Introduction to Physical Compiler and ILM Flow

Timothy Chiun

Simon Koval

Corporate Applications Engineering

Agenda

- Introduction to Physical Compiler
- Hierarchical Physical Synthesis with Interface Logic Models

Introduction to Physical Compiler

Timothy Chiun

CAE, Physical Compiler

Agenda

- Introduction
  - Problems
  - Current Flows and Issues
  - Synopsys Physical Synthesis Solution
- What is Physical Compiler?
- Running Physical Compiler
- Physically Integrated Methodologies
- Summary

Agenda

- **Introduction**
  - **Problems**
  - **Current Flows and Issues**
  - **Synopsys Physical Synthesis Solution**
- What is Physical Compiler?
- Running Physical Compiler
- Physically Integrated Methodologies
- Summary

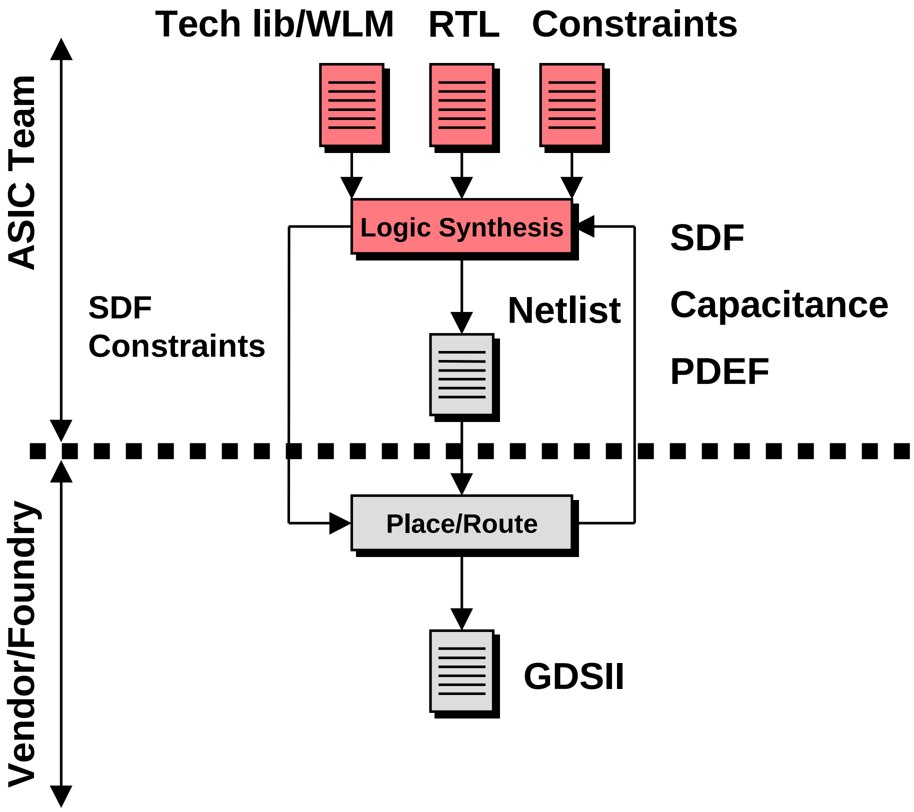
Problems

- Achieving Timing Closure
- Scaling the Design Process for Multi-Million Gate Chips

Timing Critical Designs
What's not working?

- Synthesis
  - WLMs are statistical
  - Constraints are estimated

    set_input_delay,
    set_output_delay, set_load,
    set_driving_cell,
    set_clock_skew, etc.

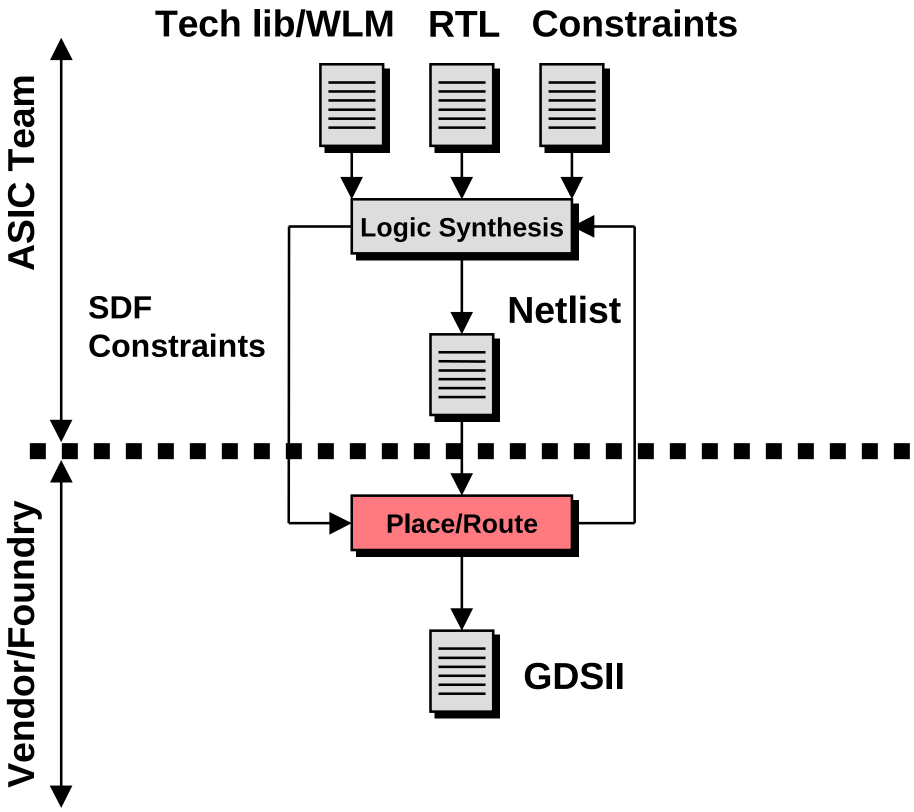


Timing Critical Designs
What's not working?

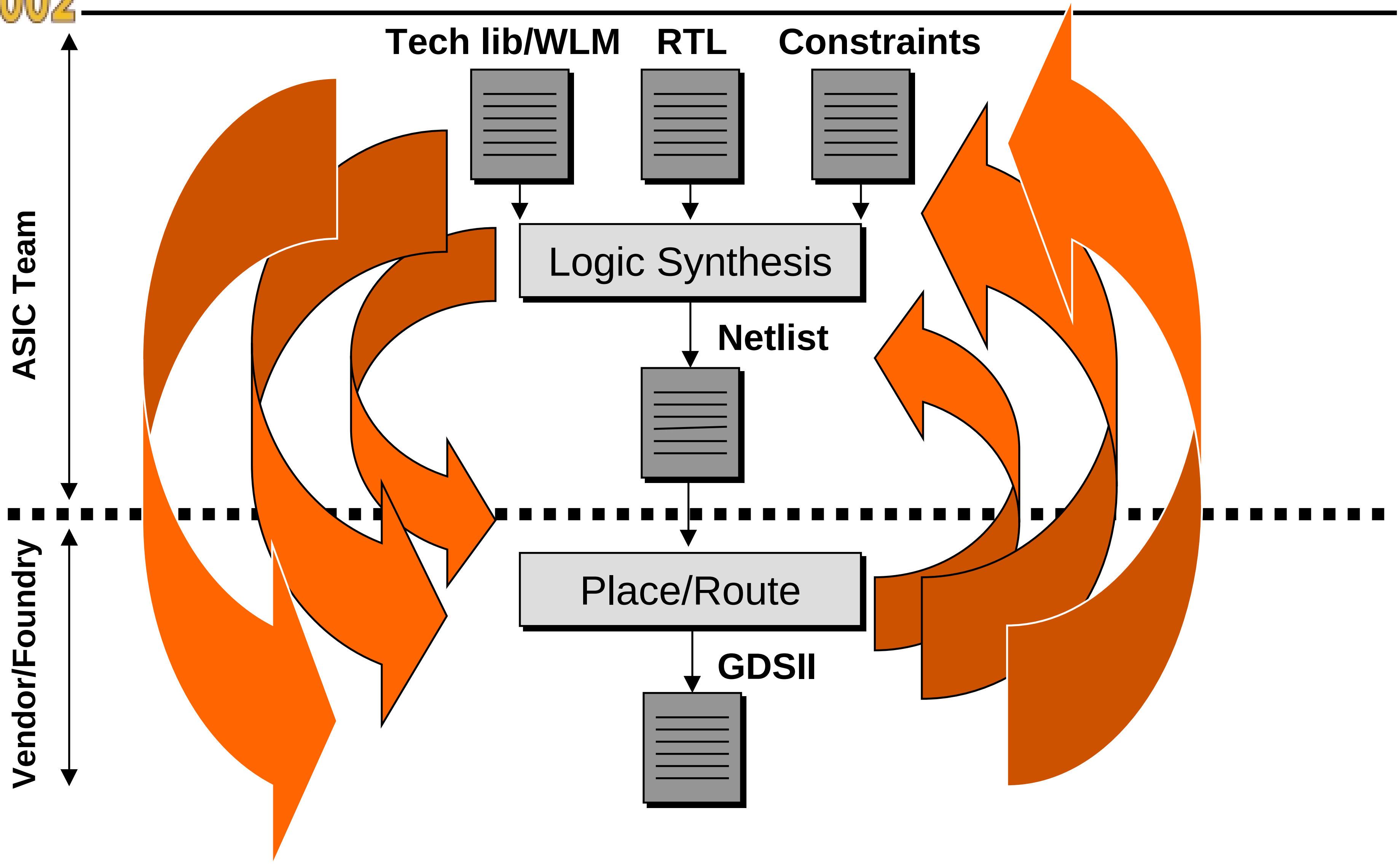
- Place/Route
  - Estimates for wire delays are off!

    Nets with the same fanout have very different delays in the placed design
  - ECO's are required

    Timing closure becomes a moving target



Flows and Issues – Traditional Flow



ASIC Team

Vendor/Foundry

Tech lib/WLM

RTL

Constraints

Logic Synthesis

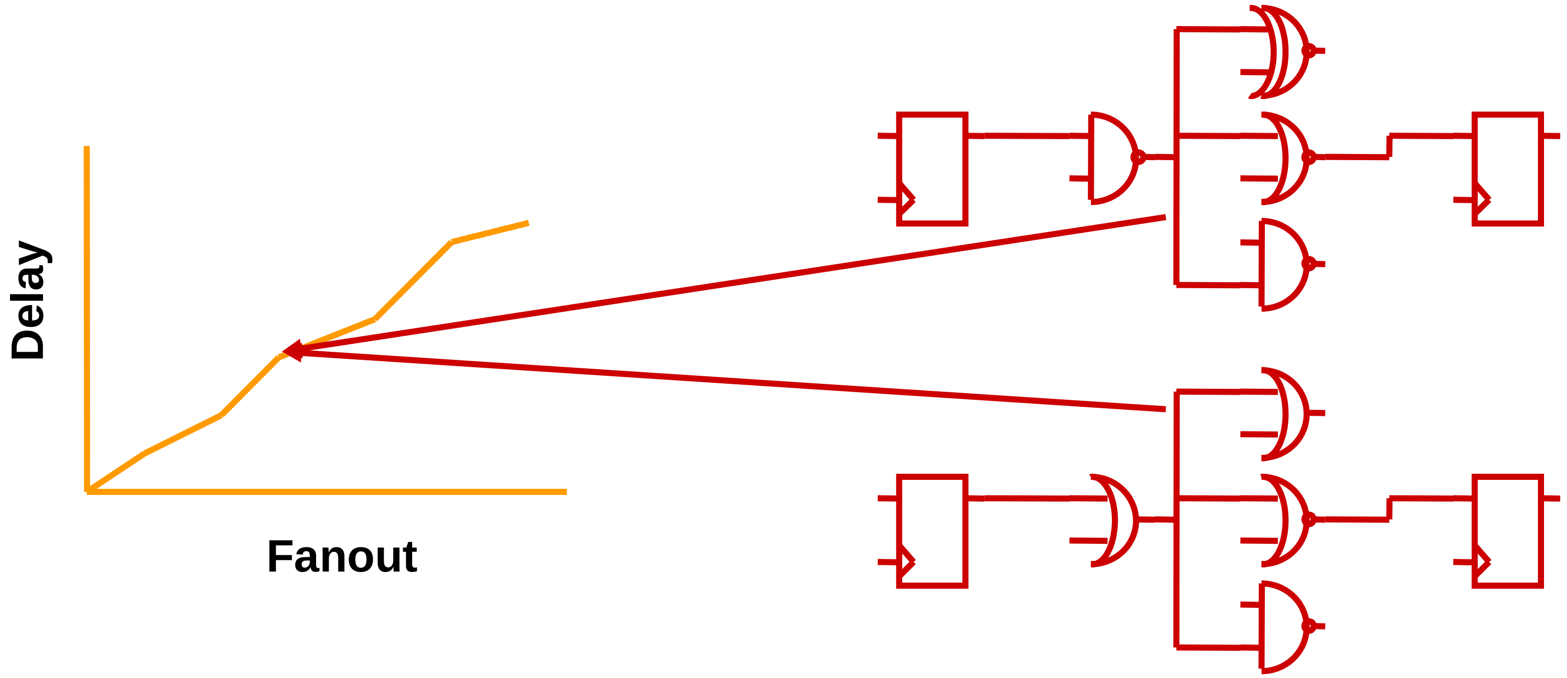
Netlist

Place/Route

GDSII

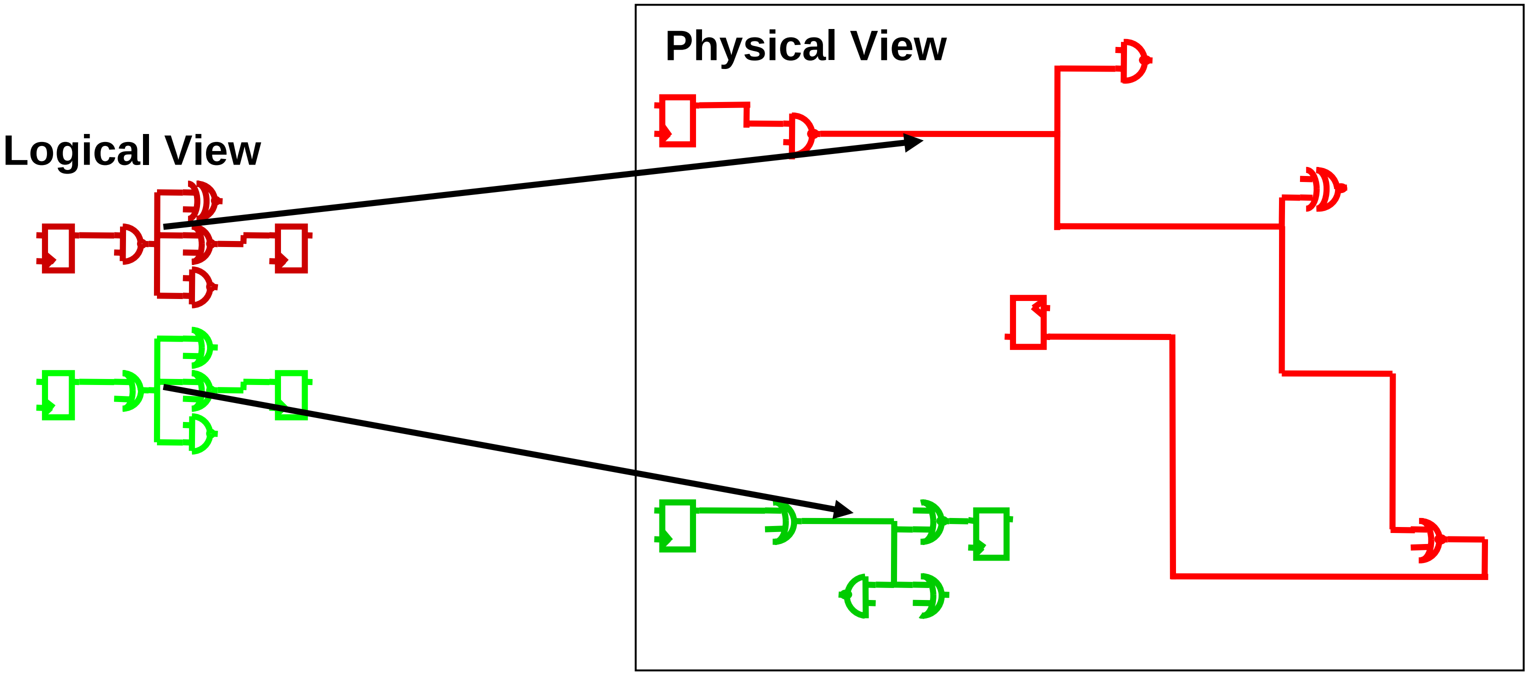
Unifying Synthesis & Placement

1. Front-end timing is becoming unreliable
   - With traditional flows, all nets with the same fanout have the same estimated interconnect delay during front-end design



Unifying Synthesis & Placement is the Best Technical Solution

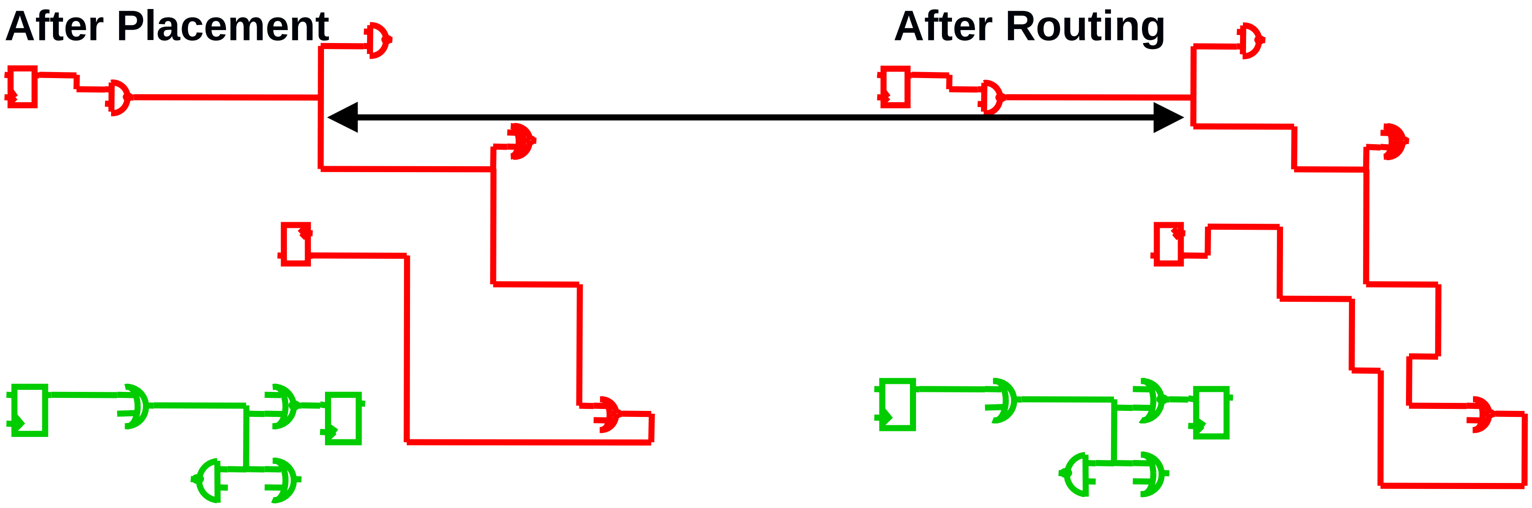
1. Front-end timing is becoming unreliable
2. Placement can change timing dramatically
   - After placement, it is obvious that nets with the same fanout will not have the same interconnect delay



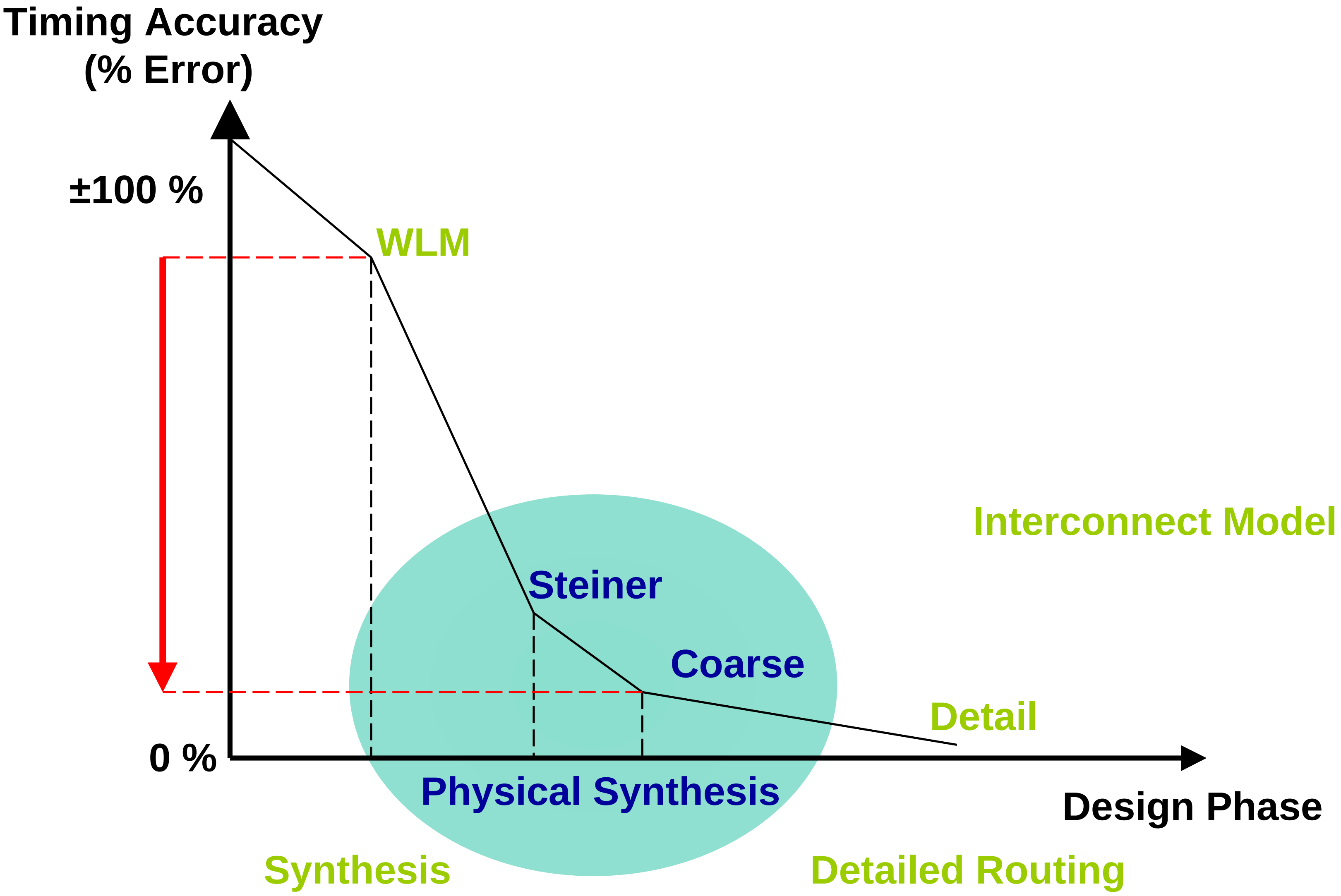
Unifying Synthesis & Placement is the Best Technical Solution

1. Front-end timing is becoming unreliable
2. Placement can change timing dramatically
3. Detailed routing has only a minor effect

   when good global routing is done to model interconnect

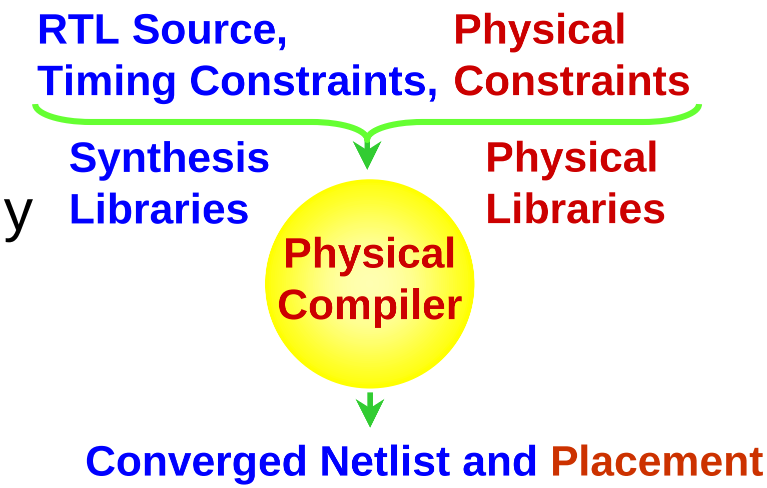


Placement is Key!



Physical Compiler: Power of Synthesis + Placement

- Faster timing closure
- Better timing correlation between synthesis result and post-layout result
- Resultant netlist is more intelligently created, resulting in reduced die area
- World class cell placement technology produces highly routable designs that meet timing



RTL Source, Timing Constraints,

Physical Constraints

Synthesis Libraries

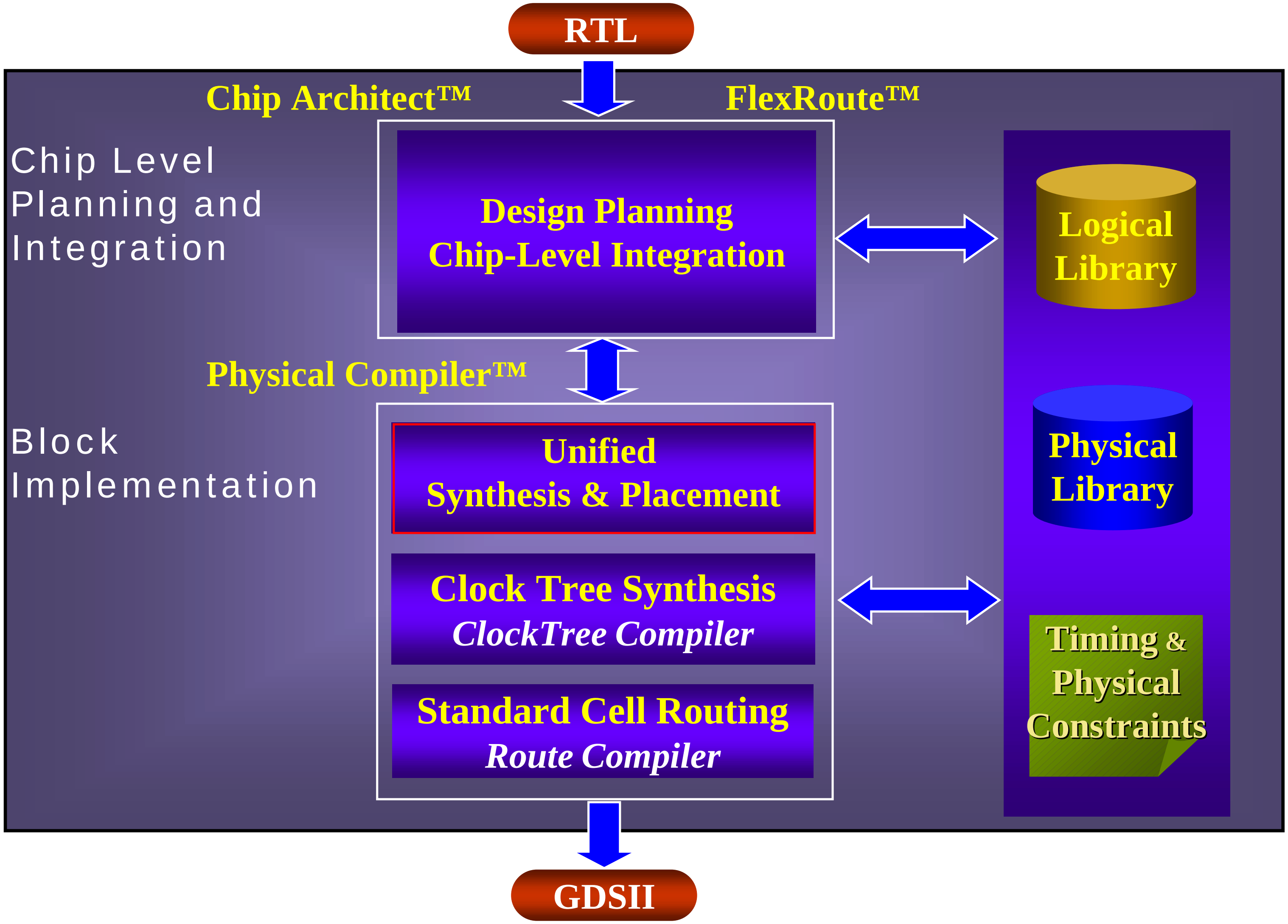
Physical Libraries

Physical Compiler

Converged Netlist and Placement

Synopsys Physical Synthesis

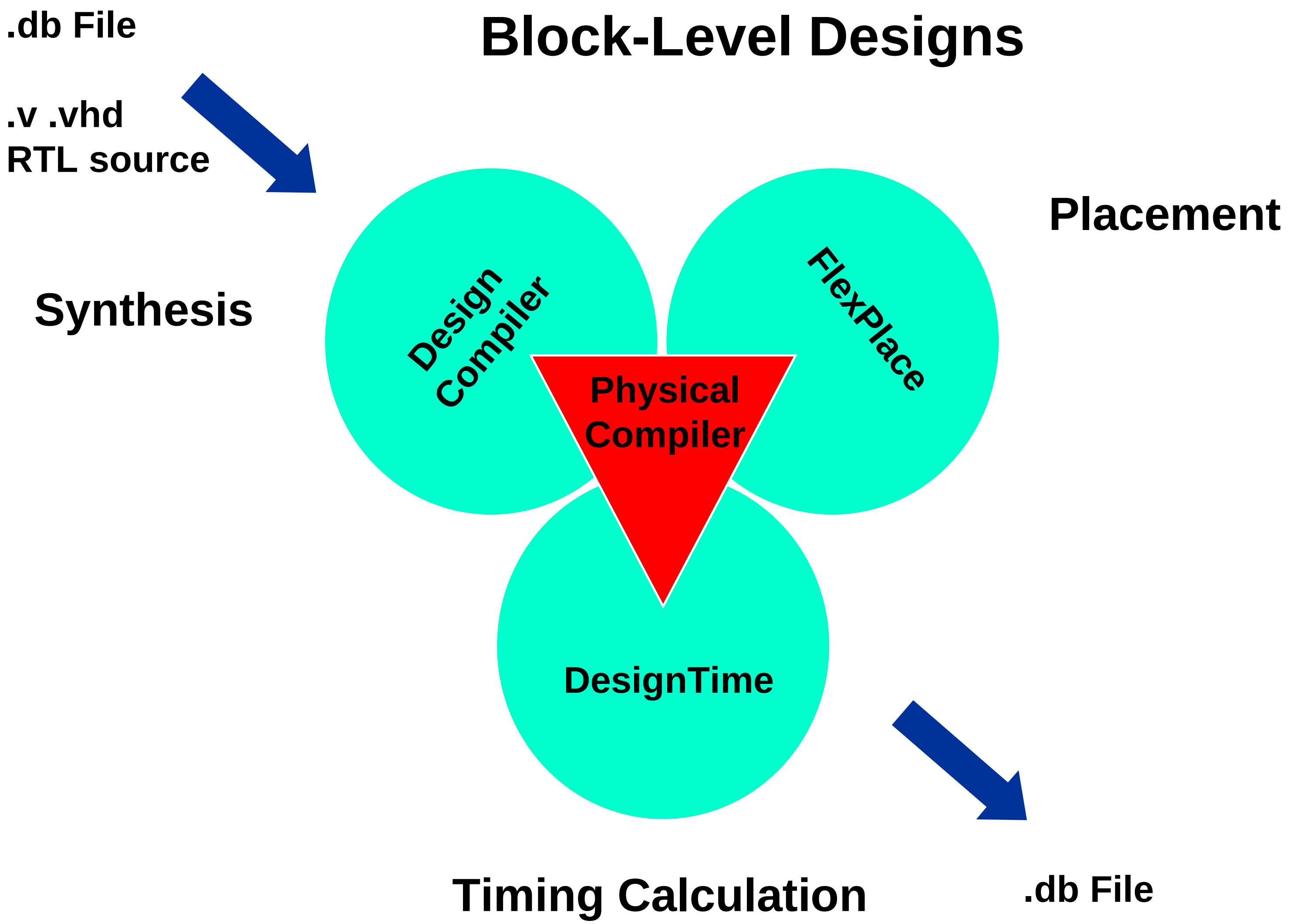
Complete RTL to GDSII Solution



Agenda

- Introduction
  - Problems
  - Current Flows and Issues
  - Synopsys Physical Synthesis Solution
- **What is Physical Compiler?**
- Running Physical Compiler
- Physically Integrated Methodologies
- Summary

What is Physical Compiler?



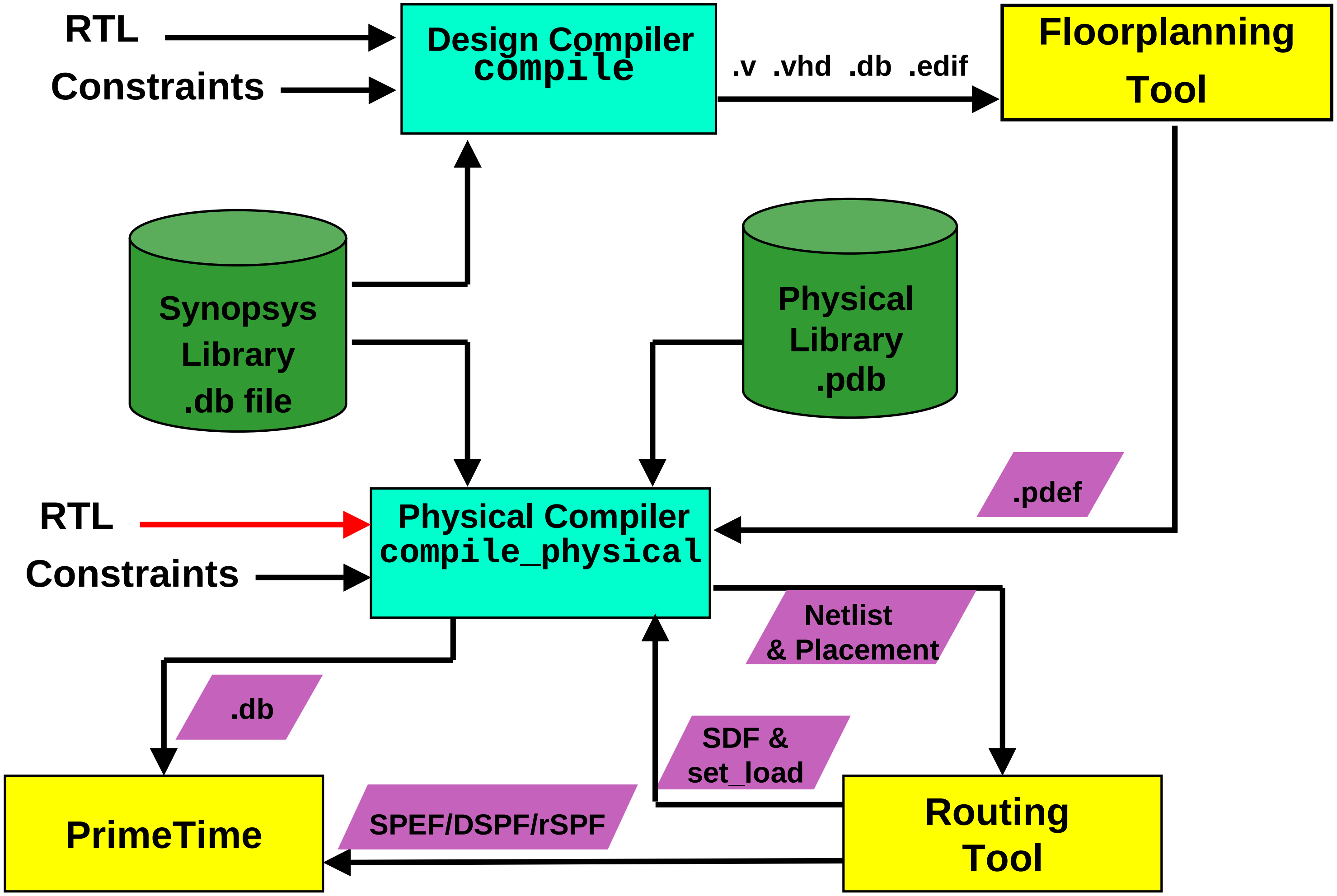
Physical Compiler is...

- Not using wireload models ~~WLM~~
- Unifies synthesis and placement
- Produces an optimized gate-level netlist AND cell placement from :
  - RTL description
  - Existing gate-level netlist
- Works on a single, flat block of physical hierarchy
  - Logical hierarchy is maintained

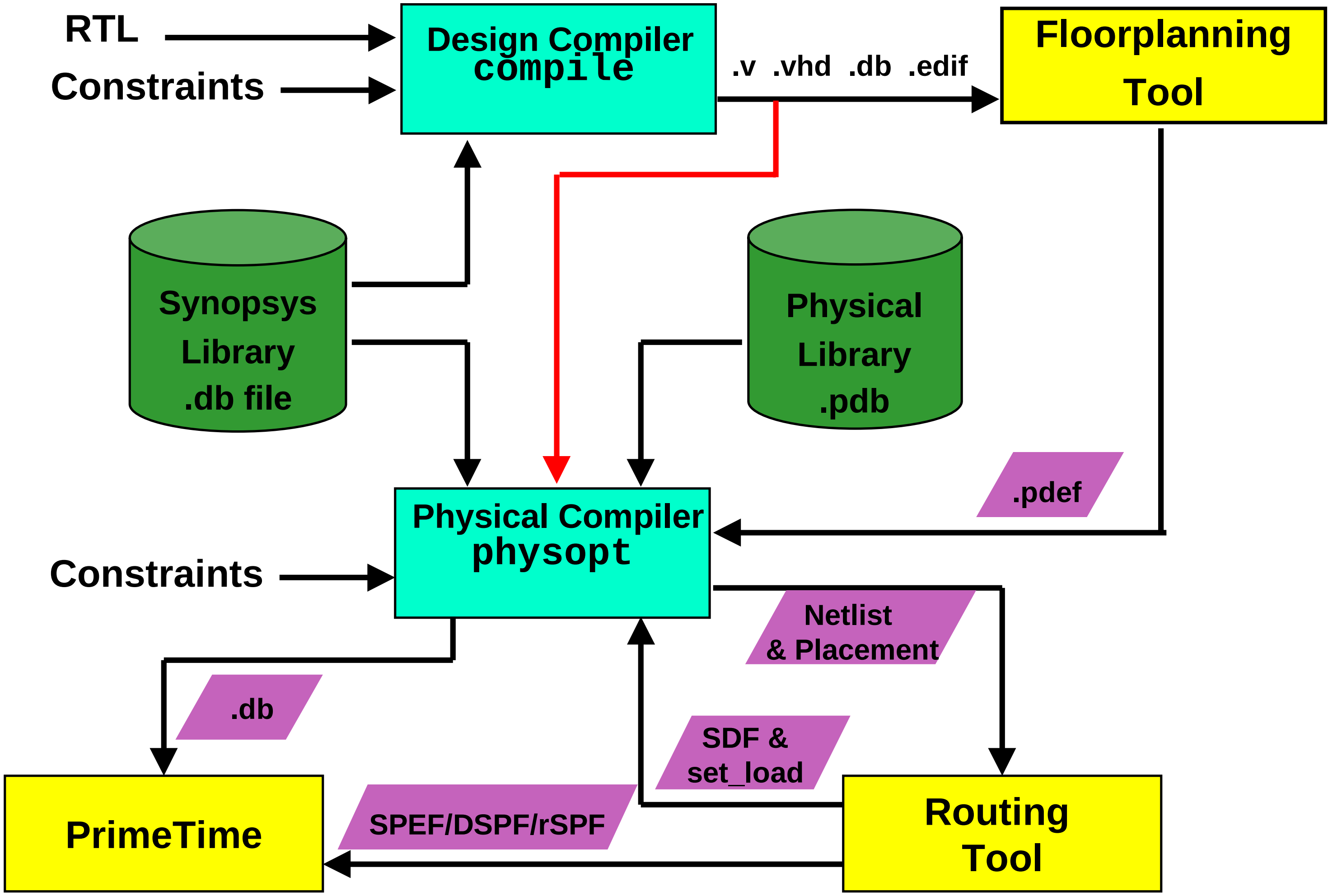
RTL or Gates ? Your choice

- RTL to Placed Gates (RTL2PG)
  - Creates an optimized netlist concurrent with placement starting from RTL
  - Provides the highest level of flexibility in architectural choices to meet design goals
- Gates to Placed Gates (G2PG)
  - Optimizes the netlist concurrent with placement starting from a gate-level netlist
  - Least impact to existing design flows
  - Higher capacity

Physical Compiler RTL2PG Flow



Physical Compiler G2PG Flow



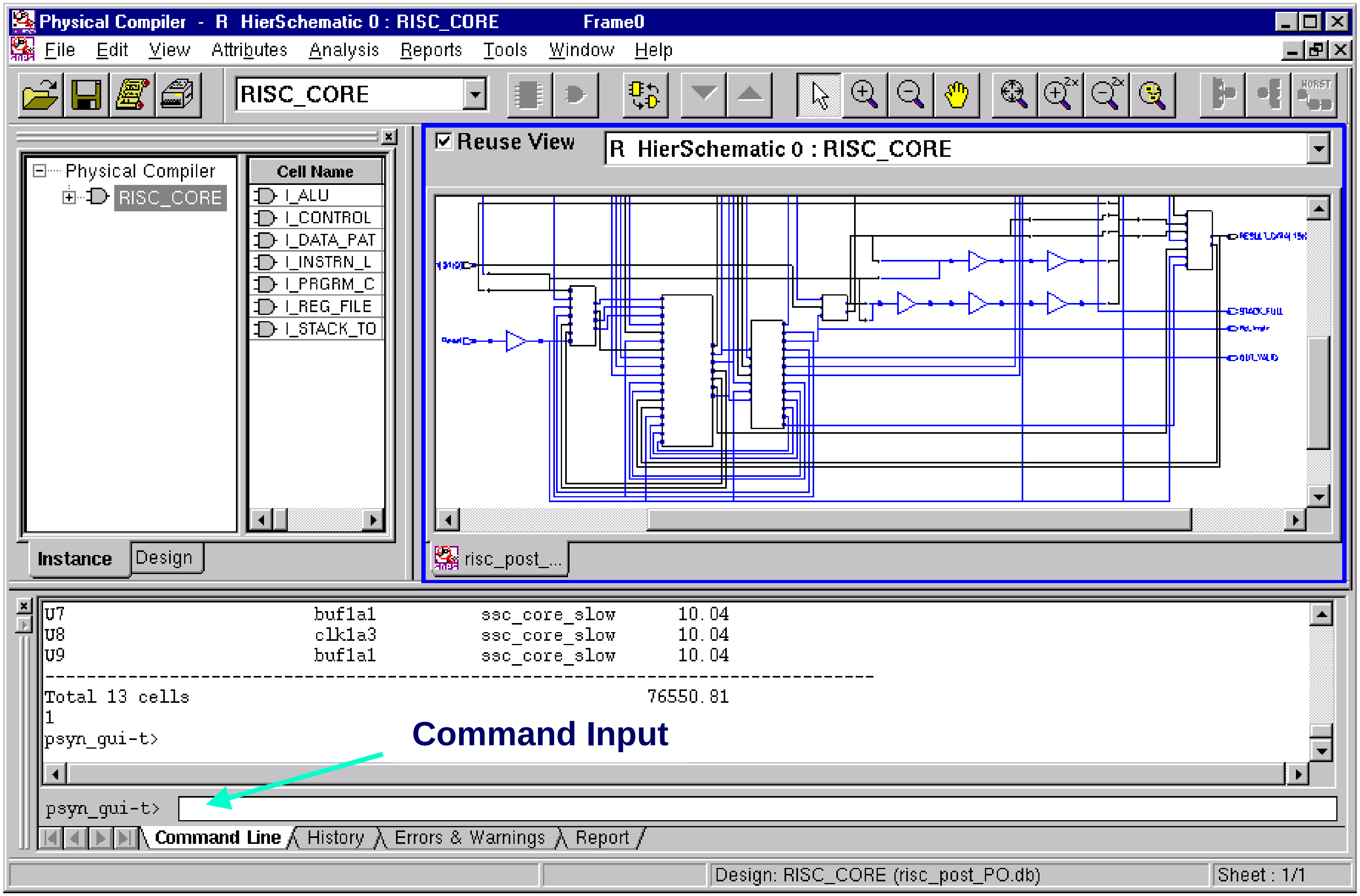
Incremental Optimizations

- Placed Gates to Placed Gates
  - Fine-tune after insertion of additional cells
  - Invoke additional effort to deal with design-specific issues (timing/congestion)
- After Routing
  - Clear up post-extraction timing violations
  - Maintain as much back-annotated timing information as possible
  - Meant for last minute timing fixes

Physical Compiler Package

- Translation Utilities
  - lef2plib
  - def2pdef
  - db2def5
- Shell Tool - psyn_shell
- GUI Tool - psyn_gui

Physical Compiler GUI



```
U7                          buf1a1          ssc_core_slow       10.04
U8                          clk1a3          ssc_core_slow       10.04
U9                          buf1a1          ssc_core_slow       10.04
-----------------------------------------------------------------------------
Total 13 cells                                                76550.81
1
psyn_gui-t>
```

Command Input

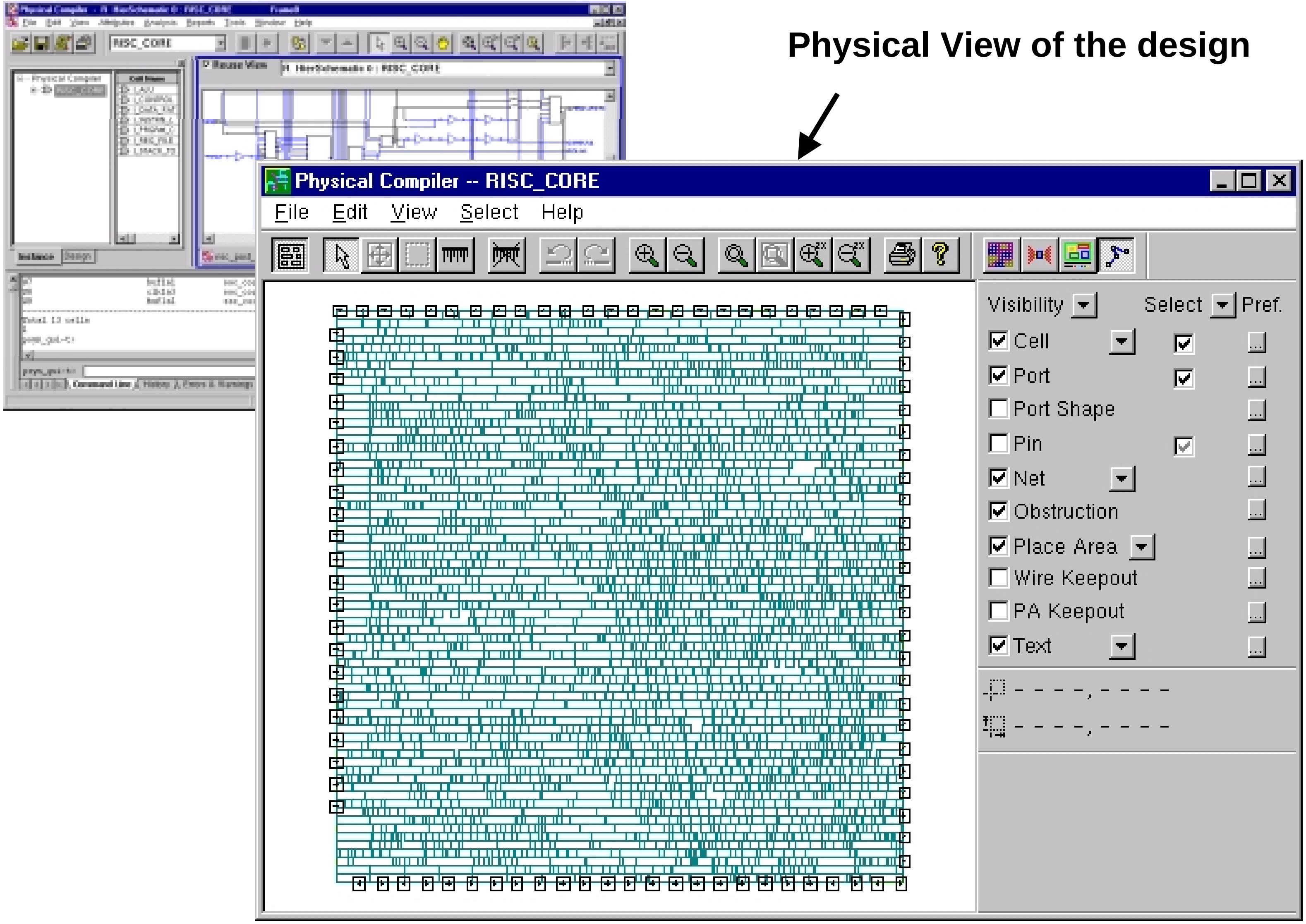
```
psyn_gui-t>
```

Design: RISC_CORE (risc_post_PO.db)

Sheet : 1/1

Viewing the Floorplan

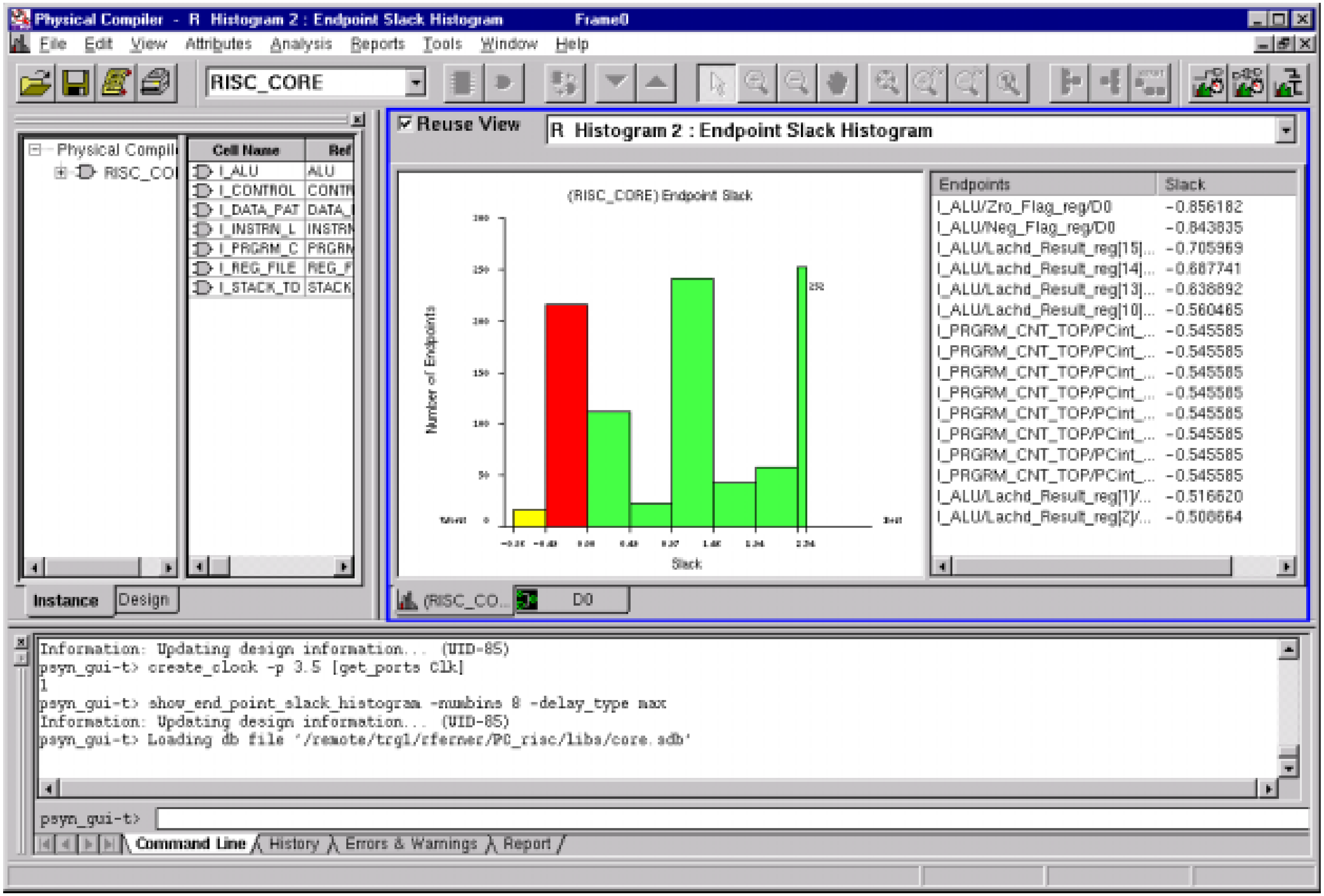
Physical View of the design



Visibility ▾ Select ▾ Pref.

☑ Cell ▾ ☑
☑ Port ☑
☐ Port Shape
☐ Pin ☑
☑ Net ▾
☑ Obstruction
☑ Place Area ▾
☐ Wire Keepout
☐ PA Keepout
☑ Text ▾

Analyzing the Histograms

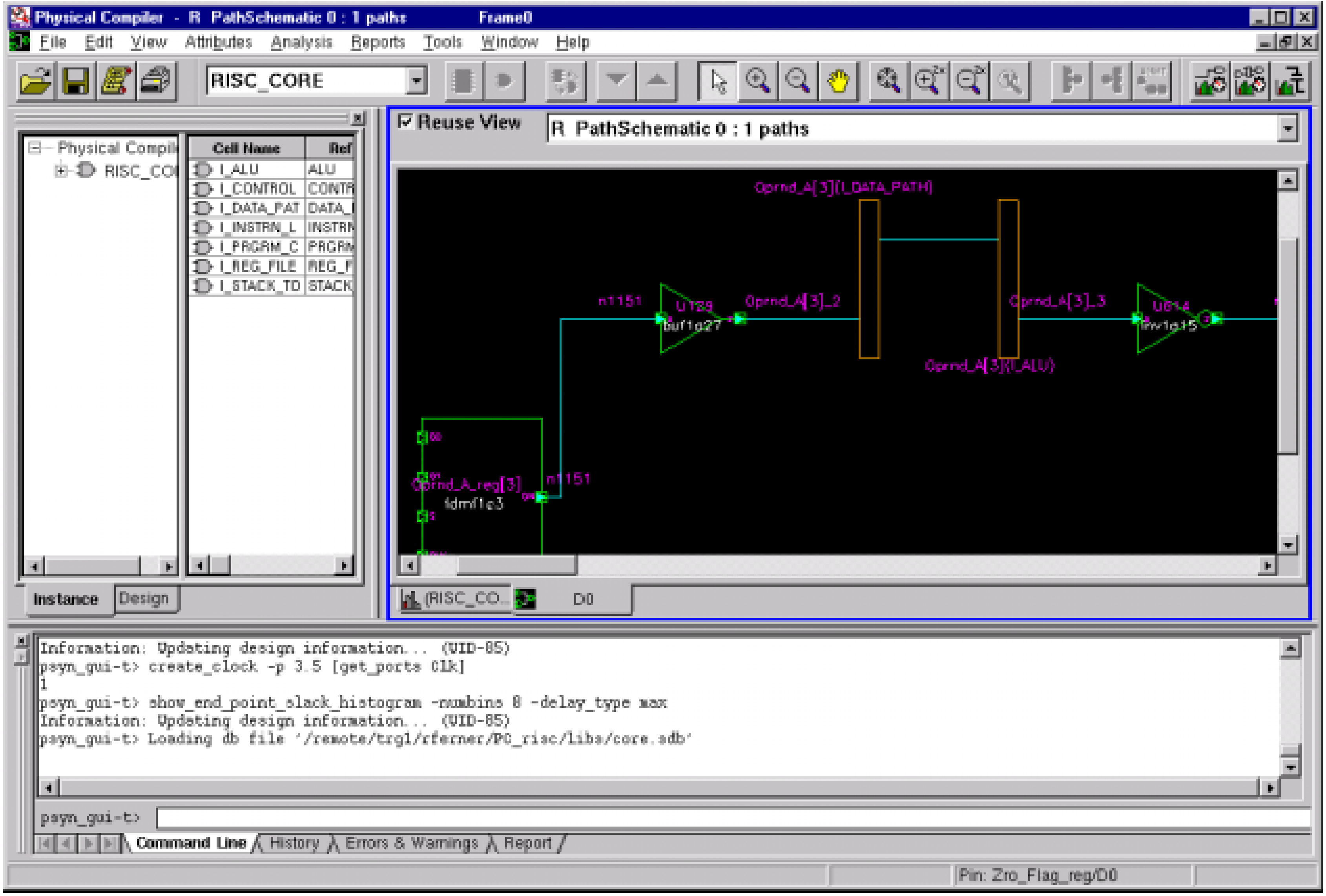

Physical Compiler - R Histogram 2 : Endpoint Slack Histogram Frame0
File Edit View Attributes Analysis Reports Tools Window Help
RISC_CORE
Physical Compil
RISC_CO
Instance
Design
☑ Reuse View
R Histogram 2 : Endpoint Slack Histogram
(RISC_CORE) Endpoint Slack
Number of Endpoints
300
250
200
150
100
50
0
252
Slack
(RISC_CO...
D0
psyn_gui-t>
Command Line
History
Errors & Warnings
Report

Cell Name	Ref
I_ALU	ALU
I_CONTROL	CONTR
I_DATA_PAT	DATA_
I_INSTRN_L	INSTRN
I_PRGRM_C	PRGRM
I_REG_FILE	REG_F
I_STACK_TO	STACK

Endpoints	Slack
I_ALU/Zro_Flag_reg/D0	-0.856182
I_ALU/Neg_Flag_reg/D0	-0.843835
I_ALU/Lachd_Result_reg[15]...	-0.705969
I_ALU/Lachd_Result_reg[14]...	-0.687741
I_ALU/Lachd_Result_reg[13]...	-0.638892
I_ALU/Lachd_Result_reg[10]...	-0.560465
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_PRGRM_CNT_TOP/PCint_...	-0.545585
I_ALU/Lachd_Result_reg[1]/...	-0.516620
I_ALU/Lachd_Result_reg[2]/...	-0.508664

```
Information: Updating design information... (UID-85)
psyn_gui-t> create_clock -p 3.5 [get_ports Clk]
1
psyn_gui-t> show_end_point_slack_histogram -numbins 8 -delay_type max
Information: Updating design information... (UID-85)
psyn_gui-t> Loading db file '/remote/trgl/rferner/PC_risc/libs/core.sdb'
```

Schematic with Selection



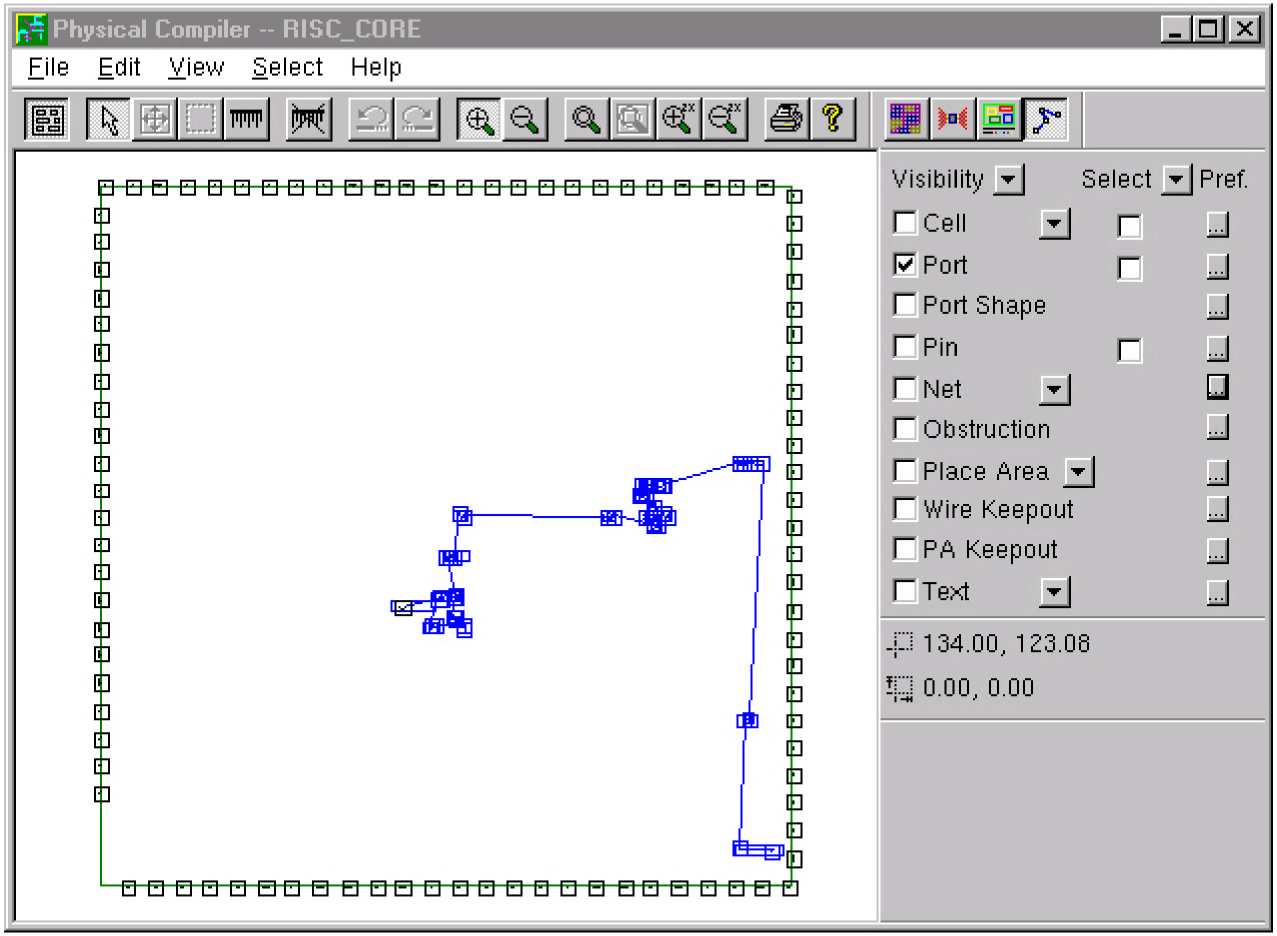
Cell Name	Ref
I_ALU	ALU
I_CONTROL	CONTR
I_DATA_PAT	DATA_I
I_INSTRN_L	INSTRN
I_PRGRM_C	PRGRN
I_REG_FILE	REG_F
I_STACK_TD	STACK

```
Information: Updating design information... (UID-85)
psyn_gui-t> create_clock -p 3.5 [get_ports Clk]
1
psyn_gui-t> show_end_point_slack_histogram -numbins 8 -delay_type max
Information: Updating design information... (UID-85)
psyn_gui-t> Loading db file '/remote/trg1/rferner/PC_risc/libs/core.sdb'
```

```
psyn_gui-t>
```

Pin: Zro_Flag_reg/D0

Floorplan with Selected Path



Visibility Select Pref.

☐ Cell ☐

☑ Port ☐

☐ Port Shape

☐ Pin ☐

☐ Net

☐ Obstruction

☐ Place Area

☐ Wire Keepout

☐ PA Keepout

☐ Text

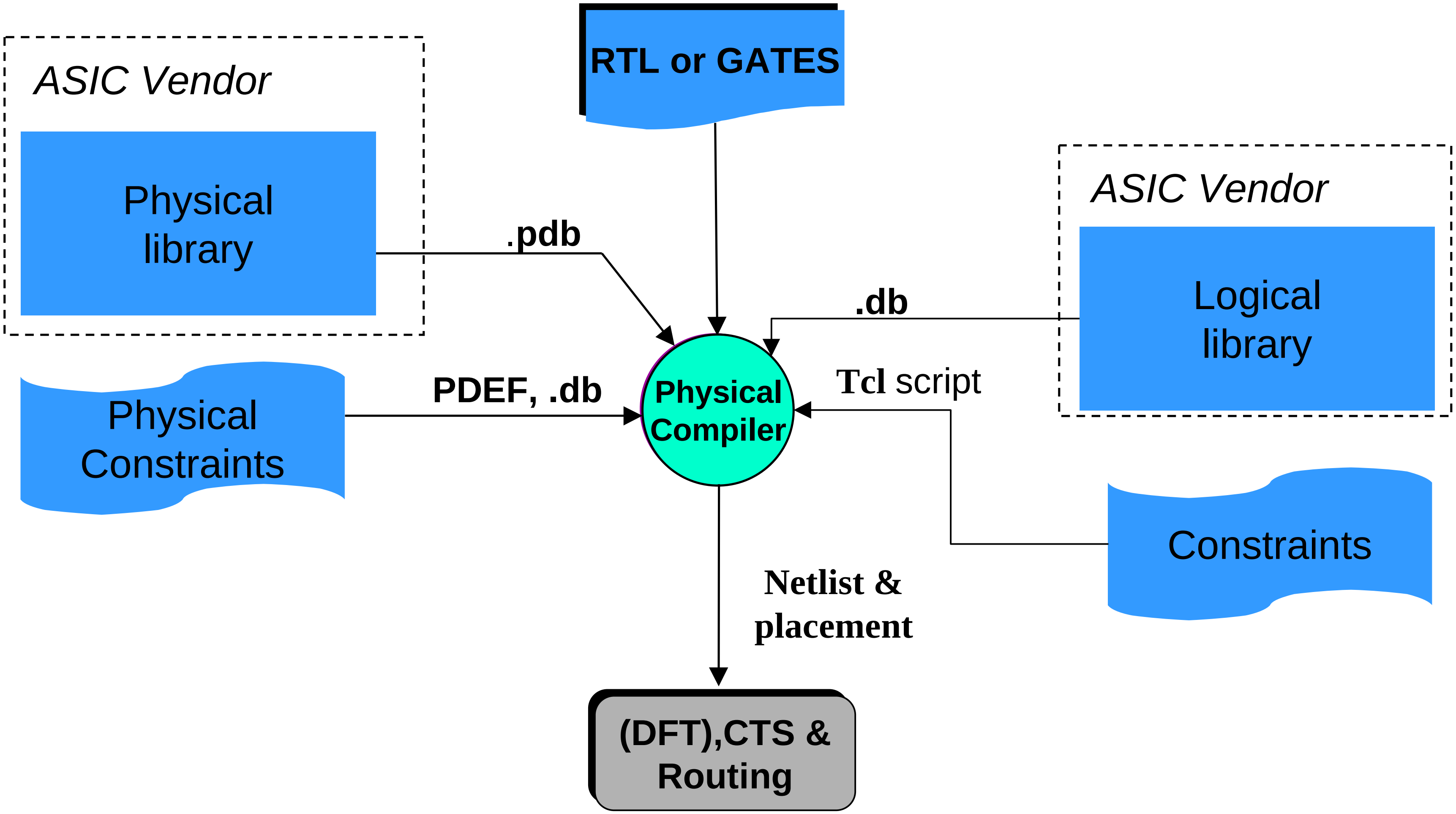
134.00, 123.08

0.00, 0.00

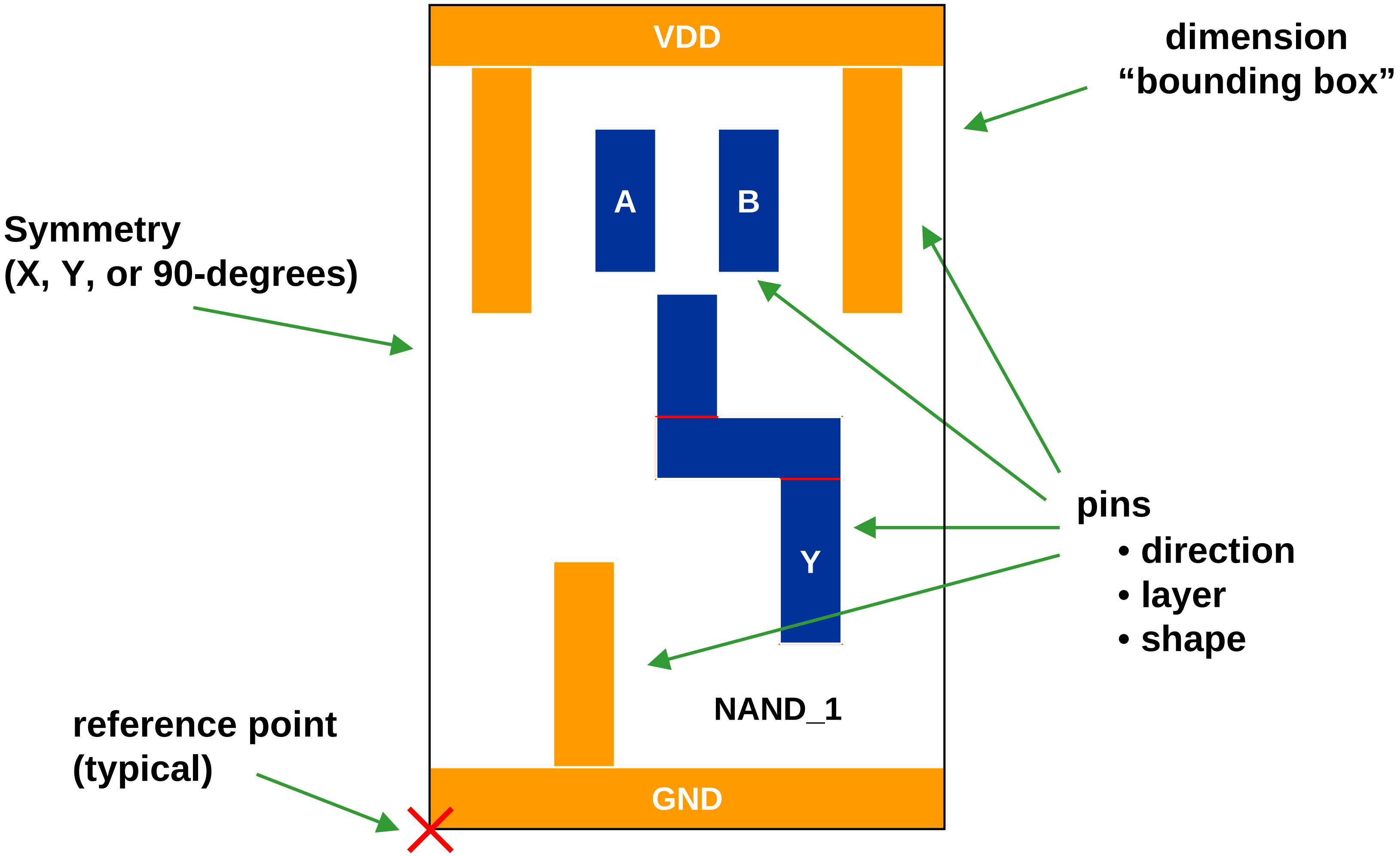
Agenda

- Introduction
  - Problems
  - Current Flows and Issues
  - Synopsys Physical Synthesis Solution
- What is Physical Compiler?
- **Running Physical Compiler**
- Physically Integrated Methodologies
- Summary

Data Required to Run PC : PC Input/Output files



LEF - Cell Data (Physical Library)



dimension
“bounding box”

Symmetry
(X, Y, or 90-degrees)

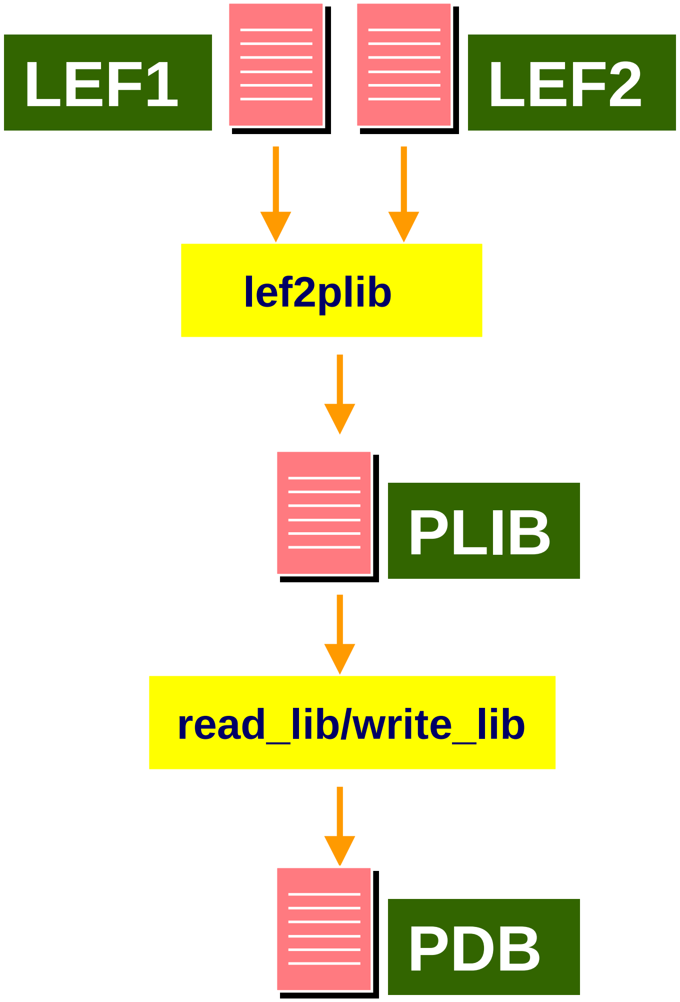
pins

- direction
- layer
- shape

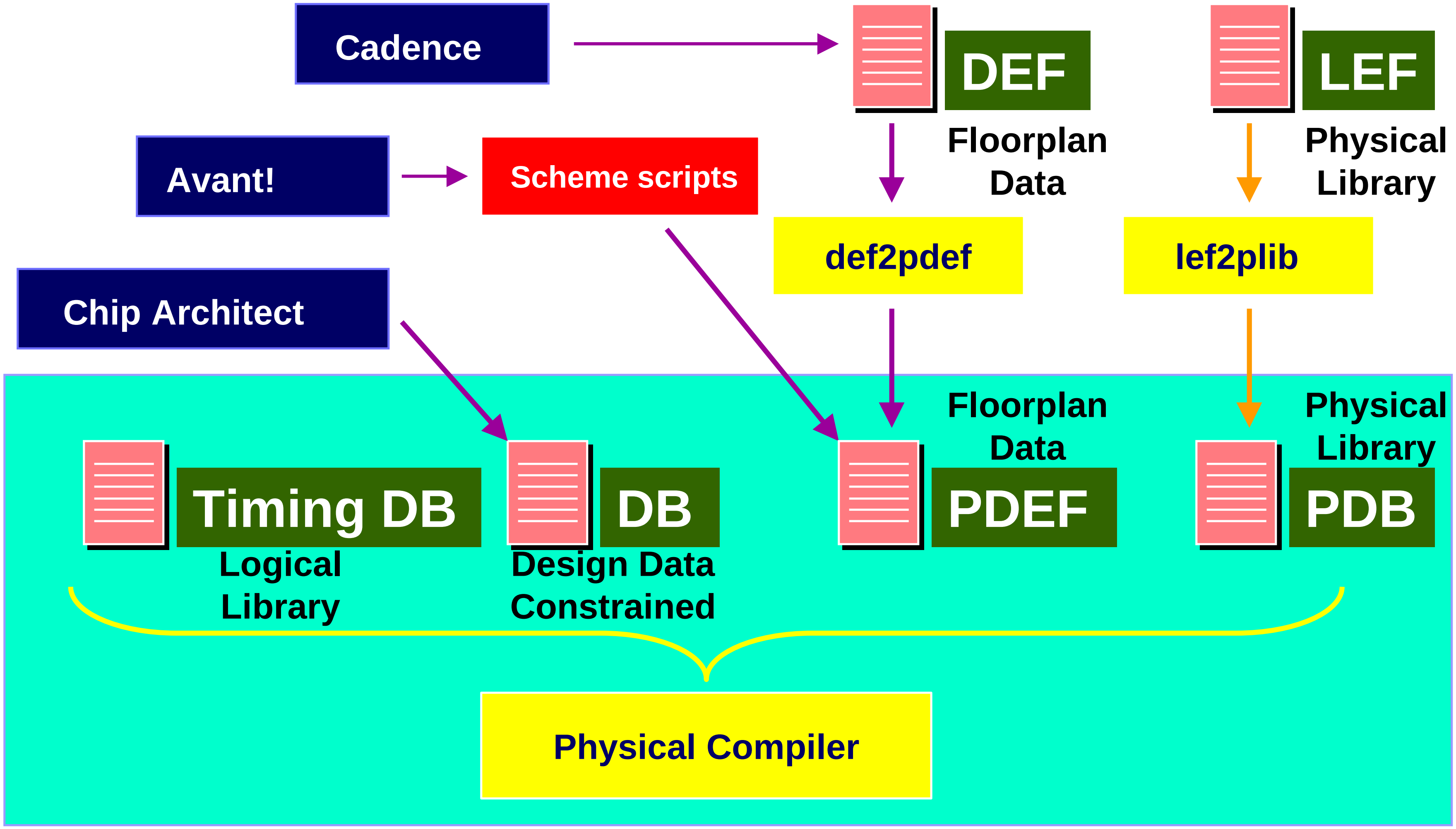
reference point
(typical)

Physical Library Utility - lef2plib

- ‘lef2plib’ is part of the standard installation
- Converts LEF syntax to Synopsys physical library syntax (PLIB)
- Physical library file can be saved into a binary DB (PDB) using read_lib /write_lib commands



Input Files – Floorplan Data

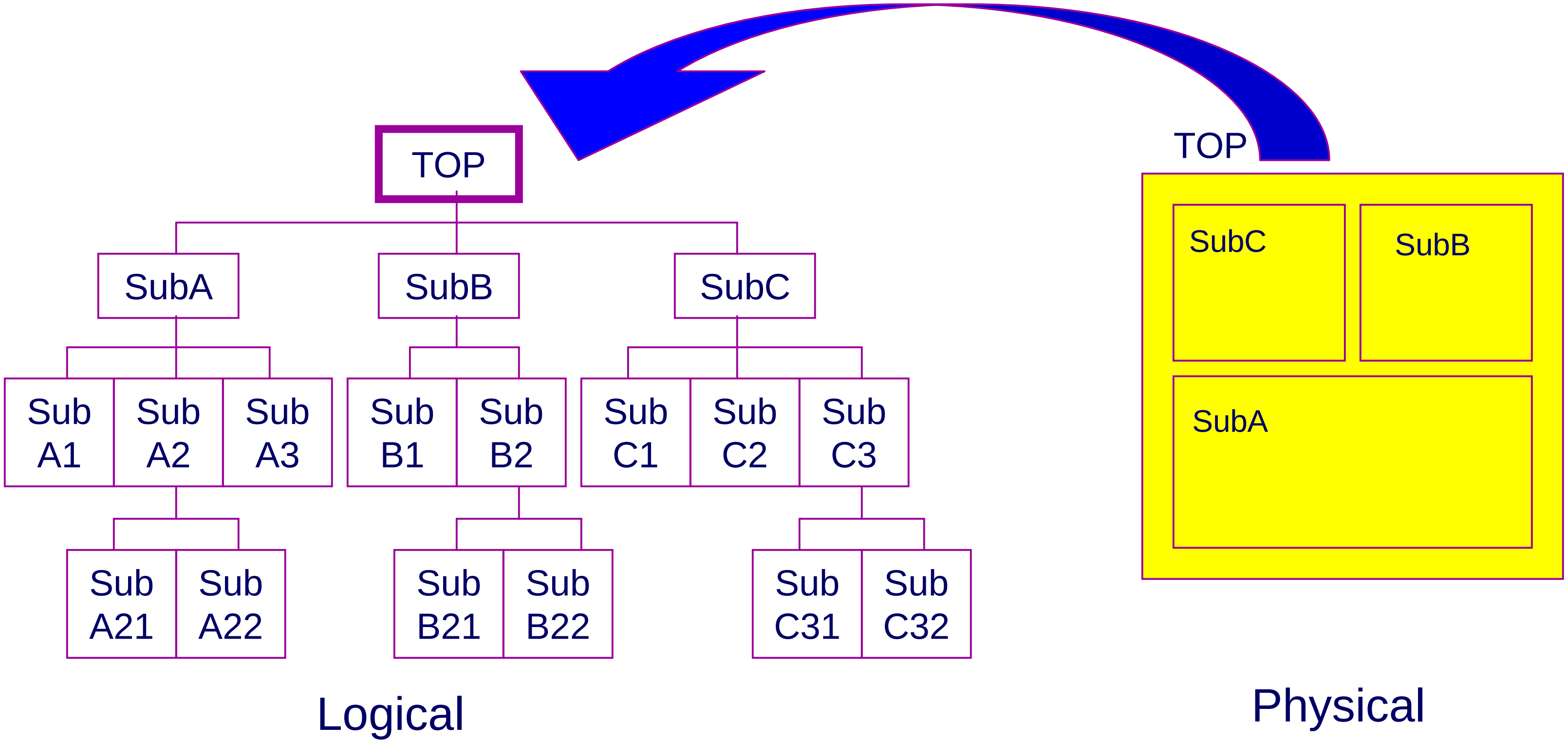


Input Files –
Logical and Physical Hierarchy

- The logical hierarchy is maintained throughout the Physical Compiler flow
- Physical Compiler manages both physical and logical hierarchies
- Both are persistently stored in the .db file

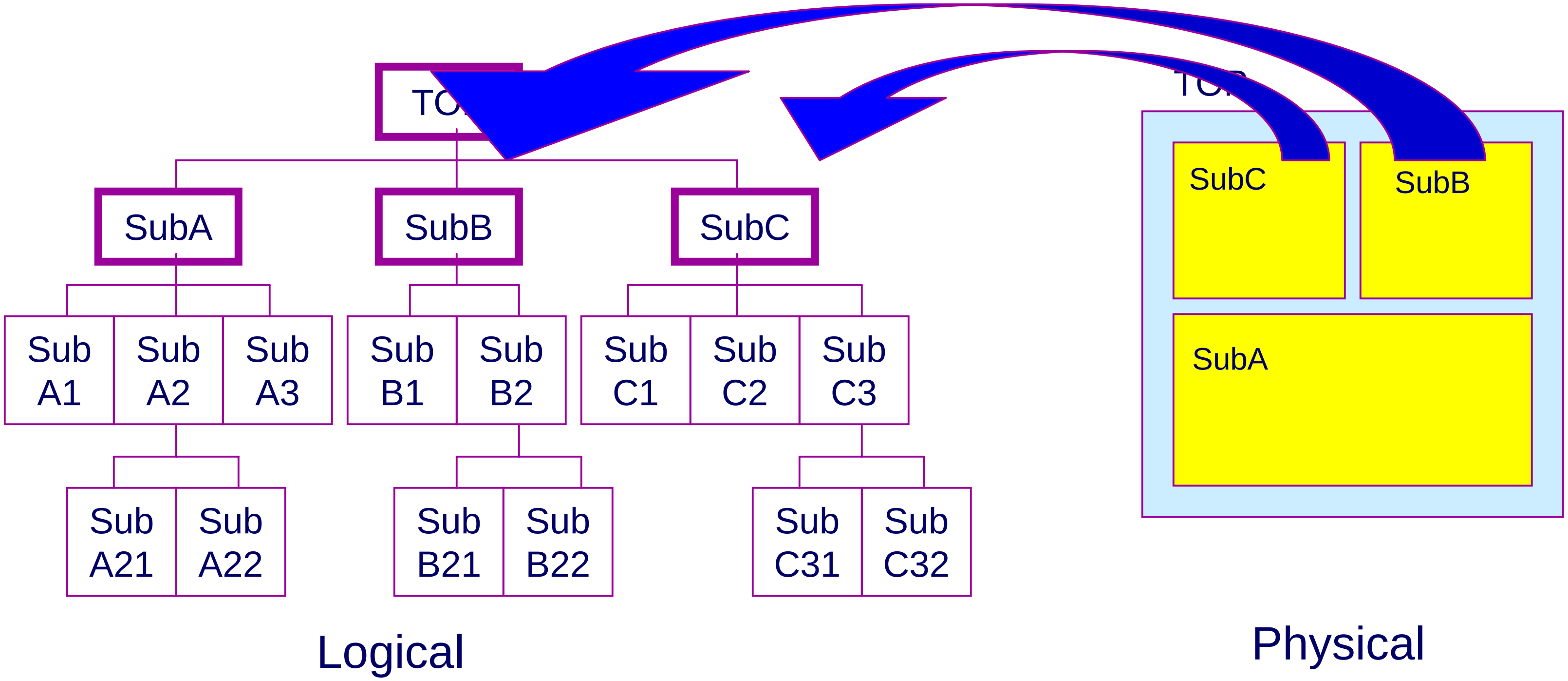
Logical & Physical Hierarchy Example - Optimizing a Whole Chip

- Only 1 PDEF is annotated to ‘TOP’
- Physical hierarchy is flat

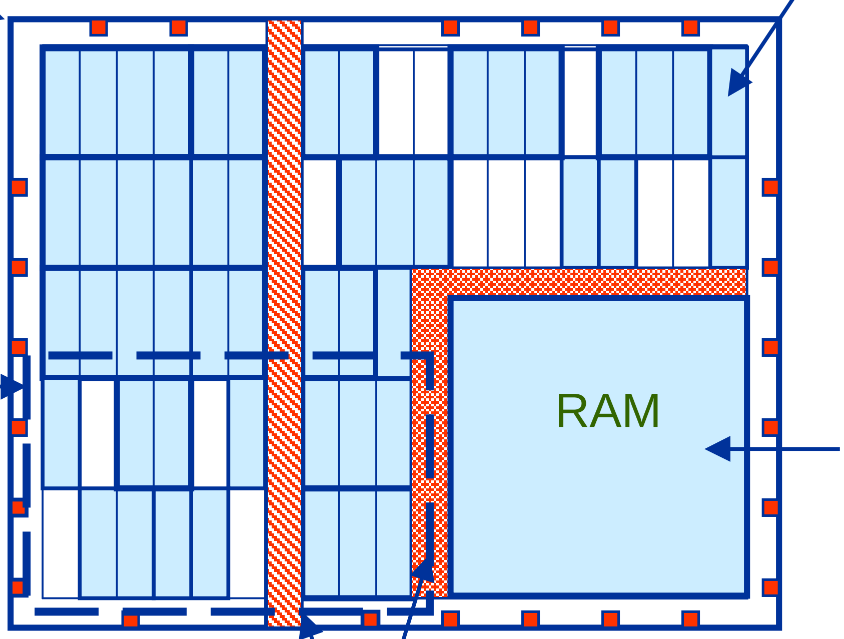


Logical & Physical Hierarchy Example - Optimizing Floorplan Blocks

- Three PDEFs are annotated to modules
- Floorplan hierarchy is maintained



Floorplanning Objects



Cluster

Hard Boundary

Site Arrays

Array of placement sites

Region

Soft Boundary

Fixed Cells

e.g. RAM placement

Port Locations

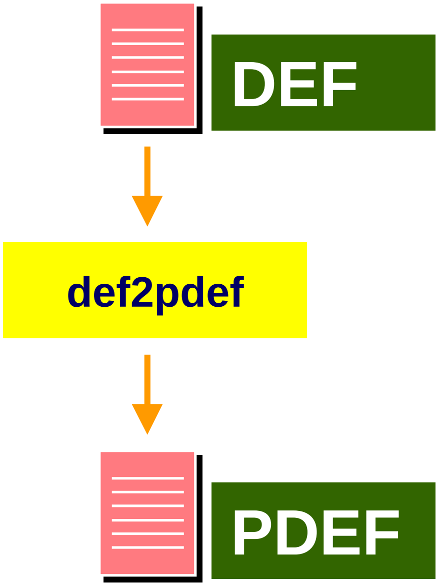
Signal I/O

Blockages

Power Routes
Placement Blockages

Floorplan Utility - def2pdef

- ‘def2pdef’ is part of the standard installation
- Converts DEF floorplan information to IEEE PDEF syntax
- DEF v5.3 is supported
- Compiled technology file (.pdb) must exist



Handling Obstructions

- PDEF describes all floorplan data
- Power nets can be full or partial blockages
- RAMs can have placement keepouts
- User can add additional obstructions
  - Routing layers or placement

Floorplanning Commands

- PC provides several commands that can be used to help floorplan the design (if the incoming PDEF needs additional information).

Command	
set_placement_area	**-> creates the core area for coarse placement**
create_site_row	**-> creates sites for detailed placement**
create_obstruction	**-> create a layer specific or placement obstruction**
set_cell_location	**-> sets the x y location of a specific cell**
set_port_location	**-> sets the x y location of a specific port**
set_dont_touch_placement	**-> creates the 'fixed_placement' restriction for a cell**
set_bounds	**-> controls grouping of specific cells**
set_keepout_margin	**-> creates keepouts for specific cells**
set_dont_touch	**-> prevents optimization but allows placement**
set_ideal_net	**-> specifies net as having zero weight for placement - prevents clumping of cells. Typically used for clocks, resets and test enables, etc.**

Setting up for Physical Compiler

- Create your RTL as usual
- Apply your design constraints as usual
- Invoke your TCL scripts as usual

- Apply floorplan data

- Forget about wireload models

Example RTL2PG Script

```
# Compile RTL using floorplan information.
read_verilog  top_bob.v
current_design  uncle
uniquify
link
source constraints.tcl
# Read Floorplan Info…
read_pdef bob.pdef
# Compile…
compile_physical -congestion
report_timing -input_pins -nets -physical
write_pdef -v3.0 -output placed_bob.pdef
write -format db -hierarchy -output placed_bob.db
exit
```

```
# constraints.tcl
set_operating_conditions { WCCOM }
set_load [ load_of Core/Buf1/A ] [all_outputs]
create_clock -period 8 CLK
set_dont_touch_network [ get_clock CLK ]
set_input_delay 0.5 -clock CLK [all_inputs]
set_output_delay 1.5 -clock CLK [all_outputs]
set_max_fanout 5 uncle
set_max_transition 0.8 uncle
```

Example G2PG Script

```
# Gates to placed gates using floorplan information.
read_db  bob.db
current_design  uncle
link
source constraints.tcl
# Read Floorplan Info…
read_pdef bob.pdef
# Compile…
physopt -congestion
report_timing -input_pins -nets -physical
write_pdef -v3.0 -output placed_bob.pdef
write -format db -hierarchy -output placed_bob.db
exit
```

```
# constraints.tcl
set_operating_conditions { WCCOM }
set_load [ load_of Core/Buf1/A ] [all_outputs]
create_clock -period 8 CLK
set_dont_touch_network [ get_clock CLK ]
set_input_delay 0.5 -clock CLK [all_inputs]
set_output_delay 1.5 -clock CLK [all_outputs]
set_max_fanout 5 uncle
set_max_transition 0.8 uncle
```

Running compile_physical (RTL2PG)

- Uses all floorplan information present
  - Optimizes RTL design based on placement
  - Produces a fully legal result
- Same switches as ‘compile’ plus

  -congestion

  -timing_driven_congestion

  -congestion_effort

Running physopt (G2PG)

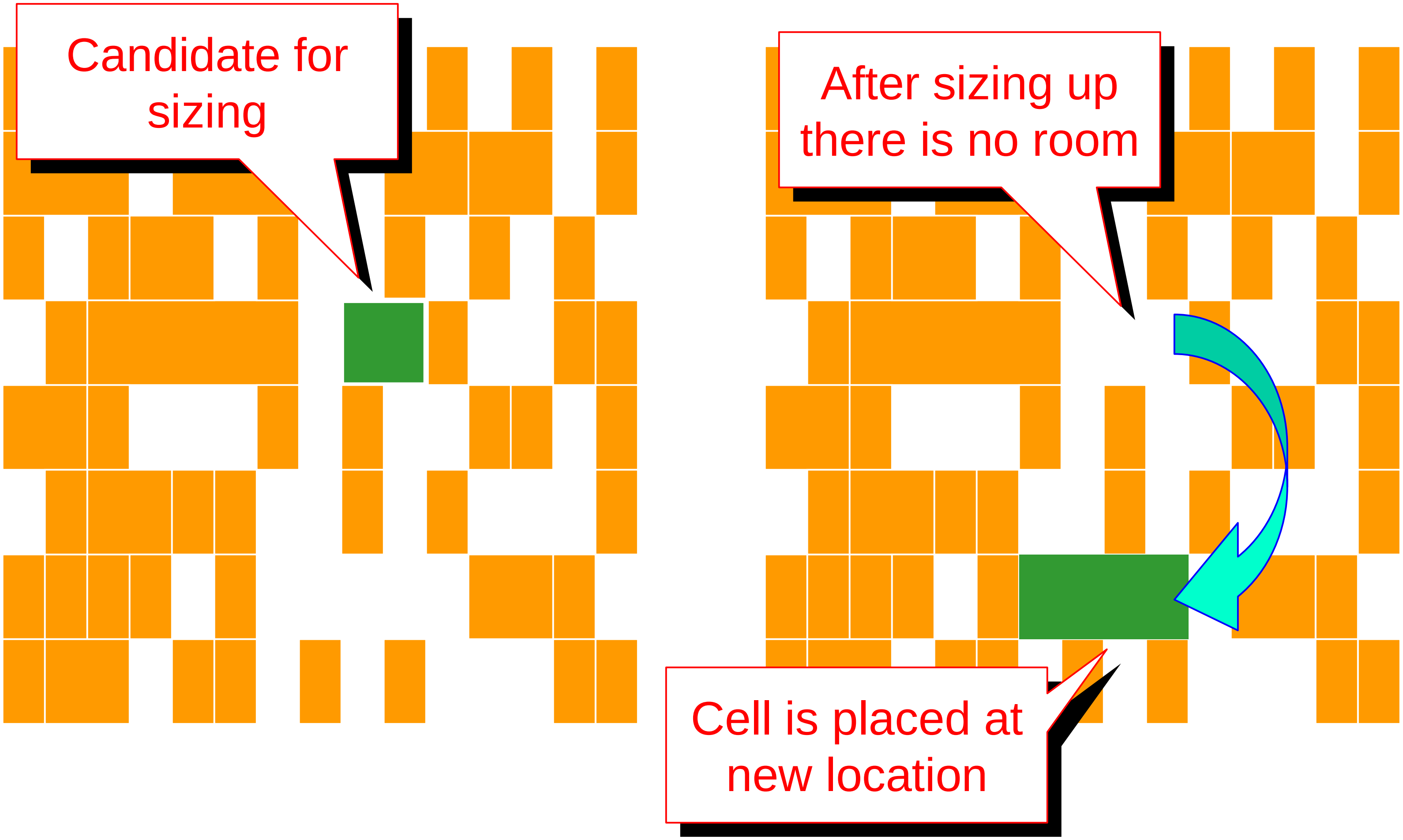
- Uses all floorplan information present
  - Optimizes an existing gate level design based on placement
  - Produces a fully legal result
- Same switches as ‘compile_physical’ plus

  -check_only

Incremental Optimizations

- `physopt -incremental`
  - Uses existing placement as starting point for `physopt`, initial placement is skipped
- `physopt -incremental -eco`
  - Allows the merge of new / changed leaf cells
- `physopt -incremental -post_route`
  - Maintains backannotated timing
- `–size_only`
  - Allows only cell sizing to take place
  - Works on all the above incremental modes

Incremental -size_only

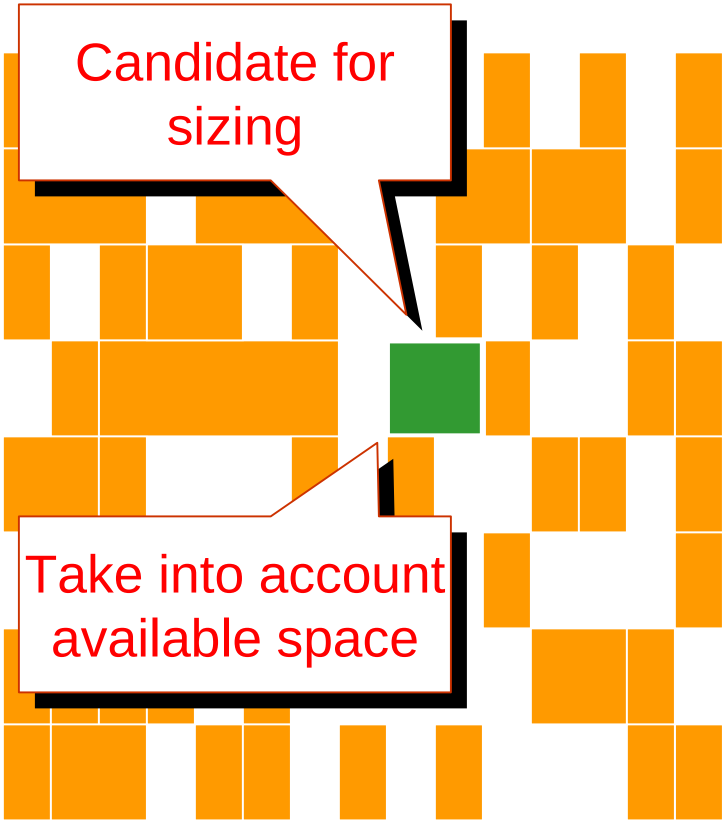


Candidate for sizing

After sizing up there is no room

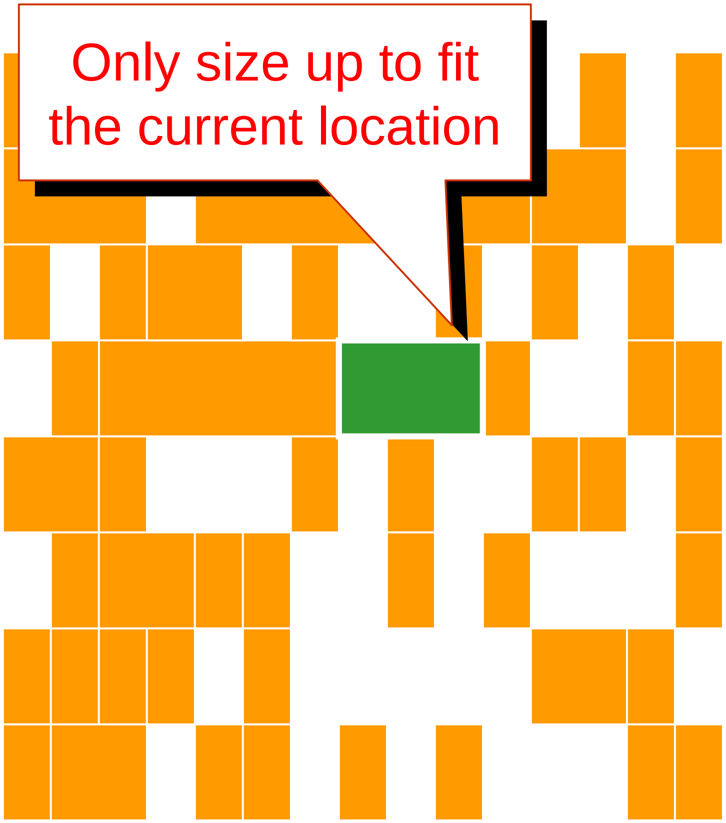
Cell is placed at new location

Incremental –in_place_size_only



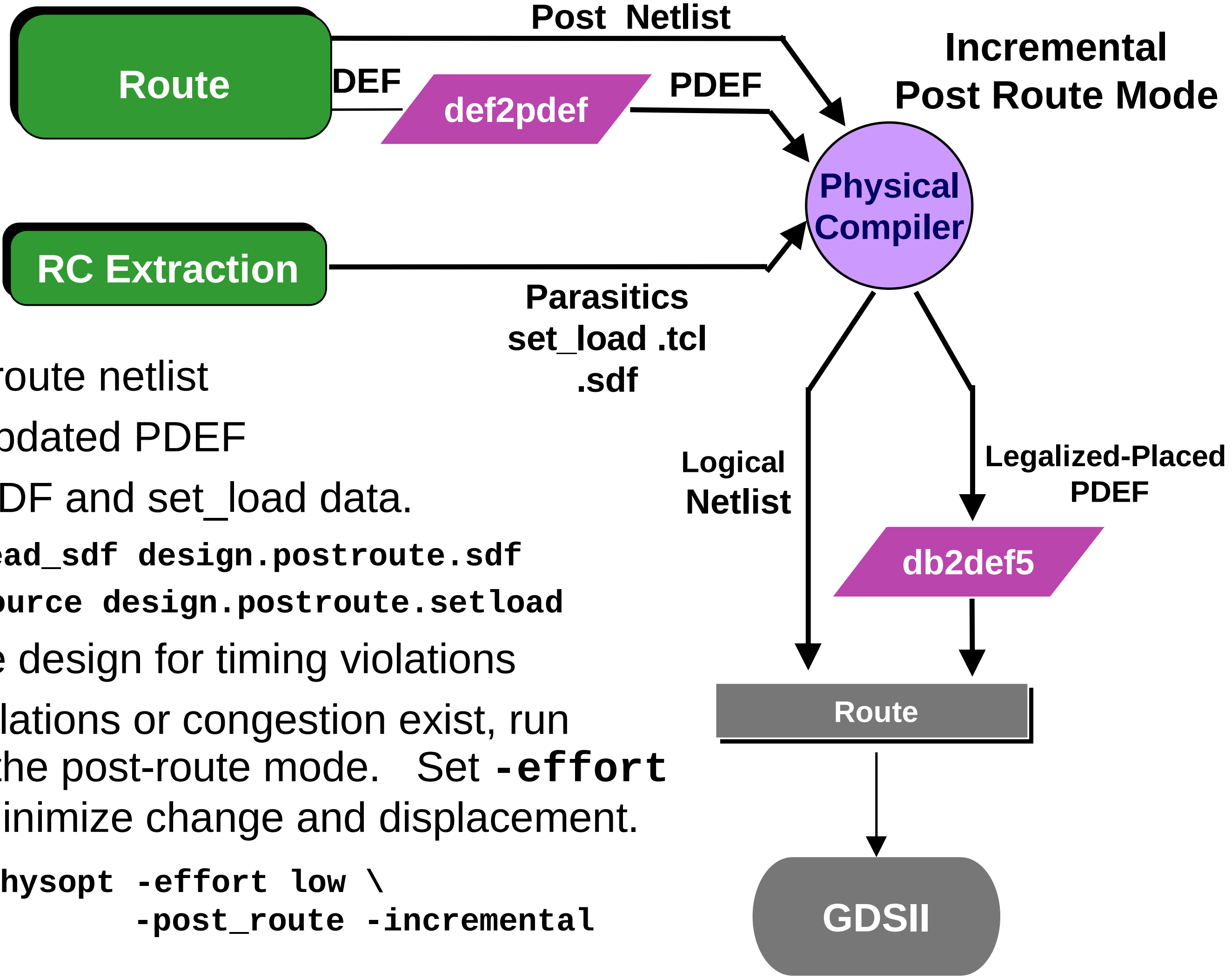
Candidate for sizing

Take into account available space



Only size up to fit the current location

Post-Route Physical Compiler Flow



- Read post route netlist
- Read the updated PDEF
- Read the SDF and set_load data.

```
psyn_shell> read_sdf design.postroute.sdf
psyn_shell> source design.postroute.setload
```

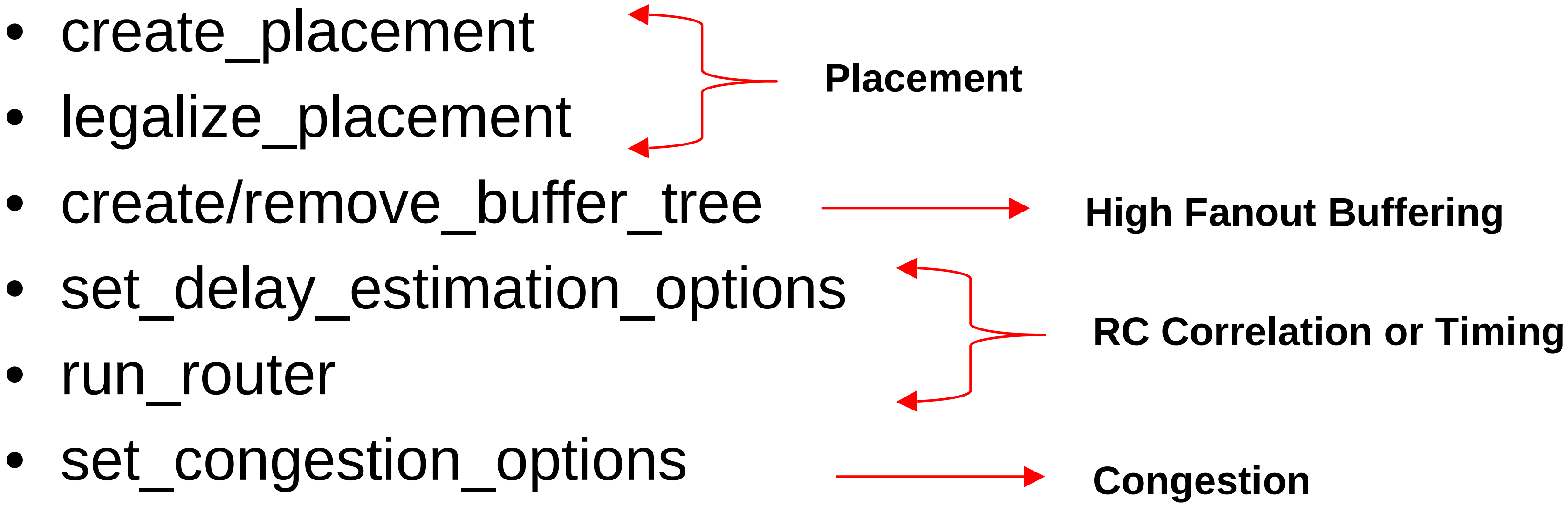
- Analyze the design for timing violations
- If timing violations or congestion exist, run physopt in the post-route mode. Set **`-effort`** to `low` to minimize change and displacement.

```
psyn_shell> physopt -effort low \
                    -post_route -incremental
```

Congestion-Driven Placement

- Additional checks provided during placement prevent routing congestion.
  - Congestion estimates used to modify placement
  - Techniques such as “adaptive tuning of cell density” in areas of high congestion
- Use congestion mode only when the design has a congestion problem.

Tactical Commands



- create_placement
- legalize_placement
- create/remove_buffer_tree
- set_delay_estimation_options
- run_router
- set_congestion_options

Placement

High Fanout Buffering

RC Correlation or Timing

Congestion

Reporting Commands

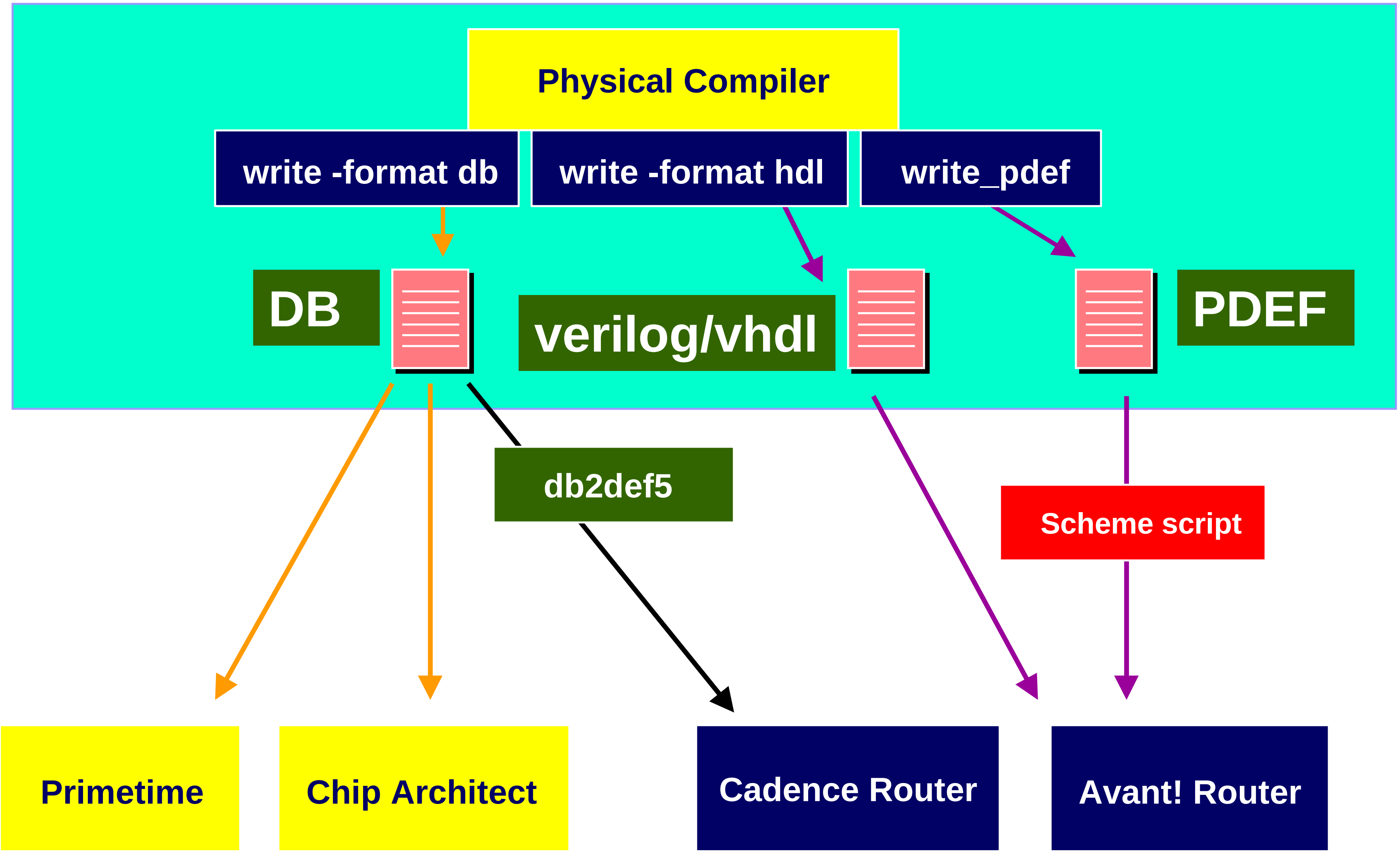
- PC provides several ‘report’ commands that detail physical information

- report_area -physical **-> shows the size of the core area and aspect ratio**
- report_lib -physical **-> shows physical library information**
- report_cell -physical -only_physical **-> shows the cell location and orientation**
- report_net -physical -only_physical **-> shows the net total length and pre-routes**
- report_clusters **-> shows the physical cluster hierarchy**
- report_port -physical **-> shows the physical location of the port**
- report_design -physical **-> shows size, area, aspect ratio, orientation, utilization and obstruction information.**
- report_congestion **-> shows the congestion prediction for the current placement.**
- report_timing -physical **-> shows location of pins and capacitive loads on the nets in the reported timing path.**
- report_keepout_margin **-> lists keepout margins for specified cells**
- report_bounds **-> lists type and size of cell groupings**

Powerful and Flexible

- Different sets of commands/switches address different problems
  - Each design has it’s own unique problems
  - The commands provide flexible, powerful solutions for all your designs
- Proceed step by step, and check your logs
  - Use the reporting commands and the GUI

Output Files



Preparing Routing Files

- psyn_shell-t>

```
change_names -rules verilog
write -format db -hierarchy -output design.db
write -format verilog -hierarchy -output design.v
write_pdef -v3.0 -output design.pdef
```

Creating DEF

- All floorplan data is written to DEF v5.2
- UNIX > db2def5

```
-search “lib_dir1> <lib_dir2>”
        Directory containing library files
-pdb <pdb1>   -pdb <pdb2>
        Name of PDB file(s) to use
-out <DEF_file>
<design_DB>
```

Agenda

- Introduction
  - Problems
  - Current Flows and Issues
  - Synopsys Physical Synthesis Solution
- What is Physical Compiler?
- Running Physical Compiler
- **Physically Integrated Methodologies**
- Summary

Physical Compiler in the Flow

- Floorplanning from Chip Architect
- Timing analysis with PrimeTime
- Scan Methodology
  - Placement-based scan ordering
- Power Optimization
  - Clock gating and logic optimization
- Datapath Optimization
  - Use structured placement from MC (or not)

Agenda

- Introduction
  - Problems
  - Current Flows and Issues
  - Synopsys Physical Synthesis Solution
- What is Physical Compiler?
- Running Physical Compiler
- Physically Integrated Methodologies
- **Summary**

Summary - Physical Compiler

- Concurrent Synthesis + Placement Tool
  - Produces netlist & highly routable placement that meet timing
  - Improves Productivity, reduces iterations
- Easy to Adopt
  - Proven in Cadence, Avant! and IBM Flows
- Best Technology
- Proven Customer Success
  - ~~90+~~ ~~150+~~ ~~180+~~ 200+ Tapeouts

SNUG
SAN JOSE
2002

Hierarchical Physical Synthesis
with
Interface Logic Models

Simon Koval
Physical Synthesis CAE

Outline

- ILM Goals
- Modeling Concepts
- ILM Flow
- ILM and Test
- Modeling Results
- Roadmap
- Summary

Outline

- **ILM Goals**
- Modeling Concepts
- ILM Flow
- ILM and Test
- Modeling Results
- Roadmap
- Summary

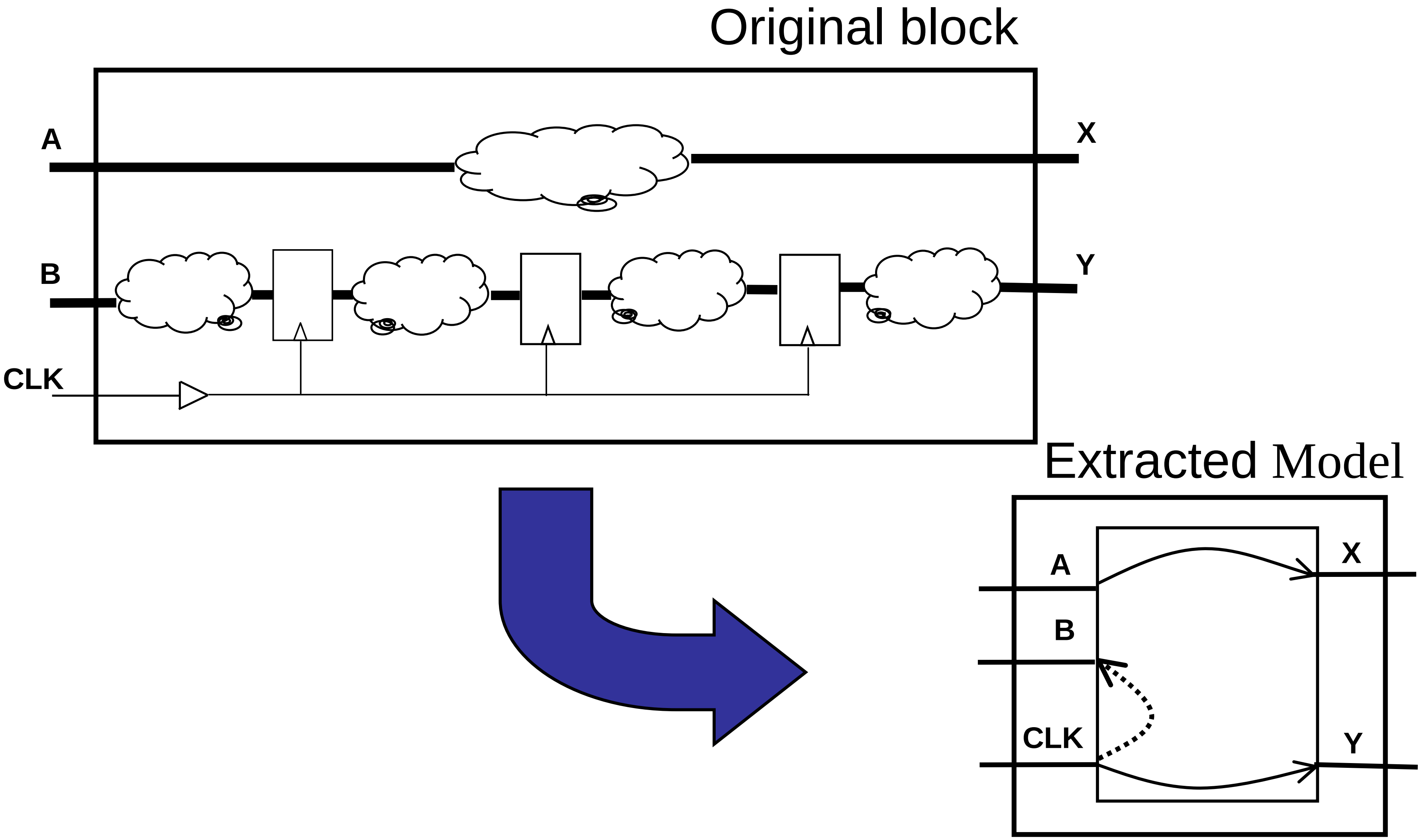
ILM Goals

- Provide design abstraction capability that can be used throughout the hierarchical design implementation flow
- Improve runtime and capacity compared to using original netlist
- Generate highly accurate model
- Make ILMs easy to use
- Easy to debug

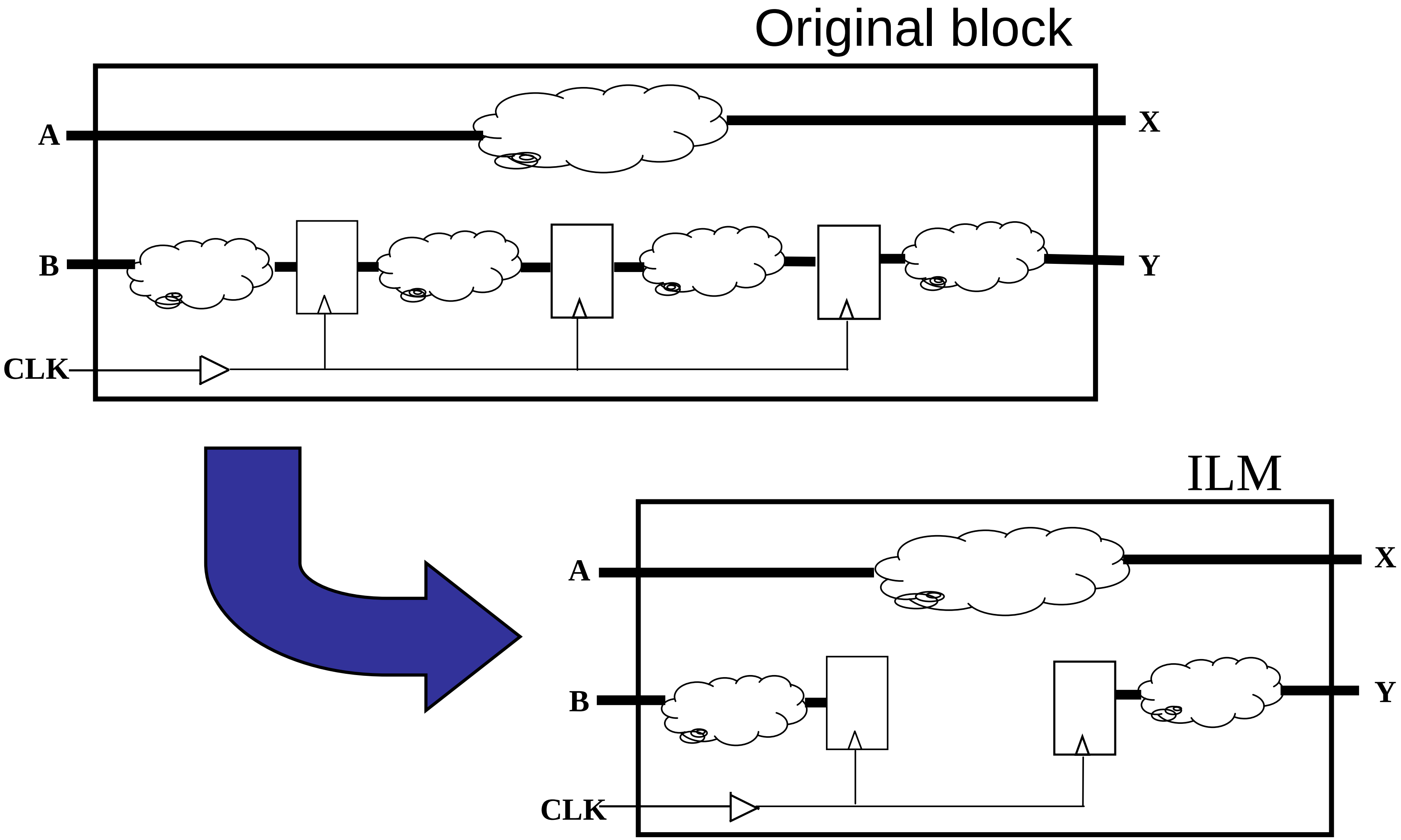
Outline

- ILM Goals
- **Modeling Concepts**
- ILM Flow
- ILM and Test
- Modeling Results
- Roadmap
- Summary

Extracted Timing Models



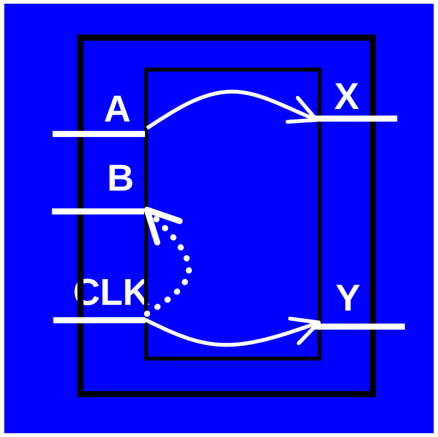
ILM Concept



Original block

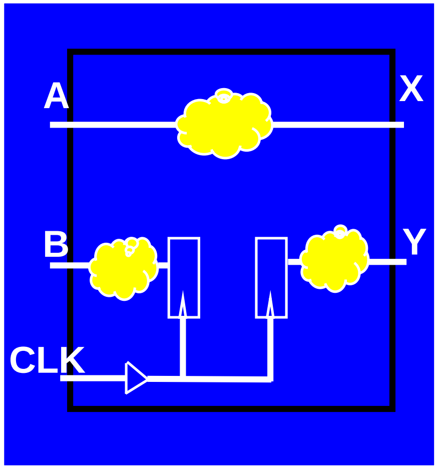
ILM

ETM vs. ILM Comparison



Extracted Timing Models

- Moderate model generation times, runtime improvement, reasonable accuracy
- Use for IP Reuse, 3rd Party Tools, non-STA Tools
- Hides implementation details



Interface Logic Models

- Fast model generation times, highly accurate, context independent model
- Use for Hierarchical STA / Chip-level optimizations
- Improves runtime and decreases memory for chip-level tasks

PrimeTime ILM vs. Synthesis ILM

- PT ILM is flat
- Has no physical information
- Written out as a verilog netlist
- Supports distributed parasitics and SDF
- Used for STA

- PC ILM maintains original block logical hierarchy
- Can be generated with physical information
- Can be written out in DB format with no loss of attributes
- Supports set_load and SDF
- Can be used for STA as well as design implementation
- Placement and back annotation data can be propagated up to the top-level

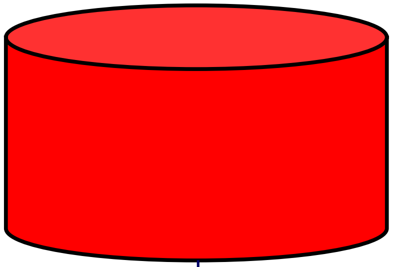
Outline

- ILM Goals
- Modeling Concepts
- **ILM Flow**
- ILM and Test
- Modeling Results
- Roadmap
- Summary

ILM flow

- Define clocks for the design
- Apply constraints on the design (optional)
- Apply back annotation data (set_load & SDF) onto the design (optional step for post route flow)
- Create ILM for some/all top-level blocks
- Create pdbs for the blocks in CA (optional)
- Replace original designs with ILMs
- Run physopt at the top-level
- Write out the top-level design and top-level PDEF

Creating ILM (with 2001.08-Psyn-Jet / 2002.05)

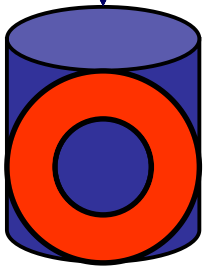


block db

(post physopt)

Physical Compiler

```
read_db block1.db; link
create_clock -period pd clk
set_clock_skew –ideal clk
identify_interface_logic [-latch_levels levels] [–ignore_ports port_list]
extract_ilm -out block1_ilm.db [–physical] –verbose
```



model db (replaces original design in memory)

ILM script example

```
read_db top.db

# Replace the original design with the ILM
remove_design block1
read_db block1_ilm.db
current_design top
link
read_pdef top.pdef

# Propagate placement and timing information for all ILMs
propagate_placement_up –adjust_location –verbose
propagate_annotated_delay_up

physopt
```

ILM script example (continued)

```
# Save the top-level design
write –f db –out top_level_only_post_physopt.db

# Write out the top-level PDEF to route the design
# in a hierachical manner
write_pdef –no_hierarchy –o top_level_only_post_physopt.pdef
exit


%# To write the top-level DEF from unix shell
% db2def5 –no_hierarchy top_level_only_post_physopt.db \
  –out top_level_only_post_physopt.def
```

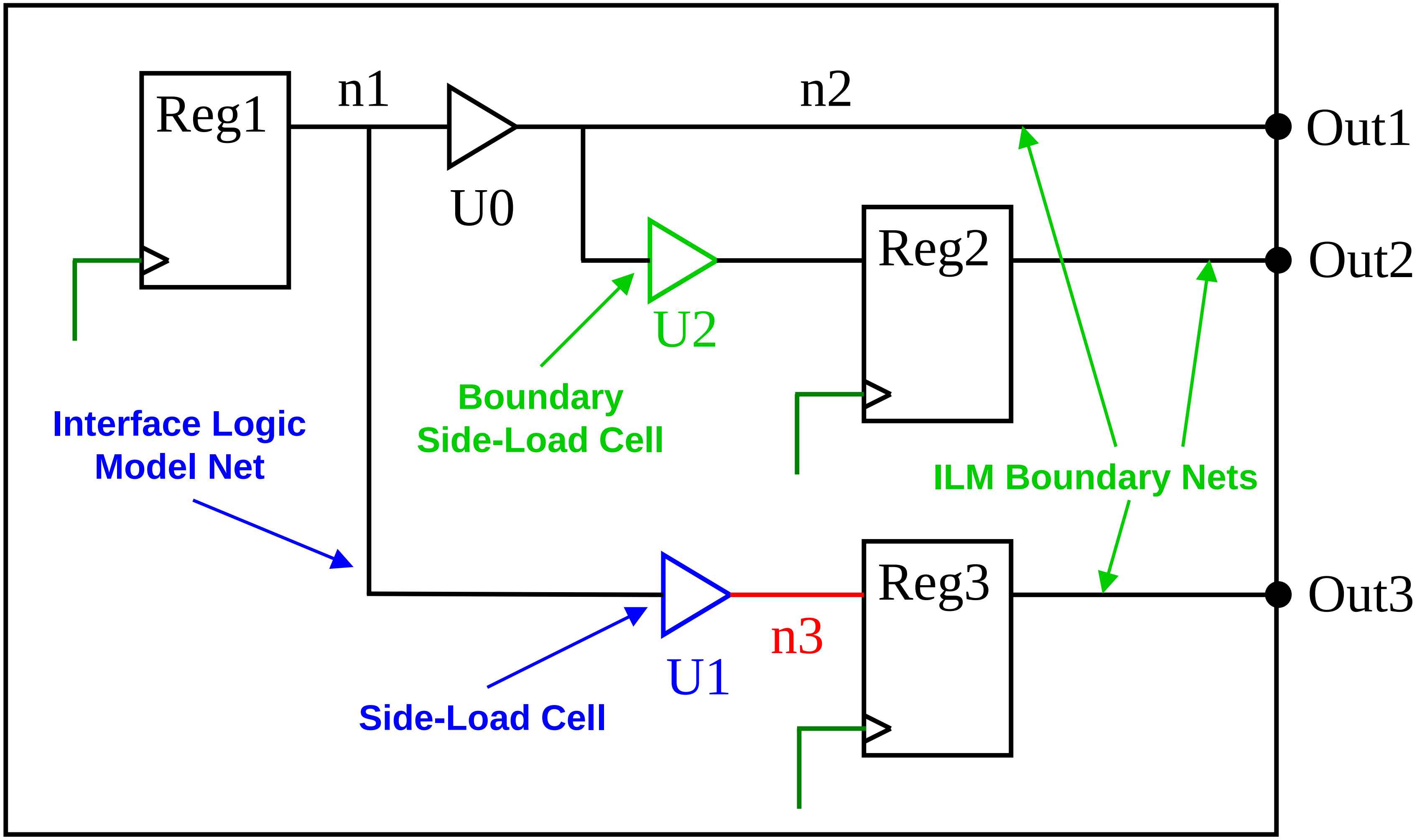
ILM Commands/Reporting

- **identify_interface_logic [-latch_levels levels] \
  [-ignore_ports port_list]**
- **extract_ilm -output filename [-physical] [-verbose] \
  [-include_side_load boundary | all | none]**
- **propagate_annotated_delay_up [cell_list]**
- **propagate_placement_up [-adjust_location] [-verbose] \
  [cell_list]**
- **report_area**
- **report_design**
- **report_annotated_delay**
- **report_cell –physical**
- **write_pdef [-no_hierarchy]**
- **db2def5 [-no_hierarchy]**

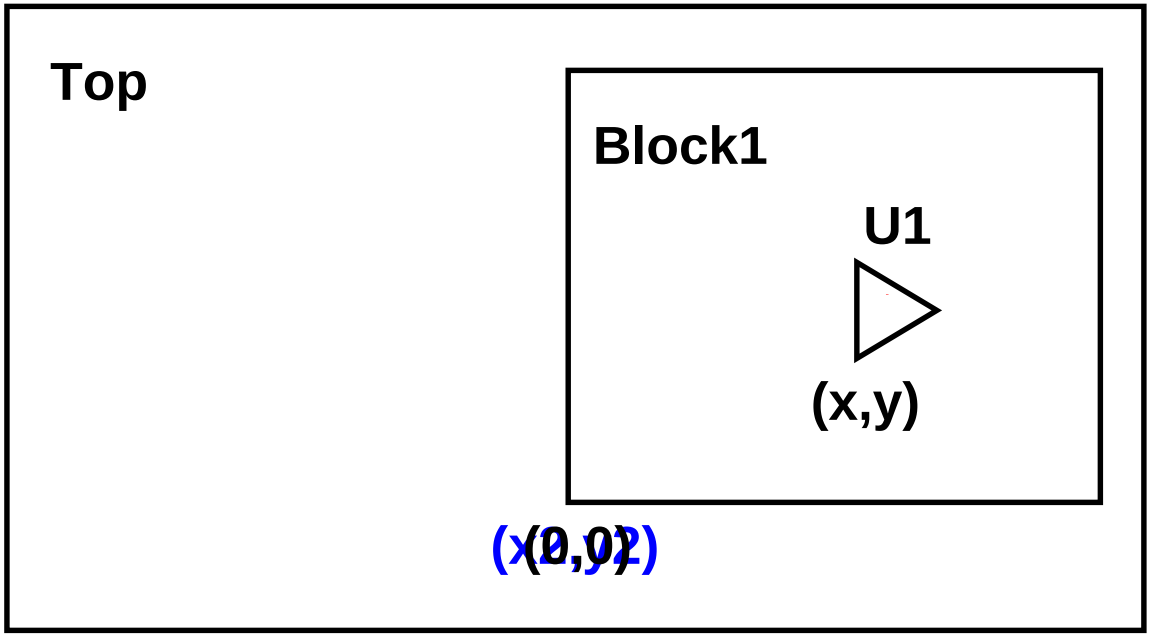
Identifying Interface Logic

- **`identify_interface_logic [-latch_levels levels] \`**
  `[-ignore_ports port_list]`
- Default behavior includes all interface latches in the ILM
- Use –latch_levels *levels* to limit the number of latch levels for which time borrowing can occur for latch chains that are part of the interface logic. For example, if *levels* = 1, then a path originating from an input port will continue through the first latch encountered, but will stop at the second latch in the path. The second latch is treated as an edge triggered register.
- Use –ignore_ports for ports like reset and scan_enable that fanout to all registers. Ignored ports are included in the ILM, but not fanout / fanin logic from input / output of these ports.

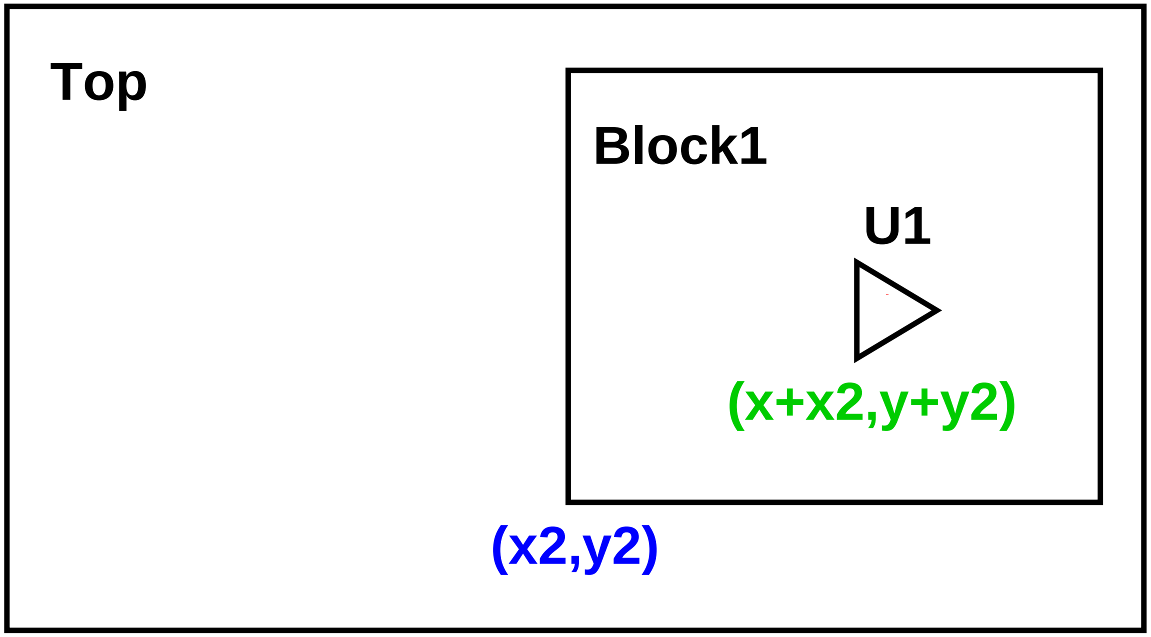
ILM Sideload Description



ILM Propagate Placement Up



ILM Propagate Placement Up

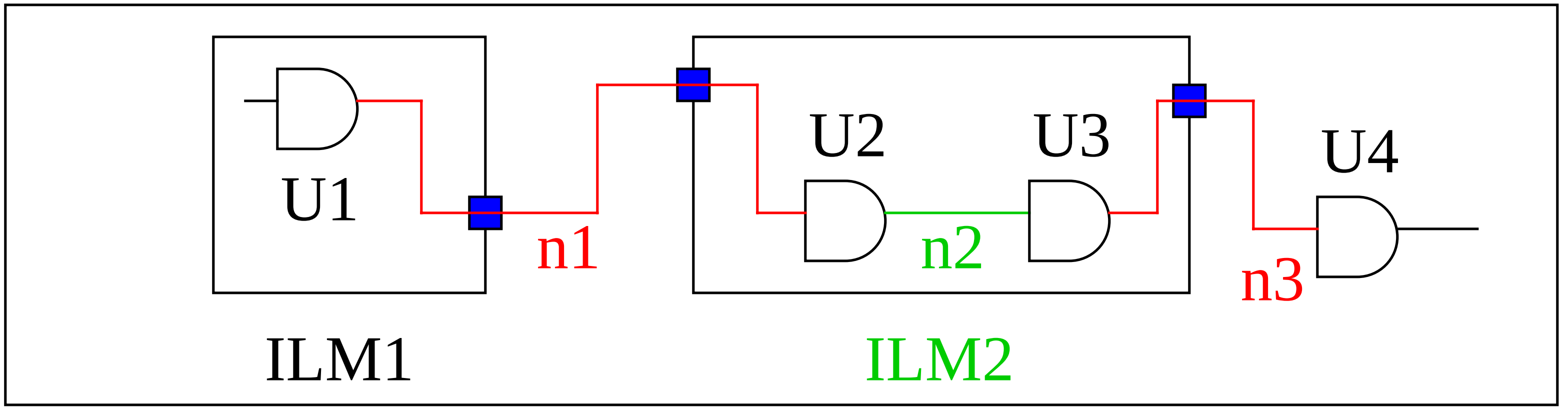


If the lower left co-ordinate of the reference design for Block1 is (0,0), when Block1 is placed in Top at (x2,y2), the location of the cell U1 placed at (x,y) in Block1 will become (x+x2, y+y2) after running the command propagate_placement_up –adjust_location.

ILM Delay Estimation

- Use propagate_annotated_delay_up before running physopt
- physopt uses ILM delay and capacitance annotations for nets fully within the ILM (i.e. ILM2/n2).
- For nets crossing ILM boundary, ILM cell and port locations are used to compute wirelength and estimate the delay (example n1, n3)

.



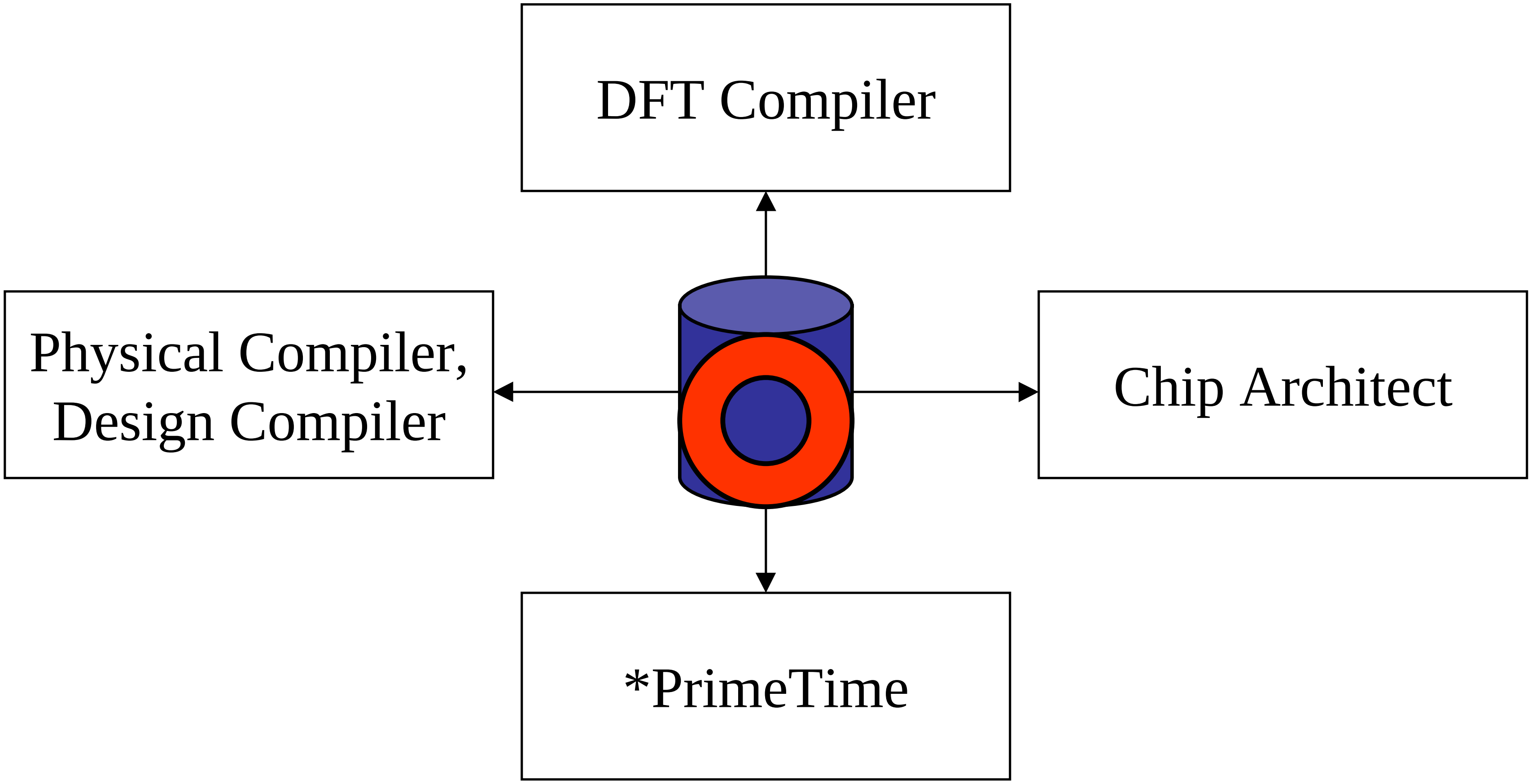
Obstructions for ILMs

- Create physical lib cell for Synthesis ILM block for providing obstruction information to PC.
  - Can be done using Chip Architect (write_abstraction) as .pdb or LEF.
  - Required for rectilinear blocks / over-the-block routing
- If an ILM block does not have a pdb cell, physopt will automatically derive the obstruction information
  - Derived obstruction is rectangular.
  - Creates a placement and all-layer routing obstruction over ILM blocks.
- Cells within the ILM are automatically marked as dont_touch and dont_touch_placement

ILM Recommendations

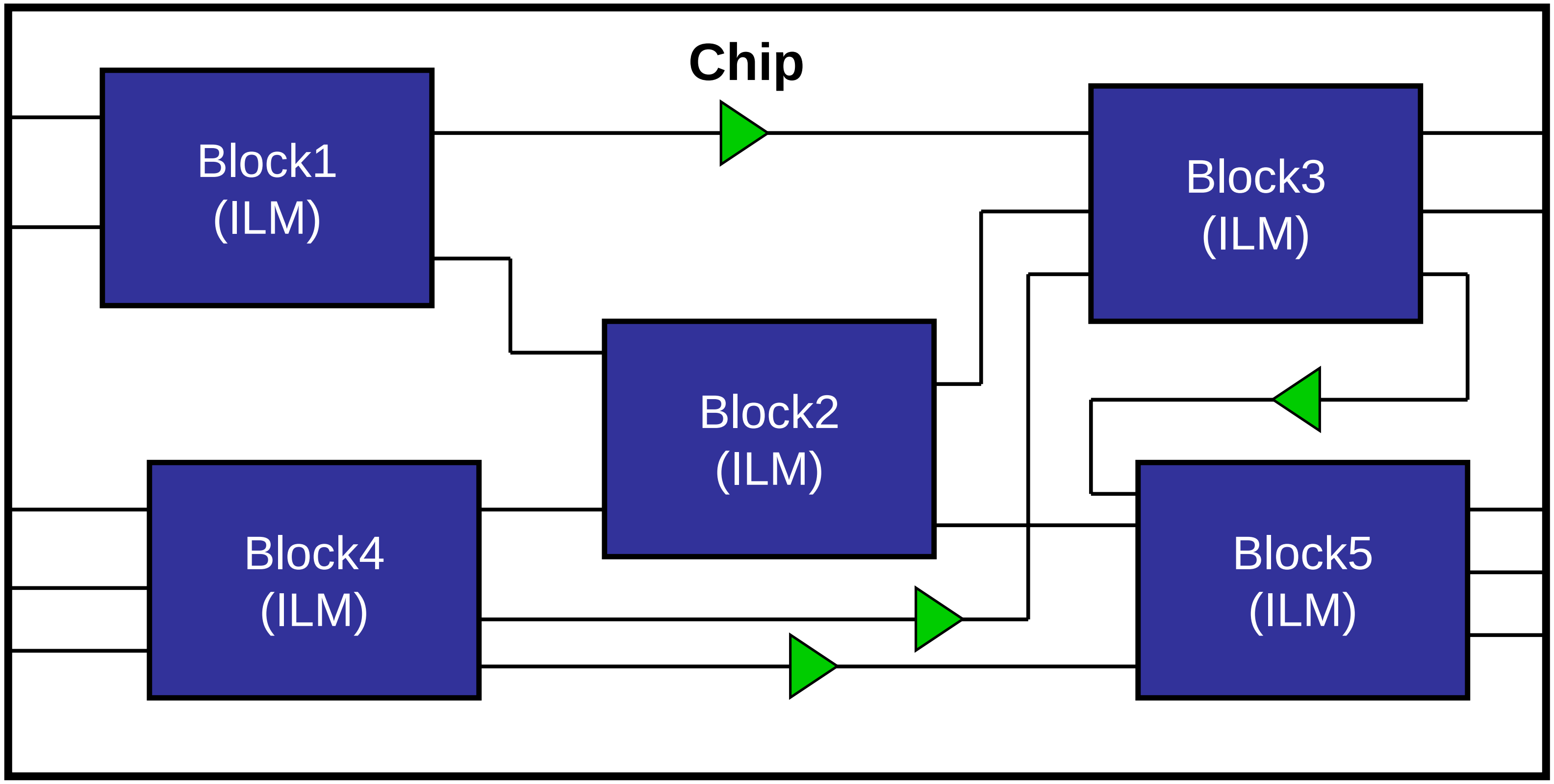
- Use –ignore_ports option to identify_interface_logic for ports such as reset, scan_enable that fanout to all registers (if not used then ILM could be excessively large).
- Specify the number of latch levels to be included by using the –latch_level option to identify_interface_logic.
- Use the –physical option to extract_ilm to create an ILM with physical information
- Use a pdb model for each ILM to not incur the restrictions caused by using a derived obstruction (routing obstruction on all layers & rectangular obstruction model).
- Designs with registered inputs and outputs will see the greatest reduction in size (original netlist vs. ILM)

Using ILM



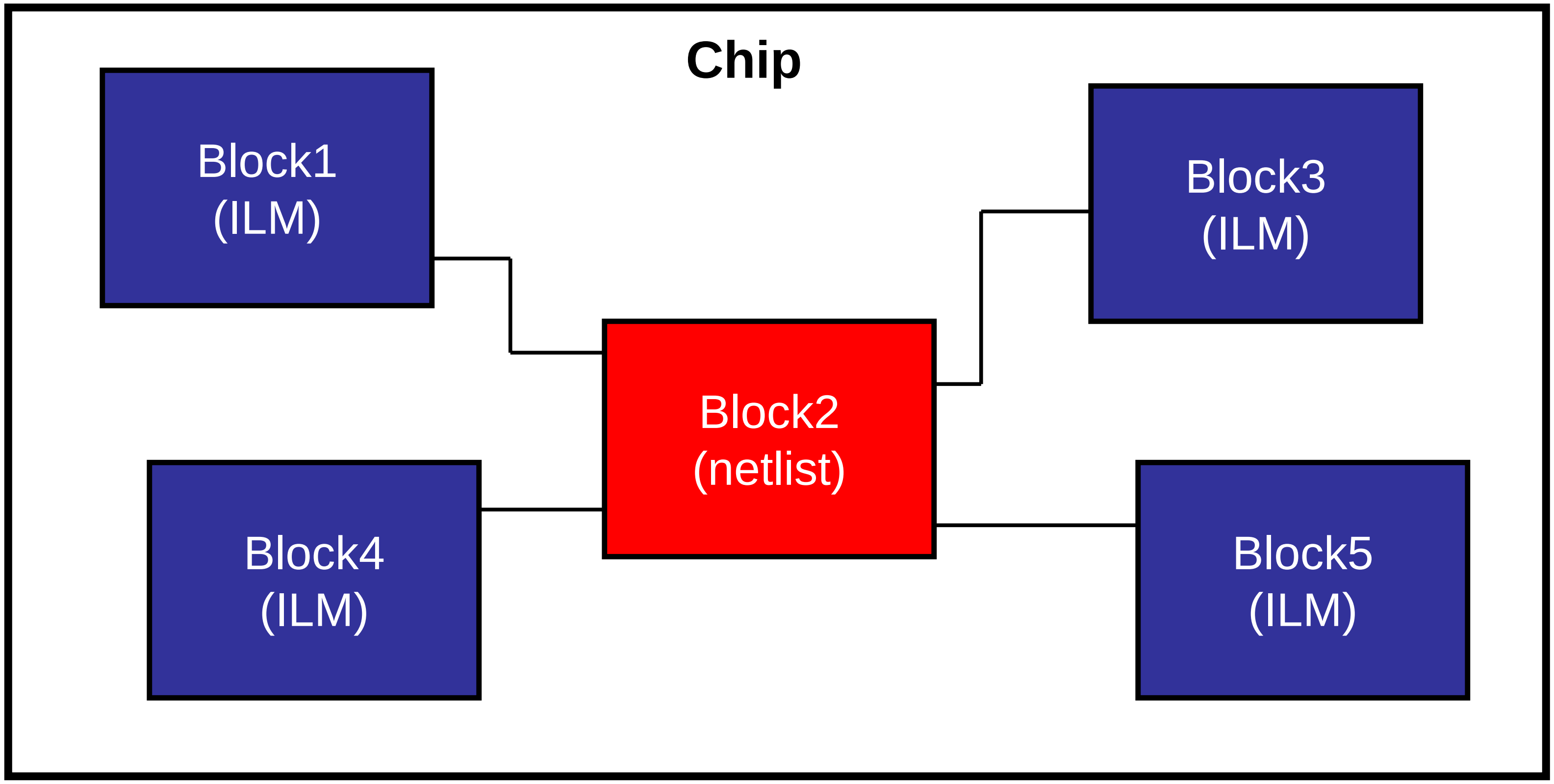
* Use PT ILMs if using distributed parasitics

Full-Chip Analysis / Top-level Optimization with ILMs



- ILM allow accurate analysis of paths between blocks
  - – User can efficiently find timing problems due to top-level routing, snake paths, etc.

Block Analysis / Budgeting with ILM



- Efficiently analyze block level netlist timing in context of entire chip by representing surrounding blocks as ILM

Outline

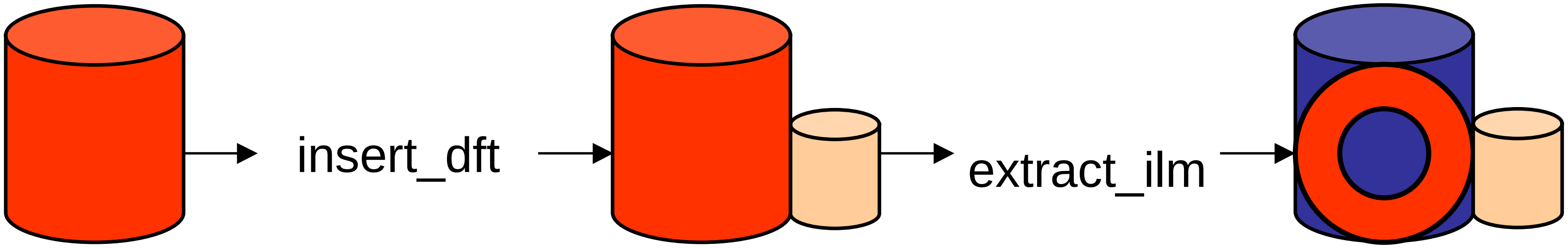
- ILM Goals
- Modeling Concepts
- ILM Flow
- **ILM and Test**
- Modeling Results
- Roadmap
- Summary

Using ILM in DFT Compiler

- DFT Compiler is using a similar approach to address capacity and performance issues
- Test behavior of a design is abstracted into a “Test Model”
  - Test model is based on IEEE Proposed Standard Core Test Language (CTL)
  - Contains information such as: scan-in, scan-out, scan enable, async set/reset, scan clock, chain count, chain length, scan shift timing, etc
- At top level, DFT Compiler uses information contained in Test Models to perform Test Design Rule Checking and scan insertion / assembly
- CTL is stored as an attribute attached to the design; use of test models is transparent to users

ILM containing Test model

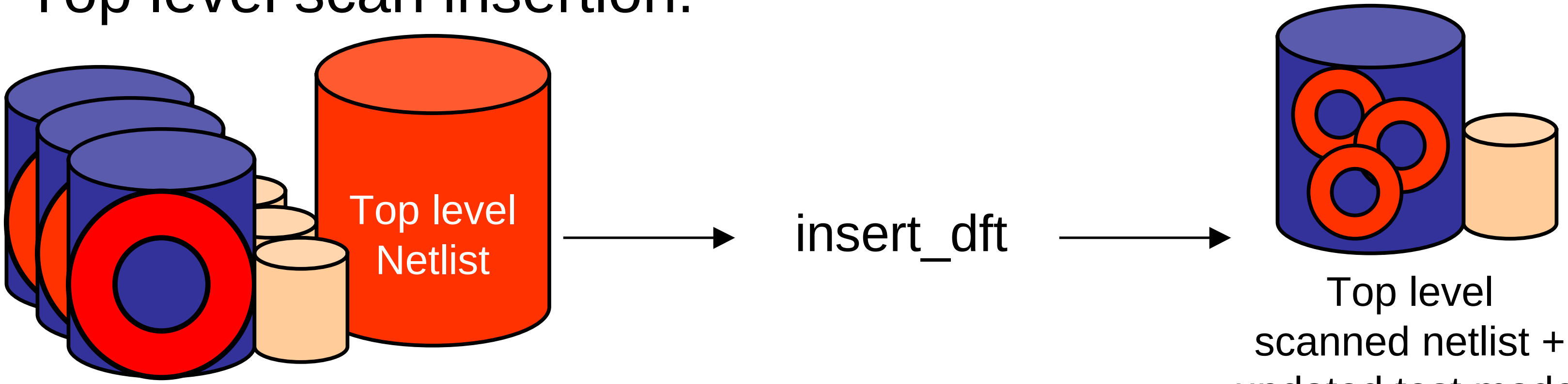
Block level scan insertion:



insert_dft

extract_ilm

Top level scan insertion:



Top level Netlist

insert_dft

Top level scanned netlist + updated test model

Sample script

Block level scan insertion

```
set test_use_test_models true
rtldrc
...
set_scan_configuration …
preview_dft -physical
insert_dft -physical
check_dft
list_test_models
write -f verilog
write -f db
…
extract_ilm
```

Top level scan insertion

```
set test_use_test_models true
read_db <mdb>
list_test_models
...
set_scan_configuration …
preview_dft -physical
insert_dft -physical
check_dft
list_test_models
…
write -f verilog
write -f db
…
```

Outline

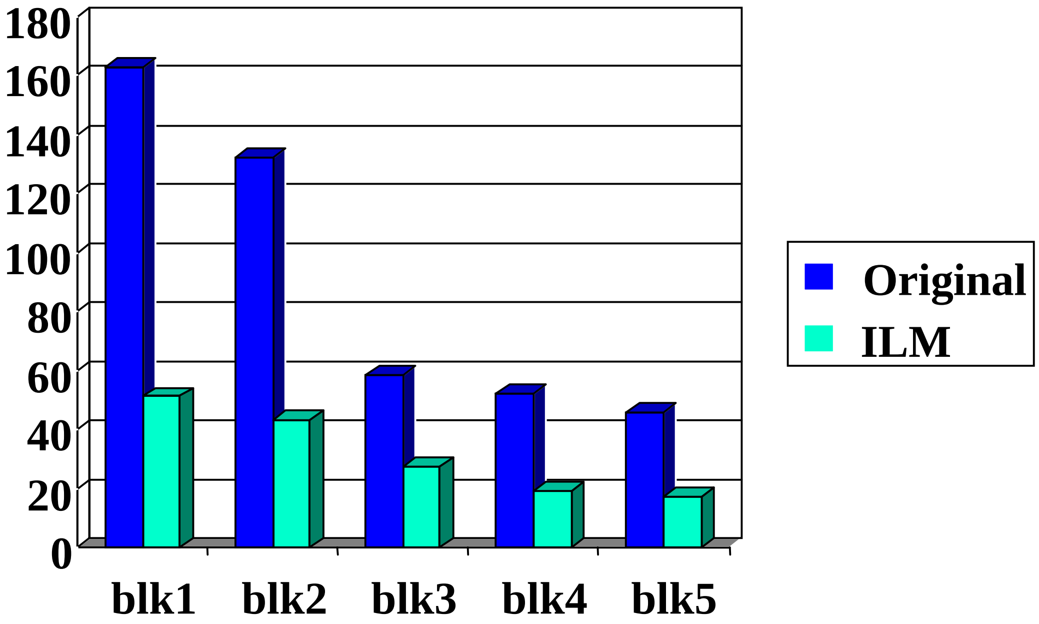
- ILM Goals
- Modeling Concepts
- ILM Flow
- ILM and Test
- **Modeling Results**
- Roadmap
- Summary

Results (memory)

Design ‘A’

Full chip memory reduction: 63% (2.7X)
Original: 2659 MB; With ILM: 994 MB

Mem (MB)

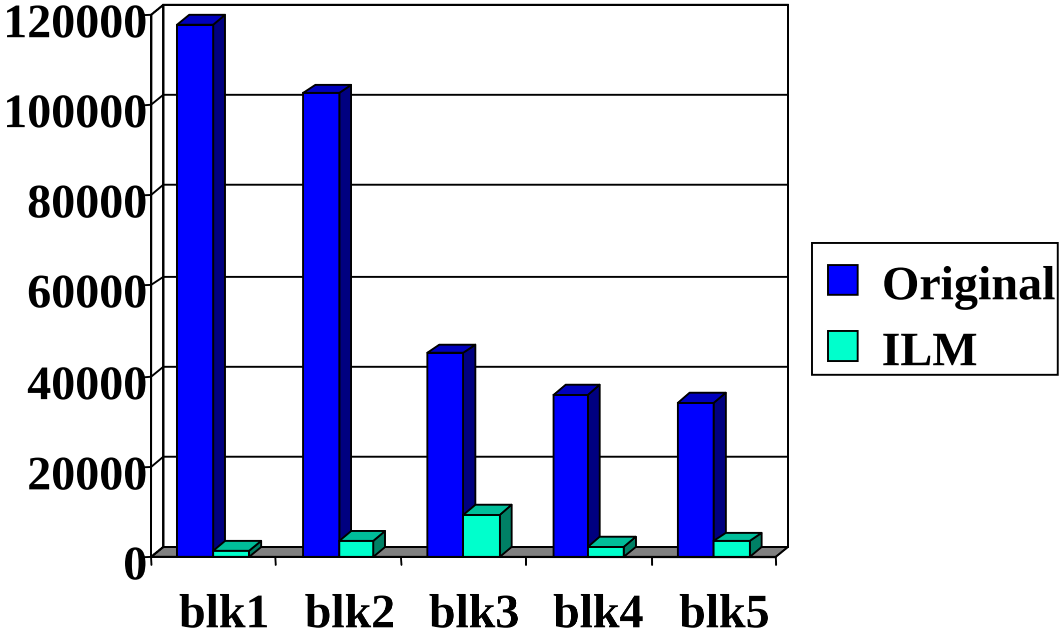


Results (cell count)

Design ‘A’

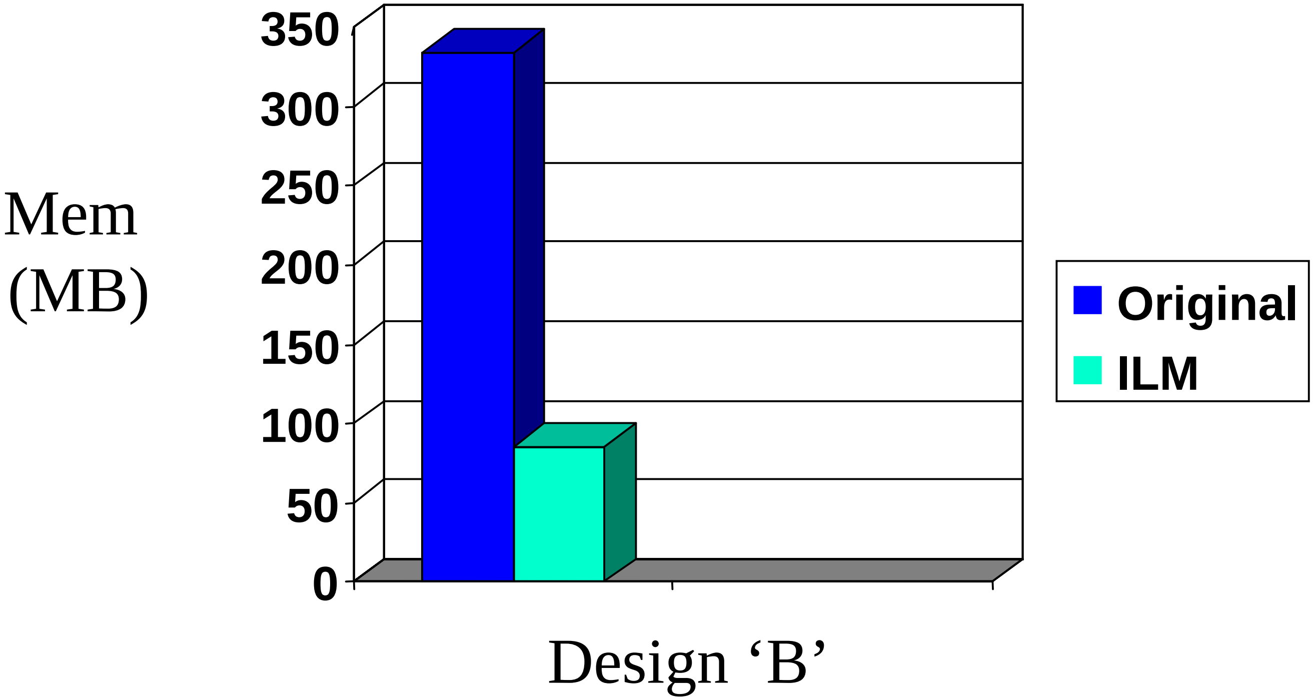
Full chip reduction: 81%
Original: 510,009 ILM: 96,939

Cell count



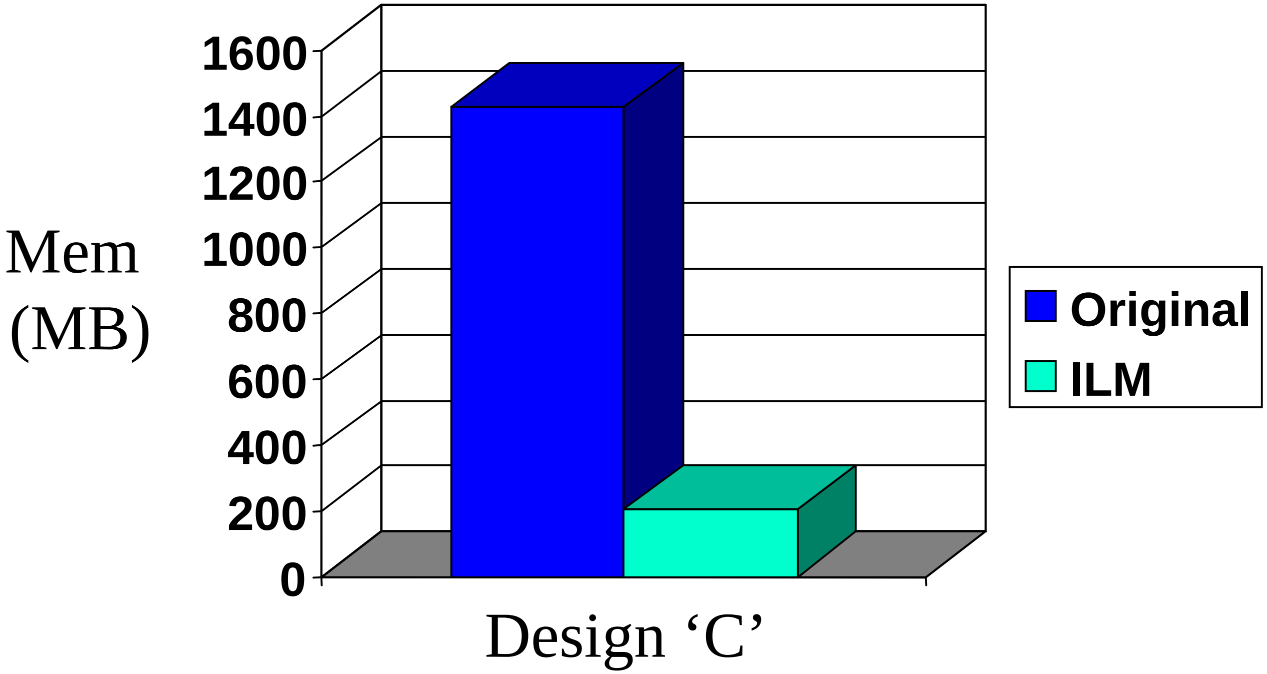
Results

memory reduction: 74% (3.9X)
original: 334 MB; ILM: 85 MB



Results

memory reduction: 86% (7X)
original: 1427 MB; ILM: 203 MB



Results (cell count)

Design ‘D’

	netlist	ilm	% reduction
blka	268918	8708	96.76
blkb	233655	3958	98.30
blkc	235448	31536	86.60
blkd	153656	7986	94.80
blke	200921	9757	95.14

Outline

- ILM Goals
- Modeling Concepts
- ILM Flow
- ILM and Test
- Modeling Results
- **Roadmap**
- Summary

Roadmap

2001.08 release

- Provided ILM extraction capability as Beta feature
- Enhanced PC to use ILM

2001.08-PSYN-JET release

- Support ILM extraction and usage in PC as a production feature
- Provide commands to propagate ILM data to top-level

Future Enhancements

- Top-level scan insertion using ILMs
- Top-level clock tree synthesis using ILMs
- Add commands to derive / remove obstructions
- Enable optimization within ILM (resizing / buffering)

Outline

- ILM Goals
- Modeling Concepts
- ILM Flow
- ILM and Test
- Modeling Results
- Roadmap
- **Summary**

Summary

Advantages of ILM

- High accuracy, easy to use/debug
- Model is context independent
- Contains physical info such as port and cell locations
- Preserves logical hierarchy and constraints
- Can be used by Physical Compiler, Design Compiler, Chip Architect, DFT Compiler, *PrimeTime (*use PT ILMs with distributed parasitics)
- Can be written out in DB format