Agenda

DAY 2

2	Design Planning (Lab – continued)
3	Placement
4	Clock Tree Synthesis

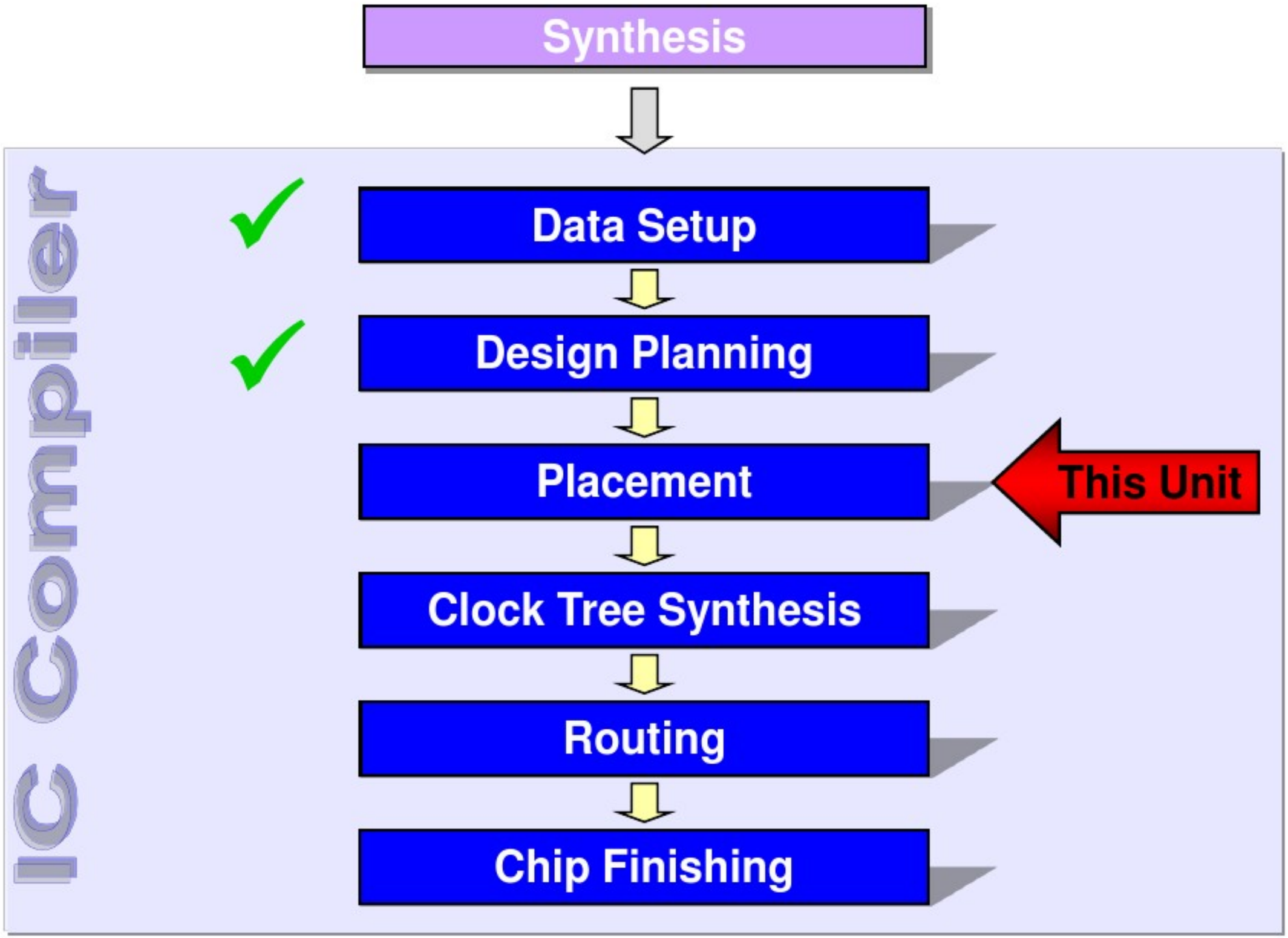
© 2010 Synopsys, Inc. All Rights Reserved

Unit Objectives

After completing this unit, you should be able to:

- Apply placement, DFT and power optimization settings before placement
- Perform placement and optimization
- Analyze congestion maps and reports
- Perform incremental congestion and timing optimization
- Perform additional placement techniques which allow more user-control

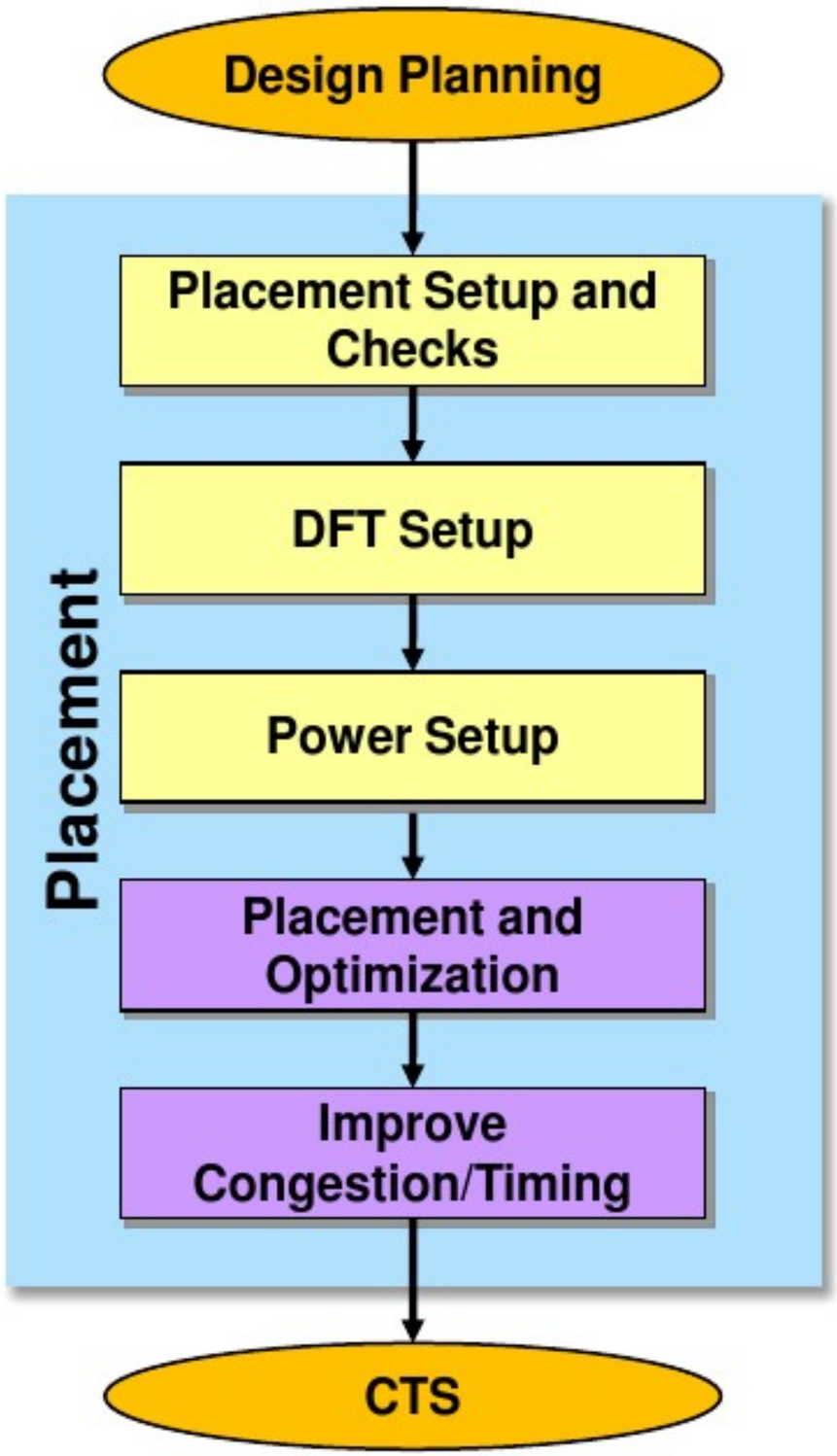
General IC Compiler Flow



Design Status Prior to Placement

- ***Design Planning* is completed**
  - If using 3rd party floorplanning tool "Floorplan Exploration" is done (Refer to "Design Planning" Unit 2 Appendix)
- **Second-Pass *Synthesis* is completed**
  - Based on actual floorplan
- **Second-Pass *Data Setup* is completed**
  - New design cell is generated based on 2nd pass netlist
- **"Floorplanned cell" is generated – ready for placement**
  - Core and periphery areas defined
  - Macros are placed and "fixed"
  - Placement blockages defined
  - Power grid pre-routed
  - Standard cell placement is discarded

IC Compiler Placement Flow

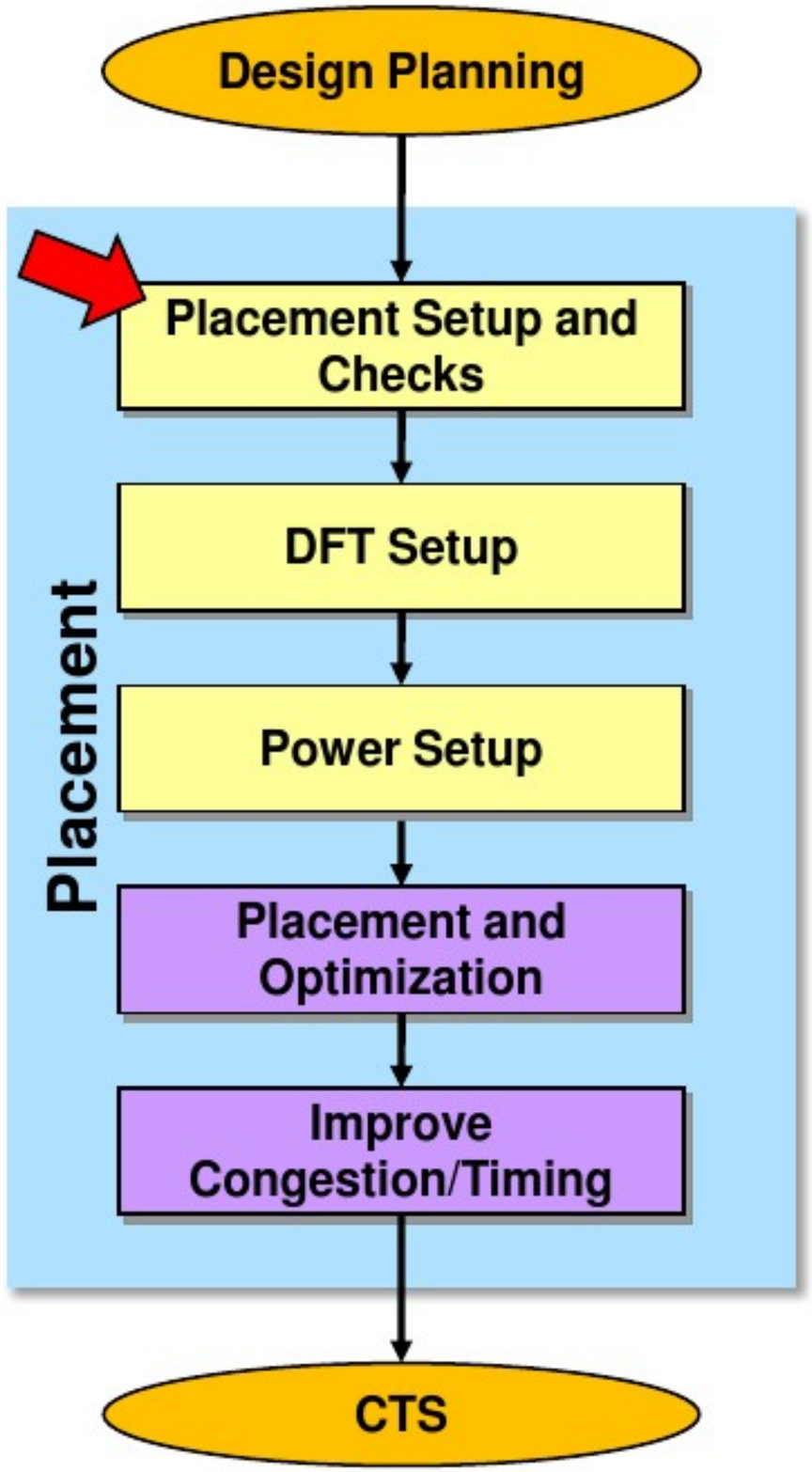


The "placement phase" involves several key steps:

- Setup steps to control placement, DFT and power
- Placement and optimization
- Incremental refinement to improve congestion and/or setup timing

Note: The flow diagrams included in this unit represent an **example** flow, not the recommended flow

Placement Setup and Checks



"Fix" all Macro Cell Placements

- **In most situations macro cell placement is determined during design planning and their placement is "fixed"**
- **It is a good practice to fix all macro placements again, just in case….**
  - You may have manually moved a macro after design planning and have forgotten to fix its placement

```
open_mw_cel DESIGN_floorplanned
set_tlu_plus_files \
      -max_tluplus ./libs/abc_max.tlup \
      -min_tluplus ./libs/abc_min.tlup \
      -tech2itf_map  ./libs/abc.map
source tim_opt_ctrl.tcl
set_dont_touch_placement [all_macro_cells]
```

Re-apply TLUplus

Re-apply timing and optimization controls

Verify *Layer* and *Placement* Constraints

The following checks are recommended because their settings are not saved in the *DEF* floorplan file – they must be re-applied after creation of the 2nd pass design library and loading of the floorplan

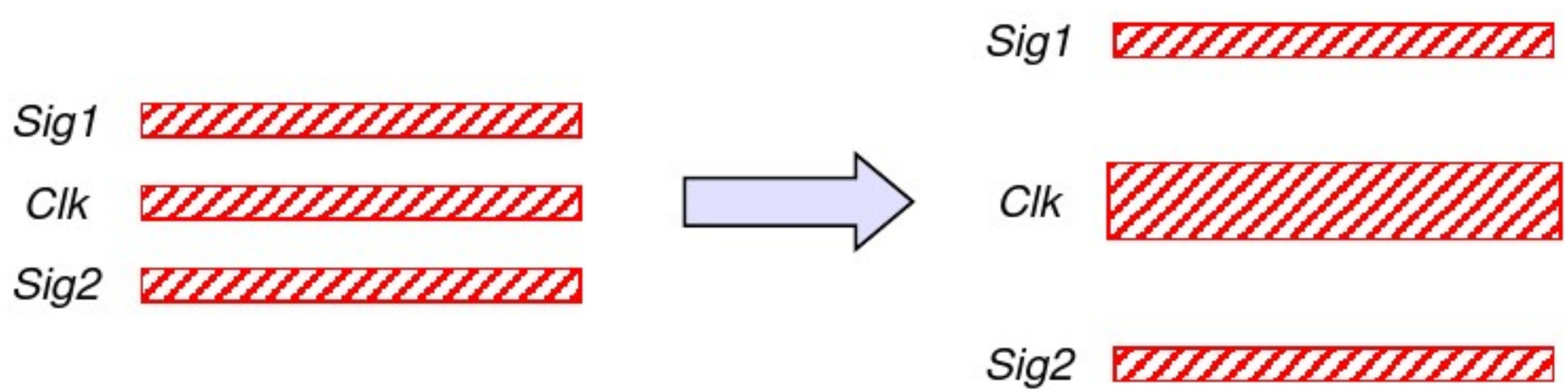
- **Ensure that the following settings have been re-applied, as appropriate:**
  - Ignored routing layers
  - Placement blockages under P/G straps - *pnet* options
  - Global placement *keepout* variable settings

```
report_ignored_layers
report_pnet_options
printvar physopt_hard_keepout_distance
printvar placer_soft_keepout_channel_width
```

- **Apply or correct as needed – see notes below**

Non-Default Clock Routing

- IC Compiler can route clock nets using non-default routing (NDR) rules, e.g. double-spacing, double-width, shielding
- Non-default rules are often used to "harden" the clock, e.g. to make the clock routes less sensitive to *cross-talk* or *electro-migration (EM)* effects
- NDR nets use more routing resources (tracks) which can increase congestion
- This congestion impact can be taken into account during global-route driven placement



Default Routing Rule

Effect of NDR route on Clk

Specify Non-Default Routing Rules

- **Define the NDR rules:**

Can also be used to define shielding and tapering rules – see notes below

```
define_routing_rule MY_ROUTE_RULES  \
  -widths {METAL3 0.4 METAL4 0.4 METAL5 0.8} \
  -spacings {METAL3 0.42 METAL4 0.63 METAL5 0.82}
```

- **Configure the clock tree routing:**

```
set_clock_tree_options -clock_trees [all_clocks] \
                       -routing_rule MY_ROUTE_RULES \
                       -layer_list "METAL3 METAL5"
```

You may also specify the min/max layers to be used for clock tree routing

Check Placement Readiness

`check_physical_design -stage pre_place_opt` **checks the readiness for placement of:**

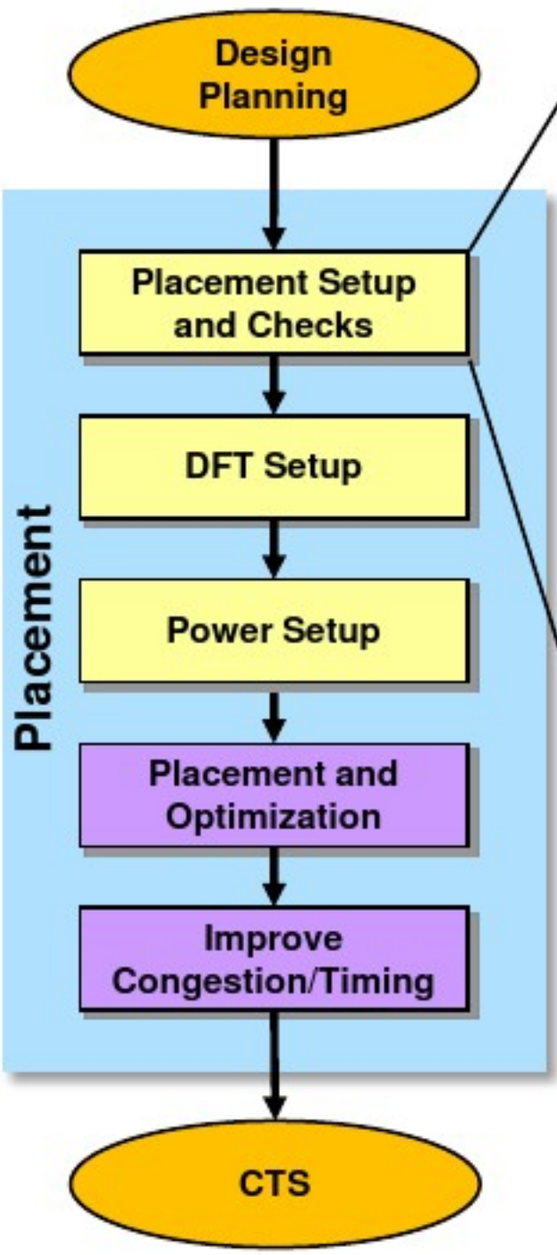
- Floorplan
- Netlist
- Design constraints

`check_physical_constraints` **reports:**

- Cells placed in “hard placement blockage” areas
- Metal layer inconsistencies against the library
- R/Cs for routing layers
- Narrow placement regions (“chimneys”)
- Legal sites for cell placement
- Large RC variations between metal layers

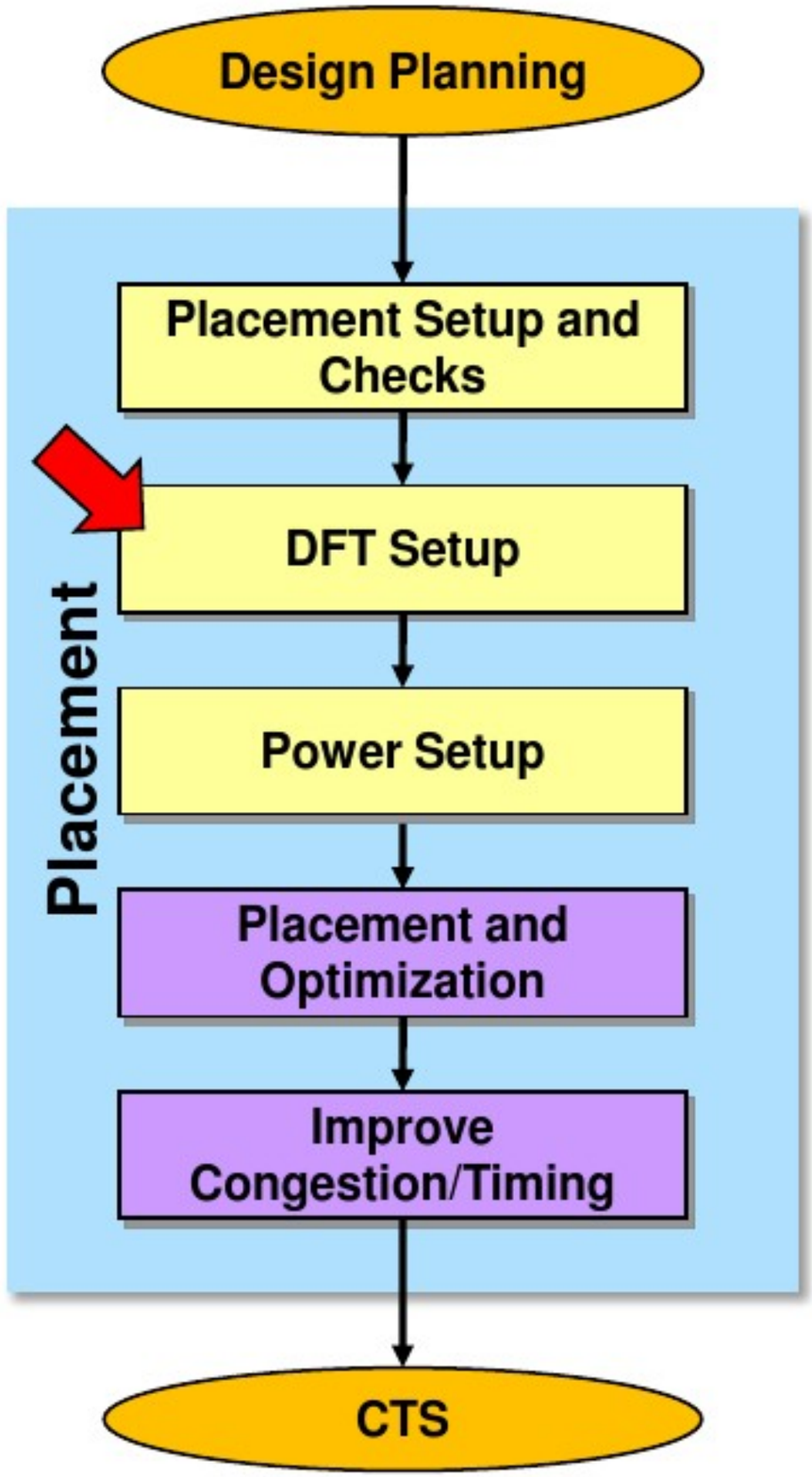
Modify the floorplan, constraints or libraries as needed

Summary: Placement Setup and Checks



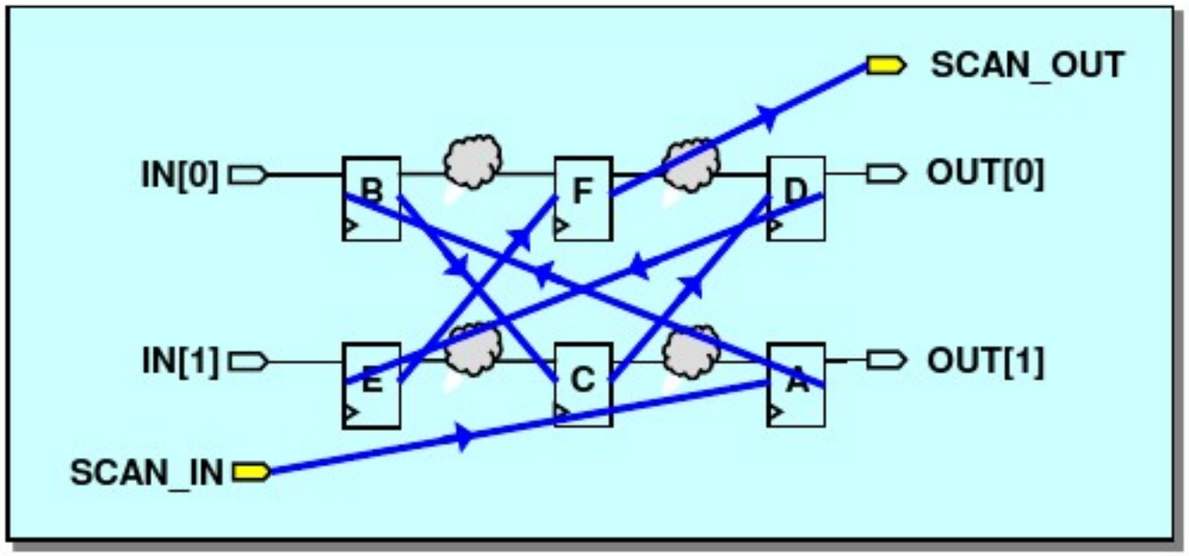

```
open_mw_cel DESIGN_floorplanned
set_tlu_plus_files \
        -max_tluplus ./libs/abc_max.tlup \
        -min_tluplus ./libs/abc_min.tlup \
        -tech2itf_map  ./libs/abc.map
source tim_opt_ctrl.tcl
set_dont_touch_placement [all_macro_cells]

report_ignored_layers
report_pnet_options
printvar physopt_hard_keepout_distance
printvar placer_soft_keepout_channel_width

# Define non-default routing rules, if applicable
define_routing_rule MY_ROUTE_RULES  \
  -widths {METAL3 0.4 METAL4 0.6 METAL5 0.6} \
  -spacings {METAL3 0.5 METAL4 0.65 METAL5 0.65}
set_clock_tree_options -clock_trees [all_clocks] \
                 -routing_rule MY_ROUTE_RULES \
                 -layer_list "METAL3 METAL5"

check_physical_design -stage pre_place_opt
check_physical_constraints
```

Design-for-Test (DFT) Setup



Skip this if your design does not include scan chains.

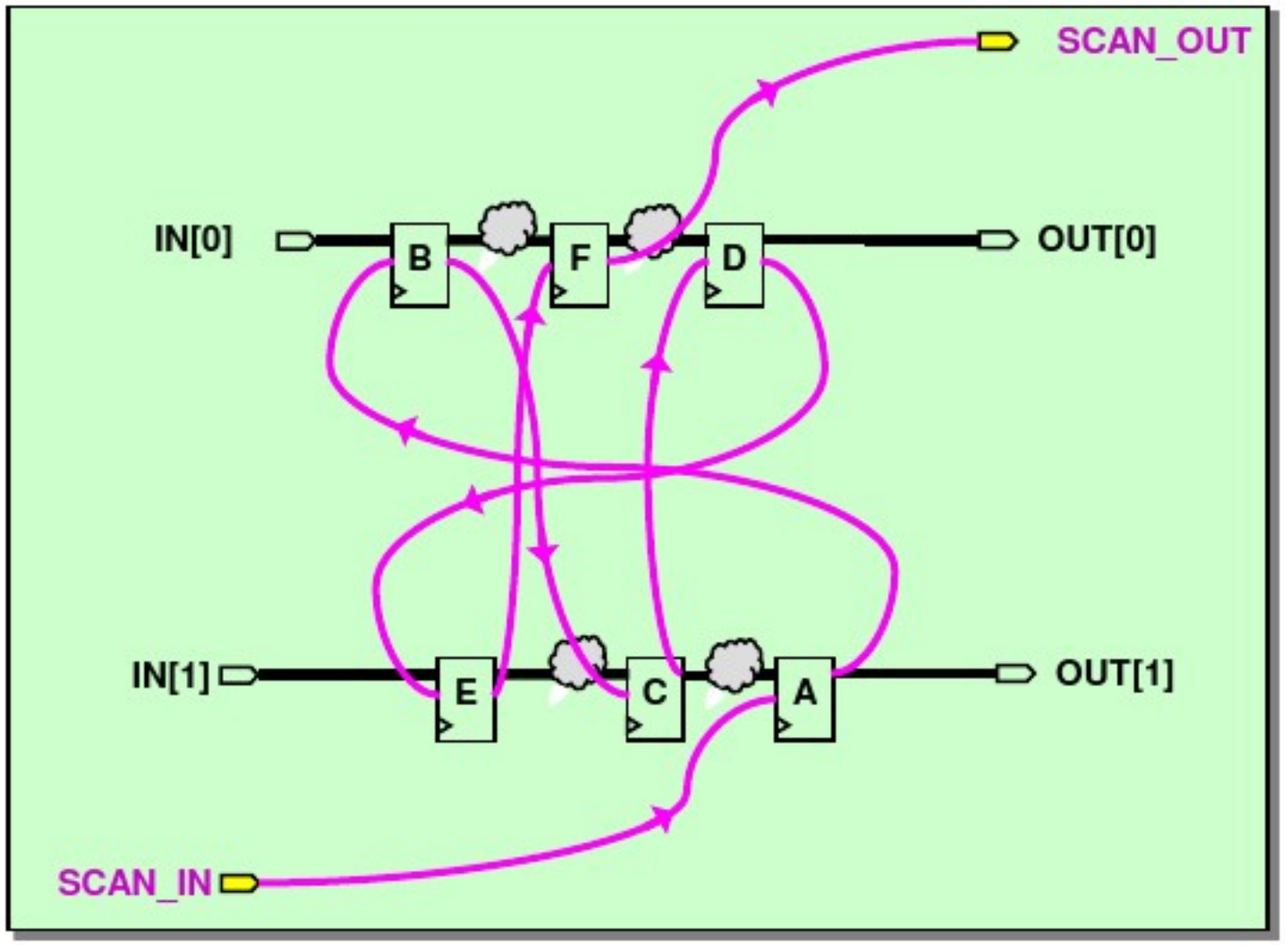
Pre-Existing Scan Chains



- **If your design flow includes "design-for-test", the netlist will contain "scan chains":**
  - Groups of "scan registers" that are serially connected through SI/SO pins, and inserted during synthesis
- **Scan chain paths are active only during "test mode", not during "functional mode"**
- **Registers are typically connected in alphanumeric order during synthesis – irrelevant for DFT, but not optimal for routing**

The Issue with Existing Scan Chains

What happens if placement is done with the pre-existing order of the scan chains?



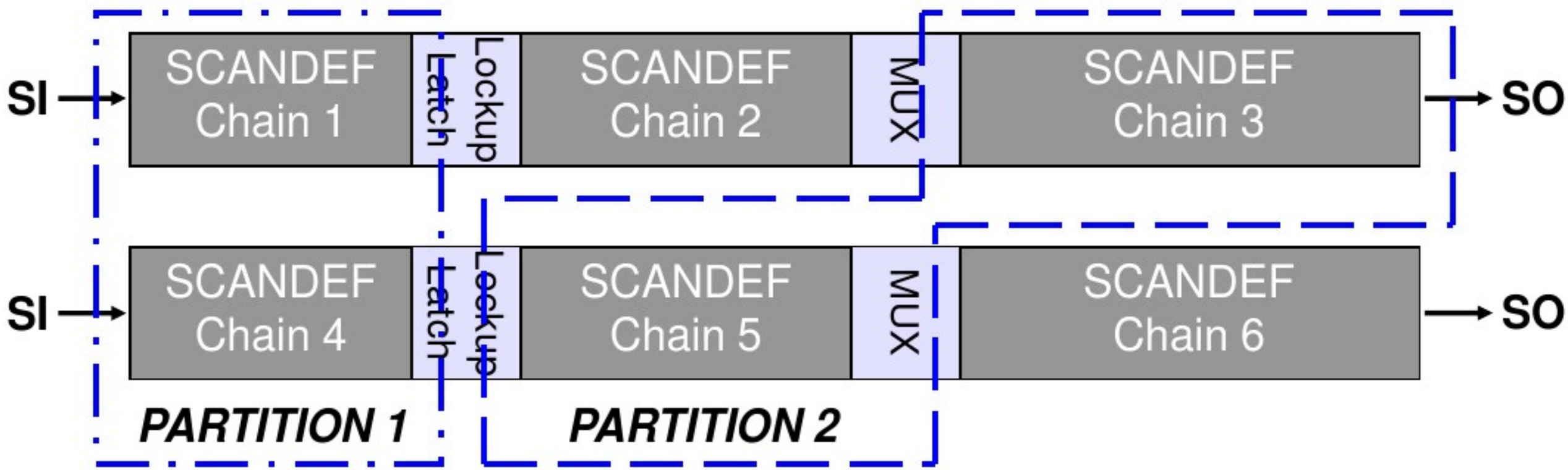
- **Serially connected scan registers may be placed far apart which may require a lot more routing resources**

SCANDEF-Based Chain Reordering

- **IC Compiler can perform placement-aware reordering of scan cells**
- **The scan chain information (*SCANDEF*) from synthesis can be transferred to IC Compiler in two ways:**
  - By loading the netlist in *ddc* format (imbedded *SCANDEF*)
  - By loading a *SCANDEF* file (generated after scan insertion)
- **Reordering occurs within each scan chain**
- **Lockup latches or multiplexers break up scan chains further into "reordering buckets"**
- **If present, reordering happens within the buckets**

Re-ordering Chains within Same "Partition"

- To extend flexibility, SCANDEF also supports reordering within partitions, across multiple buckets
- A PARTITION is a group of "SCANDEF chains" that may exchange flip-flops during reordering



Example SCANDEF from Synthesis with DFT

```
DESIGN my_design ;
SCANCHAINS 2 ;
- 1
+ START PIN test_si1
+ FLOATING A ( IN SI ) ( OUT Q )
                B ( IN SI ) ( OUT Q )
                C ( IN SI ) ( OUT Q )
                D ( IN SI ) ( OUT Q )
+ PARTITION CLK_45_45
+ STOP PIN test_so1

- 2
+ START PIN test_si2
+ FLOATING E ( IN SI ) ( OUT Q )
                F ( IN SI ) ( OUT Q )
                G ( IN SI ) ( OUT Q )
                H ( IN SI ) ( OUT Q )
+ PARTITION CLK_45_45
+ STOP PIN test_so2
```

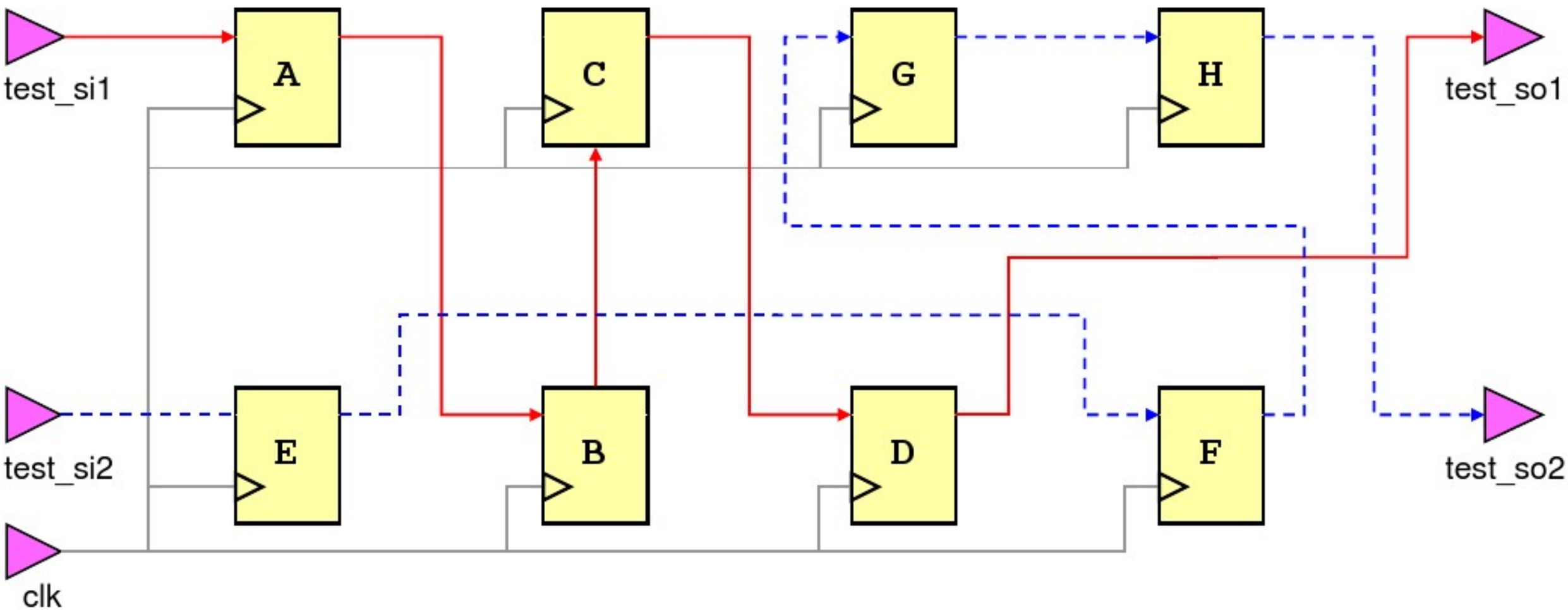
Design name

Number of chain stubs in the design

"**FLOATING**" indicates that these flipflops can be reordered

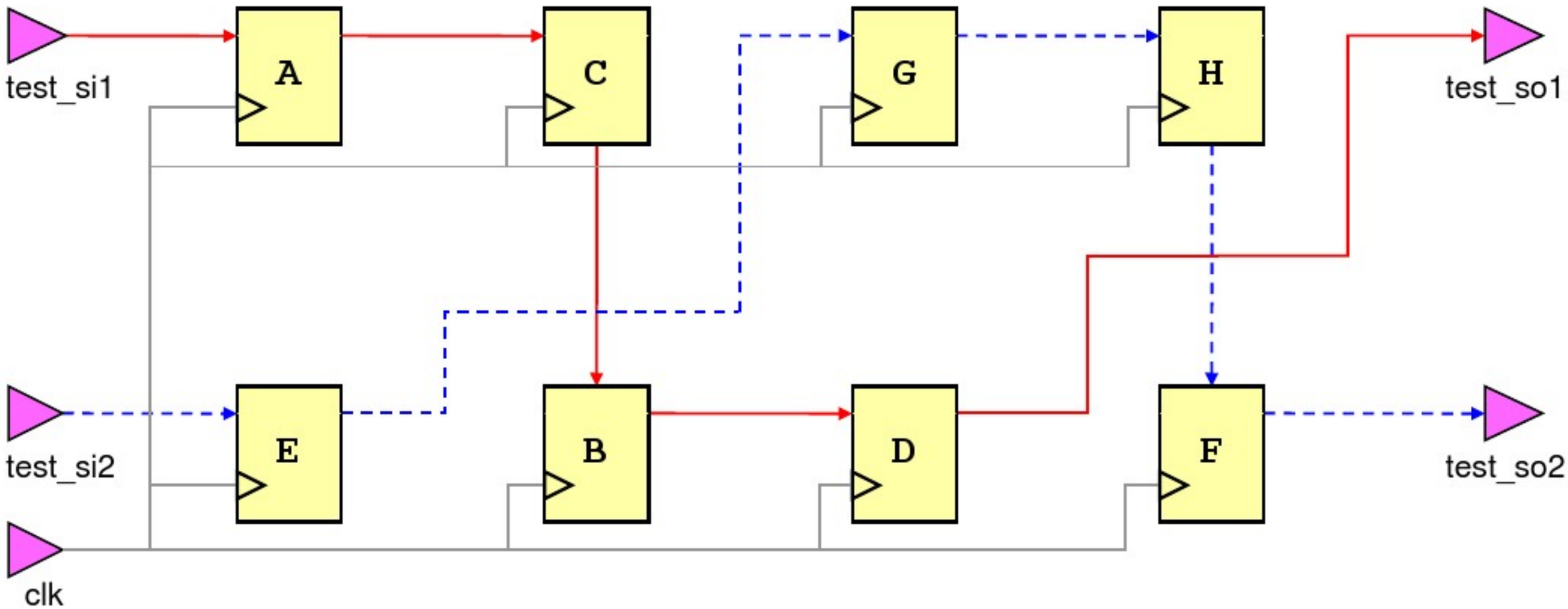
PARTITION keyword in SCANDEF. Flipflops can be swapped between two partitions with the same name

Example: Placement with Existing Ordering



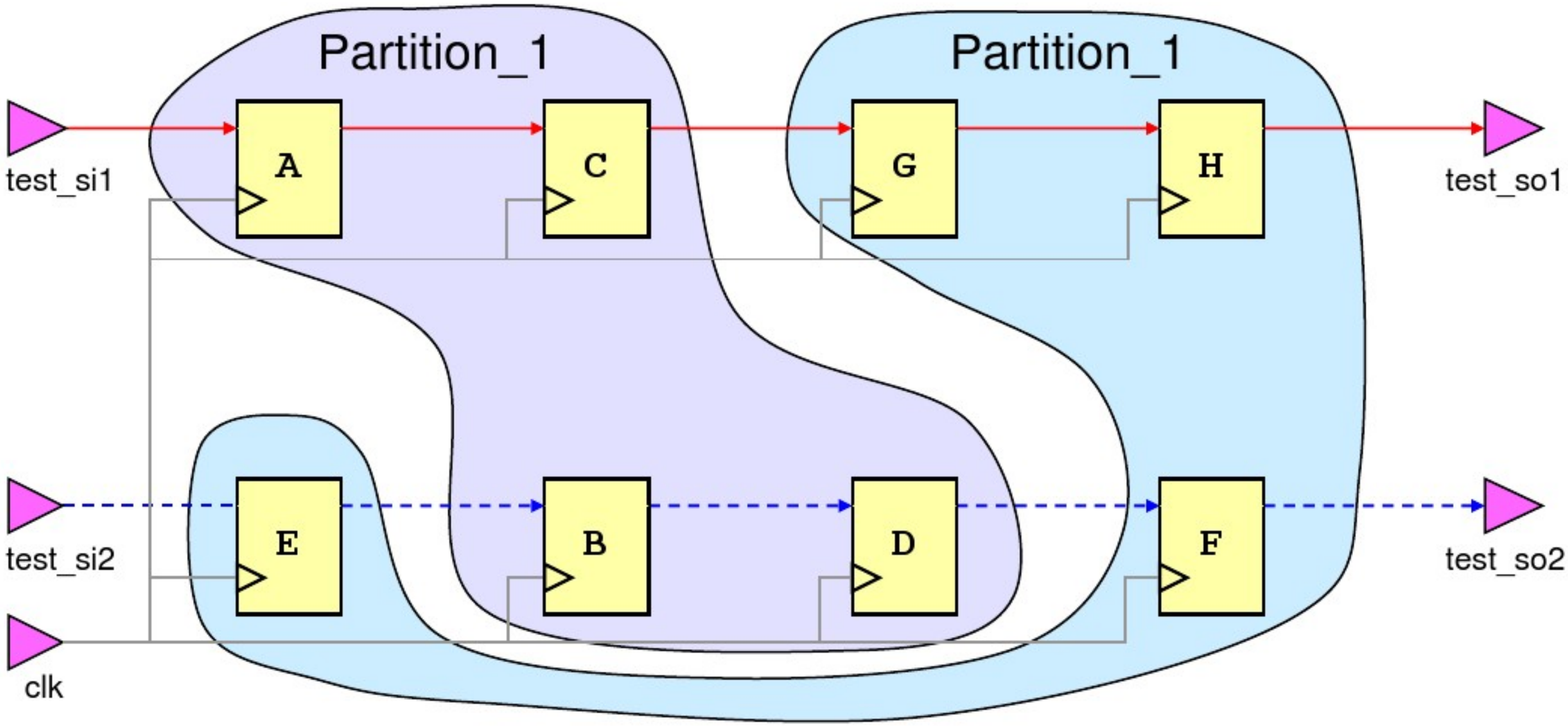
chain-order	**DC Netlist**		
Scan Chain 1	ABCD		
Scan Chain 2	EFGH		

Example: Reordering Within Scan-Chains



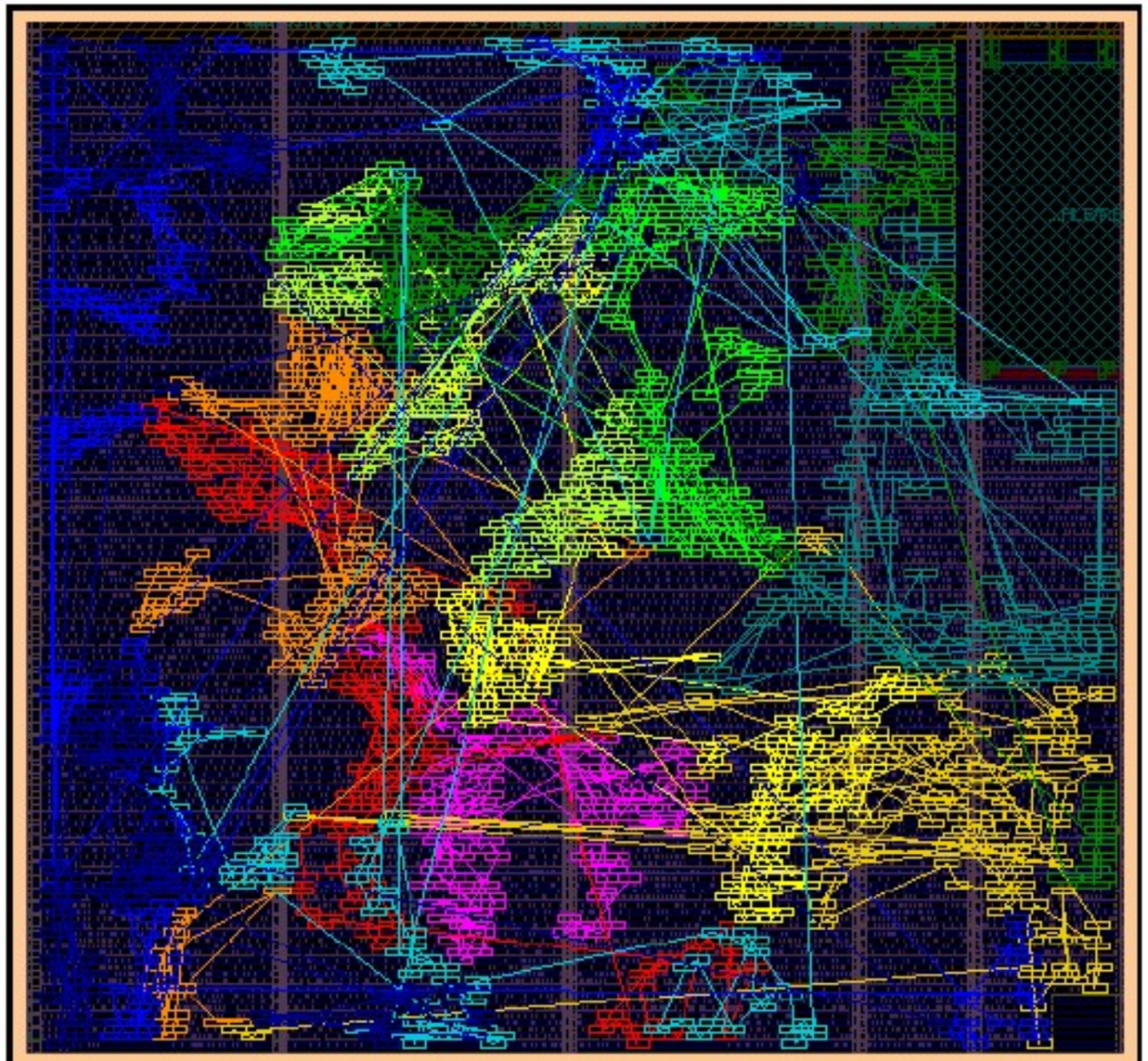
chain-order	**DC Netlist**	**SCANDEF w/o PARTITION**	
Scan Chain 1	ABCD	ACBD	
Scan Chain 2	EFGH	EGHF	

Example: Reordering Within Partitions

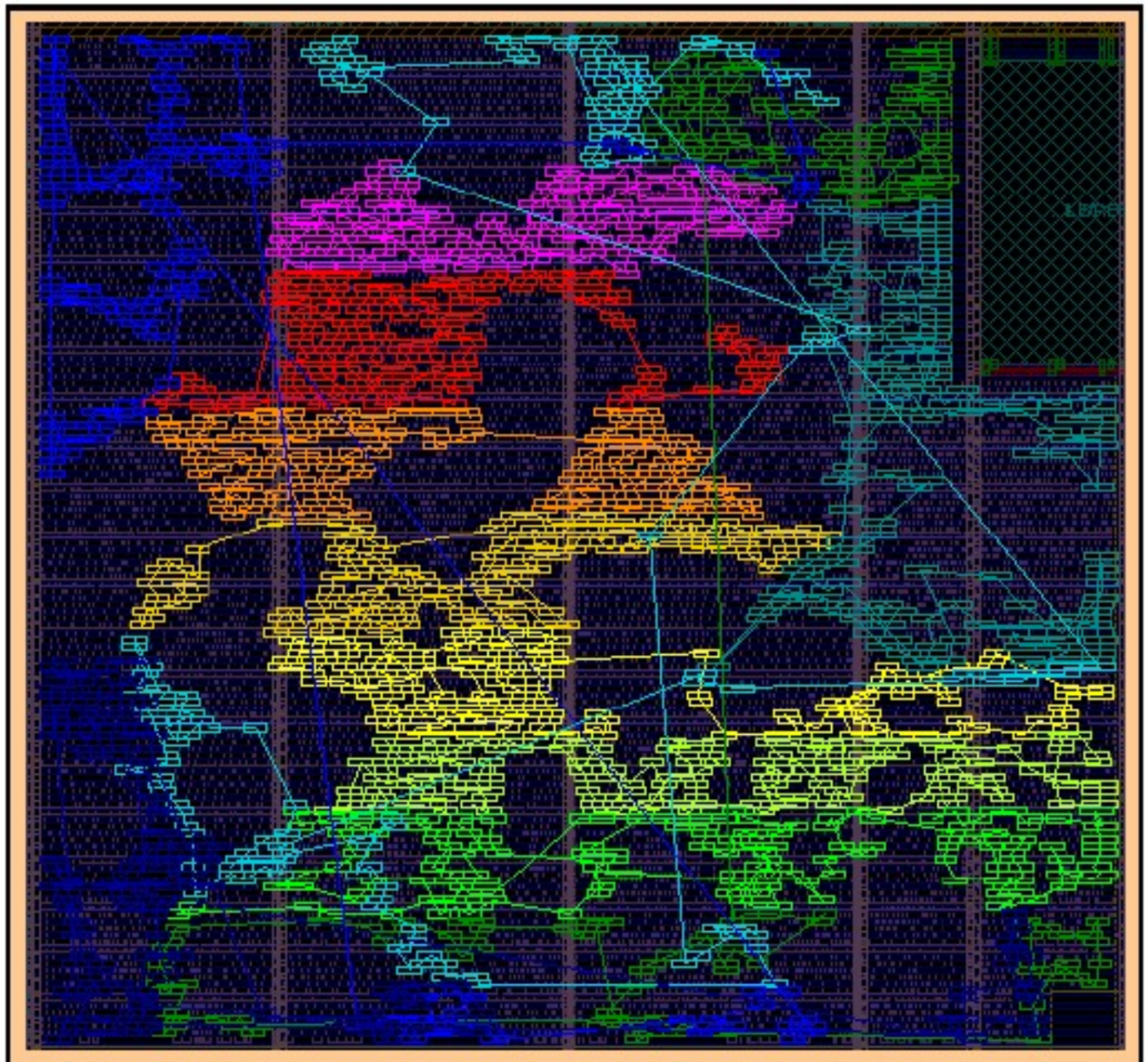


chain-order	**DC Netlist**	**SCANDEF w/o PARTITION**	**SCANDEF with PARTITION**
Scan Chain 1	ABCD	ACBD	ACGH
Scan Chain 2	EFGH	EGHF	EBDF

Placement-Based Scan Chain Re-Ordering



Pre-existing Scan Ordering



Placement-Based Re-Ordering

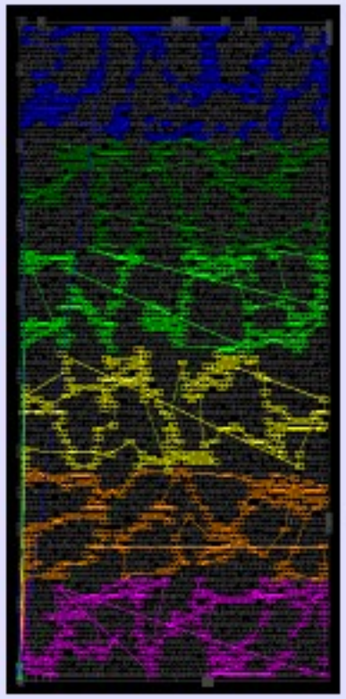
```
place_opt
```

Not needed if imported design was in *ddc* format – already included!

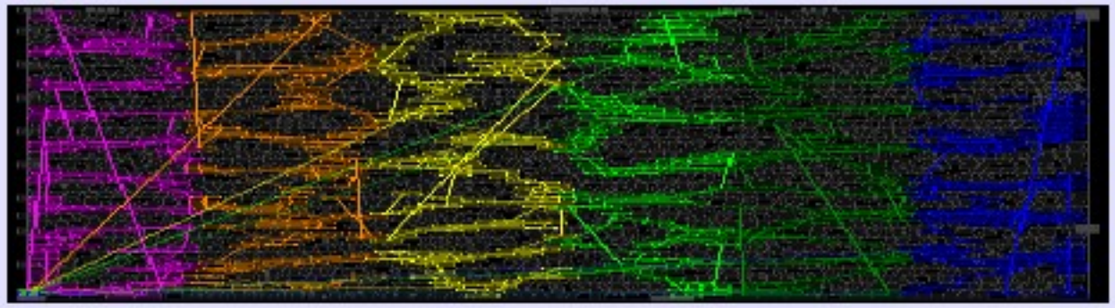
```
read_def DESIGN.scandef
report_scan_chain
place_opt -optimize_dft
```

“Adaptive” Partitioning ON by Default

Horizontal partitioning may be better when core H/W >> 1



Vertical partitioning may be better when core H/W << 1

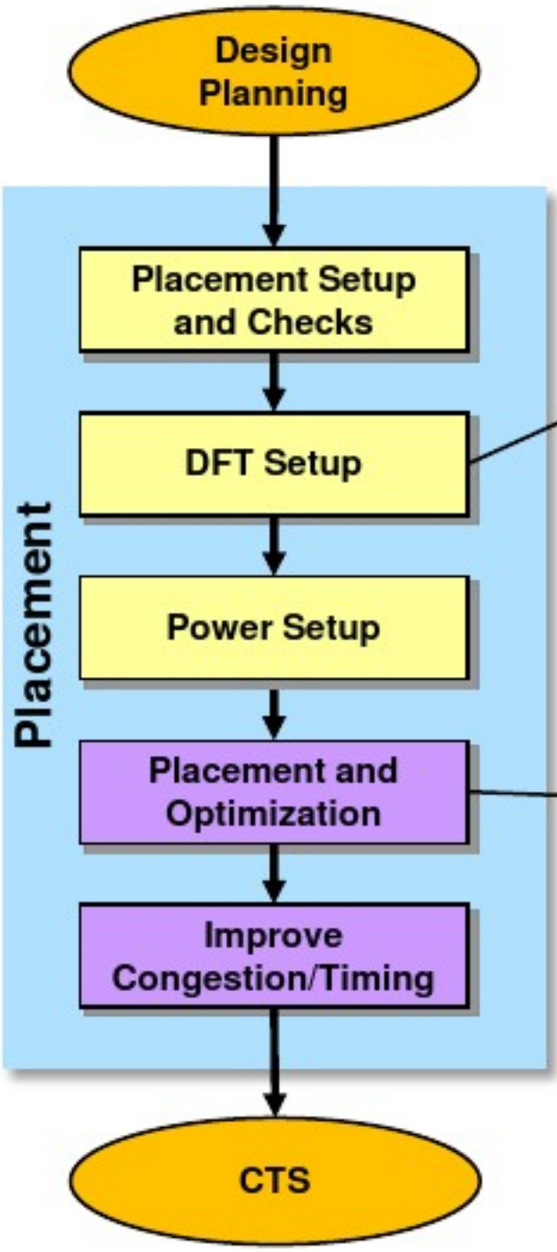


Reduces congestion in the “long” direction

Default behavior can be modified, if desired:

```
set_optimize_dft_options
 [-repartitioning_method none | single_directional | \
                         multi_directional | adaptive]
 [-single_dir_option horizontal | vertical]
```

Summary: DFT Setup



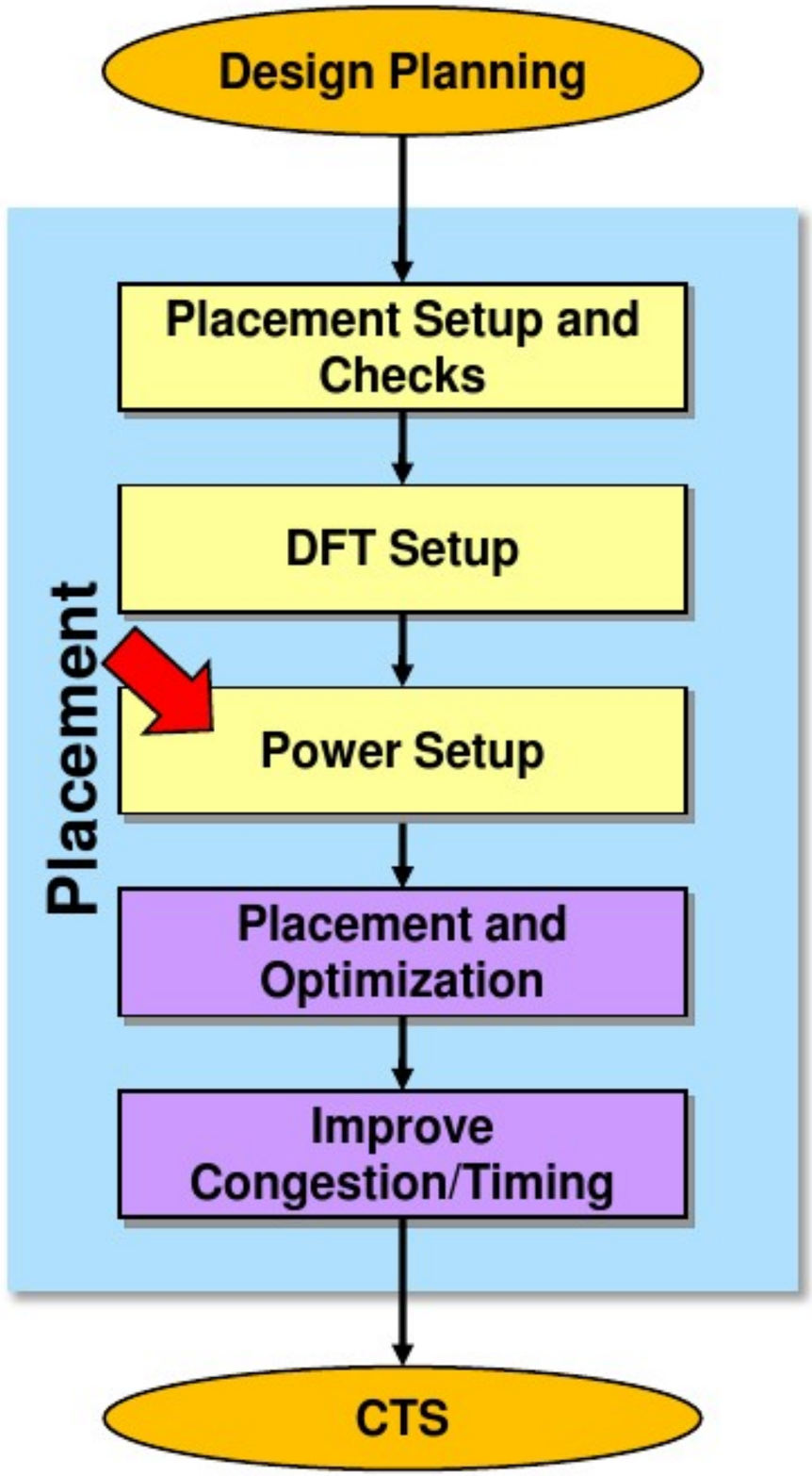
DFT Setup

```
# If the netlist loaded into ICC was NOT ddc:
read_def DESIGN.scandef

report_scan_chain

# If prefer to change the default "adaptive"
# repartitioning method:
set_optimize_dft_options -repartitioning_method ...
```

```
place_opt -optimize_dft
```

Skip these steps if your design does not include scan chains.

Power Setup



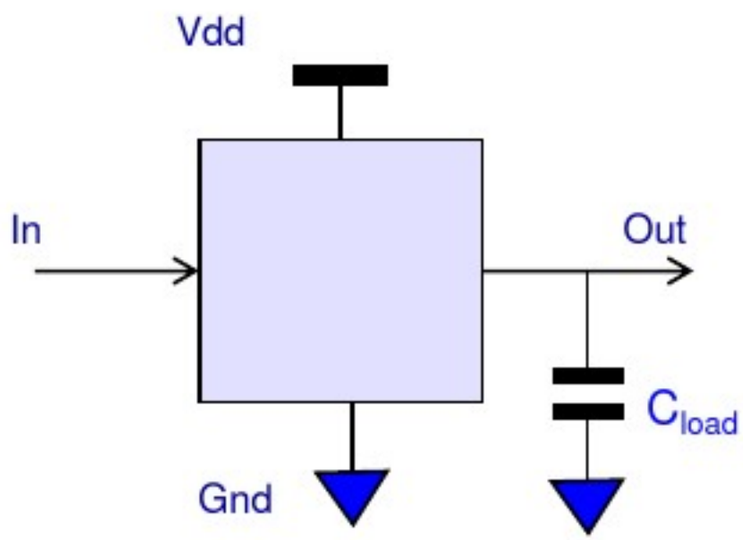
Skip this if power optimization is not a priority

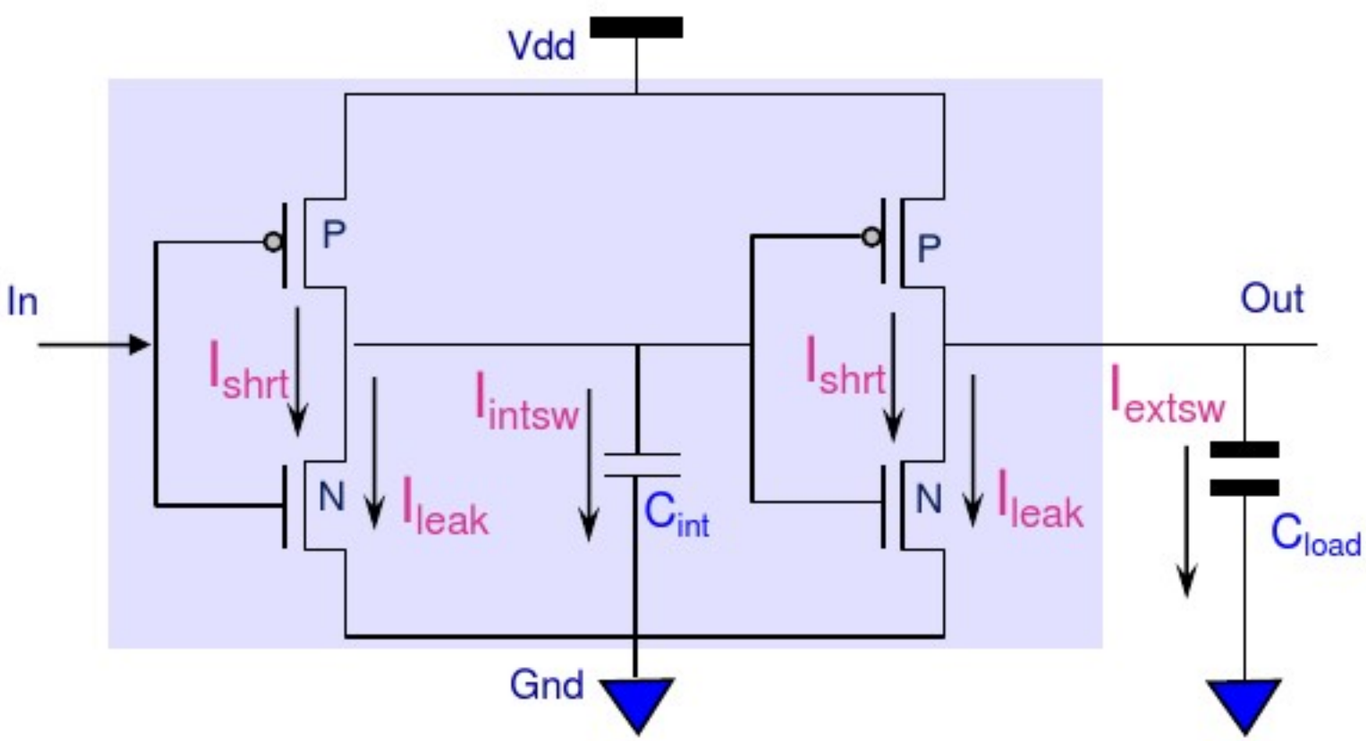
Power optimization will generally:

- Not impact critical path delays or congestion
- Increase run time

Where Does Power Dissipation Occur?

Total Power = Static Power +Dynamic Power





- **Static or leakage power dissipation comes from:**
  - Current leakage though devices that are “OFF” during static states
- **Dynamic or switching power dissipation comes from:**
  - Charging external loads
  - Charging internal loads
  - Short circuit or transient current while both devices are ON during a logic swing

Leakage Power Optimization

Low-V_{th} Cells
Fast, High Leakage

High-V_{th} Cells
Slow, Low Leakage

- **A multi-V_{th} library is the key to minimize leakage power**
- **Low V_{th} cells used on critical max-delay paths to help timing**

- **High V_{th} cells used on non-critical max paths to save leakage power**

```
set_app_var target_library  "sc_max_hvt.db sc_max_lvt.db"
set_app_var link_library "* $target_library io_max.db  \
                                              macros_max.db"
set_min_library sc_max_hvt.db -min_version sc_min_hvt.db
set_min_library sc_max_lvt.db -min_version sc_min_lvt.db
set_min_library io_max.db -min_version io_min.db
set_min_library macros_max.db -min_version macros_min.db
create_mw_lib  ... -mw_reference_library \
   "mw/sc_hvt  mw/sc_lvt  mw/io  mw/ram32"
...
set_power_options -leakage true|false
```

Report Multi-V_{th} Cells

After placement you can generate a report summarizing how many of each type of cells were used

Report

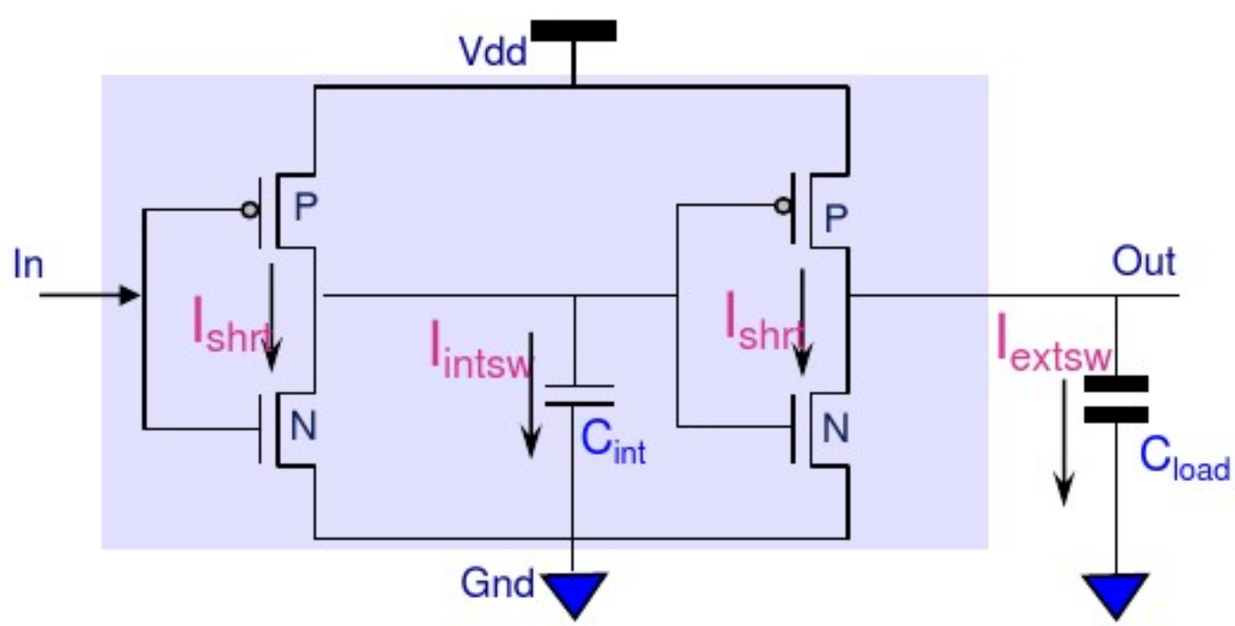
```
icc_shell> report_threshold_voltage_group
****************************************************
                    Threshold Voltage Group Report
Threshold Voltage Group      Number of Cells     Percentage
****************************************************
LVt                                 90                8.33%
HVt                                931               86.12%
SVt                                 59                5.46%
undefined                            1                0.09%
****************************************************
```

These are typically macro or IP cells, which are not defined as low or high V_{th}

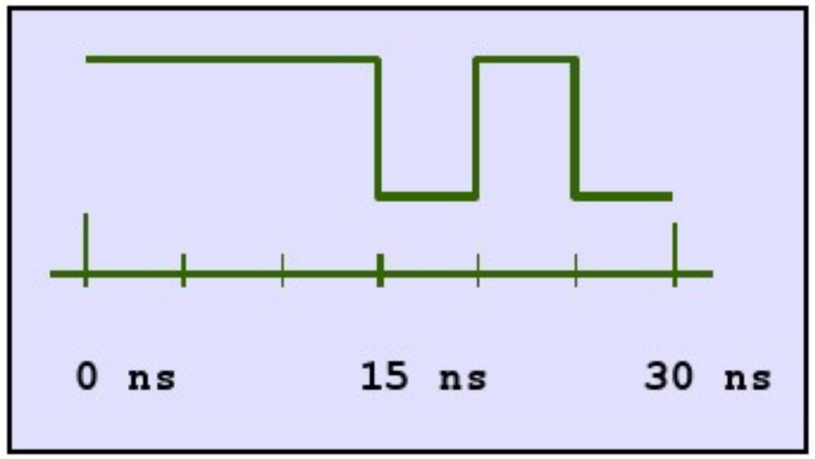
Note: This reporting requires special library attributes. See notes below.

Reducing Dynamic Power Dissipation

- **Dynamic power optimization reduces power by identifying high "toggle rate" nets and:**
  - Reducing wire capacitance
  - Downsizing gates
- **Two optimization techniques available during placement:**
  - Low power placement (LPP)
  - Gate-level power optimization (GLPO)



Switching Activity Terminology



- **Toggle Count (TC):**

  A toggle is a logic transition (0→1 or 1→0). Toggle count (`TC`) is the number of toggles.

- **Toggle Rate (Tr):**

  Number of toggles per unit time (usually the simulation duration)

  `Tr = TC / duration`

- **T1 , T0:**

  Total time a signal is at logic 1, 0

- **Static Probability (Sp):**

  Probability of a logic 1 for a signal

  `Sp = T1 / duration`

?

Using the graph on the left, fill in the following:

TC = , Tr = , T1= , and SP=

SAIF File Provides Switching Activity

DESIGN.saif

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(DATE "Mon May 17 02:33:48 2009")
(VENDOR "Synopsys, Inc")
(PROGRAM_NAME "VCS-Scirocco-MX Power
   Compiler")
(VERSION "1.0")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 10000.00)
(INSTANCE I_TOP
   (INSTANCE macinst
      (NET
       (z\[3\]
         (T0 6488) (T1 3493) (TX 18)
         (TC 26) (IG 0)
       )        ... ...
```

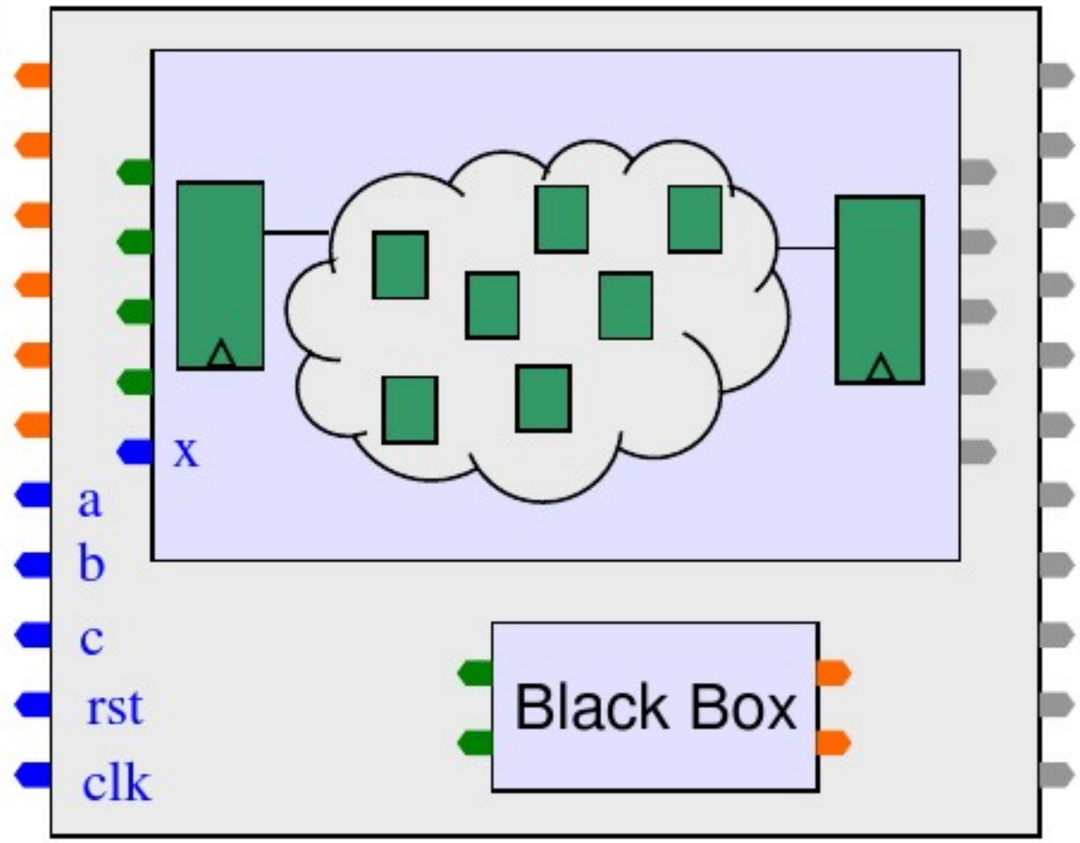
```
    (z\[32\]
       (T0 6488) (T1 3493) (TX 18)
       (TC 26) (IG 0)
    )
   ... ...
)
(INSTANCE U3
 (PORT
  (Y
   (T0 4989) (T1 5005) (TX 6)
   (COND ((D1*!D0) | (!D1*D0))
   (RISE)
       (IOPATH S (TC 22) (IG 0)
       )
   COND ((D1*!D0) | (!D1*D0))
   (FALL)
       (IOPATH S (TC 21) (IG 0)
       )
   COND_DEFAULT (TC 0) (IG 0)
   )     ... ...
```

```
read_saif -input DESIGN.saif -instance_name I_TOP
report_saif
```

What if SAIF is Not Available?

DESIGN_toggle_rate.tcl

```
create_clock -p 4 [get_ports clk]
set_case_analysis 0 [get_ports "rst se tm"]
set_switching_activity -toggle_rate 0.02 a
set_switching_activity -toggle_rate 0.06 b
set_switching_activity -toggle_rate 0.11 c
set_app_var power_default_toggle_rate 0.003
report_power
… …
```



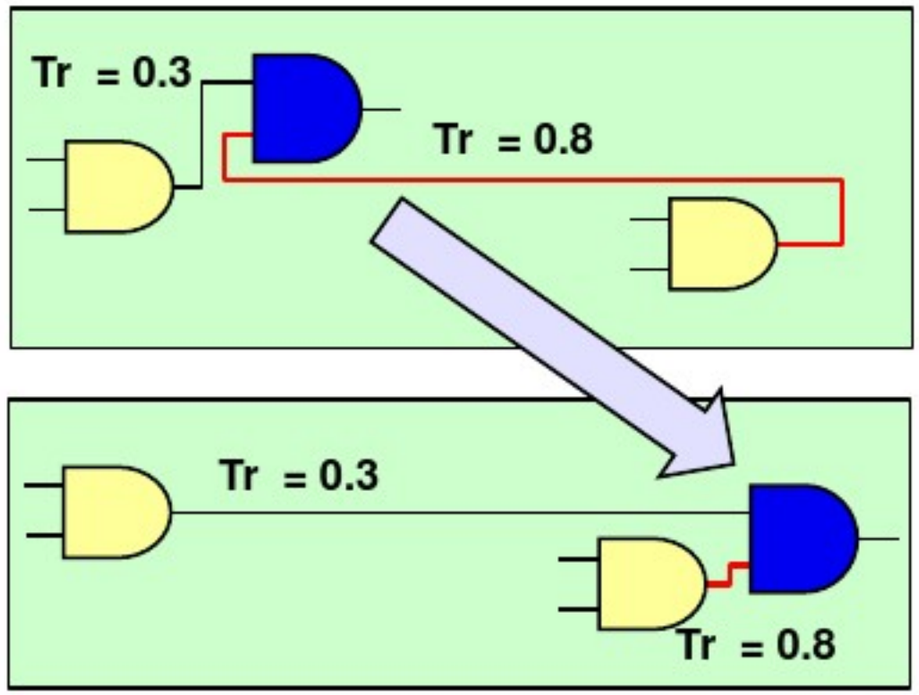
`source DESIGN_toggle_rate.tcl`

- **Ensure properly defined clocks**
- **Use `set_case_analysis` for signals that do not toggle**
  - **E.g. reset, test_mode and scan_en signals**
- **Specify a toggle rate on propagation starting points:**
  - Set any known toggles on primary inputs and black box outputs
  - Rest of the starting points will use the default toggle rate
- **IC Compiler will propagate the toggles throughout the design**

Dynamic Power Optimization: LPP

Low Power Placement (LPP):

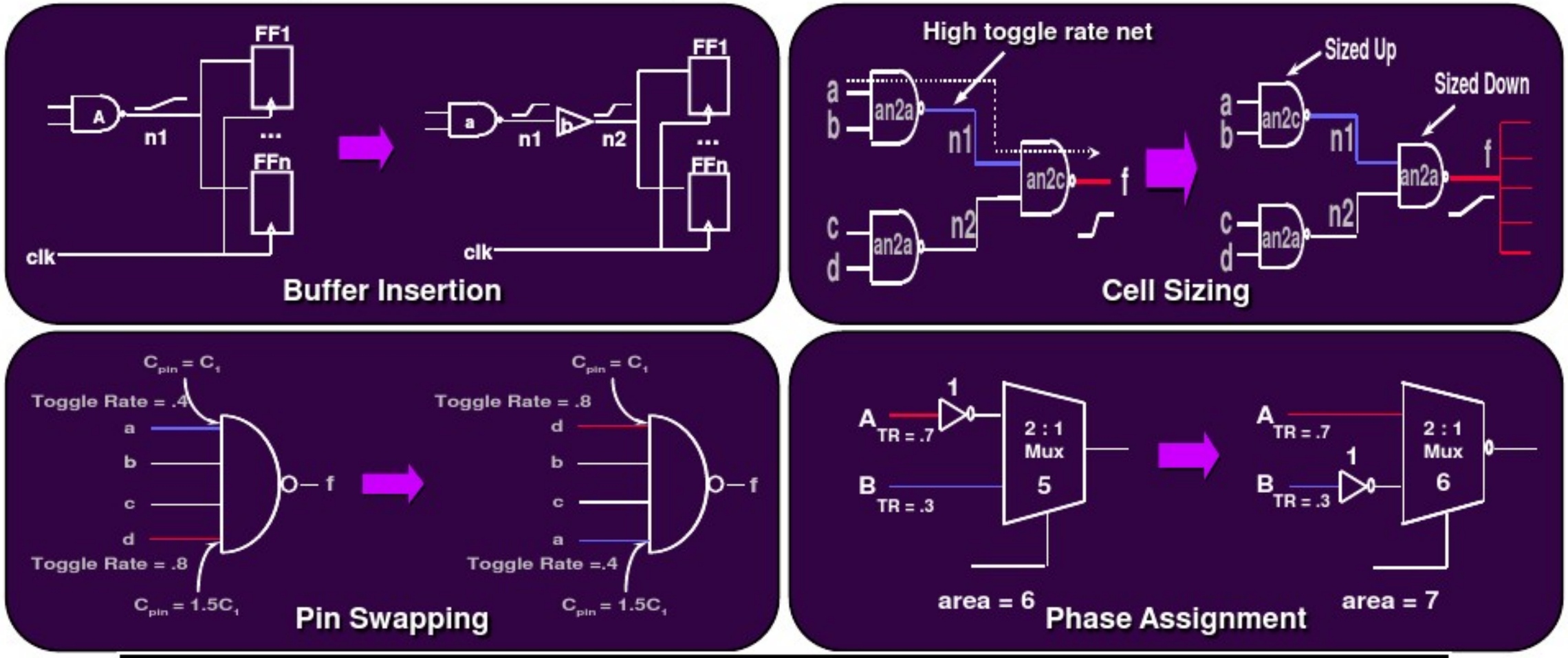
- **Moves cells closer to shorten <u>non-clock</u> high-activity nets**
  - Clock tree LPP is done later, after `place_opt`, but pre-CTS
- **Must be enabled – OFF by default**




```
read_saif -input DESIGN.saif -instance_name I_TOP
# If SAIF is not available:
# source DESIGN_toggle_rate.tcl
set_power_options -low_power_placement true
report_power_options
...
place_opt -power
```

Dynamic Power Optimization: GLPO

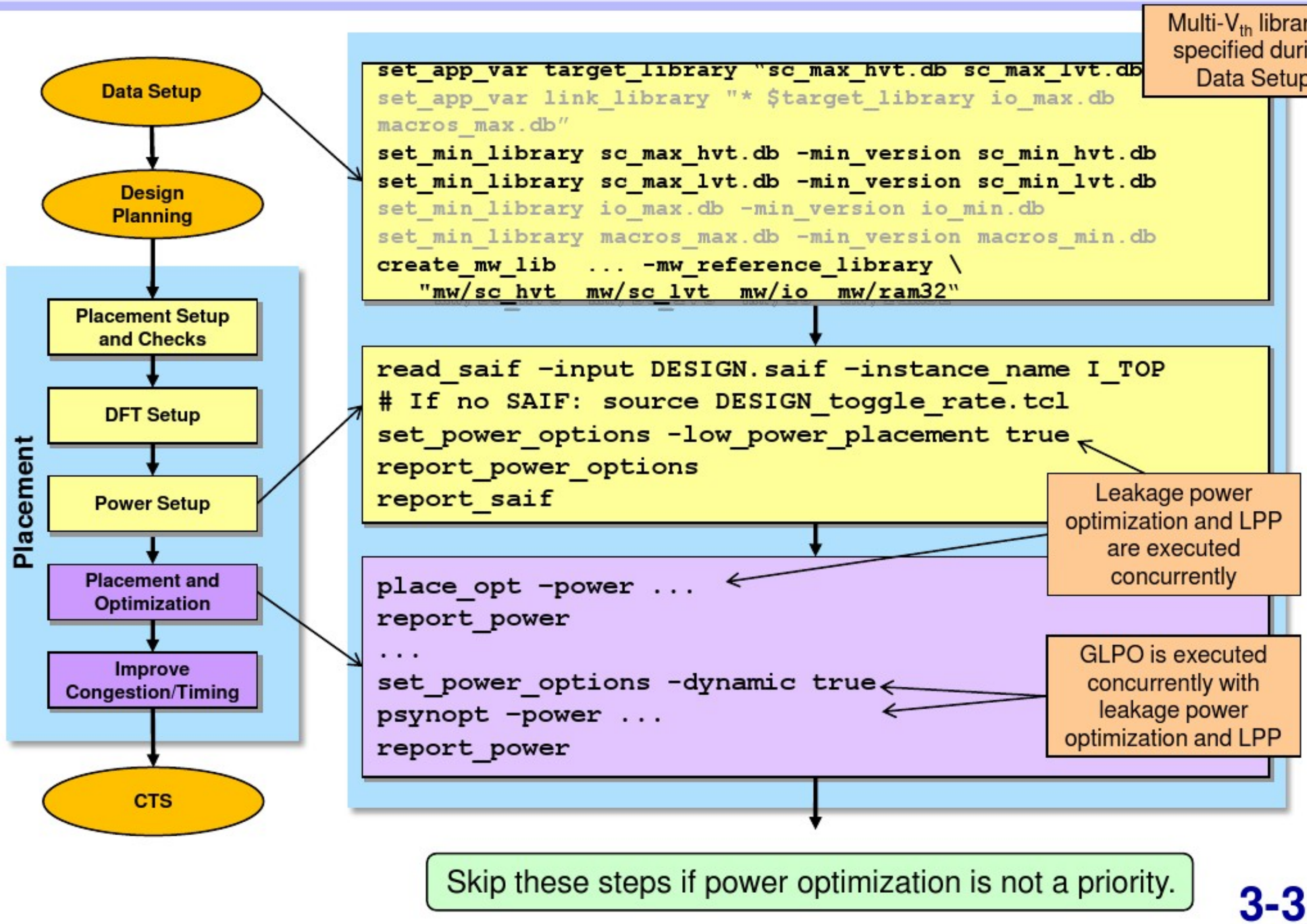
Gate-level power optimization (GLPO) optimizes logic to reduce power – must be enabled




```
set_power_options -low_power_placement true
place_opt -power
...
set_power_options -dynamic true
psynopt -power
```

*Note: *GLPO* can also be invoked concurrently with LPP during `place_opt`

Summary: Power Optimization Flow

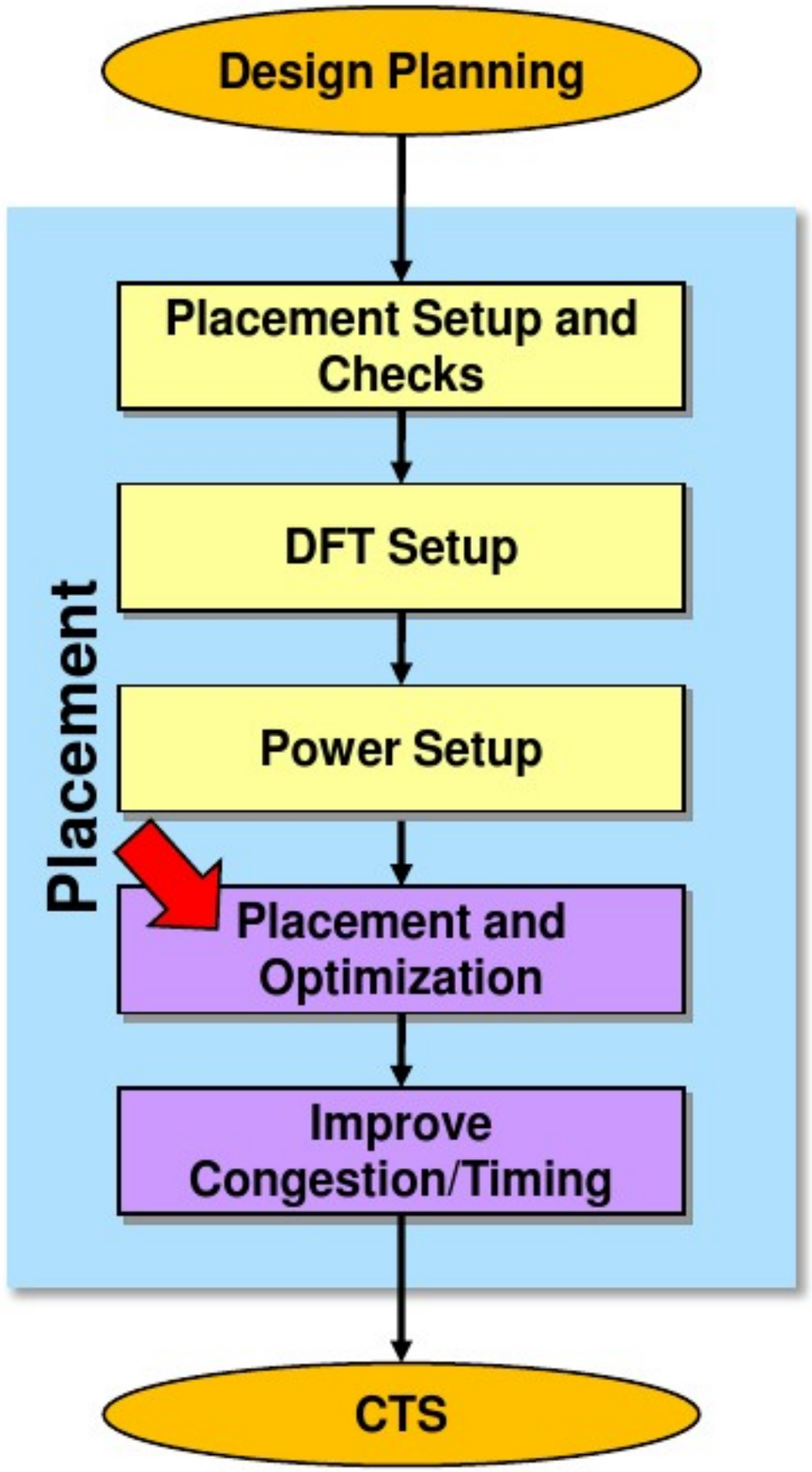


Skip these steps if power optimization is not a priority.

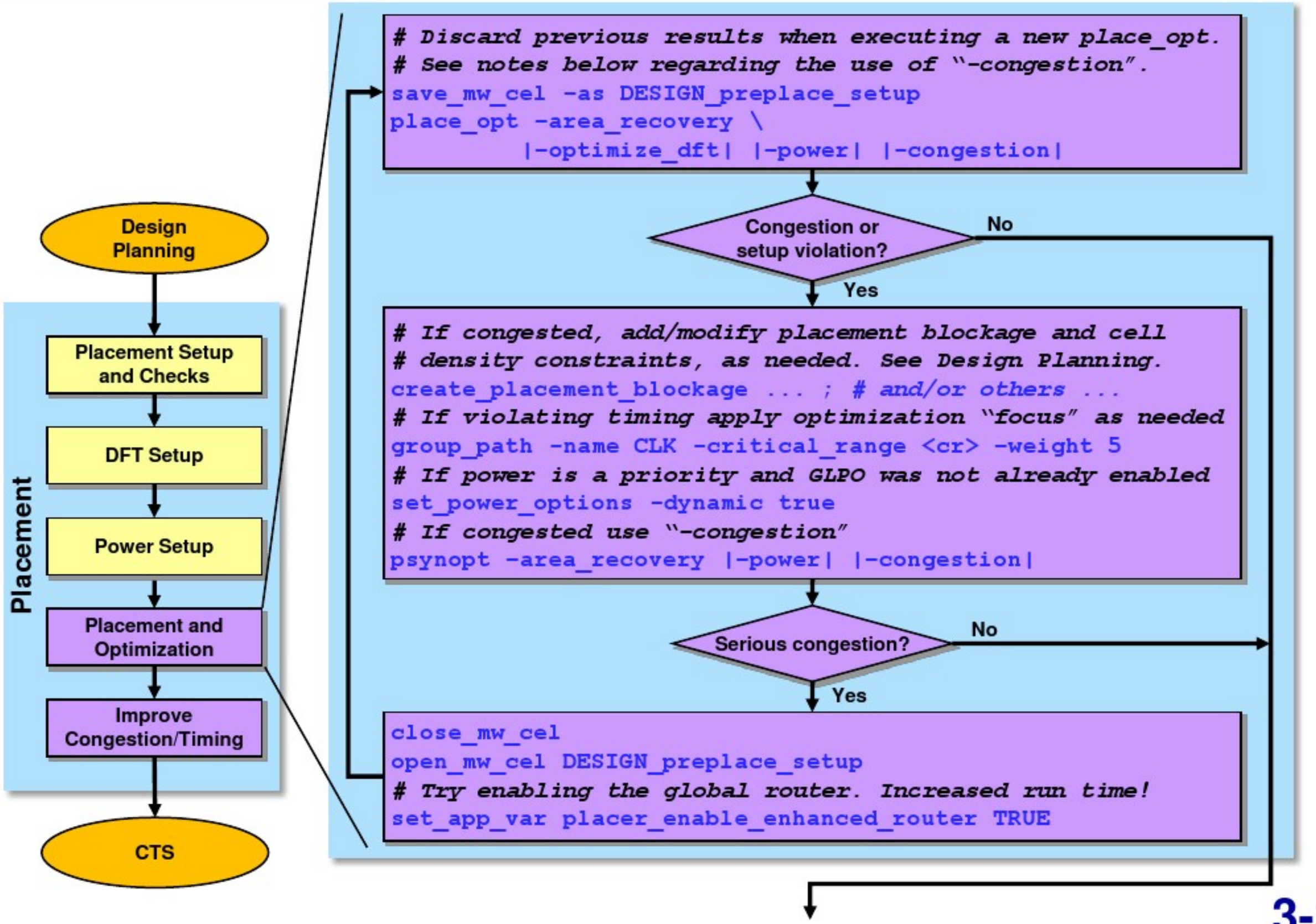
Test For Understanding

1. **NDR rules define non-default rules for <u>routing</u>. Why apply these before <u>placement</u>?**
2. **The *SCANDEF* file contains the scan ordering information which will be maintained during placement with `place_opt -optimize_dft`. *True* or *False*?**
3. **Leakage power optimization is enabled by default with `place opt -power`, and is more effective with multiple V_{th} libraries. *True* or *False*?**
4. **Low Power Placement (LPP)**
   a. Moves cells to shorten high-activity nets
   b. Spreads out high activity cells to reduce power density
   c. Is enabled with `set_power_options -dynamic true`
   d. All of the above
5. **SAIF is preferred over user-defined toggle rates for accurate leakage power optimization. *True* or *False*?**

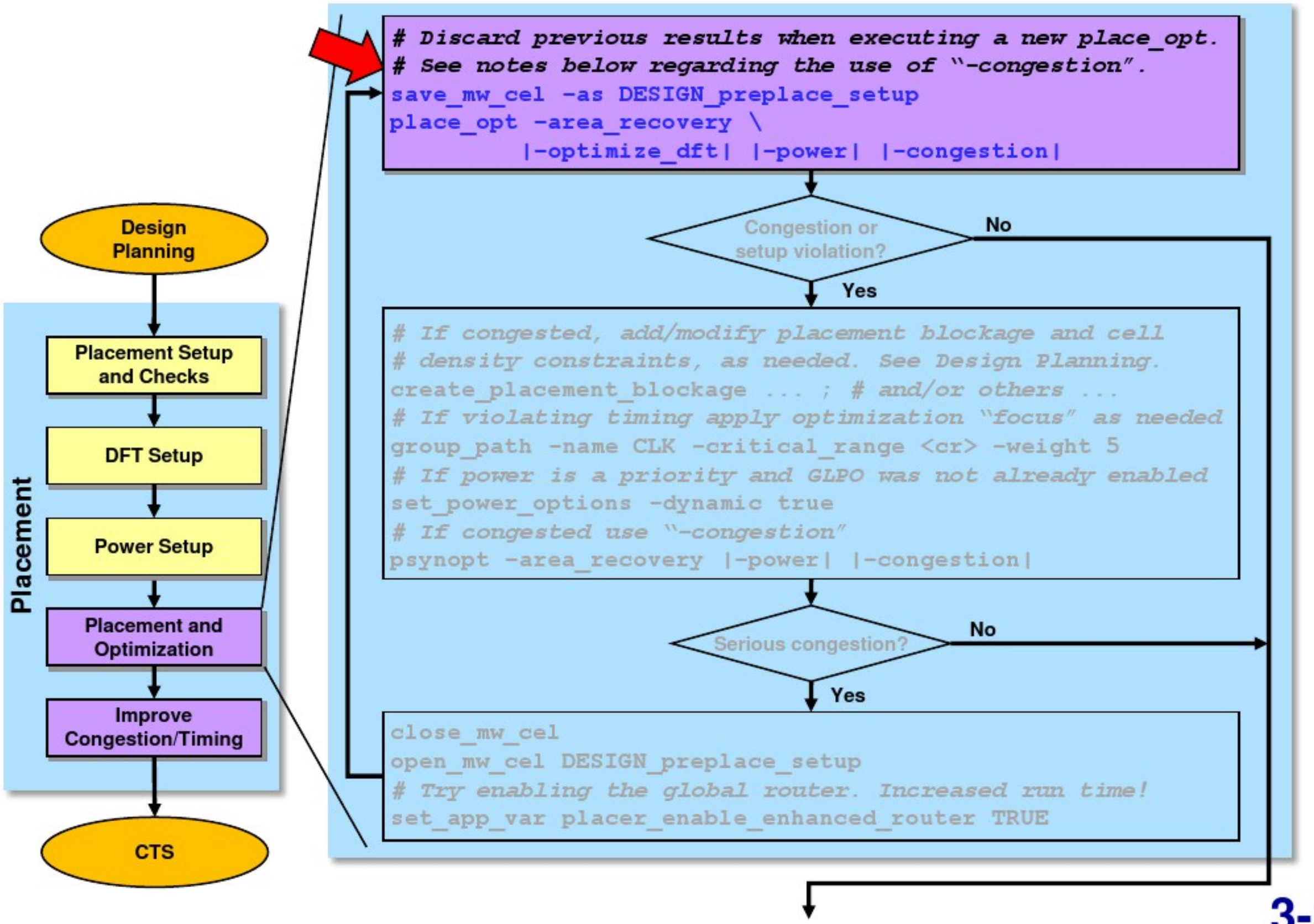
Placement and Optimization



Overview: Placement and Optimization

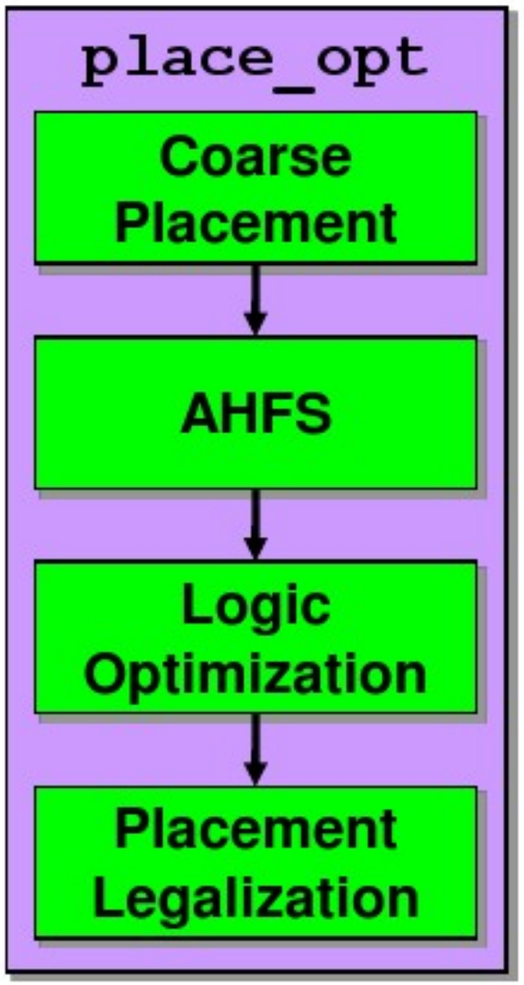


The Initial Placement and Optimization



Placement and Logic Optimization

`place_opt`	Performs timing- and congestion-driven placement and logic optimization
`-area_recovery`	Enables buffer removal and cell down-sizing of non-critical paths
`-optimize_dft`	Performs scan chain re-ordering
`-power`	Invokes leakage and/or dynamic power optimization
`-congestion`	Invokes additional congestion-driven algorithms




```
place_opt -area_recovery \
          |-optimize_dft| |-power| |-congestion|
```

Consider:
- Using `-area_recovery`: Can help reduce congestion and power; May impact run time
- Using `-optimize_dft` and `-power` only if DFT and power optimization are required, resp.
- Using `-congestion` only if congestion is known to be a serious issue (see next slide)

Considerations for Using `-congestion`

```
place_opt ... -congestion
```

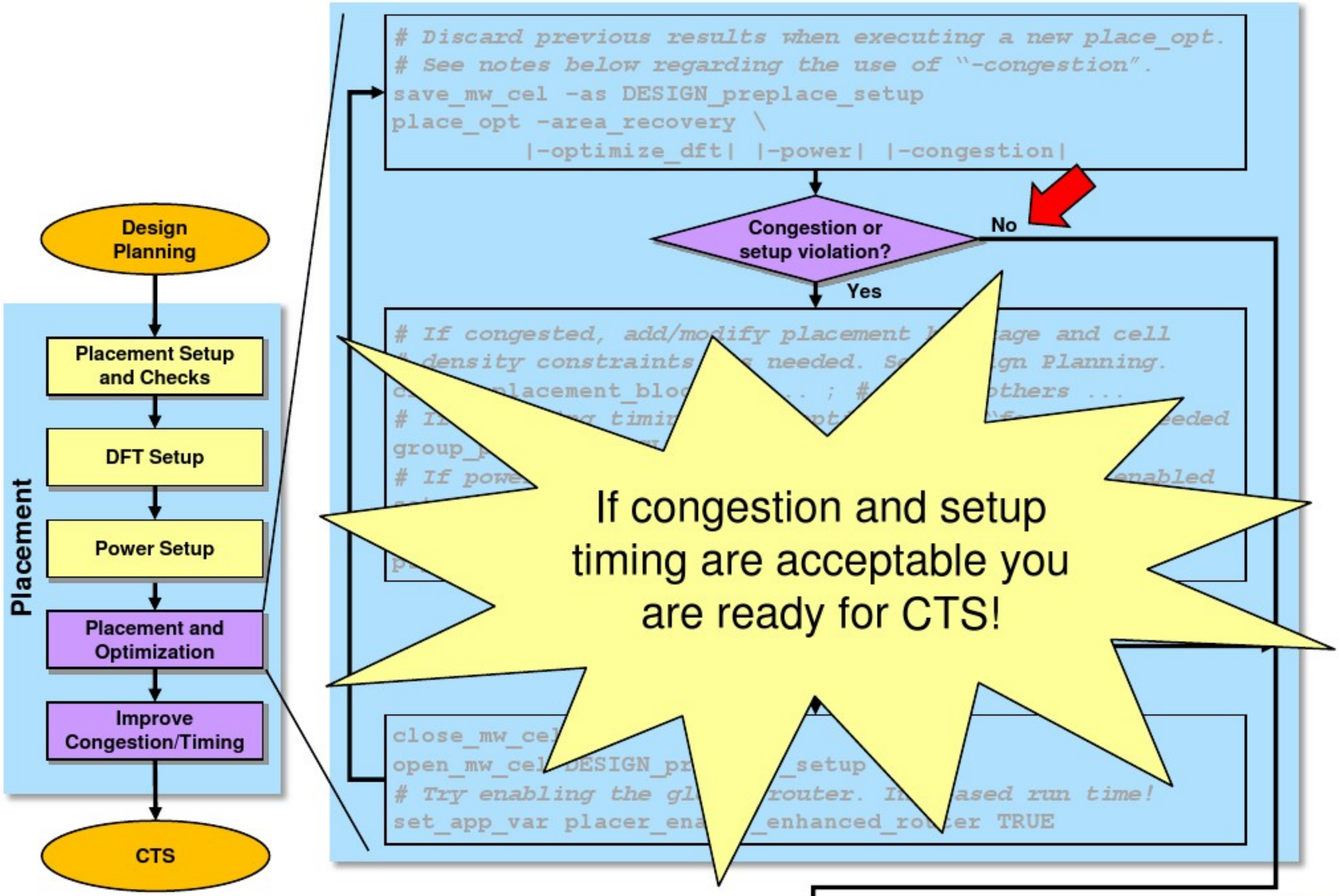
Use `-congestion` <u>only</u> if congestion is known to be a serious issue prior to placement, for example:

- `place_opt` **was first performed without** `-congestion` **and the resulting design was congested**
  - Discard previous results before re-running `place_opt`
- **During design planning (using IC Compiler) the "high effort" congestion algorithm was invoked:**
  `set_fp_placement_strategy` **`-congestion_effort high`**
  `create_fp_placement` **`-congestion`** `-timing`
- **During design exploration (after 3rd party design planning)** `-congestion` **was required:**
  `place_opt -effort low` **`-congestion`**

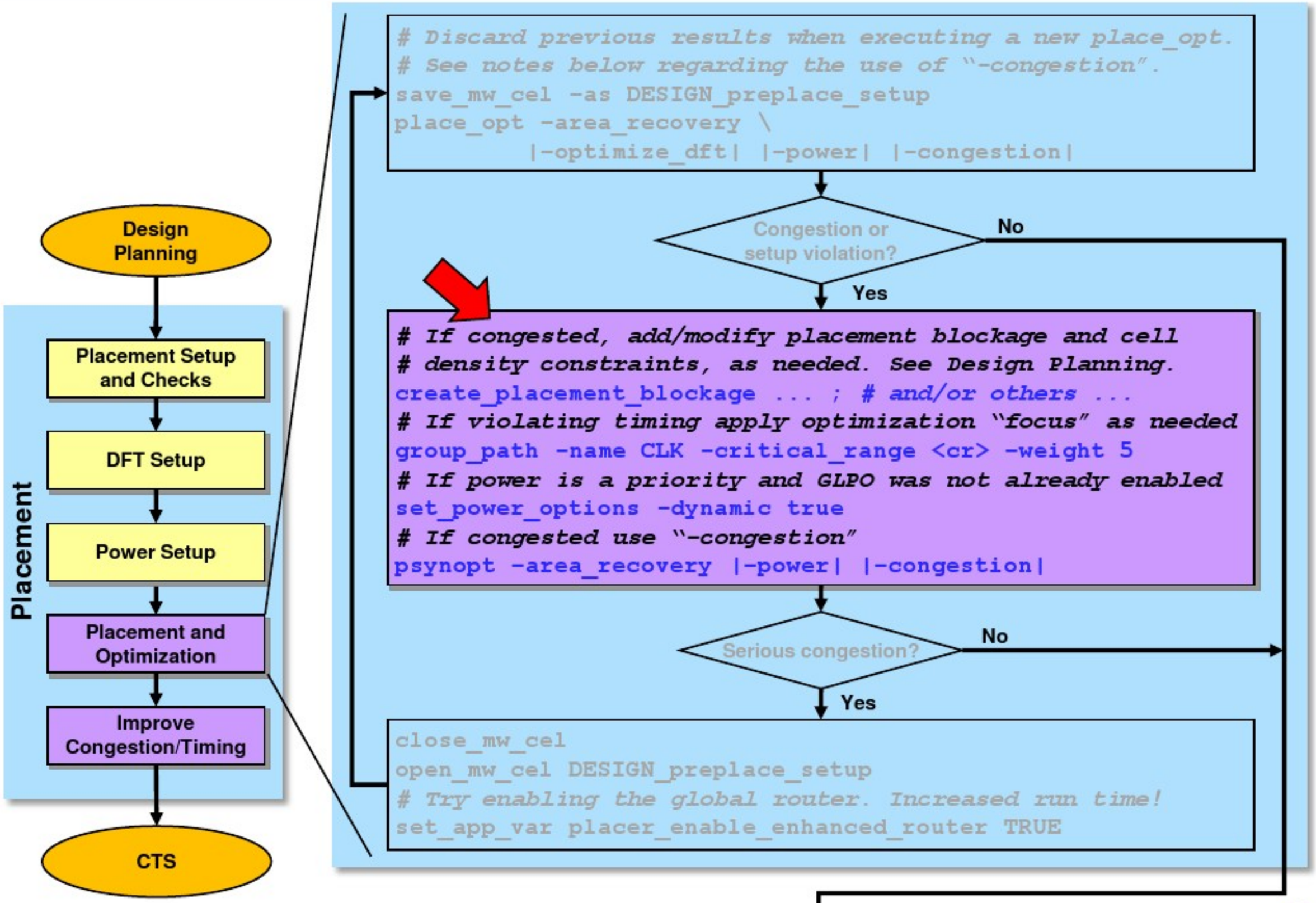
No Hold Time Fixing

- By default `place_opt` tries to fix only setup time violations - No hold time fixing
- Hold time will be addressed during clock tree synthesis

Post-Placement Analysis



Incremental Optimization



Analyze Congestion

If the design is congested after `place_opt` analyze the congestion, and if applicable:

- **Apply new, or modify existing placement blockages**
- **Apply new, or modify existing cell density constraints**
  - See “Design Planning” for details

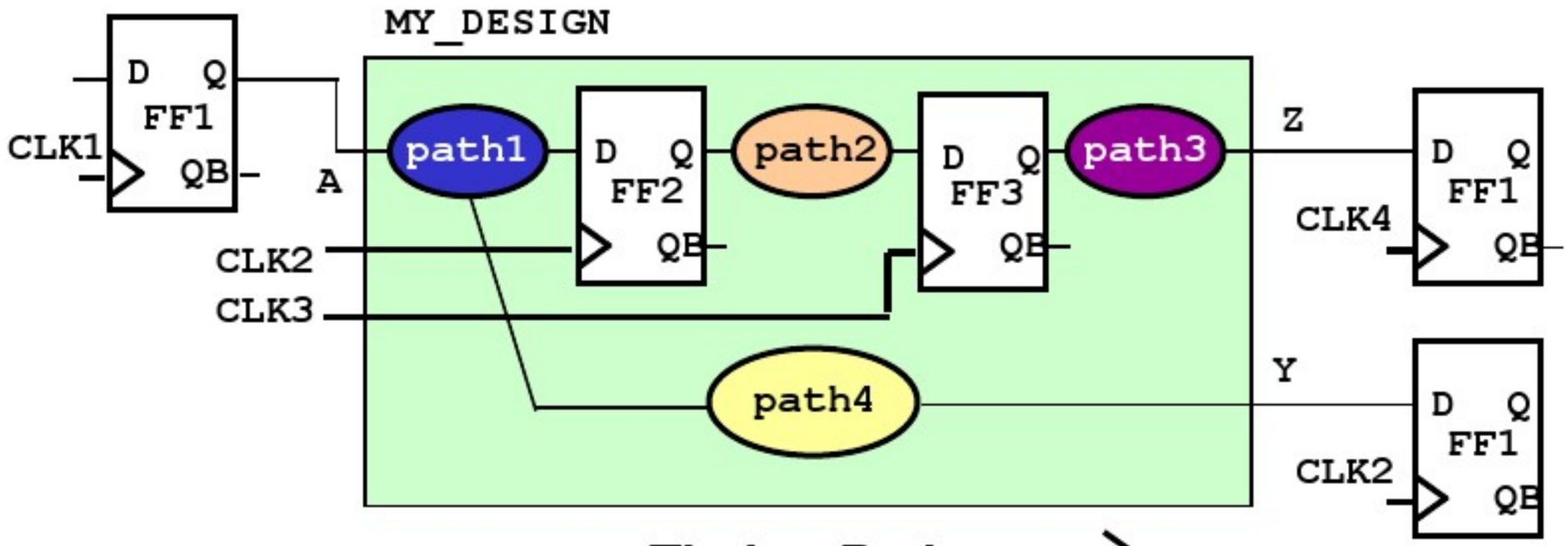
```
set_pnet_options -partial | -complete ...
set_app_var physopt_hard_keepout_distance ...
set_app_var placer_soft_keepout_channel_width ...
set_congestion_options ...
set_keepout_margin ...
create_placement_blockage ...
```

Analyze Timing Violations

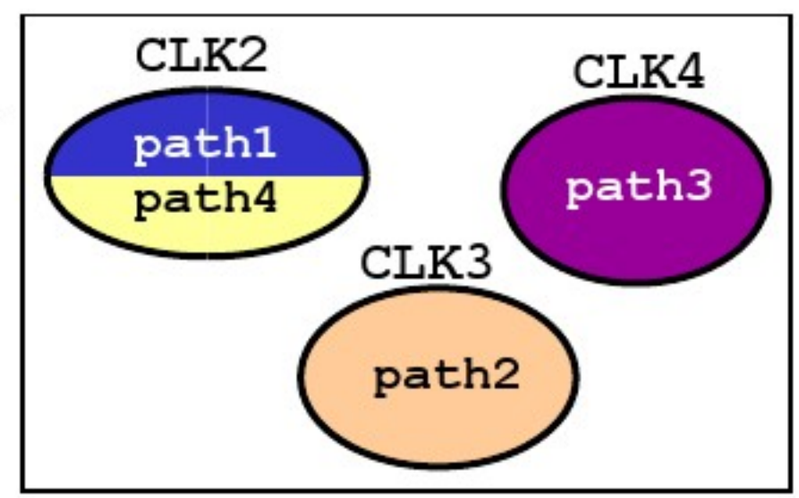
If the design has timing violations after `place_opt` analyze the violations:

- **If the critical path is an input or output path**
  - May not be a true violation - caused by over-constraining
  - May prevent true “sub-critical” reg-to-reg violations in the same “path group” from being optimized
- **If the critical path is a reg-to-reg path**
  - May be preventing “sub-critical” reg-to-reg violations in the same “path group” from being optimized

Paths are Grouped for Efficient Optimization



Timing Paths



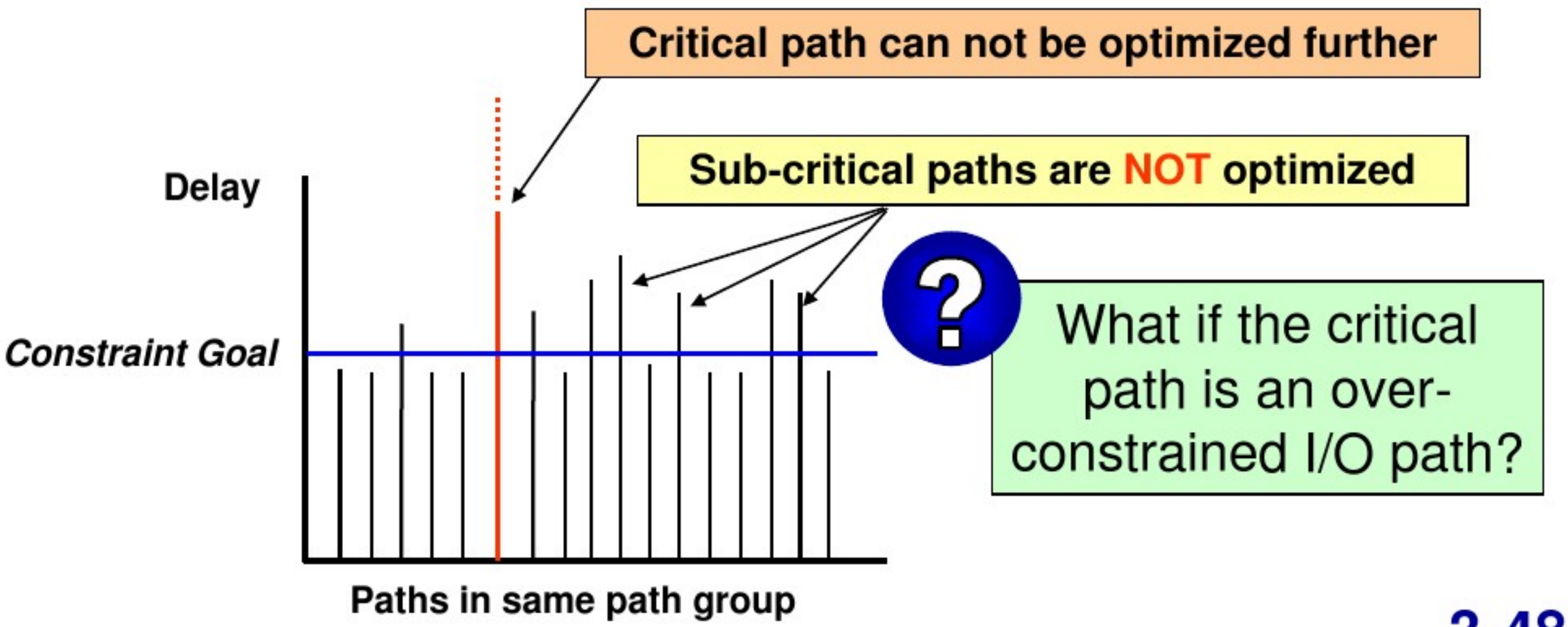
Path Groups
(named after capturing clock)

- **Paths are grouped by the clocks controlling their <u>endpoints</u>**
  - Path groups inherit the name of the related end-point clock, by default
- `place_opt` **optimizes timing efficiently: One *path group* at a time, starting with the <u>critical path</u> in each group**

General Problem: Sub-Critical Paths Ignored

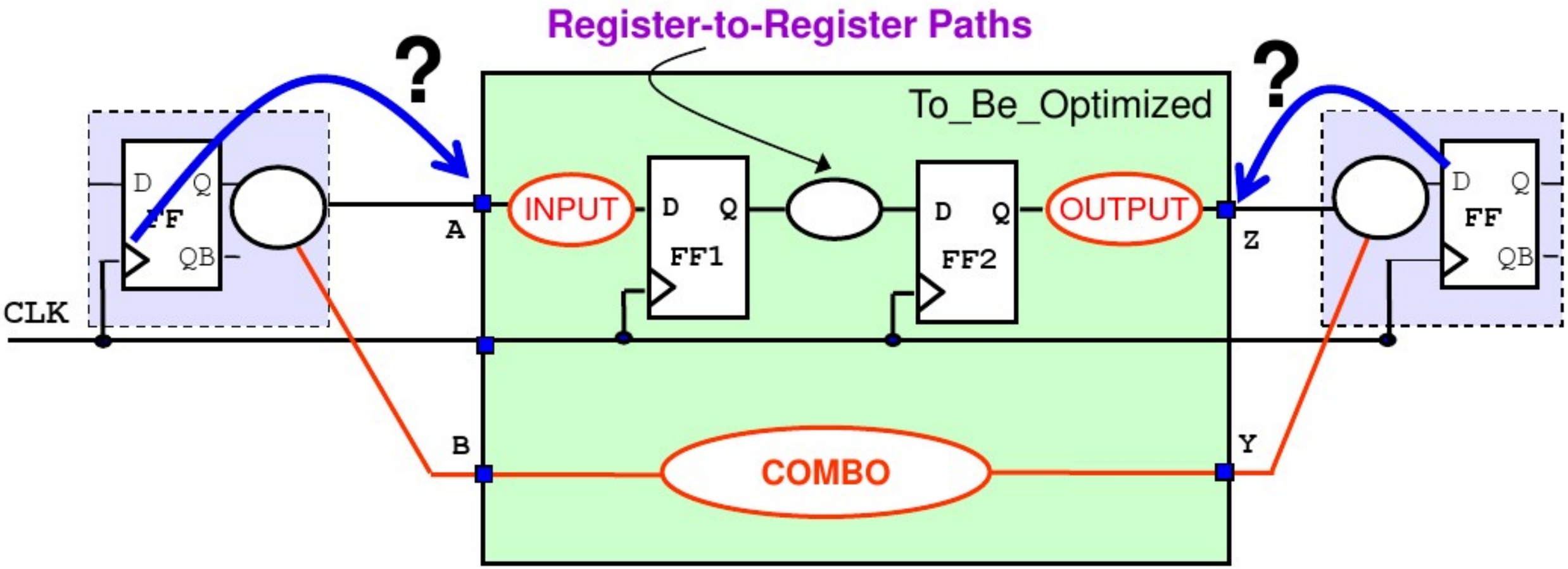
By default, optimization within a path group stops when:

- **All paths in the group meet timing, or**
- **Optimization gets stuck on the critical path**
  - Sub-critical paths are not optimized - to save run time!



Serious Problem: Reg-to-Reg Paths Ignored

If your design's hierarchy 'slices through' combinational logic, the I/O constraints may be conservatively constrained



The *critical path* will most likely be in the I/O logic

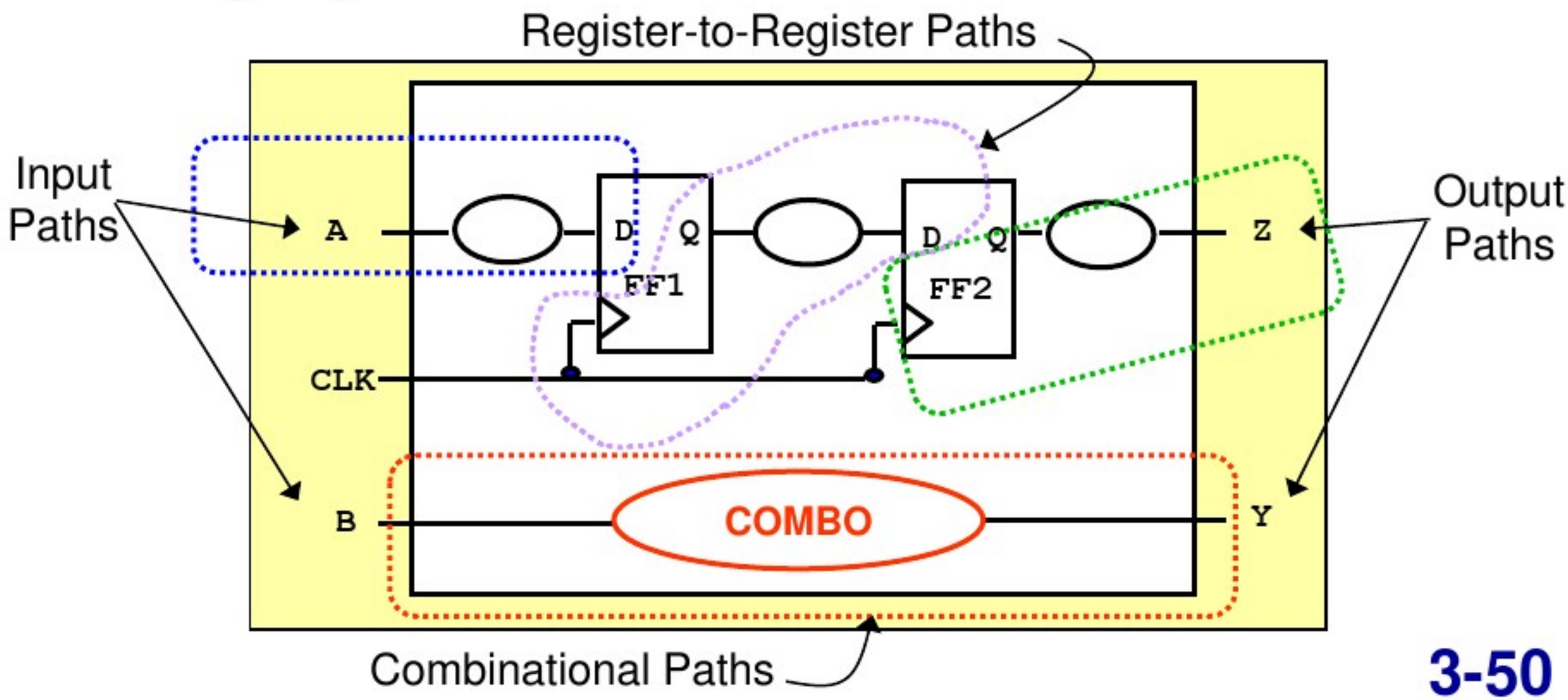
? What happens to reg-to-reg path violations if IC Compiler gives up on the I/O "critical path"?

Solution #1: User-Defined *Path Groups*

- **Check if user-defined path groups exist for I/O paths**

  `report_path_group` and if not,

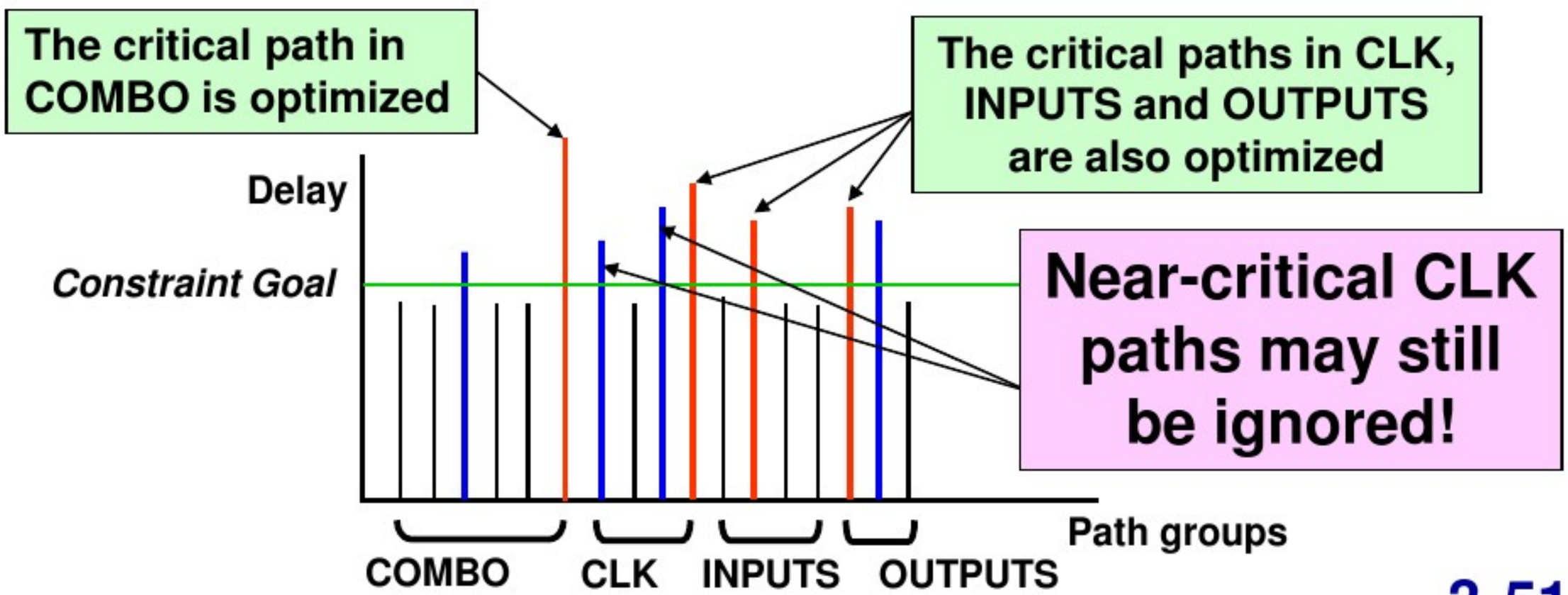
- **Create *path groups* for I/O paths**
  - Each *path group* is optimized independently
  - Worst violator in one path group does not prevent optimization in another group



Define Path Groups for I/O Paths, if needed

```
# Ensure that the reg-reg paths get optimized
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
```

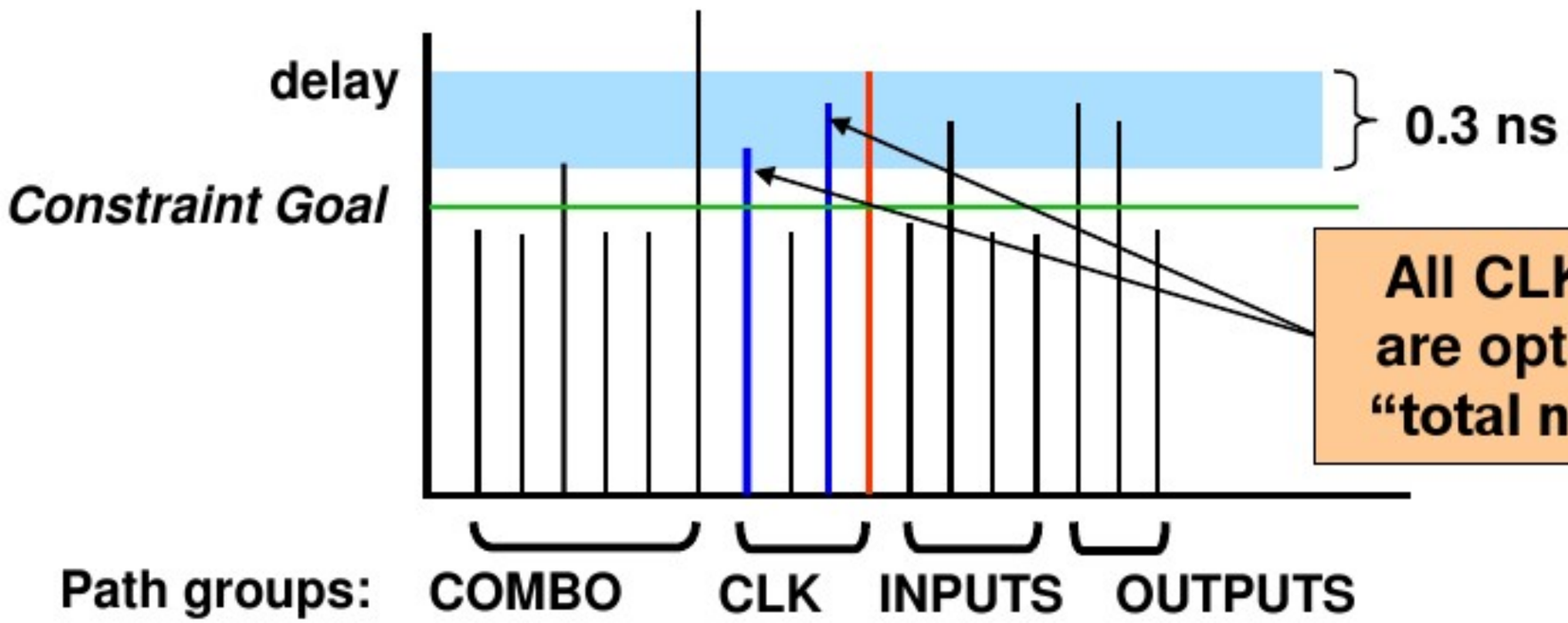
Where are the reg-to-reg paths?
Are the COMBO paths in three path groups?



Solution #2: Apply a *Critical Range*

Reg-to-reg paths are in path groups corresponding to their capturing clock's name, by default.

```
group_path -name CLK -critical 0.3
```

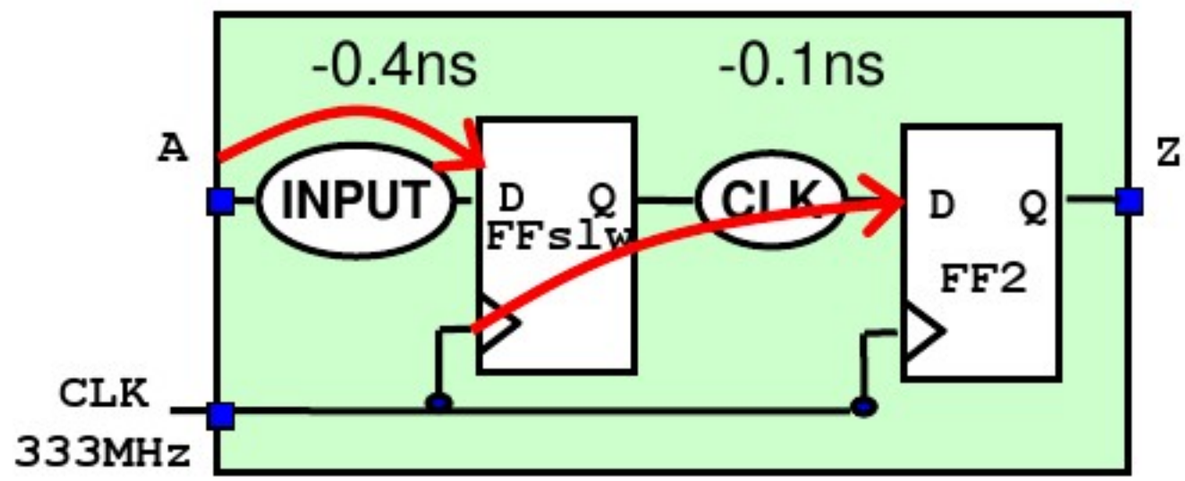


All CLK paths within range are optimized, reducing the "total negative slack" (TNS)

- **The *critical range* is always relative to the critical path delay**
- **Fixing related sub-critical paths may help the critical path**
- **Should not exceed 10% of the group's effective clock period**
  - Will increase run time

Solution #3: Prioritizing Path Groups

- **All path groups have equal priority, by default**
- **Applying a *weight* to a select group (e.g. CLK) allows delay improvements to its paths, which may degrade another, less critical group's worst violator (e.g. INPUTS)**

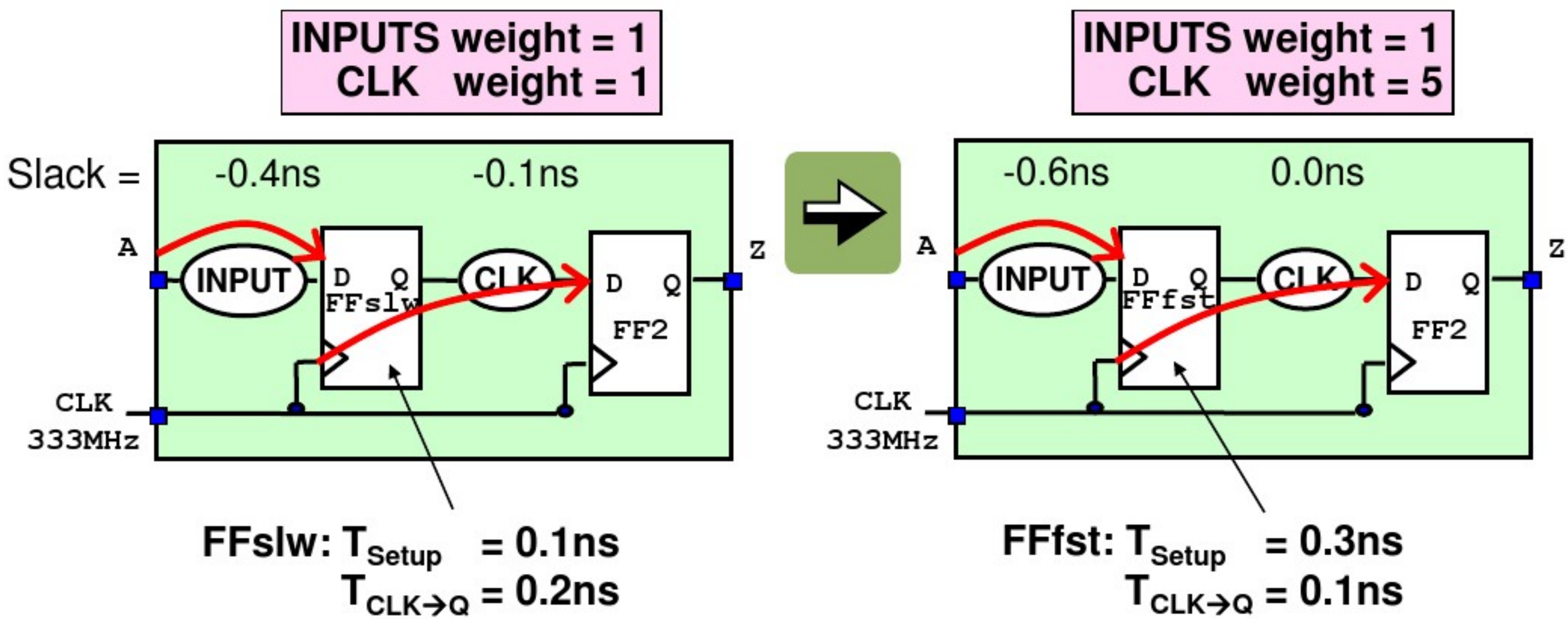


- **Suggestion:**
  - Apply a weight of 5 to the most critical paths (reg-to-reg)
  - Apply a weight of 2 to less critical paths
  - A default weight of 1 is assigned to all other paths (e.g. over-constrained I/O paths)

```
group_path -name CLK -weight 5
```

Example: Weighted Path Group

```
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] -to [all_outputs]
group_path -name CLK -weight 5
```



Complete Path Group Example

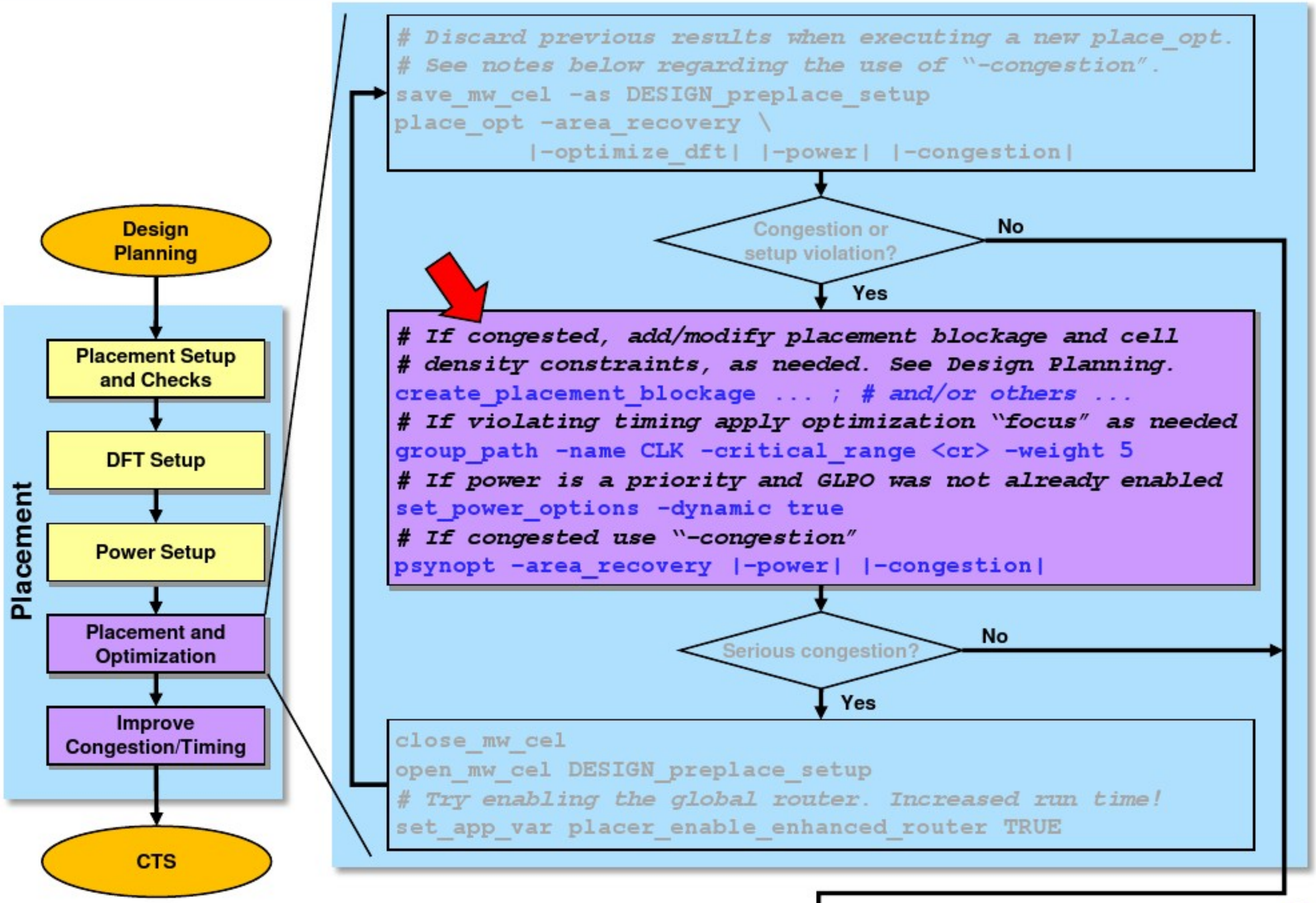
```
# Example: Assign a critical range
# and weight to clock groups
group_path -name CLK1 -critical_range 0.3 -weight 5
group_path -name CLK2 -critical_range 0.1 -weight 5
group_path -name CLK3 -critical_range 0.2 -weight 5
group_path -name INPUTS -from [all_inputs]
group_path -name OUTPUTS -to [all_outputs]
group_path -name COMBO -from [all_inputs] \
                       -to [all_outputs]
report_path_group
```

Incremental Logic Optimization: `psynopt`

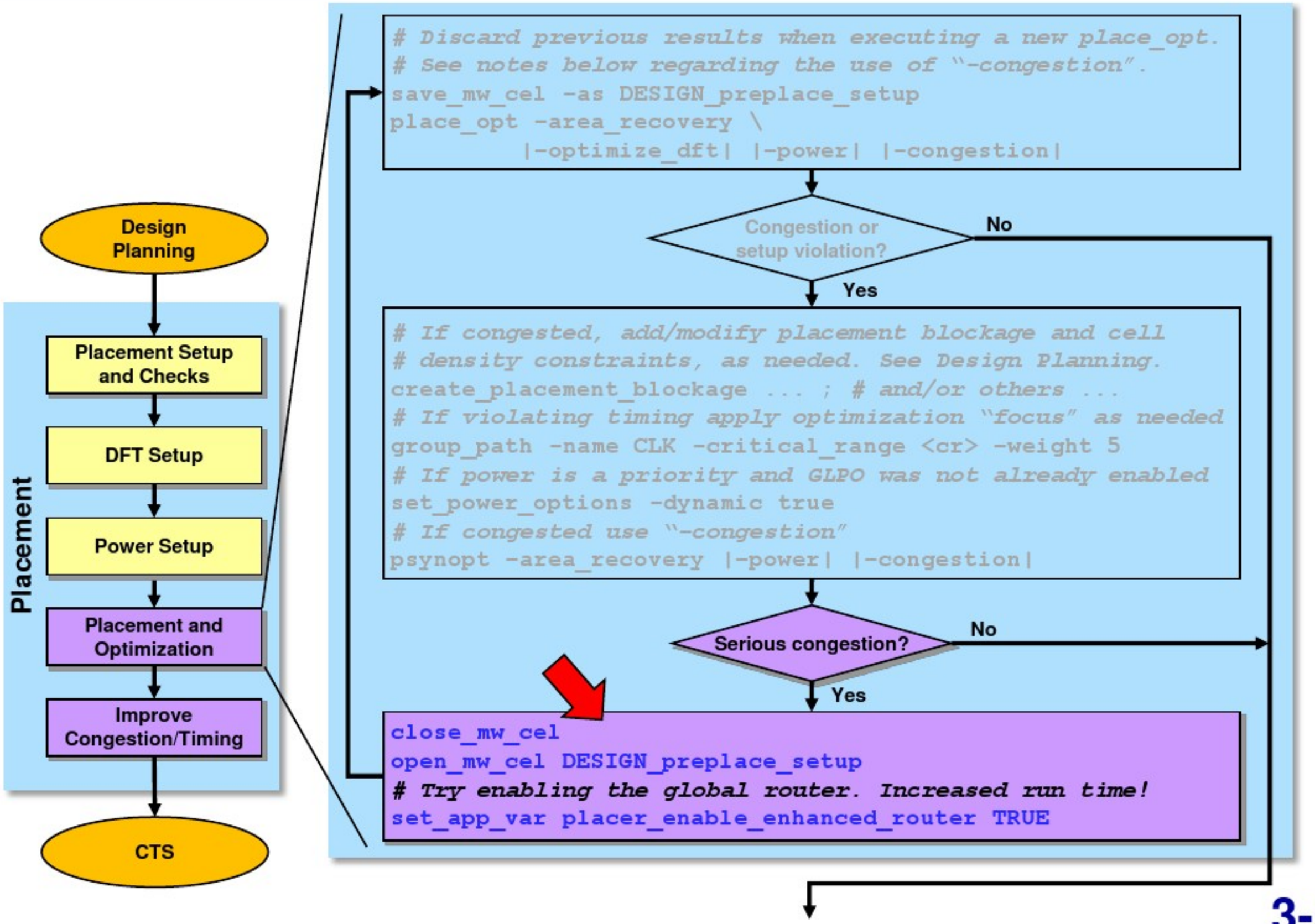
- **After modifying placement and path group parameters (as appropriate) execute `psynopt`:**
  - Performs incremental timing-driven logic optimization
  - Legalizes placement
- **If the design is congested include `-congestion`**
- **Include `-area_recovery`**
  - May help reduce congestion and power – may impact run time
- **If power optimization is a priority include `-power`**
  - Enable *GLPO* dynamic power optimization (if not already enabled prior to `place_opt -power`)

```
set_power_options -dynamic true; # GLPO
psynopt -area_recovery |-power| |-congestion|
```

Summary: Incremental Optimization



If the Design is Still Seriously Congested …

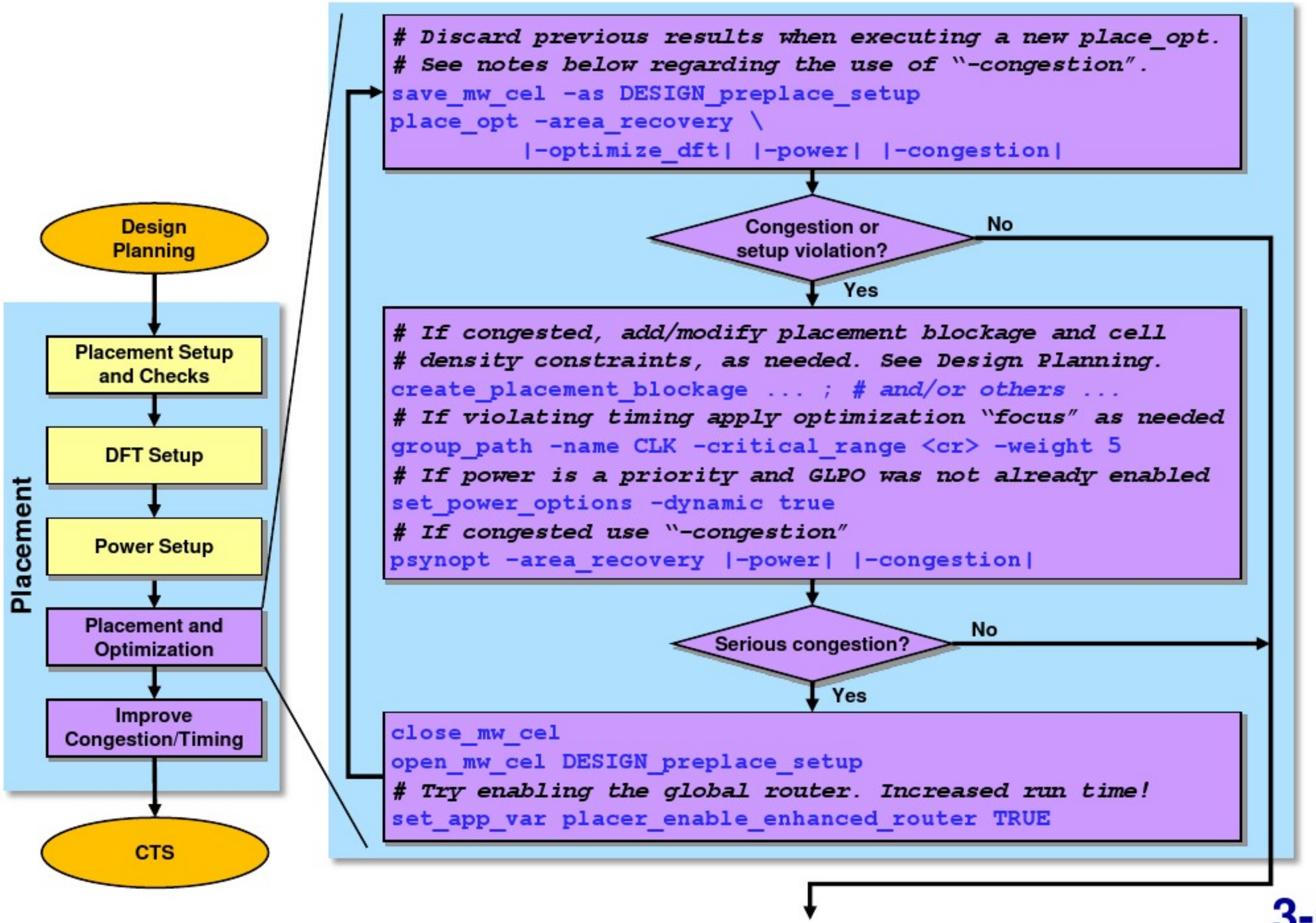


Enable Global Router During Optimization

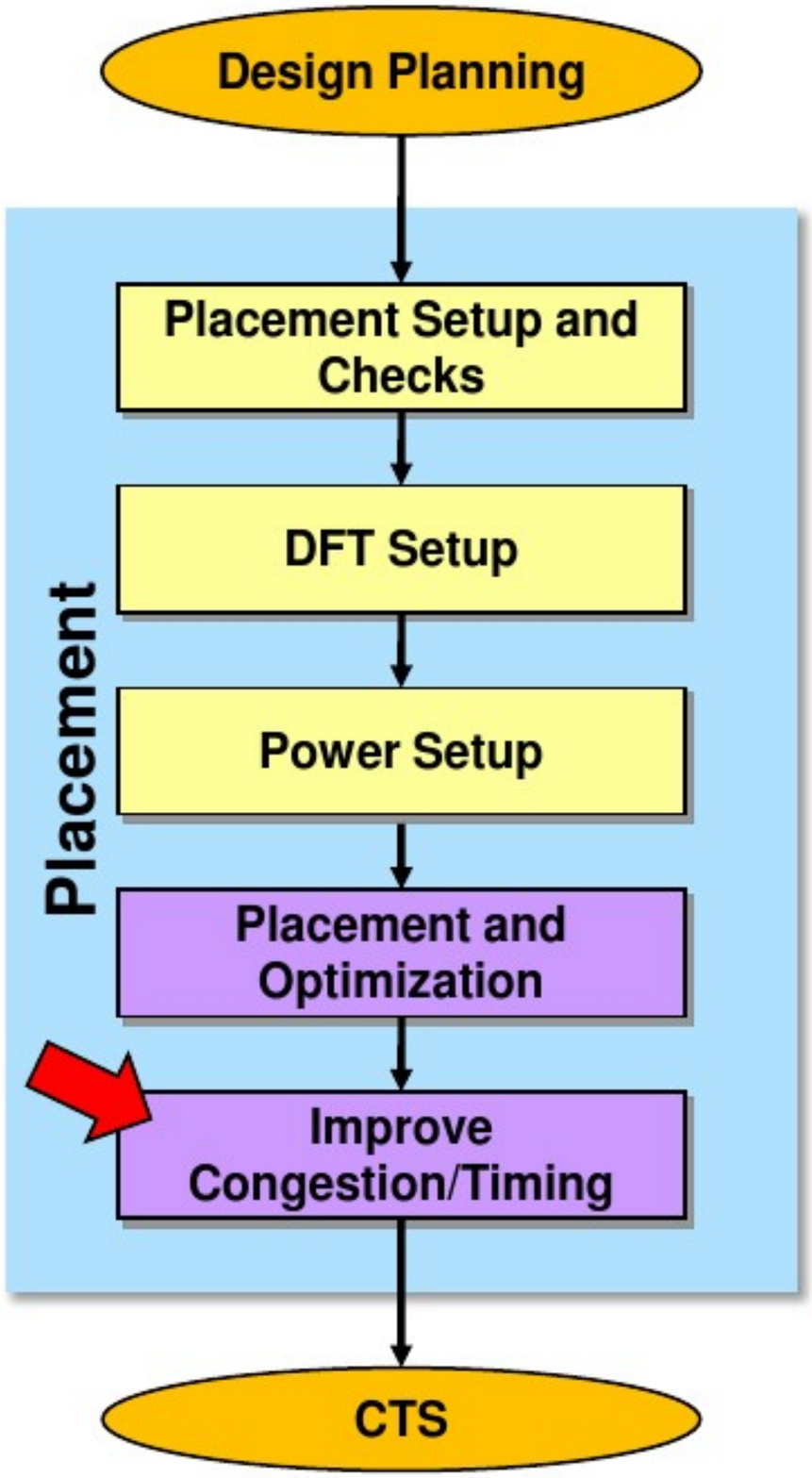
- **Setting the variable below to *TRUE* enables congestion optimization using the actual global router (instead of a route estimator)**
- **Most designs will show little change – some may show significant congestion improvement**
- **All designs will see an increase in run time!**
  - Try this only if congestion is serious
  - First discard results from previous `place_opt`
  - Re-run `place_opt` with **`-congestion`**

```
close_mw_cel
open_mw_cel DESIGN_preplace_setup
set_app_var placer_enable_enhanced_router TRUE
place_opt -area_recovery -congestion \
          |-optimize_dft| |-power|
```

Summary: Placement and Optimization

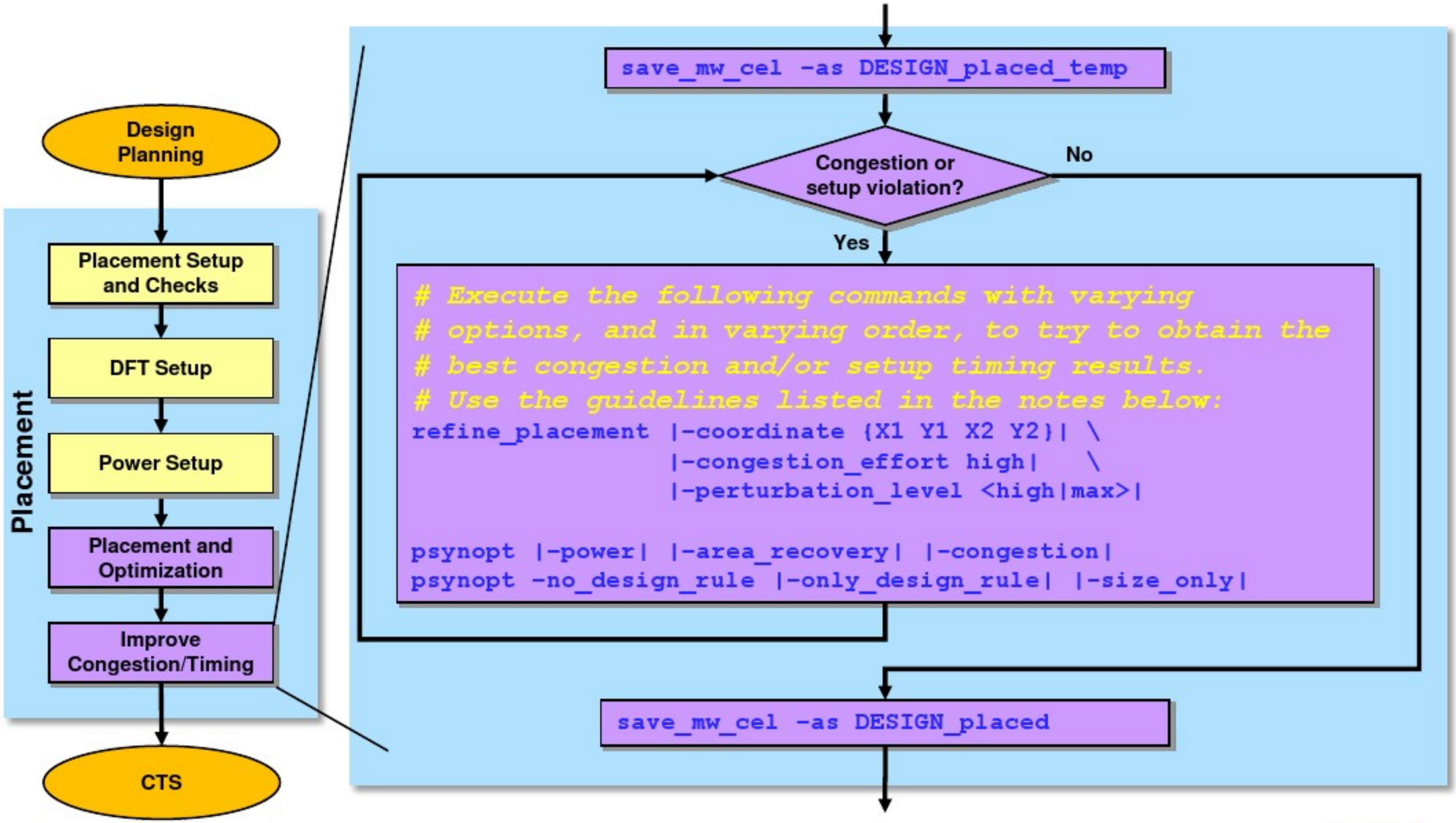


Improving Congestion and Setup Timing



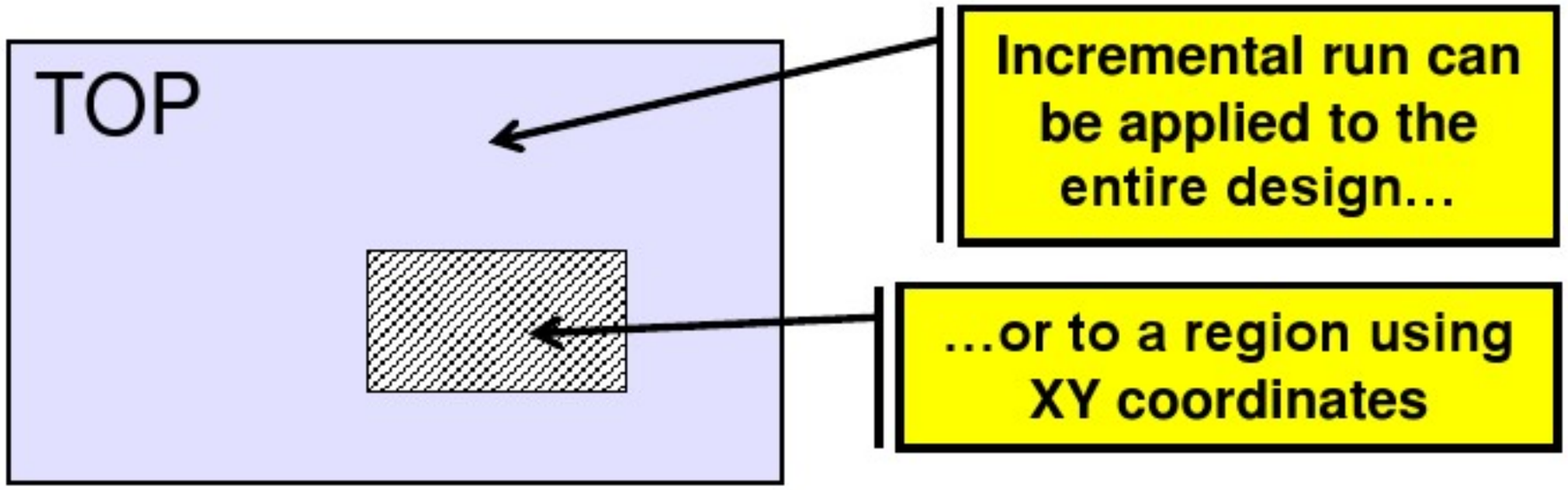
If setup timing or congestion are still a problem there are additional optimizations that can be done

Overview: Improve Congestion/Timing



refine_placement

- **Optimizes placement incrementally to improve congestion**
  - Does not modify the netlist → Timing may degrade
- **Can be executed back-to-back with different options**
  - Discarding previous placement may improve results
- **Can be followed with `psynopt` to improve timing**



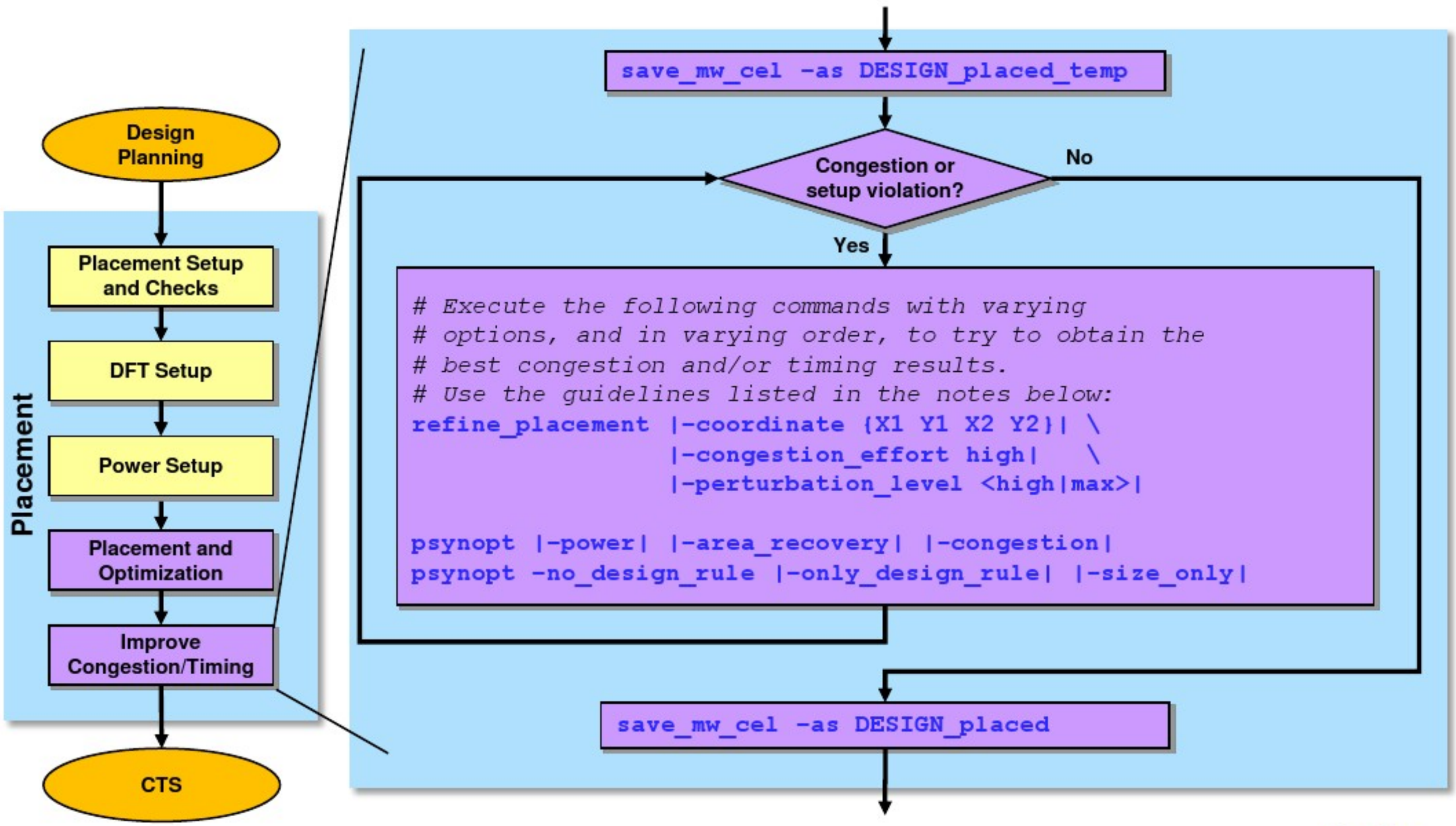

```
refine_placement  |-coordinate {X1 Y1 X2 Y2}| \
                  |-congestion_effort high|   \
                  |-perturbation_level <high|max>|
```

psynopt

- **Performs incremental timing-driven logic optimization with placement legalization (setup timing, by default)**
  - With `-congestion` will try to maintain or improve congestion
- **Use `-power`, `-congestion` or `-area` as necessary**
  - If timing is higher priority try omitting one or more of these options
- **Sometimes better timing is obtained by focusing or limiting `psynopt` with `-no_design_rule`, `-only_design_rule` or `-size_only`**
- **It is not necessary to discard previous results before executing a subsequent `psynopt`**

```
psynopt |-power| |-area_recovery| |-congestion|
psynopt -no_design_rule |-only_design_rule| |-size_only|
```

Summary: Improve Congestion/Setup Timing



Test For Understanding (1 of 2)

1. **By default, `place_opt` will**
   a. Optimize placement and logic for congestion
   b. Optimize placement and logic for setup timing
   c. Optimize logic for leakage power
   d. A and B
   e. A, B and C

2. **What is the benefit of having separate path groups for I/O logic paths?**

3. **What does applying a timing critical range do, and what is the benefit of this?**

4. **Applying a *weight* > *1* to a path group may increase a design's critical path delay. *True* or *False*?**

Test For Understanding (2 of 2)

5. `refine_placement` **performs incremental**
   a. Timing-driven placement
   b. Congestion-driven placement
   c. Congestion-driven logic optimization
   d. A and B
   e. A, B and C

6. **By default,** `psynopt` **performs incremental**
   a. Timing-driven logic optimization
   b. Congestion-driven placement
   c. Congestion-driven logic optimization
   d. A and B
   e. A, B and C

Techniques with More User Control

Buffer trees

Relative placement

Build User-Controlled Balanced Buffer Trees

- **AHFS buffers high fanout nets with little user control**
  - All types of buffers may be used
  - Buffer tree depth may be unbalanced, since only DRC focused
- **After `place_opt` analyze the resulting buffer tree(s)**
- **Remove unwanted buffer tree(s)**
- **Re-create a balanced buffer tree with more user control:**

```
remove_buffer_tree -from <pins_or_nets>
set_cbt_options -references <buffer_list> -threshold <t>
create_buffer_tree -from <pins_or_nets>
```

Build Skew-Optimized Buffer Trees

- If skew minimization is important for certain non-clock high fanout nets, use `compile_clock_tree`
- Must define the pins or nets as "ideal networks" prior to `place_opt`
- Additional commands available to exert more user control (see notes below)

```
set_ideal_network -no_propagate [get_nets Reset]
place_opt
remove_ideal_network -no_propagate [get_nets Reset]
compile_clock_tree -high_fanout_net [get_nets Reset]
```

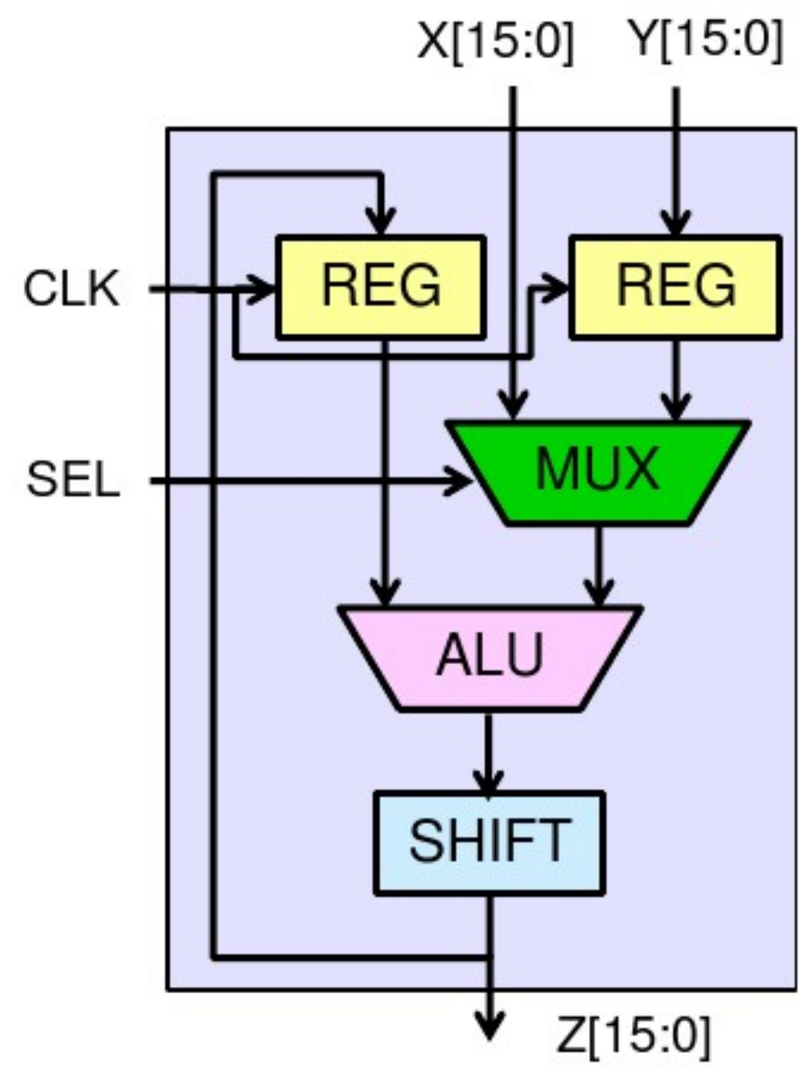
Relative Placement

Buffer trees

Relative placement

What's Special about Data Path Logic?

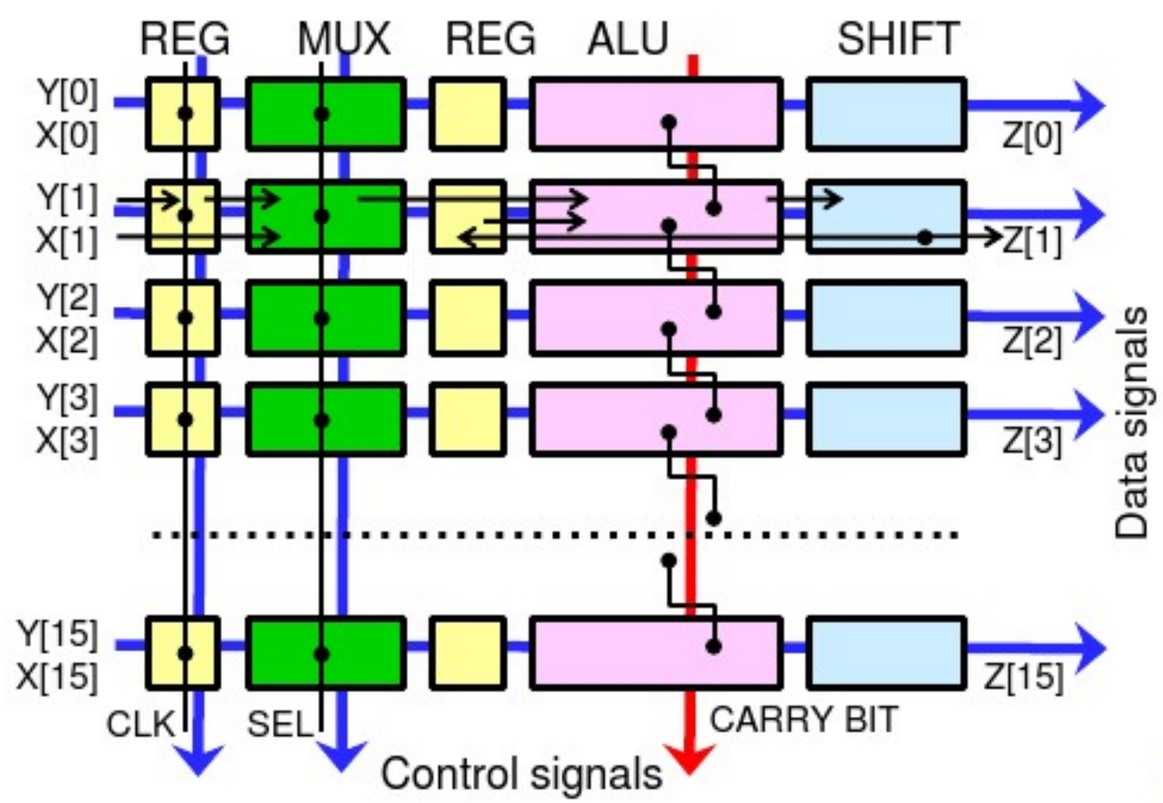
- **Bit-wise data operations are performed in parallel on each bit of a bus**
  - Each operation corresponds to a dedicated function, e.g. *adder*, *multiplier*, *register*, *multiplexer*, etc
- **Two groups of signal flows:**
  - Data flow: *data-in* to *data-out*
  - Control flow:
    - Global: *clock*, *select, enable*
    - Local: *carry-in, carry-out*



Example data path functional module

The Ideal Layout for Data Path

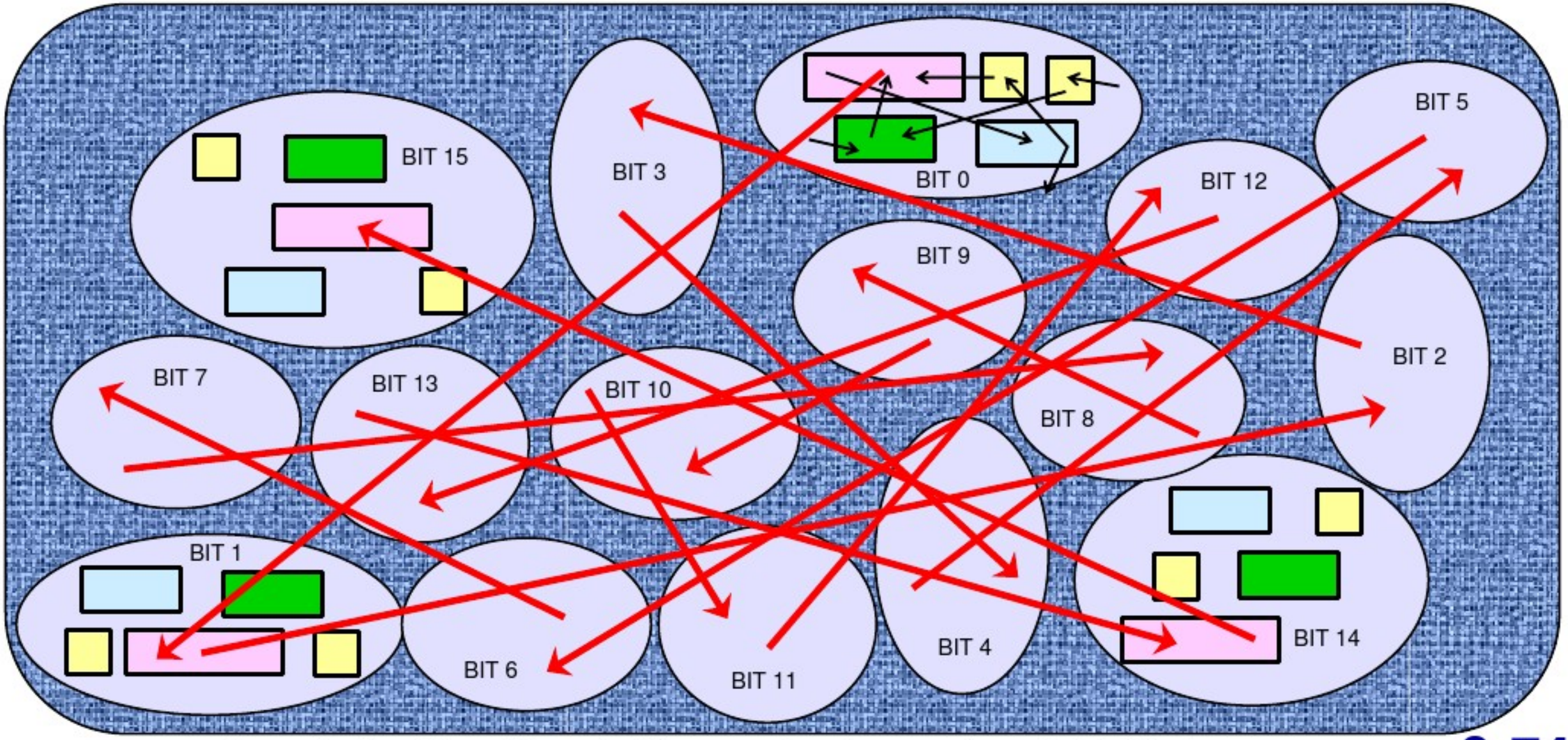
- **Best placed in a bit-sliced structure, for example:**
  - Cells operating on one bit are placed in one row, abutted next to each other horizontally
  - This row is repeated and abutted vertically for each bit
- **Benefits of this placement structure:**
  - Overall placement area of data path cells is minimized
  - Control and data signals can be routed vertically and horizontally, respectively
    - Minimizes routing congestion
    - Minimizes interconnect parasitic RCs
      - Reduced delay
      - Reduced power
    - Minimizes clock and data skew



Data Path Layout using Traditional P&R Tool

Traditional layout tools can not take advantage of the regular bit-sliced structure, resulting in:

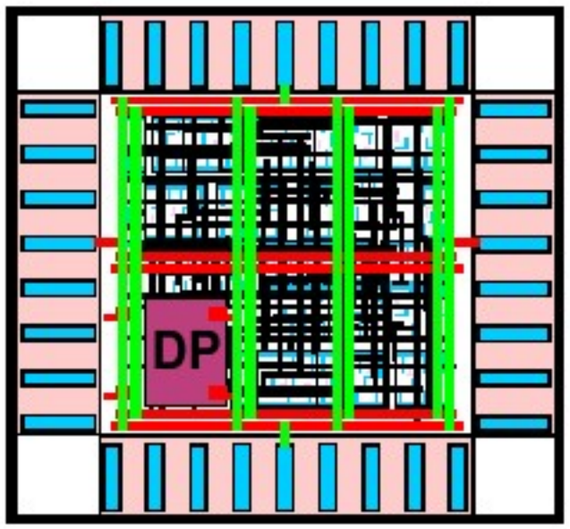
- Larger placement area and routing congestion
- Larger delays, skews and power consumption



Example “traditional” layout with carry-signal connections highlighted

Traditional Solution: Custom/Manual Layout

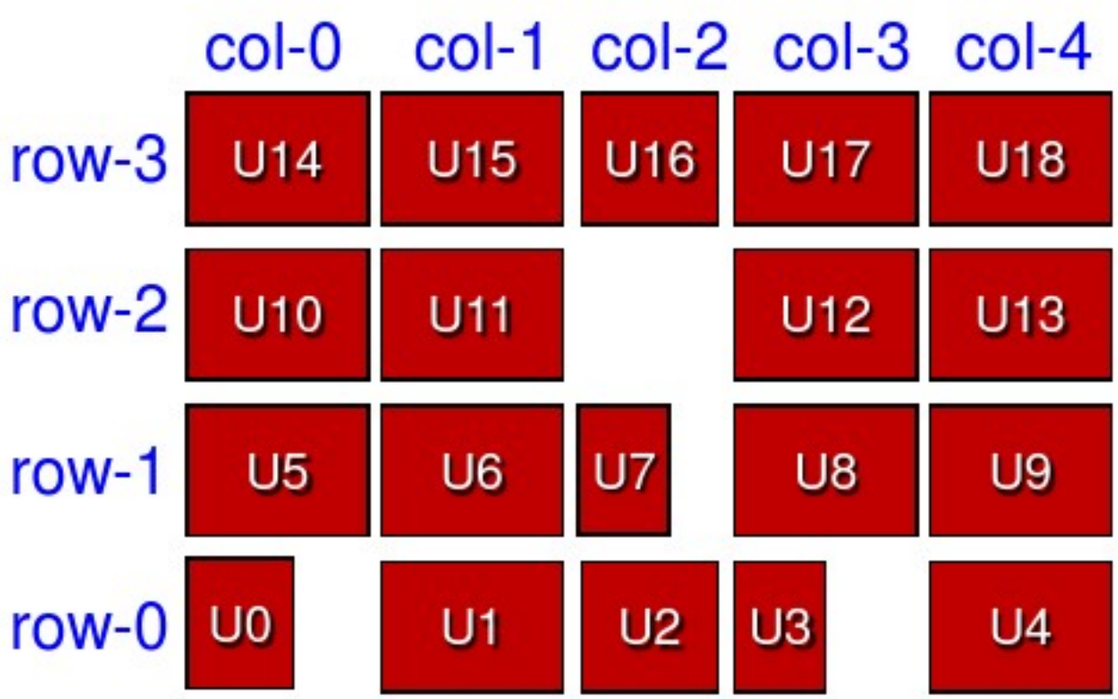
- **Traditionally one of these layout approaches is used:**
  - Creation of a full-custom data path macro or “hard IP”
  - Manual placement of the data path cells in the standard cell core area



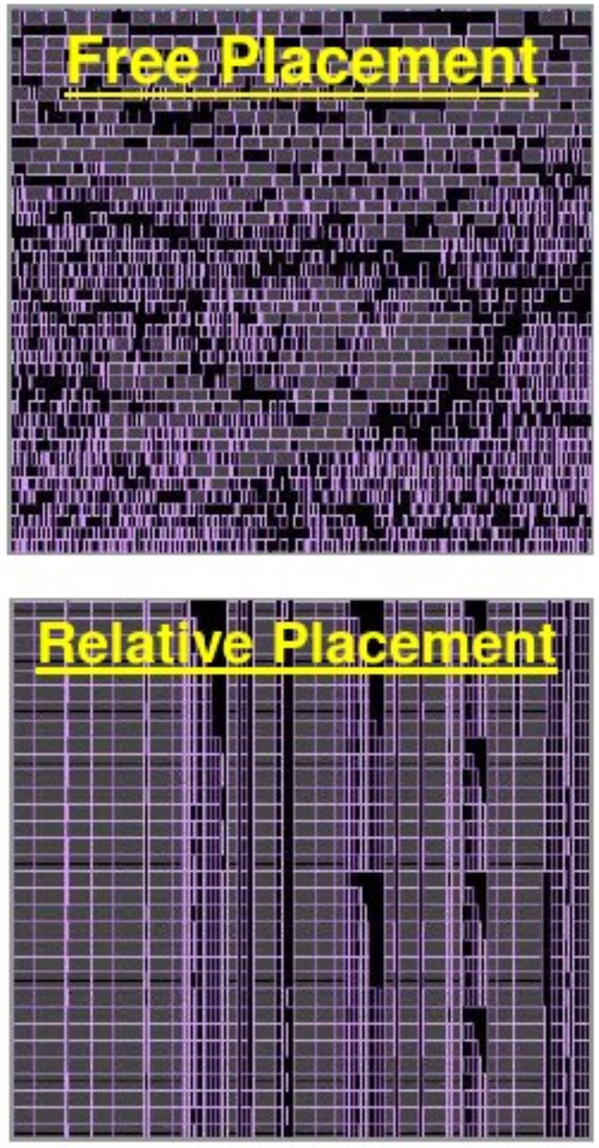
- **Downsides of manual/custom approaches:**
  - Very time-consuming - Increased design cycle
    - Manual gate size selection
    - Manual placement of hard IP or data path cells in the core area
    - Manual/limited logic optimization for timing, power, area, DRCs
    - Re-creation of hard IP and manual re-placement after logic change
  - Increased delay, power and area due to limitation of manual logic optimization

IC Compiler's Solution: *Relative Placement*

- **Each data path functional module is defined once as a *relative placement (RP) group***
- **Each cell of an *RP group* is assigned a row and column**
  - Also known as "tiling"
  - Done prior to placement through TCL constraints



Example relative placement group of a data path functional module



Features and Benefits of *Relative Placement*

Features

- **RP groups are placed concurrently with standard cells**
  - Relative placement is maintained through post-route
- **RP logic is optimized as needed during placement through post-routing phases**
  - RP groups are re-sized to accommodate cell-sizing
  - Placement of RP groups in core is automatically adjusted
- **Easy to use GUI (Graphical User Interface) for visualization and editing**

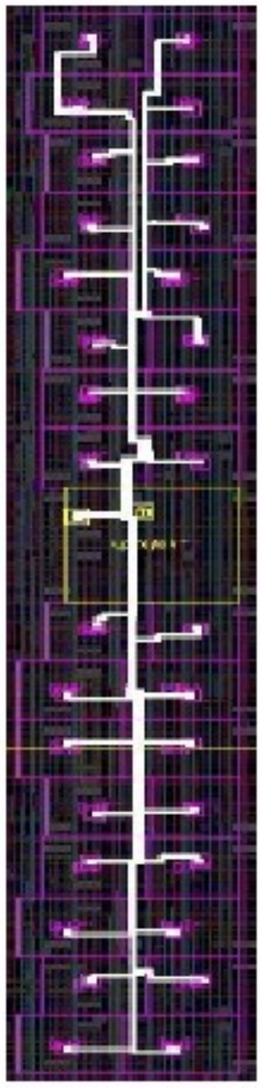
Benefits

- **Improved congestion, skew, timing, power and area**
- **Increased design predictability**
- **Shorter design cycle**

Candidates for *Relative Placement*

- **The following functions are good candidates for taking advantage of *Relative Placement***
  - Classic data path structured elements
  - RAMs, FIFOs
  - Clock structures
  - Register banks

- **Some key design applications include**
  - Processors, DSPs
  - Graphics
  - IP providers



Relative placement of RAMs with clock “pin alignment”
→ Reduced clock skew and power

More Information on *Relative Placement*

- IC Compiler User Guide on SolvNet
- White Paper: Physical Datapath – Improved Productivity for All Designs
  http://www.synopsys.com/apps/protected/docs/pdfs/iccwp/icc_datapath_wp.pdf
- SNUG 2008

  Structured Methods for Delay, Power Tuning and Variation: A Case Study

  https://www.synopsys.com/news/pubs/snug/sanjose08/binney_pres.pdf
- SNUG 2007

  Efficient Physical DataPath Specification: Streaming Relative Placement

  https://www.synopsys.com/news/pubs/snug/sanjose07/dunham_pres.pdf

  Design of a 1GHz DSP using IC Compiler

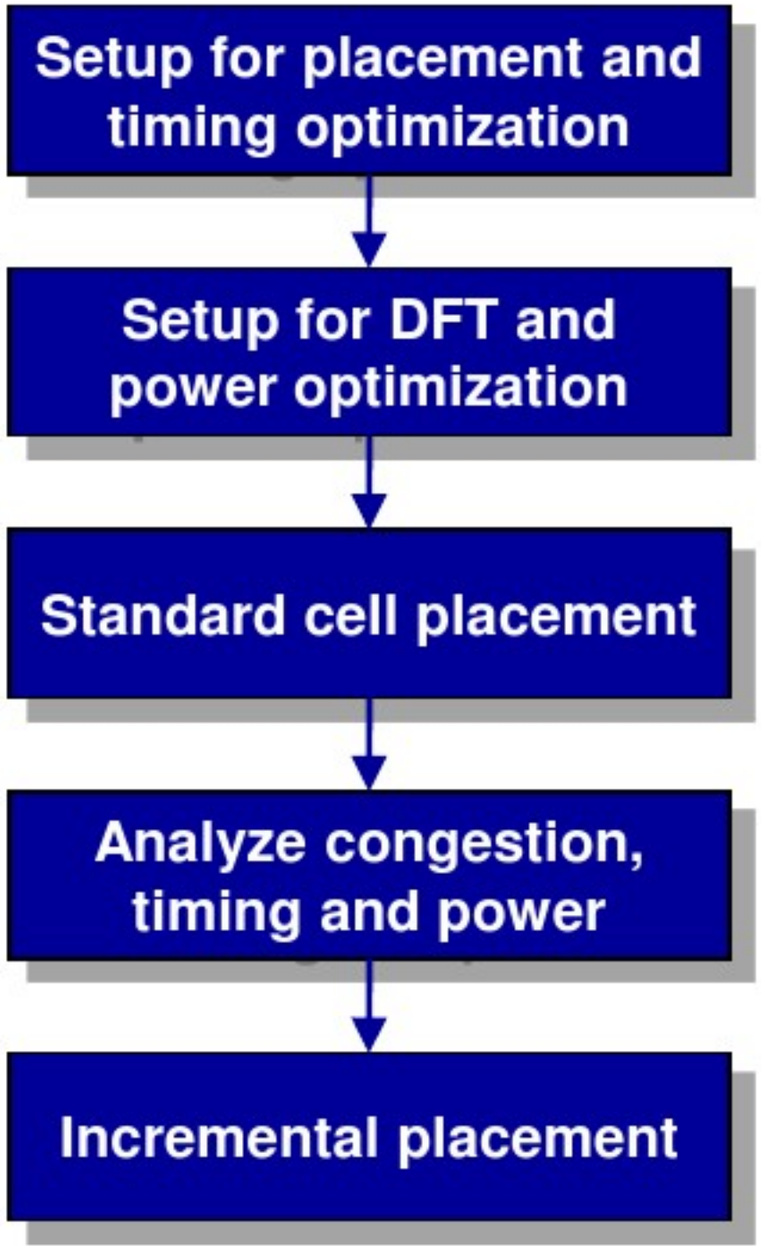
  http://www.synopsys.com/news/pubs/snug/sanjose07/hill_pres.pdf
- Contact your local Synopsys Applications Consultant for Relative Placement tutorial and labs

Summary

You should now be able to:

- Apply placement, DFT and power optimization settings before placement
- Perform placement and optimization
- Analyze congestion maps and reports
- Perform incremental congestion and timing optimization
- Perform additional placement techniques which allow more user-control

Lab 3: Placement

70 minutes

Perform standard cell placement and optimization with the goal of meeting the timing targets and eliminating congestion.