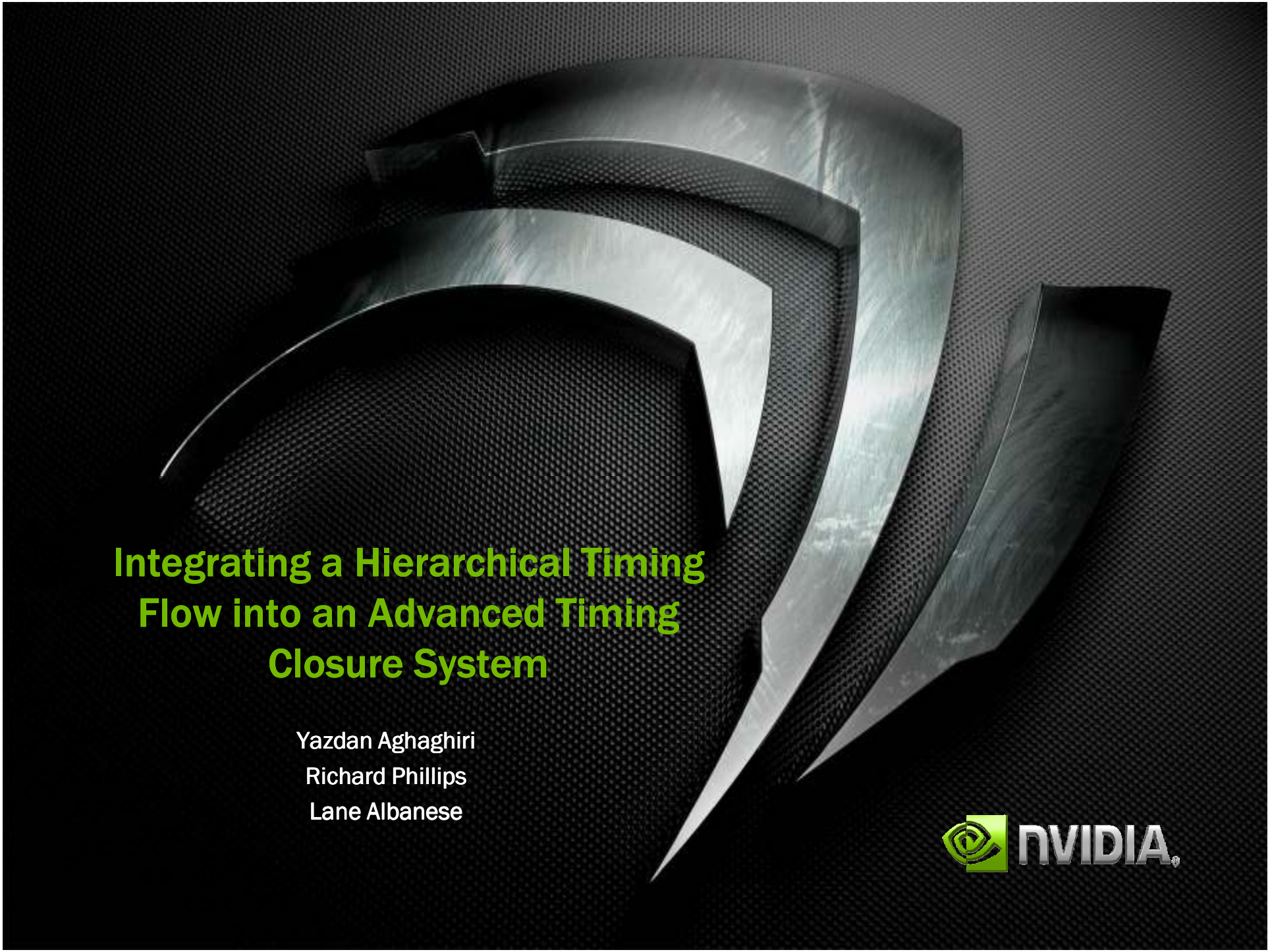


Integrating a Hierarchical Timing Flow into an Advanced Timing Closure System

Yazdan Aghaghiri
Richard Phillips
Lane Albanese

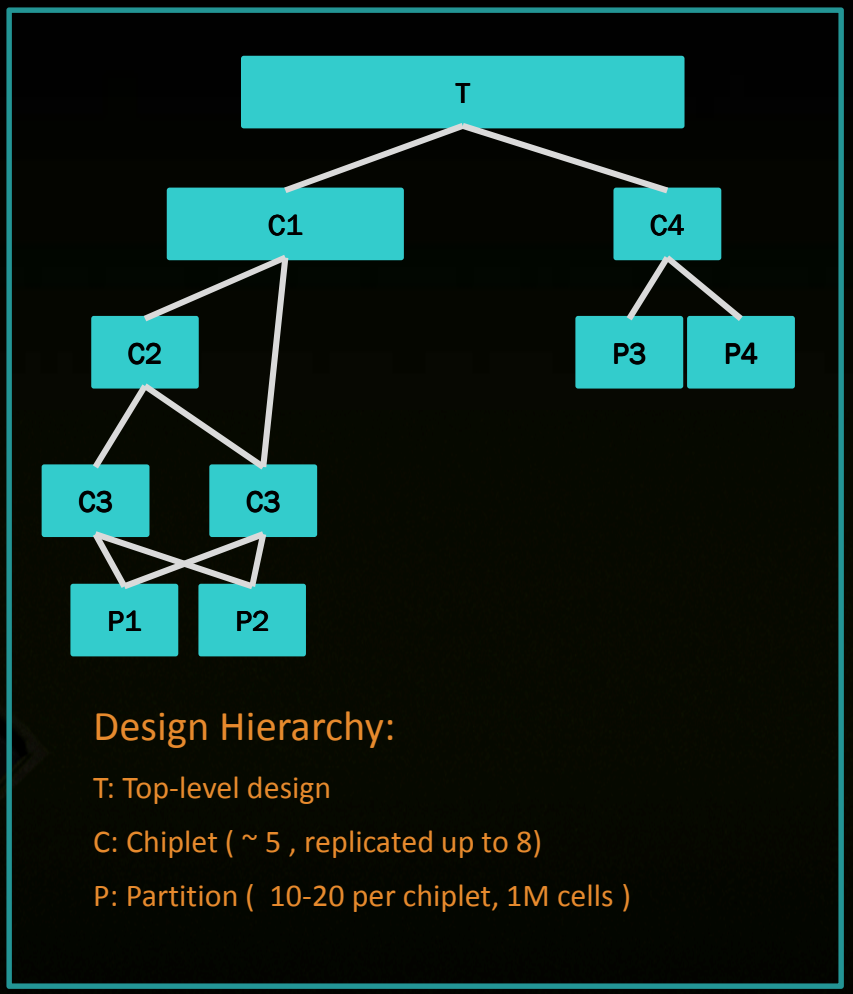
NVIDIA

Overview

- Sign-off workflow
  - Challenges
- Hierarchical timing
- HyperScale methodology
  - Configuration
  - Data
- HyperScale ecosystems
- Deployment
- Conclusion

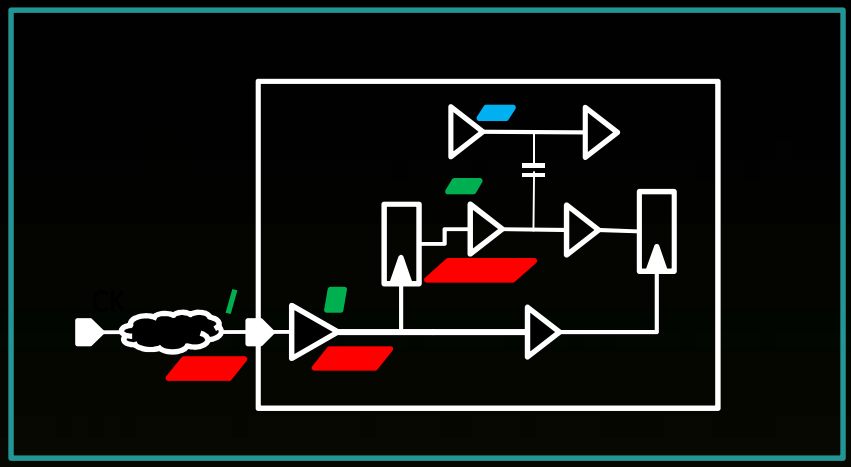
Sign-off workflow

- Design Hierarchy:
  - Child-parent (not instances)
  - Partitions single-parent
  - Chiplets multi-parent
  - Not a tree
- Sing-off workflow:
  - Timing /Fixing at different levels
  - Timing intra
  - Pushdown violation
  - Track ECO
  - Fixing inter/intra
  - Asynchronous/Different cadence

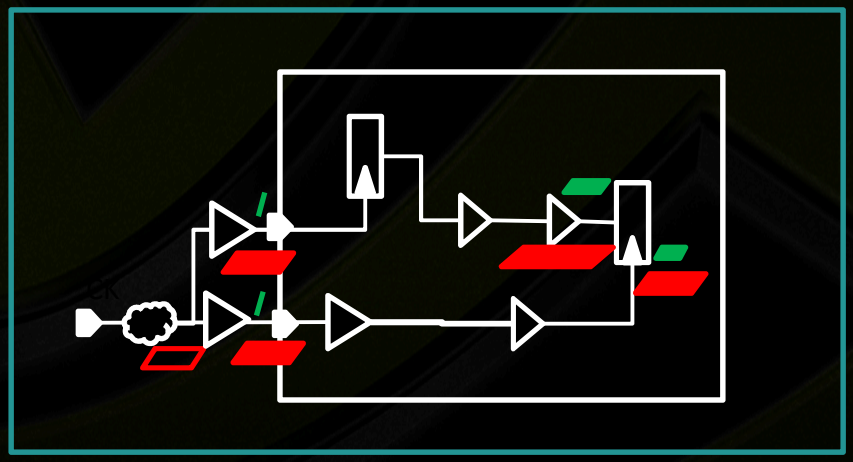


Timing Closure Challenges

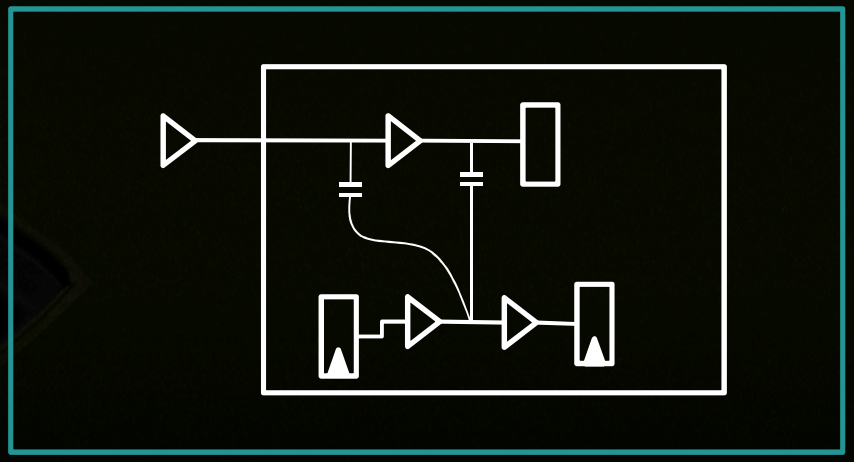
- Inter inaccuracy
- Intra potentially inaccurate too
- Functional vs test
- Setup vs hold



1. SI optimism due to missing clock OCV



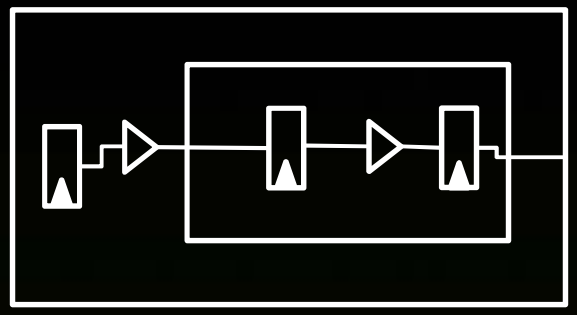
2. External non-common clock path



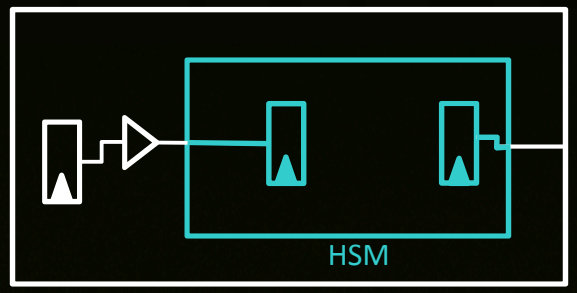
3. Inter to intra coupling

Hierarchical Timing

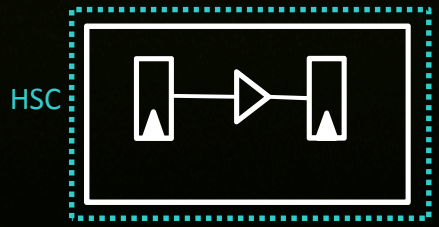
- Hierarchical Timing
  - Accuracy
  - Avoid re-analysis
- HyperScale
  - HSM: HyperScale Model
    - Interface logic netlist/etc
  - HSC: HyperScale context
    - Advanced constraints
- HyperScale principles
  - As of pt_2014.06
  - Need HSM to generate HSC
    - No HSC from Flat
  - Merge HSC when multiple instances
    - No Merged-HSC multi-parent



(a) Flat- Full Hierarchy



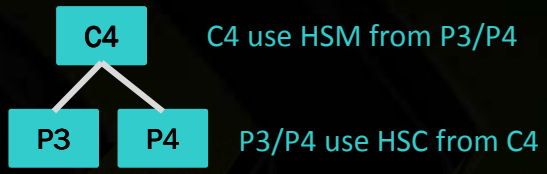
(b) HyperScale- Parent Hierarchy



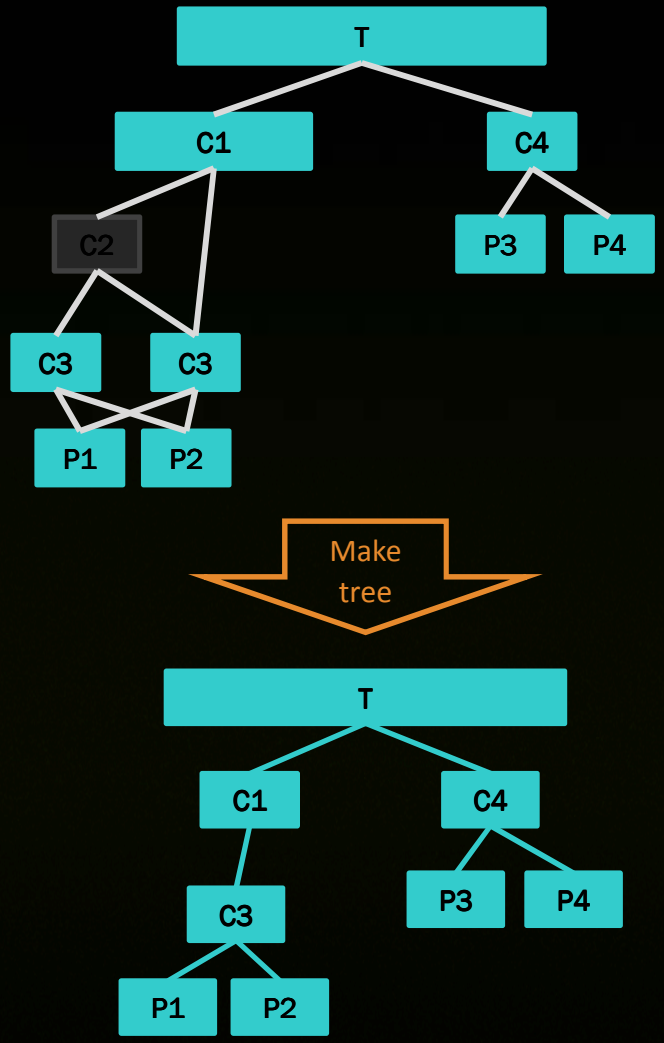
(c) HyperScale- Child Hierarchy

Configuration

- HyperScale recipes
  - Which hierarchies
  - How to run each

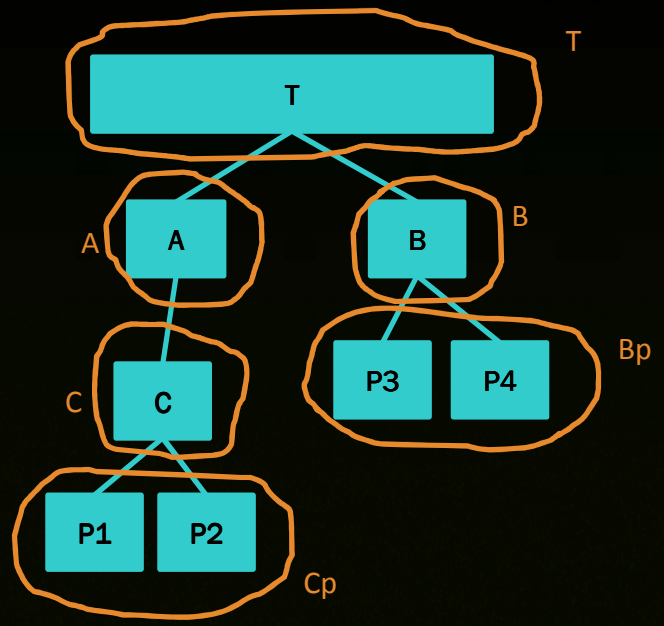


- Challenges
  1. One timing analysis per hierarchy
     1. Use merged-HSC (all instances)
     2. Make tree so no multi-parent
  2. Flexibility & Simplicity
     - Many hierarchies(~ 50 nodes)



Compact recipes

- Pre-generate wouldn't work!
- Need to enable team to generate on their own, very quickly



Solution : Compact Recipes

1. Divide the hierarchies into disjoint sets with compact name
2. Pick the sets in recipe and specify recipe as:

`<root>:<set 1>:<set 2>:…`

Example:

`T:A:B` → Top and its immediate chiplets

`T:C:B` → Use C instead of A

`T:Cp:B` → See through chiplets

`A:Cp` → chiplet and partitions

HyperScale Data

Data dependency

- Find/link data based on recipe
- Extend recipes to include data
- Put everything in Design Configuration File (i.e. DCF)
- Load DCF and fire HyperScale
- Enabled easy HyperScale run

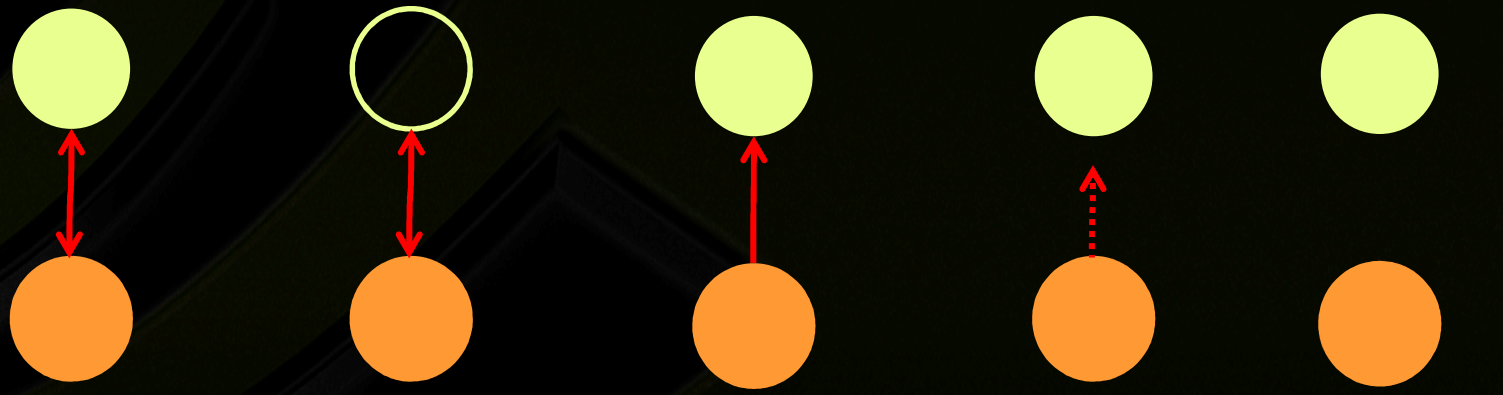
Data Validation/Deletion

- Different runs linked together
- Invalid data should not be used
  - Blacklisting
    - Diligently label bad data
  - Whitelisting
    - Manually make valid data available
    - Generally safer
- New policies for deleting data due to dependencies
  - Use standalone tool sessions

HyperScale Ecosystems

- Relationships between hierarchies
- Producer and/or consumer
  - Risk/Effort/Benefit
  - At least be a producer

- Purpose of a run:
  1. Generate fixing data and HyperScale data (solid below)
  2. Just to generate HyperScale Data (Hollow below)



Deployment

Phase 1 (Intra accuracy)

- Over two years ago
- Partition timing optimistic
- A lot of SI issues pop up at chiplet level
- Use HSC to improve intra timing accuracy
- 25% faster timing closure

Phase 2 (Fix inter)

- Fix inter based on HSC
- Major tool/methodology preparation
- Fix inter/intra setup/hold simultaneously i.e. *jackhammer*
- massive chip/biggest chiplet

	P&R Hand-off TNS (ns)	Post-jackhammer TNS (ns)	TNS Reduction
Inter setup	-145.9	-31.5	74%
Intra setup	-204.5	-41.3	83%
inter hold	-213	-16.3	92%
intra hold	-53.7	-16.1	69%

One round of fixing (11% of Timing Closure) for the most complex chiplet (~ 40 partitions)

Conclusion

- Successfully integrated hierarchical timing
- Abstraction of flow/tool complexities
- Significant impact on timing closure workflow

Future work:

- Adopt latest tool enhancements
- More flexible configurations
- Larger and more efficient ecosystem
- Better data management