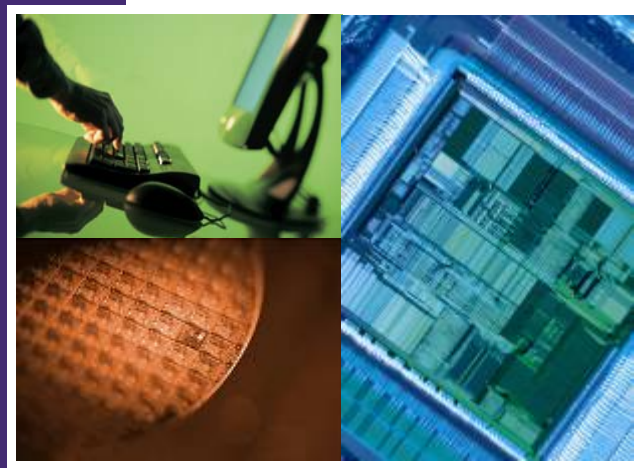
How (and Why!) To Use PrimeTime's Distributed Multi-Scenario Analysis

Chris Papademetrious
PrimeTime CAE



SNUG SAN JOSE 2007

SYNOPSYS®
Predictable Success

About This Presentation

- This presentation explores:
  - …why people need multiple STA analyses
  - …how they are sometimes combined together
  - …the advantages and disadvantages of merging analyses together versus analyzing them separately
  - …how PrimeTime's DMSA can help you get the most accurate analysis, in less time, with less user effort, and fewer resources
- This presentation does not:
  - …cover slide after slide of boring DMSA command syntax without teaching you why the feature is worth learning in the first place!
    - Your local Synopsys AC would be happy to give that ☺

Agenda

- **Where Multiple Analyses Come From**
- Combining Modes
  - Introduction
  - Accuracy Impact
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - Example: Hold-Fixing
- Summary

Where Multiple Analyses Come From

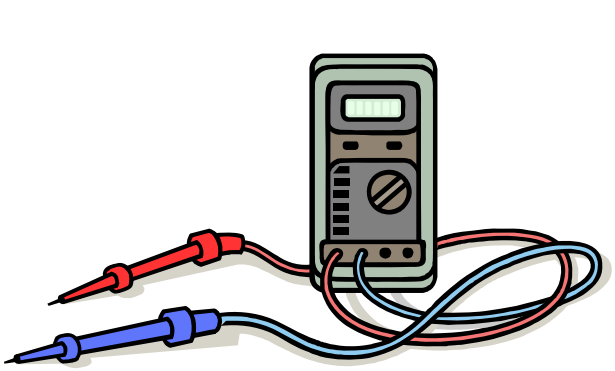
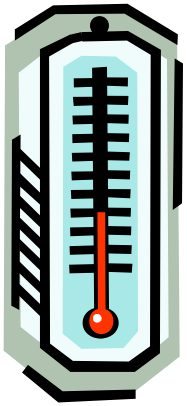
- The two primary causes of multiple analyses:
  - multiple operating *modes*
  - multiple analysis *corners*
- What are they?

Where Multiple Analyses Come From

- A design typically has multiple operating *modes*
  - Multiple mission modes
    - Different clock frequencies
    - Different clock muxing configurations
    - Different configuration constants to control logic behavior
  - Multiple test modes
    - ATPG scan shift
    - ATPG scan capture
    - BIST modes
    - JTAG

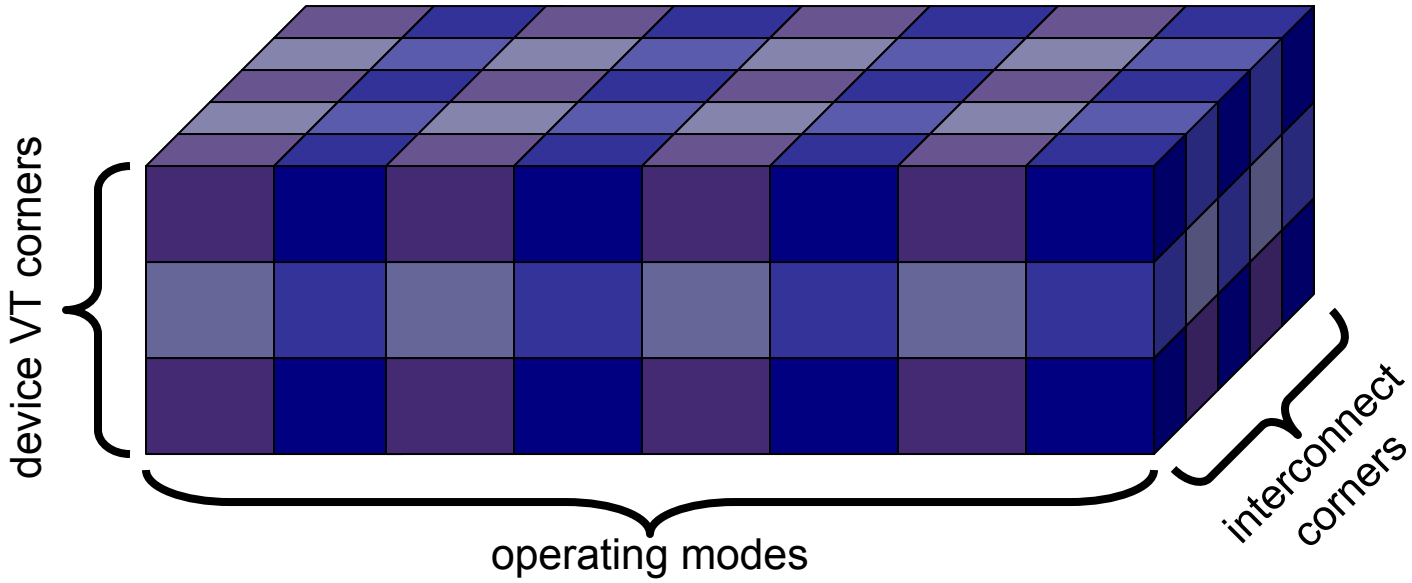
Where Multiple Analyses Come From

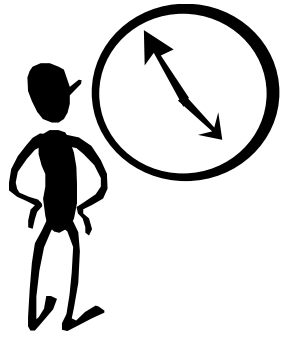
- A design must reliably operate across:
  - its specified range of voltage/temperature (VT) conditions: the chip's environment
  - the range of possible process/interconnect (P) conditions: the chip's manufacturing characteristics
- A specific set of conditions (environmental VT and manufacturing P) is called a *corner*


Where Multiple Analyses Come From

- STA must pass in every operating *mode* under every PVT *corner* condition
  - These terms multiply:





- That's a lot of analyses!

Simplifying the Problem

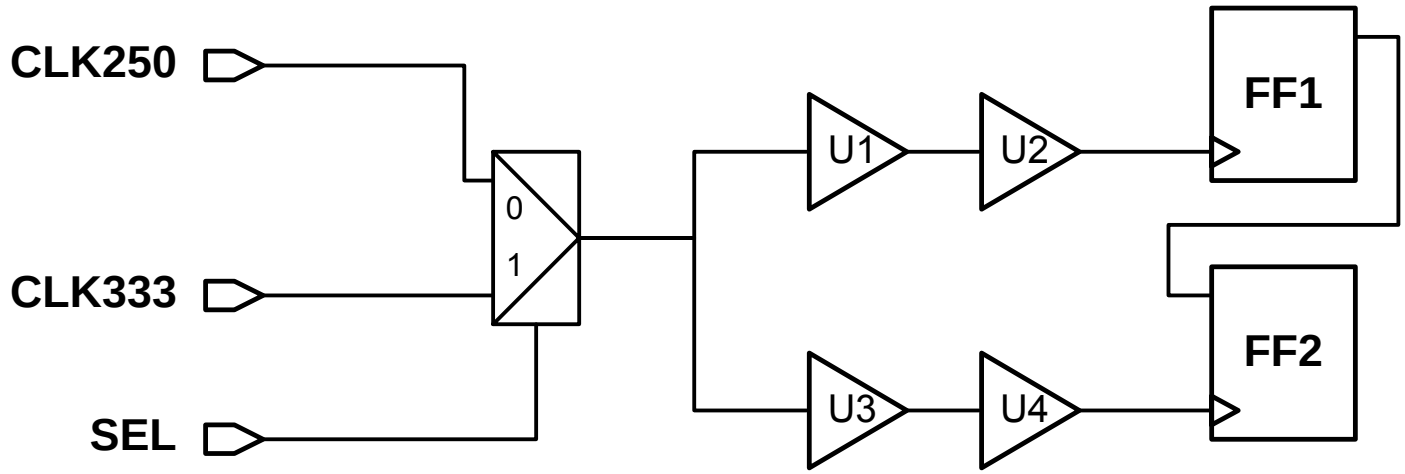
- Designers sometimes try to reduce the number of runs using various techniques:
  - Skipping certain mode/corner combinations entirely
  - Limiting corner combinations and adding more margin
  - Combining similar operating modes together
- The first two are simply omissions of *corner* analyses based on engineering judgment
- The third - whether to combine operating *modes* together or keep them separate - is our focus
  - Let's talk more about this…

Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - **Introduction**
  - Accuracy Impact
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - Example: Hold-Fixing
- Summary

Combining Modes: An Example

- Consider the following example circuit:



- CLK250 is selected in mode 0, and CLK333 is selected in mode 1

Combining Modes: An Example

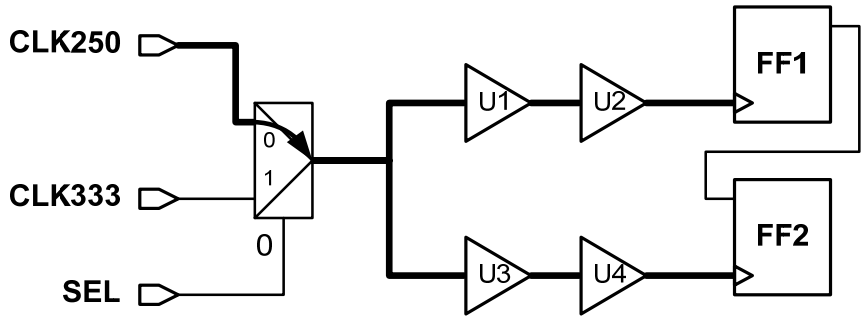
- Analyzed separately, case analysis is used to select the appropriate clock for each mode

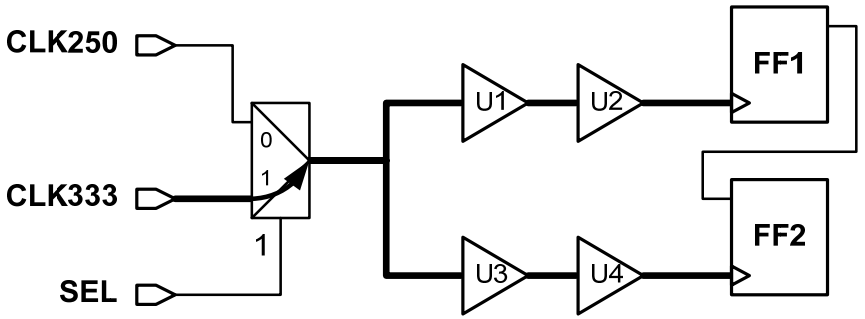
mode 0:

```
create_clock -period 4.00 CLK250
create_clock -period 3.00 CLK333
set_case_analysis 0 SEL
```

mode 1:

```
create_clock -period 4.00 CLK250
create_clock -period 3.00 CLK333
set_case_analysis 1 SEL
```



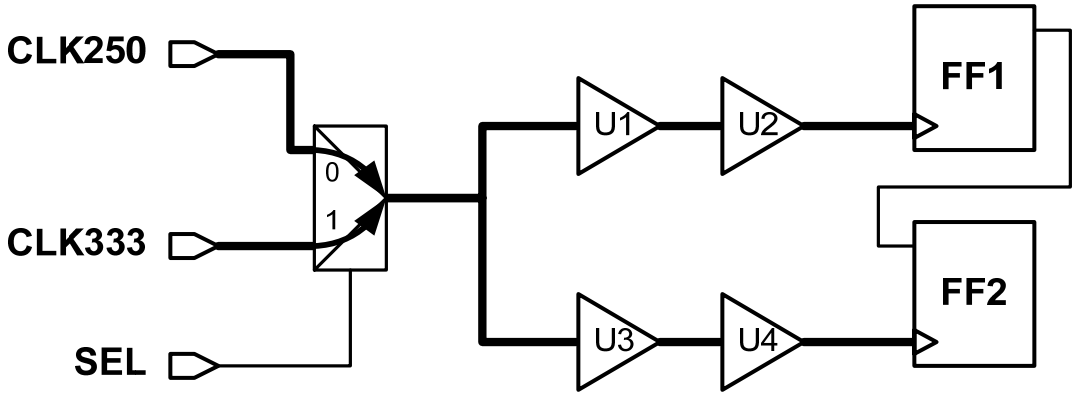


Combining Modes: An Example

- Analyzed together, we could combine these modes together with the following commands:
  - Need **`set_clock_groups`** to prevent CLK250↔CLK333 paths

modes 0&1 combined:

```
create_clock -period 4.00 CLK250
create_clock -period 3.00 CLK333
set_clock_groups -logically_exclusive -group CLK250 -group CLK333
```



Combining Modes: Limitations

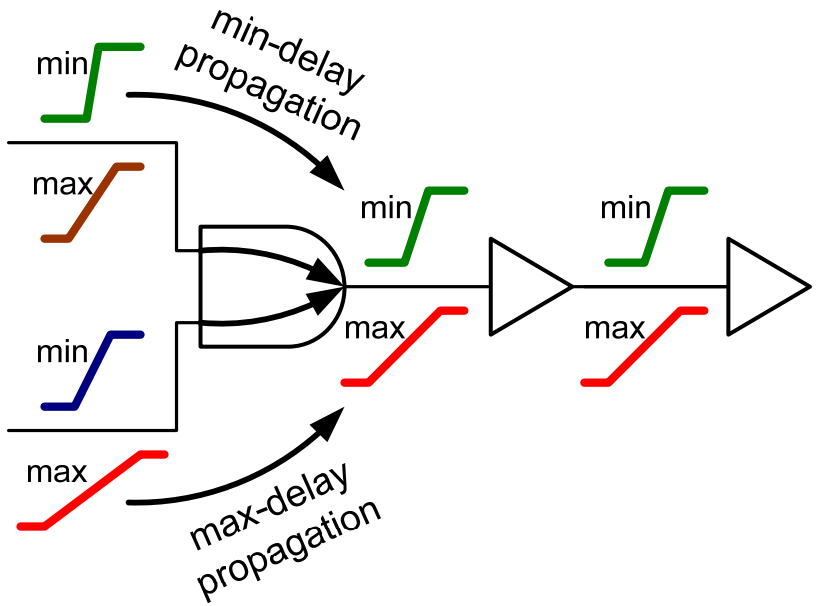
- On the surface, it seems like combining modes is the way to go!
  - Fewer runs to get the same amount of work done
- However, there are drawbacks to combining modes
  - Timing pessimism
  - Increased memory/runtime
  - Increased script complexity
- Let's examine each of these areas in more detail...

Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - **Timing Pessimism**
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - Example: Hold-Fixing
- Summary

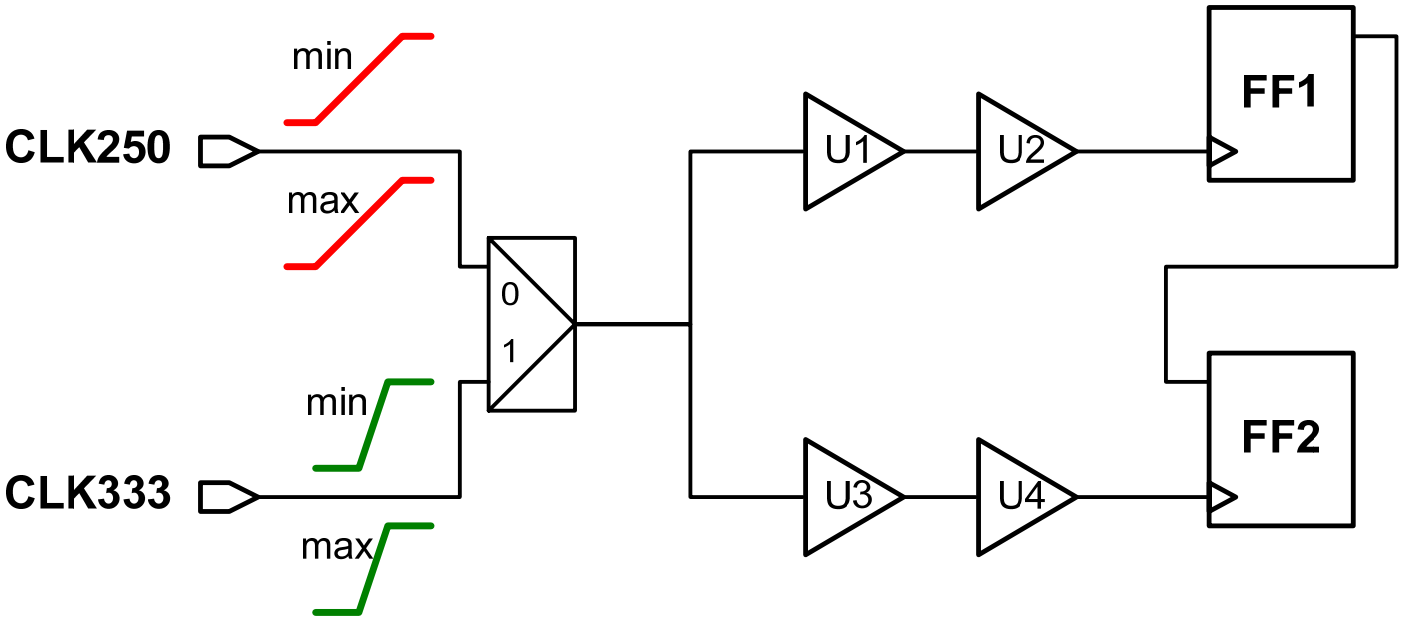
Combining Modes: Slew Propagation Pessimism

- PrimeTime computes both min and max timing for every timing arc in the design
  - Fast slews always propagated for min-delay computation
  - Slow slews always propagated for max-delay computation



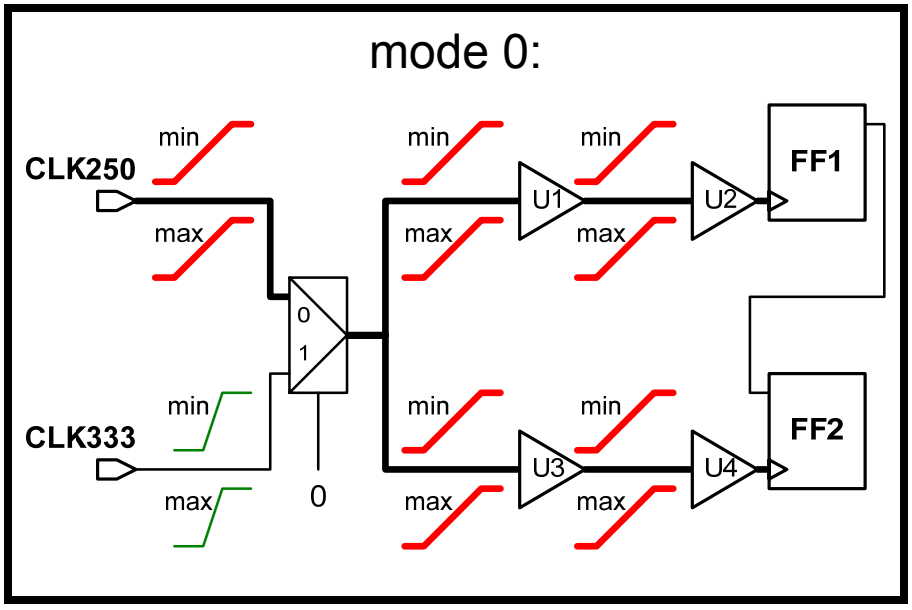
Combining Modes: Slew Propagation Pessimism

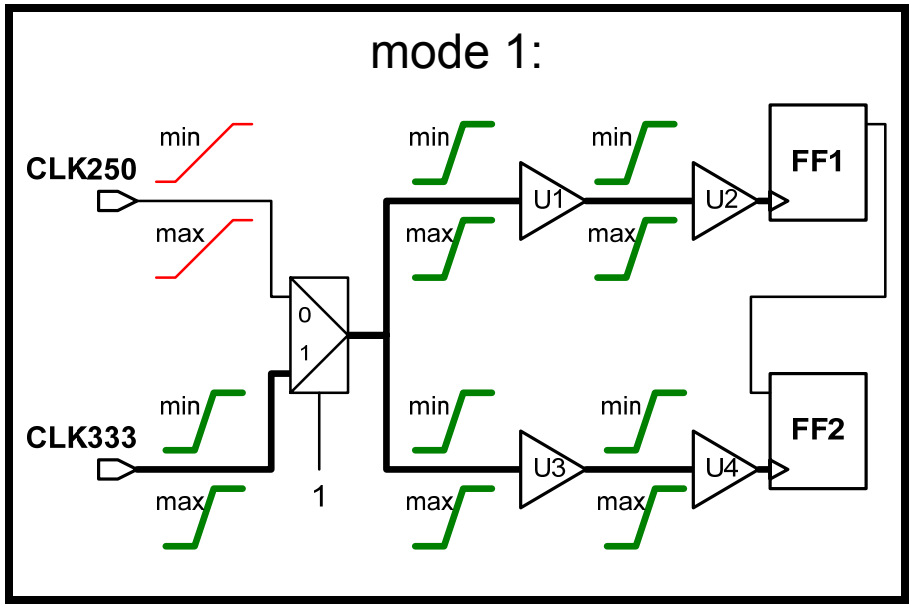
- In the example circuit below:
  - CLK250 propagates a slow slew (red) into the clock network
  - CLK333 propagates a fast slew (green) into the clock network



Combining Modes: Slew Propagation Pessimism

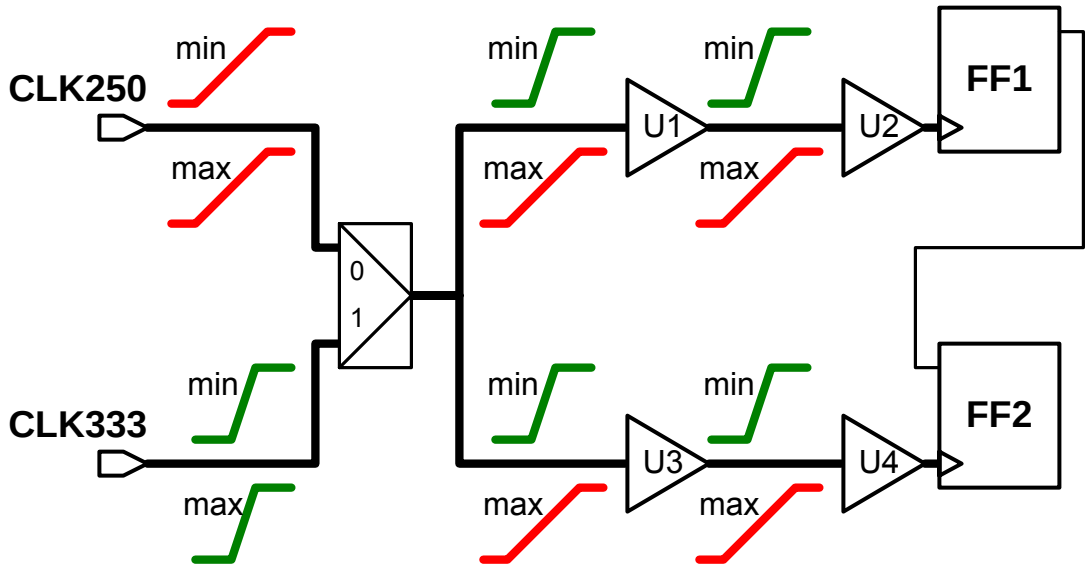
- When the modes are analyzed separately:
  - The clock buffers have slow min and max delays in mode 0
  - The clock buffers have fast min and max delays in mode 1





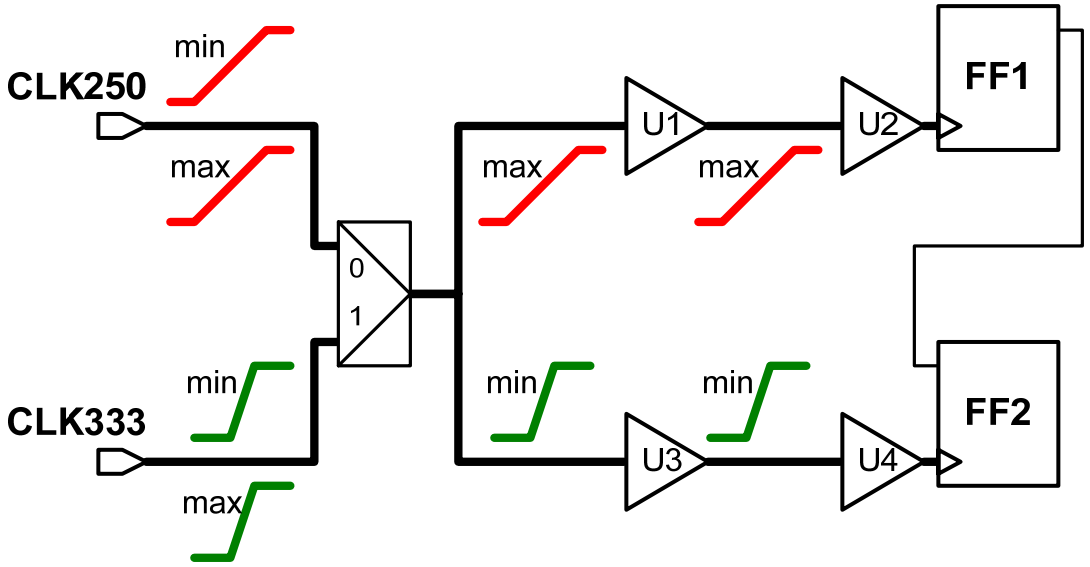
Combining Modes: Slew Propagation Pessimism

- However, when the modes are analyzed together, both clocks propagate into the network:
  - The slow CLK250 slew (red) is used for max-delay propagation
  - The fast CLK333 slew (green) is used for min-delay propagation



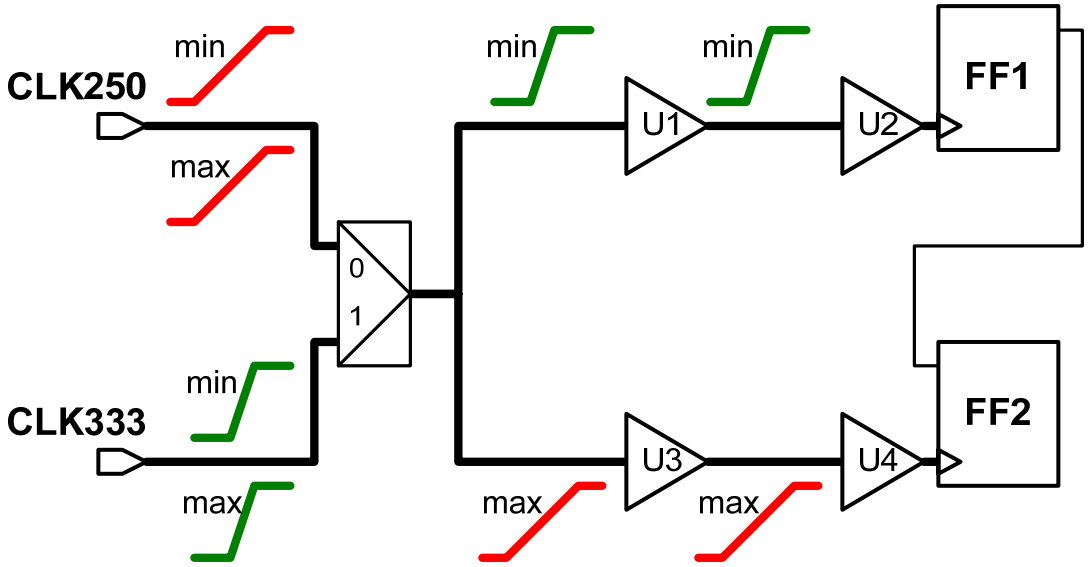
Combining Modes: Slew Propagation Pessimism

- Setup paths use max-delay launch, min-delay capture
  - For setup path in CLK250 domain, capture is pessimistic
  - For setup path in CLK333 domain, launch is pessimistic



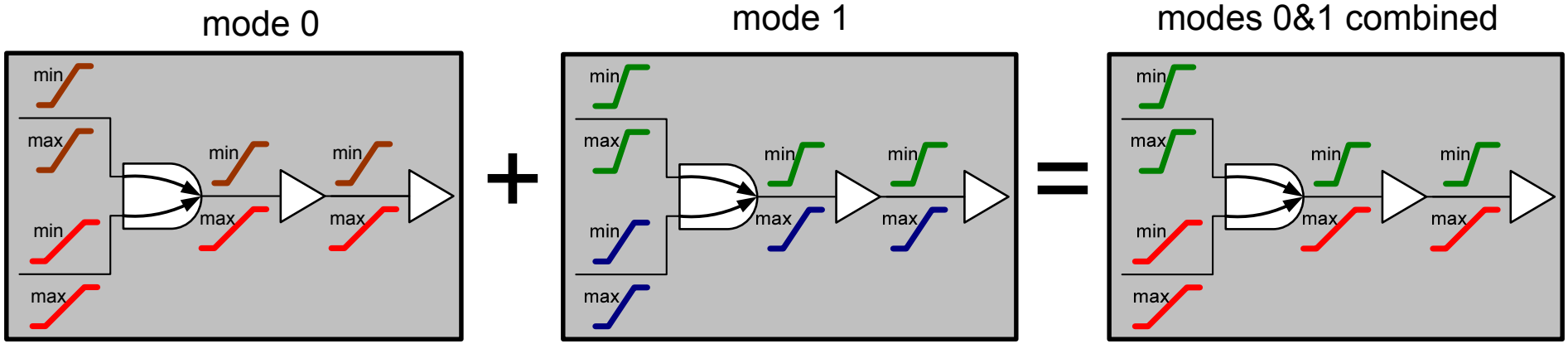
Combining Modes: Slew Propagation Pessimism

- Hold paths use min-delay launch, max-delay capture
  - For hold path in CLK250 domain, launch is pessimistic
  - For hold path in CLK333 domain, capture is pessimistic



Slew Propagation Pessimism: What Have We Learned?

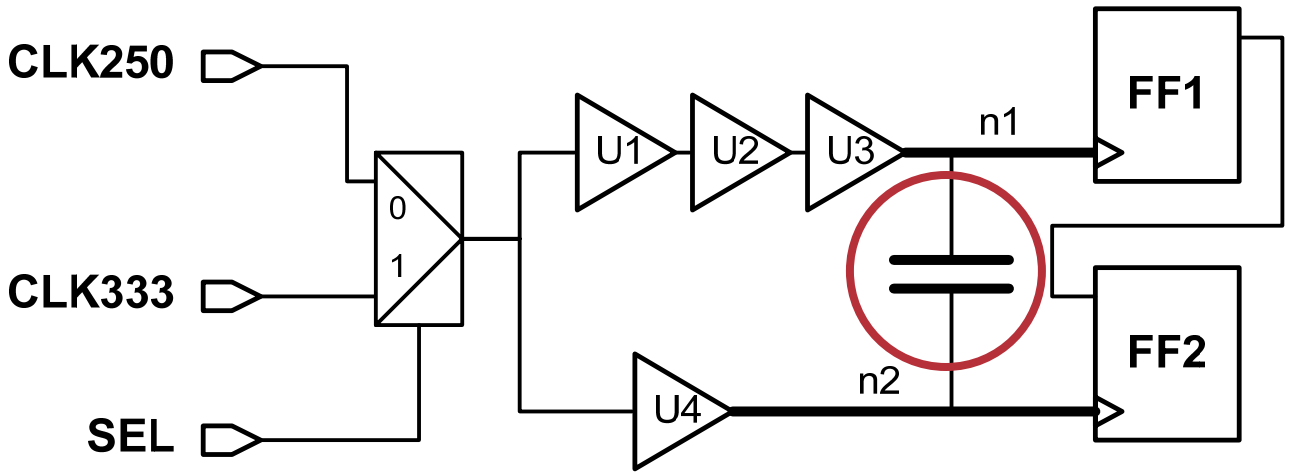
- When multiple operating modes are collapsed together, a loss of information (and accuracy) must occur
  - A timing arc can only have a single set of min/max timing behaviors
  - The only safe behavior is to keep the most pessimistic min/max timing across all modes, and then use that timing for every mode



- Analyzed separately, each operating mode has its own unique timing
  - Every analysis is accurate

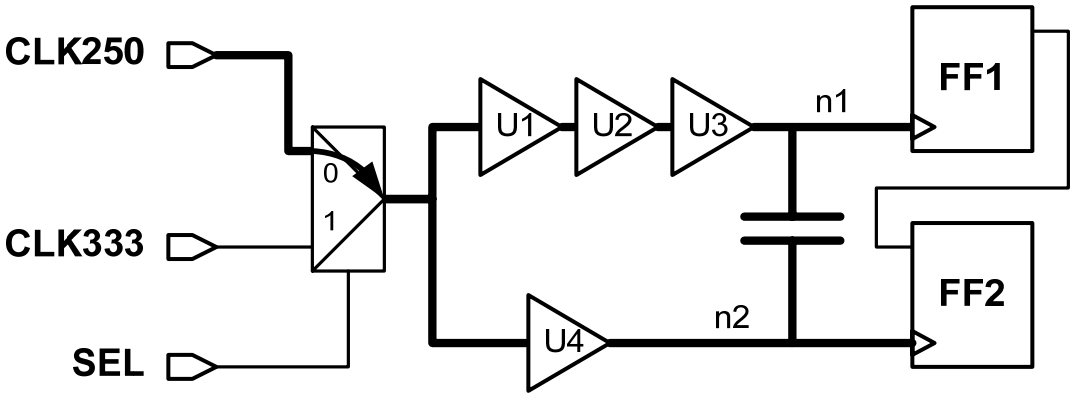
Combining Modes: Cross-Clock SI Interactions

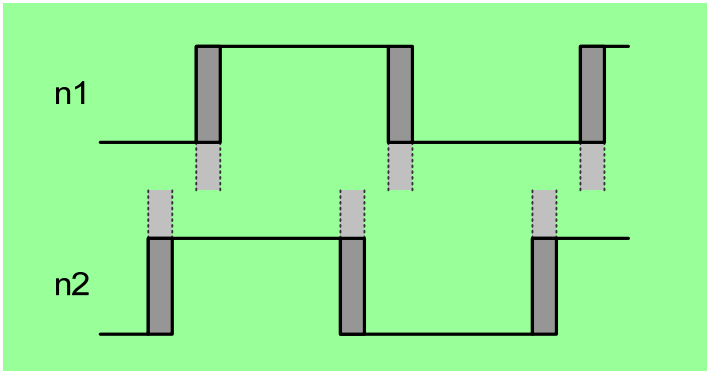
- When we bring SI into the picture, things get more complicated
  - Let's add some coupling and see what happens...



Combining Modes: Cross-Clock SI Interactions

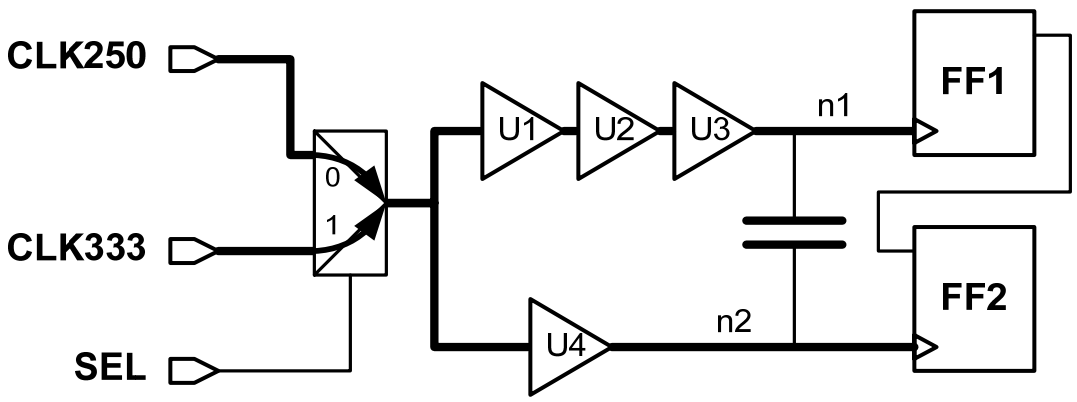
- When the modes are analyzed separately (mode 0 shown):
  - Only a single frequency clock is present on n1 and n2
  - The edge on n1 is always later than the edge on n2
  - Overlap can never occur (regardless of this single clock's frequency) so no delta delays occur on either net

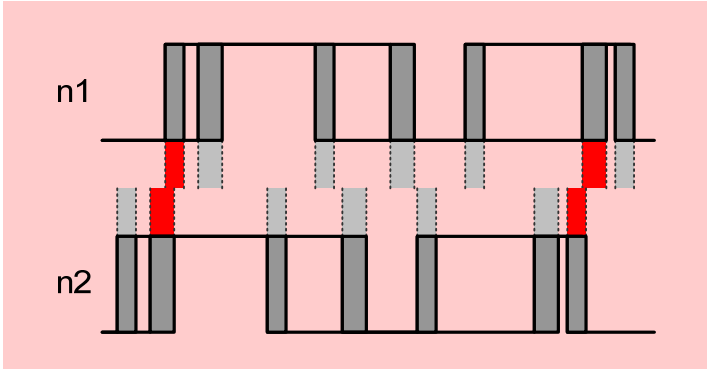




Combining Modes: Cross-Clock SI Interactions

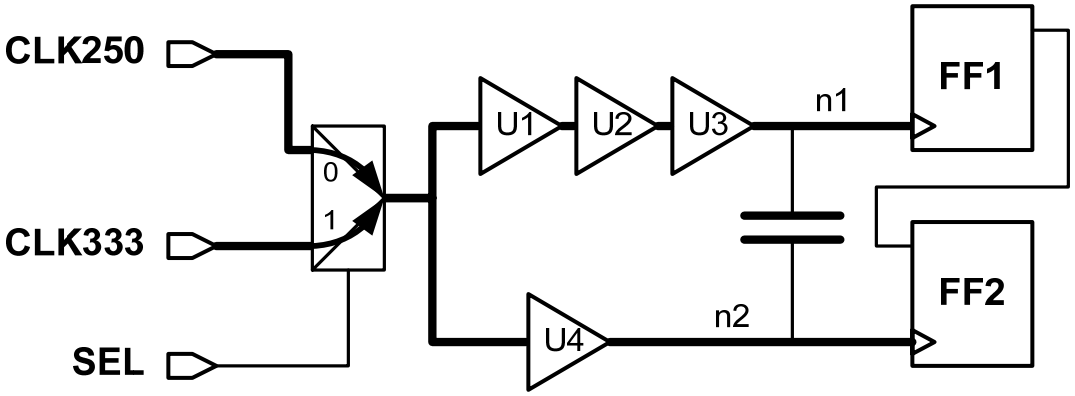
- When the modes are analyzed together:
  - Both clocks are seen on both nets
  - Crosstalk interactions between the clocks are probable
    - In fact, they are a certainty if the clocks are asynchronous

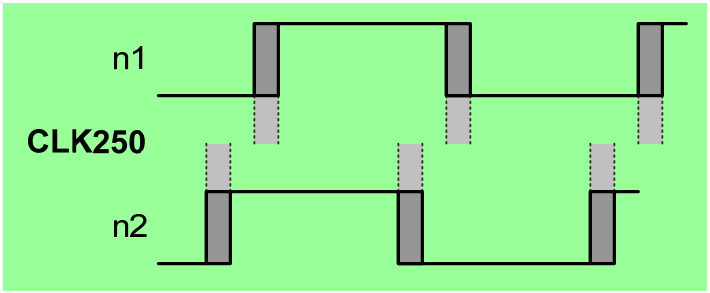


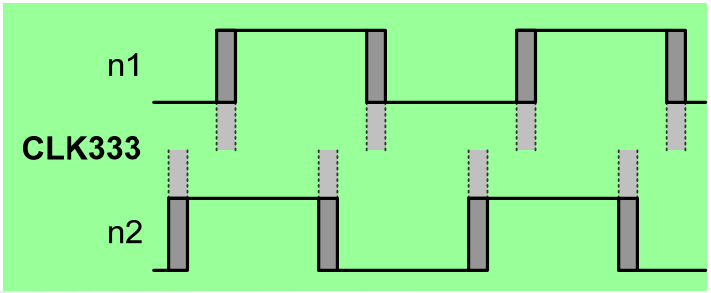


Combining Modes: Cross-Clock SI Interactions

- To handle this, clocks can be made *physically exclusive*
  - Interactions between physically exclusive clocks are not considered
  - We will revisit the concept of physically exclusive clocks a little later

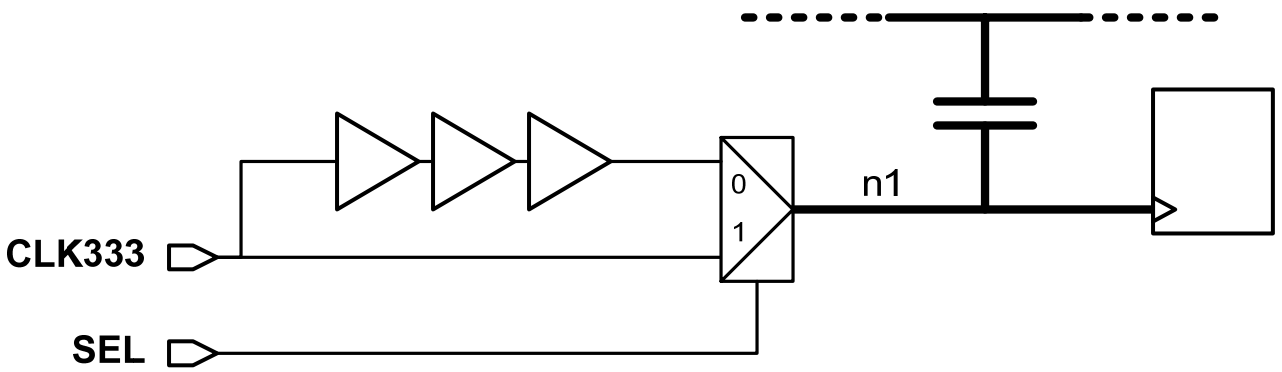






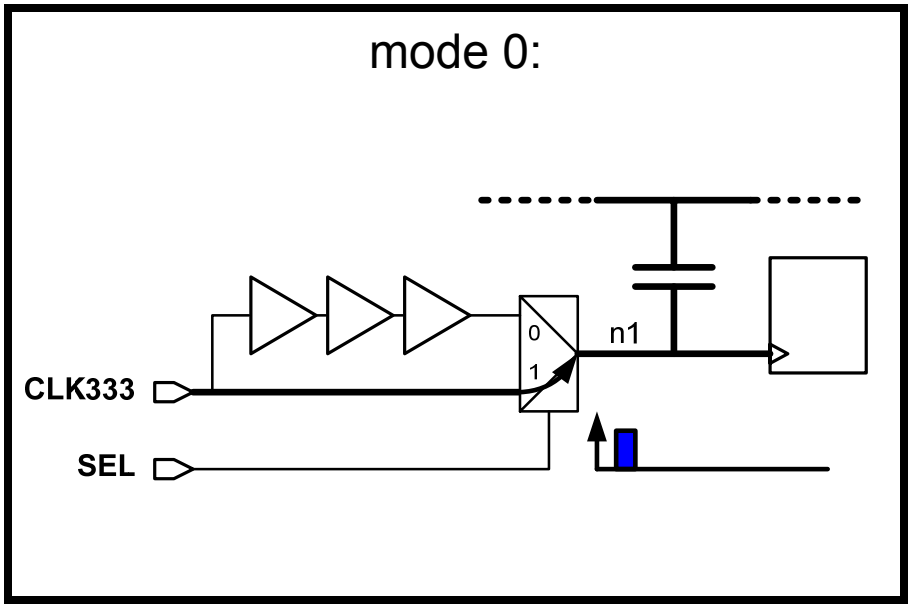
Combining Modes: Window Merging Within Clock Domain

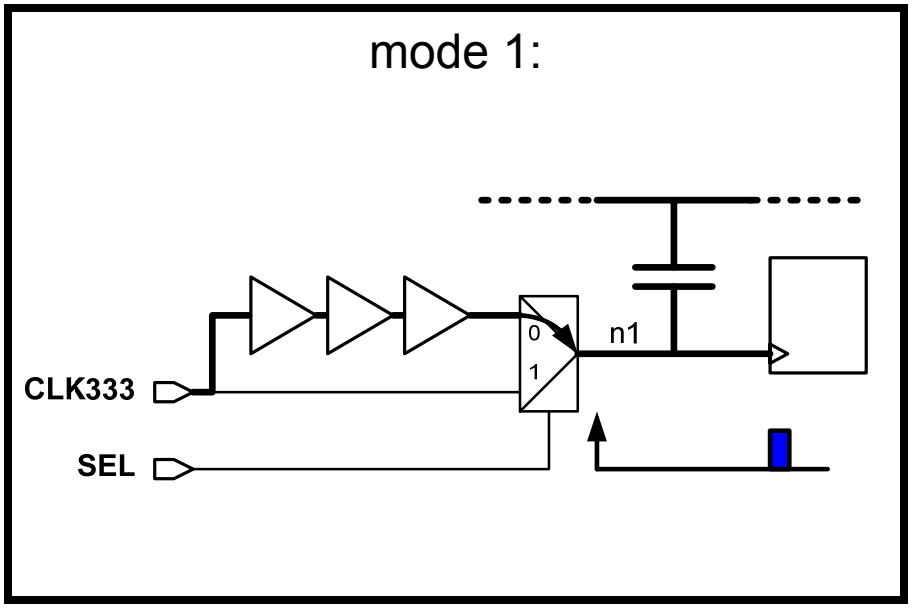
- Combining modes can cause pessimism even within a single clock domain
- In the example circuit below:
  - We select between the normal and delayed version of a clock
  - The clock net couples to some other net in the design



Combining Modes: Window Merging Within Clock Domain

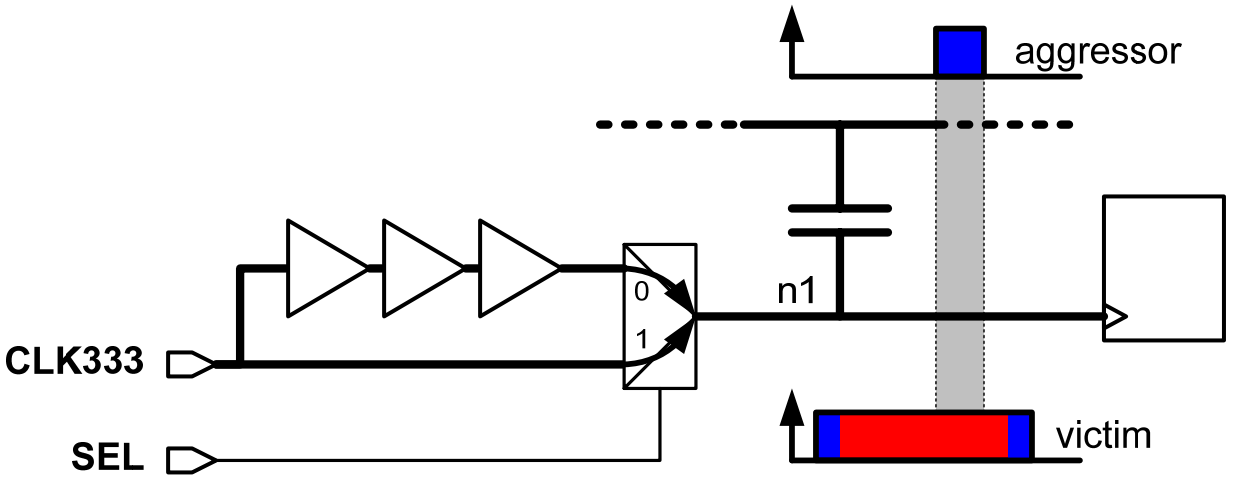
- When the modes are analyzed separately:
  - The non-delayed mode has an early arrival window on n1
  - The delayed mode has a late arrival window on n1
  - Both arrival windows are narrow





Combining Modes: Window Merging Within Clock Domain

- However, when the modes are analyzed together, both clocks propagate into the network:
  - All possible arrival windows for the clock are merged together
  - A single wide arrival window results
  - The middle section in red represents the pessimistic portion of the window which could not occur in either mode

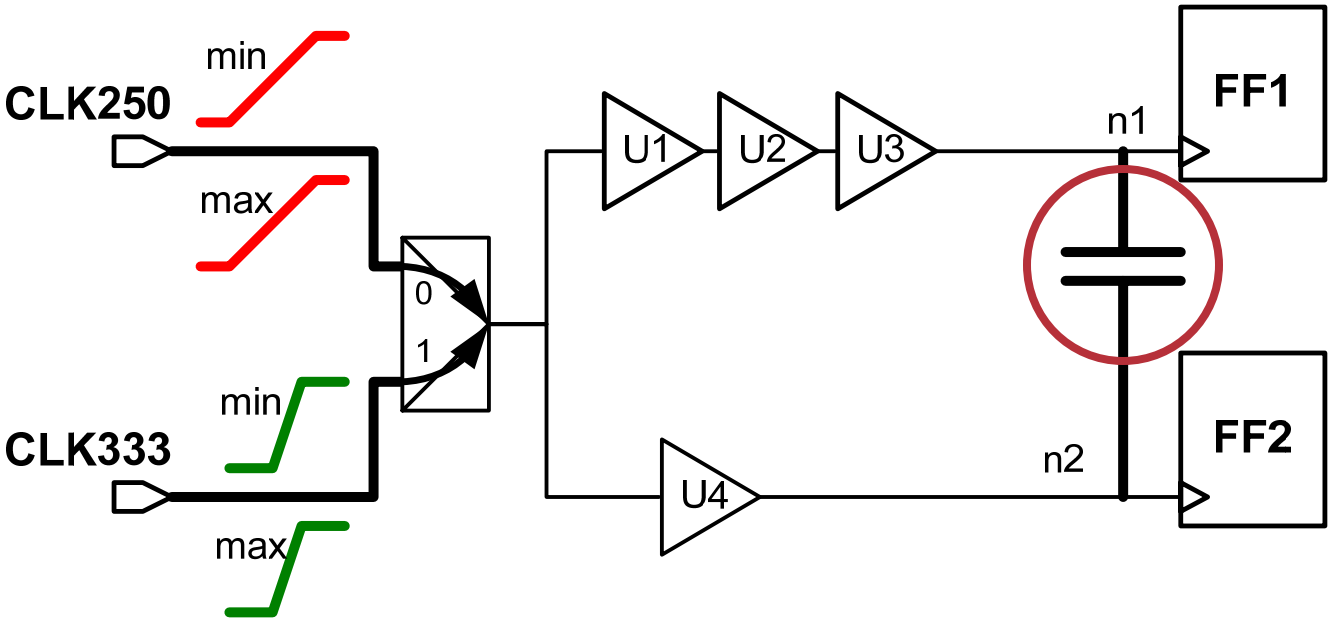


Combining Modes: Pessimism Effects Interact

- The different causes of pessimism…
  - slew merging pessimism
  - SI pessimism across clocks
  - window merging pessimism within clock domain
- …do interact with each other
- Let's take a look at some examples to see how these effects interact and compound

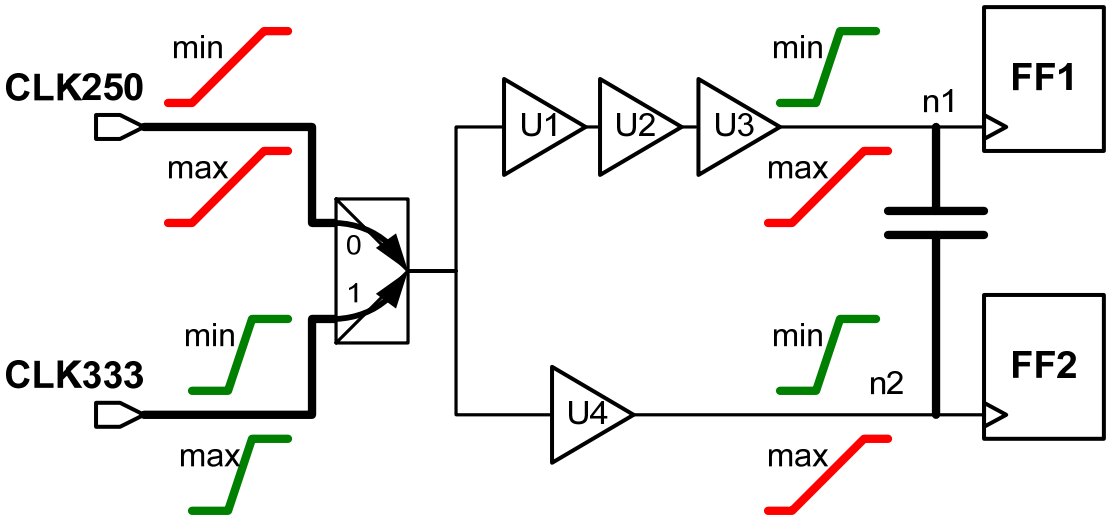
Pessimism Effects Interact: Example #1

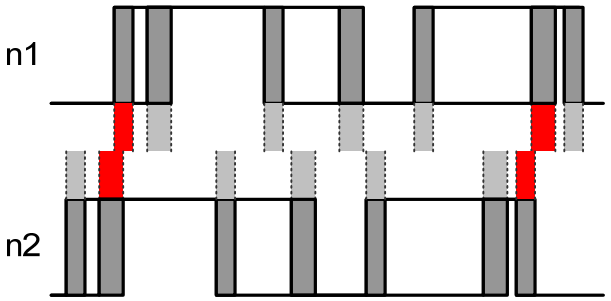
- In the example circuit below:
  - We have slew merging pessimism across modes
  - Nets n1 and n2 are coupled



Pessimism Effects Interact: Example #1

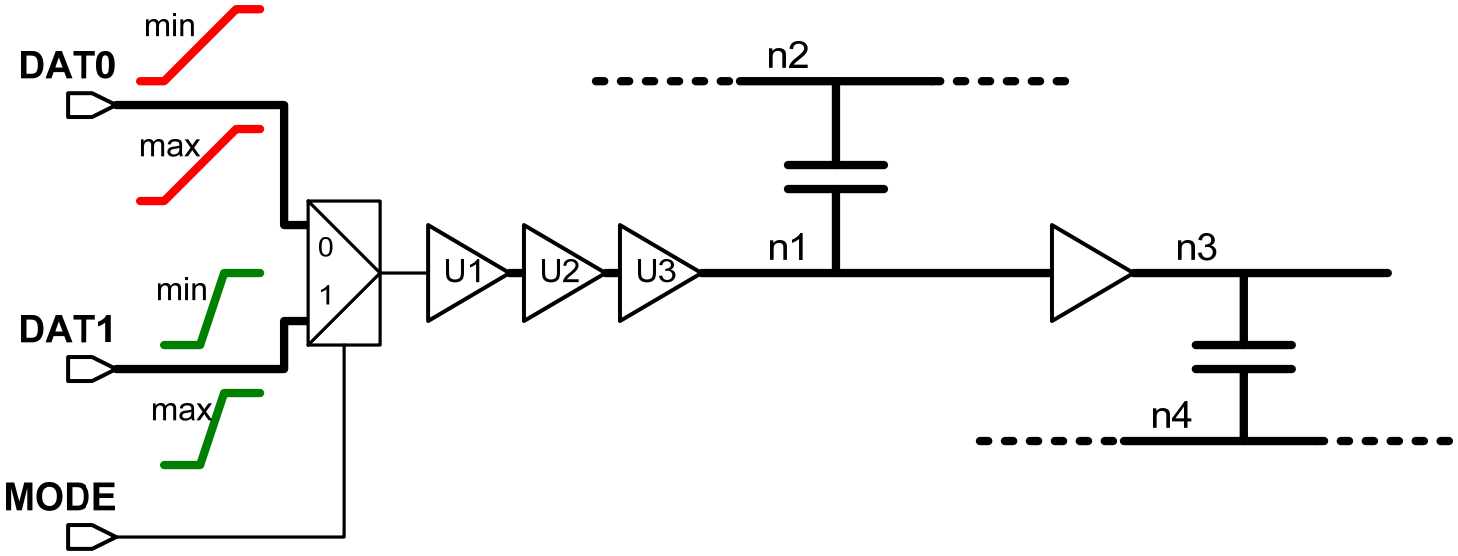
- When the modes are analyzed together:
  - Window alignment is found which isn't possible in either mode (due to the difference in buffer delays)
  - These victim/aggressor calculations also use a min/max slew combination which isn't possible in either mode





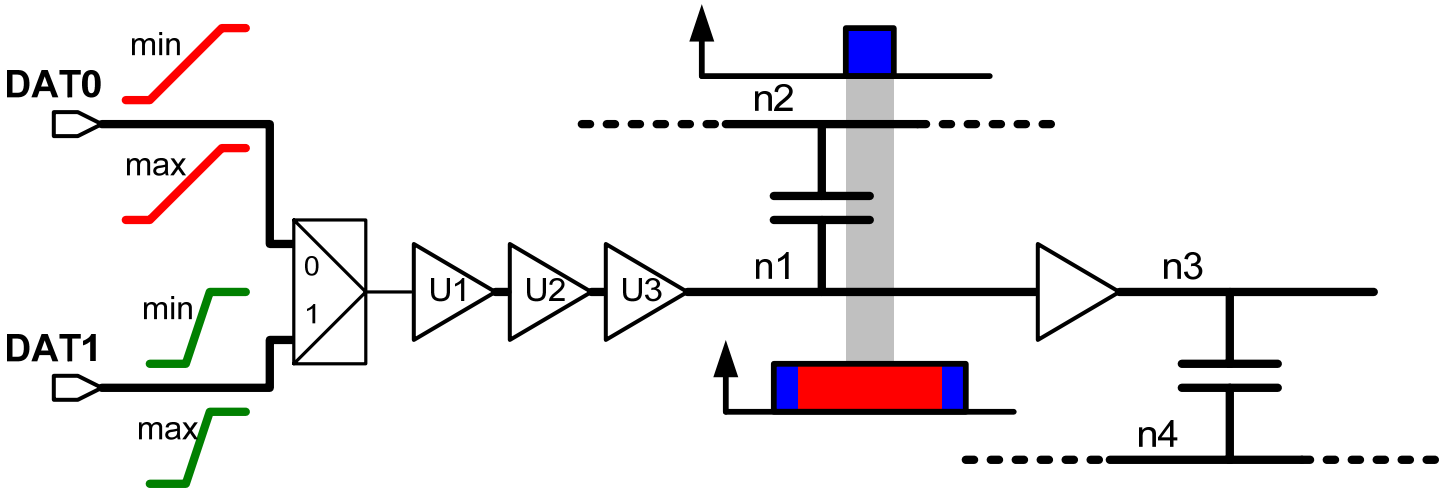
Pessimism Effects Interact: Example #2

- In the example circuit below:
  - We have slew merging pessimism across modes
  - There are multiple coupled stages downstream of the slew merge



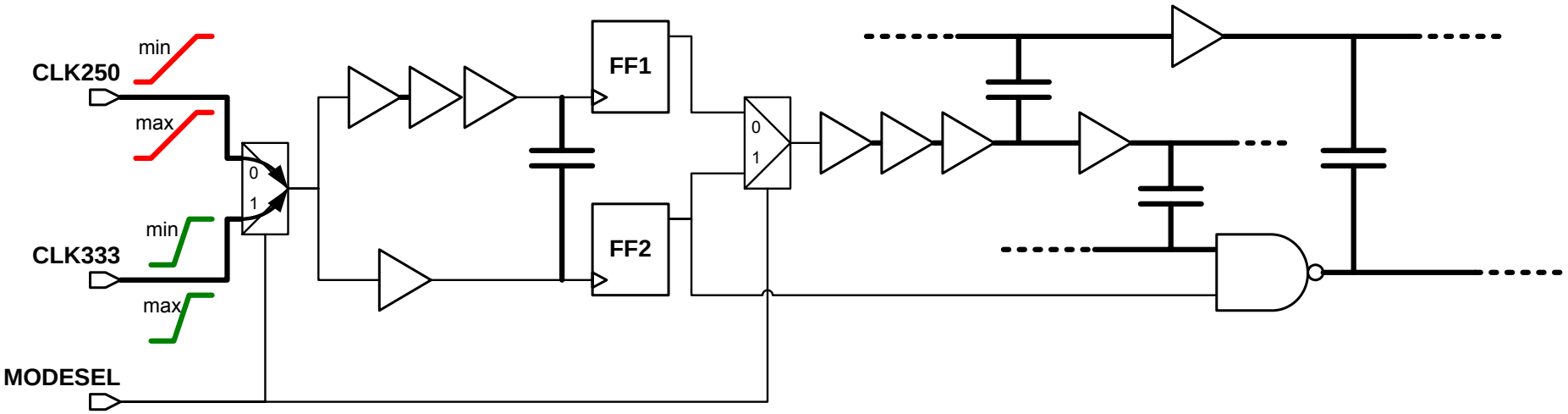
Pessimism Effects Interact: Example #2

- When the modes are analyzed together:
  - Slew merging pessimism causes the min/max timing of buffers U1-U3 to widen
  - n1 is attacked by n2 although it couldn't happen in either mode
  - Now the window/slews on n3 pessimistically attack n4…



Combining Modes: Pessimism Effects Interact

- And of course these effects continue to propagate…
  - Pessimistic slews yield pessimistic arrival/slew timing
  - These pessimistically-timed nets attack other victim nets
  - Those aggressions widen their downstream windows, worsen their downstream slews
  - The process continues… across both clock and data logic!



Pessimism From Combining Modes: What Have We Learned?

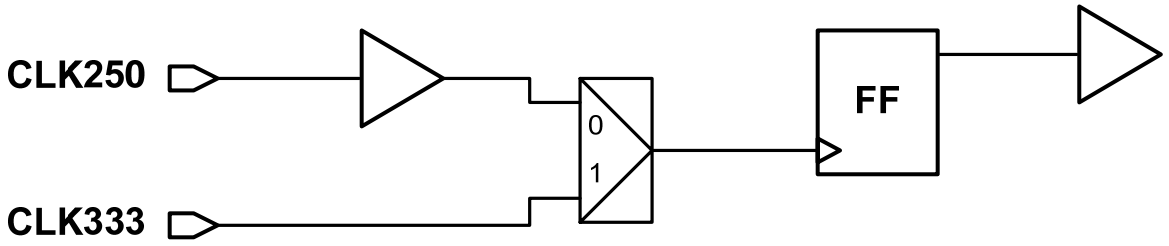
- When different modes are merged together:
  - Slew merging pessimism occurs
  - SI pessimism across clocks occurs
  - Window merging pessimism within clock domain occurs
  - These effects interact and compound
- The resulting merged-mode timing is more pessimistic than any of the original modes
  - The resulting analysis is safe but pessimistic
  - PrimeTime is doing the right thing, but with pessimistic input
  - We have indeed reduced the number of runs, but at an accuracy cost

Agenda

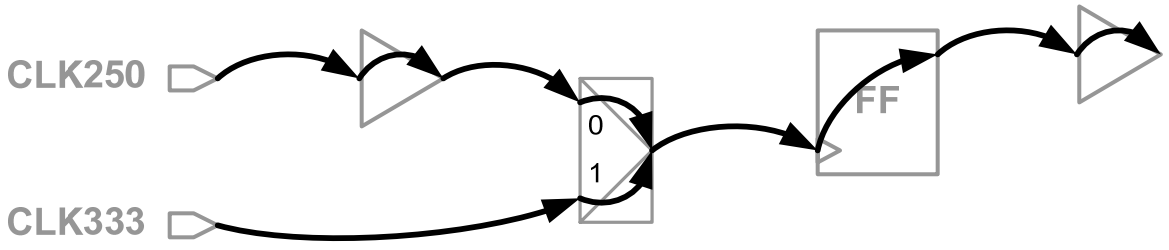
- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - Accuracy Impact
  - **Memory/Runtime Requirements**
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - Example: Hold-Fixing
- Summary

Combining Modes: Analysis Memory/Runtime

- When combining runs together, memory and runtime increase
- To understand why, let's look at how PrimeTime stores the timing data for the following example circuit:

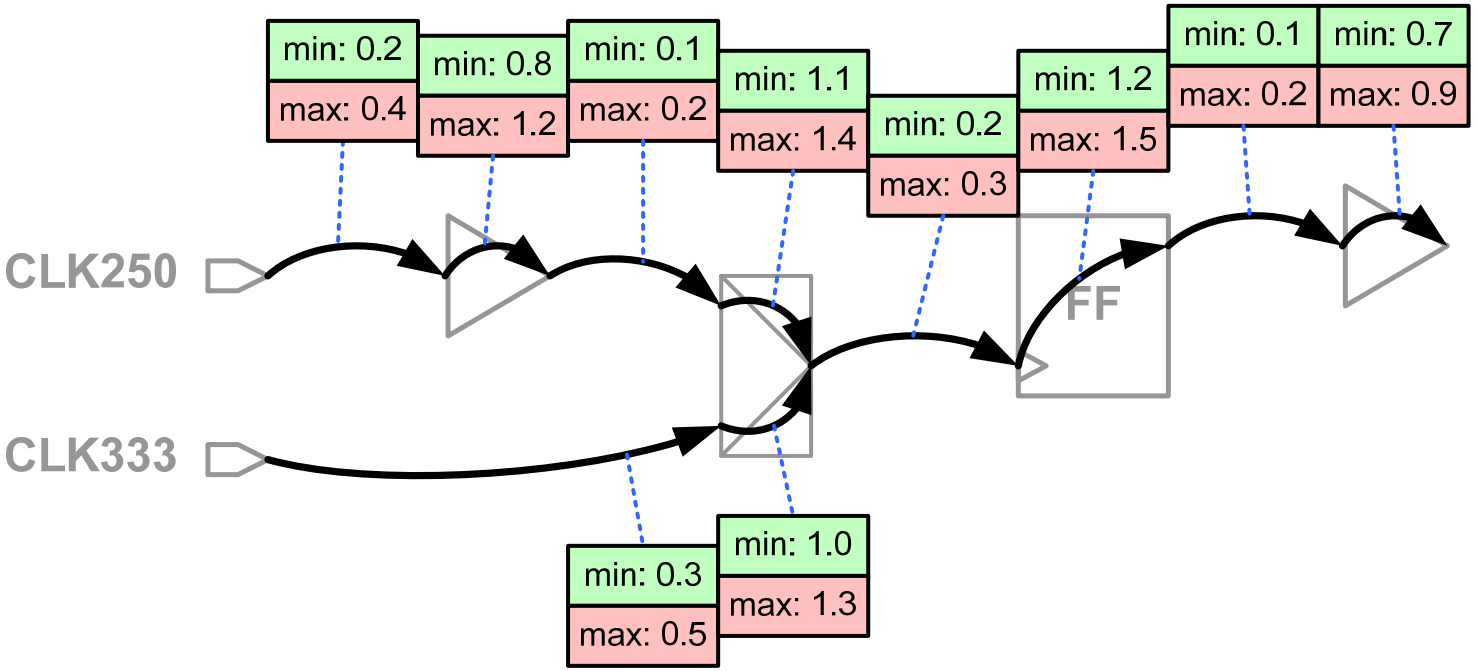


- Since PrimeTime deals with timing arcs, let's represent this circuit using timing arcs:



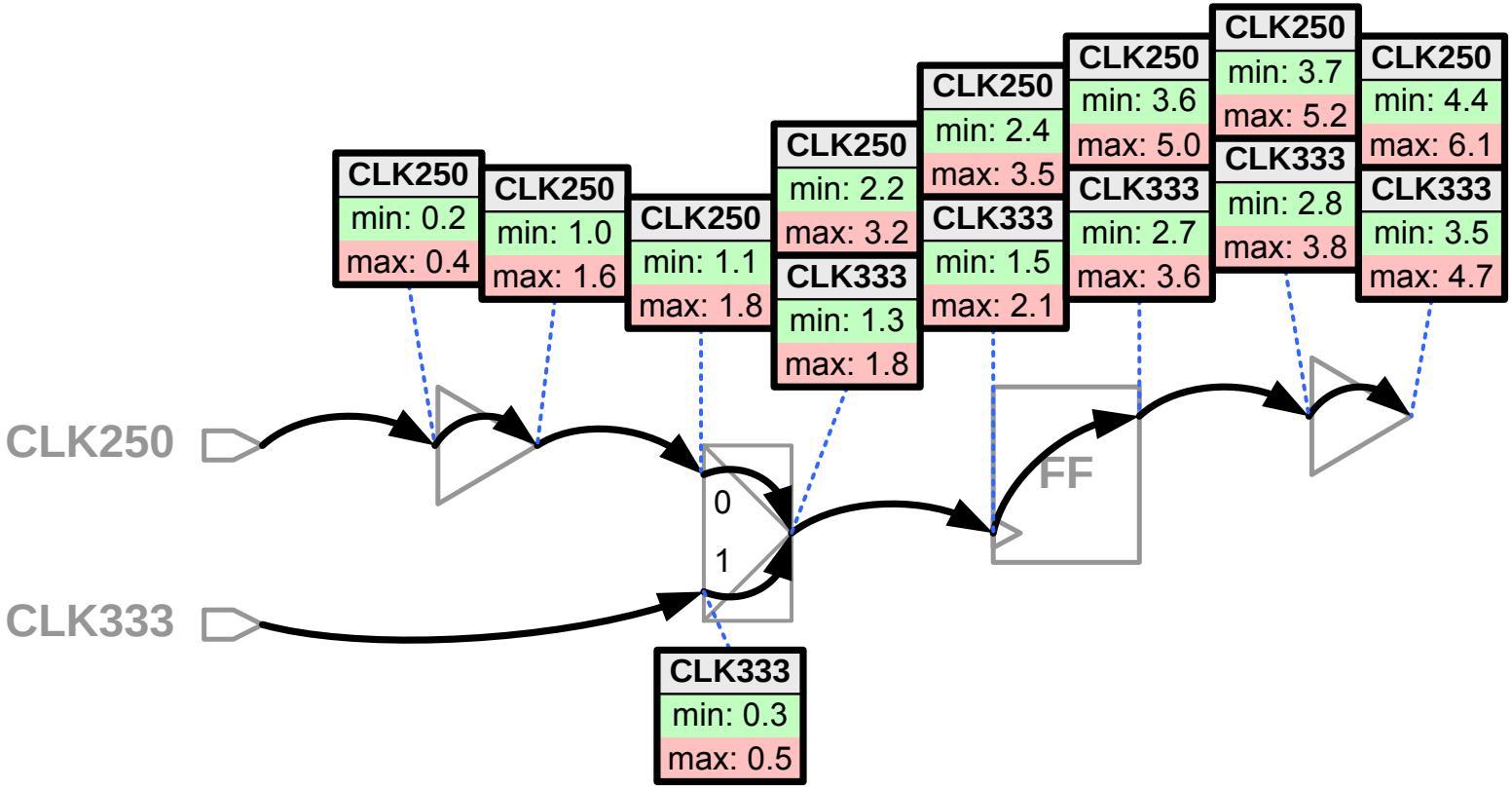
Combining Modes: Analysis Memory/Runtime

- PrimeTime computes the timing in two passes
  - Pass #1: timing arc min/max delays and slews are computed
    - Arc delays and slews are merged across multiple clock domains



Combining Modes: Analysis Memory/Runtime

- Pass #2: merged arc delays are used to propagate pin arrival times
  - Arrival times are computed per clock domain



Combining Modes: Analysis Memory/Runtime

- As the number of simultaneous clock domains increases:
  - The memory required to store all the arrivals increases
  - The runtime required to compute and propagate the delays/slews increases

Combining Modes: Analysis Memory/Runtime

- Below is some data from real customer designs
  - "Original Clocks" is the analysis with original clocks
  - "Duplicated Clocks" is the analysis with a second set of physically exclusive clocks with slightly different clock periods

	Original Clocks		Duplicated Clocks		% difference	
Design	CPU	memory	CPU	Memory	CPU	memory
1	973 s	826 MB	1276 s	998 MB	+31%	+21%
2	1373 s	1.28 GB	1483 s	1.47 GB	+8%	+15%
3	2832 s	4.40 GB	4136 s	6.40 GB	+46%	+45%
4	6051 s	3.11GB	7351 s	3.52 GB	+21%	+13%

Analysis Memory/Runtime: What Have We Learned?

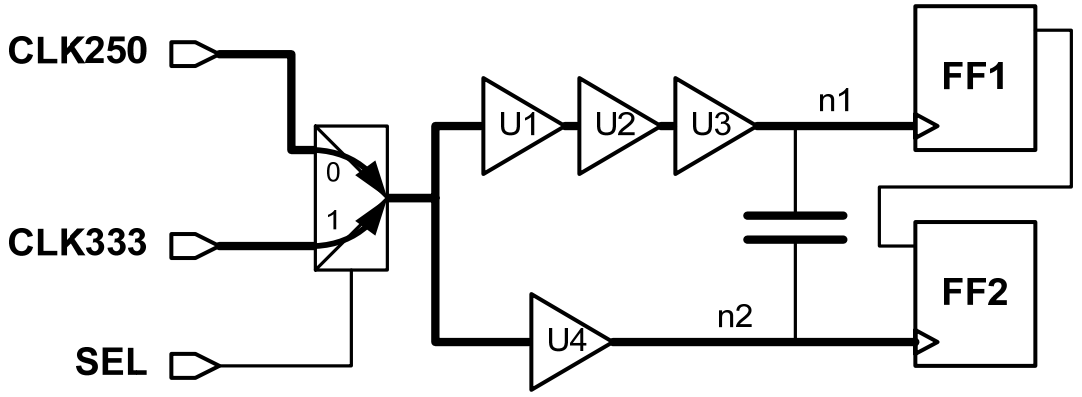
- As we combine modes and add more clocks:
  - More time is needed for extra delay calculations and clock propagation
  - More memory is needed to store each clock's arrivals
  - These per-clock arrivals are computed using the worst arc value across all clocks
- We have indeed reduced the number of runs, but at a memory, runtime and accuracy cost

Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - Accuracy Impact
  - Memory/Runtime Requirements
  - **Script Complexity**
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - Example: Hold-Fixing
- Summary

Combining Modes: Script Complexity

- Let's revisit this example circuit:



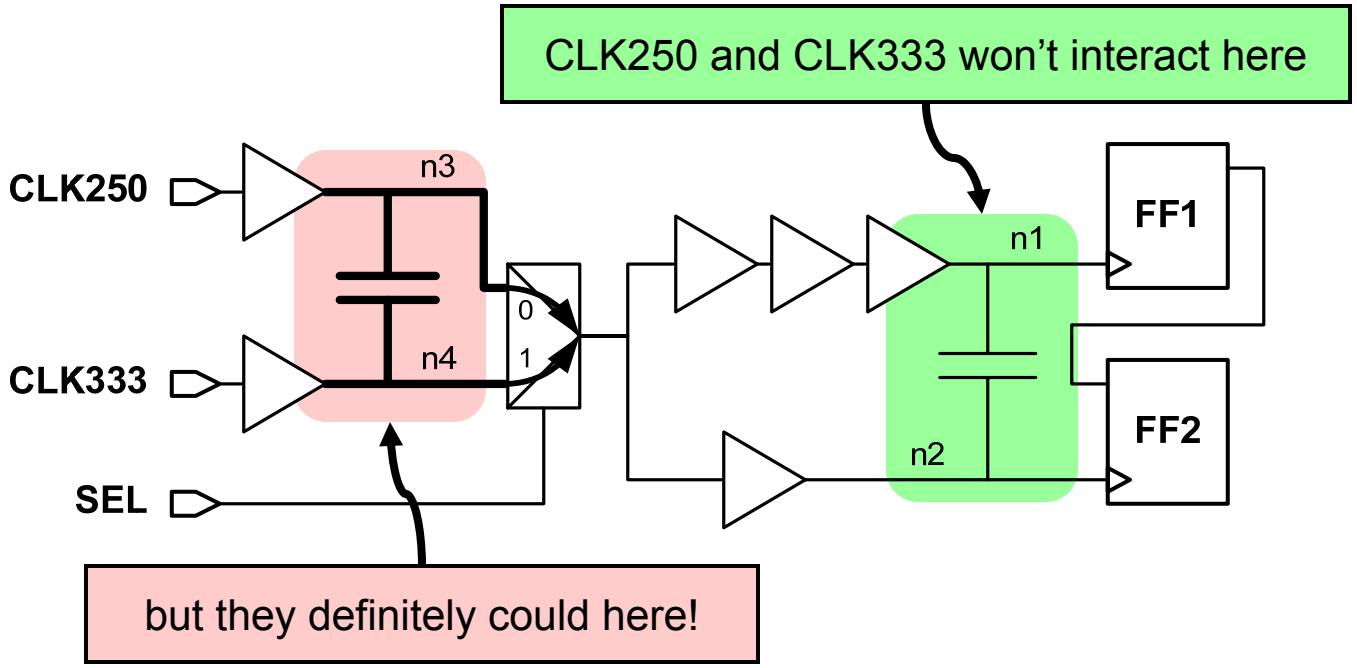
- We mentioned that *physically exclusive clocks* would allow us to tell PrimeTime SI to ignore SI interactions between CLK250 and CLK333:

```
set_clock_groups -physically_exclusive \
  -group CLK250 -group CLK333
```

- This is great! There must be a catch...

Combining Modes: Script Complexity

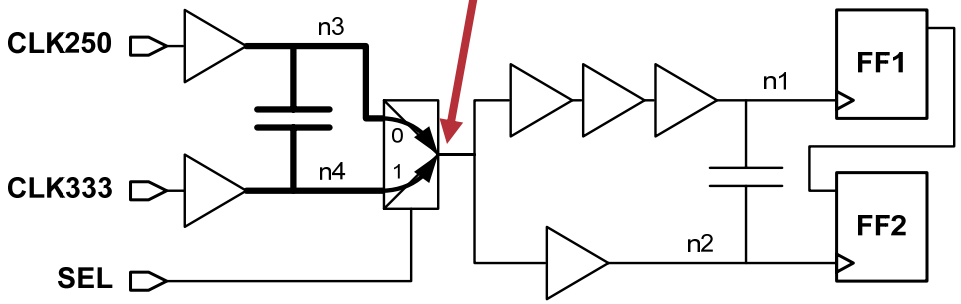
- The catch is, it may not be safe to suppress all CLK250↔CLK333 SI interactions
  - These clocks are only physically exclusive downstream of the mux
  - There could be real interactions before the clocks are muxed together:



Combining Modes: Script Complexity

- We can address this by:
  - creating generated clock versions of the original clocks on the mux input pins
  - applying the physically exclusive property only on these generated clocks
  - making the two frequency domains asynchronous (since now we realize there really could be interactions)

```
create_clock -period 4.00 [get_ports CLK250]
create_clock -period 3.00 [get_ports CLK333]
create_generated_clock -name CLK250_mux -combinational \
  -source [get_ports CLK250] [get_pins UMUX/A]
create_generated_clock -name CLK333_mux -combinational \
  -source [get_ports CLK333] [get_pins UMUX/B]
set_clock_groups -physically_exclusive -group {CLK250_mux} -group {CLK333_mux}
set_clock_groups -asynchronous -group {CLK250 CLK250_mux} -group {CLK333 CLK333_mux}
```



Combining Modes: Script Complexity

- We have gone from two separate runs varying a mode select...

```
create_clock -period 4.00 [get_ports CLK250]
create_clock -period 3.00 [get_ports CLK333]
set_case_analysis 0 [get_ports SEL]
```

```
create_clock -period 4.00 [get_ports CLK250]
create_clock -period 3.00 [get_ports CLK333]
set_case_analysis 1 [get_ports SEL]
```

- ...to a single run with this:

```
create_clock -period 4.00 [get_ports CLK250]
create_clock -period 3.00 [get_ports CLK333]
create_generated_clock -name CLK250_mux -combinational \
  -source [get_ports CLK250] [get_pins UMUX/A]
create_generated_clock -name CLK333_mux -combinational \
  -source [get_ports CLK333] [get_pins UMUX/B]
set_clock_groups -physically_exclusive -group {CLK250_mux} -group {CLK333_mux}
set_clock_groups -asynchronous -group {CLK250 CLK250_mux} -group {CLK333 CLK333_mux}
```

- We reduced the number of runs, but at a complexity cost
  - This process must be repeated for every pin where clocks are combined

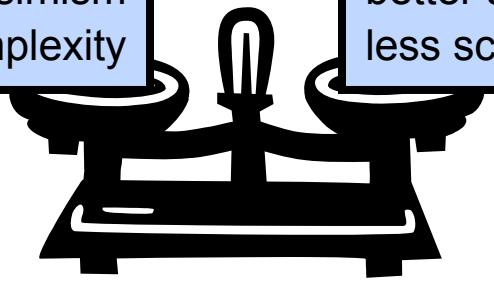
Script Complexity: What Have We Learned?

- When combining modes, timing and SI interactions between modes must be carefully considered
  - Know what clock relationships should and shouldn't be applied
  - Analysis scripts with combined modes require additional scripting complexity to apply the correct relationships
- To be fair, these are not showstopper effects
  - For some designs, the accuracy, memory, runtime, and complexity effects may be minimal
  - Sometimes combined modes are the only option!
- The goal of this section is to provide knowledge to:
  - Know which modes are good merge candidates and which aren't
  - Know what to look for as you combine different modes

Merging Modes: What Have We Learned?

- Merging modes is a balancing act

Merged modes?	Separate modes?
fewer runs	more runs
more memory/runtime per run	less memory/runtime per run
some pessimism	better accuracy
more script complexity	less script complexity



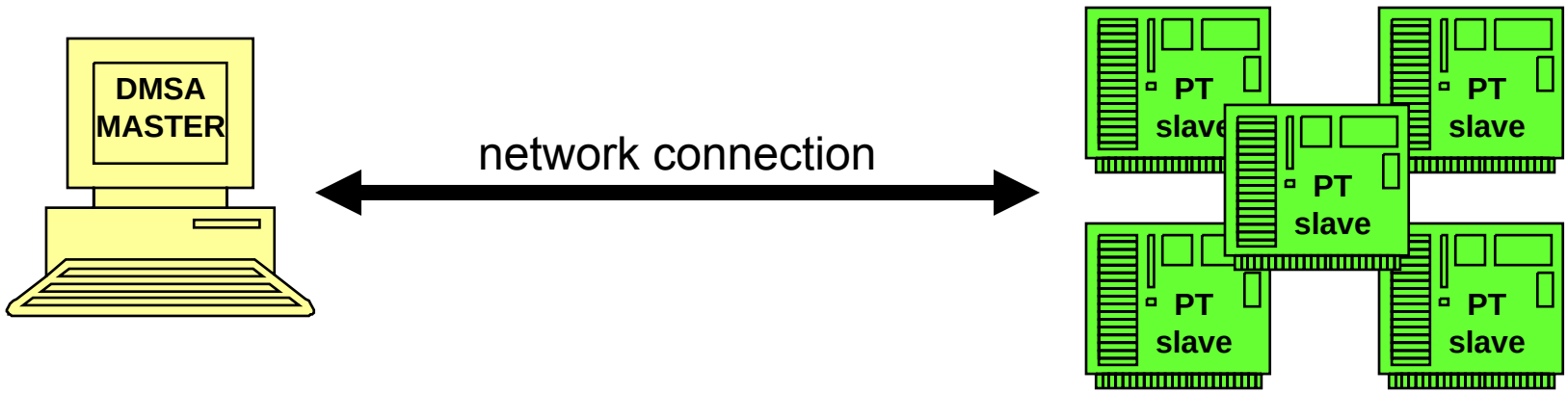
- Merging modes is not always a bad thing
  - For some designs, the accuracy, memory, runtime, and complexity costs may be minimal
- Again, the goal of this presentation is to provide the knowledge to:
  - Know which modes are good merge candidates and which aren't

Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - Accuracy Impact
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - **Introduction and Setup**
  - Features
  - Example: Hold-Fixing
- Summary

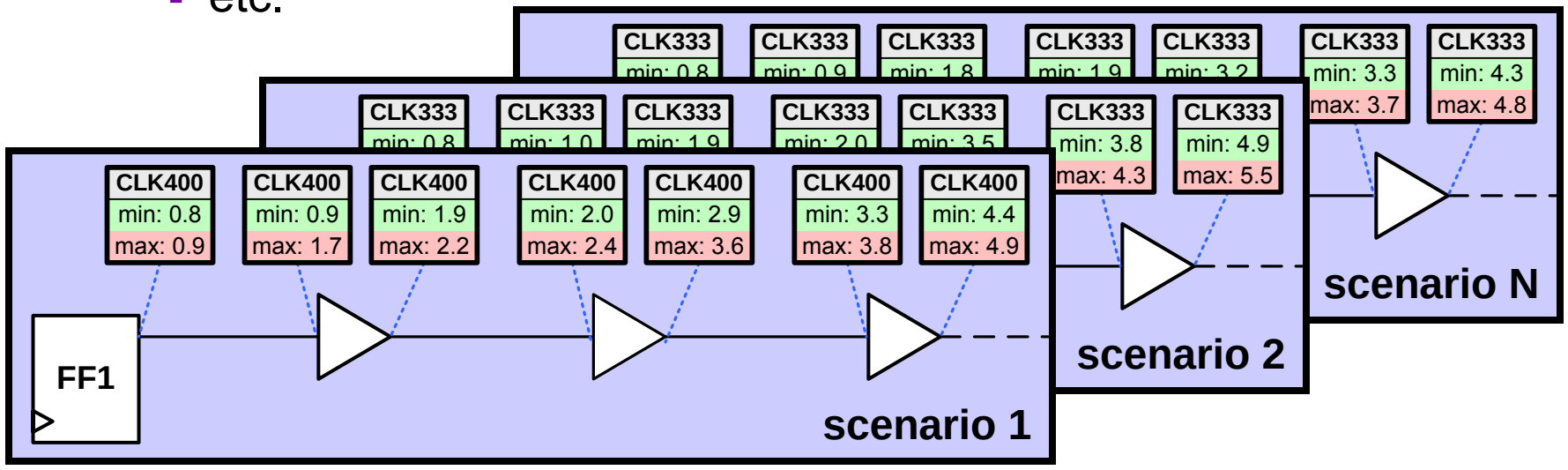
What Is Distributed Multi-Scenario Analysis (DMSA)?

- DMSA provides efficient unified analysis of multiple PrimeTime analyses (or *scenarios*) from a single *master* PrimeTime
  - We can work with multiple analyses as easily as with a single PrimeTime analysis run!



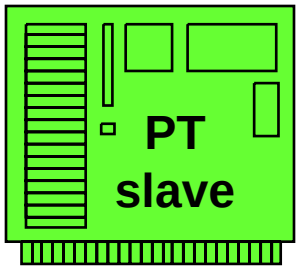
DMSA Uses Independent Accurate Analyses

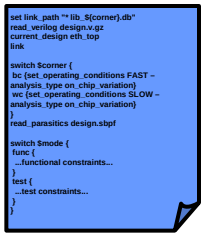
- Each scenario in a DMSA run is a full PrimeTime analysis which can have its own unique:
  - delay and slew timing
  - detailed parasitics
  - PrimeTime variable settings
  - etc.



Setting Up DMSA

- To set up a DMSA run, three key components must be configured:
  - Number of licenses
  - Number of remote slave processes
  - Scenario definitions
- Let's take a look at how these are configured

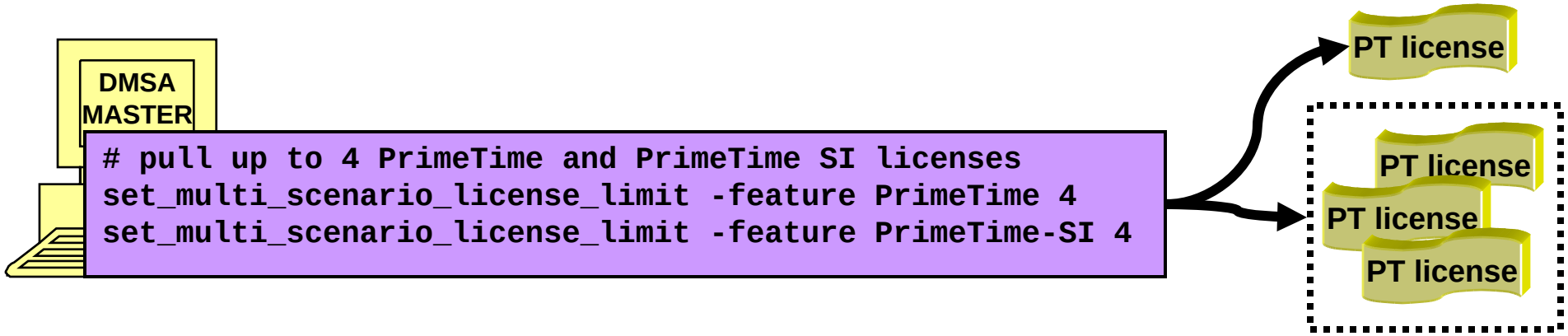




analysis script(s)

Setting Up DMSA: Configuring Licenses

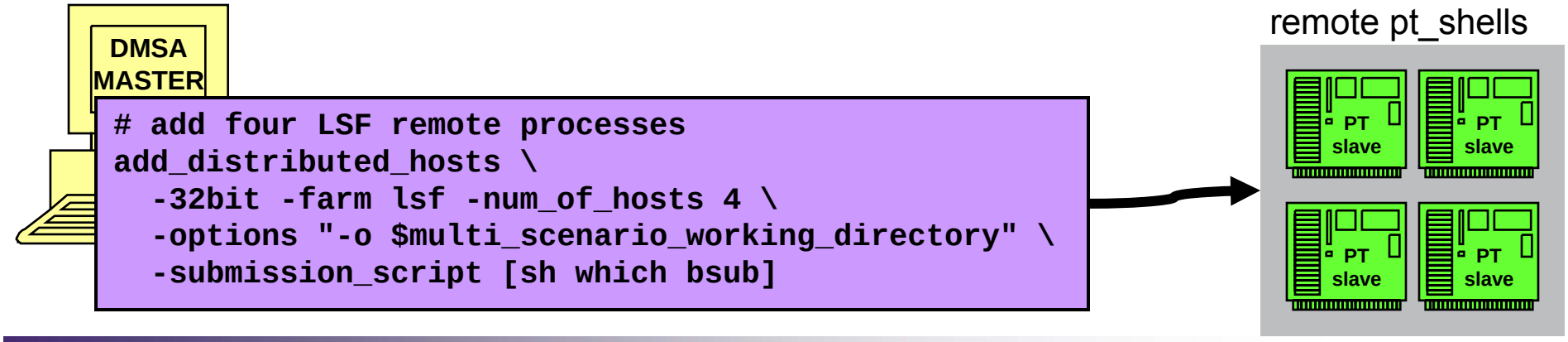
- We give DMSA the upper limit for the number of licenses to use
  - A minimum of one is needed to do any work
  - Additional licenses are pulled from the license server as needed
  - The master does not require an extra license
- In the example below:
  - We allow DMSA up to 4 PrimeTime and PrimeTime-SI licenses



SYNOPSYS® Predictable Success

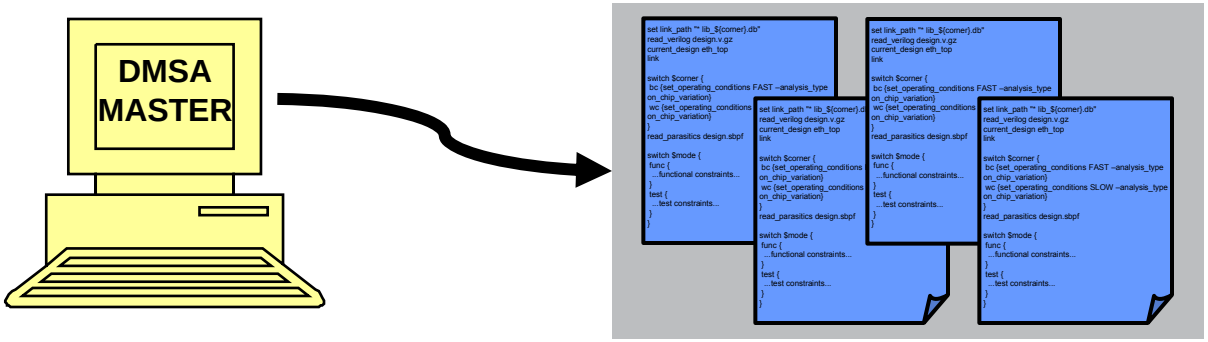
Setting Up DMSA: Configuring Remote Processes

- We give DMSA information on the remote processes to invoke
  - Compute farms are supported (LSF, GRD, proprietary)
  - Discrete named machines can be given
  - A mix of platforms (linux, sparc, 32/64-bit, etc.) can be used
- In the example below:
  - We ask DMSA to invoke 4 processes on an LSF farm

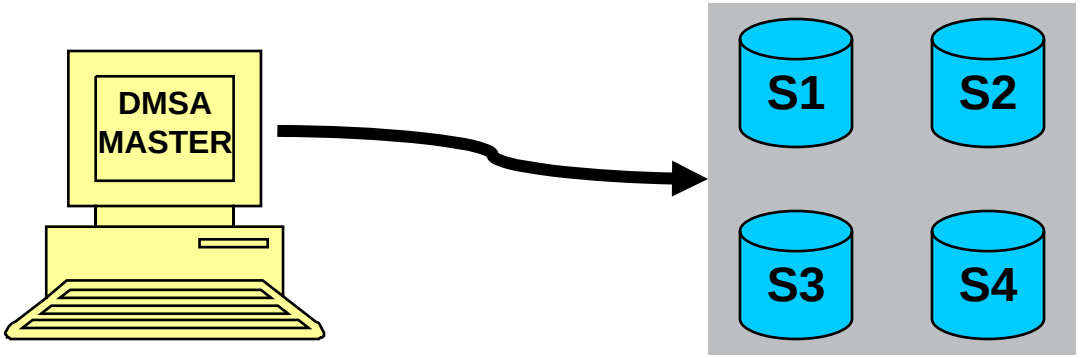


Setting Up DMSA: Defining Scenarios

- The analysis *scenarios* are defined in one of two ways:
  - Providing PrimeTime scripts which load/constrain the analyses:



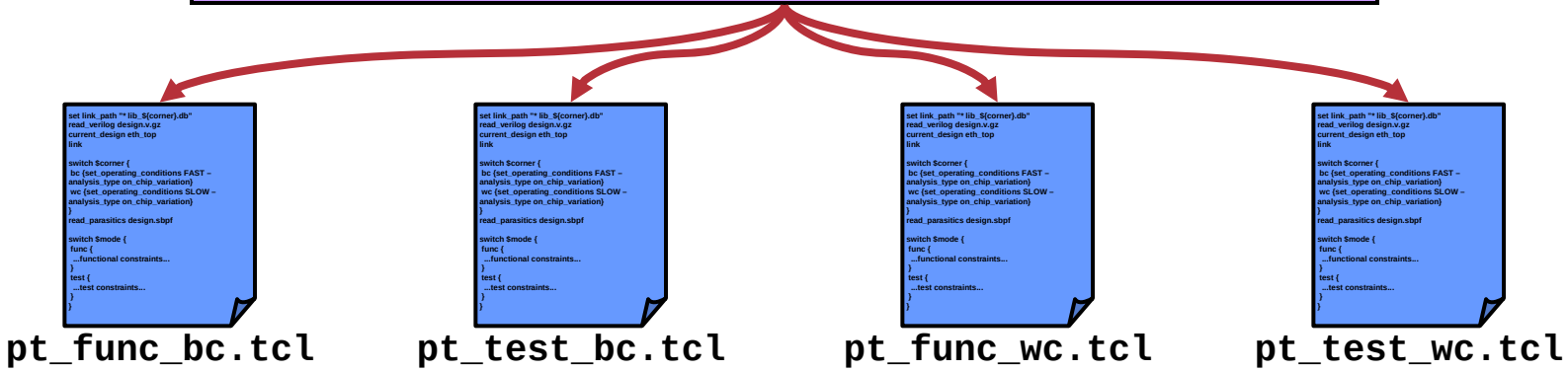
  - Providing existing PrimeTime saved sessions:



Setting Up DMSA: Defining Scenarios Using Scripts

- If you have a different script for each scenario, just point to the proper script for that scenario definition

```
foreach corner {bc wc} {
 foreach mode {func test} {
  create_scenario \
   -name ${mode}_${corner} \
   -specific_data "pt_${mode}_${corner}.tcl"
 }
}
```



Setting Up DMSA: Defining Scenarios Using Scripts

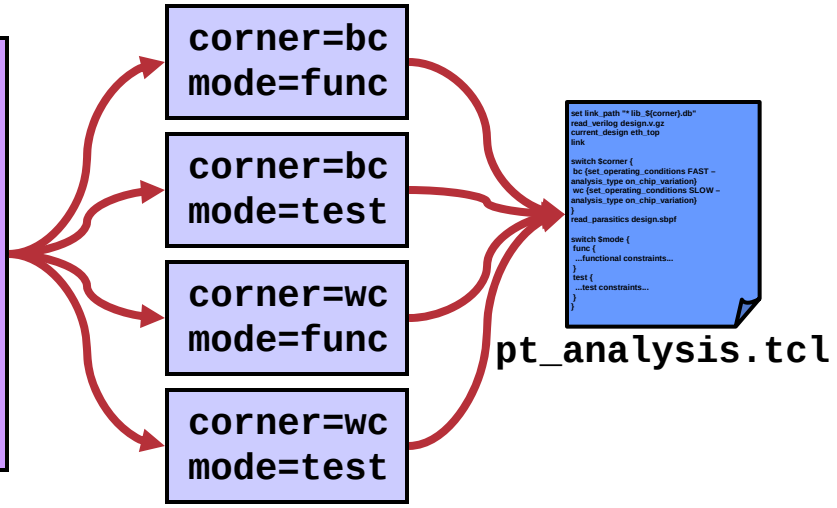
- The more common case is a single analysis script using variables to control which mode/corner is analyzed
  - Below, **mode** and **corner** control the analysis

```
set link_path "* lib_${corner}.db"
read_verilog design.v.gz
current_design eth_top
link

switch $corner {
 bc {set_operating_conditions FAST -analysis_type on_chip_variation}
 wc {set_operating_conditions SLOW -analysis_type on_chip_variation}
}
read_parasitics design.sbpf

switch $mode {
 func {
  ...functional constraints...
 }
 test {
  ...test constraints...
 }
}
```

Setting Up DMSA: Defining Scenarios Using Scripts

- In this case, we simply vary these variables across their different combinations and create scenarios:

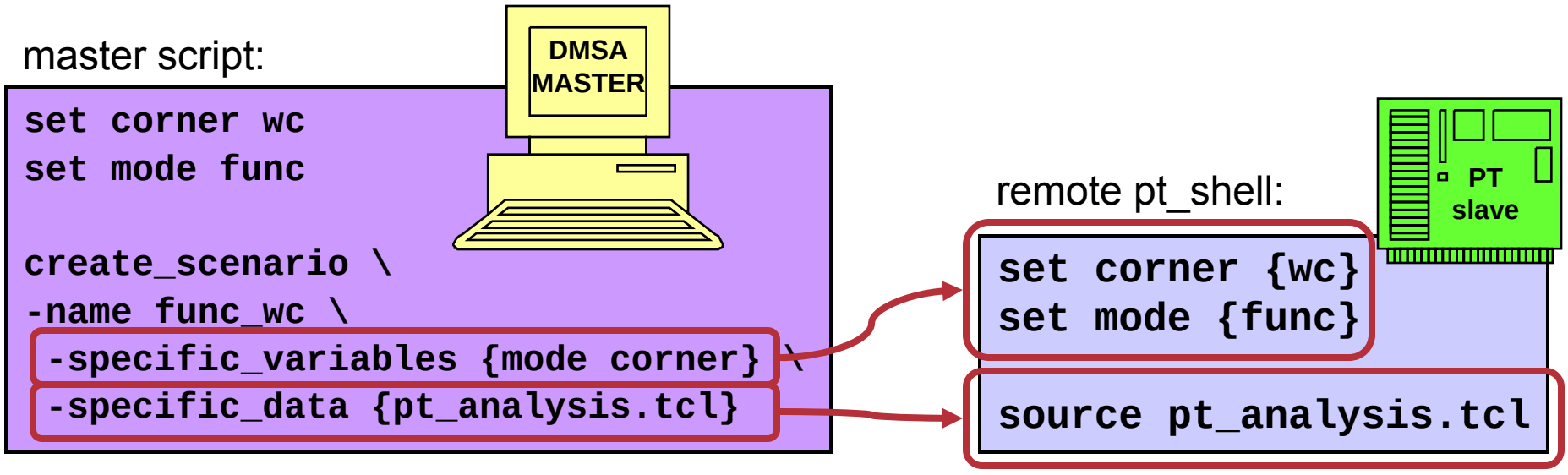
```
foreach corner {bc wc} {
 foreach mode {func test} {
  create_scenario \
   -name ${mode}_${corner} \
   -specific_variables {mode corner} \
   -specific_data {pt_analysis.tcl}
 }
}
```



- The **`-specific_variables`** option "pushes" a variable's current master value into the scenario
  - How does this work?

Setting Up DMSA: Defining Scenarios Using Scripts

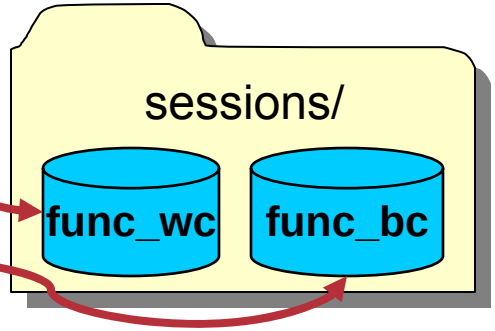
- Using **`-specific_variables`** in a scenario definition can be thought of as:
  - invoking a PrimeTime session
  - setting the specified variables
  - sourcing the scenario script



Setting Up DMSA: Defining Scenarios Using Sessions

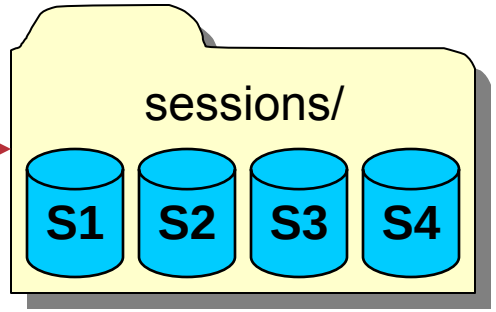
- You can also define scenarios with saved sessions:

```
create_scenario \
  -name func_wc –image sessions/func_wc
create_scenario \
  -name func_bc –image sessions/func_bc
```



- SolvNet article 018039 provides a Tcl procedure to easily restore a directory of saved sessions:

```
restore_dmsa_session sessions/
```

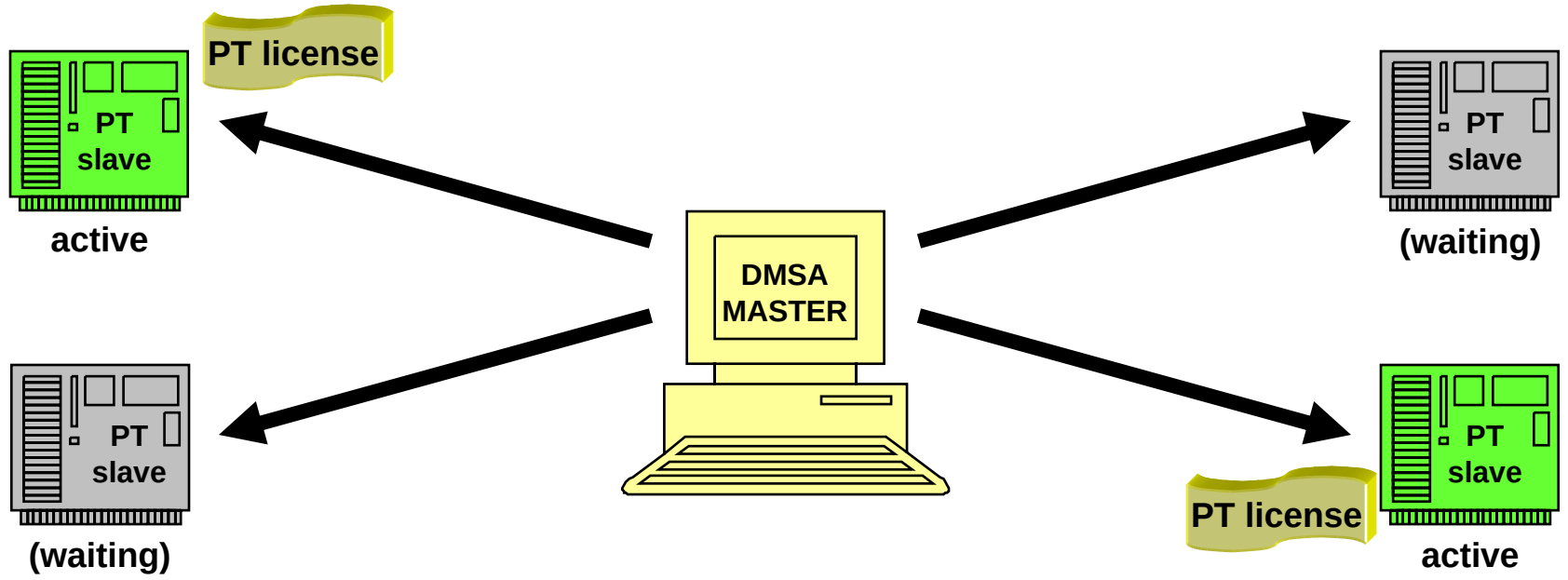


Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - Accuracy Impact
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - **Features**
  - Example: Hold-Fixing
- Summary

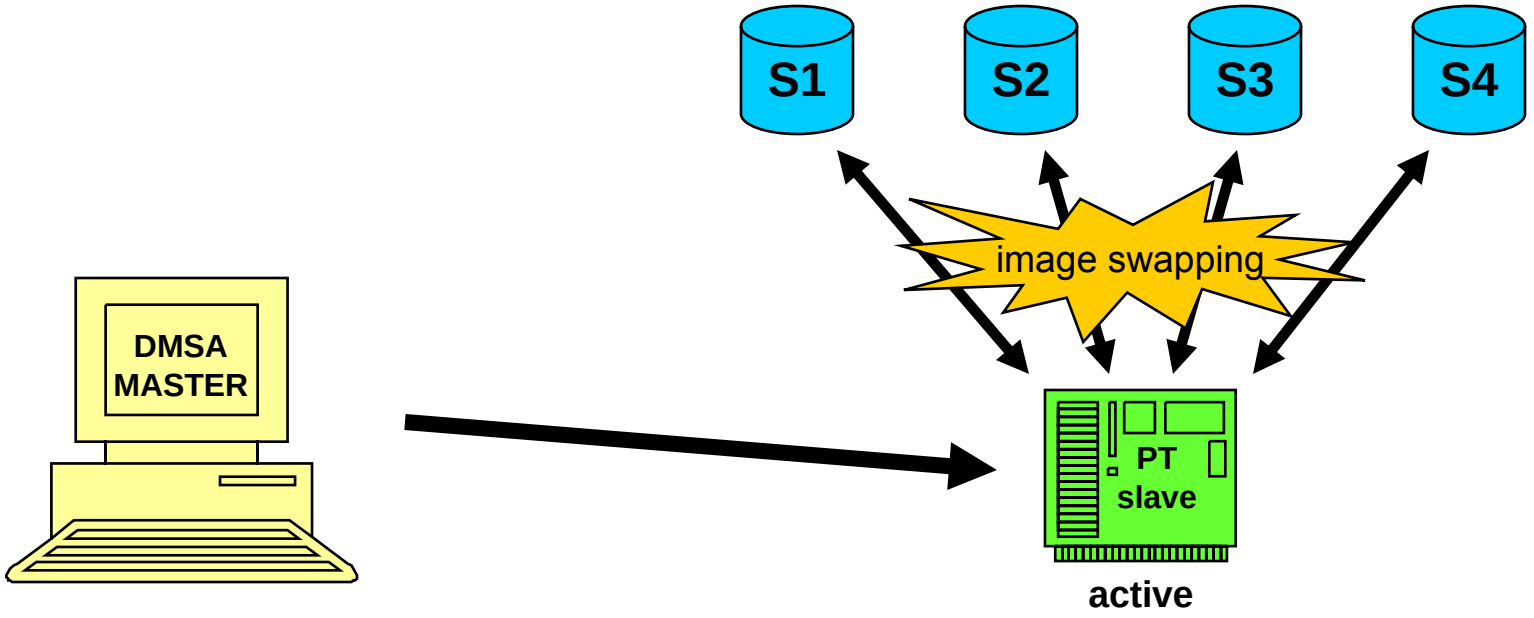
Flexible License Management

- When (# licenses) < (# remote processes):
  - The master distributes licenses to the remote processes
  - When a task finishes at a remote process, the license is freed to be immediately used by another process
  - Remote processes are not terminated to free license



Flexible Machine Management

- When (# remote processes) < (# scenarios):
  - Scenario images must be swapped to disk to execute the task across all scenarios
    - This is expensive in terms of disk activity and runtime
  - This should be avoided unless absolutely necessary!

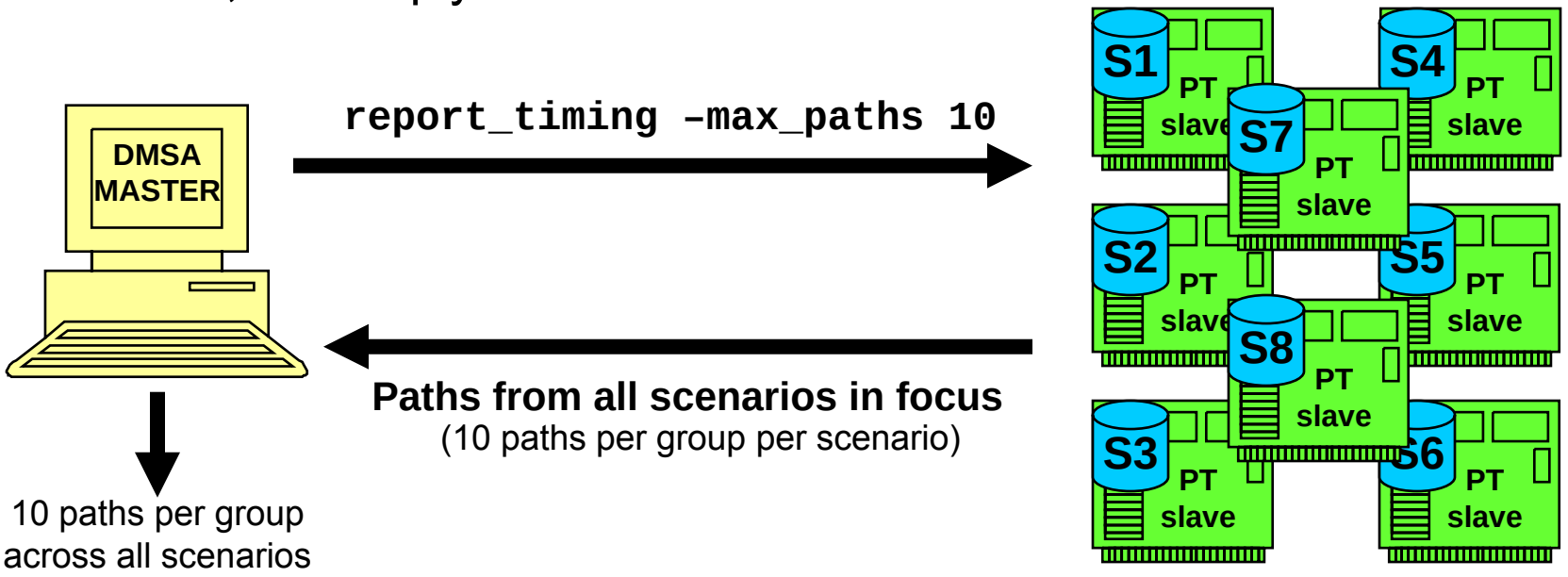


Merged Reporting

- We have seen that DMSA lets us easily run all scenarios together
- The big payoff is that we can also easily analyze all scenarios together!
  - The easiest method is using *merged reporting*, where we can issue reporting commands across all scenarios
- Merged reporting significantly helps complexity management when there are many scenarios
  - Let's take a look at some example reports...

Merged report_timing

- To see the 10 worst paths in each path groups across all scenarios, we simply issue:



- DMSA automatically merges the results across all scenarios, saving us from sifting through hundreds of nearly identical reports
  - Topological duplicates are automatically removed

Merged `report_timing`

- Merged reports typically include the scenario name in the report
  - Below is a timing report from **`report_timing`**

```
Startpoint: m_wb_ack_i (input port clocked by wb_clk_i)
Endpoint: core/wishbone/IncrTxPointer_reg
            (rising edge-triggered flip-flop clocked by wb_clk_i)
Path Group: inputs
Path Type: max
Scenario: func_wc
Max Data Paths Derating Factor  : 1.00
Min Clock Paths Derating Factor : 0.95
Max Clock Paths Derating Factor : 1.00
```

Merged report_timing

- We can also use "**-path_type end**" to see path summaries listed with scenario and slack columns

```
pt_shell> report_timing -path_type end -max_paths 10 –delay min
...

Endpoint                                Path Delay  Path Required   Slack  Scenario
-------------------------------------------------------------------------------------
Busy_IRQ_sync2_reg/D (DFQD1)              0.68 f          0.75    -0.07   wc
RxAbortSync4_reg/D (DFCNQD1)              0.72 f          0.78    -0.06   wc
ethreg1/SetTxCIrq_sync3_reg/D (DFCNQD1)   0.41 f          0.47    -0.05   bc
SendControlFrame_sync3_reg/D (DFCNQD1)    0.57 f          0.62    -0.05   tc
WriteRxDataToFifoSync3_reg/D (DFCNQD1)    0.58 f          0.63    -0.05   tc
miim1/RStat_q2_reg/D (DFCND1)             0.41 f          0.46    -0.05   bc
SyncRxStartFrm_q_reg/D (DFCND1)           0.73 f          0.78    -0.05   wc
Busy_IRQ_sync3_reg/D (DFQD1)              0.70 f          0.74    -0.05   wc
WriteRxDataToFifoSync2_reg/D (DFCND1)     0.58 f          0.63    -0.05   tc
miim1/EndBusy_reg/D (DFCND1)              0.41 f          0.46    -0.05   wc
```

Merged `report_si_bottleneck`

- The merged **`report_si_bottleneck`** shows the worst victims across all scenarios
  - Great for finding and fixing the worst crosstalk problems across all scenarios

```
Bottleneck Cost:  delta_delay
net                                                    scenario  cost
-------------------------------------------------------------------------
n42                                                    wc           0.08
m_wb_adr_o[8]                                          wc           0.06
wishbone/bd_ram/C36257/n295                            wc           0.05
temp_wb_dat_o[18]                                      wc           0.03
n46                                                    bc           0.02
wishbone/bd_ram/C36257/n311                            wc           0.01
wb_sel_i[0]                                            wc           0.01
wishbone/WriteRxDataToFifoSync1                        wc           0.01
wishbone/ReadTxDataFromFifo_sync1                      wc           0.01
wishbone/TxDone_wb                                     bc           0.01
wishbone/TxAbortSync1                                  tc           0.01
```

ECOs Across Scenarios

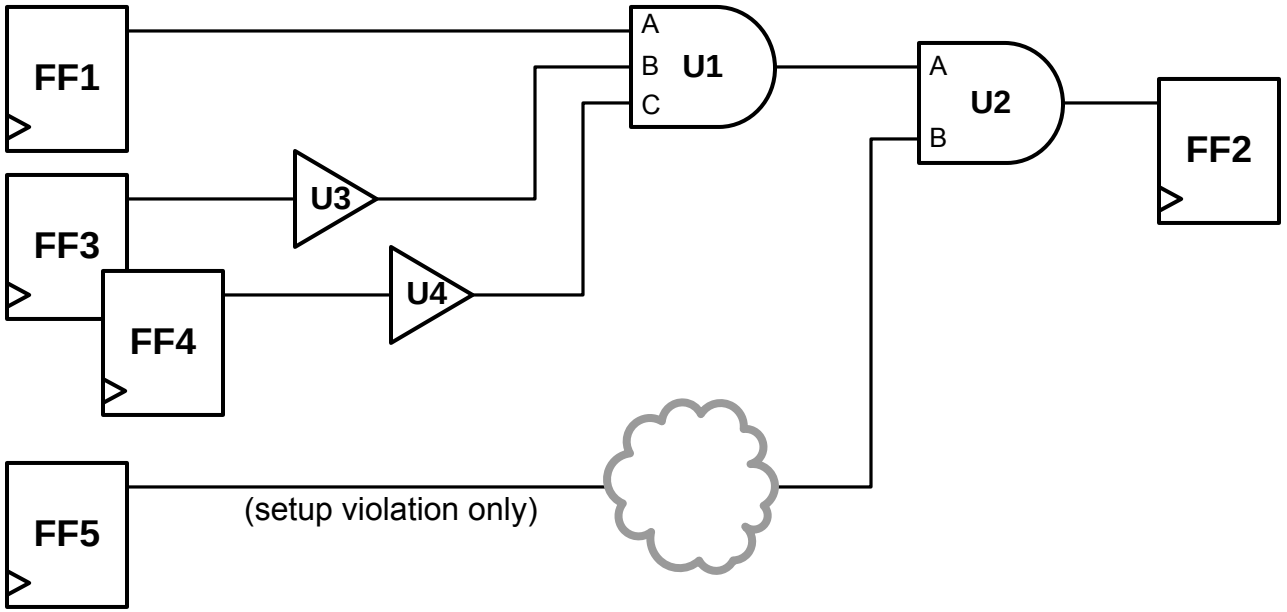
- All the usual ECO commands are available:
  - **`size_cell`**
  - **`insert_buffer`**
  - **`remove_buffer`**
  - **`create_cell`**, **`create_net`**
  - **`remove_cell`**, **`remove_net`**
  - **`connect_net`**, **`disconnect_net`**
- You can resize a cell or insert a buffer and immediately see the results across all scenarios
  - All scenarios update incrementally

Agenda

- Where Multiple Analyses Come From
- Combining Modes
  - Introduction
  - Accuracy Impact
  - Memory/Runtime Requirements
  - Script Complexity
- Distributed Multi-Scenario Analysis (DMSA)
  - Introduction and Setup
  - Features
  - **Example: Hold-Fixing**
- Summary

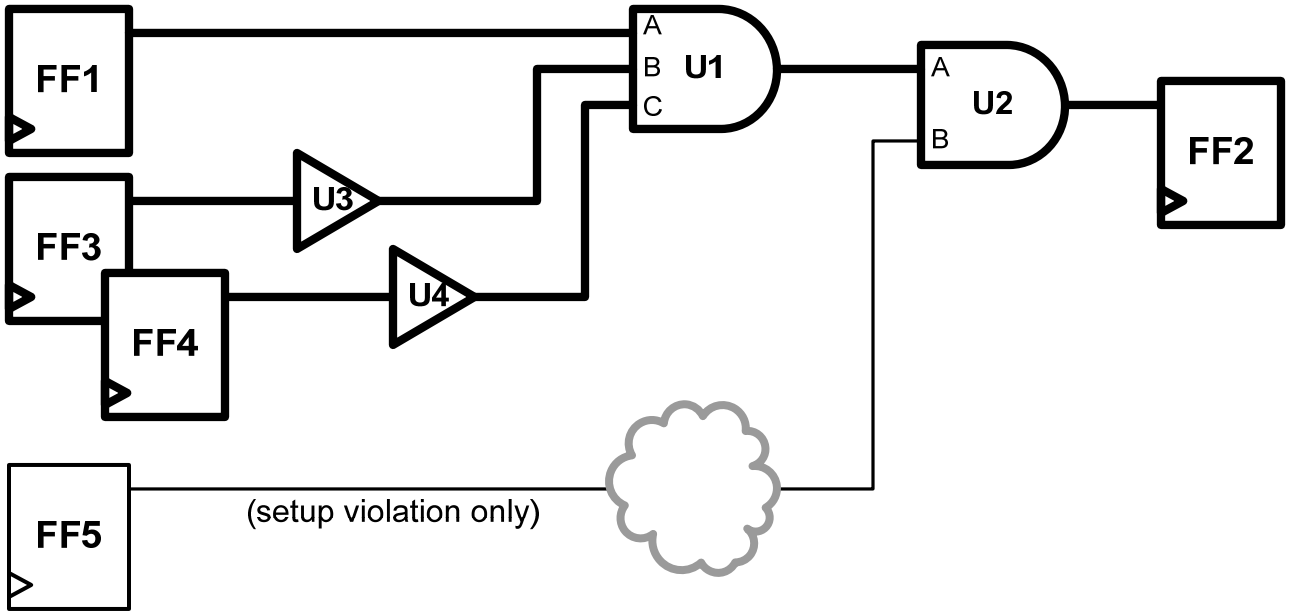
DMSA Example: Hold-Fixing Across Scenarios

- How can we fix hold violations in all scenarios without introducing setup in any scenarios?
  - Furthermore, how can we optimize the buffer placement to minimize the buffer insertion count?



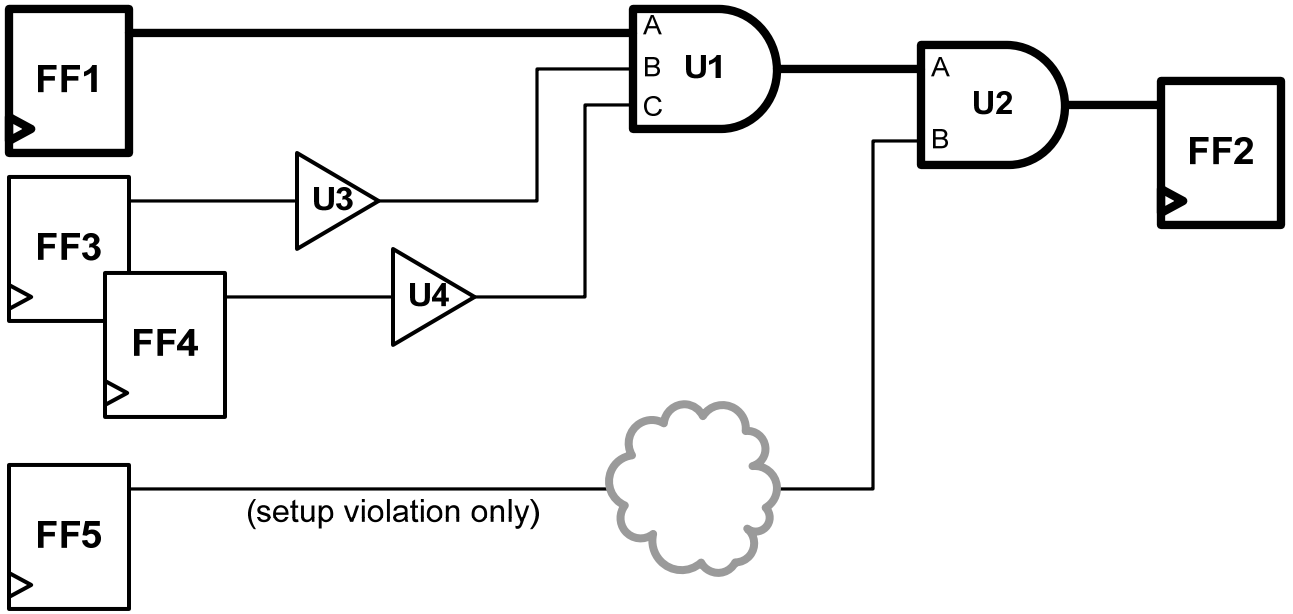
DMSA Example: Hold-Fixing Across Scenarios

- In our example circuit below:
  - The worst hold violation goes through U1/A
  - Sub-critical hold violations go through U1/B and U1/C
  - A critical setup path comes in on a side pin



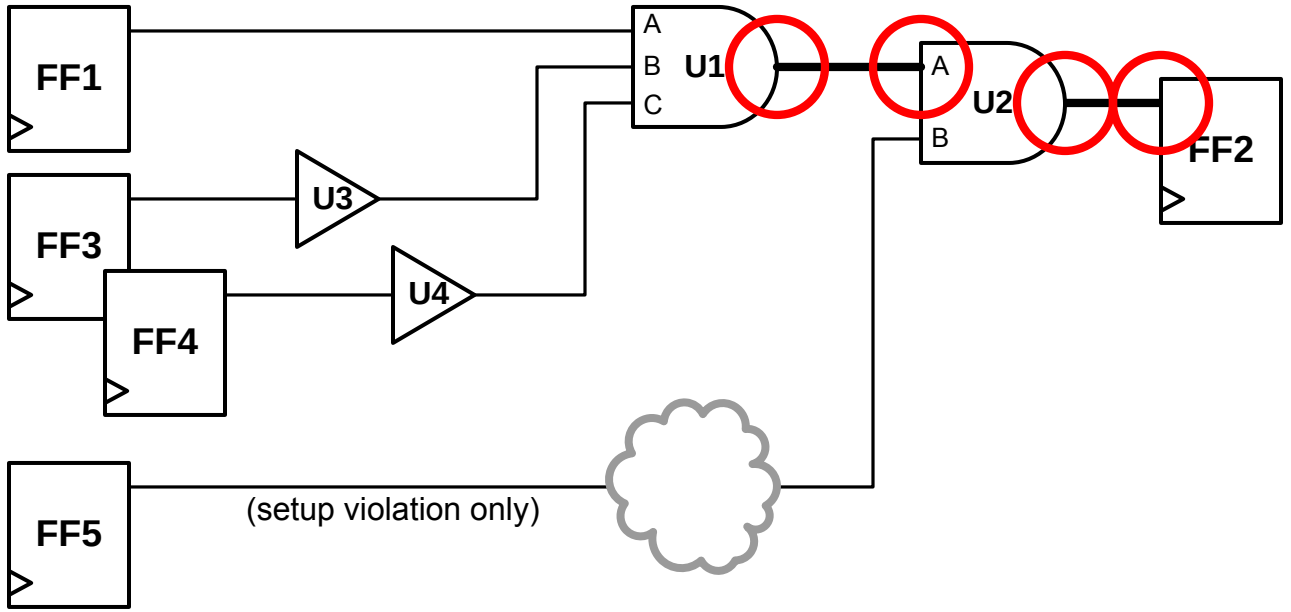
DMSA Example: Hold-Fixing Across Scenarios

- Priority #1 is to work on the worst hold violating path in a cloud of logic
  - Below, we want to improve hold slack along the bolded path



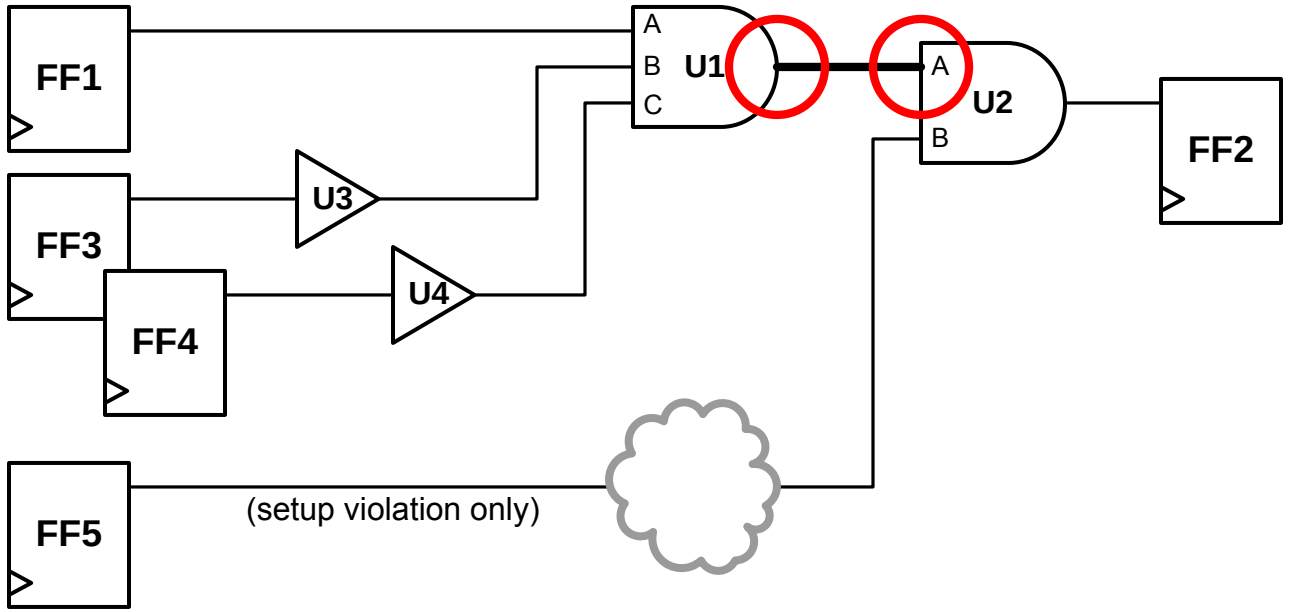
DMSA Example: Hold-Fixing Across Scenarios

- Priority #2 is to find the pin associated with the largest number of hold violations
  - Below, we prefer the indicated pins which have three hold violations through them



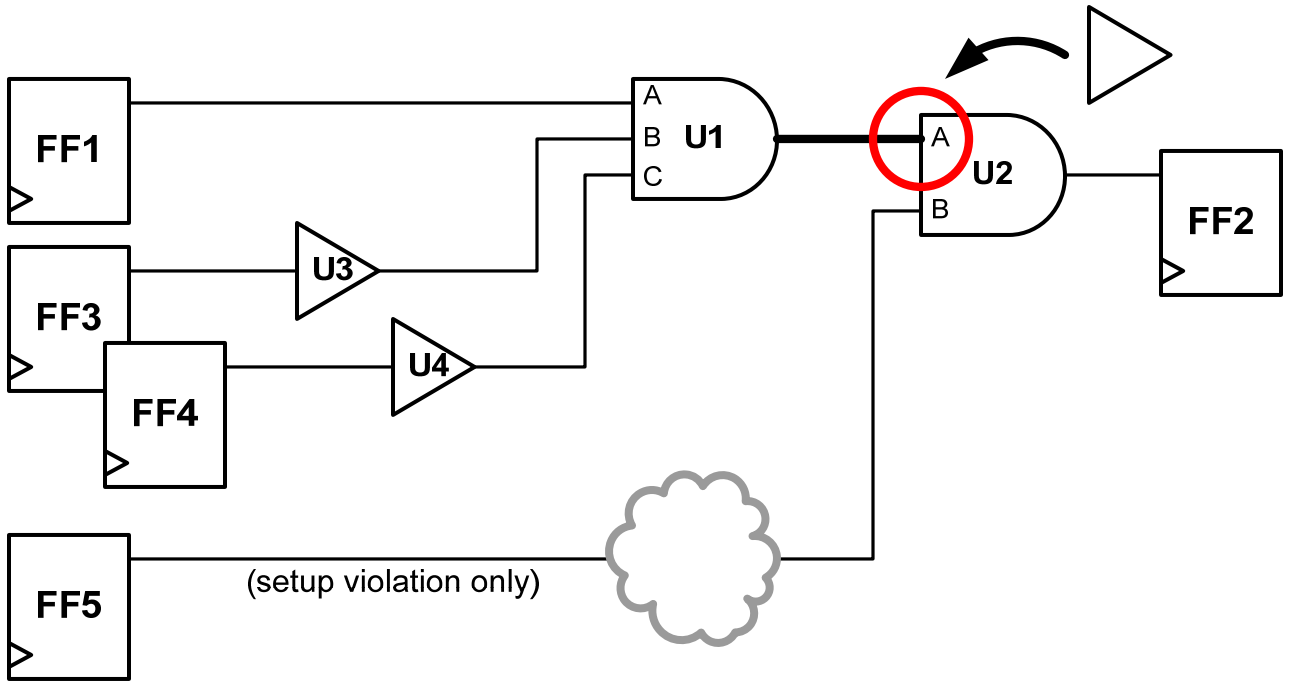
DMSA Example: Hold-Fixing Across Scenarios

- Priority #3 is to find the pin with the largest setup slack
  - Below, the indicated pins have the most setup slack “room” to tolerate additional delay



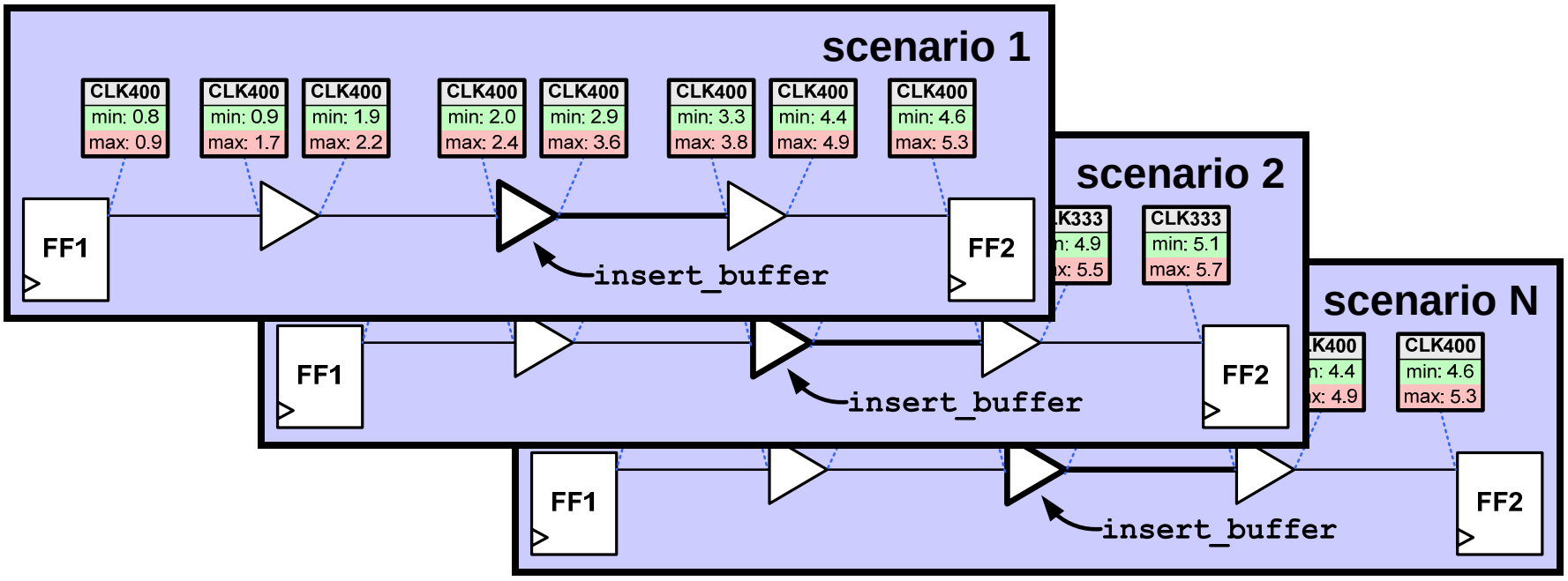
DMSA Example: Hold-Fixing Across Scenarios

- Priority #4 is to prefer load pins
  - If the two pins are otherwise identical (same # paths, slack, etc.), inserting a buffer at the load pin has more predictable delay



DMSA Example: Hold-Fixing Across Scenarios

- Each scenario has unique min/max rise/fall timing
  - Inserted delay cells have different timing characteristics in each scenario (depending on PVT corner, voltage, parasitics, etc.)



DMSA Example: Hold-Fixing Across Scenarios

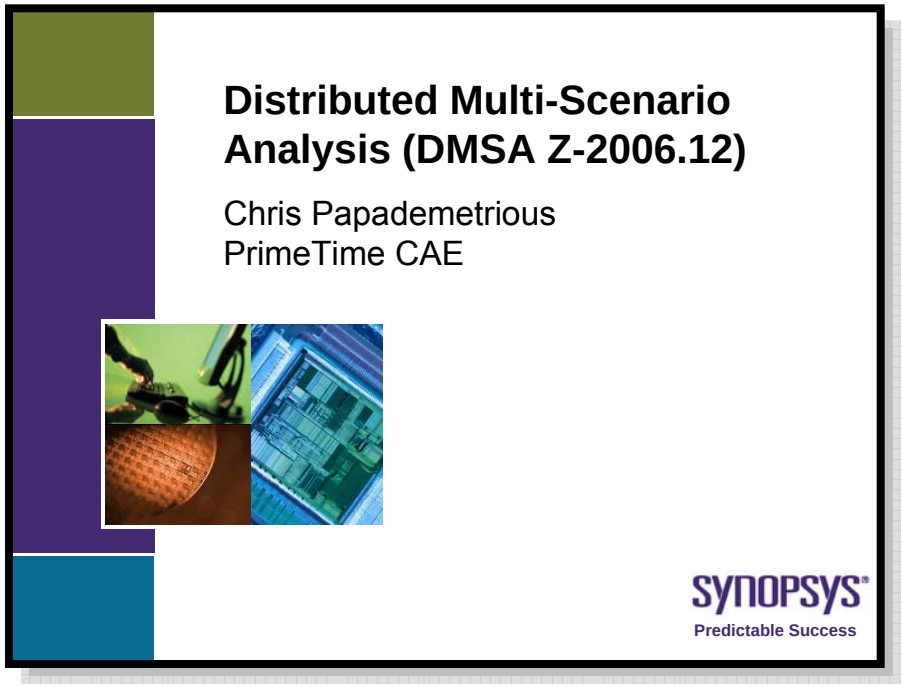
- This DMSA hold-fixing script is available in SolvNet
  - Fixes hold violations while considering setup requirements across all scenarios
  - Considers the timing and slack of every scenario thoroughly
  - Finds optimal placement for delay according to cost factor previously described
- This script is just one example of the power of DMSA!

018510: "Fixing Hold Violations Across Many Modes and Corners with DMSA"

http://solvnet.synopsys.com/retrieve/018510.html

Learning More About DMSA

- Your Synopsys AC can deliver a more detailed training presentation which goes into more detail on:
  - setting up DMSA
  - available features and commands
  - using existing STA scripts



Distributed Multi-Scenario Analysis: What Have We Learned?

- Modes and corners are increasing in number!
  - Corner analyses cannot be combined
  - Operating modes can be combined by trading off accuracy and capacity
- DMSA enables easy setup and execution of large numbers of analyses
- DMSA allows you to manage precious machine and license resources
- With DMSA, the goal is for you – the engineer – to be able to get...
  - the most accurate signoff analysis
  - in less time
  - with less effort
  - and with fewer machine/license resources

Q&A

- Feel free to ask any questions!
  - I have the DMSA technical training slides available if we need them

SYNOPSYS®

Predictable Success