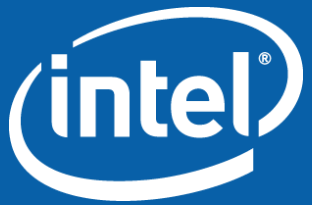
EDA Scripting Unleashed: Real-Life Examples Using oaScript and oaxPop

James D. Masters

Intel Corp

Si2CON

October 6, 2015



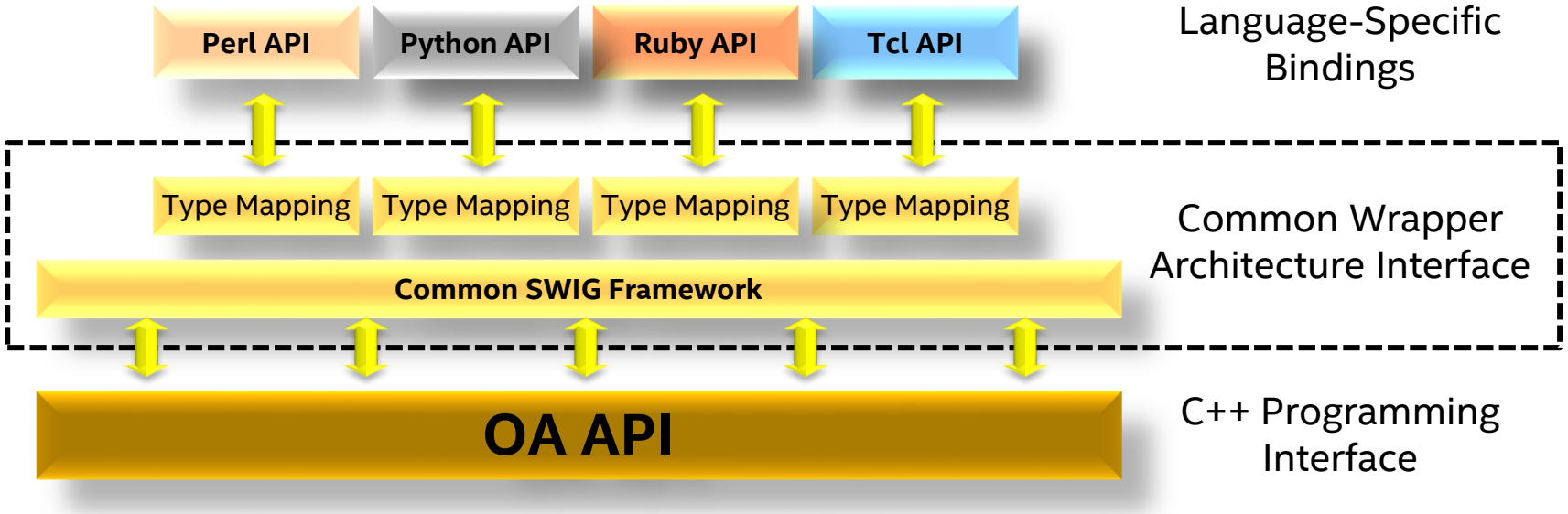
Agenda

- oaScript overview
- oaxPop overview
- Intel's experience with oaScript and oaxPop
  - Framework bundle of required packages
  - Performance observations
  - Density Calculator application
  - Methodology Checker application
- oaScript/oaxPop Roadmap
- Summary

oaScript overview

- Standalone direct interface to OpenAccess (OA) using Perl, Python, Ruby, or Tcl
  - Enables rapid development of powerful OA-based software
  - Performance and memory usage is mostly comparable to that of a C++ application (a few exceptions will be mentioned later)
- Matches OA C++ API very closely
  - Existing C++ API documentation can be referenced
  - Auto conversion of types between C++ and scripting language
- Includes convenience functions to reduce code and improve productivity
- Initiated in 2009, and refined over the past 6 years in Si2's oaScript working group (code base is stable)

oaScript interaction with OA API (via SWIG)

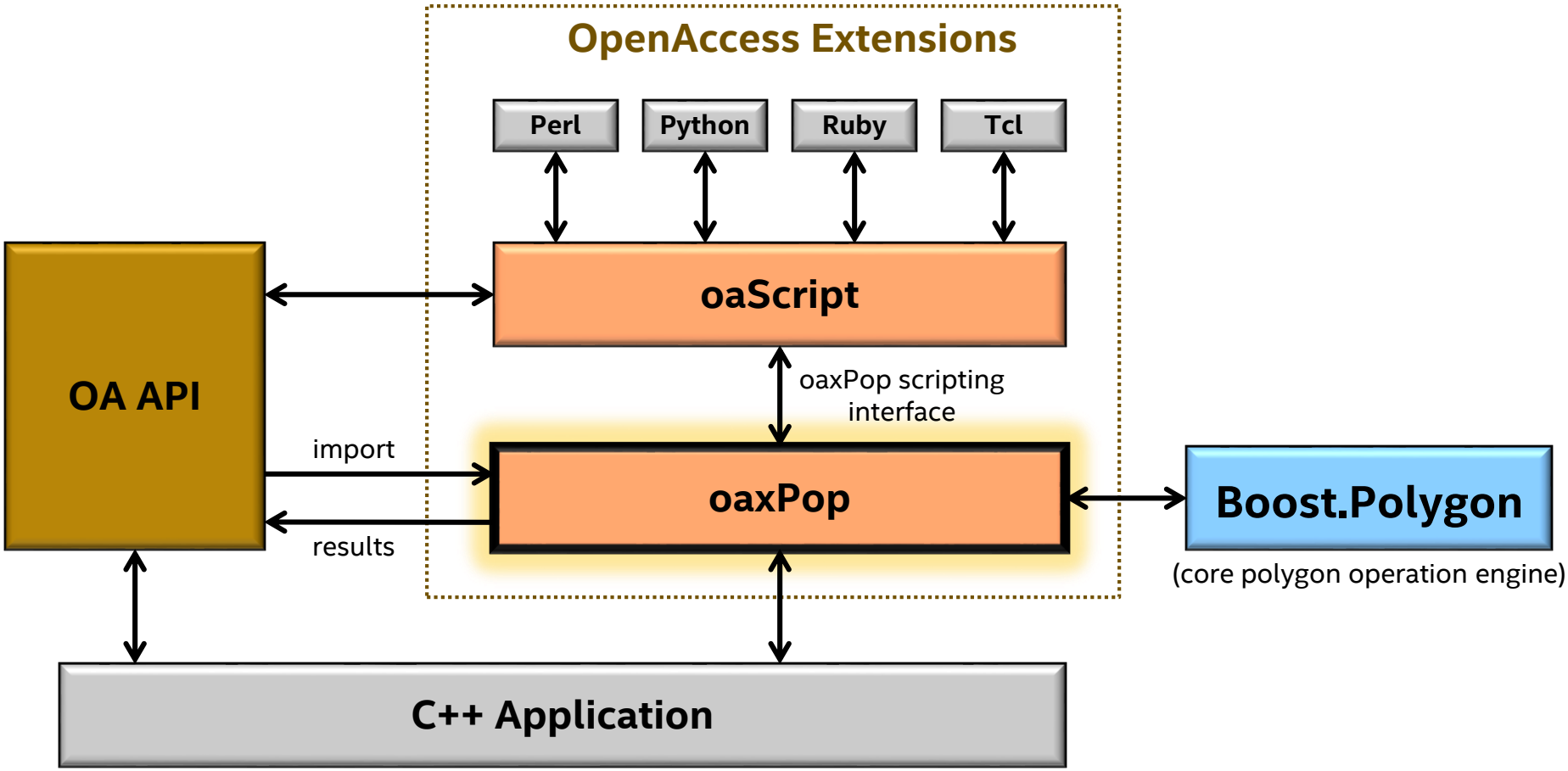


- Uses the Simplified Wrapper and Interface Generator (SWIG) tool to expose C++ APIs to scripting languages
  - Common interface through SWIG ensures cross-language consistency and reuse
  - All languages interface OA through the official OA API

oaxPop overview

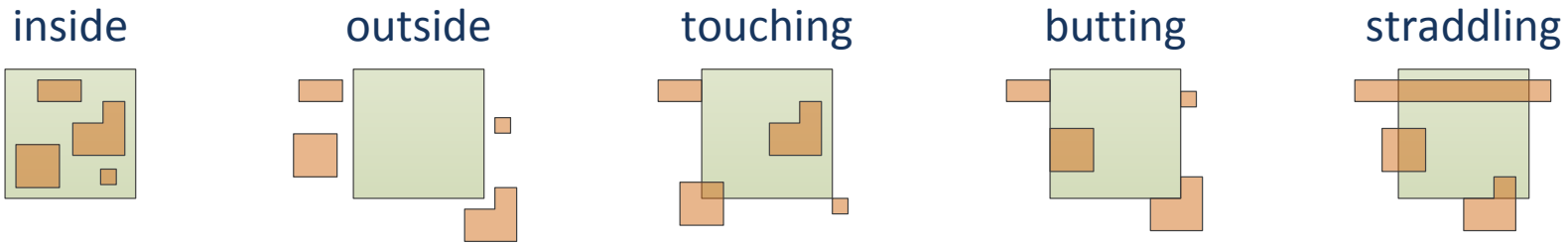
- Provides high-speed polygon manipulation capabilities in OA-based applications
  - Leverages the open-source Boost* **Boost.Polygon** high-speed polygon manipulation library contributed by Intel in 2008
  - Works directly with OA object types (oaShape, oaBox, oaPointArray, etc.)
- Work started with a proof-of-concept sample in 2011 and created a formal Si2 oaxPop WG in 2012
  - Added Unified Layer Model (ULM) high-level shape relationship operations
  - Added scripting interface (leveraging oaScript work) to enable in Perl, Python, Ruby, and Tcl

oaxPop interaction with OA API and Boost



oaxPop has more than boolean operations…

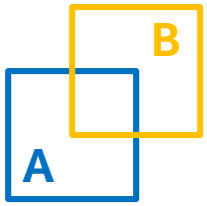
- Standard operators using +, |, *, &, -, ^
- The Unified Layer Model (ULM) operations are also included
  - Industry-standard description of relationships between polygons
  - Provides a rich set of functionality



- Area selection operators to find polygons meeting a criteria using <, <=, ==, >, >=
- Resize operations to resize all edges of a polygon (all types) or orthogonal directional-based resize: north, south, east, west (only on 90-degree shapes)

oaxPop performance

- oaxPop performance compared against other polygon manipulation libraries/software
  - Boost.Polygon outperforms other open-source libraries according to results published on the Boost.Polygon website
  - oaxPop FigSet90 performance is competitive with other professional polygon manipulation libraries/tools (see table)



- 50M shapes on “A”
- 50M shapes on “B”
- All flat – no hierarchy
- GDSII size was 6GB
- Only used single thread

Tool	AND	OR	XOR	NOT
oaxPop	55s 1x	47s 1x	58s 1x	45s 1x
Data1	29s 0.53x	58s 1.23x	93s 1.60x	57s 1.27x
Data2	35s 0.63x	61s 1.30x	113s 1.95x	80s 1.78x
Data3	109s 1.98x	61s 1.30x	204s 3.52x	82s 1.82x

Note: Data* time/factor data are scrambled (sorted)

oaxPop performance (basic + ULM ops)

# shapes	and	or	xor	sub	avoi	bO	encl	insi	inte	outs	over	stra	touc	butt	bOC	bOO	coin	cO
10000	0	0	0	0	0.1	0.1	0.1	0	0.1	0.1	0.1	0.1	0.1	0.3	0.8	0.4	0.5	0.5
20000	0	0	0	0	0.2	0.3	0.2	0.1	0.2	0.2	0.2	0.2	0.2	1.7	3.9	1.9	2.2	2.2
30000	0	0	0	0	0.3	0.5	0.3	0.1	0.2	0.3	0.3	0.3	0.3	8.8	17.4	8.6	9.9	9.3
40000	0.1	0.1	0	0	0.4	0.7	0.4	0.1	0.3	0.4	0.4	0.5	0.3	25.6	52.9	26.6	27	26.3
50000	0.1	0.1	0.1	0.1	0.5	0.9	0.6	0.2	0.4	0.5	0.5	0.6	0.4					
100000	0.2	0.1	0.1	0.1	1	1.8	1.3	0.5	0.9	1.1	1	1.6	1					
200000	0.4	0.3	0.3	0.3	2.4	3.8	3.1	1.3	2	2.5	2.3	4.5	2.3					
400000	0.9	0.8	0.9	0.8	5.9	7.7	7.7	4.1	4.3	6.7	5.4	13.2	5.2					
800000	1.9	1.4	1.5	1.4	13.6	16.5	21.8	13.5	9	13.8	13.3	38	12.7					

(randomized shape sizes and locations; time is in seconds)

Operations Key

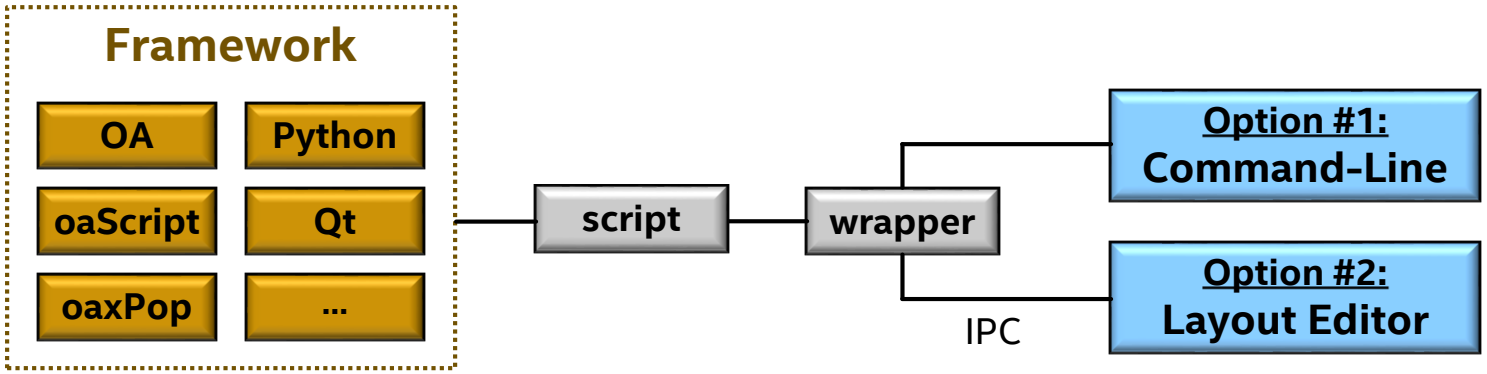
and = &
or = |
xor = ^
sub = -
avoi = avoiding
bO = buttOnly
encl = enclosing
insi = inside
inte = interacting
outs = outside
over = overlapping
stra = straddling
touc = touching
butt = butting
bOC = buttingOrCoincident
bOO = buttingOrOverlapping
coin = coincident
cO = coincidentOnly

Intel's experience with oaScript and oaxPop

- Deployed oaScript in small production flows since 2011
- Deployed both oaScript and oaxPop in larger production flows including deployment to design groups starting in 2014
  - All oaScript languages used to some extent throughout Intel, but Python has recently been gaining popularity
- Areas of opportunities for oaScript/oaxPop:
  - Intel-specific flow needs that may not be immediately found in or are outside the scope of EDA tool features
  - QA flows to check large amounts of OA-based data very quickly
  - OA data mining flows to get quick statistical answers

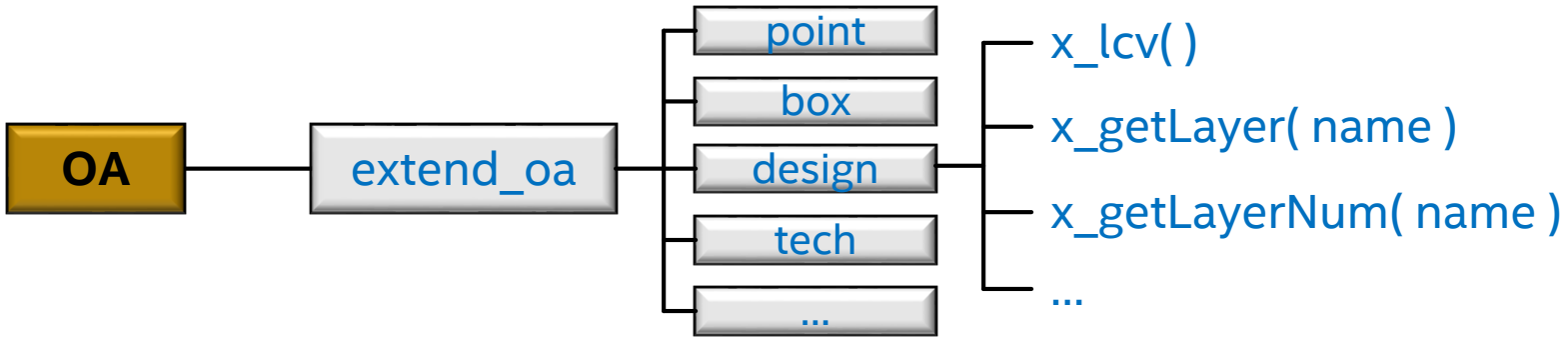
Bringing it all together: Intel's "framework"

- Intel defines a "framework" for required components
  - Developers don't need to compile and install separate modules
  - Users receive a self-contained tested package
- Other modules are available for non-OA features
- Automation can be run standalone or from within a layout editor (using inter-process communication hooks)
- Access to the framework occurs through a "wrapper" which sets the needed environment (like an EDA tool)

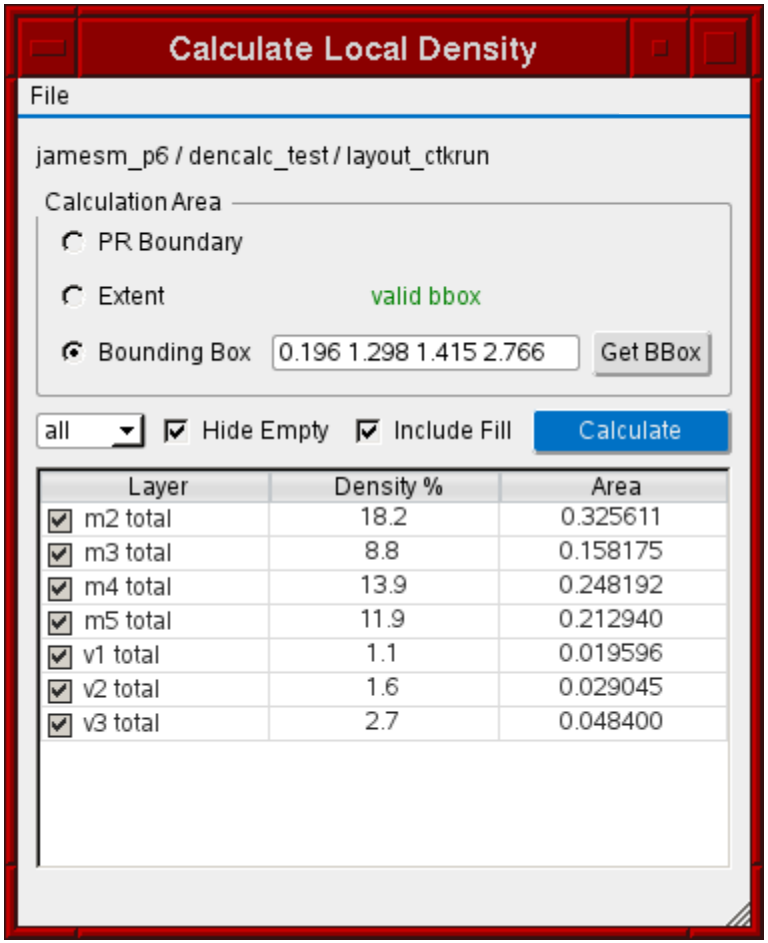


Extending OpenAccess

- Intel added high-level functions directly in Python OA classes for common sequence of steps
  - Overrode Python object “dictionary” to give a feel of functions being directly in the object (instead of disconnected functions)
  - Use a convention of prefixing extended functions with “x_” to clearly separate OA functions from extended functions
- Extensions are built-into the framework to allow for easy access during development



Density Calculator



Density calculated using oaShapeQuery and oaxPop

Density Calculator: Quick in-editor solution for real-time density feedback

- All mask layers with full hierarchy depth supported
- Derived layer calculations available (not shown here)

Performance on medium-size block

- Hierarchy depth: 4
- Number of queried layers: 19
- Shape count: ~277k
- Runtime: ~1 second*

* With C++ helper class (details on next slide)

Bounding box query received from layout editor

oaShapeQuery performance issues

```
class MyShapeQuery(oa.oaShapeQuery):

  def __init__(self):
    oa.oaShapeQuery.__init__(self)
    self.figset = oaxPop.FigSet90()

  def queryShape(self, shape):
    if isinstance(shape, oa.oaRect) or
       isinstance(shape, oa.oaPolygon) or
       isinstance(shape, oa.oaPath) or
       isinstance(shape, oa.oaPathSeg):
      occ_shape = self.getOccShape(shape)
      hier_path = occ_shape.getHierPath()
      self.figset.append(shape, hier_path.getTransform())
```

Long runtimes using oaScript-based shape query (function call overhead)

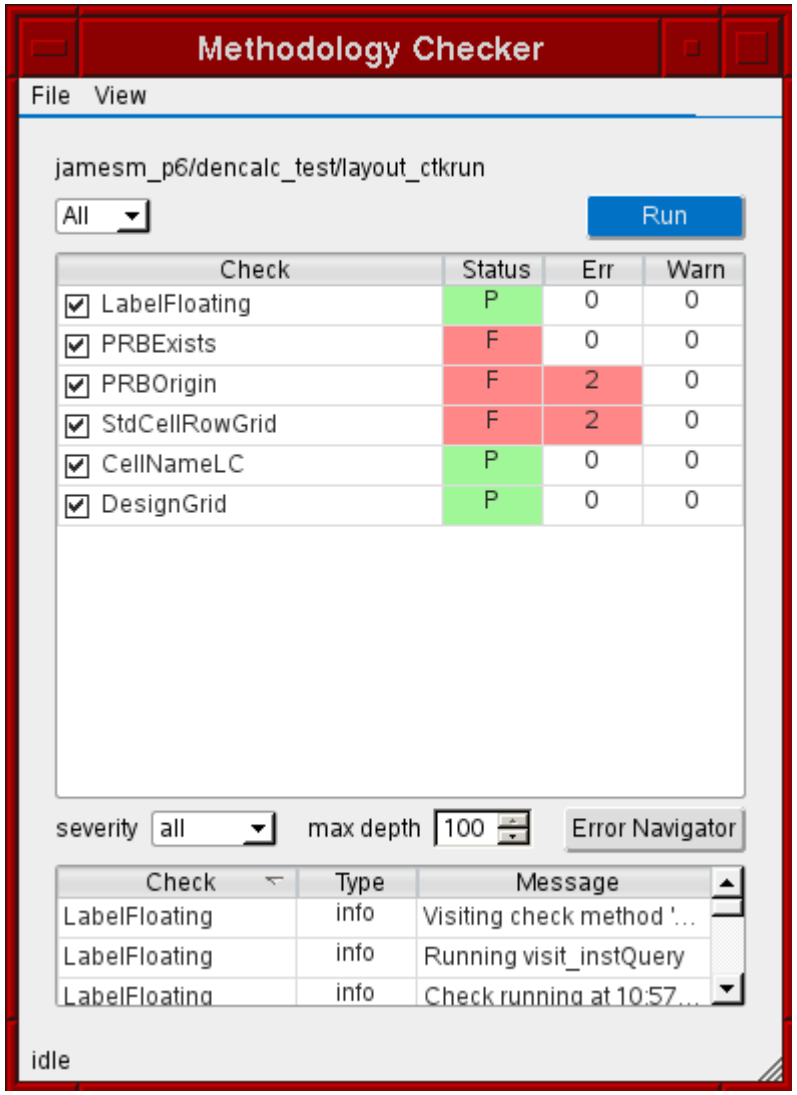
Biggest Time Consumer

Created custom C++ based shape query

- Created small C++ class to perform shape query operations
  - Avoided python function call overhead for each shape
  - Abstracted inner-workings of oaShapeQuery to allow script author to only make high-level function calls
  - Added “caching” mechanism to avoid re-query in areas that were already visited
- Observed **400x~2000x improvement** in runtime versus the pure-python shape query

```
squery = design.x_initShapeQuery()
squery.setRegion(bbox)
lpps = (("m1", "drawing"), ("m1", "fill"))
polygons = squery.group(lpps).figset() ➔ Queried FigSet90
```

Methodology Checker



Methodology Checker: Quick check of design methodology compliance

- All mask layers with full hierarchy depth supported
- Many other Intel-specific checks exist (not shown here)

Use of oaxPop essential

- DesignGrid: `err = wires - grid`
- StdCellRowGrid: synthesize row grid; `err = stdcell_prb - row_grid`
- LabelFloating: 2x2 DBU label squares; `err = lbls - lbls.touching(shps)`

Error Navigator started in layout editor

oaScript/oaxPop scripting limitations

- Need to save design between layout editor and OA script
  - Mitigation can be achieved through auto-detection of unsaved edits which are saved and re-read in the OA script application
- Careful management of OA_PLUGIN_PATH and LD_LIBRARY_PATH for pcell and lib.defs plug-ins
  - Setup wrapper code to detect environment and set accordingly
  - There are still many “gotcha” scenarios to address
- No native DRC-checking features in oaxPop
  - No edge/vertex distance operations
  - Some workarounds exist, but require several layer operations and/or iterating over shapes in a layer set

Roadmap

2015 Si2CON 1.3 Release (October)

- Invalid shape detection (inverted box)
- Updates to documentation

2016 DAC Release (May 2014)

- Contribute shape query improvements

Beyond:

- Multi-threading and other performance improvements
- Vertex and edge-based operations

Summary

- oaScript and oaxPop are stable extensions available from Si2 (have had several years of development)
- Powerful OA-based applications can be written very quickly using oaScript and oaxPop
- Intel has successfully used oaScript and oaxPop in production applications
  - Applications are available either standalone or from an integration in a layout editor
  - Excellent performance observed in applications using oaScript and oaxPop (some adjustments required in certain cases)

References

oaScript:

- https://www.si2.org/openeda.si2.org/projects/oascript/

oaxPop:

- https://www.si2.org/openeda.si2.org/projects/oaxpop/
- http://www.boost.org/doc/libs/release/libs/polygon/doc/index.htm