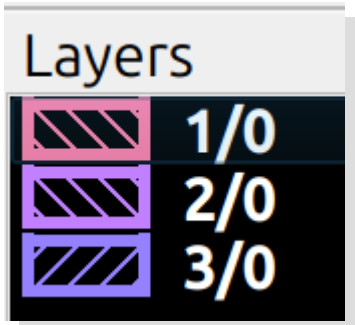
- Simple & easy to implement
- Scales well
- Fairly efficient
- “Good enough”
- Limitations are accepted practice

GDS2 – in a nutshell

- Simple shapes: polygons, paths, labels
- Simple layer system: layer #, datatype #
- Hierarchy by cell / reference
- Basic grid (database unit) and 32bit coordinates
- Additive composition model

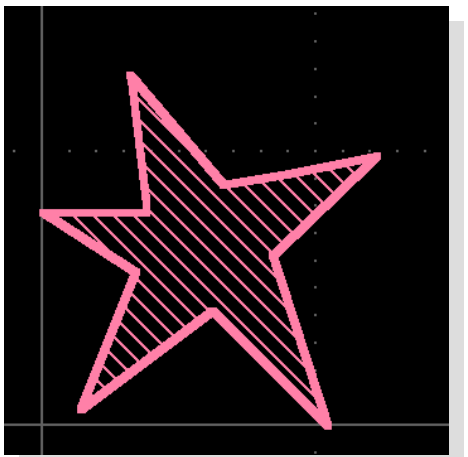
GDS2 - Shapes

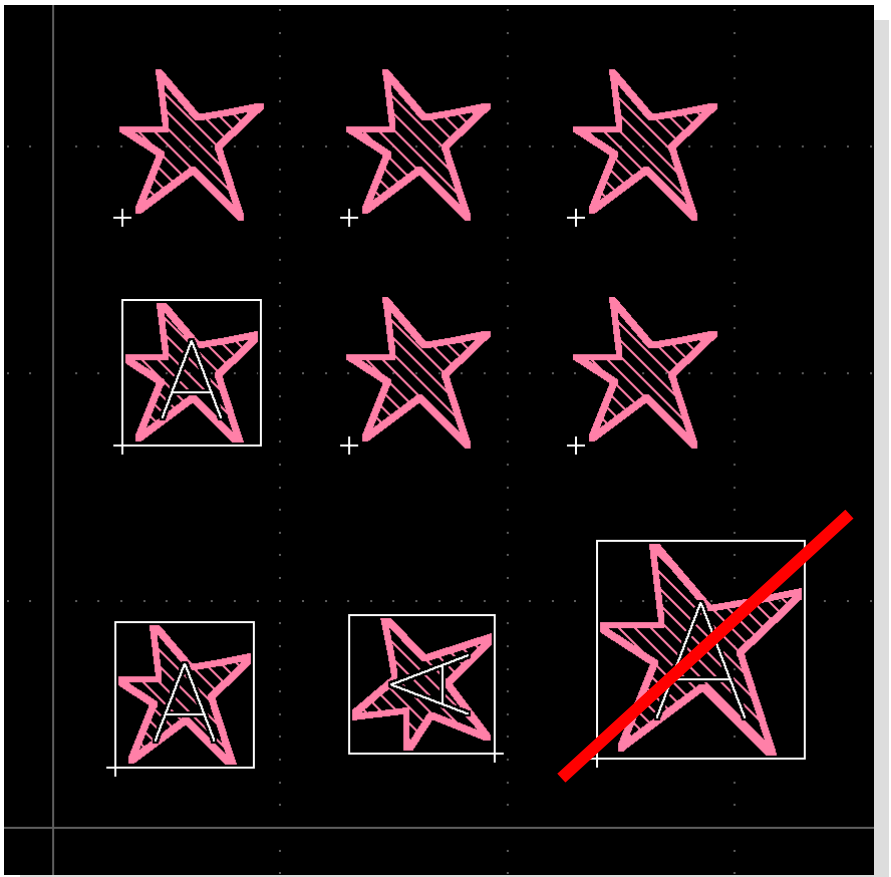




GDS2 – Cells and References

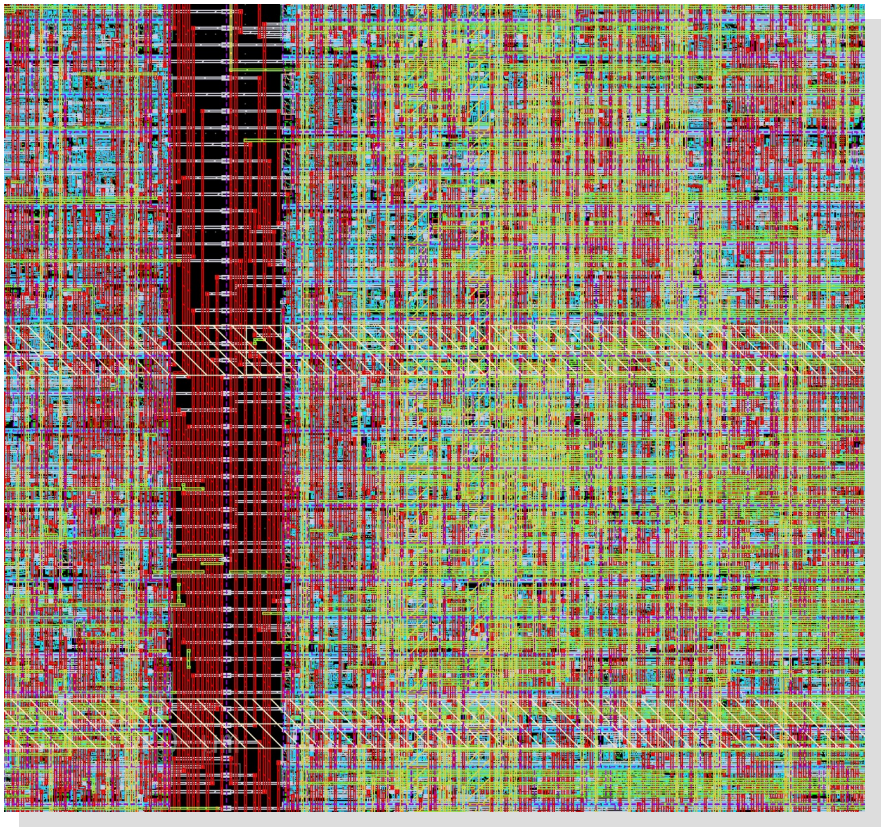
Cell “A”





GDS2 – That's all …

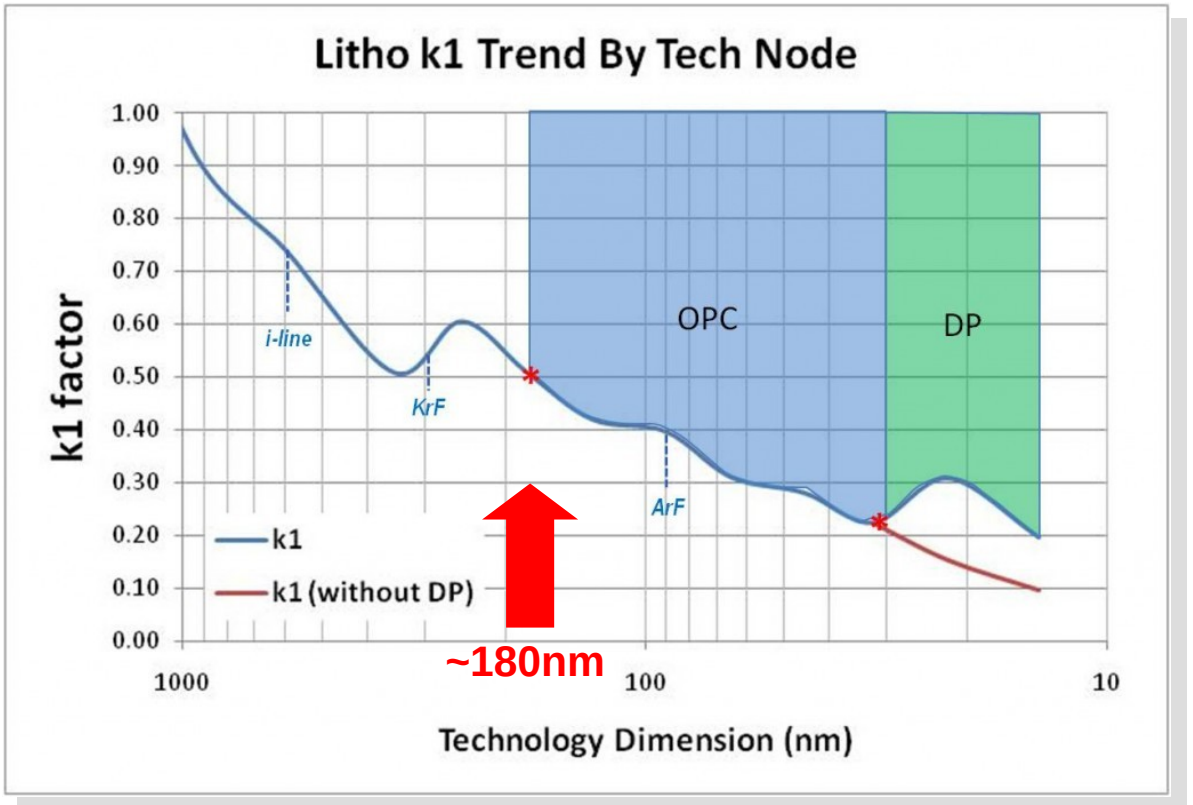
Made from 3 shape types, cells and references



Then came a new millennium and OPC ...

What happened?

RET: Resolution enhancement techniques



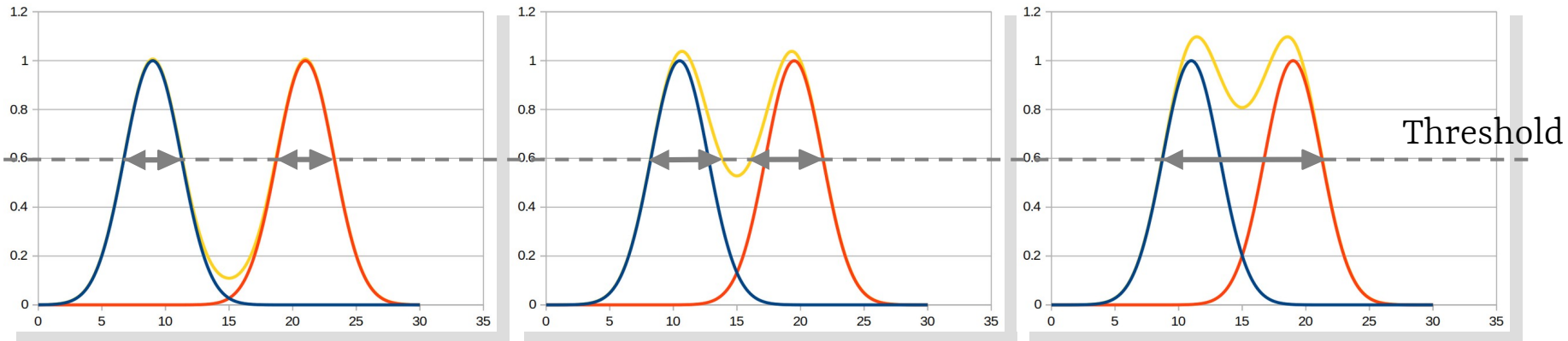
Resolution of a lithography system (Rayleigh criterion)

$$R = k_1 \frac{\lambda}{NA}$$

k_1 needs to go down:
- off-axis illumination
- resist technology
- phase-shift masks
- immersion
- **OPC**
- **Assist features**

The proximity effect

The lens of the imaging system acts as a low-pass filter.
Combining the intensities of two clear lines gives the yellow curve.
The resist opens when the exposure reaches a threshold dose.



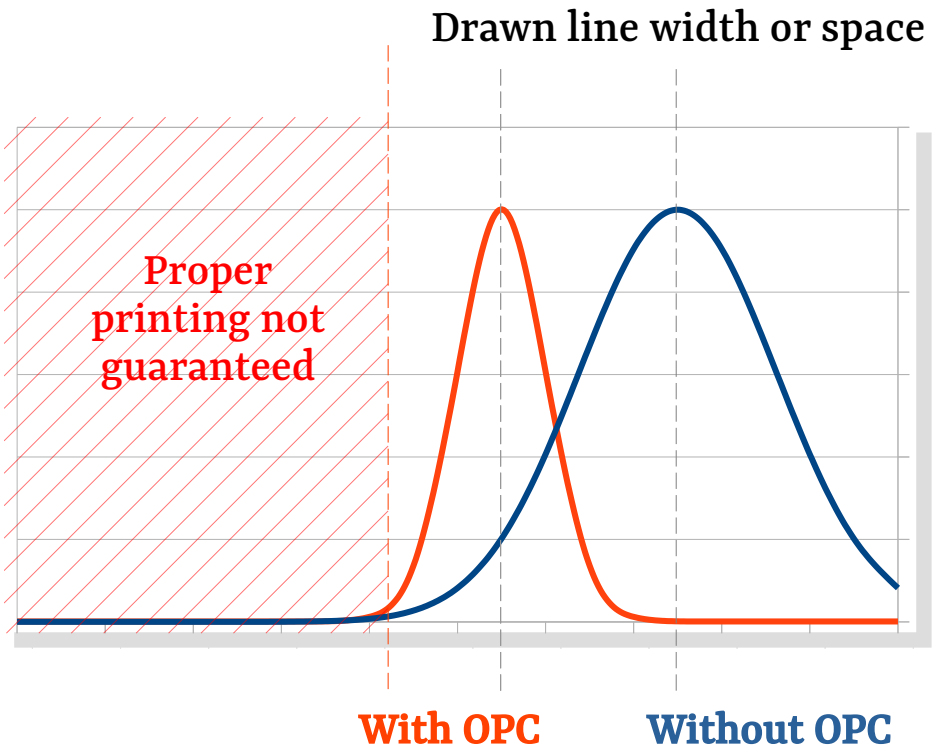
Isolated lines

Dense lines → line widening, space reduced

Bridging → lines cannot be resolved

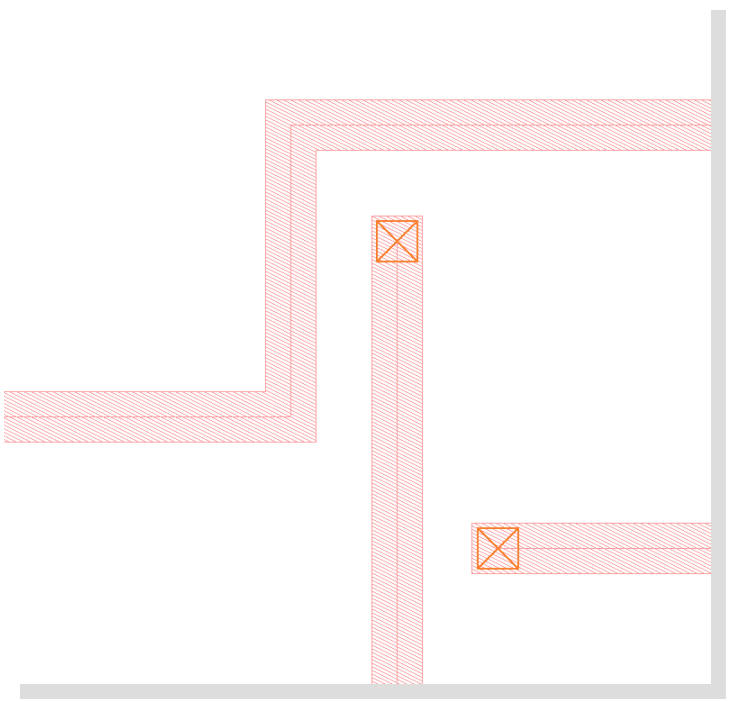
Why is OPC required?

OPC improves control over the feature variations

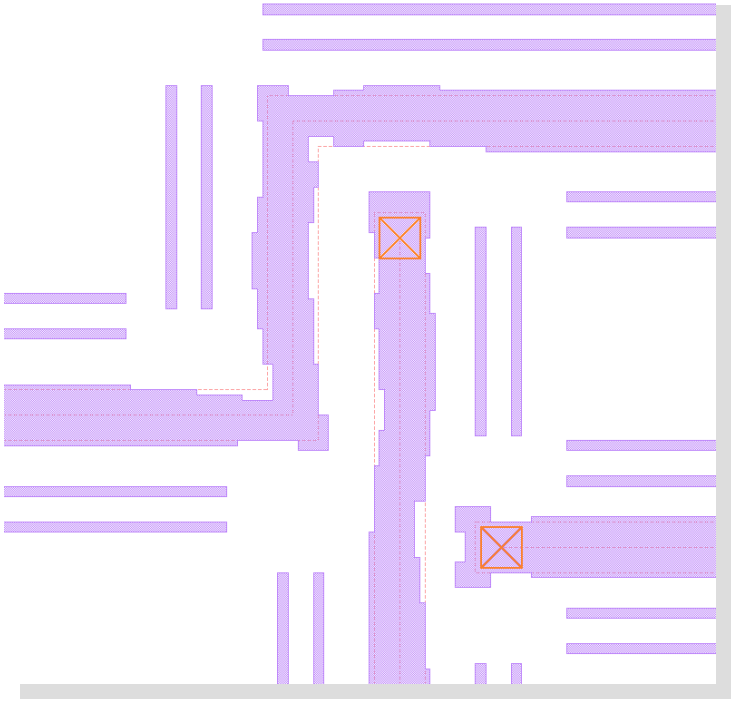


Line width or space variation due to proximity effect under different neighborhood configurations and process conditions

OPC increases data complexity

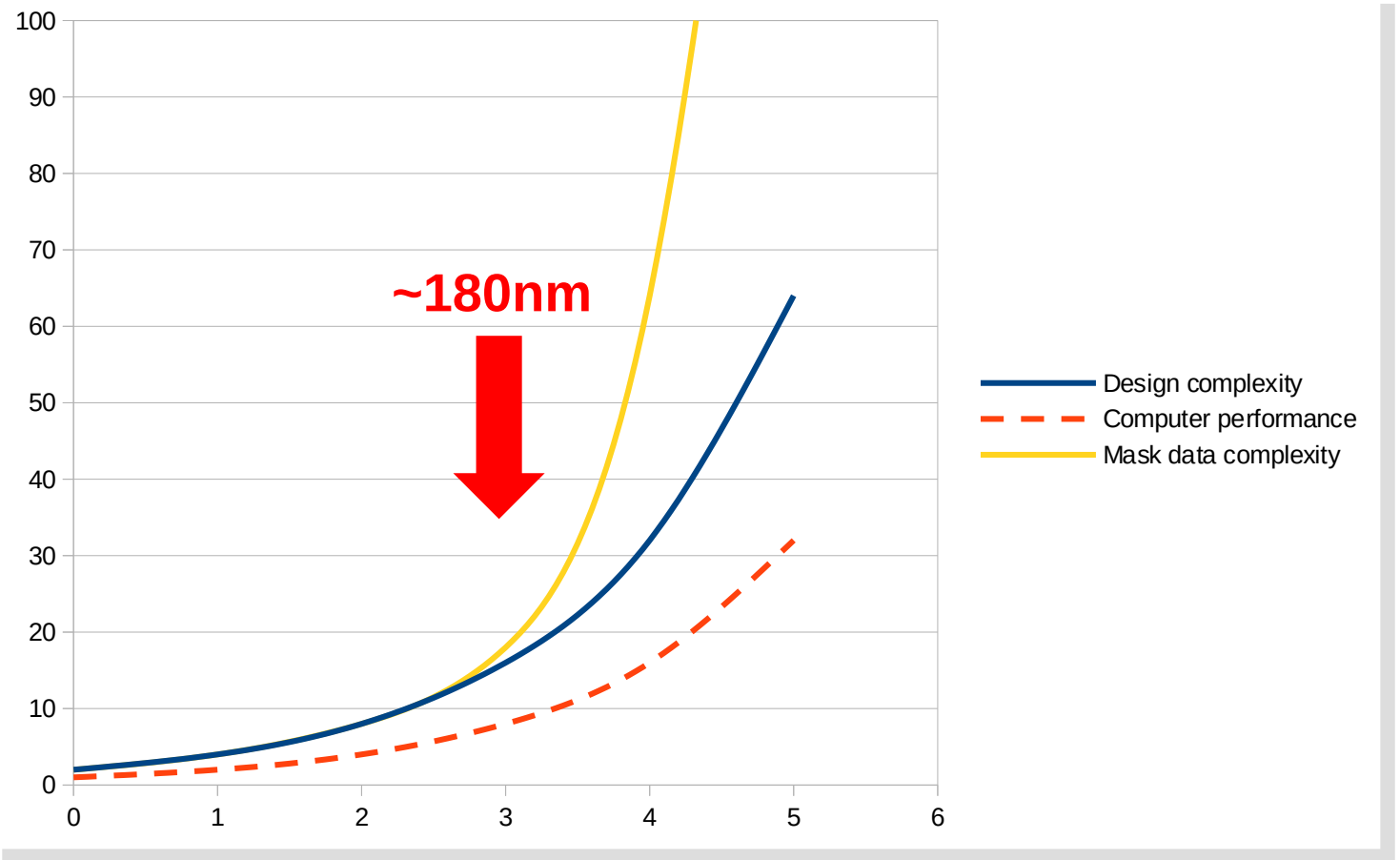


Drawn



On Mask

Mask data volume explosion



A Solution was needed

- We got a new data format: OASIS
  - Comes with shape arrays and internal compression: much smaller files

```
361322674 Jul  7 22:52 sunrise_top_padring_sealring_wfill.gds
   849813 Jul 25 19:04 sunrise_top_padring_sealring_wfill.oas
```

  - Variable-length coordinates
  - Named layers, extended property system, unlimited polygon size, layer numbers, text length ...
- New viewers and editors were required

My layout tool wish list

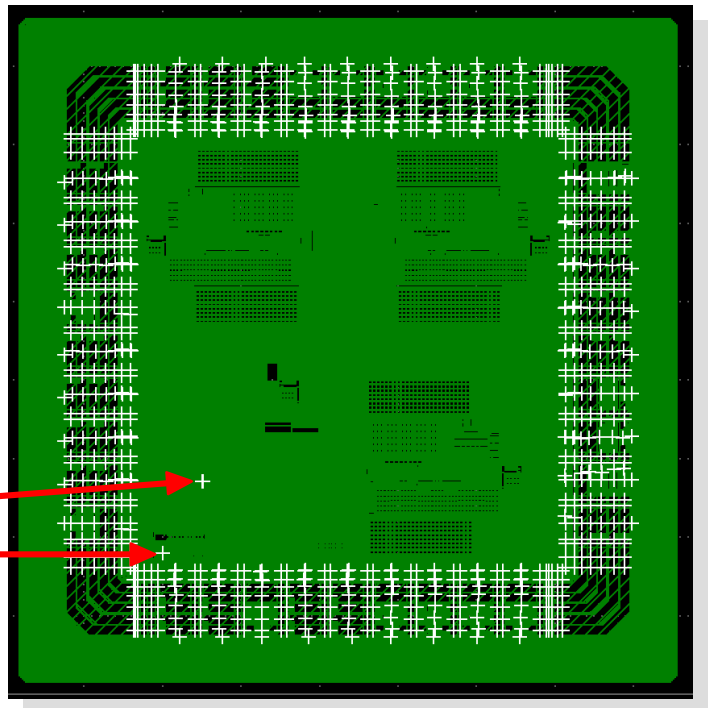
- Runs everywhere: Windows, Unix, consumer hardware ..
- Fast zoom and pan in detail view
- Non-blocking drawing thread
- Simple and modern UI
- High display quality
- All-in-one tool

The display quality issue

Existing viewers tended to “optimize” away objects

“I often need to find the needle in the haystack”

Can you spot these with your viewer?



There was the problem, an idea and some time

The History of KLayout

I guess that's the birth of it

```
67 ------------------------------------------------------------------------
68 r255 | matthias | 2006-11-13 20:37:06 +0100 (Mo, 13 Nov 2006) | 3 lines
69
70 Renamed koeview to klayout
71
72
```

20 years!

OMG!

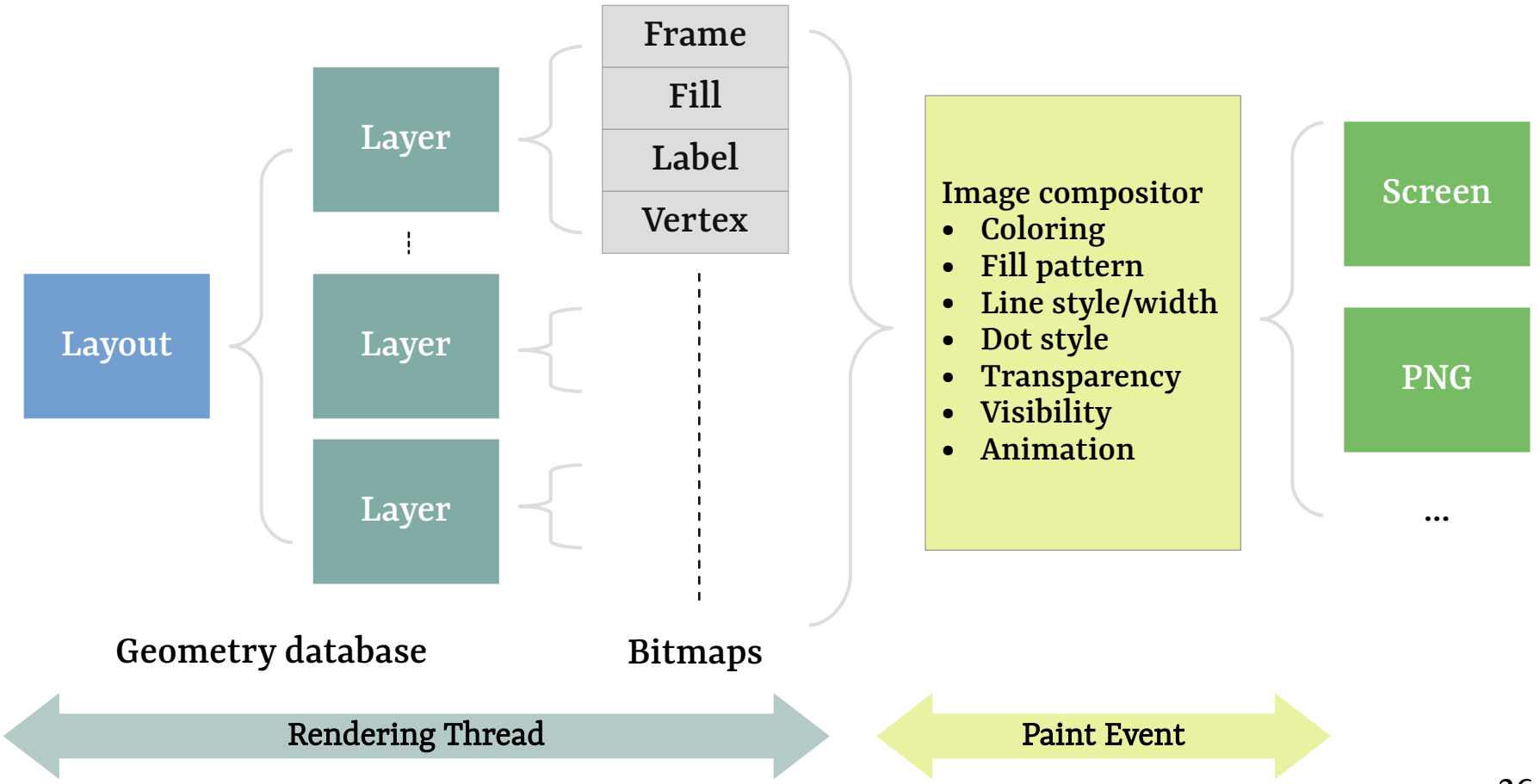
What else happened 2006?

- Pluto was declared a dwarf planet
- Twitter was launched
- The FIFA World Cup was held in Germany
- World population was 6.6 billion (now: 8.3)
- US president was George W. Bush

The License

- GPL was chosen initially because I picked Qt for the UI toolkit
  - Using Qt for free required me to choose GPL
- But I personally favor GPL
  - It's a clean deal with maximum security for the users
  - Not ideal for growing a developer community – but again, it's a clean deal (e.g. does not need contributor agreements)
  - I made a clear cut at the scripting layer to enable integration

Idea: KLayout's Rendering Engine



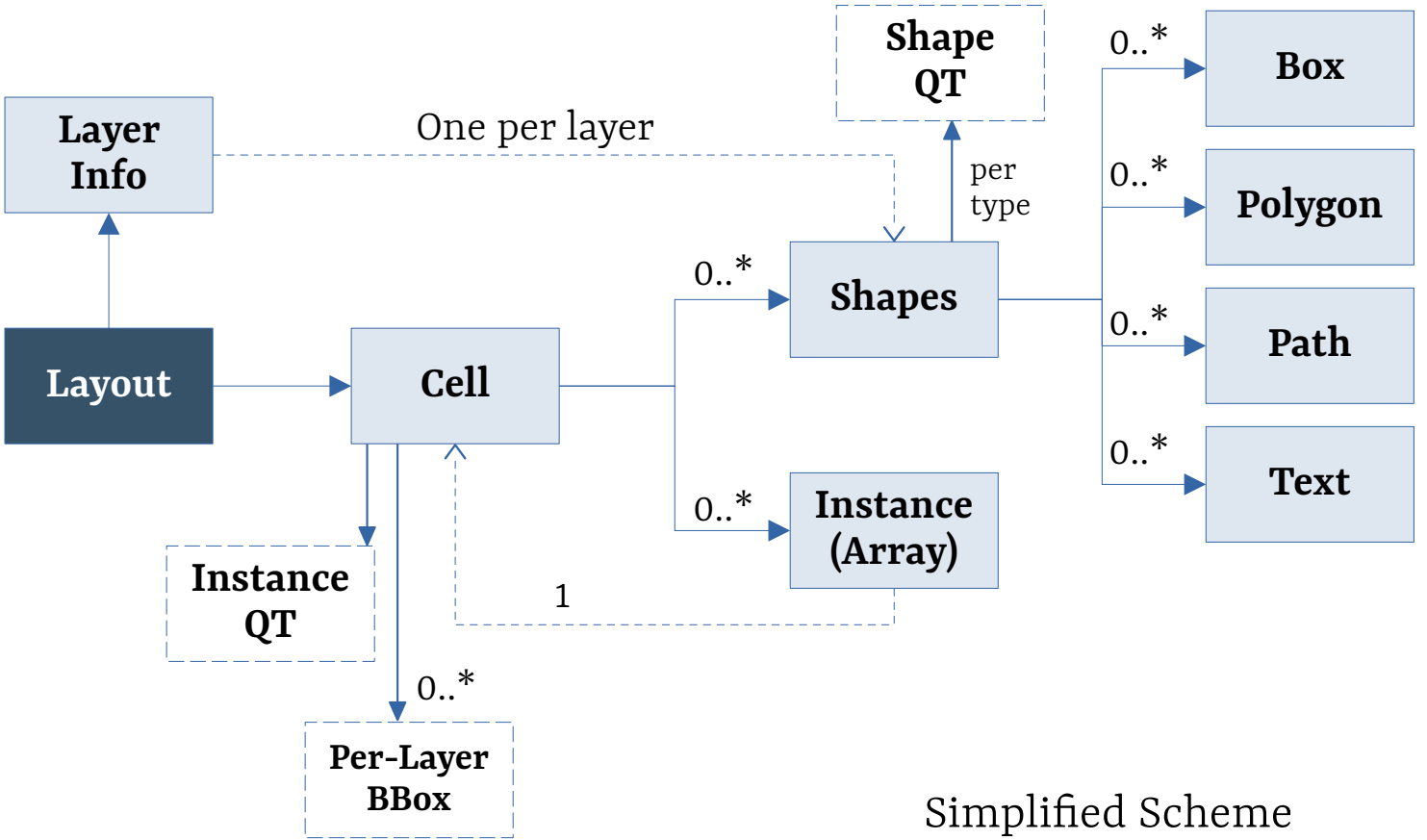
Why is it fast?

- Setting single pixels in a bitmap is extremely fast (ns)
- A small polygon can be reduced to one pixel
- A small non-empty cell can be reduced to one pixel
- A small non-empty quad can be reduced to one pixel
- A small dense array (of cells, shapes) can be reduced to few pixels
- Switching visibility or styles is fast (no redraw needed)
- **AND: No GPU required :)**

Why is it accurate?

- The key to accuracy are per-layer bounding boxes
  - These keep information about “emptyness”
  - Every layer can optimize independently
- The drawing optimization allows hierarchy traversal shortcuts without quality compromises
  - Whole cells can be reduced to a single pixel while maintaining accuracy
- Quads allow tile skipping without loss of accuracy

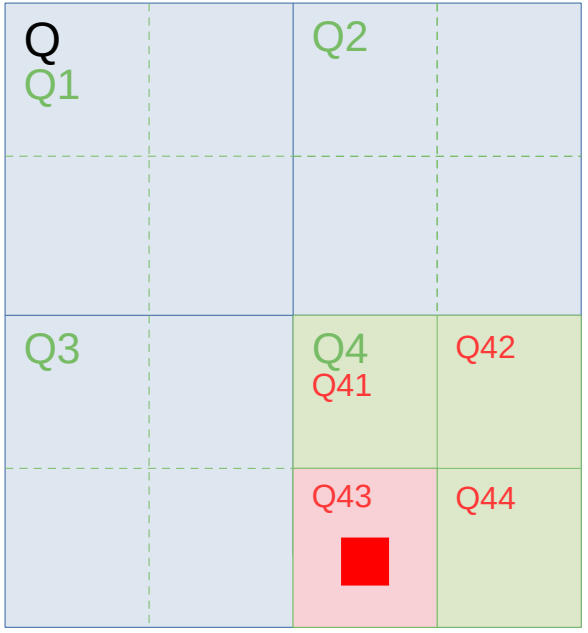
The Geometry Database

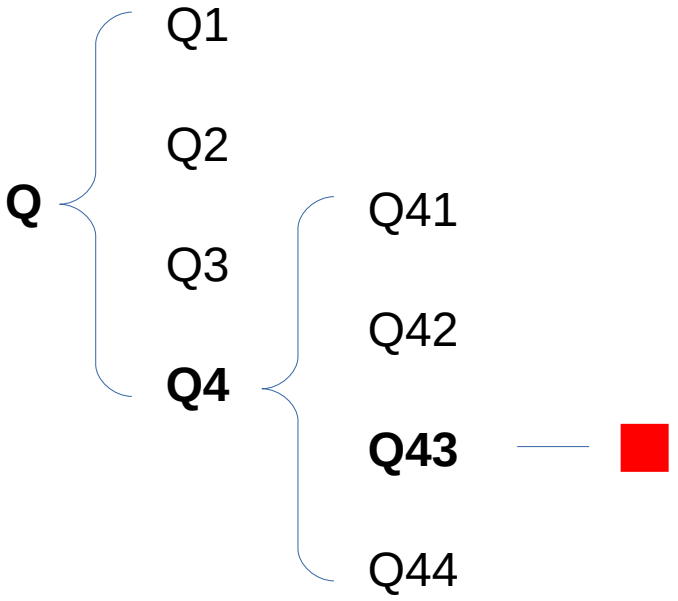


Simplified Scheme

Quad Trees

- Bin objects into recursively divided “quads”
- For efficient region queries and empty tile optimizations





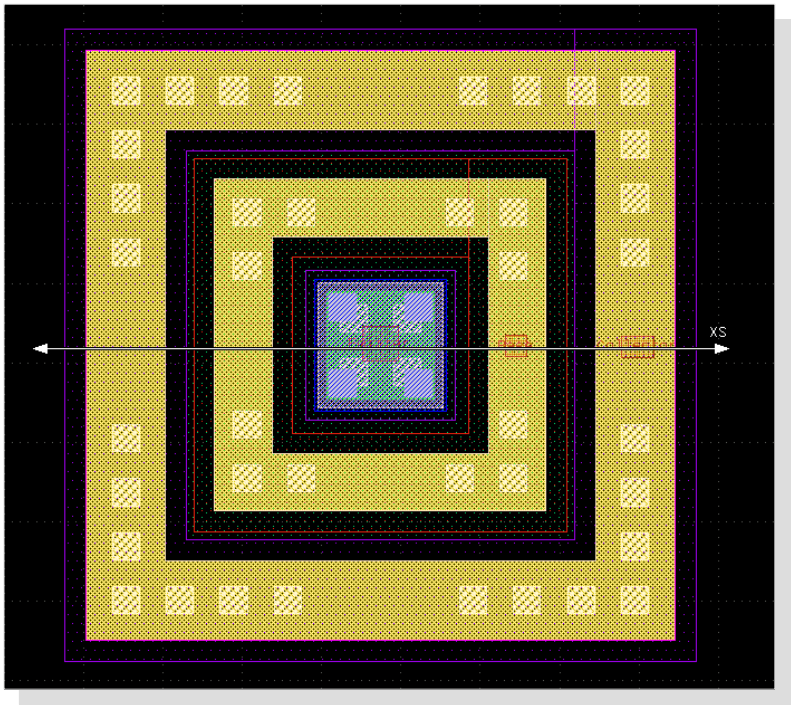
Wow .. what else can we do?

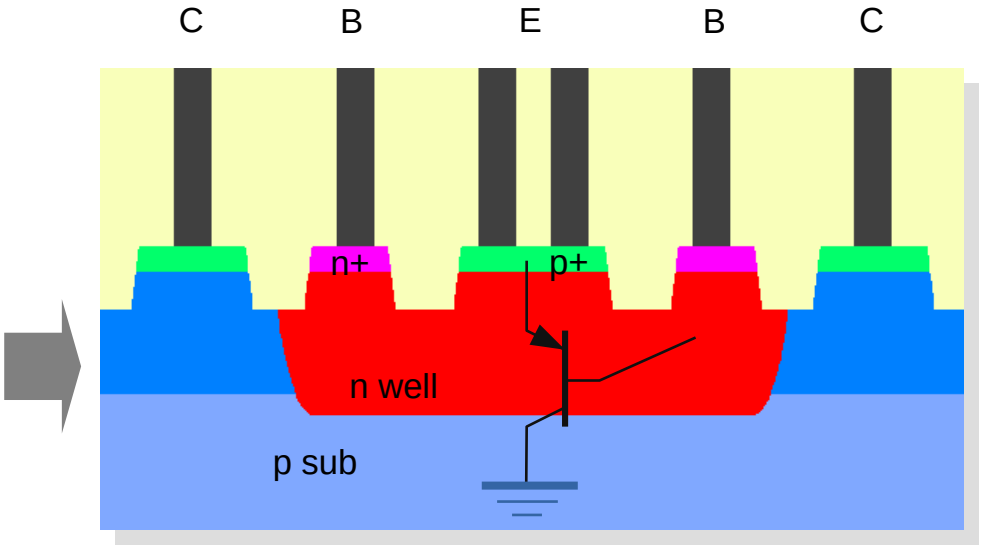
- Editing
- Script API
- PCells (parametric cells)
- More formats
- Rulers, Overlay images ...
- Tools: Clip, XOR, Search & Replace, ...
- Verification: DRC, LVS, Antenna, Density ...
- ...

i.e. XSection ...

PNP BJT pnp_w0p68_l0p68 from

https://github.com/mabrains/sky130_klayout_pdk





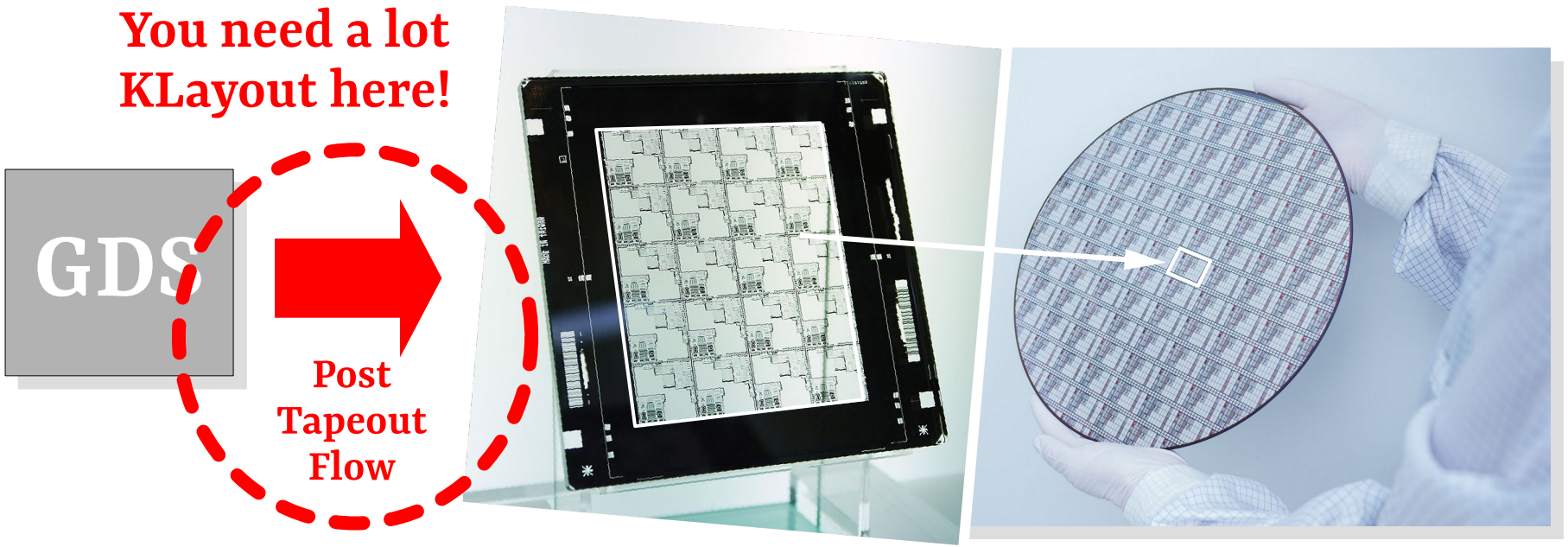
Collector is tied to substrate!
This device is useful for bandgap references for example


20 YEARS LATER
...

“KLayout is everywhere”

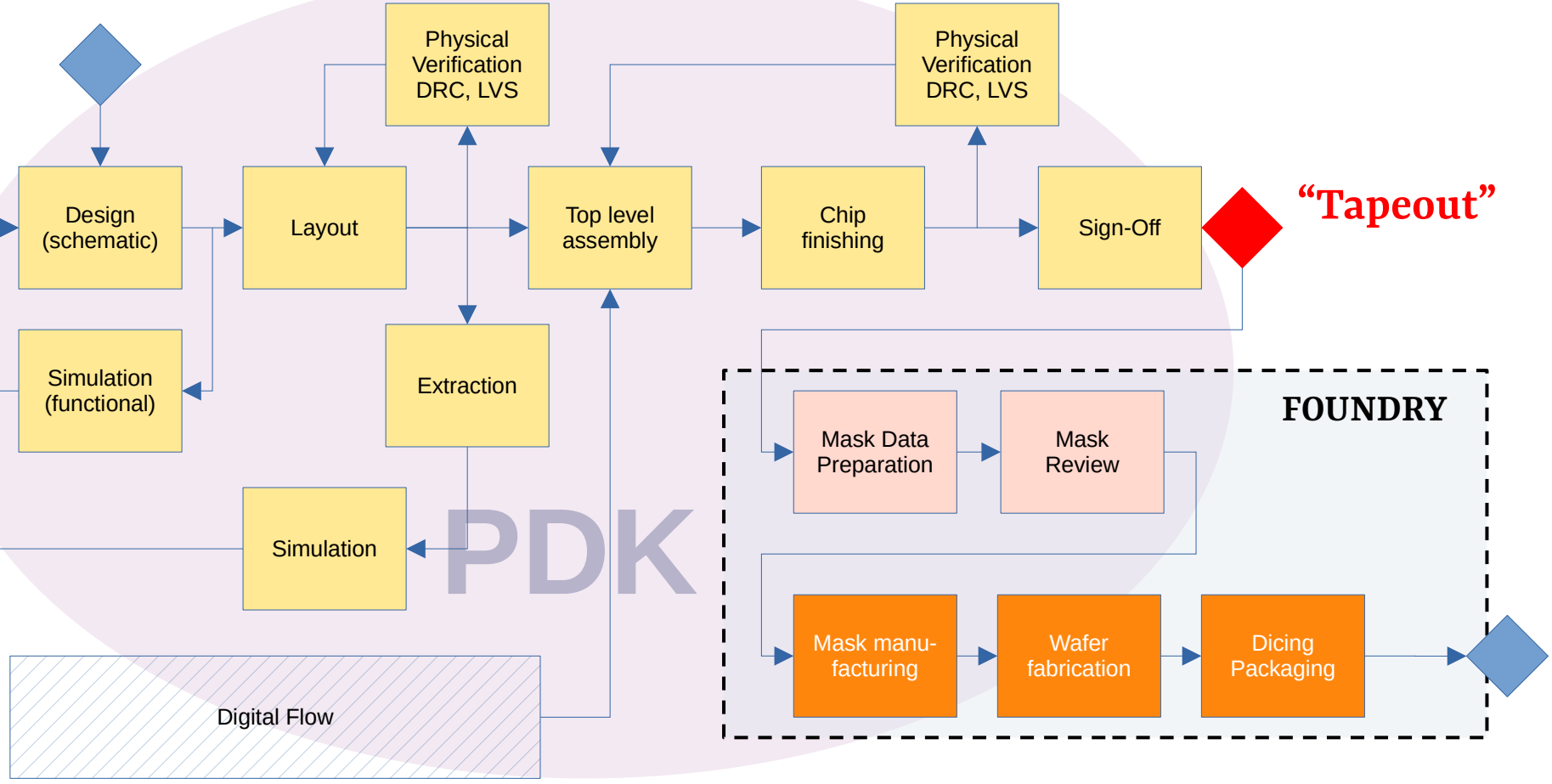
- Semiconductor companies, design houses, tool suppliers and foundries use it
- It’s a daily companion for process/product/layout engineers
- Still the only tool available on Windows
- Commercial systems have adopted KLayout for specific purposes
- A plus in your CV skills section :)

“I’m my own best user”

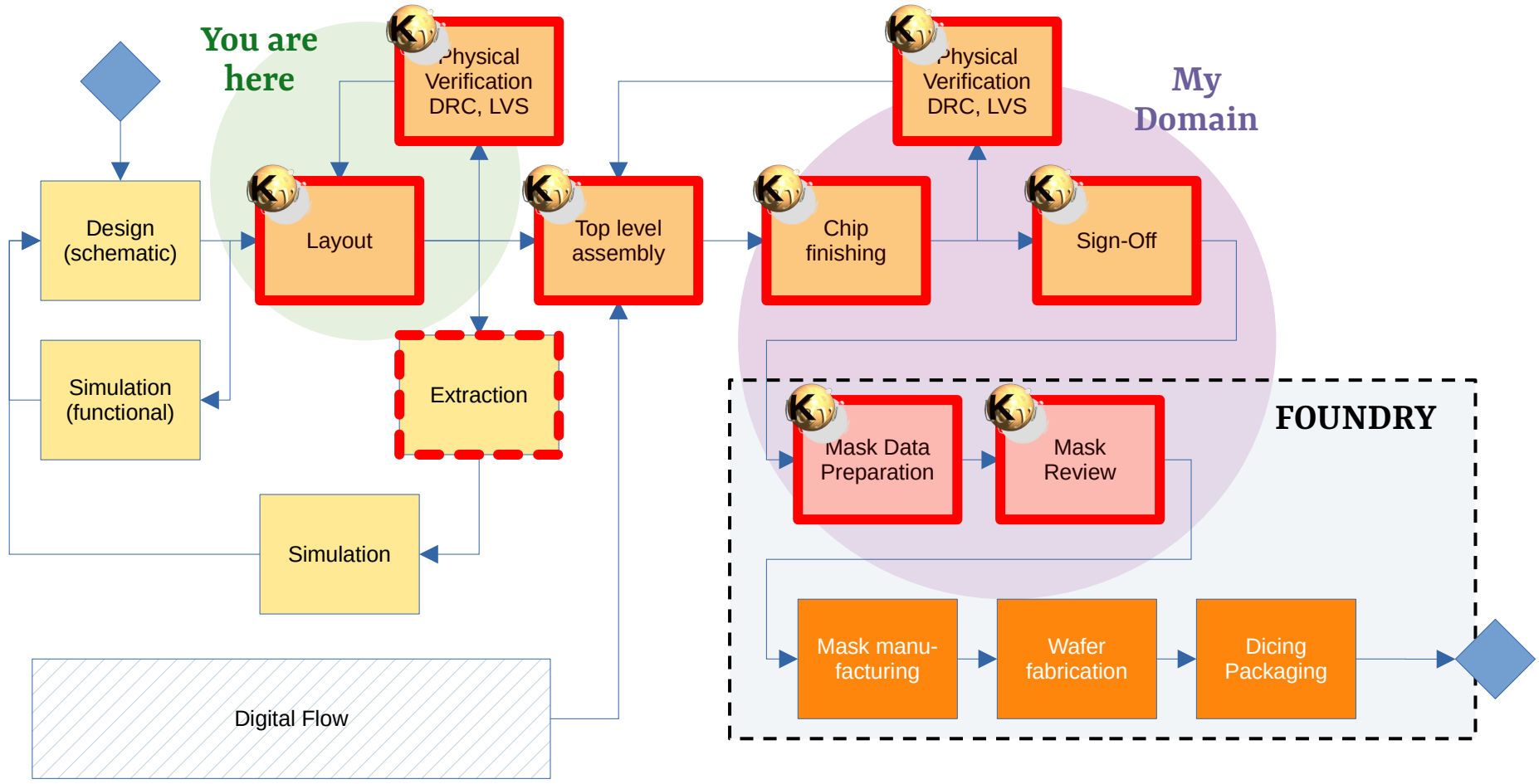


Let’s see what KLayout can do for you today …

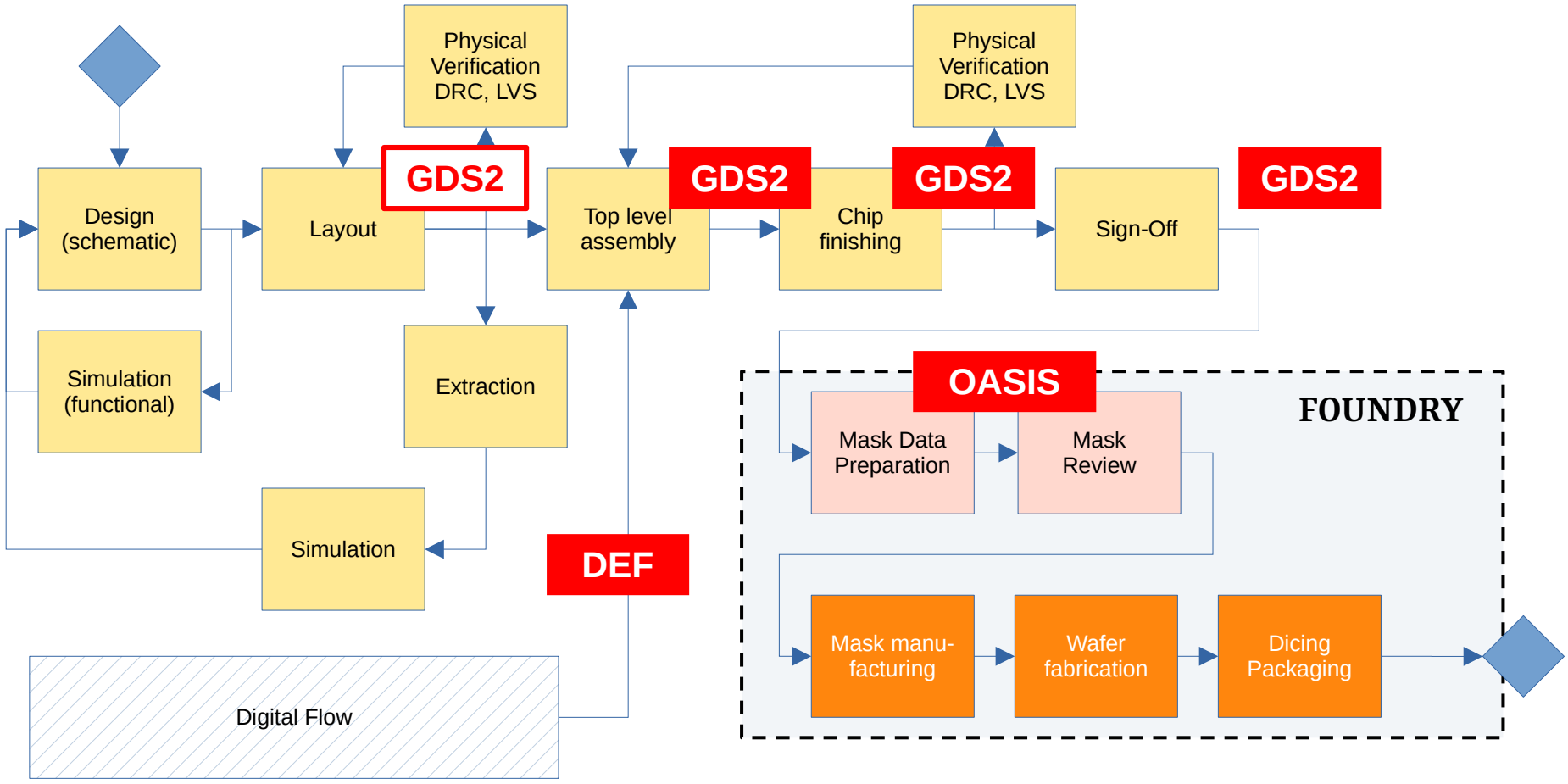
The (nonlinear) path from idea to silicon



The (nonlinear) path from idea to silicon



The (nonlinear) path from idea to silicon



My Favorite Features

- Script API
- PCells
- Built-In (simple) IDE
- Custom queries
- XOR feature
- (Hierarchical) DRC/LVS engine
- XSection (with disclaimer!)

Script API

- KLayout embeds both a Python and Ruby interpreter
  - AFAIK that is a unique feature
- API is the same thanks to a generic dispatcher layer
  - Python and Ruby interpreters can access the same object through different channels
- For external applications there is the klayout Python module
  - See https://pypi.org/project/klayout/

More later ...

PCells

- KLayout PCells are written in Python or Ruby
- They are based on classes and interfaces
- Can utilize the full power of the geometry API
- Have some nice features you don't find in commercial tools, like
  - Embedded copies
  - Guiding shapes

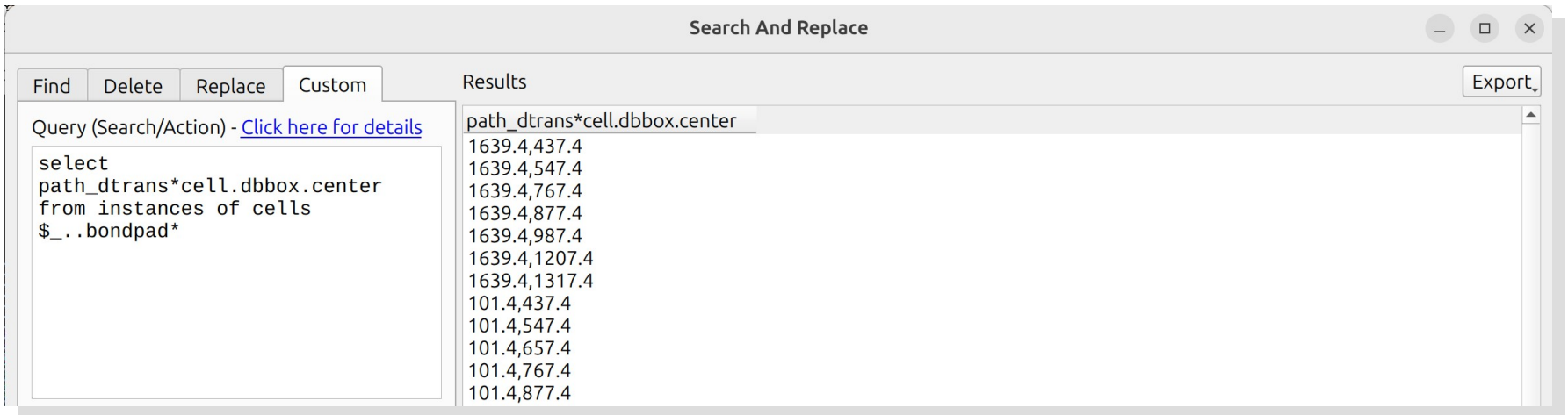
More later ...

Built-In (simple) IDE

- It’s not PyCharm or VS Code
- But it’s good enough to write and debug small ad-hoc scripts
- Integrates with the application

Custom queries

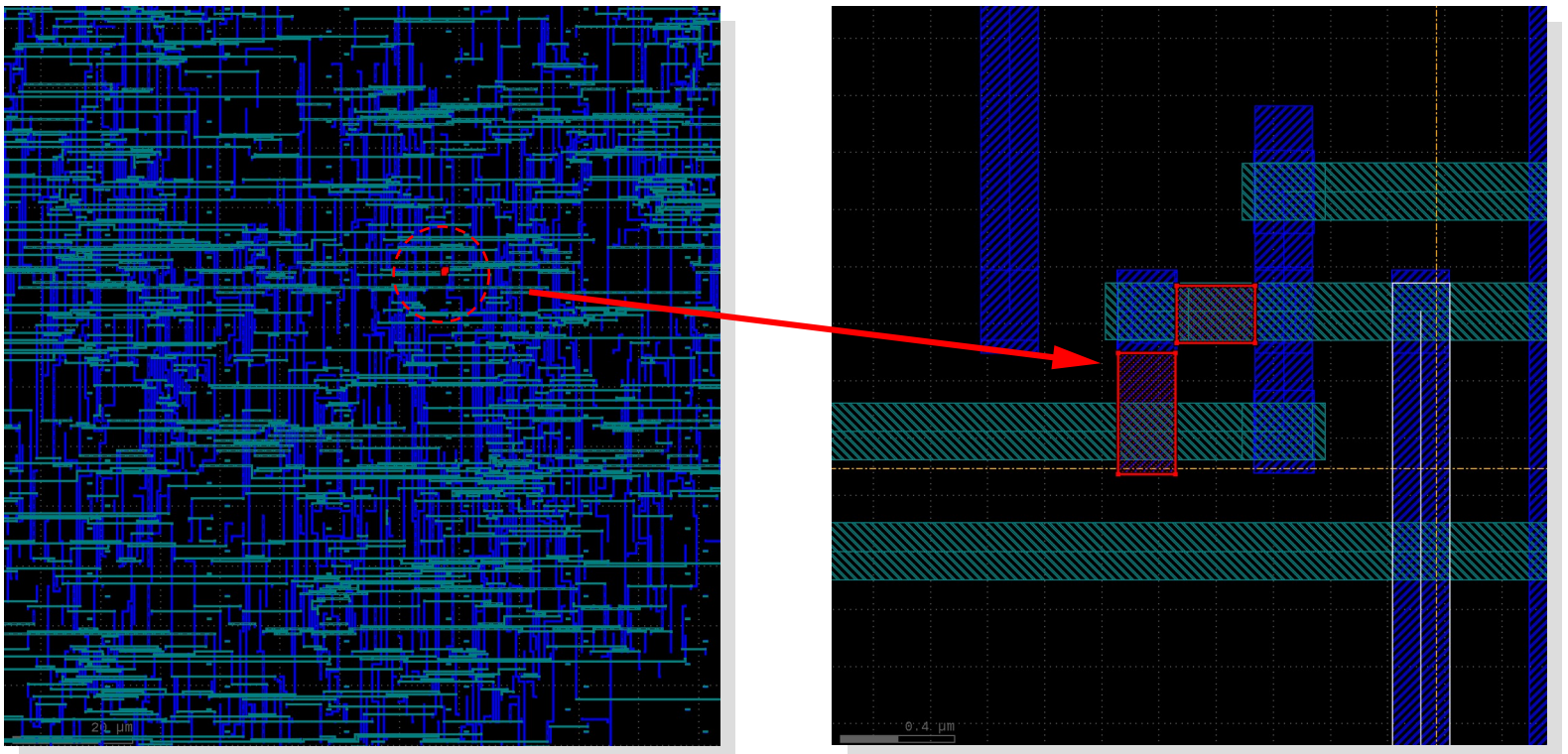
Generate structural reports with a SQL-like language



Example: List all pad locations looking for “bondpad…” instances

XOR Feature (Polygon diff)

My TOP 1 tool !!!



Hierarchical DRC/LVS engine

- As PDK user, you will not get in touch with the engine much
  - Except to complain it runs too slowly …
- But it’s in fact extremely useful to create ad-hoc checks
  - Like sanity checks not covered by the PDK
  - Or to derive design metrics

DRC Scripts

Can be written, run and debugged in the IDE

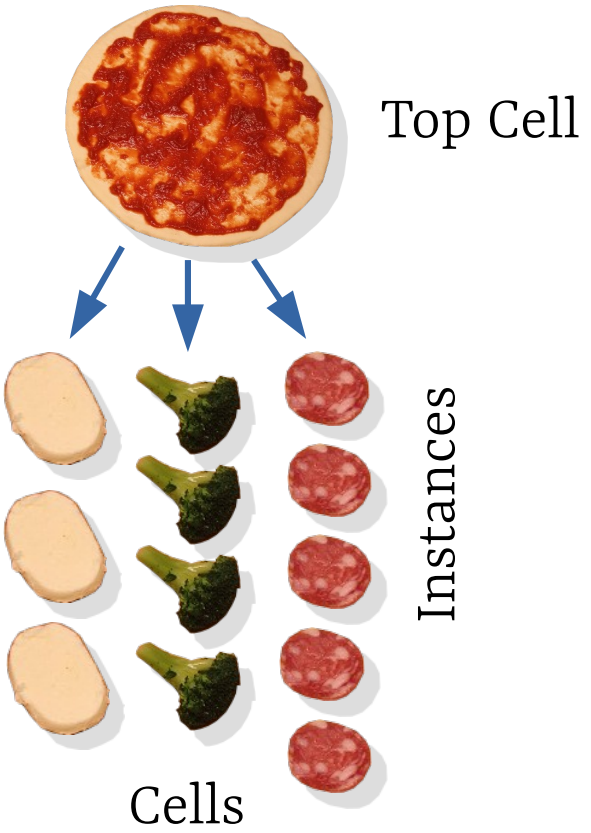
45deg

```
# Enable hierarchical mode, so edges are reported
# as low a possible in the hierarchy
deep

report("Non 45 degree")

layers.each do |l|

  e = input(l).edges
  e45 = e.with_angle(45.degree)
  e135 = e.with_angle(135.degree)
  e0 = e.with_angle(0.degree)
  e90 = e.with_angle(90.degree)
  (e - (e0 + e45 + e90 + e135)).output("Odd edges on layer #{l.to_s}")

end
```

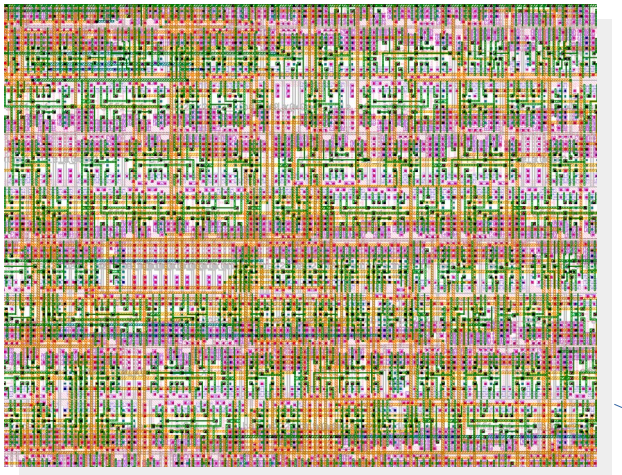
Hierarchical DRC Engine

The pizza hierarchy

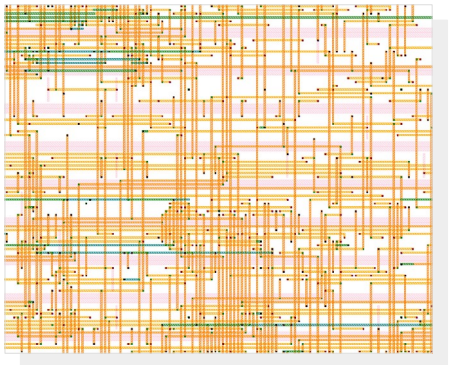
Flat View



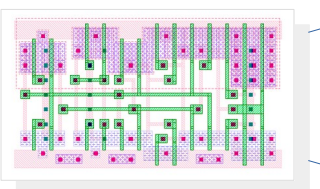
A Layout Hierarchy



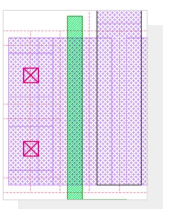
Top-level cell
Flat view

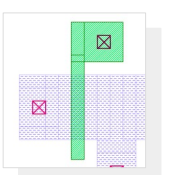


Top-level network



Library Cells
Macros





Devices

Utilizing Hierarchy in DRC

- If possible, perform operations inside cells
- Compute once, get the results everywhere
- Problems
  - Interactions between cells
  - Interaction range beyond cell boundaries
- Benefits (potentially)
  - Faster
  - Smaller files
  - Preserves functional hierarchy (→ good for LVS)

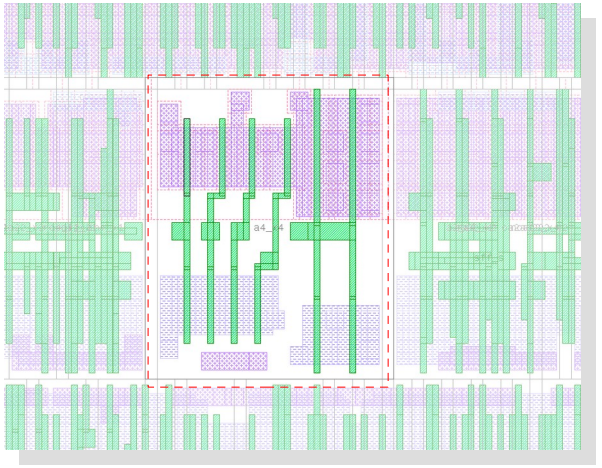
Hierarchy preservation

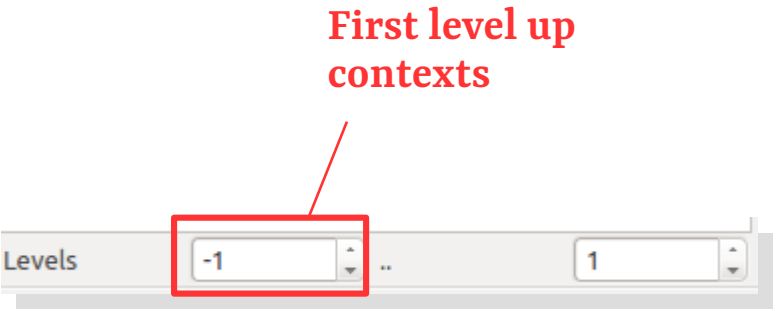
If you bake the pizza, it can no longer be decomposed – the hierarchy is lost

→ Keeping pieces separated is the main goal of a hierarchical layout engine

Hierarchical Ops in a Nutshell

- Context tree (aka “inverse layout tree”)





- Computation is done over all individual contexts
- Boolean core → all area which is common over all contexts will go to the child cell
- Everything else is promoted to parent cells
- No hierarchy manipulation required (for booleans at least)

Hierarchical DRC and LVS

- DRC functions are used to derive recognition and device layers for LVS
- Hierarchical DRC means the LVS also runs hierarchically
  - Devices are detected inside the macros
  - Connections are made inside the macros
  - Connections across the hierarchy make pins
- Ideally, the layout hierarchy then matches the schematic hierarchy

The Future

There is so much to do ..

- Integration with schematics
  - Connectivity
  - Pin objects
  - Flight lines
  - Backannotation
- Integration with library manager
- DRC engine performance
- Parasitic extraction

- In another 20 years I'll be about 80!
- And things change so quickly now
  - AI revolutionizes software development
  - Alternatives are popping up (China?)
  - New programming paradigms (massive parallel, async, functional ..) make a 20 year old C++ code base obsolete?
- BUT: KLayout is not dead yet!

And still, you're welcome to use it :)

Thank you for Listening