SYNOPSYS®

Pervasive AI

Eric Bouet

11/13/2025

Legal Disclosure

CONFIDENTIAL INFORMATION

The information contained in this presentation is the confidential and proprietary information of Synopsys. You are not permitted to disseminate or use any of the information provided to you in this presentation outside of Synopsys without prior written authorization.

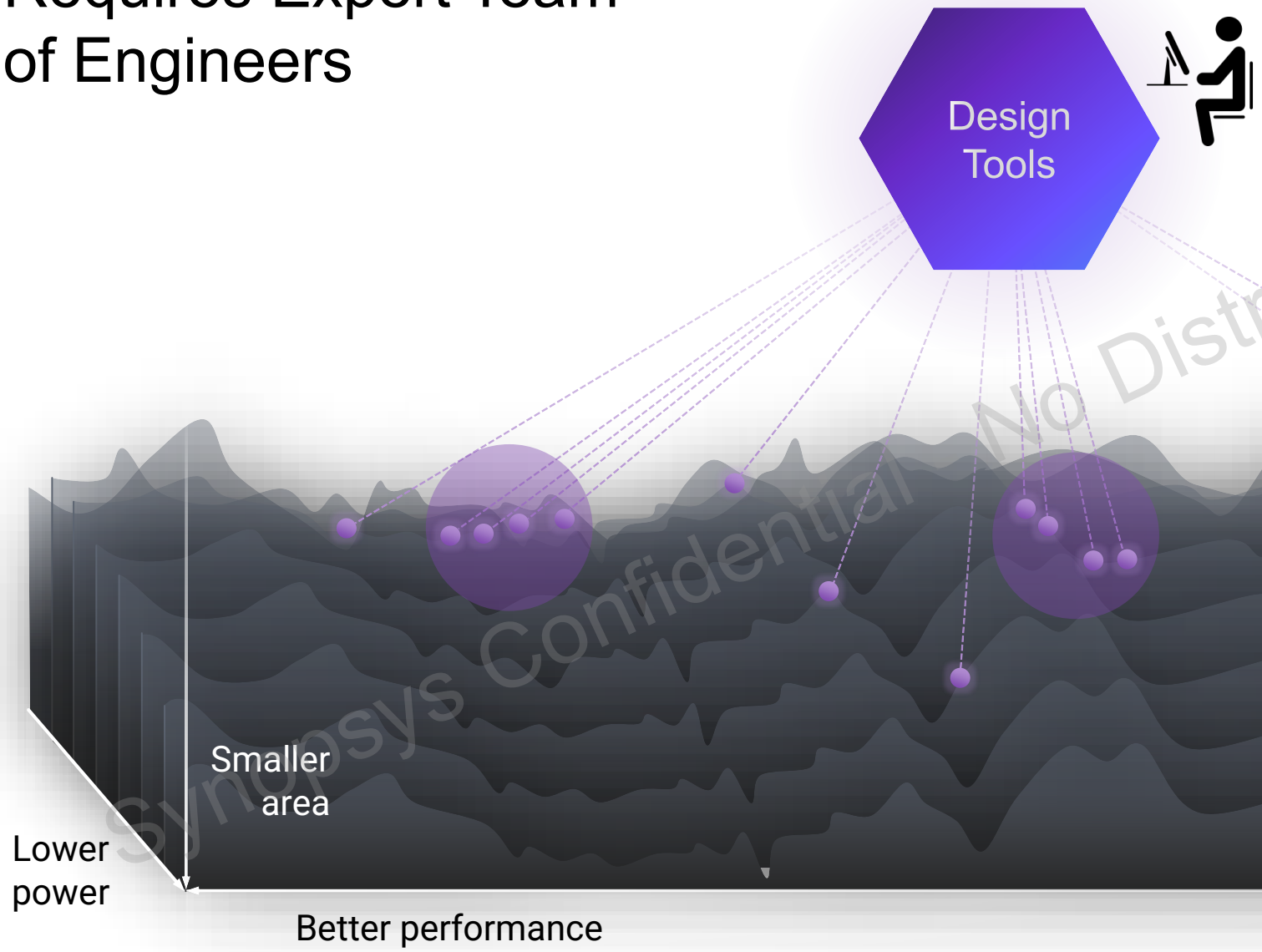
IMPORTANT NOTICE

This presentation may include information related to Synopsys' future product or business plans. Such plans are as of the date of this presentation and subject to change. Synopsys is not obligated to update this presentation or develop the products with the features and/or functionality discussed in this presentation. Additionally, Synopsys' products and services may only be offered and purchased pursuant to an authorized quote and purchase order or a mutually agreed upon written contract.

FORWARD LOOKING STATEMENTS

This presentation may include certain statements including, but not limited to, Synopsys' financial targets, expectations and objectives; business and market outlook, business opportunities, strategies and technological trends; and more. These statements are made only as of the date hereof and subject to change. Actual results or events could differ materially from those anticipated in such statements due to a number of factors. Synopsys undertakes no duty to, and does not intend to, update any statement in this presentation, whether as a result of new information, future events or otherwise, unless required by law.

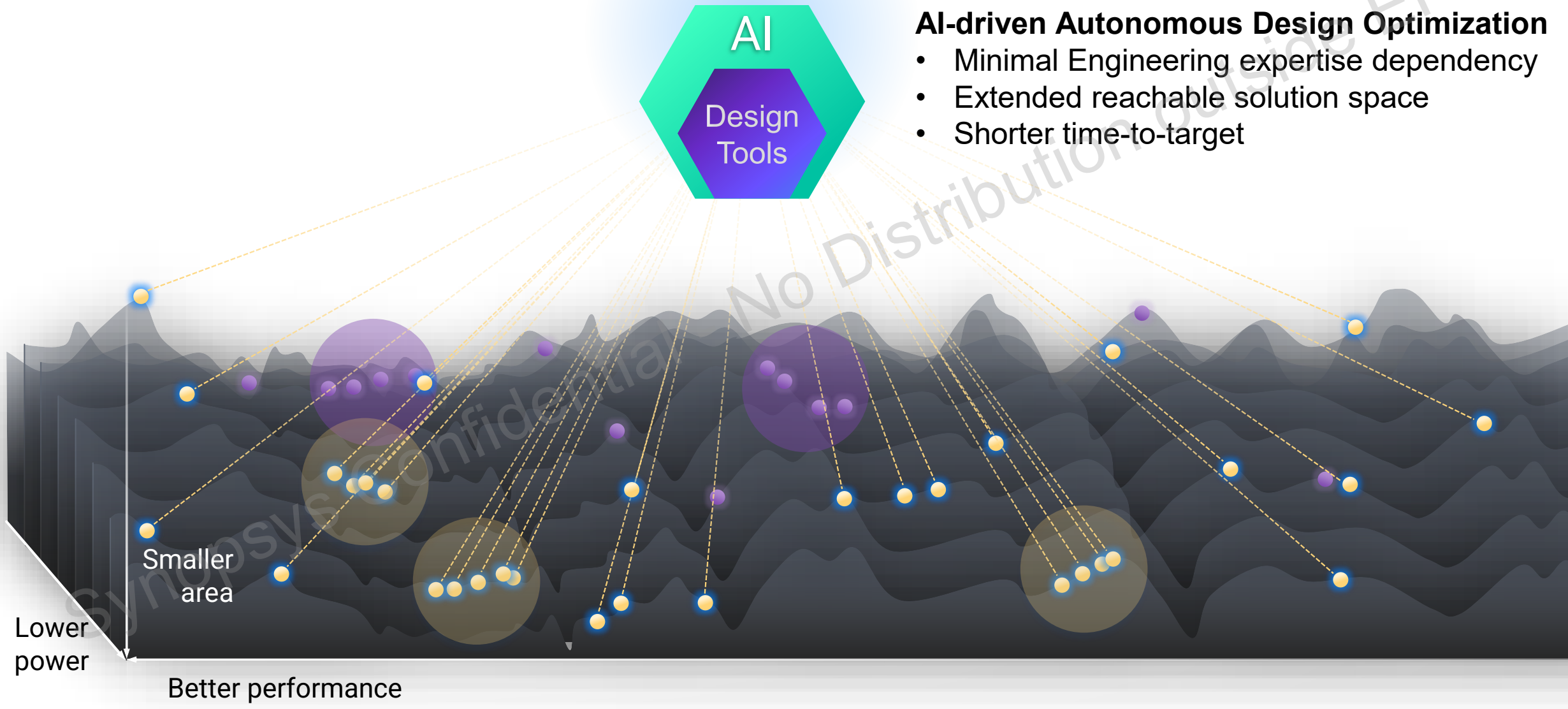
Design Optimization Requires Expert Team of Engineers



Manual Analysis by dedicated expert-level, experienced team of engineers

- Engineering expertise dependency
- Limited reachable solution space
- Longer time-to-target

Artificial Intelligence Helps the Entire Team Perform Like Experts



AI-driven Autonomous Design Optimization

- Minimal Engineering expertise dependency
- Extended reachable solution space
- Shorter time-to-target

Synopsys.ai™: First Full-Stack, AI-Driven EDA Suite

- System Architecture
- Design Capture
- Verification
- Implementation
- Signoff
- Test & SLM
- Silicon Manufacturing

Design Space Optimization 2.0	Achieve power, performance and area (PPA) targets, and boost productivity with DSO.ai
Signoff and ECO	Achieve faster last mile design closure and reduce ECO iteration
Verification Space Optimization	Faster functional testing closure, higher coverage and predictive bug detection with VSO.ai
Test Space Optimization	Automated test generation resulting in fewer, optimized test patterns with TSO.ai
Analog and Manufacturing	Rapid migration of analog designs and development of lithography models

Pervasive AI Delivers EDA Productivity Boost

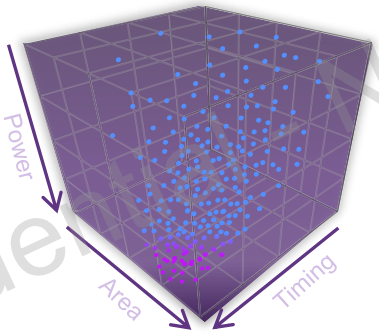
AI-DRIVEN IMPLEMENTATION

Broad AI investment in the market

Widely accessible LLM models

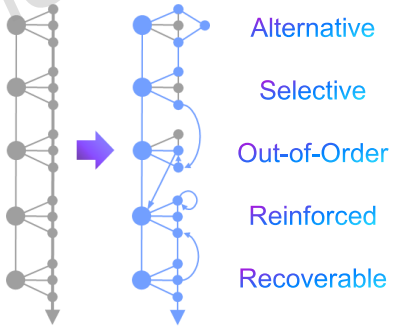
Limitless opportunity to leverage AI for EDA productivity

Exploration
Broader Scope



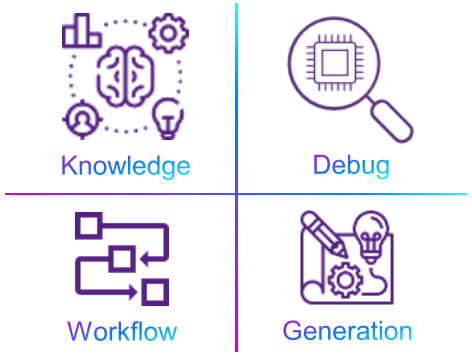
Floorplan / Memory
Metal Stack / Library

Optimization
Adaptive Flow



Autonomous Flow-tuning
Elastic Compute

Assistance
GenAI CoPilot



Workflow / Debug Assist
TechFile / RTL Generation

intel

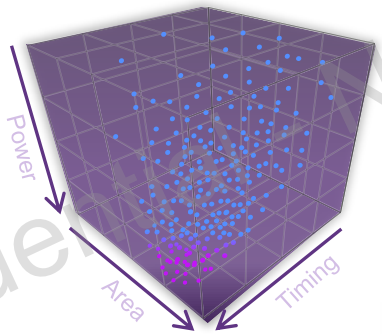
SV SNUG: Push-Button Auto-convergence for GPU Designs: leading to ~25% better execution efficiency; ~7% Total Power Reduction at full-chip level

Pervasive AI Delivers EDA Productivity Boost

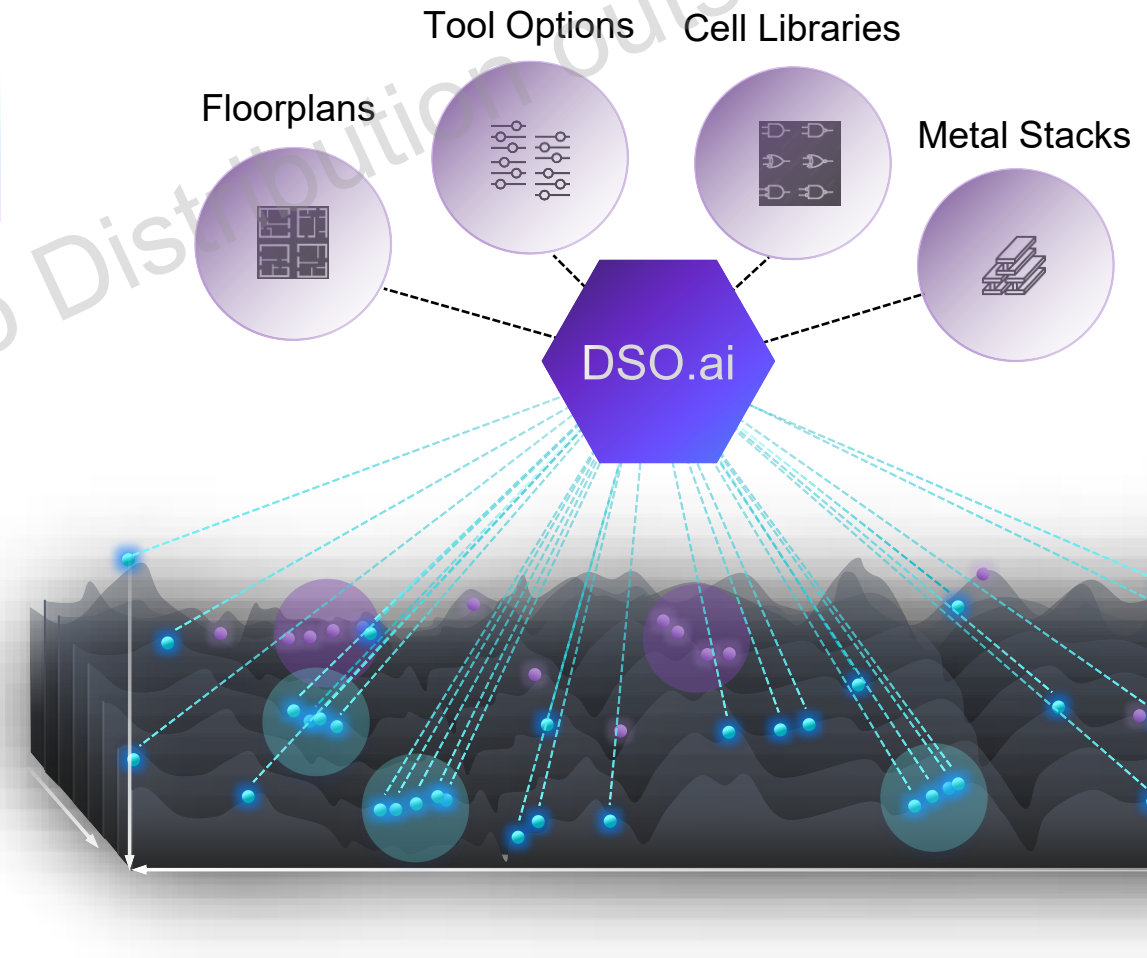
AI-DRIVEN IMPLEMENTATION

Limitless opportunity to leverage AI for EDA productivity

Exploration
Broader Scope



Floorplan / Memory
Metal Stack / Library



Synopsys DSO.ai — AI-driven Digital Design

Breakthrough Reinforcement Learning

Capable of searching trillions of RTL-to-GDSII design permutations for best results

Fast Ramp-Up

Fully-integrated with the Digital Design Family—industry's richest technology foundation

Cloud-Ready

Supports on-prem, public, and hybrid clouds for fast deployment

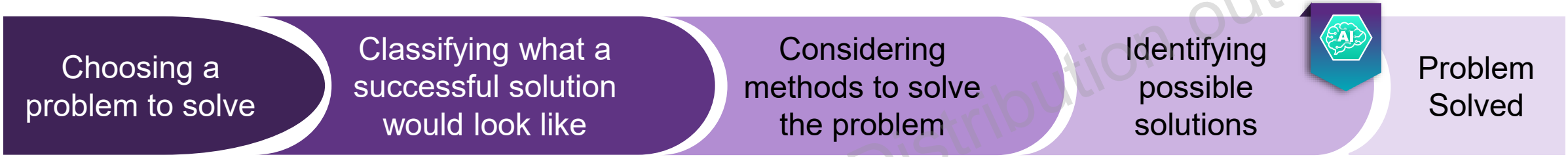
World's First Autonomous Design Space Optimization

DSO.ai Success: Any Node, Any Type

Event	Customer	Design	Benefits
SNUG IN 2024	**Samsung**	Mobile SoC	11% dynamic power
SNUG TW 2024	**Realtek**	Networking	12% floorplan shrink
SNUG EU 2024	**Infineon**	Automotive	70% WNS, 80% TNS, 65% congestion improvement
SNUG SV 2024	**Intel**	GPU	**7%** lower total power; ~75% of total die area
SNUG SV 2024	**Synaptics**	ARC CPU	20% less WNS; 60% less TNS; 20% less leakage
SNUG SV 2024	**Intel**	CPU	2-4 weeks schedule reduction; 10% timing improvement
SNUG SV 2023	**ST Micro**	Arm CA510	3X productivity
SNUG SV 2023	**Microsoft Azure**	HPC SoC	4.5% Fmax; 4% lower total power (On Cloud)
SNUG SV 2023	**Microsoft**	HPC CPU	3% total power
SNUG SV 2023	**Intel**	x86 XEON CPU	6% Total Power
CDC 2023	**UniSoC**	Arm CPU	1.6X faster time-to-target
CDC 2023	**ESWIN**	RISC-V CPU	5% Fmax; 5% Power; 5% Area
SNUG SC 2022	**Synopsys IP**	PCI-E X16	6% die shrink; 70% less TNS; 1.5% IR-drop
SNUG SV 2022	**Samsung SLSI**	Arm CPU	13%-80% Fmax; 25% Power
SNUG SV 2022	**Samsung Foundry**	RISC-V CPU	70% Less TNS (On Cloud)
SNUG SV 2022	**Intel**	X86 CPU	40% Less TNS; 20% Faster TAT
SNUG SV 2022	**Mediatek**	Mobile SoC	6.5% smaller area
SNUG SV 2022	**Sony**	Image Sensor	12% Smaller Area

Break Down of Problem-Solving

- Problem-Solving is a series of tasks that consists of:



- An AI can efficiently identify “better” possible solutions
  - The AI is considered an agent in this step of the process
- Users of AI tools control the direction the AI searches
  - The user must understand the other steps in the problem-solving process to direct the AI agent

Choosing A Problem To Solve

- Consider solving the problem of “Personal Transportation”

Choosing a problem to solve

- This problem has many possible solutions to search for
- Will use as example to walk through the steps of problem solving
- An AI agent can provide efficiency in the search process

Classifying a Successful Solution

- For this problem, what does an optimal solution look like?

- To compare possible solutions, need quantitative values to measure
- The optimal personal transportation solution might have:
  - Low cost
  - High speed
  - High safety
- Trade-offs exist, but some solutions can be universally better

Considering Methods to Solve the Problem

- Considering the possibilities constructs a search space to explore

- Possible choices might be:
  - Style of vehicle (pedestrian, cycle, motorcycle, automobile)
  - Repository of listing (personal network, dealership, online)
  - Manufacturer of vehicle (you, known brand, unknown brand)
- Introducing new dimensions increases the total search space available

Identifying Possible Solutions

- After establishing the direction and extent of search space:

- The process of finding a solution can begin
- A large search space can be efficiently navigated using an AI agent
- The AI agent will determine what possible solutions to evaluate

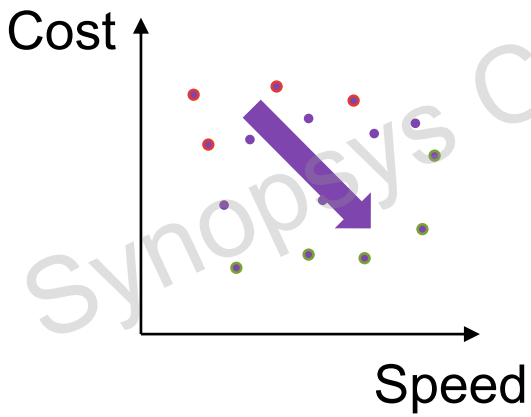
Problem Solved

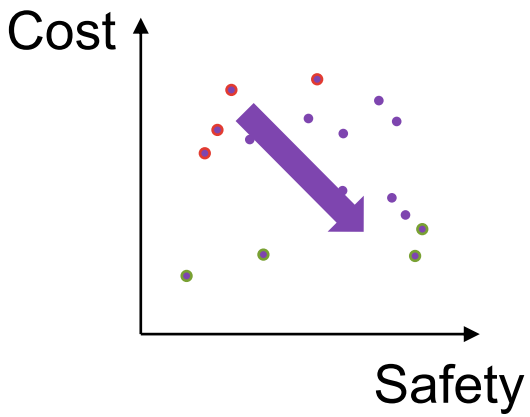
- After identifying enough possible solutions, the problem is solved!

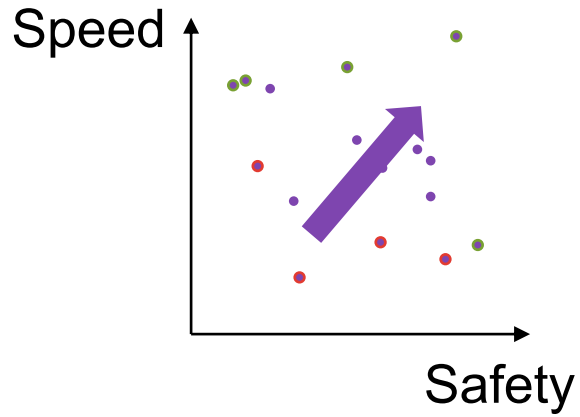
- The AI agent will have converged towards a usable solution
- This solution can be taken and applied for immediate benefit
- The knowledge about how the possible solutions performed can also be applied

Establishing Direction of Solution Space

- In the Personal Transportation example, these metrics are stated as desirable:
  - Cost ↓ Minimize
  - Speed ↑ Maximize
  - Safety ↑ Maximize
- While there are tradeoffs between these metrics, some solutions are clearly better than others
  - This is the direction that an AI agent will search

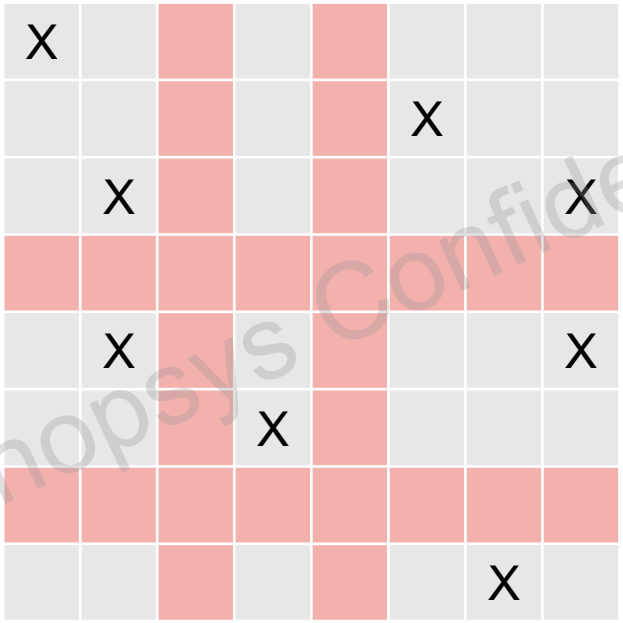






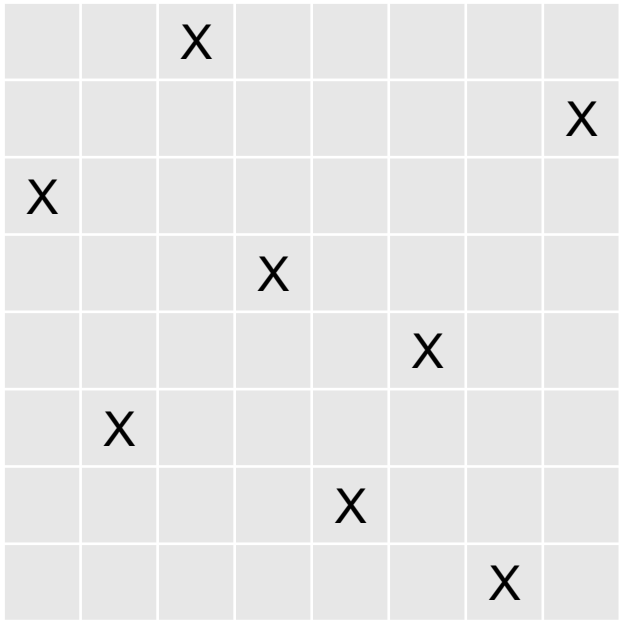
Exploration of a Search Space

- When a search space is first presented to an AI agent, exploration is required to understand this space
- Randomly sampling a search space can leave unexplored regions
- Uniform sampling is more efficient to ensure entirety is covered



Random

Exploration
Of
2 Dimensions

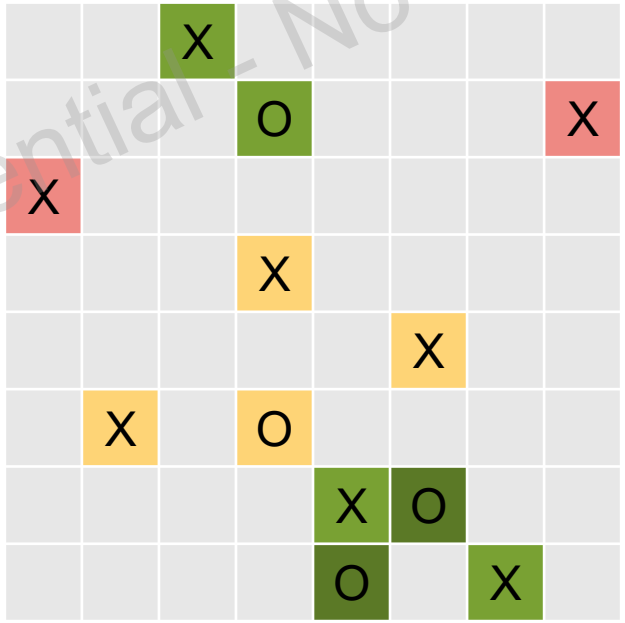


Uniform

Synopsys
Search
Optimization
Tools

Learning on a Search Space

- Initial sampling of search space explores possible solutions
- Solutions can be compared to each other based on targeted metrics
- Direction to converge on is established
- AI agent searches in the direction of the better results

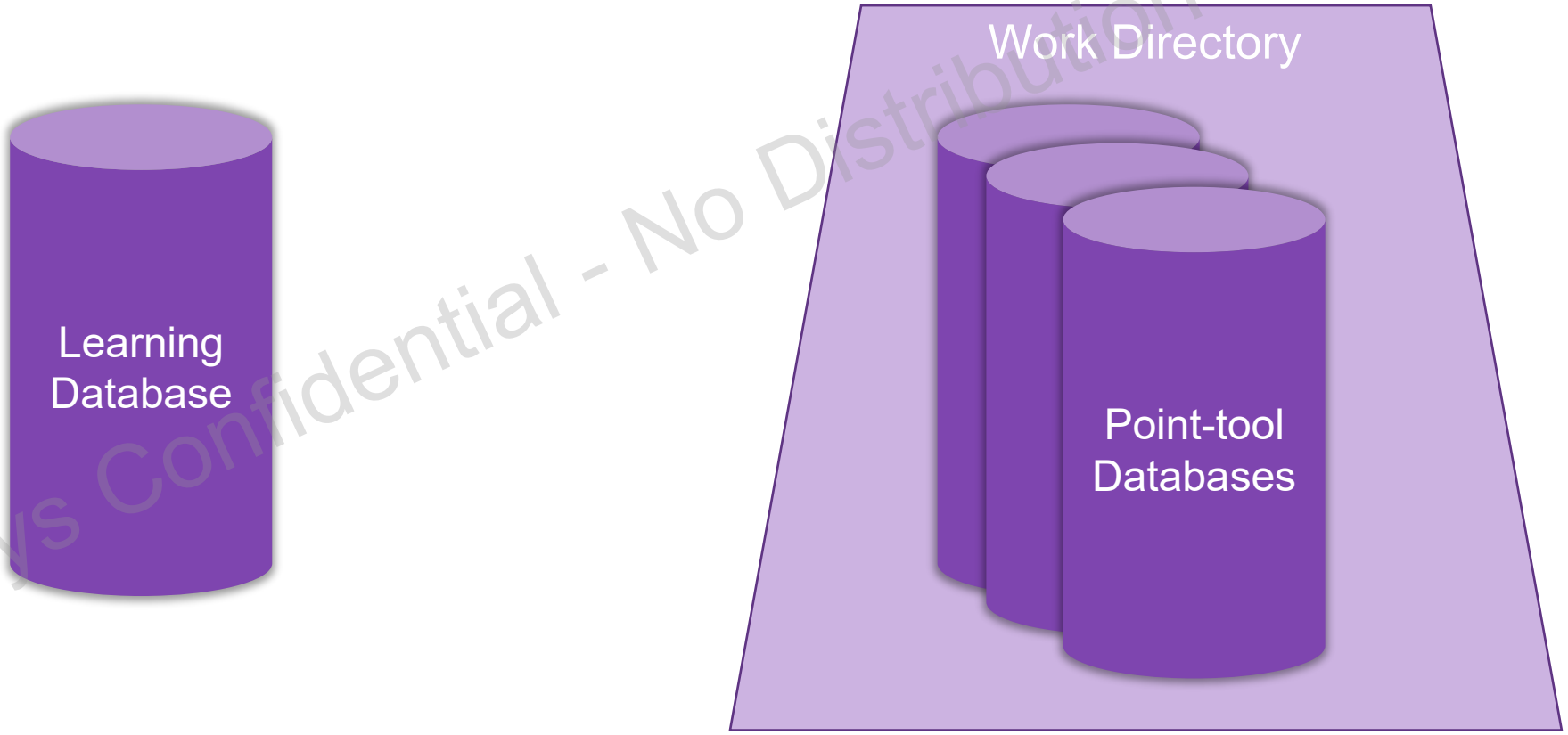


X – Initial Samples
O – AI Suggestions

Poor
Okay
Good
Great

Where is the Data?

- Types of data handled by Synopsys AI Search Optimization Tools:

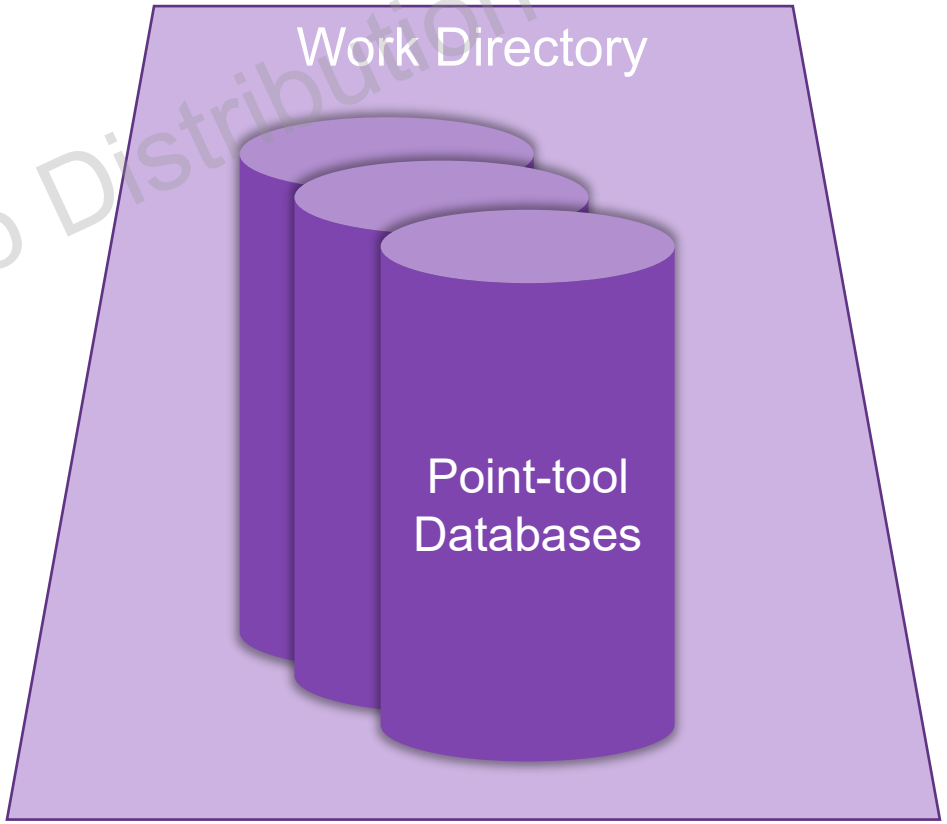


Learning Database

- The Synopsys learning databases contains the record of results
- Allows this information to be applied as learning to other projects
- Available for users to query information that was collected from the possible solutions

Work Directory

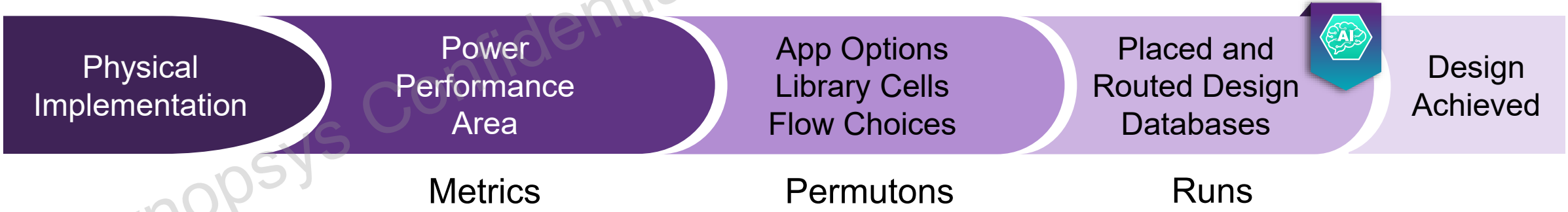
- The work directory contains all the point-tool results
- The work directory should be maintained while the Synopsys AI Search Optimization tool is running
- Upon completion, extract desirable point-tool results
- After extraction, safe to clear work directory



Problem Solving in Physical Implementation

- Previous example:

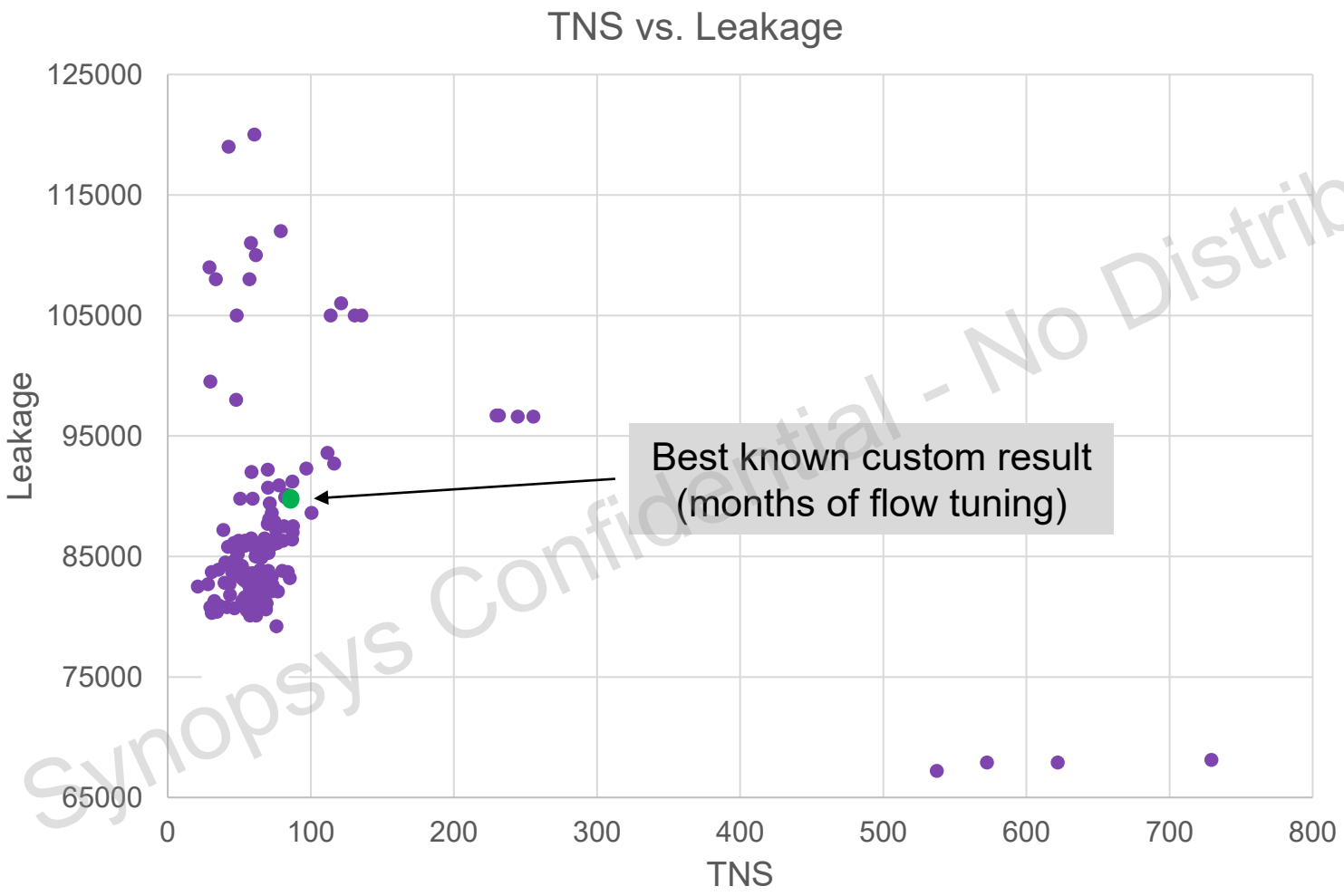
- DSO.ai can be applied to solve physical implementation problems



DSO.ai is a Search Engine that Searches for Improved Results

- DSO.ai searches **parameter space** looking for better results
  - **Parameter space** are inputs that impact PPA (design settings, tech, lib, tool, etc.)
  - “better” is determined using multiple metrics (TNS, area, power, etc.)
- DSO.ai outputs better point-tool results (better databases)
- Both inputs and outputs are same as traditionally used
  - DSO.ai uses tool scripts (largely) unmodified
  - DSO.ai driven by standard metrics (TNS, area, power, DRC, etc.)
- DSO.ai uses ML to efficiently search large spaces (10^9 is not uncommon)
  - Enables engineers to search **more input parameters** while targeting **more output metrics**
  - Frees engineers from the drudgery of manual search (setup runs, copy data, launch jobs, etc.)

DSO.ai Example Search



Problem Statement:

Designer wants to achieve lowest power while keeping TNS <-100ns

DSO.ai Search Space:

- Design, tool, flow permutons
- Library cell permutons

Objectives:

- Leakage
- TNS
- Plus secondary (DRC etc)

Compute Envelope

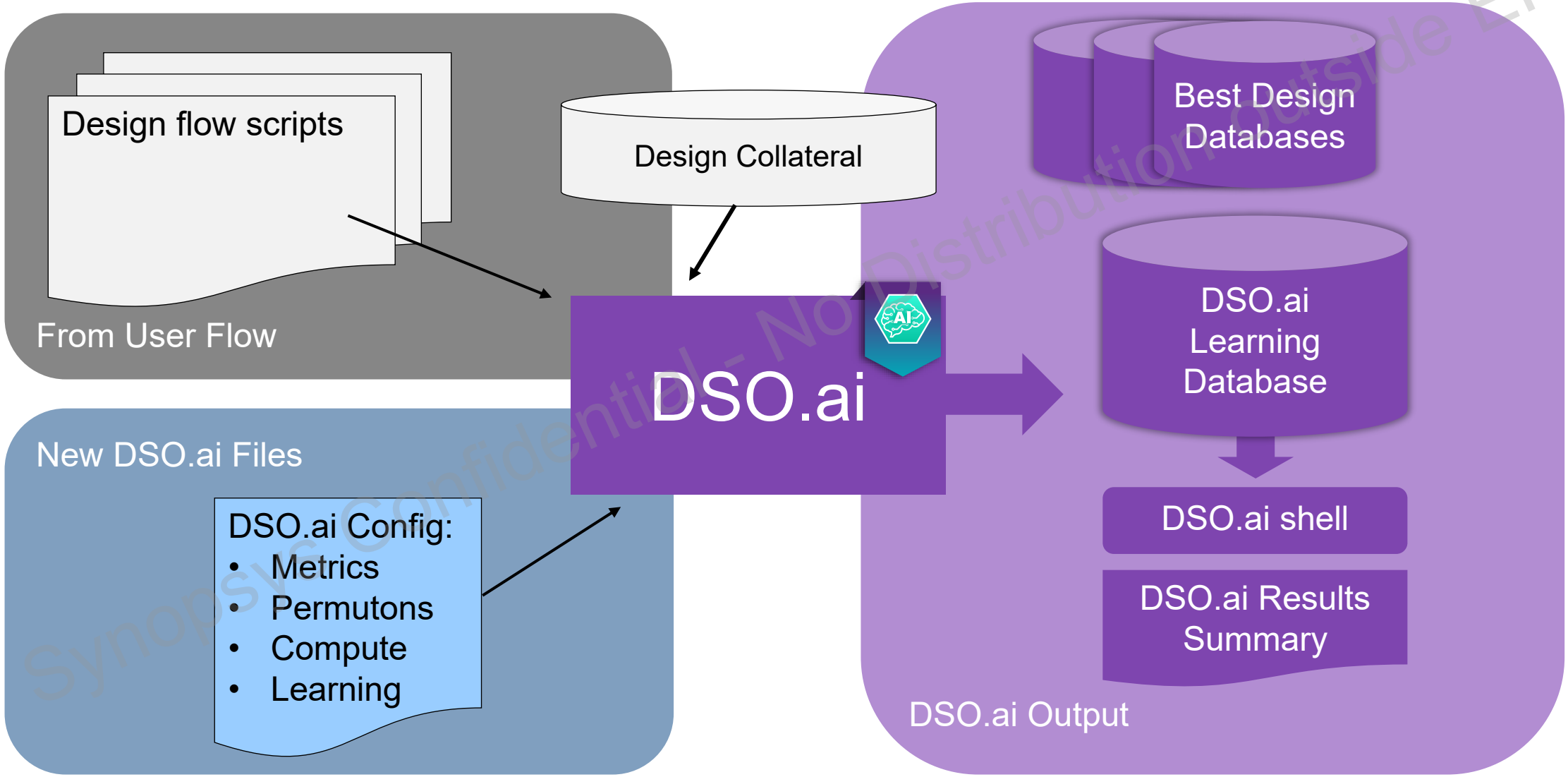
- 30 parallel jobs, ~1 week

DSO.ai Inputs and Outputs

- DSO.ai inputs
  - A working baseline flow (tool and TCL scripts)
  - An objective (metrics)
  - A search space (design settings, tech, lib, etc.)
  - Additional search instructions (compute, runtime limit, etc.)

- DSO.ai outputs
  - Best point-tool results (best TNS database, best power database, etc.)
  - Analytics (reports, graphs, etc.) that show the quality of the search space
  - Database records for future learning
    - Every search result captured in a database for later use
    - Can be applied to future searches through learning

DSO.ai Workflow



What Does DSO.ai Not Do?

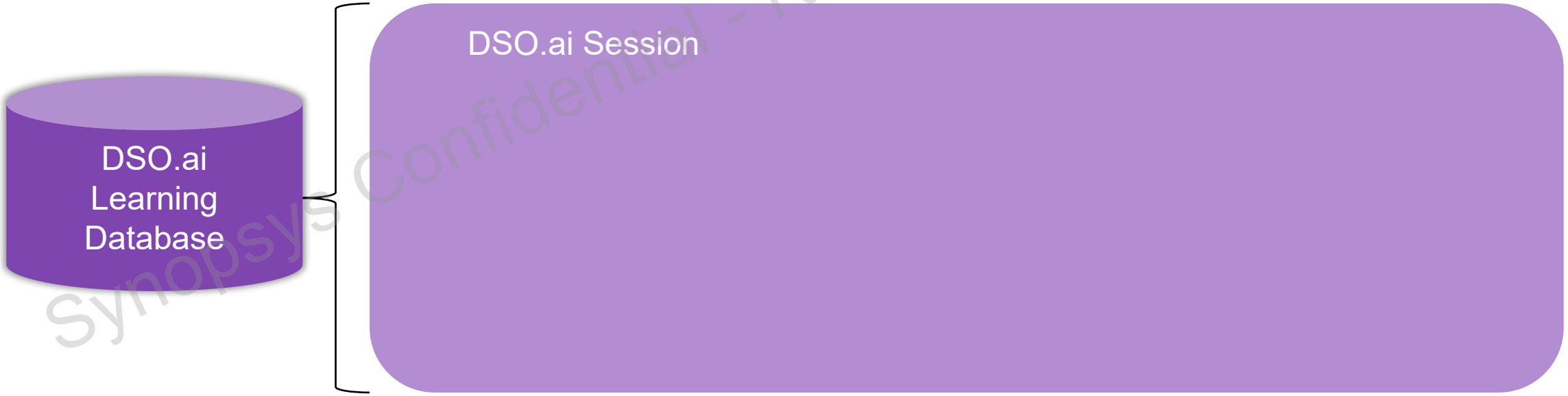
- Automatically create a flow from nothing
  - User must provide a working baseline flow
  - Flow must run without failing (does not have to produce great PPA)
  - Can be as simple as `compile_fusion` or as complex as Reference Methodology scripts
- Automatically debug design/flow problems
  - Floorplan too small, macros overlap, flow script Errors…
  - Incorrect constraints impossible to meet
  - Library cell flaws make pins unroutable
- Automatically create the perfect search space for any design
  - Good search space construction requires design and SNPS tool knowledge
  - DSO.ai is a tool that helps engineers be more efficient
- Require you to use enormous compute; use whatever compute you like

DSO.ai is Not a “sweeper”

- DSO.ai can be used to execute “exhaustive” searches
  - The only benefits are run automation and ADES ranking
- But using DSO.ai as a “broom” is a waste
  - You’re not leveraging the power of ML
  - You’re using DSO.ai to do something you could easily do yourself
- Use DSO.ai when you want to find better PPA in a large search space

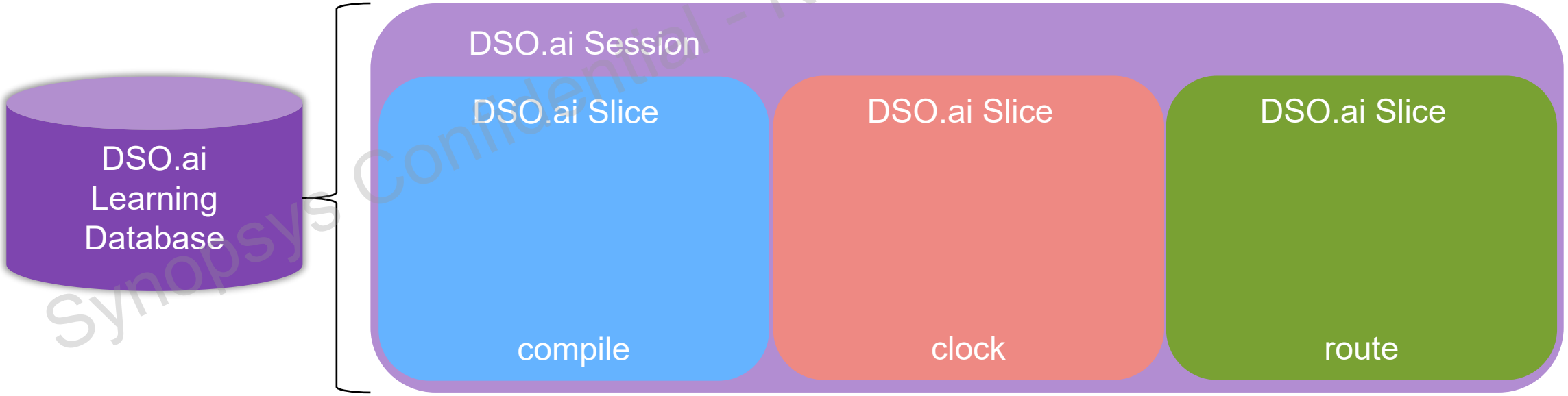
DSO.ai Sessions

- The primary container of information within DSO.ai is a Session
  - Each DSO.ai database can contain many Sessions
- Initially a Session has a `setup` status
- The 4 inputs of DSO.ai are configured in this state (Metrics, Permutons, Compute, Learning)
- After a Session “runs” it stores data available for later access



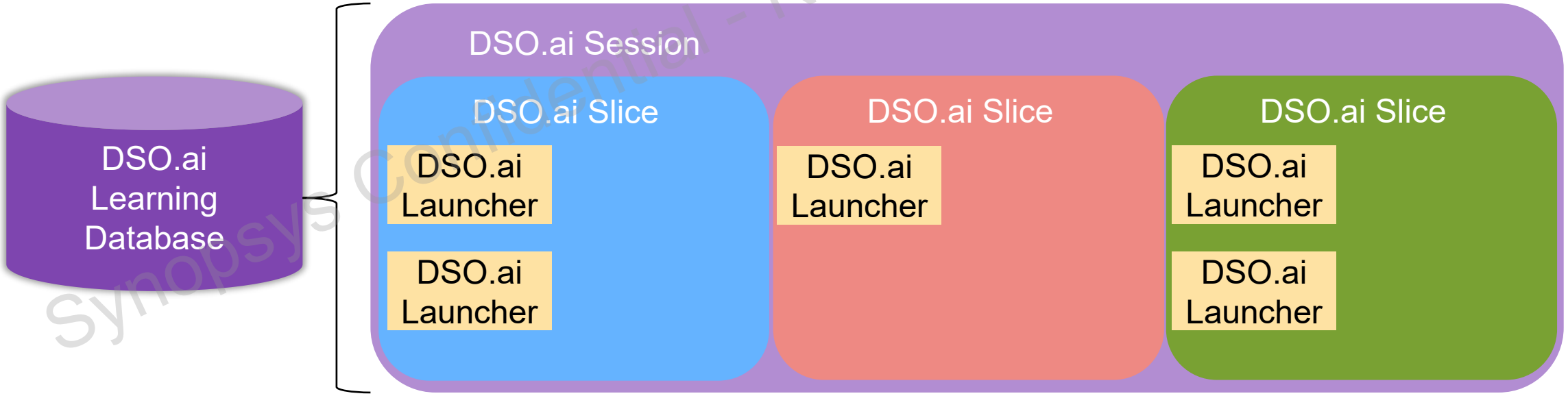
DSO.ai Slices

- Each Session is created with specific Slices
- Each Slice should contain search space relevant to the implementation stage
- DSO.ai supports arbitrary Slice definitions
  - These should relate to the implementation flow and be consistently named in a project
  - Suggested to match the stages: “compile” or “place”, “clock”, and “route”



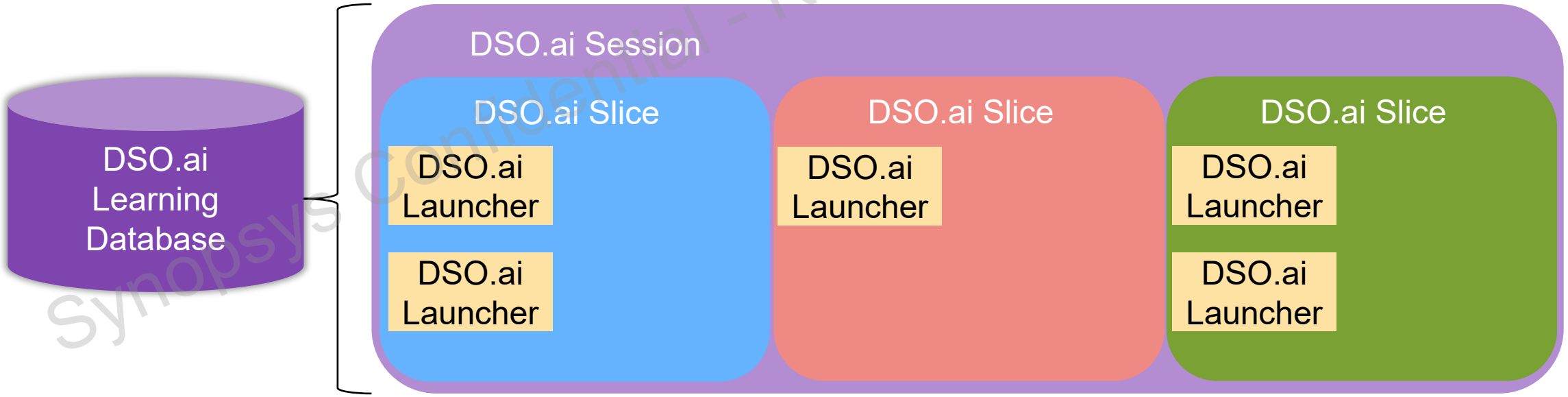
DSO.ai Launchers

- The control of the flow is determined through Launchers
  - The flow will perform in the order the Launchers are created within each Slice
- Each Launcher specifies the point-tool being used and scripts to perform
  - Multiple Launchers can be specified within the same Slice to keep scripts organized



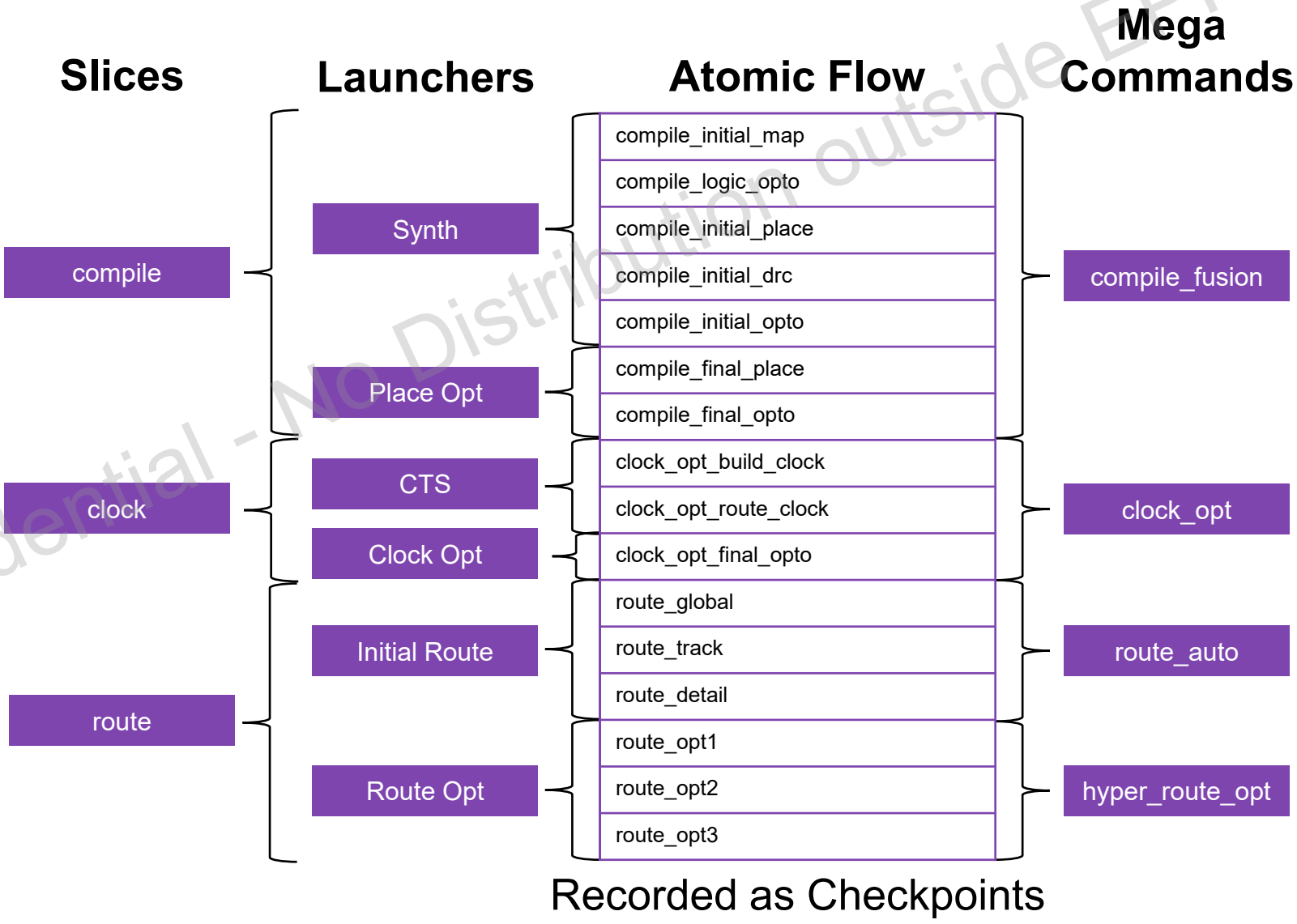
DSO.ai Checkpoints

- When the flow is performed, Checkpoints that occurred will be recorded
  - Ex: `compile_initial_place`
- Each Slice keeps track of Checkpoints separately
- Each Slice has a final Checkpoint of `DSO_final`



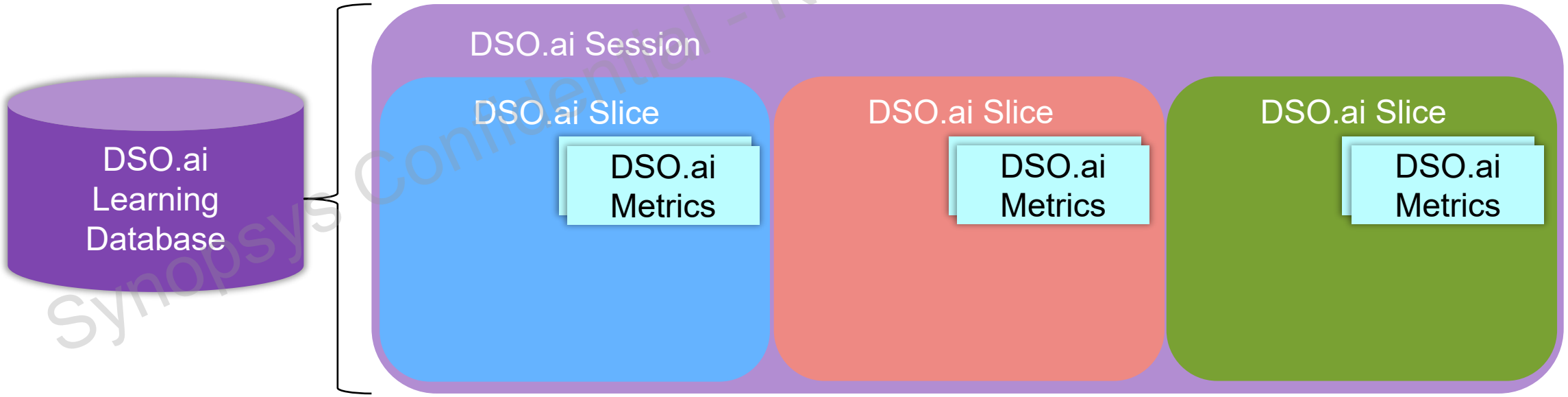
Example DSO.ai Flow Slicing Strategy

- Start with a baseline flow
- Establish Launchers that can execute the flow
  - Each Launcher should end with a saved block for easier flow recovery
  - Trade-off between ease of recovery and disk usage
- Combine Launchers into Slices where the search space is shared
  - Each slice **must** have a final saved block in the last launcher



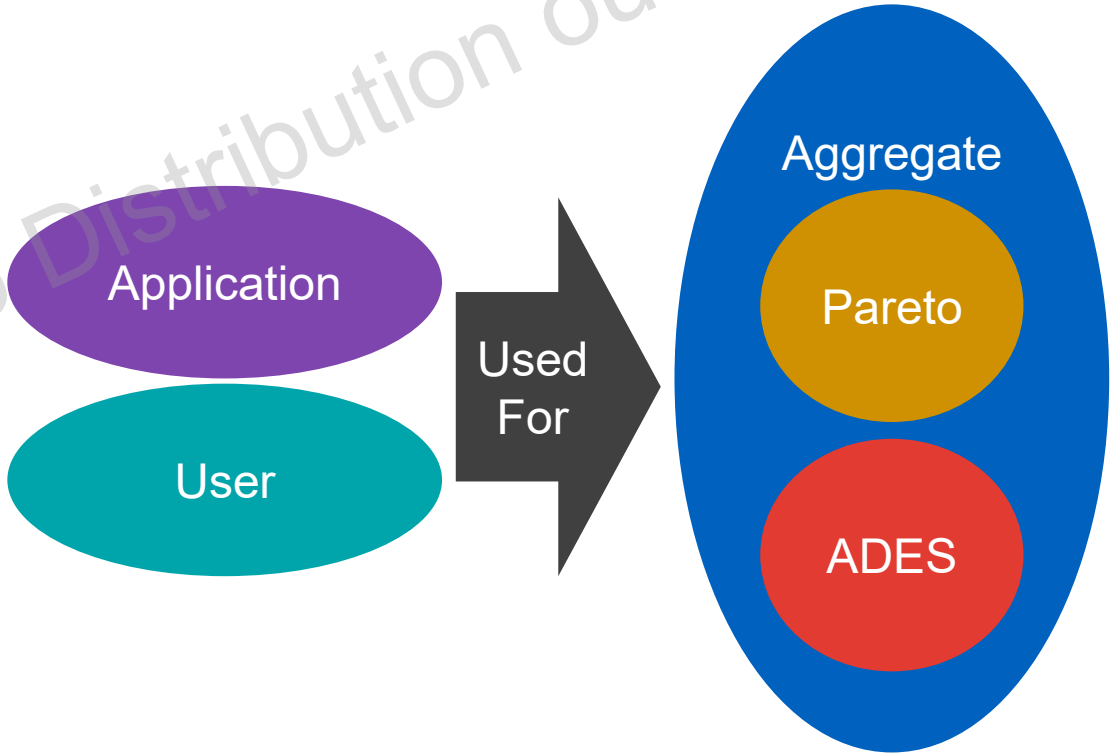
DSO.ai Metrics

- The quantitative measures that will be recorded are defined as Metrics
- Each Slice has Metrics that are recorded at specified Checkpoints
  - Commonly collected at the `DSO_final` Checkpoint
- Each Run in each Slice will collect a value for its Metrics at enabled Checkpoints



Type of Metrics

- Application
  - Provide common definitions for implementation metrics
- User
  - Can be defined within each Slice to collect additional information
- Aggregate
  - User-defined Metrics that combine multiple component Metrics
  - Pareto
    - Provide a ranking of which “Pareto front” the run is on
  - ADES (Aggregate Design Score)
    - Provide a scoring by combining the components through calculation



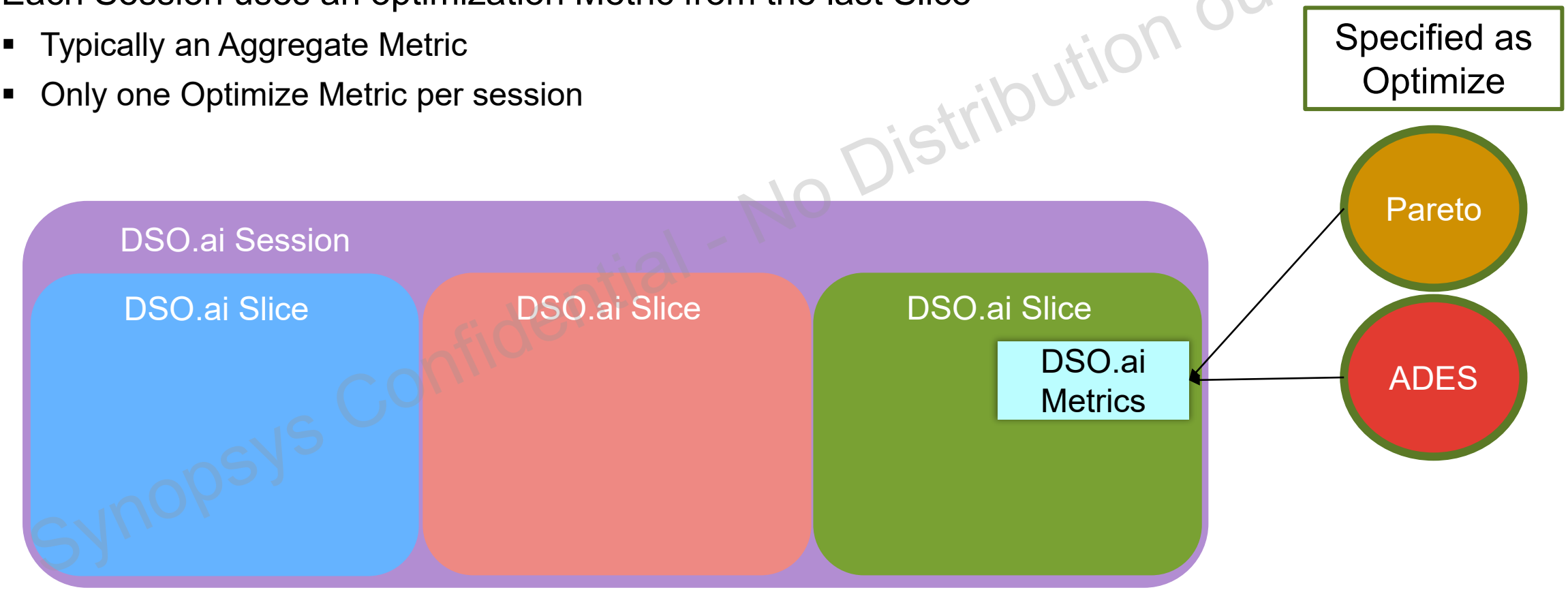
DSO.ai Setup – Sample ‘Application’ Metrics

Application Metric Name	Meaning
WNS	Worst setup slack
TNS	Total negative setup slack
NVP	Number of endpoints failing setup
FREQ	Frequency of worst path
STDCELL_AREA	Standard cell area
WIRELENGTH	Total wirelength
TOT_DRC	Total number of DRCs
SHORT_DRC	Total number of Shorts
CONGESTION	Congestion (% overflow)
TOT_RUNTIME	Total runtime
UTILIZATION	Design utilization (%)
MEMORY_USAGE	Peak memory usage
REP_AREA	Repeater area
REP_COUNT	Number of repeaters
BUFFERS	Number of buffers
INVERTERS	Number of inverters
H_BUFFERS	Number of hold buffers
MAX_OVERFLOW	Maximum gcell overflow

Application Metric Name	Meaning
HWNS	Worst hold slack
HTNS	Total negative hold slack
HNVP	Number of endpoints failing hold
TOT_POWER_STD_CELL	Total power of standard cells
TOT_POWER_CLK	Total power of clock trees
INT_POWER_STD_CELL	Internal power of standard cells
INT_POWER_CLK	Internal power of clock trees
SW_POWER_STD_CELL	Switching power of standard cells
SW_POWER_CLK	Switching power of clock trees
LEAKAGE_STD_CELL	Leakage power of standard cells
LEAKAGE_CLK	Leakage power of clock trees
LVT	Percent of LVT cells
SVT	Percent of SVT cells
HVT	Percent of HVT cells
TOTAL_POWER	Total power of entire design
LEAKAGE	Leakage power of entire design
INT_POWER	Internal power of entire design
SW_POWER	Switching power of entire design

Optimization Metric

- Metrics can be used to inform the AI Agent what to seek to improve
- Each Session uses an optimization Metric from the last Slice
  - Typically an Aggregate Metric
  - Only one Optimize Metric per session



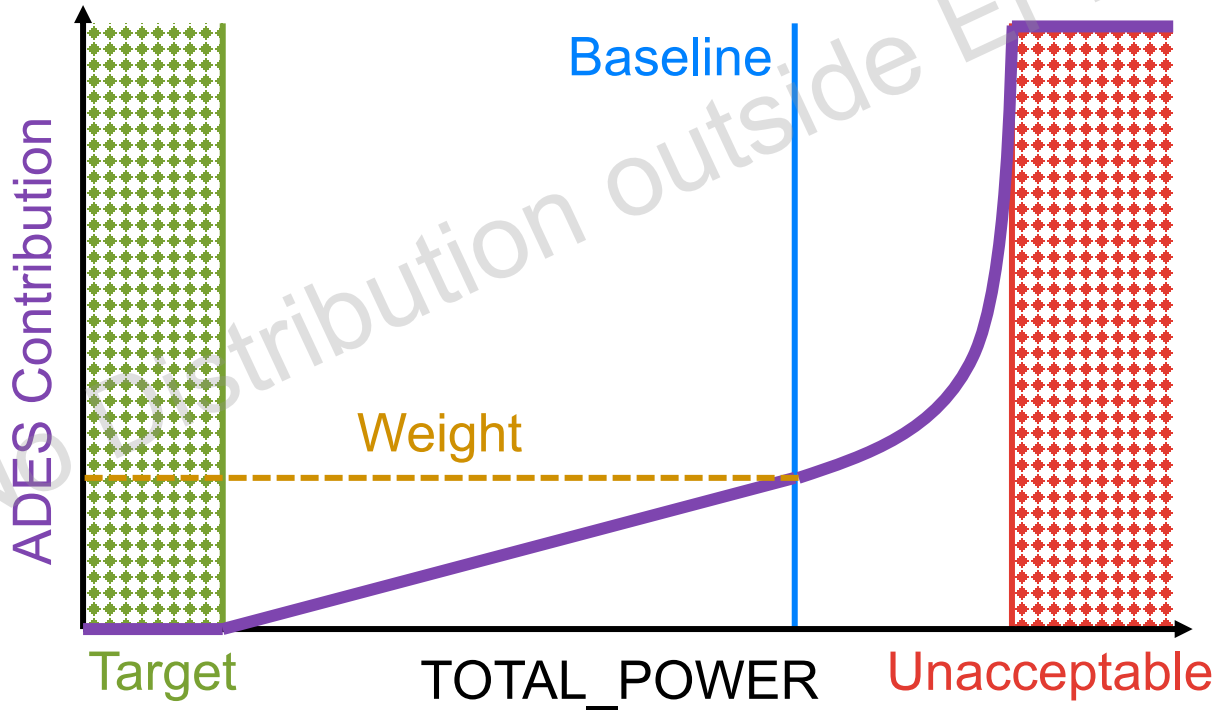
ADES Drives the DSO.ai Search Process

- DSO.ai optimizes multiple metrics through Aggregate DEsign Scoring (ADES)
  - ADES is made up of metric components
  - Each metric component has independent settings
  - Settings control metric targets and tradeoffs
- Application metric components have defaults for all settings
  - You can only specify their name if you like
- Custom metric components have fewer default settings
  - Must at least specify their name, Tcl proc, direction, target and baseline
- DSO.ai tries to minimize ADES
  - The best possible ADES is 0.0

```
create_aggregate_metric \
  -name ADES \
  -component TNS \
  -component SHORT_DRC \
  -component WNS
```

Creating Aggregate DEsign Score Metrics

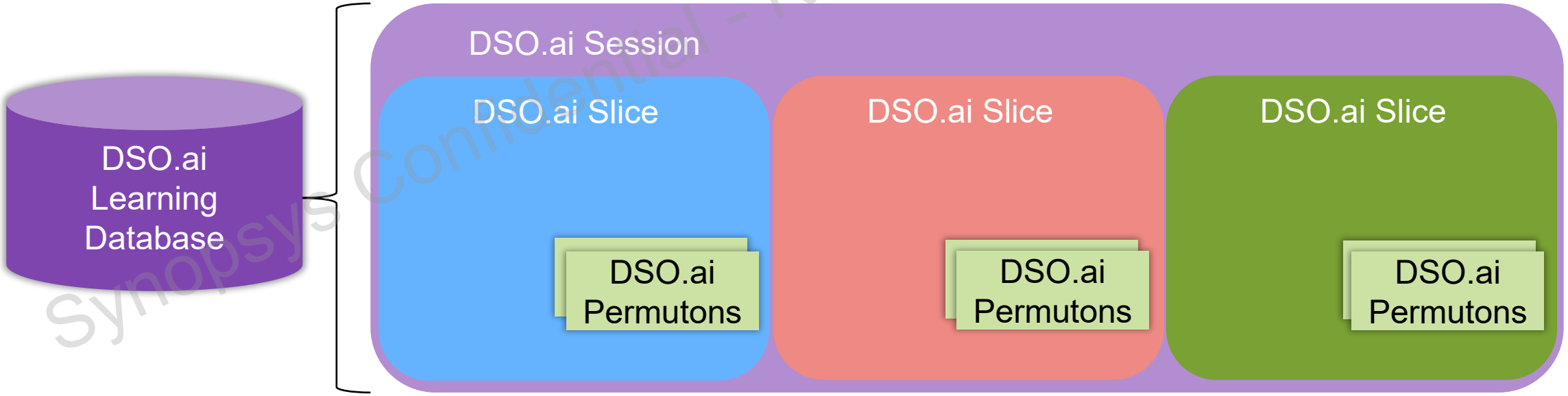
- Smaller ADES value represents better
- Weight
  - Scaling factor for the component
- Target
  - Completely satisfies the design requirements
- Baseline
  - Already known to be achievable
  - Can come from the baseline Run automatically
- Unacceptable
  - Design would be infeasible
- Value from each component is summed to calculate ADES value




```
create_aggregate_metric    -name simple_ades \
    -type ADES                               \
    -component TOTAL_POWER                   \
        -weight       3                      \
        -target       10000                  \
        -baseline     12345                  \
        -unacceptable 15000
```

DSO.ai Permutons

- The methods available to solve design problems are defined as Permutons
  - Permutons “permute on” the search space for Runs
- Each Slice can contain multiple Permutons
  - These dimensions form the search space of each Slice; together for the Session
- Each Permuton has a range of possible values



Permuton Types and Datatypes

Three types of permutons:

- Application
  - Controls tool behavior (eg: `set_app_options`)
- Global
  - Global Tcl variable used directly in tool driver scripts
- Custom
  - Completely defined by your own Tcl procedures
  - Can be applied at specific flow stages
  - Can search complex things like stage-specific setup margin, library cells, floorplans, etc.

Permuton datatypes:

- Continuous
  - Floating point between and including min and max (0.0 5.0)
- Discrete
  - Specific numerical values in order (0 1 2)
- Categorical
  - Specific named string values, no implied order (low high medium)

Permuton Types: Application and Global

- Application (use `-type app`)
  - Example Fusion Compiler application option

```
dso_shell> create_permuton -type app -name opt.timing.effort -datatype categorical -range {medium high}
```

- Global (use `-type global`)

```
dso_shell> create_permuton -type global -name DSO_core_utilization -datatype continuous -range {0.5 0.90}
dso_shell> create_permuton -type global -name DSO_side_ratio       -datatype continuous -range {0.5 2.0}
```

  - To use a global permuton, your tool runscript must reference it; for example:

```
fc_shell> initialize_floorplan -core_utilization ${DSO_core_utilization} \
                               -side_ratio [list 1 ${DSO_side_ratio}]
```

- Custom

```
proc congestion_placement {congestion_effort} {
    create_placement -incremental -congestion \
            -congestion_effort $congestion_effort
}
```

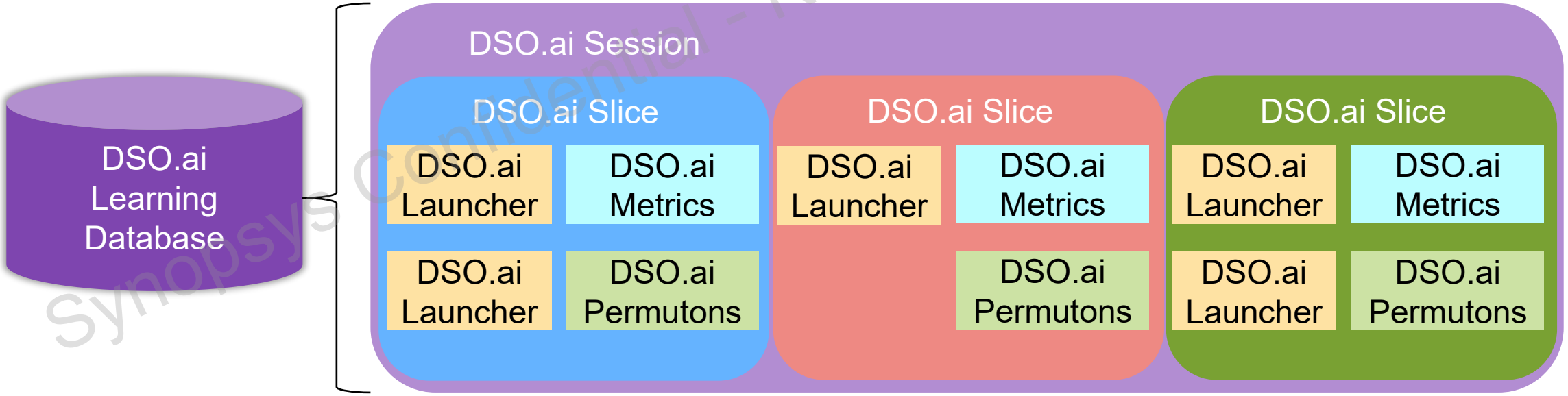

Permuton Constraints

- Some Permutons can have a known relationship between their values within the search space
- Permuton constraints enforce a constraint on each Run that the values of Permutons must satisfy
- Evaluated as Tcl-like expression using Permuton names as replaced variables

```
current_db -fs dso_db
open_session LAB_1234

## Shortcut for a new Session with same settings
create_session -from LAB_1234

current_slice compile
create_permuton -name dso_init_util              \
    -type global                                 \
    -datatype discrete                           \
    -range {0.79 0.80 0.81 0.82 0.83}            \
    -disable_value 0.80

create_permuton -name place.coarse.max_density   \
    -type app                                    \
    -datatype discrete                           \
    -range {0.79 0.80 0.81 0.82 0.83}            \
    -disable_value 0.80

set_permuton_constraint                          \
    {dso_init_util <= place.coarse.max_density}
```

DSO.ai Runs

- The results of each potential solution are kept as Runs
- Each Solution uses a value from each Permuton's range
- Each Slice has many Runs that performed all Launchers of the Slice
- Each Run records a Metric value for each Metric for enabled Checkpoints



Reporting Session Results

- To review the record of Session results kept in a DSO.ai database:

```
## dso_shell>
current_db -fs /path/to/dso_db
open_session LAB_fused
set_result_columns “AVE_FREQ UTILIZATION TOTAL_POWER TOT_DRC” -slices route
## This reports the top 5 results of the “route” slice according to application metric TOTAL_POWER
report_session_results -num 5 -sort TOTAL_POWER -slices route -anchor_baseline
```

Set the Current db to a specific path

Open the Session of interest from the db

Set the columns to display for the last slice

Columns set to display

```
Reporting top 5 result(s) from a total of 60 runs:
      AVE_FREQ    UTILIZATION    TOTAL_POWER    TOT_DRC    ID          BLOCK_SAVE
      --------    -----------    -----------    -------    --------    --------------
 567585100.002          66.41     1061124820          0    a8a98b9f
 580046255.790          63.73     1075658700          0    b79646a4    qor_strategy:0
 578239583.869          63.58     1076165850          2    46edd877
 571430974.972          63.09     1078738340          0    1a44c960
 604021311.316          63.44     1089010120          0    9ca12e35
      --------    -----------    -----------    -------    --------    --------------
 568908571.800          65.01     1133714930          1    671bd643    baseline
```

Run id for reference

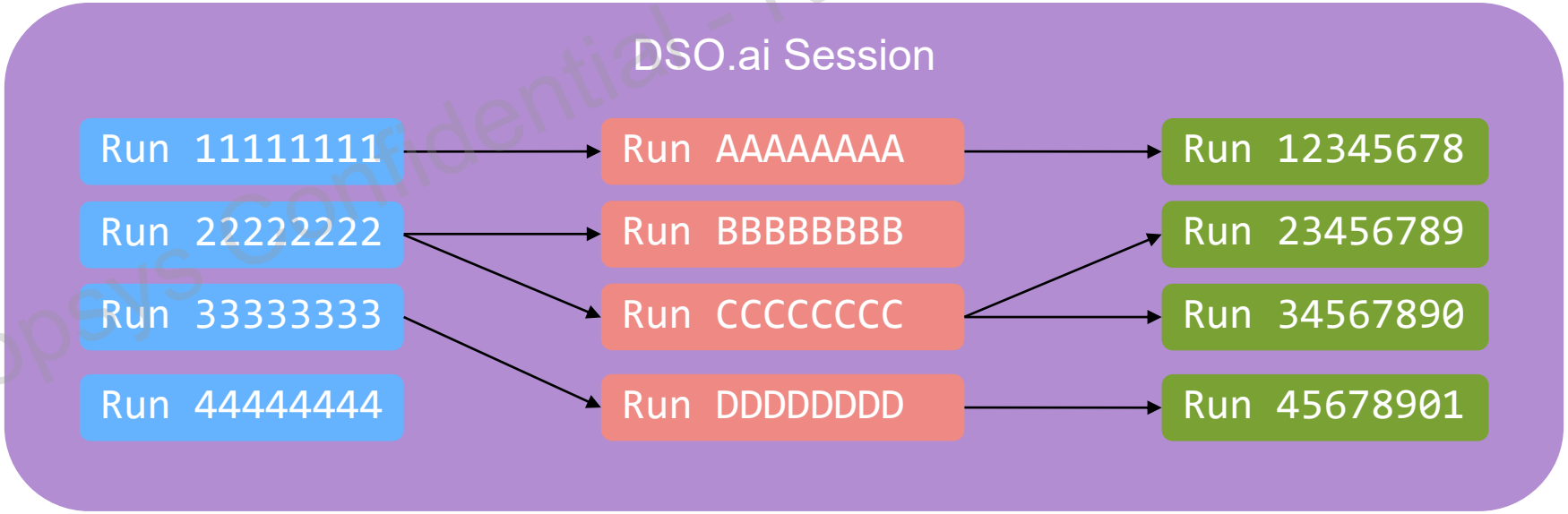
Value of Metric for Run

Key details about Run

Runs sorted according to TOTAL_POWER

Lineages

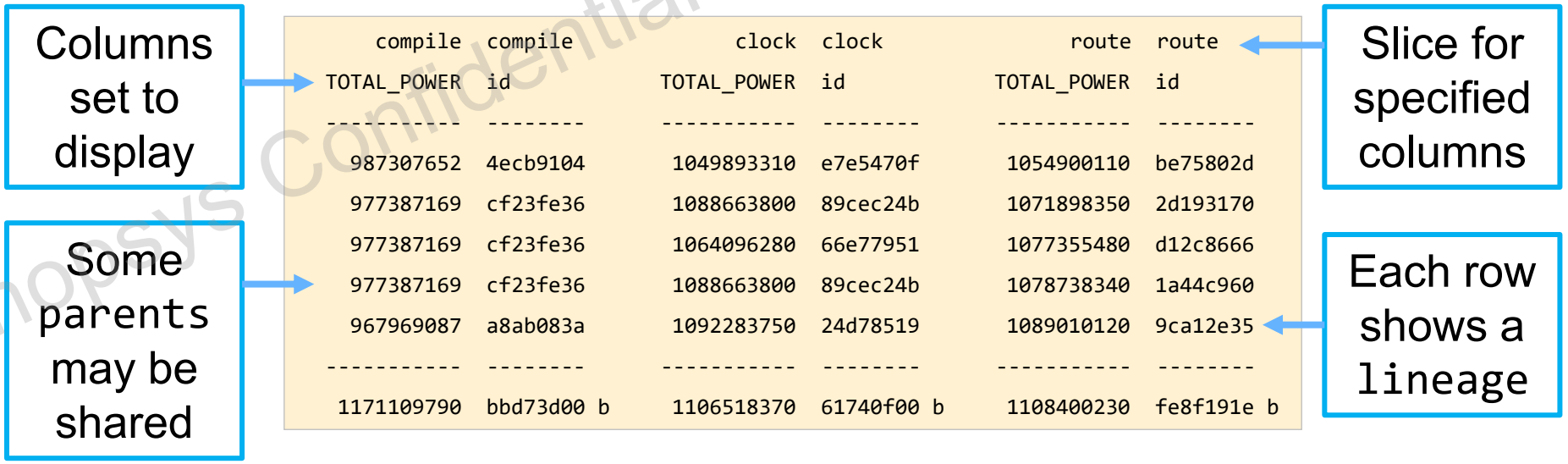
- The Runs of the final Slice have a `lineage` of previous Runs
- Each downstream Slice has a `parent` Run
- An upstream Run may be propagated multiple times
- The final results are optimized by selecting which Runs to propagate



Reporting Lineage Results

- To review the results as they span across the Slices:

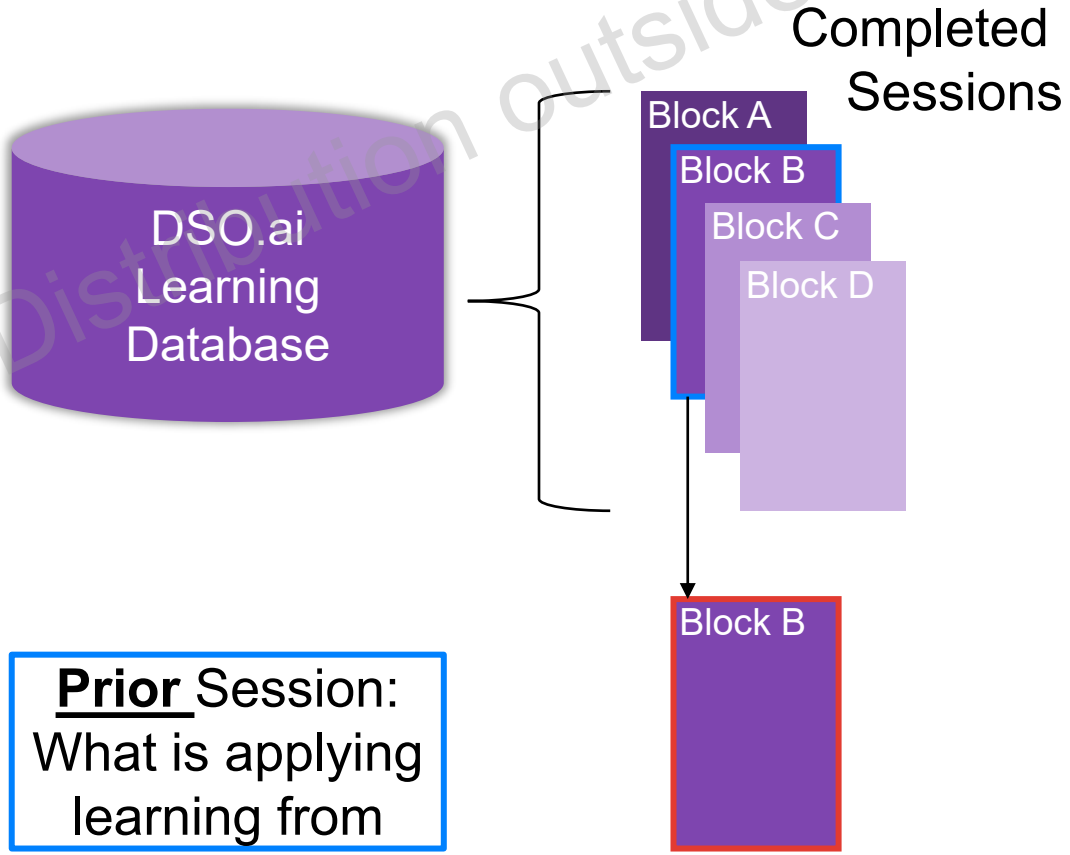
```
## dso_shell>
current_db -fs /path/to/dso_db
open_session LAB_fused
set_result_columns “TOTAL_POWER” -slices * -sessions *
## This reports the top 5 results of the “route” slice according to application metric TOTAL_POWER
report_lineage_results -num 5 -sort route:TOTAL_POWER -anchor_baseline
```



compile TOTAL_POWER	compile id	clock TOTAL_POWER	clock id	route TOTAL_POWER	route id
987307652	4ecb9104	1049893310	e7e5470f	1054900110	be75802d
977387169	cf23fe36	1088663800	89cec24b	1071898350	2d193170
977387169	cf23fe36	1064096280	66e77951	1077355480	d12c8666
977387169	cf23fe36	1088663800	89cec24b	1078738340	1a44c960
967969087	a8ab083a	1092283750	24d78519	1089010120	9ca12e35
1171109790	bbd73d00 b	1106518370	61740f00 b	1108400230	fe8f191e b

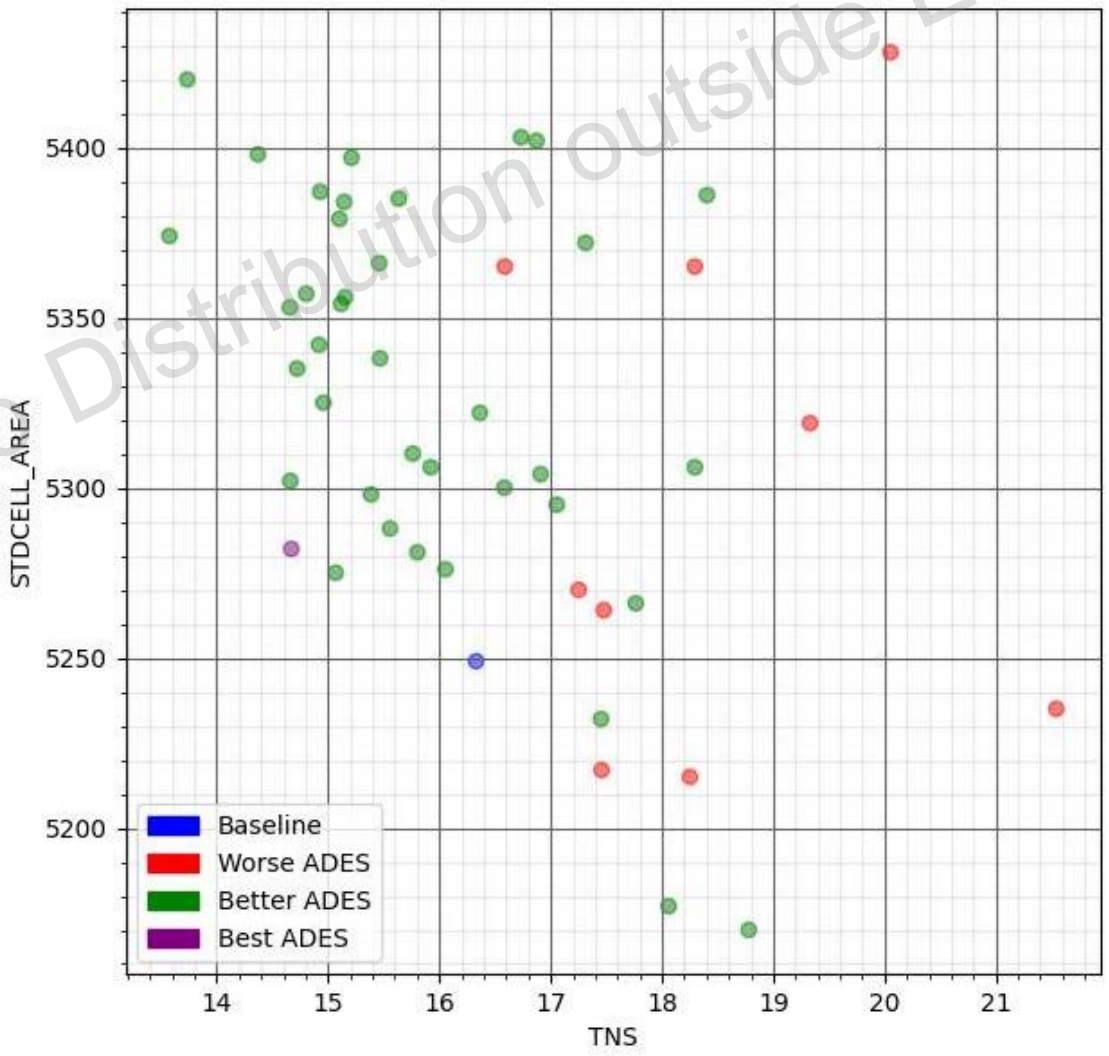
Applying Learning

- As DSO.ai Sessions are performed, more learning data becomes available in the DSO.ai database
- A completed Session can be used to apply learning into new Sessions
  - The completed Session is a “prior” Session
  - This new Session becomes a “guided” Session
- What information to learn from must be explicitly configured in new Sessions
- To achieve similar QoR with less compute requirements, provide the most “applicable” information available



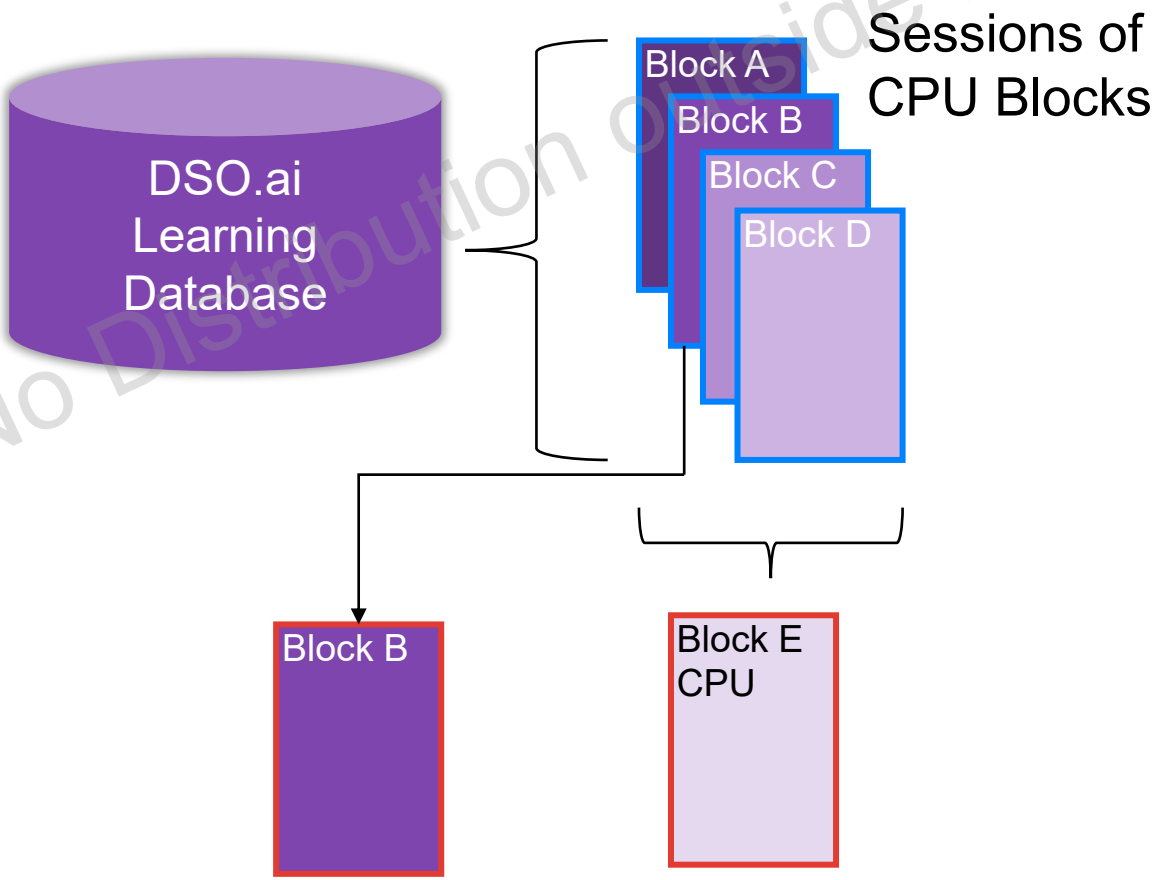
Selecting Information to Scale Learning

- An applicable prior Session is based on how similar, relevant, and good it is
- A “similar” prior Session is one that has
  - Same block or block type (CPU, GPU, …)
  - Same node
  - Same implementation tool version
- A “relevant” prior Session is one that has
  - Explored Permutons that are still in the search space
  - Optimized the same Metrics
- A “good” prior Session is one that has
  - Many Runs
  - Many Runs that beat the baseline Run



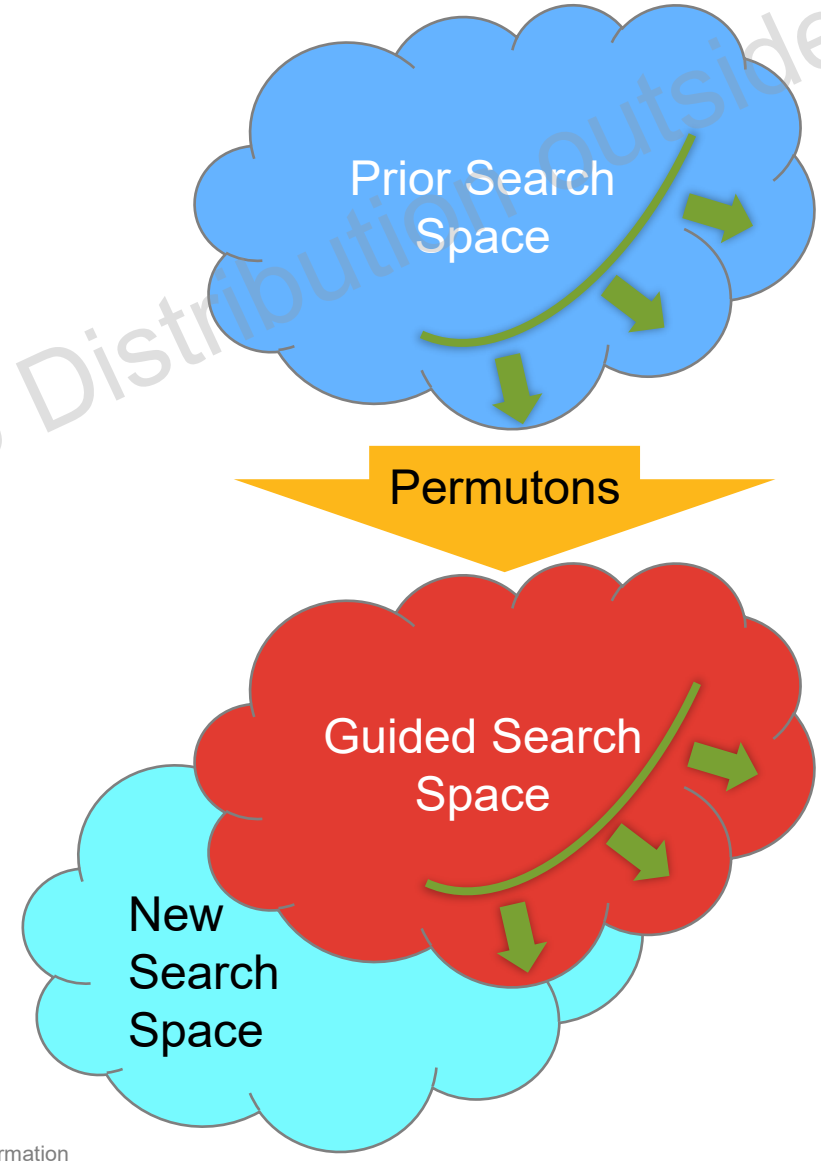
Similar Prior Sessions

- When there are multiple Sessions available to learn from:
  - Use the most applicable prior Session
- Can utilize multiple Sessions if no one Session is more applicable than others
- Ideally use prior Sessions that used same version of implementation tool



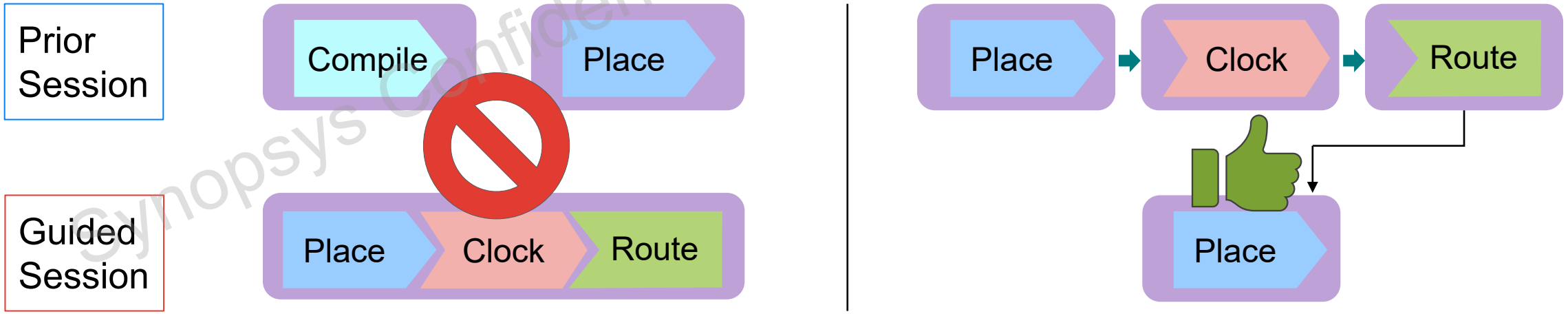
Relevant Prior Sessions

- The optimization can be different in a guided Session from its prior Session
  - Best practice is to use a prior Session that optimized relevant Metrics
- The search space can be different in a guided Session from its prior Session
  - Best practice is to use a prior Session that explored relevant search space
  - Automatically imported into the guided Session
  - Can be adjusted as appropriate
- Any new search space will not have learning but will be explored in the guided Session



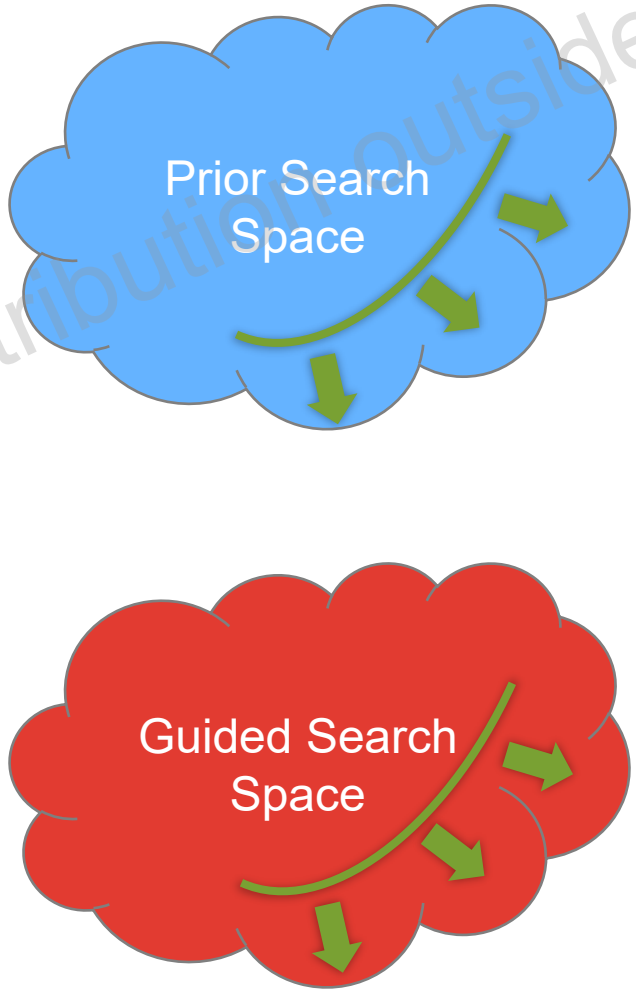
Slice Applicability of Applying Learning

- The names of Slices should align between a prior Session and the guided Session
- The last Slice of a guided Session must be in the prior Session or its lineage
- The new Session's Slices should contain a subset of the prior Session's Slices and its lineage
  - Guide a "place" Session with the results of a "route" Session (as it had fetched a "place" Slice)
- Use the most downstream results available to apply learning

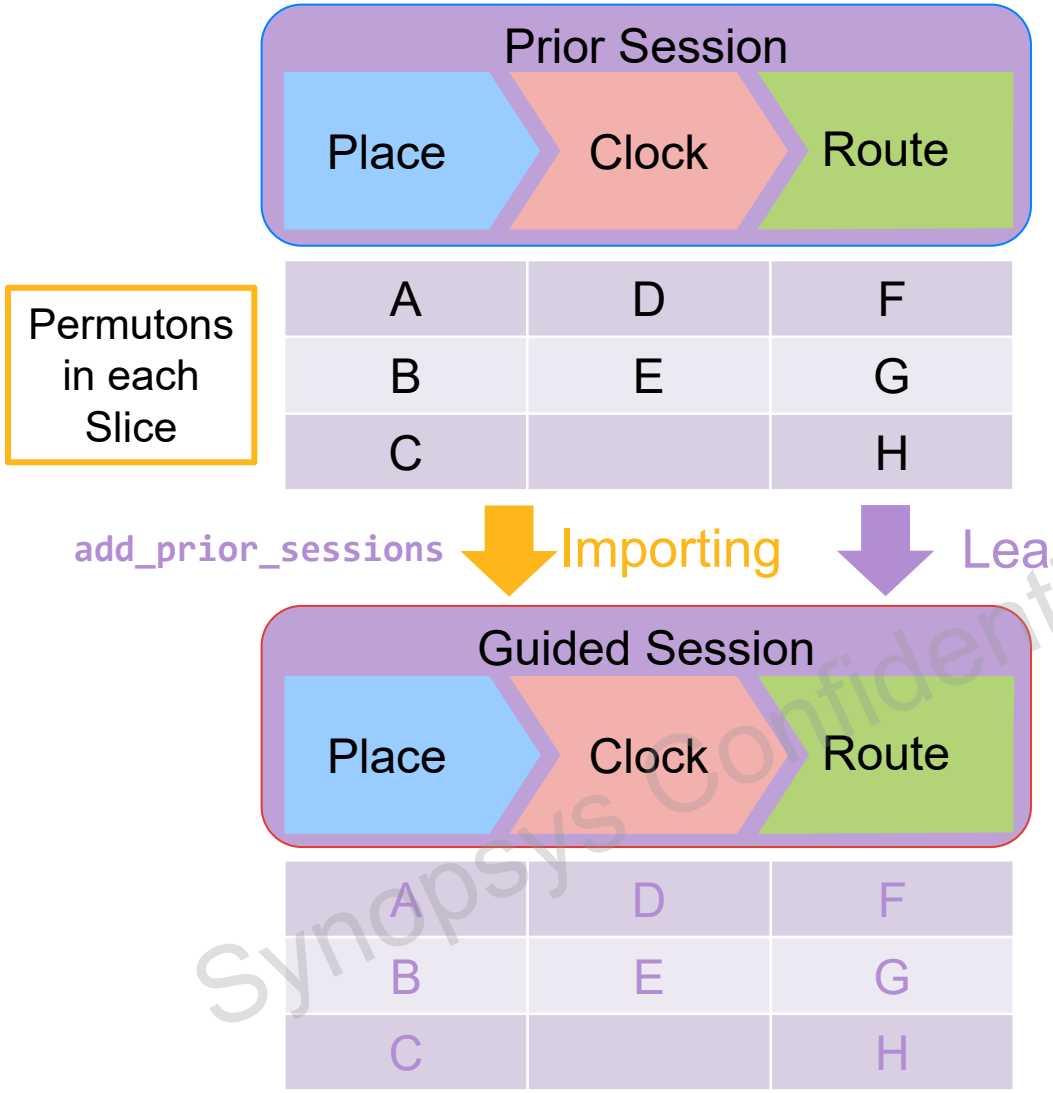


Performing Continued Search

- To apply learning from a prior Session into a guided Session:
  - Use the `add_prior_sessions` command
- For Permutons that are in both the prior Session and the guided Session:
  - Suggestions from the DSO.ai optimizer will be made instead of samples
- Can specify which Metric from the prior Session should be used to guide the new Session
  - By default, uses the optimization Metric of the prior Session



Continued Search: Usage

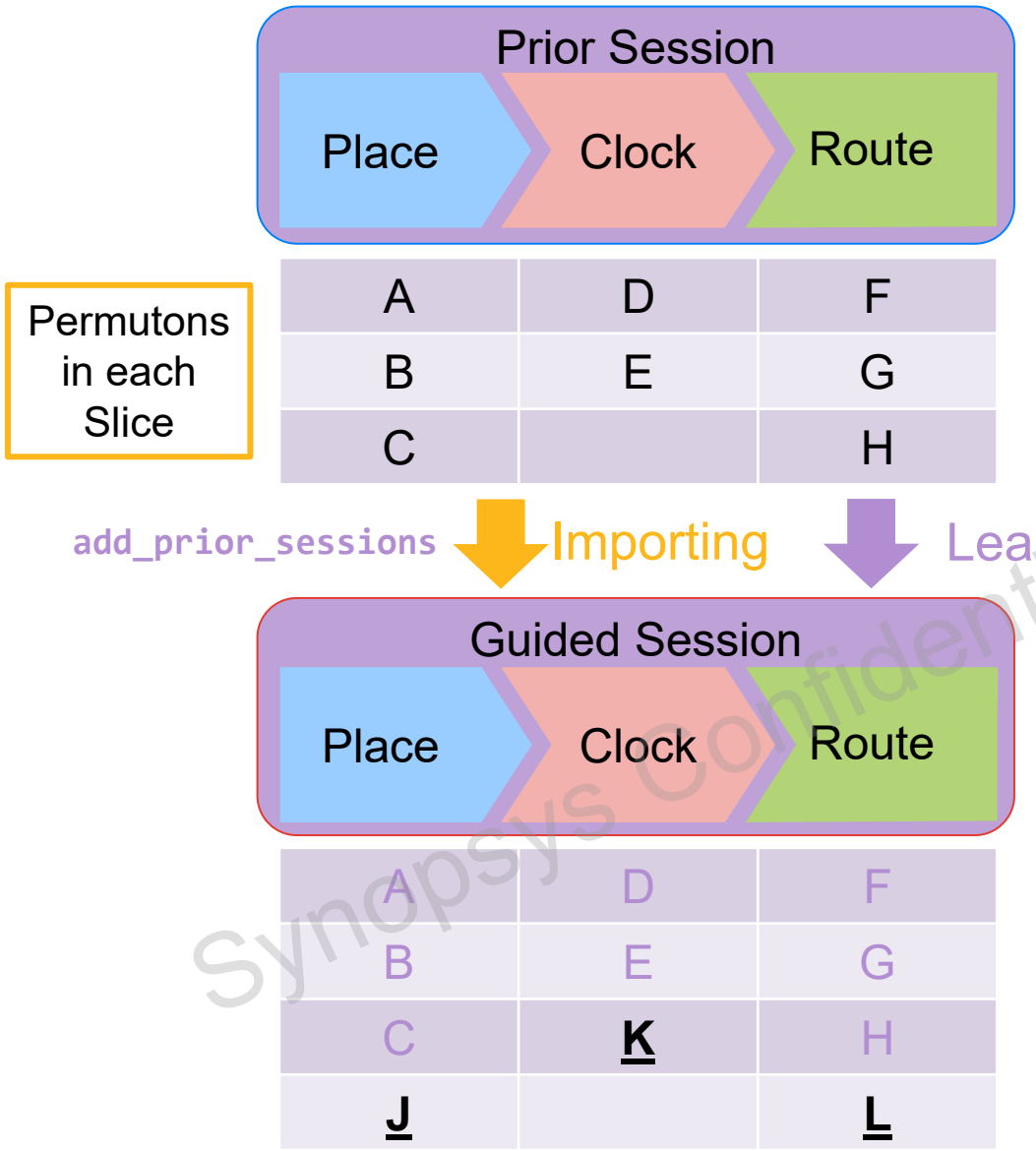


- Adding prior Sessions automatically imports its Permutons into the new Session
- DSO.ai applies learning from the prior Session to form suggestions for all Permutons that are in both Sessions
  - The Permuton values in the guided Session will be selected for lineages based on the prior Session's last slice results of the search space
- A smaller compute envelope can be used in the guided Session

Continued Search: Example

```
set prior_session [open_session -dir 0_place_clock_route]

create_session -from $prior_session
set_session_options -work_dir 1_place_clock_route

add_prior_sessions $prior_session

current_slice [index_collection [get_slices] 0]
set_compute_options -parallel_effort 15
current_slice [index_collection [get_slices] 1]
set_compute_options -parallel_effort 15
current_slice [index_collection [get_slices] 2]
set_compute_options -parallel_effort 15

report_session_config
run_session
```

Continued Search With Expanded Search Space: Usage



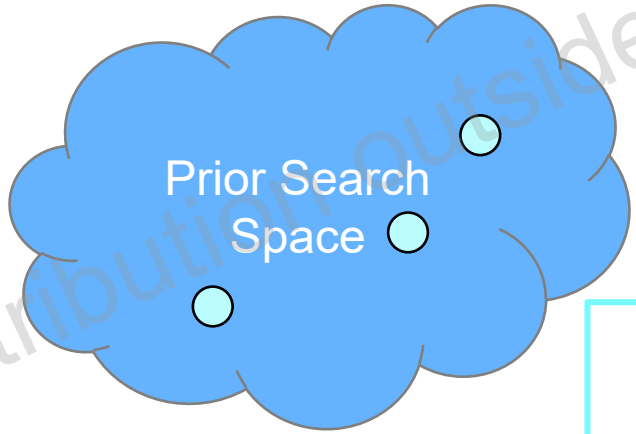
- For Permutons that are in both Sessions, DSO.ai applies learning from the prior Session to form suggestions
- For new Permutons, DSO.ai performs sampling as no learning exists for them
  - The guided Session will still form suggestions of Permuton values for the matching prior search space

Continued Search With Expanded Search Space: Example

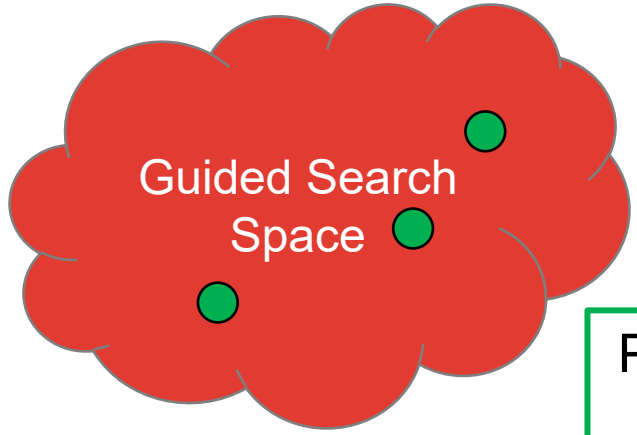
```
set prior_session [open_session -dir 0_place_clock_route]

create_session -from $prior_session
set_session_options -work_dir 1_place_clock_route

add_prior_sessions $prior_session

current_slice [index_collection [get_slices] 0]
add_permutons ...
current_slice [index_collection [get_slices] 1]
add_permutons ...
current_slice [index_collection [get_slices] 2]
add_permutons ...

report_session_config
run_session
```


Performing Applied Search

- To perform specific prior Runs in a guided Session:
  - Use the `add_priority_runs` command
  - Best used near tape-out
- For Permutons that are in both the prior and guided Session, the same Permuton values will be used for these priority Runs
- Additional modifications for priority Runs are allowed
- Can be used in a Session that also uses `add_prior_sessions`

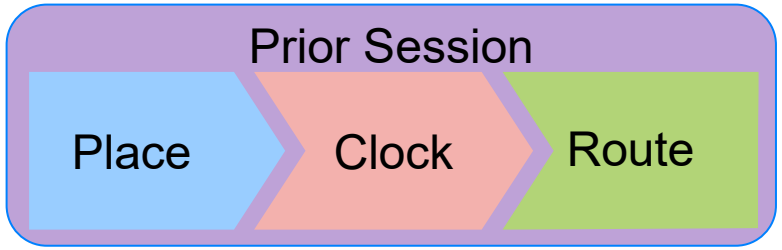


Prior Runs location in search space



Priority Runs location in search space

Applied Search: Usage

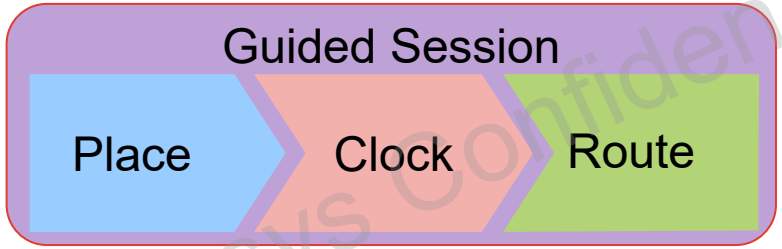


Permuton values in prior Run

Permutons in each Slice

A	D	F	low	0.5	0.01
B	E	G	5	15	medium
C		H	1000		true

`add_prior_sessions` Importing Learning



`add_priority_runs` Replay

A	D	F	low	0.5	0.01
B	E	G	5	15	medium
C		H	1000		true

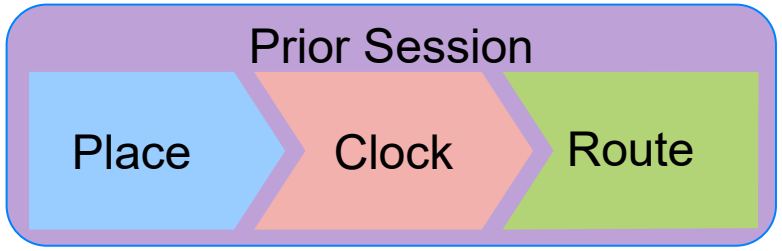
Permuton values in priority Run

- All Permutons that are in both Sessions use the Permuton values of the prior Runs for the specified priority Runs
- Collateral change affects Run's results

Applied Search: Example

```
set prior_session [open_session -dir 0_place_clock_route]
set my_favorite_runs [get_runs -num 5 -session $prior_session -slice route]
create_session -from $prior_session
set_session_options -work_dir 1_place_clock_route
add_prior_sessions $prior_session
add_priority_runs $my_favorite_runs -equal_permutons

current_slice [index_collection [get_slices] 0]
set_compute_options -parallel_effort 15
current_slice [index_collection [get_slices] 1]
set_compute_options -parallel_effort 15
current_slice [index_collection [get_slices] 2]
set_compute_options -parallel_effort 15

report_session_config
run_session
```


Applied Search With Expanded Search Space: Usage



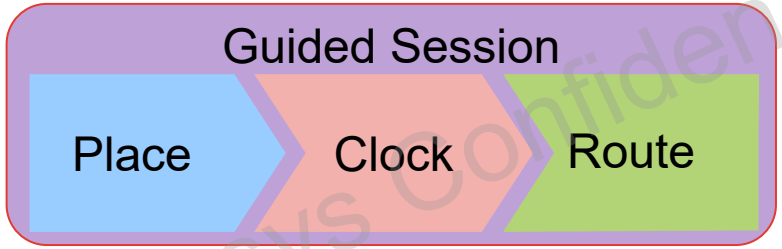
Permutons in each Slice

A	D	F
B	E	G
C		H

Permuton values in prior Run

low	0.5	0.01
5	15	medium
1000		true

`add_prior_sessions` Importing Learning



`add_priority_runs` Replay

A	D	F
B	E	G
C	**K**	H
J		**L**

low	0.5	0.01
5	15	medium
1000	**high**	true
true		**0.7**

- For common Permutons, priority Run will use same Permuton value as in prior
- For new Permutons, priority Run performs sampling as no learning exists for them

Applied Search With Expanded Search Space: Example

```
set prior_session [open_session -dir 0_place_clock_route]
set my_favorite_runs [get_runs -num 3 -session $prior_session -slice route]
create_session -from $prior_session
set_session_options -work_dir 1_place_clock_route
add_prior_session $prior_session
add_priority_runs $my_favorite_runs -multiply 3

current_slice [index_collection [get_slices] 0]
add_permutons ...
current_slice [index_collection [get_slices] 1]
add_permutons ...
current_slice [index_collection [get_slices] 2]
add_permutons ...

report_session_config
run_session
```


There are Four Main “inputs” to DSO.ai

- A working baseline flow
  - Starting input design
  - Tool flow/scripts (FC/ICC2)
- A search space
  - DSO.ai applies the search on top of your baseline flow
  - You define the search space (tool settings, design settings, etc.)
  - DSO.ai provides a default search space for each flow stage
- Success criteria (ADES)
  - Aggregate DEsign Score (ADES) combines multiple output metrics (TNS, DRCs, etc)
  - Target values and weights can be automatic or user-specified
- Search instructions
  - How many machines to use in parallel, # of cores per job, memory per job
  - Total number of jobs to run, runtime limit, etc.

DSO.ai Sample Runscript (compile_fusion)

Baseline Flow

```
create_config -name compile

set_session_options \
  -project_name MYPROJECT \
  -project_license MYLICENSE \
  -design_name vcore \
  -work_dir dso_work_compile

create_launcher -name compile \
  -tool "/path/to/fc_shell -output_log_file compile.log \
  -tool_script rm_fc_scripts/compile.tcl"

set_design_data_options \
  -data_mode local_copy \
  -data_dir vcore.nlib
```

Search Space

```
create_permuton -type app \
  -name opt.area.effort -datatype categorical \
  -range {low medium high}

create_permuton -type global -name init_util \
  -datatype continuous -range [list .79 .83]
```

Success Criteria

```
create_aggregate_metric -name ADES \
  -component TNS \
  -component SHORT_DRC -target 10 \
    -baseline 20 -unacceptable 30;

set_session_options -include_baseline user -optimize ADES
```

Search Instructions

```
set_session_options -max_workers 20 \
  -effort_multiplier 3

set_sge_options -project_name bnormal \
  -resource_list {mem_free=6G} \
  -parallel_environment {mt 8}
```

Database Setup

```
set_database_options -type simple_fs \
-path delphi_db
```

Launch DSO.ai

```
run_dso
```


Flow Setup Checklist

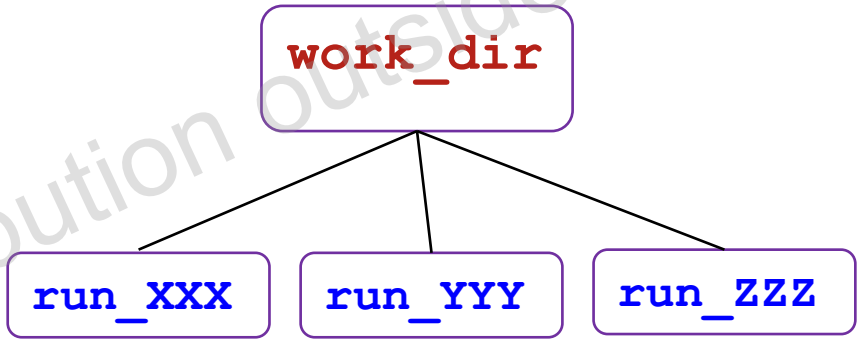
- Starting design
  - Usually an NDM design library with a starting block
  - Could also be ASCII files
- Flow scripts
  - First script loads starting design data
  - Scripts must run to completion without issue
  - Scripts must work within the DSO.ai child directory structure
- Determine flow slices (ie compile, clock, route)
- Tool version you want to execute with each flow script
- Tool memory requirements
- Number of cores required per tool job

Reduce data as much as possible

1) Starting design library contains one block
2) No unnecessary block saves
3) No dumping large files/reports

DSO.ai Directory Structure

- DSO.ai runs in its own **work_dir**
  - Automatically created if it doesn't exist
  - Automatically reused if it does exist
- Your baseline flow is explored inside **work_dir**
  - Explored in parallel with different settings
  - Each individual run is called a **child job**
  - Each **child job** runs in its own subdirectory (**run_XXX**)
- DSO.ai manages child jobs automatically
  - **Child jobs** that pass success criteria are saved
  - **Child jobs** that fail success criteria are deleted
  - Successful **child jobs** can be passed forward to the next flow slice



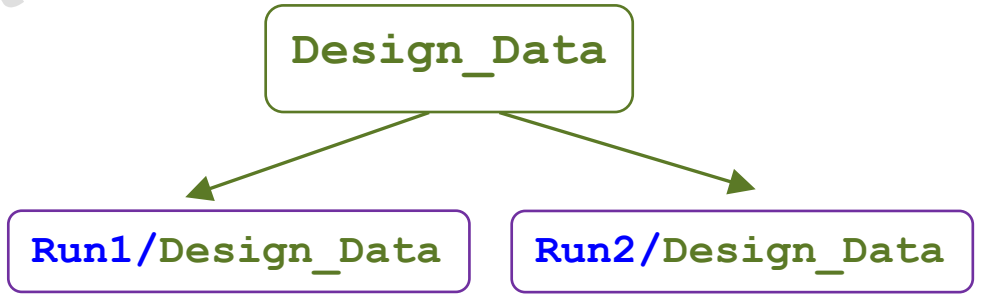
If **work_dir** already exists make sure it's empty before launching DSO.ai

DSO.ai Design Data Management

- DSO.ai copies starting design data into each child run directory
- Specify the Linux directory to be copied with

```
set_design_data_options -data_mode local_copy -data_dir ./Design_Data
```

- DSO.ai does the following:
  - Copies entire Design_Data directory into child run directories
  - Successful runs left in place on disk
  - Location of successful runs stored in DSO.ai database
  - Successful runs can be fetched by the next slice

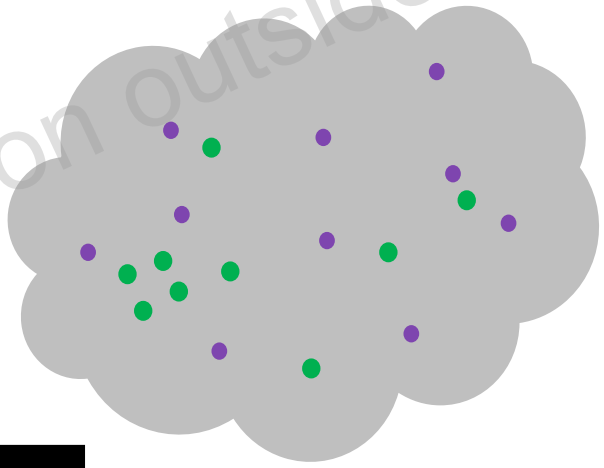


DSO.ai Setup – Total Compute Usage

- Each child job selects a sample from the permuton space
- You must tell DSO.ai
  - Number of child jobs to run in parallel (controls initial samples)
  - Learning effort (controls total number of samples)
- Max # of child jobs = (max_workers)*(effort_multiplier)

```
set_session_options -max_workers 30 -effort_multiplier 3
```

- 30 child jobs in parallel with max of 90 total child jobs
- More cold-start jobs means more data for warm-start to learn from later
- There are no restrictions on compute; DSO.ai uses whatever you like



DSO.ai Analysis

- DSO.ai manages many runs with many different settings and characteristics

- `dso_shell` lets you analyze data produced by all the runs
  - Report completed runs
    - Available metrics
    - Scatter plots
    - Permuton distribution reporting
    - Calculate new ADES to allow adjusting setup for future runs
    - Scatter plots can be used to provide analysis of metric relationships
  - Report any sessions available in current database
    - Allows loading & reporting of multiple sessions

- Both text-based and GUI analysis are available

DSO.ai Results

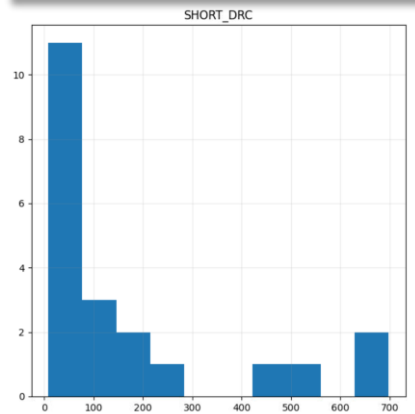
- Main logfile
  - Search progress
  - Child run information
  - Estimated disk use

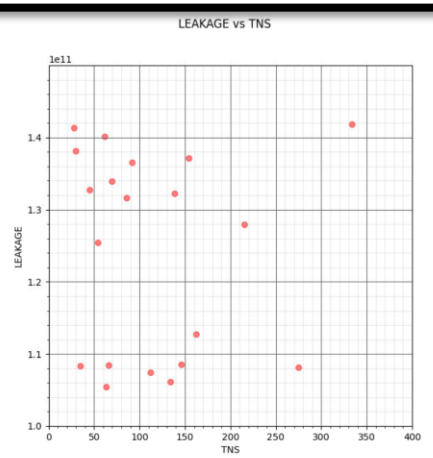
```
INFO  - Creating 20 workers
INFO  - Utilizing bash as worker shell.
INFO  - Session directory is: /path/to/your/dso/work_dir/
INFO  - symlinks in design_data_dir will be copied as-is, not their contents
INFO  - Design data management mode - local_copy
INFO  - Progress:   3.33%      Avg disk/saved run: 86.46 MB    Peak session disk est: 2.28 GB (DSO-1186)
INFO  - Progress:   6.67%      Avg disk/saved run: 87.16 MB    Peak session disk est: 2.30 GB (DSO-1186)
INFO  - Progress:  10.00%      Avg disk/saved run: 87.14 MB    Peak session disk est: 2.30 GB (DSO-1186)
...
WARNING - Run "0d758d0f" complete with status unsuccessful.
...
INFO  - Progress:  98.33%      Avg disk/saved run: 87.38 MB    Peak session disk est: 2.30 GB (DSO-1186)
INFO  - Progress: 100.00%      Avg disk/saved run: 87.37 MB    Peak session disk est: 2.30 GB (DSO-1186)
Finished
Return code is 'success'
```

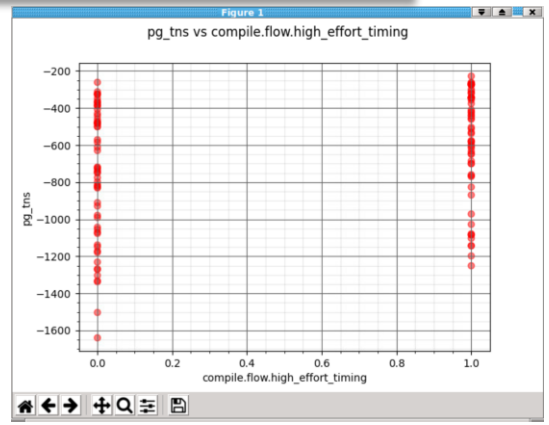
- Text-based analysis
  - Report metrics
  - Report results saved
  - Session/run status

ADES	TNS	LEAKAGE	SHORT_DRC	TOT_DRC	ID	BLOCK_SAVE
0.61749	-35.2	108300000000	62	73	bd8cf60c	ADES:0 R2R_WNS:0 TOTAL_POWER:8
0.66301	-62.0	140100000000	13	21	4bf74416	ADES:1 TOT_DRC:1
0.69778	-28.2	141300000000	62	95	36659608	ADES:2 R2R_TNS:1 STDCELL_AREA:1
0.78029	-45.4	132700000000	82	159	693f9d04	ADES:3
0.98827	-69.9	133900000000	19	41	dd508a15	ADES:4
1.36193	-92.1	136500000000	22	42	0754261b	ADES:5
1.55128	-86.1	131600000000	120	284	699be100	ADES:6 STDCELL_AREA:2
1.74385	-112.4	107400000000	63	97	6de8bc0b	ADES:7 R2R_WNS:1 TOTAL_POWER:7
2.20296	-54.5	125400000000	213	554	9df83106	ADES:8 STDCELL_AREA:0
2.22389	-138.9	132200000000	12	45	ecf64400	user_baseline

- Graphical analysis







Text-based Analysis to Assess Search Space Quality

- Analyze the permuton distribution for runs (minimum of 5):

```
dso_shell>  report_session_results -num 5 -permuton_distribution

  Permuton distribution for runs:

     ADES      ADES+        WNS        TNS    FREQ    R2R_WNS      R2R_TNS    R2RnoMem_Freq    R2RnoMem_top300_Avg_Freq    CONGESTION    ID
    -----      -----        ---        ---    ----    -------      -------    -------------    ------------------------    ----------    --------
  1.08061    1.45264    -0.0767     -961.5    1734    -0.0758    -912.9491           2605.3                      2758.5          0.61    1f32b711
  1.09391    1.79737    -0.0826    -1125.8     196    -0.0633   -1101.7086           2657.2                      2727.3          0.61    dfc1e304
  1.00418    2.99481    -0.1089    -1598.5    2348    -0.0617   -1561.0131           2693.6                      2772.0          0.57    5a62766f
  1.11602    3.82907    -0.0982    -1851.0    2408    -0.0662   -1796.0839           2609.7                      2725.7          0.62    d43e0154
  1.15277    4.24194    -0.0954    -1961.9    2479    -0.0834   -1893.5698           2555.3                      2738.1          0.63    f1197e5b

  FIXED PERMUTON             VALUE
  --------------             -----
  dso_use_syn_constraints    true
  dso_design_permuton_2      design

  PERMUTON                                        VALUES                          PERCENT                VARIANCE
  --------                                        ------                          -------                --------
  dso_mcp_memories                                true                            100                           0
  dsovar_filter_ao_oa                             4, 0                            60, 40                       20
  dso_adjust_memory_skew                          true, false                     80, 20                       60
  dso_frequency                                   3150, 3250, 3200                40, 40, 20                   20
  dsovar_initial_place                            2ndcongfactor, 2ndcong          80, 20                       60
  dsovar_compile_scale_setup_uncertainty          -20, -25, -15                   40, 40, 20                   20
  dsovar_compile_scale_design_maxtran             5, 0, -5                        60, 20, 20                   40
. . .

  Session Search Space: 3251005457001120000000000000000000000
  Reduced Search Space: 4738381338321616896000000
```

Interpreting Search Results

- DSO.ai sessions search many settings and design characteristics
- `dso_shell` lets you analyze and interpret all the search results
  - Analyze a session that's still running
  - Analyze multiple sessions at once
- Invoke a separate `dso_shell` to analyze all sessions
  - Report any session(s) available in current database
  - Report progress of ongoing session(s)
  - Report metrics, scatter plots, etc. for completed runs
  - Analyze search space quality
  - Calculate new ADES to allow adjusting setup for future runs
- main commands:

```
report_session_progress  and
set_result_column -add
report_session_results
list_metrics -application
```


Loading results with `dso_shell`

- Two ways to load session results
  - Automatic (connects to database and opens the session in the Linux `work_dir`)

```
unix> /path/to/dso_shell
dso_shell> open_sessions -dir /path/to/dso/work_dir
```

  - Manual

```
unix> /path/to/dso_shell
dso_shell> current_db -fs /path/to/common/dso/db
dso_shell> get_db_sessions
dso_shell> open_sessions my_dso_session
```

Remember use common database with `set_database_options`

Monitoring Session Progress: Completed Session

- Use **`report_session_progress`** to monitor session progress
- Completed session

```
dso_shell> report_session_progress

Loading: DSO_lab1_20210630-191741-056336
Session loaded 60 runs

   Session DSO_lab1_20210630-191741-056336
   Session 85% complete (51 of 60 runs reached DSO_final)

   RUN        LABEL          PROGRESS     RUNTIME   STATUS        LAST_CHECKPOINT          ADES
   --------   ------------   --------     -------   ------        ---------------          ----
   f31c2332   -              #########|    0.0689   DONE          DSO_final              0.90548
   58e70d28   -              #########|    0.0650   DONE          DSO_final              0.91494
   0df6eb00   user_baseline   #########|    0.0789   DONE          DSO_final              1.23386
   1b208716   -              #######..|    0.0400   UNSUCCESSFUL  compile_final_place         -
```

Session progress

Child run progress

Baseline run has user_baseline label

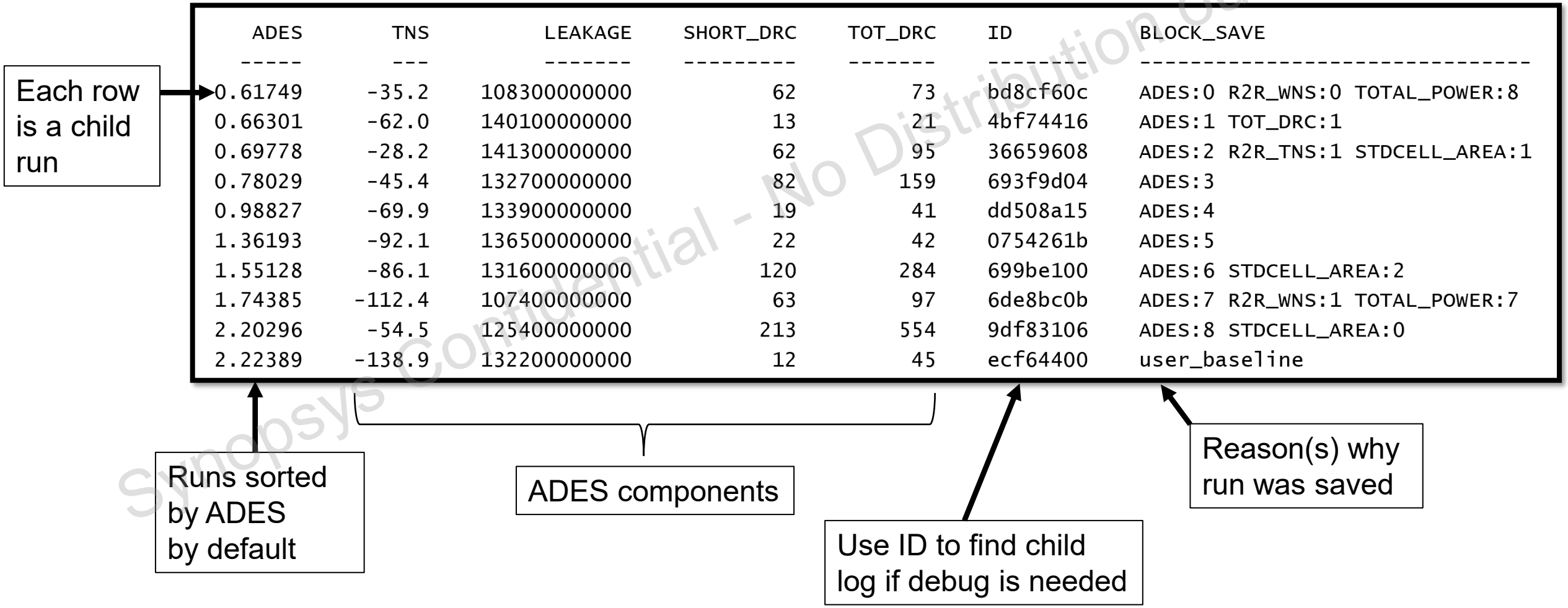
Each # represents a flow checkpoint

Last flow checkpoint

Metrics for completed runs

Interpreting Search Results

- Sample output from **`report_session_results`**

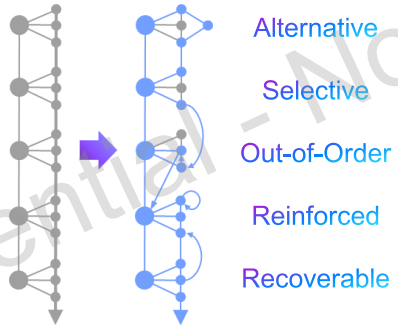


Pervasive AI Delivers EDA Productivity Boost

AI-DRIVEN IMPLEMENTATION

Limitless opportunity to leverage AI for EDA productivity

Optimization
Adaptive Flow



Autonomous Flow-tuning
Elastic Compute

AI Fusion
Fusion Compiler Native AI
Optimization Flow

Fusion Compiler – Streamlined AI-Driven Optimization

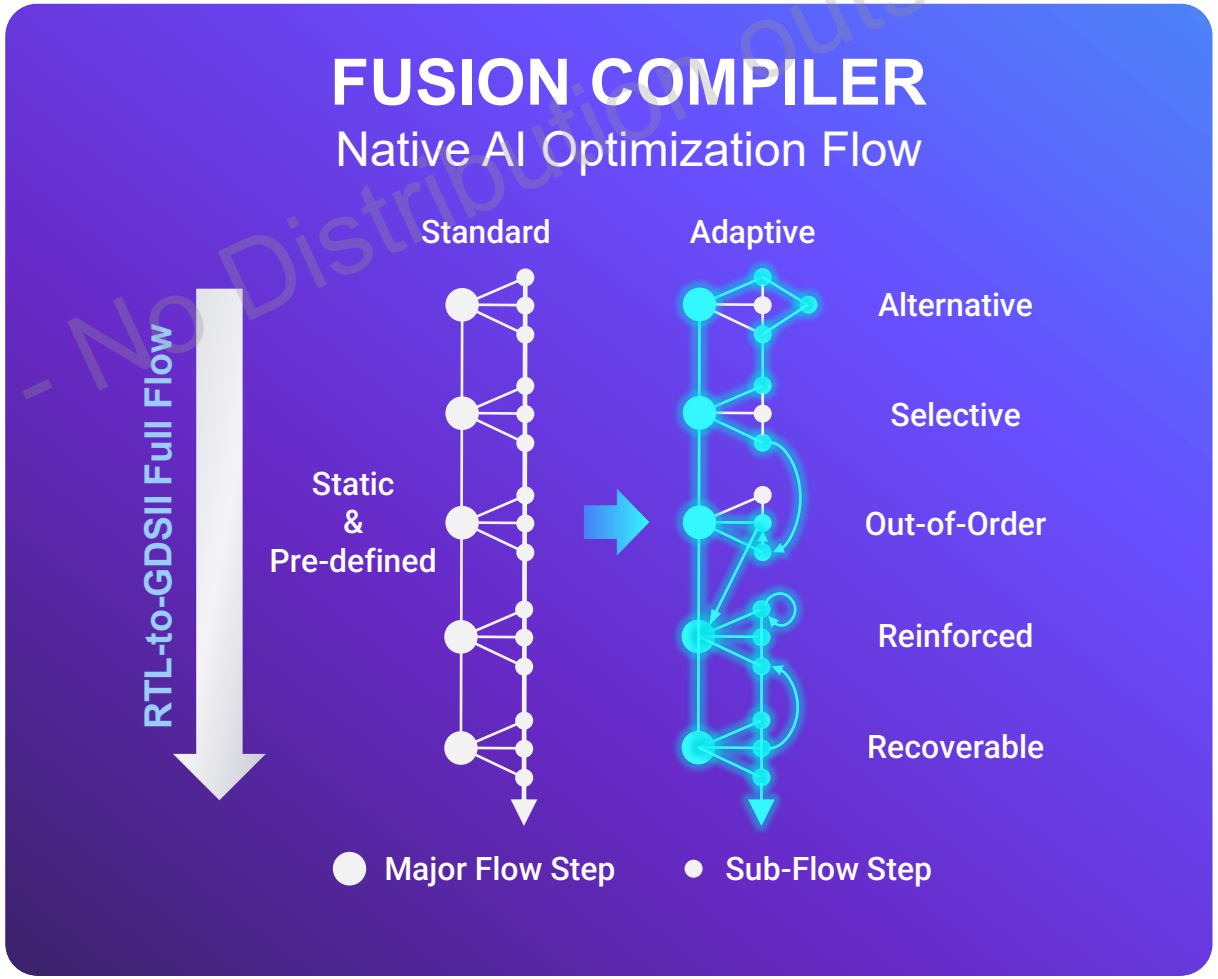
Natively fused DSO.ai enables intuitive user experience and improved efficiency

- **Interleaved AI with core engines**
  - Real-time and in-context flow adaptation
  - Fine-grain cross engine/flow learning and communication
- **Intuitive human AI interface**
  - Simple objective-based AI strategy setup
  - AI insights into decision vs. outcome
- **Scalable AI deployment**
  - Minimum 1-5 compute hosts required
  - Minimum flow changes for enablement
  - Available in FC V and W releases

Fusion Compiler – Adaptive HyperConvergence

Strategized Exploration to Efficiently Utilize Small-Scale Compute

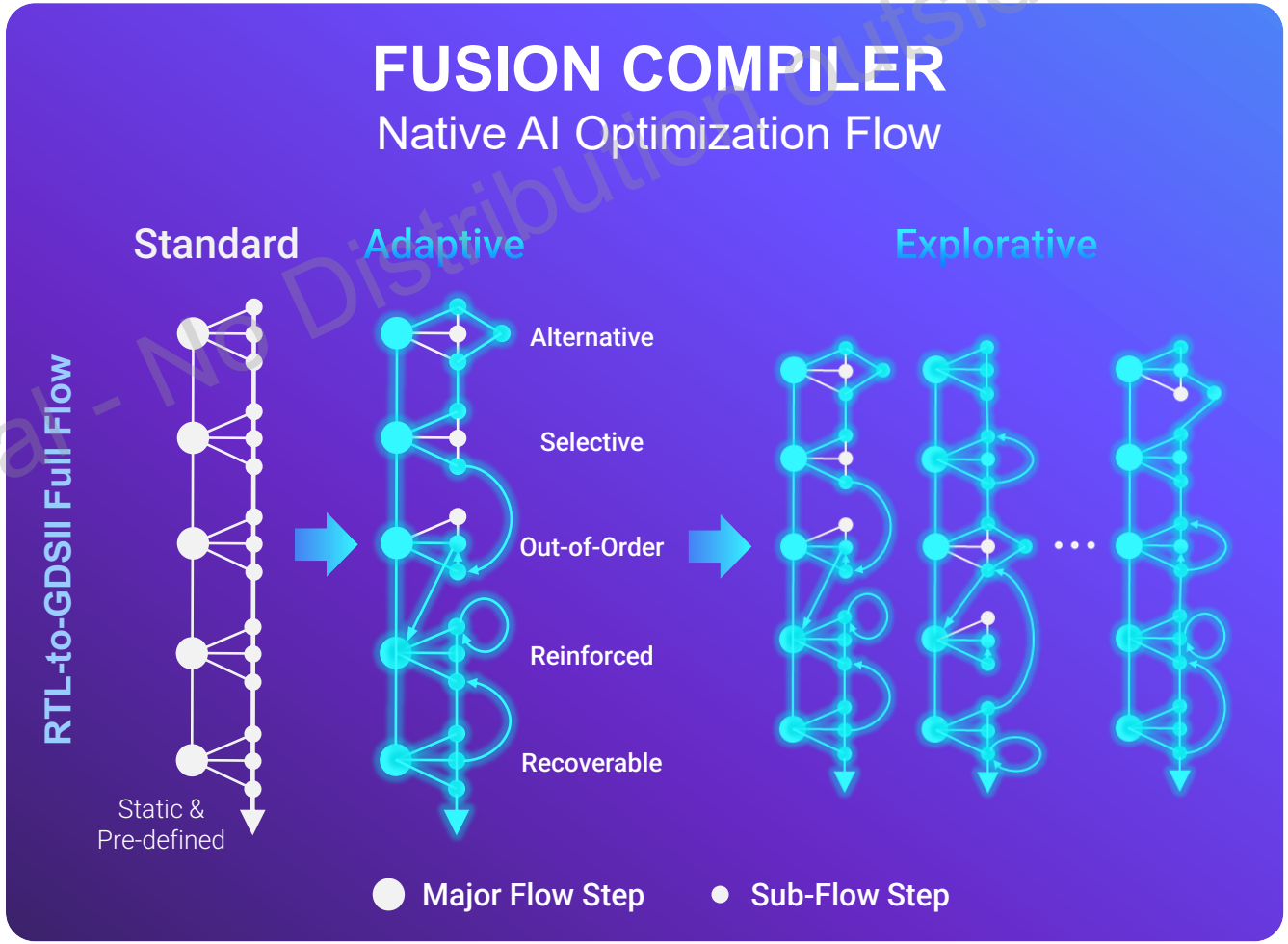
- **Predictive Machine Learning techniques with design learnings**
  - From prior sub-flows within command
  - From prior commands
  - From prior and parallel runs
- **Dynamically adapts based on real-time design metrics**
  - Flow & engine heuristics
  - Sub-flow sequences
- **Highly effective space exploration with 1-5 compute hosts**



Fusion Compiler – Adaptive HyperConvergent Flow

Strategized Exploration to Efficiently Utilize Small-Scale Compute

- **Predictive Machine Learning techniques with design learnings**
  - From prior sub-flows within command
  - From prior commands
  - From prior and parallel runs
- **Dynamically adapts based on real-time design metrics**
  - Flow & engine heuristics
  - Sub-flow sequences
- **Highly effective space exploration with 1-5 compute hosts**



Production: May '25

Fusion Compiler Streamlined AI-Driven Opt.

AI-driven PPA gains demonstrated at 10+ Pilot Customers; ~30 partitions in-flight

US HPC <2nm
1 - 4% total power, up to 25% TNS reduction (4 Cases)
Baseline: Highly-customized Design Flow

ASIA MOBILE 2/3nm
5 - 7% total power, 2 - 7% Fmax gain, 1 - 3% area increase (2 Cases)
Baseline: Optimized CAD Flow

US HPC 2nm
1.4% total power reduction, 18% TNS reduction, 65% HTNS reduction
Baseline: Highly-customized Design Flow

ASIA MOBILE 4nm
12.3% total power reduction, 1.5% urate improvement
Baseline: Optimized CAD Flow

ASIA ADAS 3nm
200Mhz Fmax improvement; 24% TNS reduction
Baseline: Highly-customized Design Flow

ASIA COMMS 12nm
21% leakage power, 86% TNS reduction, 25% less DRC
Baseline: Optimized CAD Flow

US AI/GPU 5nm
2.5% total power reduction, 12% TNS reduction, 1% area reduction
Baseline: Highly-customized Design Flow

ASIA MCU 22nm
5.2% total power reduction, 95% TNS reduction, 2.2% area reduction
Baseline: Optimized CAD Flow

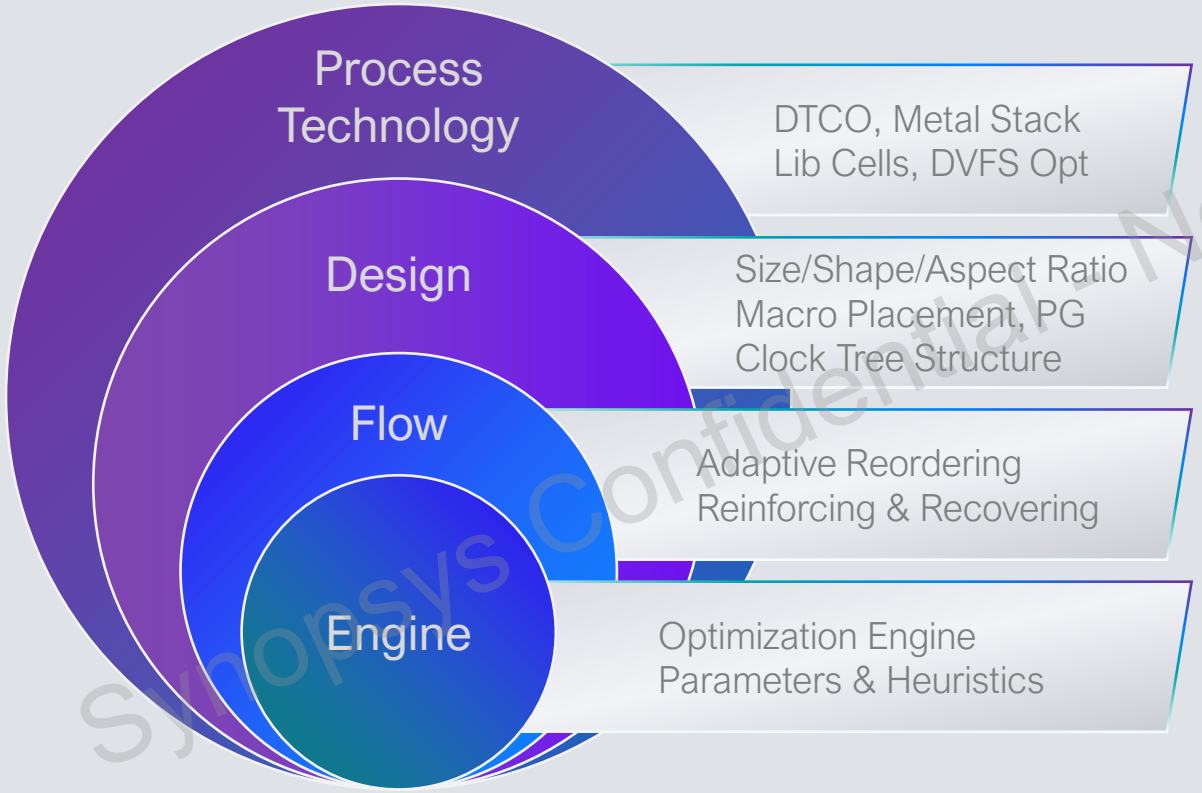
Ease-of-deployment: Direct launch from Fusion Compiler

Reduced compute: 5 machines single iteration

Finer grain: engine level learning

AI-Driven Digital Implementation Use Models

Digital Design and Optimization Solution Space



DSO.ai (Released 2020)

AI-Driven Design Space Exploration

- Custom Permutons / ADES
- Auto / Manual Space Configuration

Fusion Compiler (May 2025)

Streamlined AI-driven Optimization

- Fully Automated UX
- Adaptive HyperConvergent Flow

Understanding Slices and Waves

- AI Fusion flow slices.
  - Slices are flow stages, and AI Fusion supports 3 slices, **compile**, **clock** and **route**.
  - A slice can run a number of scripts, for example you might have a clock slice, that is comprised of two scripts, a build_clock script and a clock optimization script. You can simply pass all the scripts; you want to be run for a given slice as a list.

- You will also hear about waves and terminology like 5x1 or 5x2.
  - The first number ***5***, relates to how many runs are launched by AI Fusion, the default being 5.
  - The second number relates to how many waves are run. So:
    - **5x1:** equals 5 runs launched in parallel for the given slice. These 5 runs are orchestrated by AI Fusion, which means specific AI Fusion recipes are chosen for the 5 runs. Once the 5 runs complete, AI Fusion will pick the best run, which can then feed forward into the next slice.
    - **5x2** equals 5 runs launched in parallel, identical to 5x1. However, when these runs complete, the AI Fusion insight data is evaluated, and 5 more runs are launched, where the flow is dynamically tuned based upon the first wave of runs.

AI Fusion

- The following new commands have been added.
  - `set_optimize_ai_options`
    - Sets up the flow specific options i.e. compute, flow options etc...
  - `optimize_ai`
    - Runs the AI Fusion flow, this is where you can specify the number of waves etc…
  - `get_optimize_ai_runs`
    - Can be used for run analysis

Command: `set_optimize_ai_options`

Command definition and examples

- The `set_optimize_ai_options` command provides a generic setup for the AI Fusion flow.

```
set_optimize_ai_options
   [-design_data_dir delphi_data_dir]         Specify where the design ndm /designs source, and scripts can be found, this will be copied to each of the runs
   [-dso_root dso_root]                       Root path for Delphi install. This is optional, default will be to use the dso build included with FC.
   [-grid_settings delphi_grid]               Define Delphi grid settings
   [-reference_work_dir reference_work_dir]   Specify the directory name where AI Fusion will extract reference run data from for a replay
   [-resource_names dso_resource]             Specify the resource name(s) to be used by Delphi
   [-tool_binary tool_binary]                 Specify the executable to be used (FC, ICC2, RTL-A), if not specified the current binary will be used
   [-work_dir work_dir]                       Specify the directory name for AI Fusion to work in. The default is work_fcdso
   [-true_baseline enable_baseline]           Enables / Disables the true baseline flow, valid values are true or false (default is false)
   [-license_mode technology_subscription_license | bundled_license | atomos]. Specifies license mode, default is technology_subscription_license.
```

- The most important settings (which are mandatory) before running the `optimize_ai` command are

`-design_data`	Specify the path to the design data directory. This will be the data used by the AI Fusion runs which get launched in the directory specified by - work_dir (defaults to work_fcdso). So, this should contain your scripts / NDM's, or any files needed to run the script. Common files can by sym-linked, or referenced via absolute paths.,
`-license_mode`	Determine the license mode. The default is the `technology_subscription_license`, which means each run launched by AI Fusion will consume a license. The alternative is to use bundled licensing, of which there are two types, `bundled_license` checks out the Delphi-Adyton-Parnassus, and `atomos` will checkout Delphi-Adyton-Atomos
`-grid_settings`	Specify the compute resource(s) to use, this can be LSF, SGE, localhost etc.. Please see Defining the Compute Resources for a Session.
`-resource_names`	Specify the names of the compute resource (defined by –grid_settings that you want use), for example you could define LSF and localhost options and enable either one of the or both
`-true_baseline`	When set to true a run_slot will be allocated to running the baseline flow without any modification. This will be maintained through all slice.

Command: **optimize_ai**

```
optimize_ai # Setup for the AI Fusion flow
   [-baseline_only]                  (Only run the user baseline flow, do not launch any AI Fusion exploration runs)
   [-dont_launch]                    (Don’t launch any jobs, just create the directories and scripts)
   [-force_overwrite]                (Replaces existing slice directory, default is to error)
   [-intent intent]                  (Specify intent (power, timing or leakage), the default is power)
   [-number_of_waves num_waves]      (Specify the number of waves to run for a slice. The default is 1 (and max supported) is 2)
   [-number_of_run_slots number]     (Specify the number of runs to be launched in parallel, the default is 5)
   [-previous_slice previous_slice]  (Specify the previous slice where run data will be collected from)
   [-pre_script script]              (Specify a script to be source before the user script(s) defined by –scripts)
   [-post_script script]             (Specify a script to be source before the user script(s) defined by –scripts)
   [-replay_from_slice replay_slice] (Specify from which slice / run_id / the replay lineage should be extracted)
   -slice_name slice_name            (Specifies the slice name)
   -scripts design_run_scripts       (Specify the FC user script(s) to be run by FC, this can be a single script or lists of scripts.
```

- Most common options used

`-slice_name`	Specify the name of the slice, valid options are place, compile, clock or route.
`-intent`	Specify the design intent, valid values are power (the default) and timing. Timing is always a priority; this option will configure AI Fusion to adjust dynamic flow settings and run selection. The recommended setting is power, which will still optimize for timing, but will also perform power focused operations, resulting in reduced total power.
`-number_of_waves`	Specify how many waves are launched. The default is 1. Specifying 2 will result in two waves for (5x2) for the current slice.
`-previous_slice`	Specify the previous slice. For the place / compile slice there is no previous slice, however for clock you need to specify where the previous slice data is coming from i.e. –previous_slice compile, or –previous_slice place.
`-number_of_run_slots`	By default, the flow will launch 5 parallel runs, with the flow configuration being orchestrated by AI Fusion. If true_baseline is enabled then 6 runs will be launched, as 1 additional compute slot is needed by the true_baseline run. If you specify -number_of_run_slots, then your provided slot count will be honoured. For example, if you specify 5, then only 5 slots (parallel jobs) will be launched , and the true_baseline will use one of the 5 runs slots available.

Example Usage: `optimize_ai`

```
fc_shell> optimize_ai -script {scripts/my_place.tcl} \
                      -slice_name compile \
```

- The above example will launch AI Fusion for slice place and will run 1 wave (the default). The script that will be run will be `scripts/my_place.tcl`

```
fc_shell> optimize_ai -script {scripts/my_clock.tcl} \
                      -slice_name clock \
                      -previous_slice place \
                      -force_overwrite
```

- This will launch AI Fusion for the clock slice, using the script scripts/my_clock.tcl. The best run from the previous slice place will be brought forward (copied) to the clock slice, as will the true baseline run (if enabled).

```
fc_shell> optimize_ai -script {scripts/my_clock.tcl} \
                      -slice_name clock \
                      -number_of_waves 2 \
                      -previous_slice place \
                      -force_overwrite
```

- This is identical to the previous example, except now 2 waves will be run (1 wave after the other).

Example Usage Cont… : optimize_ai

- As only 4 slices are supported (place, compile, clock and route), you might need to use multiple scripts for a slice. For example, maybe your clock slice consists of 2 scripts i.e. build_clock.tcl and clock_opt.tcl.
  - This can easily be achieved by supplying a list scripts as shown below
  - Note each script is run in separate executables (sequentially in the order provided)

```
fc_shell> optimize_ai -script {scripts/build_clock.tcl scripts/clock_opt.tcl} \
                      -slice_name clock \
                      -previous_slice place \
                      -force_overwrite
```

- When a AI Fusion wave completes, you will see a summary for that wave. The Best run, is the run with the lowest score.

	RUN_ID	SLICE	SCORE	WCV	R2R_FCTNS	HTNS	WTOT_PWR	WLKG_PWR	STDCELL_AREA	CONGESTION
4	1f80543	clock_1	0.469673	1.0809	-0.00405	-0.5726	2.406	0.01701	18.115	0
1	bd17c22	clock_1	0.53315	1.0909	-0.00454	-0.57232	2.4062	0.01701	18.115	0
3	6ad4781	clock_1	0.53315	1.0909	-0.00454	-0.57232	2.4062	0.01701	18.115	0
2	dd564a5	clock_1	1.99725	1.1904	-0.00952	-0.57156	2.5141	0.02009	19.421	0
0 B	19057e0	clock_1	5.06492	1	0	-0.56857	2.6491	0.02445	21.216	0
0	44b95a4	clock_1	5.97095	1	0	-0.57516	2.65	0.02501	21.379	0

True Baseline Run

Best Run

Command: get_optimize_ai_runs

- The `get_optimize_ai_runs` command can be used to query QOR information for the runs.

```
get_optimize_ai_runs# Get AI Fusion run information
  -slice slice_name        (Specify the slice name)
  [-run_id run_id]         (Specify a run_id for which run information is to be queried)
  [-reference]             (Return run information from the reference run specified via set_optimize_ai_option –reference_work_dir)
  [-return_run_ids_only]   (Return the run_ids for the runs only)
  [-baseline]              (request for the baseline run_id)
  [-num_runs num_runs]     (Return information for n runs, the default is 1)
  [-table]                 (Display all metric data as a table)
  [-detailed]              (Include slice name and session information in the returned information)
```

```
fc_shell> get_optimize_ai_runs -slice clock
bc50810
```

- Gets the ‘best’ run for the slice clock.

Examples of getting run data

Reporting run metrics as a table

```
fc_shell> get_optimize_ai_runs -slice clock -table
```

	METRIC	clock_1:639bf73	clock_1:34784a0	clock_1:59a62f1	clock_1:9b12972	clock_1:ad64194
0	WCV	1	1	1	1.0249	1.0919
1	WTOT_PWR	2.6493	2.4084	2.4084	2.4077	2.5204
2	WLKG_PWR	0.02472	0.01732	0.01732	0.01703	0.02236
3	FCTNS	-0.30998	-0.30752	-0.30752	-0.30876	-0.31019
4	R2R_FCTNS	0	0	0	-0.00125	-0.0046
5	FCWNS	-0.15869	-0.15768	-0.15768	-0.15768	-0.15639
6	R2R_FCWNS	0	0	0	-0.00125	-0.0046
7	TNS	-0.30998	-0.30752	-0.30752	-0.30876	-0.31019
8	WNS	-0.15869	-0.15768	-0.15768	-0.15768	-0.15639
9	FREQ	4791.84	4815.12	4815.12	4815.12	4845.29
10	R2R_WNS	0	0	0	-0.00125	-0.0046
11	R2R_TNS	0	0	0	-0.00125	-0.0046
12	NVP	2	2	2	3	3
13	TOT_DRC	0	0	0	0	0
14	SHORT_DRC	0	0	0	0	0
15	CONGESTION	0	0	0	0	0
16	MAX_OVERFLOW	0	0	0	0	0
17	WIRELENGTH	2469	2451	2451	2451	2458
18	STDCELL_AREA	21.216	18.278	18.278	18.115	20.074
19	UTILIZATION	0.01	0.01	0.01	0.01	0.01
20	BUFFERS	9	7	7	7	8
21	INVERTERS	2	2	2	2	2
22	REP_AREA	12.0768	9.1392	9.1392	8.976	10.9344
23	REP_COUNT	11	9	9	9	10
24	LEAKAGE	0.02472	0.01732	0.01732	0.01703	0.02236
64	MAX_TRAN_WNS	0	0	0	0	0
…						
68	SCORE	6.43229	0.412626	0.412626	0.482479	3.24923

run_id and slice and wave

Run grading / Score (lower is better)

How do I change the optimization intent?

- The optimization intent for AI Fusion is controlled via the `-intent` option of **`optimize_ai`**. Valid options are **power** (Total Power), **timing** and **leakage**

```
optimize_ai -script {scripts/my_place.tcl} \
            -slice_name place \
            -intent power
```

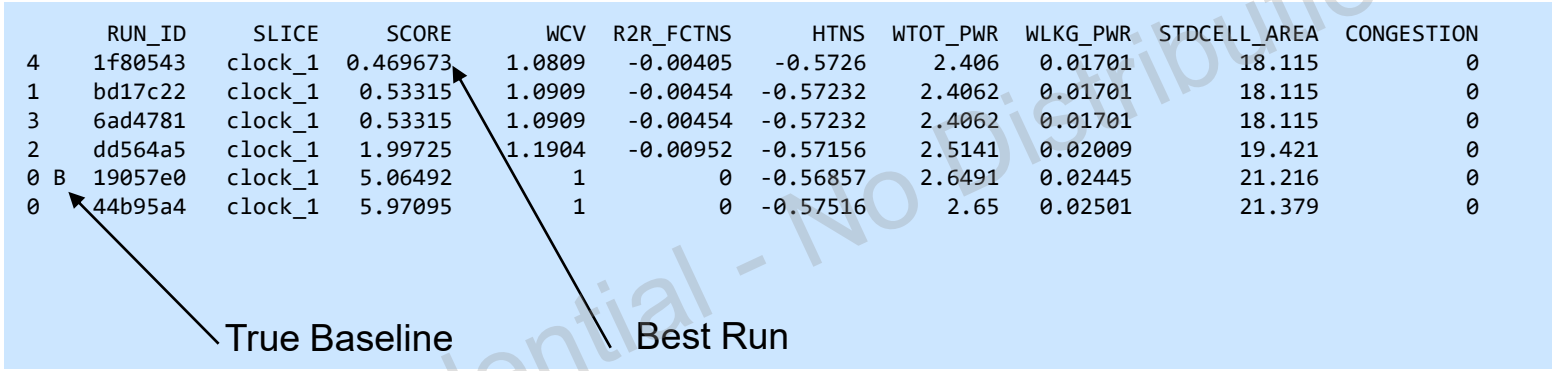
- Timing is always a priority; the intent adds secondary optimization objective.

- The `-intent` affects optimization within the runs, it also weights the run selection, i.e. **`-intent`** power will weight selection to runs with better power.

- NOTE: SQS settings used as part of the run script, should ideally match the intent specified here, however, if the intents are different IFS will apply dynamic selection, for example if SQS at the block is for timing, and you specify **`-intent power`**, then (under the hood) power optimization will be enabled.

How do I find the best run for a slice?

- In the logfile the run with the lowest SCORE is the best one.



- Also, In the work_fcdso directory a directory called ***best_run*** is created. Inside this you will find a symlink for each slice name, which will point to the best run for a given slice.

How can I control job runtimes?

- AI Fusion will launch multiple runs, and to prevent farm or flow exploration issues impacting all launched runs, we have enabled a runtime control feature.

- This can be enabled / disabled via the **`-enable_stop_rule`** option of `set_optimize_ai_options` (the default is true).

Turn ON stop rule : `set_optimize_ai_options -enable_stop_rule true`
Turn OFF stop rule: `set_optimize_ai_options -enable_stop_rule false`

- By default, we will wait for 40% of runs to complete for the running slice, so that would be 2 runs for 5x1 or 3 runs if true baseline is enabled.
  - We will then compute the average runtime for these runs and will then multiply that runtime by 2. So, if the average runtime for 40% of completed runs in the current slice is 30hours, we would then terminate any run for that slice that exceeded 60hours of runtime.
  - Note: the baseline run will never be terminated, and the runtime computation is calculated individually for each slice.

How can I control job runtimes?

Cont…

- The runtime termination controls can be adjusted via the **`-stop_rule_config`** option of `set_optimize_ai_options`.

- This command takes a list comprising of three values
  - RUN_PERCENTAGE: specifies the percentage of runs that must have completed before the average runtime is calculated. The default is 40.
  - MULTIPLIER: specifies a multiplier that is applied to the computed average runtime. The default is 2.
  - ABSOLUTE_RUNTIME: Specifies and absolute runtime in hours. Any run (except baseline) which exceeds the absolute runtime will be terminated. The default for this is 0 (disabled).

- The example below will terminate any run that exceeds the average runtime, computed after 50% of runs have finished, multiplier by 1.5. Additional any run that exceeds 200 hours of runtime will also be terminated.

```
set_optimize_ai_options -enable_stop_rule true\
                        -stop_rule_config {50 1.5 200}
```

How do I specify a pre / post Tcl script to be run?

- If you want all of your AI Fusion runs to apply settings before, or maybe run some commands after your main run script you can make use of the **`-pre_script`** and **`-post_script`** options of **`optimize_ai`** i.e.

```
optimize_ai -script {scripts/place.tcl scripts/post_place.tcl} \
            -slice_name place \
            -pre_script  scripts/place_pre_script.tcl \
            -post_script scripts/post_run_debug.tcl
```

- If you need to run multiple pre/post scripts, please specify them as a list. NOTE, the pre / post script will run before / after each script specified via *-script*

```
optimize_ai -script {scripts/place.tcl scripts/post_place.tcl} \
            -slice_name place \
            -pre_script  {scripts/generic.tcl scripts/place_pre_script.tcl}
```


How can I modify the build or resource for a relaunch?

- When using the ***–relaunch_from*** option for **`optimize_ai`**, you can modify the resource and tool settings.
- The two examples below show how to change the tool binary and resources used for the relaunched runs

```
fc_shell> optimize_ai -script {scripts/build_clock.tcl scripts/clock_opt.tcl} \
                      -slice_name clock \
                      -previous_slice place \
                      -relaunch_from {scripts/clock_opt.tcl} \
                      -tool_binary {default_binary/bin/fc_shell new_binary/bin/fc_shell}
```

```
fc_shell> optimize_ai -script {scripts/build_clock.tcl scripts/clock_opt.tcl} \
                      -slice_name clock \
                      -previous_slice place \
                      -relaunch_from {scripts/clock_opt.tcl} \
                      -resource_name {grid_small grid_big}
```


What if I disagree with AI Fusion's choice of the best run?

- It is possible to override the best run selection from the AI Fusion system.
- You can do this by specify a formatted list to **`optimize_ai`** i.e.

```
optimize_ai -slice_name clock -scripts scripts/clock.tcl \
            -previous_slice {slice:compile runid:df021d2}
```

- This tells AI Fusion to use **runid df021d2** from slice compile as the 'best' run to be used in the clock slice.

How Do I Replay a Run?

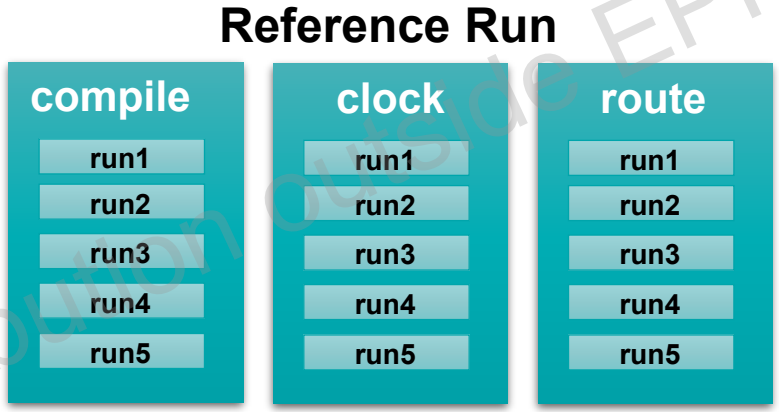
- It is possible to replay a previous run. Maybe you want to reproduce the results, or maybe you want to run the best result on new collateral.

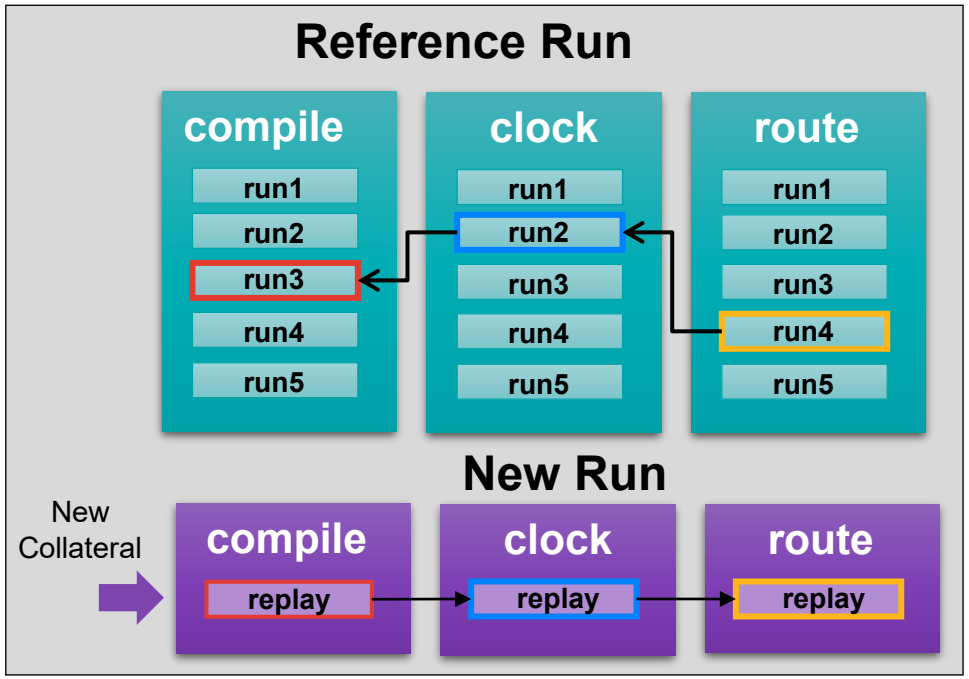
1. Replay is enabled by specifying a pointer to your reference run:

```
set_optimize_ai_options -reference_work_dir ../ref_run/work_fcdso
```

2. Replay is performed by specifying the slice (**`-replay_from_slice`**) from which you want to replay. In the example below the user has specified that the best result from the route slice will be replayed. A traversal is then performed to extract the setup required to replay that flow.

```
fc_shell> set_optimize_ai_options -reference_work_dir ../ref_run/work_fcdso

fc_shell> optimize_ai -slice compile -scripts {compile.tcl} -replay_from_slice route

fc_shell> optimize_ai -slice clock -scripts {clock.tcl} -replay_from_slice route \
            -previous_slice compile

fc_shell> optimize_ai -slice route -scripts {route.tcl} -replay_from_slice route \
            -previous_slice clock
```





How Do I change the run I want to replay?

- By default, the best run (as determined by AI Fusion), for a given slice, will be replayed.
- If you want to replay a specific *run_id* then you can also specify that via the **-replay_from_slice** option, by forming a list using *slice:<slice_name>* and *runid:<run_id>*
- For example, to replay the runid 663c610 from the route slice

```
fc_shell> set_optimize_ai_options -reference_work_dir ../ref_run/work_fcdso

fc_shell> optimize_ai -slice compile -scripts {compile.tcl} -replay_from_slice {slice:route runid:663c610}
```


exit *n*

- You have probably not considered how DSO ‘ignores’ an *exit* in your script.
- If you have a simple run script i.e.

```
open_block my_lib:my_block

source setup.tcl
compile_fusion

save_black –as compile
exit
```

- When you run this script in DSO, DSO will ‘skip’ the *exit* in the script, so that it can then report QOR information for your runs as part of the DSO_final checkpoint.
- However, if you add an exit code i.e. *exit 1* or *return -code break "exit 1“,* then DSO interprets this as a hard exit, and will exit your script at that point, meaning data collection will not occur as the session is terminated.

close_lib

- The DSO_final checkpoint runs automatically at the end of your run script.

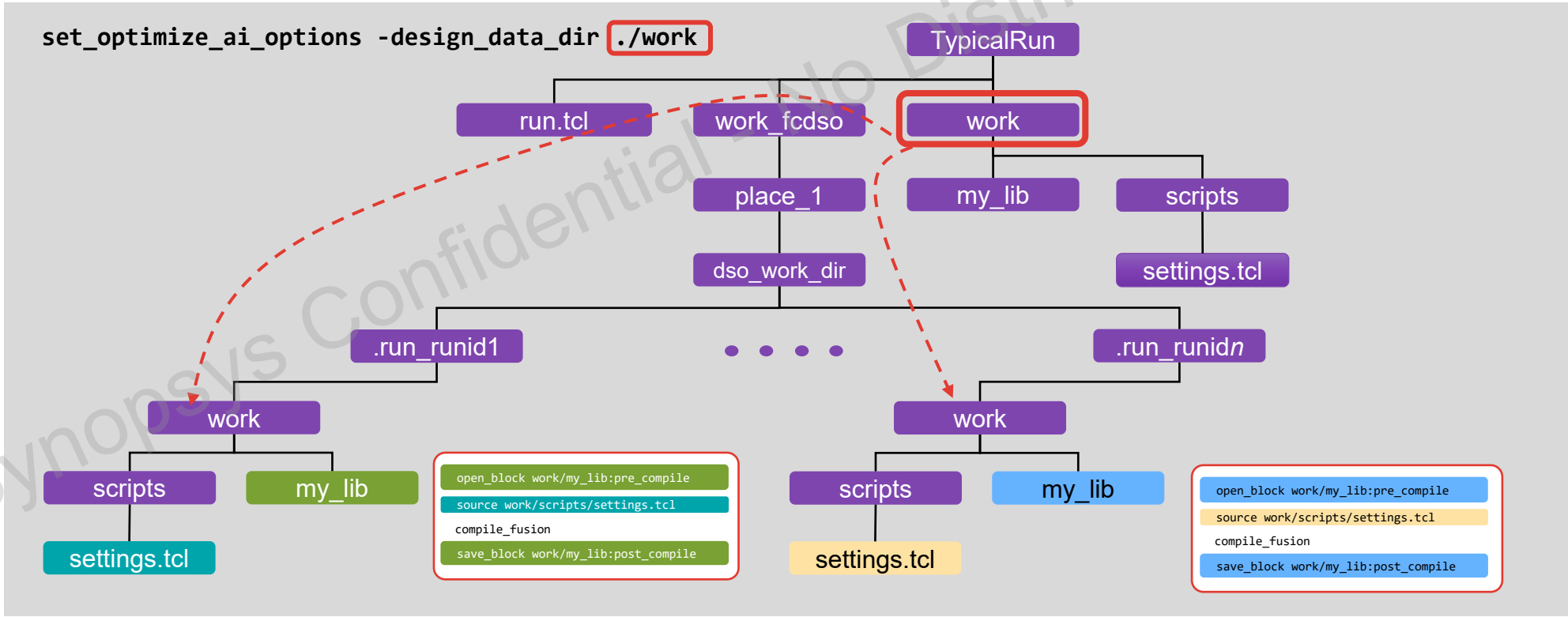
- If you are finding that the DSO_final is not executed (AI Fusion will mark runs as unsuccessful), and you have no ***exit n*** in your scripts, then maybe you have a close block / lib

```
open_block my_lib:my_block

source setup.tcl
compile_fusion

save_black –as compile
close_lib my_lib
```

- When you run this script in DSO, the **close_lib** will occur before the DSO_final checkpoint, hence there will be no QOR metrics to report at the end of the flow, and the run will be marked as unsuccessful.

How design_data_dir works in the AI Fusion Flow

- In the example below the design_data_dir points to the directory ./work.
- When AI Fusion is launched, in the 'TypicalRun' directory, DSO is called (under the hood) to launch the FC/ICC2 runs, and the following directory structure is automatically built.
- **Importantly** the **work** directory is copied to all the AI Fusion launched runs. Therefore, when each of these runs open / saves a block they are saving the design to their local copy



example:

Directory Structure

RUN_1

- OPEN_SETUP
- FUSION_LIBRARIES
- IMPORT
- PLACEANDROUTE_ICCOMPILERII_FCS
  - Makefile
  - LOGS
  - SCRIPTS
  - TECH
  - **DATA** (all what is specific to each FC run)
    - DESIGN_LIB
    - DATA
  - **run_5x1.tcl**
  - **run_5x1.csh**
  - **fc.pre.csh**

example:

run_5x1.csh

```
#!/bin/csh
module rm fc
module load fc
# Run compile
fc_shell  -f run_5_1.tcl -output_log_file  run_FCS.log
```

Avoid close_block / close_lib :

```
if { !(enable_dso) } then {
    close_blocks -force
    close_lib -force -purge
}

=> need to set the following variable:
set enable_dso  "true"
```

example:

run_5x1.tcl

```
set_optimize_ai_options \
    -design_data_dir ./DATA \
    -grid_settings {set_lsf_options -name lsf -queue_name atto -app "batch_normal" -resource_requirement { rusage[mem=70G]} -thread_count {8}} \
    -resource_names {lsf} \
    -true_baseline true \
    -work_dir work_fcdso_5x1 \
optimize_ai -script {./SCRIPTS/placeandroute/04_prectsOpt.IMPLEMENTATION.tcl} \
      -intent timing \
      -slice_name compile \
      -number_of_waves 1 \
      -force_overwrite \
      -dso_shell_pre_script [file normalize ./fc.pre.csh]
optimize_ai -script {./SCRIPTS/placeandroute/05_clockTreeSynthesis.IMPLEMENTATION.tcl
    SCRIPTS/placeandroute/06_postctsOpt.IMPLEMENTATION.tcl } \
      -intent timing \
      -previous_slice compile \
      -slice_name clock \
      -number_of_waves 1 \
      -force_overwrite \
      -dso_shell_pre_script [file normalize ./fc.pre.csh]
optimize_ai -script {./SCRIPTS/placeandroute/07_route.IMPLEMENTATION.tcl SCRIPTS/placeandroute/08_postrouteOpt.IMPLEMENTATION.tcl} \
      -intent timing \
      -previous_slice clock \
      -slice_name route \
      -number_of_waves 1 \
      -force_overwrite \
      -dso_shell_pre_script [file normalize ./fc.pre.csh]
```

example:

fc.pre.csh

```
#!/bin/csh
cd ../
ln -sf -T ../../../../OPEN_SETUP OPEN_SETUP
ln -sf -T ../../../../IMPORT IMPORT
ln -sf -T ../../../../FUSION_LIBRARY FUSION_LIBRARY
cd -
ln  -sf -T ../../../../SCRIPTS SCRIPTS
ln  -sf -T ../../../../TECH TECH
ln  -sf -T ../../../../.synopsys_fc.setup .synopsys_fc.setup

ln -s ./DATA/* .
mkdir REPORTS
```

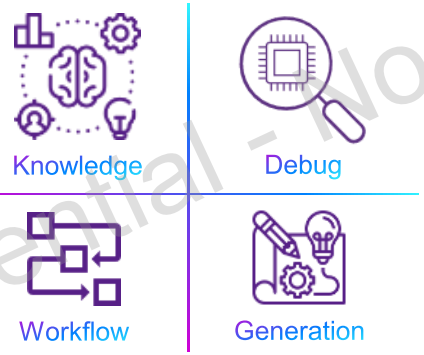
Pervasive AI Delivers EDA Productivity Boost

AI-DRIVEN IMPLEMENTATION

Limitless opportunity to leverage AI for EDA productivity

Assistance

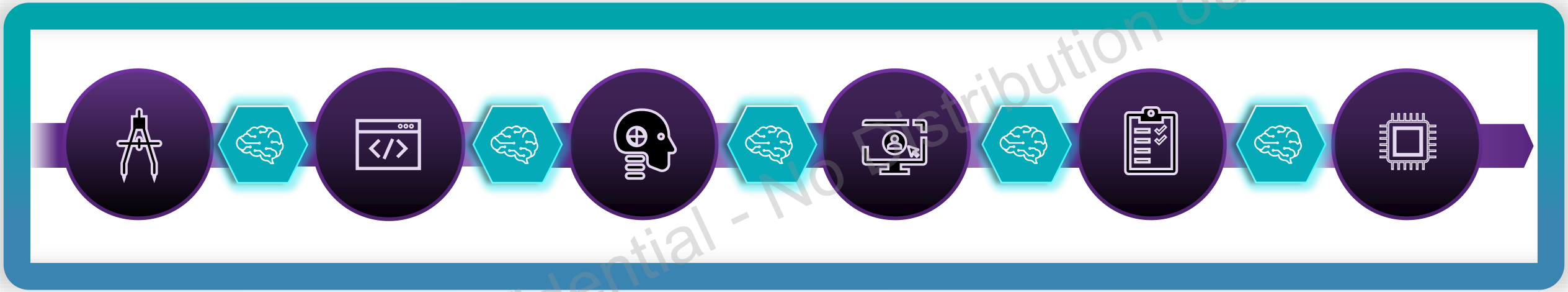
GenAI CoPilot



Workflow / Debug Assist
TechFile / RTL Generation

Generative AI-Powered

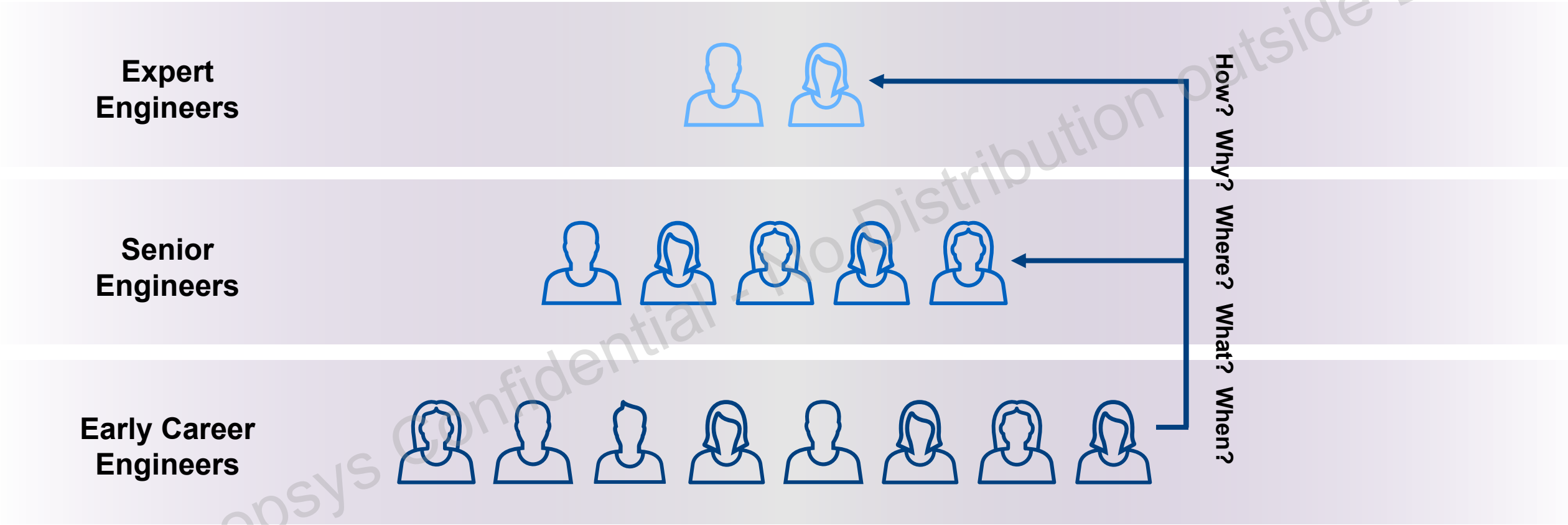
GenAI : A New Productivity Paradigm for Chip Design



Architecture | Verification | Test & SLM | Implementation | Signoff | Manufacturing

Tool / Flow Assistants
Design Collateral Creation
Agentic Workflow Automation

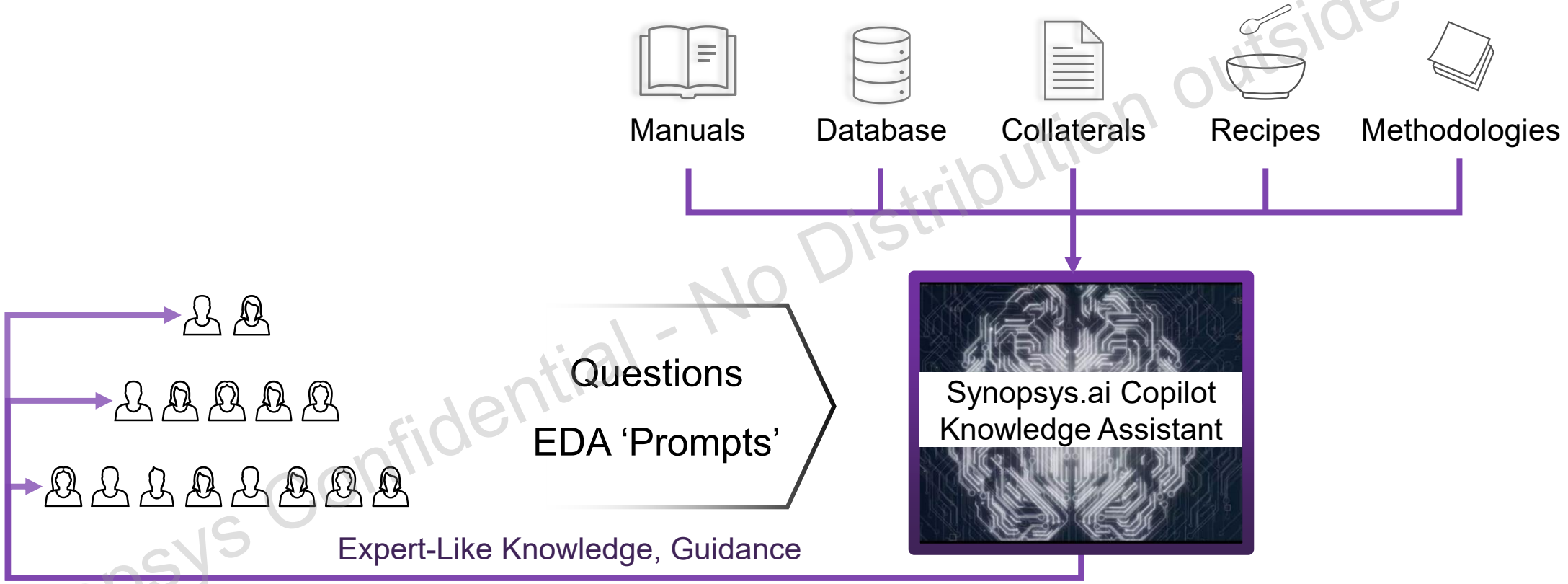
Today : Significant Time Spent in Searching for Information



40% of time searching and reorganizing raw data

4hrs per work week looking for data/information

GenAI Assistants : Expert-Level Guidance, Faster Ramp

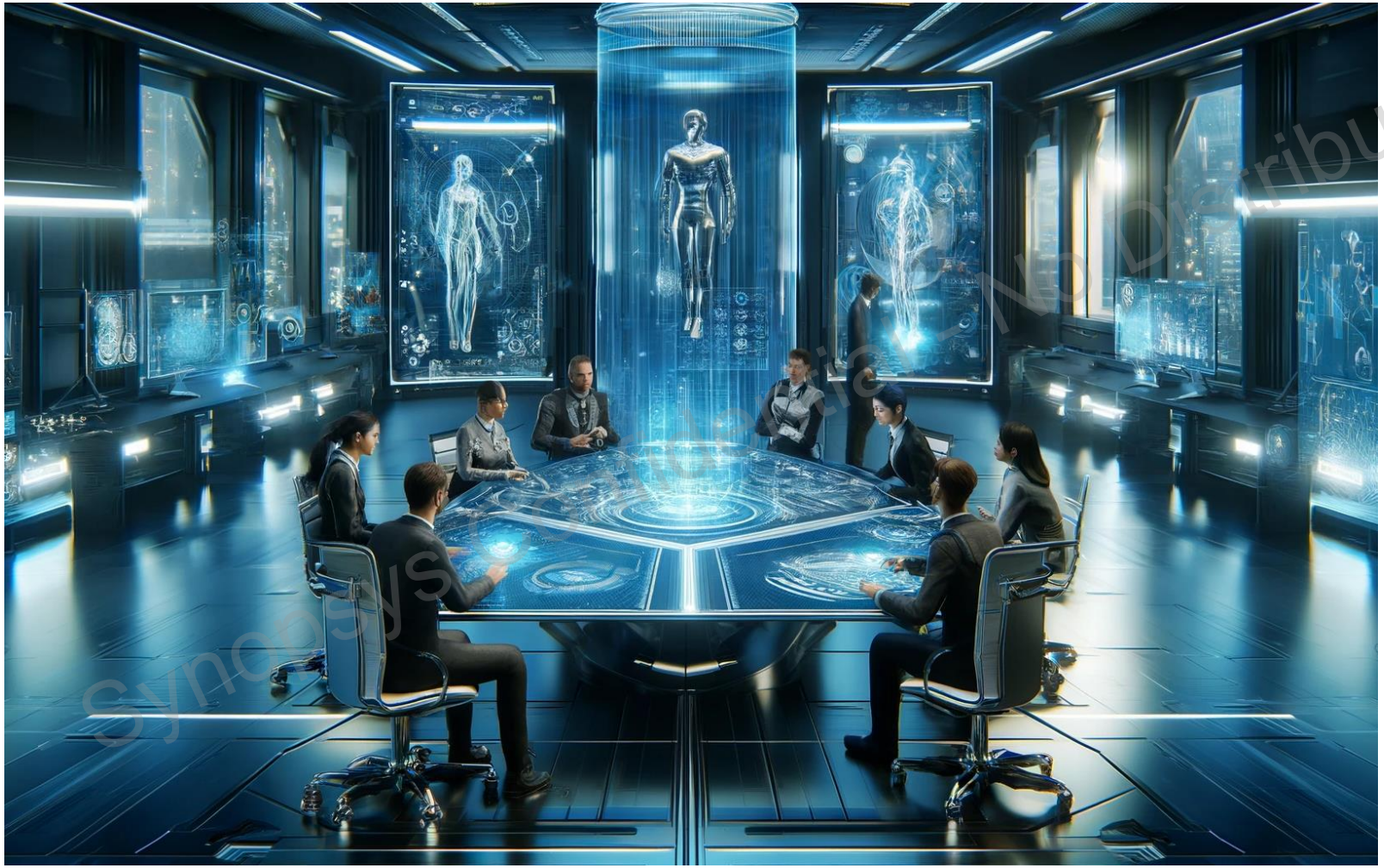


30% Faster Ramp Time For Engineers

25% Faster Closure of Tickets

2X Faster Response vs. Search Queries

Our Synopsys.ai Copilot
Single Cockpit for Everyday Workflows



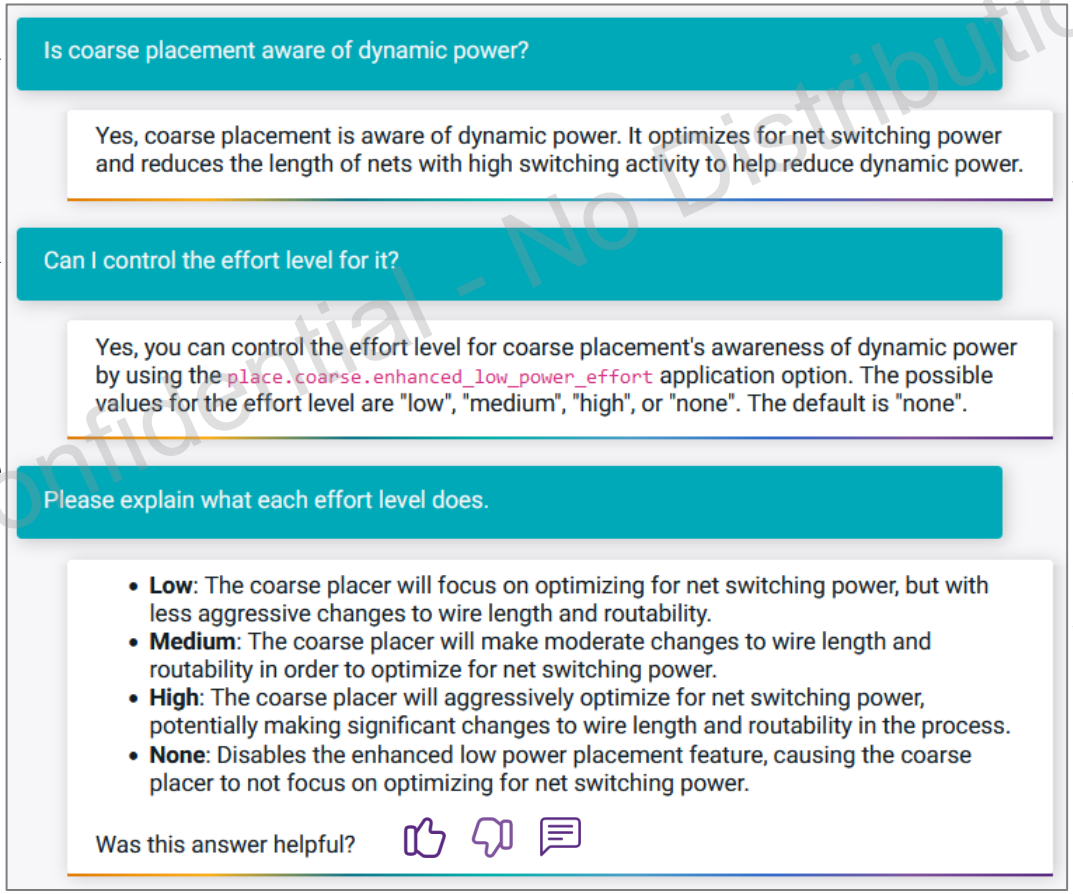
Key Capabilities

- Knowledge Assistant
  - Access to our Knowledge Base
  - Answers SNPS Tool & Flow questions
- Workflow Assistant
  - Create, debug, analyze, optimize Tcl/Python scripts
- Run Assistant
  - Interpret complex data to determine next steps
  - Summarize and Analyze Tool Runs, Suggest Next Steps

What Is Knowledge Assistant?

- Knowledge Assistant is a generative AI-based assistant, accessible from a Synopsys tool, that responds to natural-language questions about the tool
- Unlike traditional keyword-based search, Knowledge Assistant can:

Hold a conversation with you to help you explore and learn information



Online help

Man pages

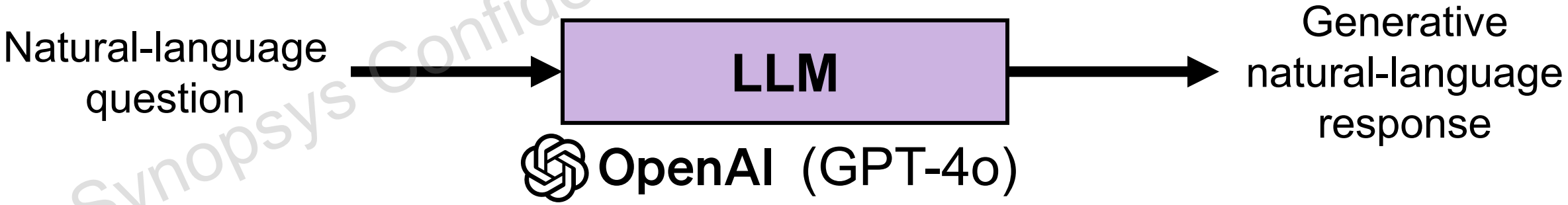
Combine information from multiple sources into custom answers to *your* questions

Glossary terms

SolvNetPlus articles

How Does Knowledge Assistant Work?

- Knowledge Assistant uses a *large language model* (LLM), which is a deep-learning neural network trained to perform content-related tasks
  - We currently use OpenAI's GPT-4o LLM
  - Other well-known LLMs are Llama 3.1 70B, Mistral/Mixtral, Grok, and Gemini

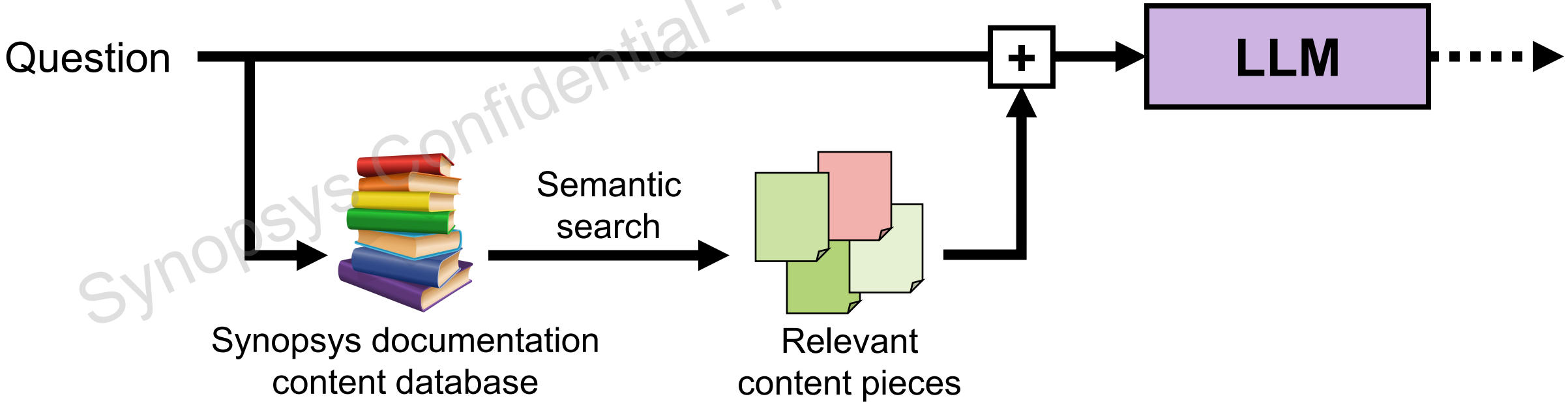


What Does Knowledge Assistant Know?

- Knowledge Assistant focuses on answering *general* tool questions, by using:
  - The tool's documentation
  - The tool's man pages (if applicable)
  - The tool's SolvNetPlus articles
- Knowledge Assistant does not know:
  - Information about your machine or environment
  - Information about your current design
  - Intrinsic Knowledge from the LLM itself
- Future Synopsys.ai Copilot features (like Run Assistant) will focus on answering design-specific and context-specific questions

How Does Knowledge Assistant Work?

- It uses a *retrieval-augmented generation* (RAG) architecture, which:
  - Retrieves additional content pieces from a database that are *semantically similar* to the question
  - Gives the question, plus the retrieved content pieces, to the LLM to generate a response
- The LLM "knows" that information (but only while answering that question)



Workflow Assistant

Helps you with your day-to-day tasks

Create a procedure called print_time_date to print the time and date.

```
proc print_time_date {} {
    set current_time_date [get_date -format "%Y-%m-%d %H:%M:%S"]
    puts "Current Time and Date: $current_time_date"
}
```

Was this answer helpful?

/generate create a procedure which query the ref_name attribute of cells bellow shorts routing DRC

```
proc query_ref_name_below_shorts {} {
    set drc_data [open_drc_error_data -name zroute.err]
    set short_errors [get_drc_errors -quiet -error_data $drc_data -filter type_name=="Short"]
    foreach_in_collection short $short_errors {
        set bbox [get_attribute -quiet $short bbox]
        set cells [get_objects_by_location -quiet -intersect $bbox -classes cell]
        foreach_in_collection cell $cells {
            set ref_name [get_attribute -quiet $cell ref_name]
            puts "Cell: [get_attribute -quiet $cell full_name], Ref Name: $ref_name"
        }
    }
}
```

Was this answer helpful?

Workflow Assistant Commands

Commands	Functions
/generate	Generate TCL code/script or procedure
/fix	Fix or optimize TCL code/script or procedure
/comment	Comment TCL code/script or procedure
/explain	Summarize or explain the code/script or procedure
/ka	Forces the question to be asked to Knowledge Assistant.

- Workflow Assistant commands (except /ka) explicitly direct the question to WA, but Synopsys.ai Copilot will also automatically route the question to WA based on how the question is worded

Workflow Assistant

/generate

/generate A procedure called print_time_date to print the time and date.

```
proc print_time_date {} {
    set current_time_date [get_date -format "%Y-%m-%d %H:%M:%S"]
    puts "Current Time and Date: $current_time_date"
}
```

Was this answer helpful?

/fix

/fix set mypaths [get_timing_paths -slack_lessar 0] foreach path $mypaths { report_timing $path }

```
set mypaths [get_timing_paths -slack_lesser_than 0]
foreach_in_collection path $mypaths {
  report_timing $path
}
```

Was this answer helpful?

Workflow Assistant

/comment

/comment

```
proc print_time_date {} {
   set current_time_date [get_date -format "%Y-%m-%d %H:%M:%S"]
   puts "Current Time and Date: $current_time_date"
}
```

```
proc print_time_date {} {
    # Get the current date and time in the specified format (YYYY-MM-DD HH:MM:SS)
    set current_time_date [get_date -format "%Y-%m-%d %H:%M:%S"]

    # Print the current date and time
    puts "Current Time and Date: $current_time_date"
}
```

Was this answer helpful?

/summary

/summary

```
proc print_time_date {} {
   set current_time_date [get_date -format "%Y-%m-%d %H:%M:%S"]
   puts "Current Time and Date: $current_time_date"
}
```

This Tcl procedure, `print_time_date`, retrieves the current date and time in the format "YYYY-MM-DD HH:MM:SS" using the `get_date` command. It then prints the formatted date and time to the console using the `puts` command. The procedure is useful for displaying the current timestamp in a standardized format.

Was this answer helpful?

Generative AI : Roadmap & Vision

2024 → 2025

Copilot

Collaborative
(Assistive features automating EDA workflows)

- **Knowledge Assistant (KA)** — Answer Expert Questions
- **Workflow Assistant (WA)** — Create Tcl/Python Workflows
- **Run Assistant (RA)** — Analyze & debug runs, suggest next steps

Generative
(Content creation using Human Assistants)

- **RTL Assistant** — RTL Gen from NL, Summarization, RTL PPA Advisor, RTL Linting
- **Verification Assistant** — Formal Testbench, UVM Creation from NL and RTL
- **Tech Collateral Assistant** — Create PyCell, ICV Runset, FC Techfiles etc
- **Design Spec from NL** — Create Design Spec (connectivity, constraints, testbench, UPF)

Autonomous
(Automated Workflows & Content Creation)

- **Agent Workflow Automation** — Automate debug and setup of tool workflows
- **Design Creation** — Generate digital netlist, layout; Generate analog layouts; Automated content creation etc

AI-Native EDA research area

Summary

- Knowledge Assistant uses information in documentation to answer natural-language questions about a tool
  - Online help, man pages, SolvNetPlus articles
- Knowledge Assistant knows only what is contained in the documentation
  - The Synopsys.ai Copilot Run Assistant will answer questions about tool results and next steps
  - The Synopsys.ai Copilot Workflow Assistant will help generate Tcl workflows for a tool
- Unlike traditional keyword-based search, Knowledge Assistant:
  - Helps you explore and learn information through interactive conversation
  - Combines information from multiple sources into custom answers to *your* questions
  - Performs better when you ask fully-worded, detailed questions
- As with any AI, Knowledge Assistant can make mistakes and require guidance
  - Submit thumbs-up and thumbs-down and/or add feedback to help us improve the system

SYNOPSYS®

Thank you