



JUNE 23-27, 2024

MOSCONE WEST CENTER  
SAN FRANCISCO, CA, USA

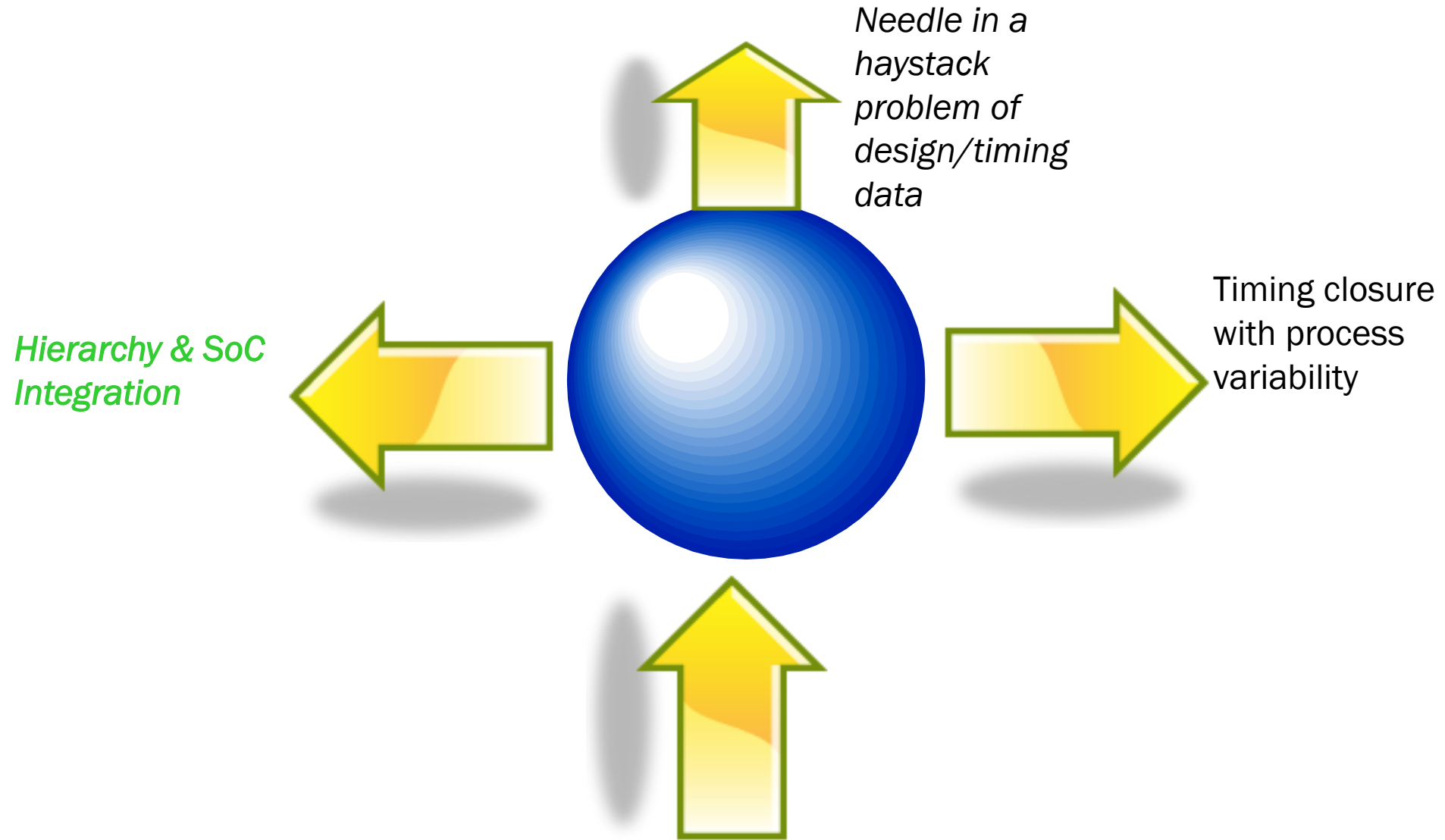
# From Hierarchy to Data Analytics: Exploring Grand Challenges in Static Timing Analysis Over 25 Years

Kerim Kalafala

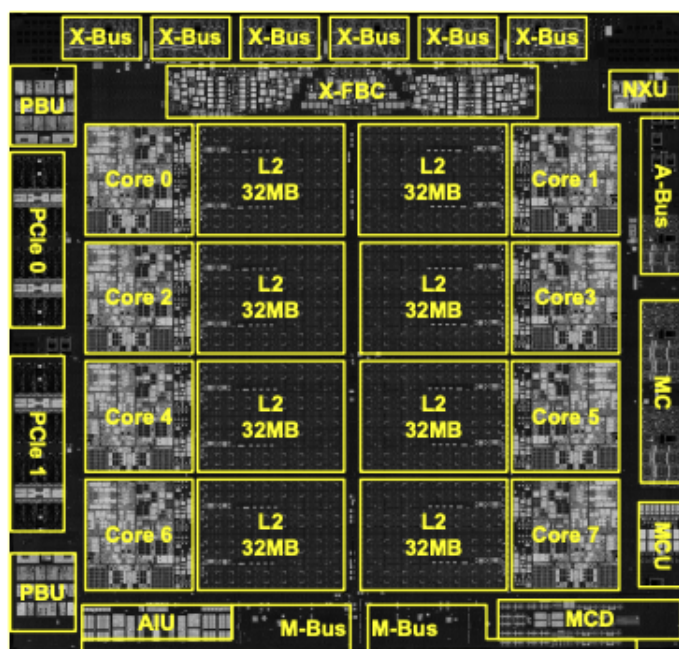
On behalf of IBM/EDA



# Grand challenges for static timing analysis



## 8-Core Processor Chip Detail (Telum)



### ▪ 7nm **FinFET** Technology

- 8 cores per CP
- 18.8 miles of wire
- ~ 23mm x 22 mm
- Up to 2 PCIe buses, and 8 DDIMMS
- 22.5B transistors versus 9.2B on z15

- 8 Telum chips per CPC Drawer in 4 DCMs
- Up to 8 active cores per Telum Chip
- Up to 200 active cores per system
- Added Integrated Accelerator for AI

### ▪ On Core L1 Cache

- Private 128K L1I and 128K L1D

### ▪ On Core/Chip L2 Cache

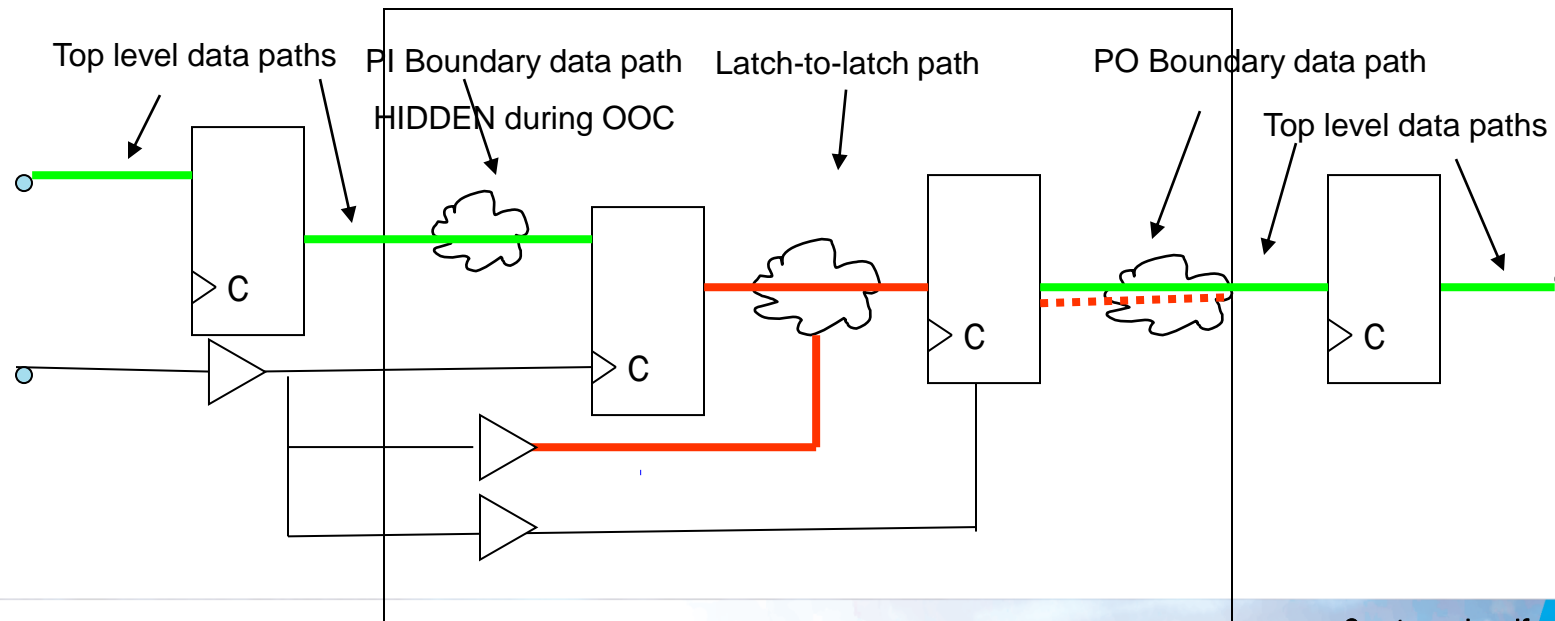
- Each core has access to a private 32 MB cache
- Up to 16MB of each cache can be used by other cores as virtual cache depending on the current activity
- L2 cache of an inactive core becomes shared virtual L3 cache by the active cores of the chip
- L2 cache of an inactive core of another CP can become virtual L4 cache

### ▪ I/O buses

- Each CP chip will support up to 2 Gen-5 PCIe buses

# Divide and conquer via hierarchical abstraction

- Macros are timed Out Of Context (OOC)
  - latch-to-latch paths signed-off at the OOC level (red)
  - PI boundary nets are “hidden”, so they will not show up in the critical path report
- Macro boundary paths are timed at the top level
  - RLM boundary paths are signed-off at the top Level (green)

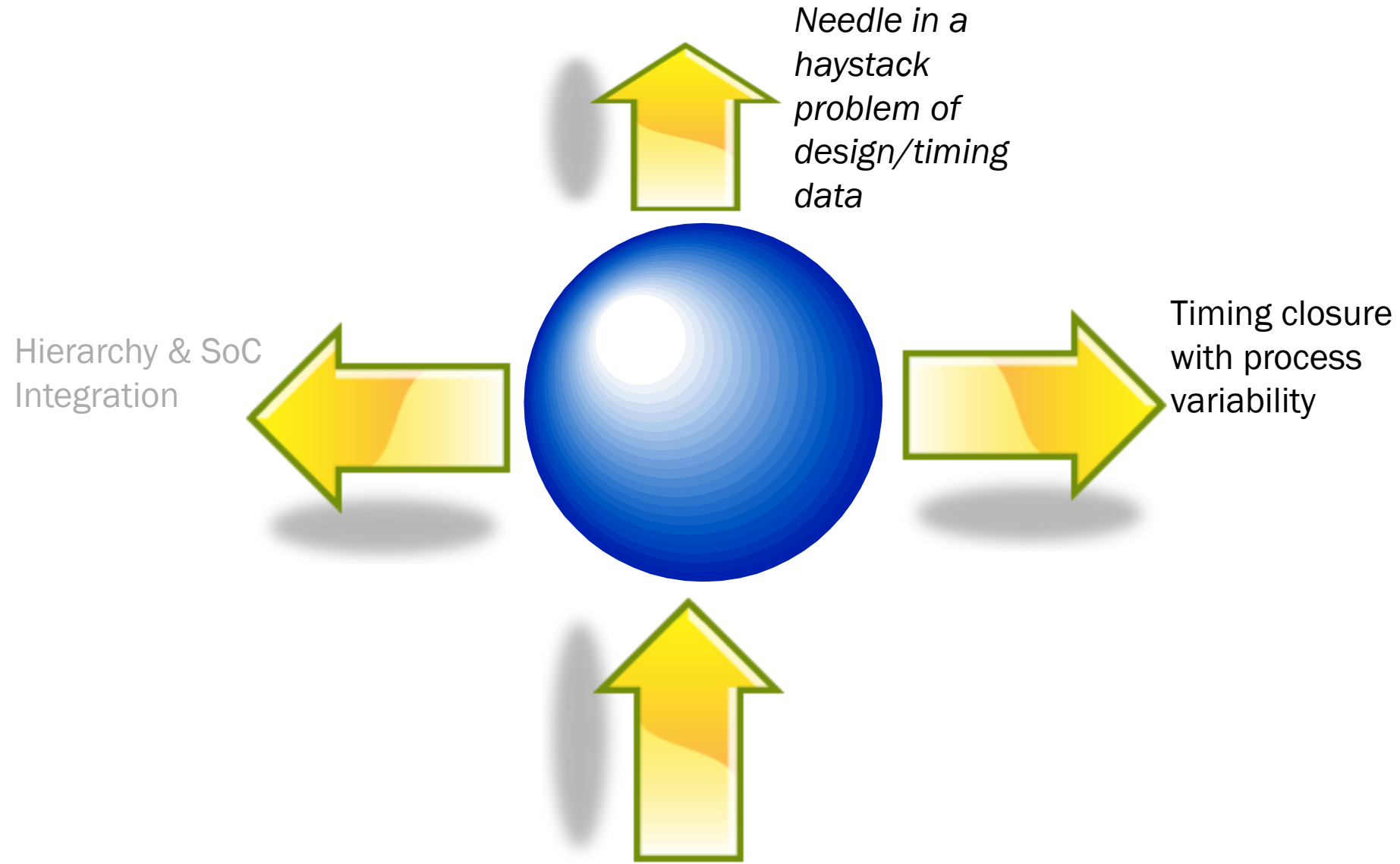


Courtesy : Jennifer Basile

# General observations regarding hierarchy

- Suppose your design consists of:
  - $U$  unique macros
  - Which have an average graph size of  $S$  nodes
  - Which are re-used on average  $R$  times
  - And for which the ratio of abstract/detailed graph size is  $C$
- To analyze this design flat requires STA on overall graph size of:  $U * S * R$
- By leveraging hierarchy we can reduce the total effective computation cost to:
  - $U * S$  (For out of context analysis) +  $U * (S * C) * R$  (For top-level analysis)
- E.g.
  - Let's pick some values:  $U = 50$ ,  $S = 100K$  nodes,  $R = 5$ ,  $C = 0.2$
  - Full flat analysis requires an STA graph of size: 25M nodes
  - By leveraging hierarchy, we reduce this to:
    - 5M (total size of all unique macros) + 5M (size of top-level with abstracted macros) = 10M node computations
  - 2.5X reduction in problem size
  - Closer to a 5X improvement in TAT if all unique macros can be analyzed in parallel across multiple systems
- This is good, but we need more, otherwise:
  - We will be fundamentally limited in macro size  $[S]$  (which makes design closure difficult)
  - And/or we will be limited in the # unique  $[U]$  IP components that can be co-analyzed in a single environment

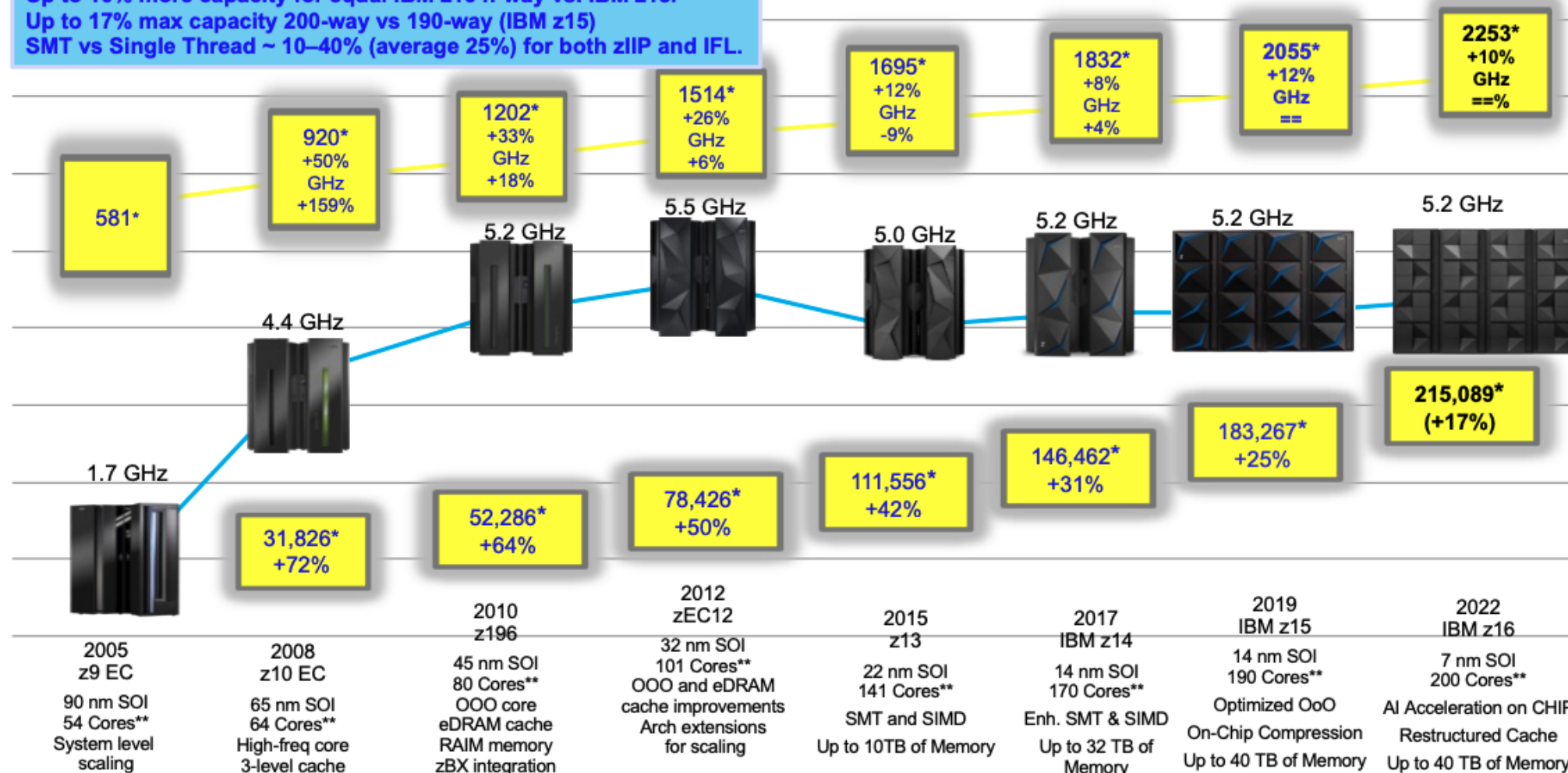
# Grand challenges for static timing analysis



*Post-exponential era in  
performance scaling*

# IBM z16 Continues the CMOS Mainframe Heritage

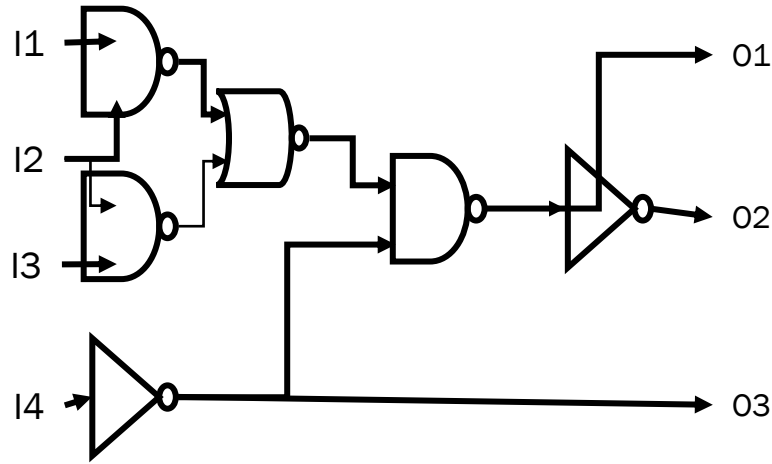
Up to 10% more capacity for equal IBM z16 n-way vs. IBM z15.  
Up to 17% max capacity 200-way vs 190-way (IBM z15)  
SMT vs Single Thread ~ 10–40% (average 25%) for both zIIP and IFL.



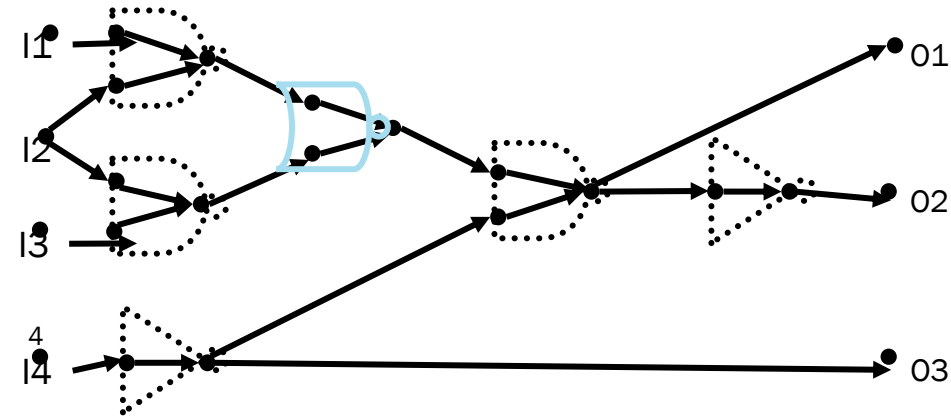
\* PCI Tables are NOT adequate for making comparisons of IBM Z processors. Additional capacity planning required

\*\* Number of PU cores for customer use

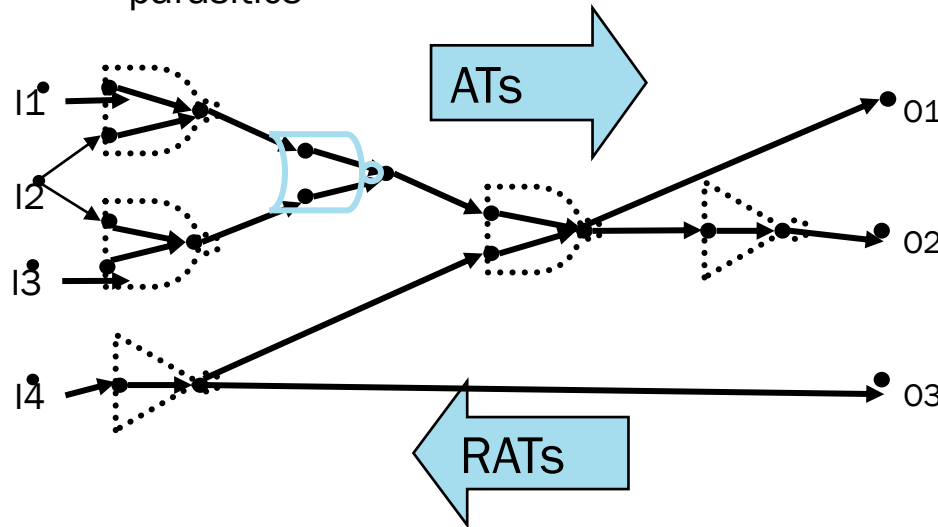
# Graph-based static timing analysis



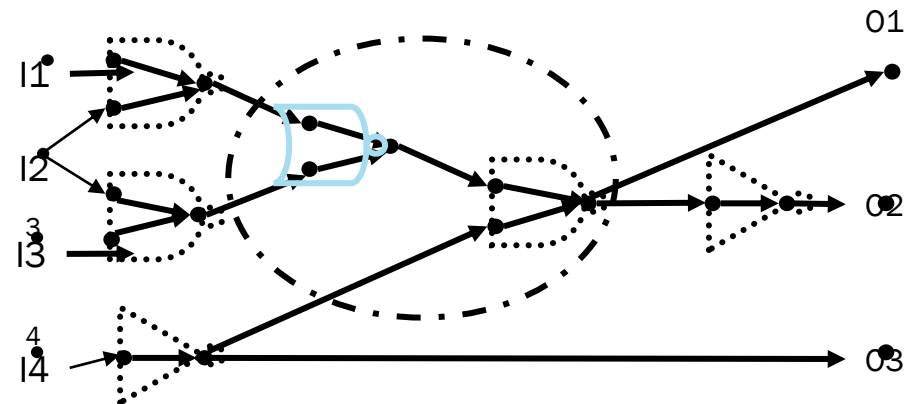
Step 1- Load netlist / assertions /  
parasitics



Step 2. - Create Timing graph (DAG)  
corresponding to the preceding circuit.

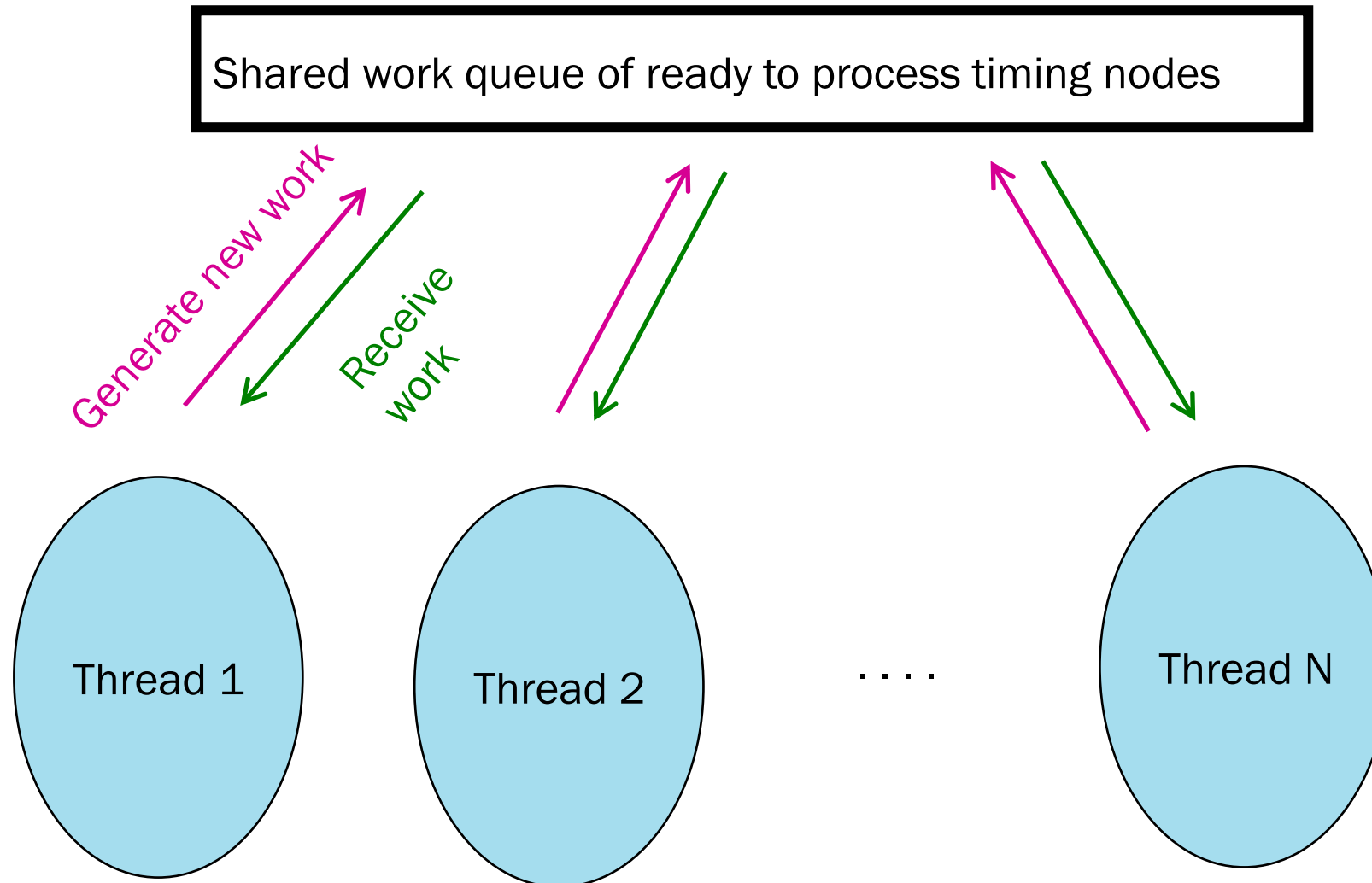


Step 3 - Propagate Arrival Times (ATs) Forward  
and Required Times (RATs) Backward

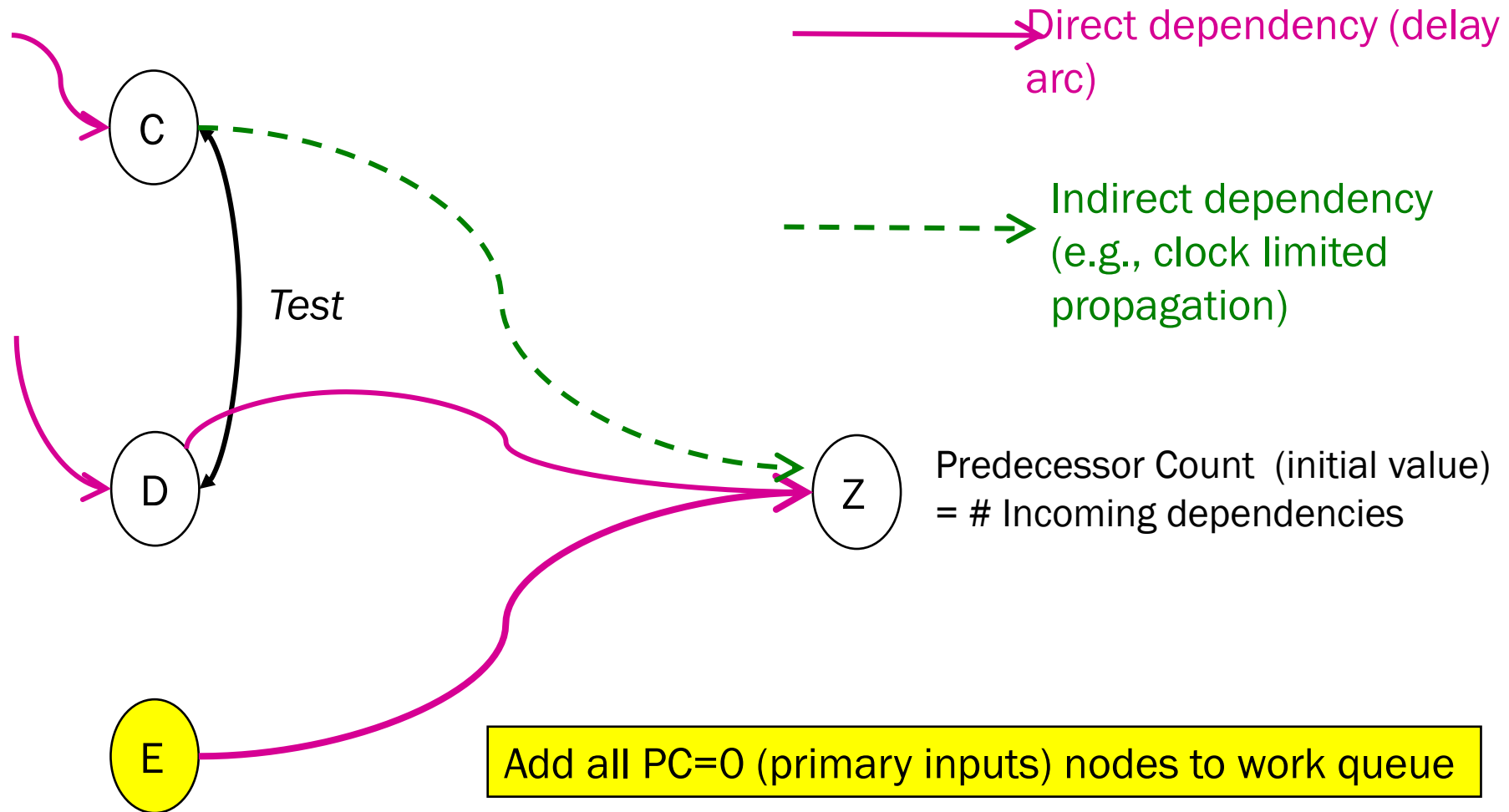


Step 4 - Fix-up and incremental re-  
analysis

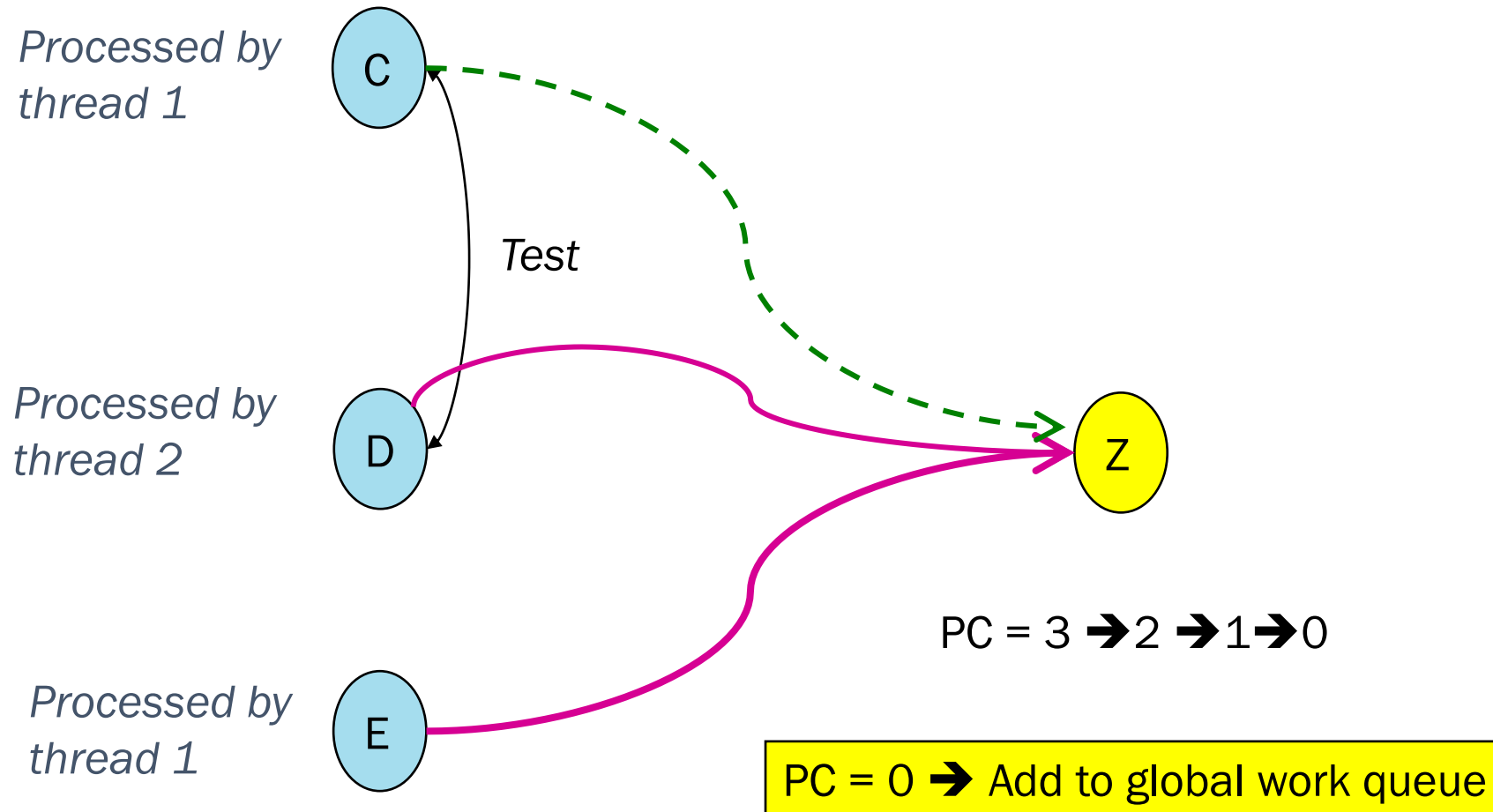
# Key idea for parallelizing base AT/RAT propagation: dynamic work queue processing



# Calculation of initial predecessor counts

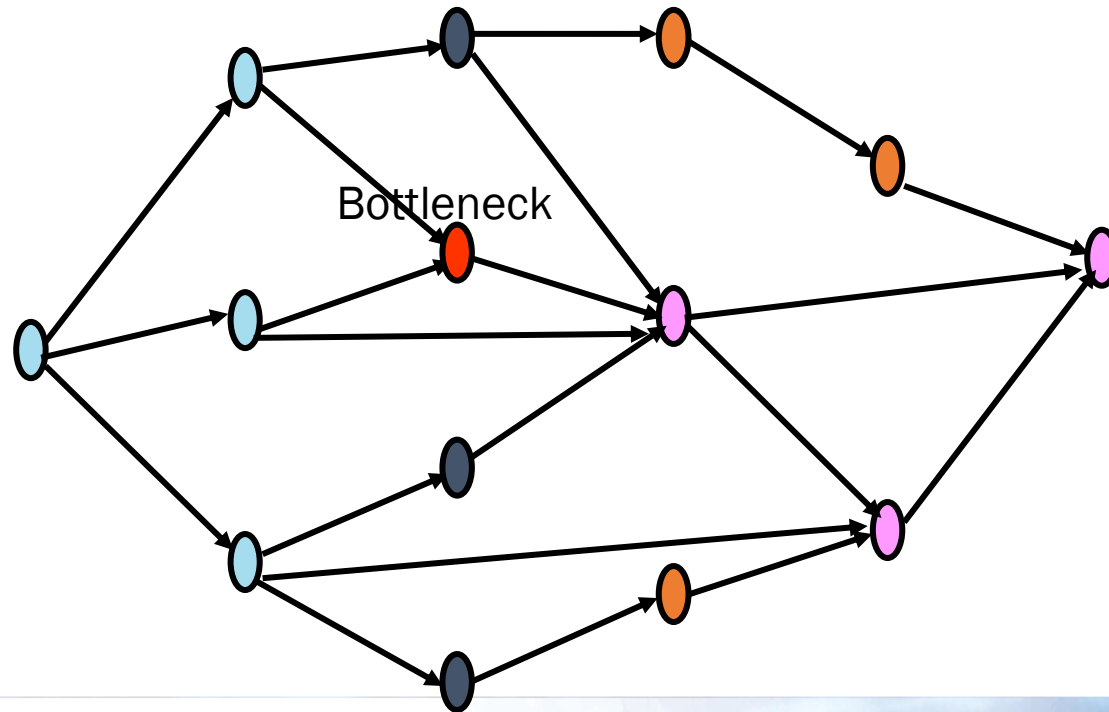


# Generating new work based on updated predecessor counts

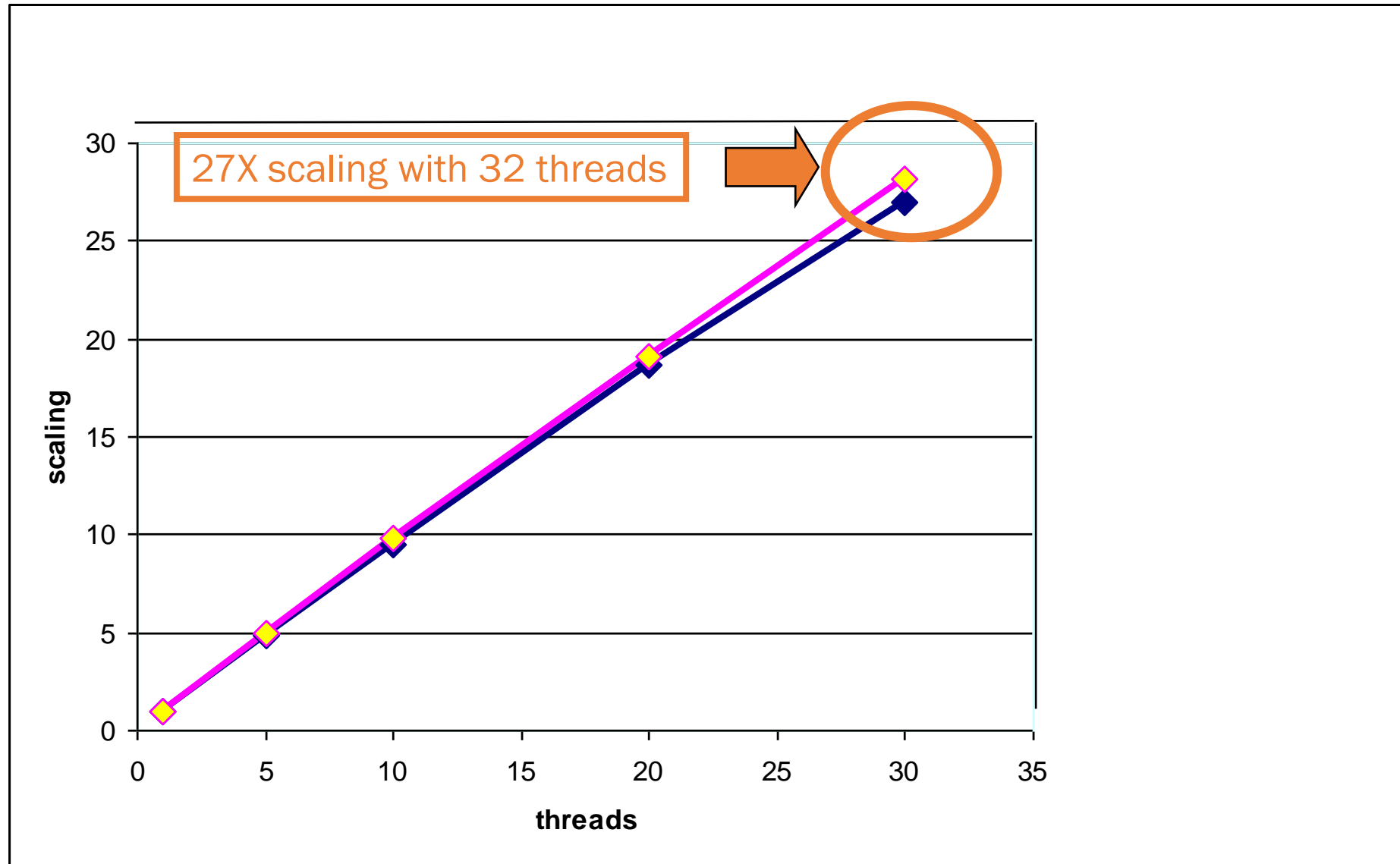


# Benefit of dynamic processing

- Bottleneck (e.g., complex delay calculation)
- Nodes where work can proceed independent of bottleneck



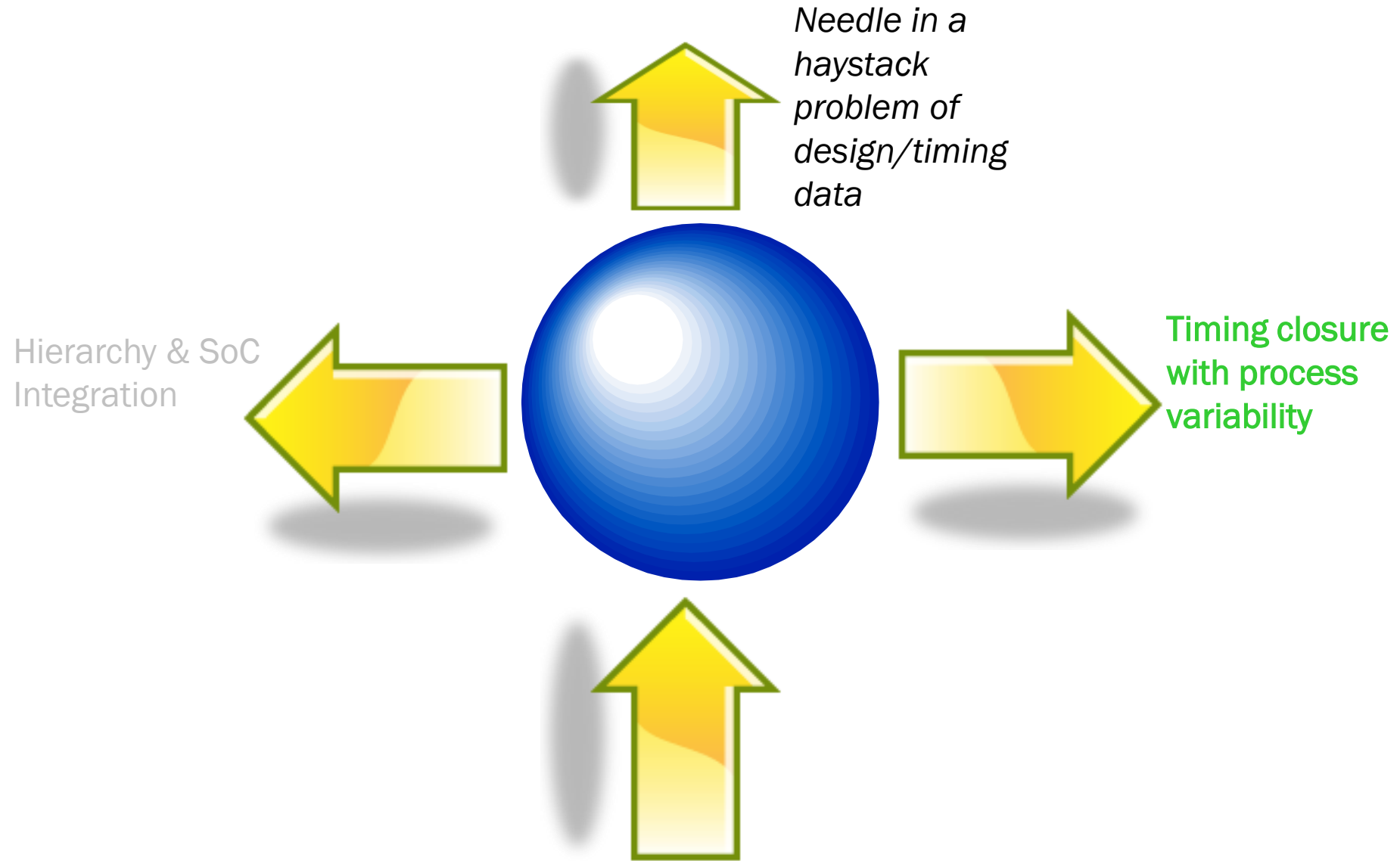
# Multi threaded dynamic arrival time scaling



# Let's go back to our earlier example

- $U = 50$ ,  $S=100K$  nodes,  $R=5$ ,  $C=0.2$
- By leveraging hierarchy, we obtained:
  - 2.5X reduction in problem size
  - Closer to a 5X improvement in TAT if all unique macros can be analyzed in parallel across multiple systems
- Saw we have 16 core SMP machines available and we can achieve an 8X overall improvement from fine-grain parallelism
- Now for STA:
  - **20X TAT improvement** vs. full flat analysis (running on a single machine)
  - **40X TAT improvement** vs. full flat analysis (if we distribute macro analysis across multiple machines)
  - All without relying on improvements in single threaded h/w performance
- Multiple avenues to drive scaling
  - As cores/machine (& memory bandwidth) increases → Deploy multiple threads per job
  - As system memory (& bandwidth) increases → Deploy multiple in parallel jobs per machine

# Grand challenges for static timing analysis

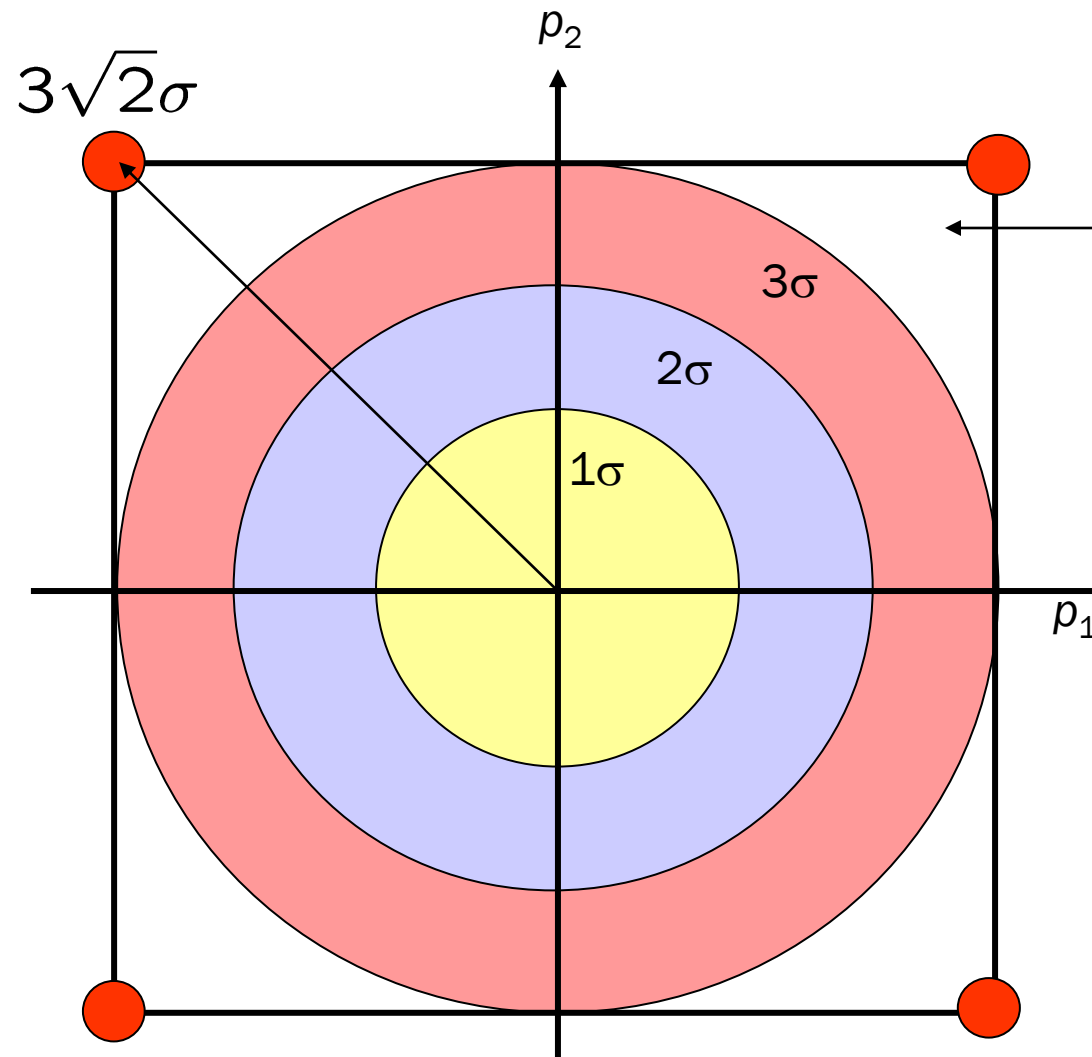


Post-exponential era in  
performance scaling

# Delay impact due to variability

Parameter	Delay Impact
<b>BEOL metal</b> (Metal mistrack, thin/thick vias)	$\pm 15\%$
<b>Environmental</b> (Voltage islands, IR drop, temperature)	$\pm 15\%$
<b>Device fatigue</b> (NBTI, hot electron effects)	$\pm 10\%$
<b><math>V_t</math> and <math>T_{ox}</math> device family tracking</b> (Can have multiple $V_t$ and $T_{ox}$ device families)	$\pm 5\%$
<b>Model to hardware uncertainty</b> (Per cell type)	$\pm 5\%$
<b>N/P mistrack</b> (Fast rise/slow fall, fast fall/slow rise)	$\pm 10\%$
<b>PLL</b> (Jitter, duty cycle, phase error)	$\pm 10\%$

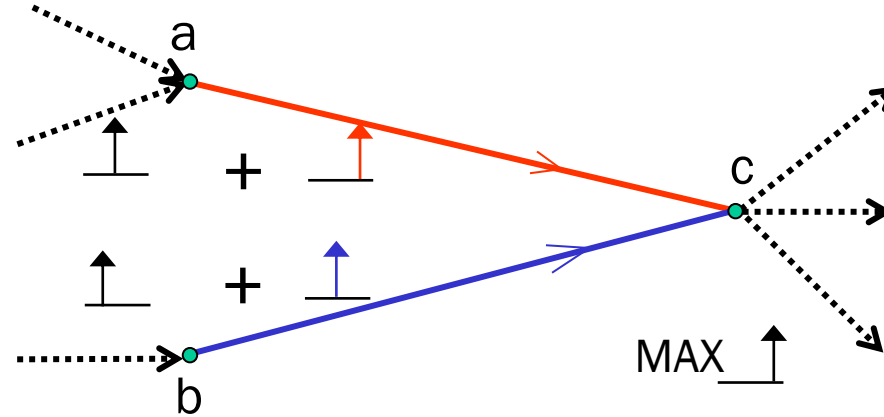
# The (hyper-) sphere vs. the (hyper-) cube



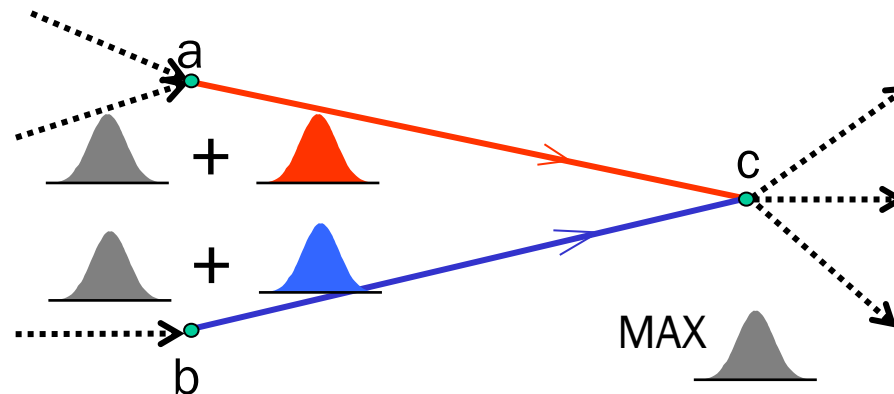
- Worst of all red corners is “exhaustive corner timing”
- Very low probability regions included
  - Performance limited by the most limiting path at the most limiting corner
- Virtually the same parametric yield can be obtained with  $3\sigma$  coverage within the “circle”
- Statistical timing permits some parameters to be worst-cased and others to be treated statistically

# How statistical static timing analysis (SSTA) works

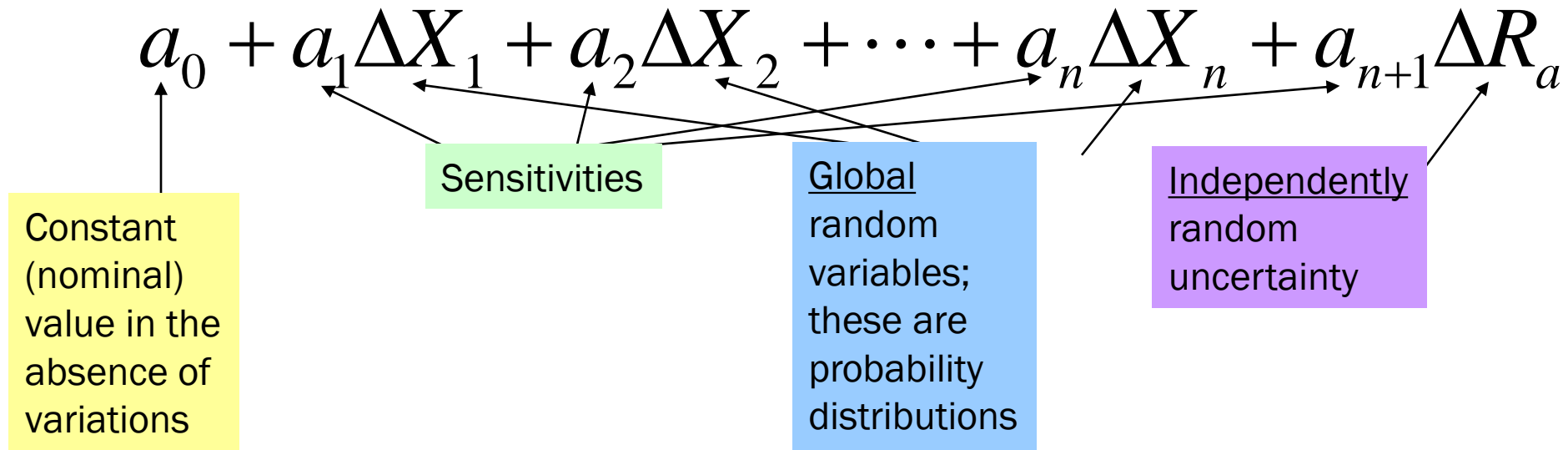
- Deterministic: timing quantities are single numbers



- **Statistical:** timing quantities are probability distributions



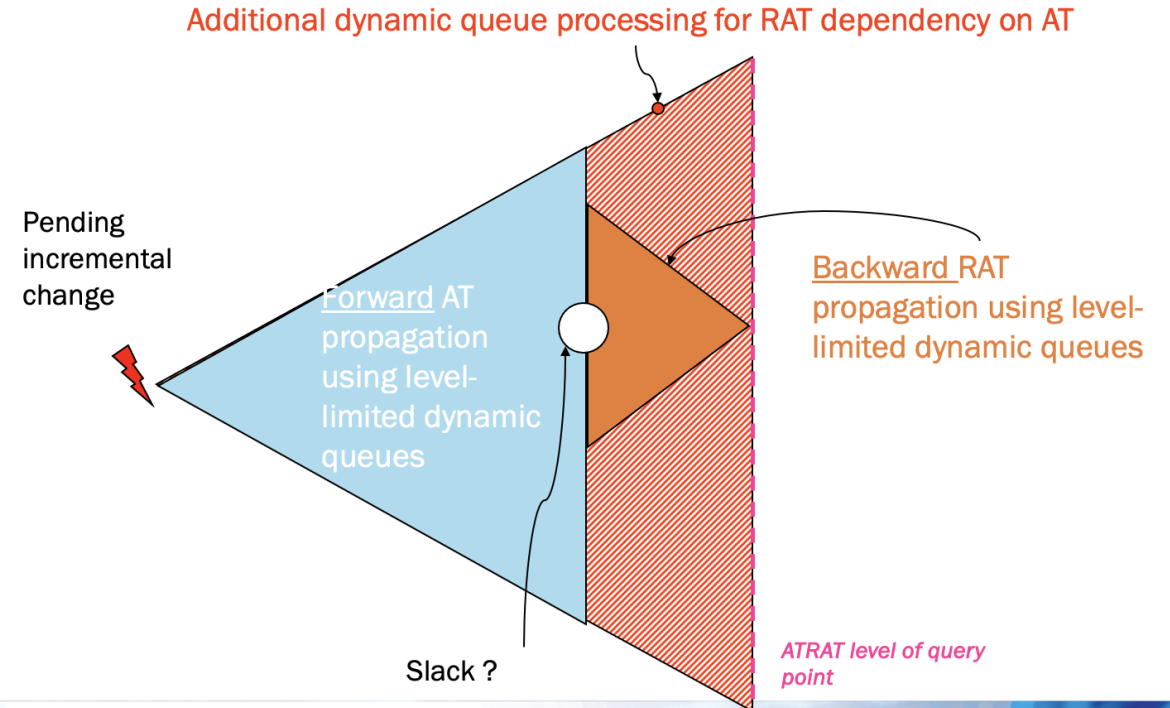
# The first-order canonical form



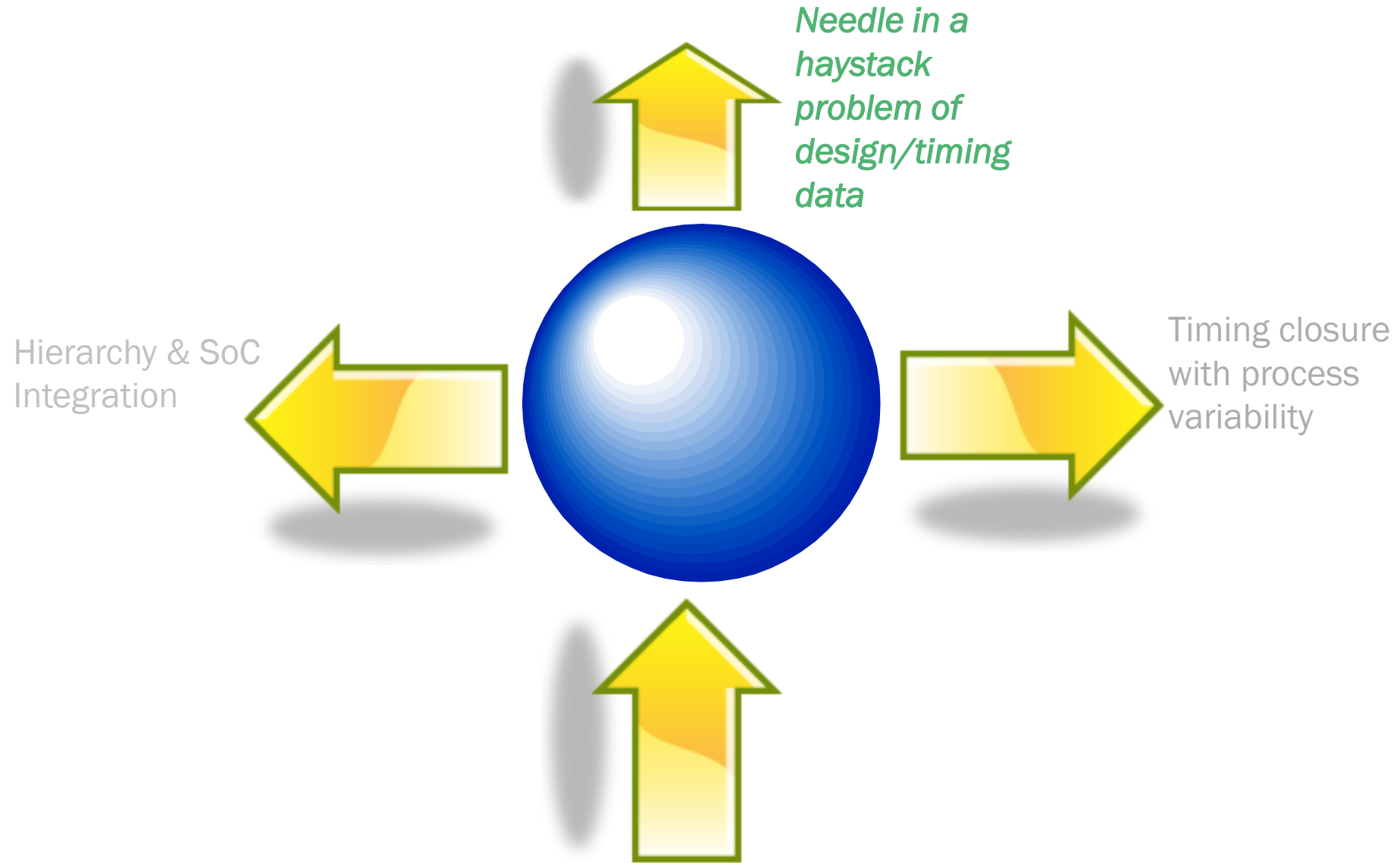
- All timing quantities computed and propagated in a parameterized form
  - ATs, RATs, slacks, slews, guard times, PLL adjusts, delays, CPPR adjusts, coupling adjusts, assertions, ...
  - The correlation between two canonical forms can be computed on-demand based on common dependence on global sources of variation
- This is the key idea that makes statistical timing work
  - Patent filed by IBM in September 2003

# Timing closure needs

- Incremental analysis
  - Fine grain capabilities: Respond quickly in the inner loop of optimization
  - Coarse grain capabilities: Periodically redo CRPR, noise, slack steal, etc.
- Integration
  - Modular plug-and-play architecture
  - Integration into optimizers: we have a physical synthesis tool and a low-perturbation routing-aware polishing tool with our single golden timer integrated at the C++ level;
- Approximation modes
  - Sign off accuracy is expensive → need cheaper modes which are less accurate, but much faster
  - Asserted vs. computed statistical sensitivities, pin vs. pat.. slew mode, local retiming queues



# Grand challenges for static timing analysis

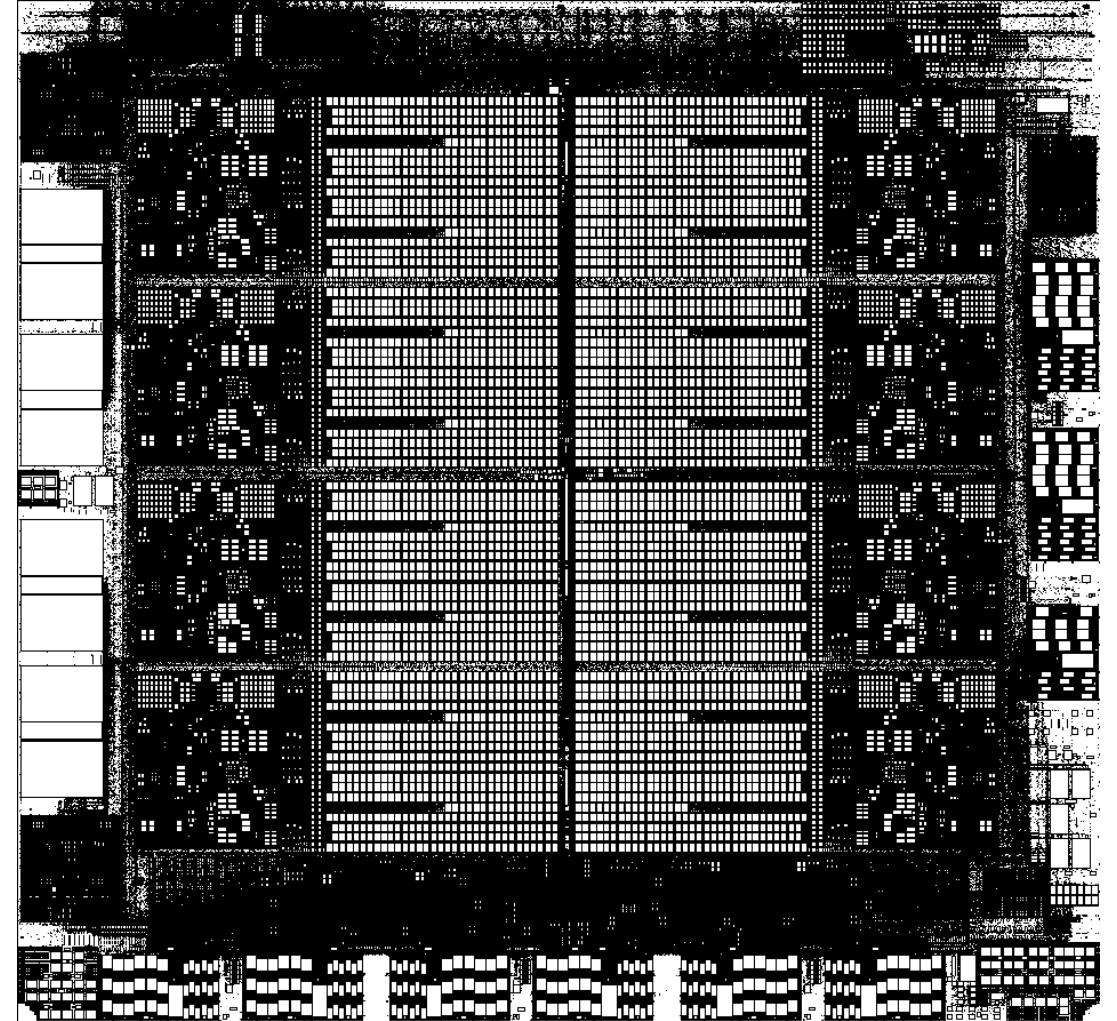


Post-exponential era in  
performance scaling

# The Problem – A haystack of design data

- High-performance microprocessors are complicated devices.
- Processor designs are separated into hierarchical components
- Each of these are analyzed separately and then stitched back together

= A LOT OF DATA (41 GB\*)

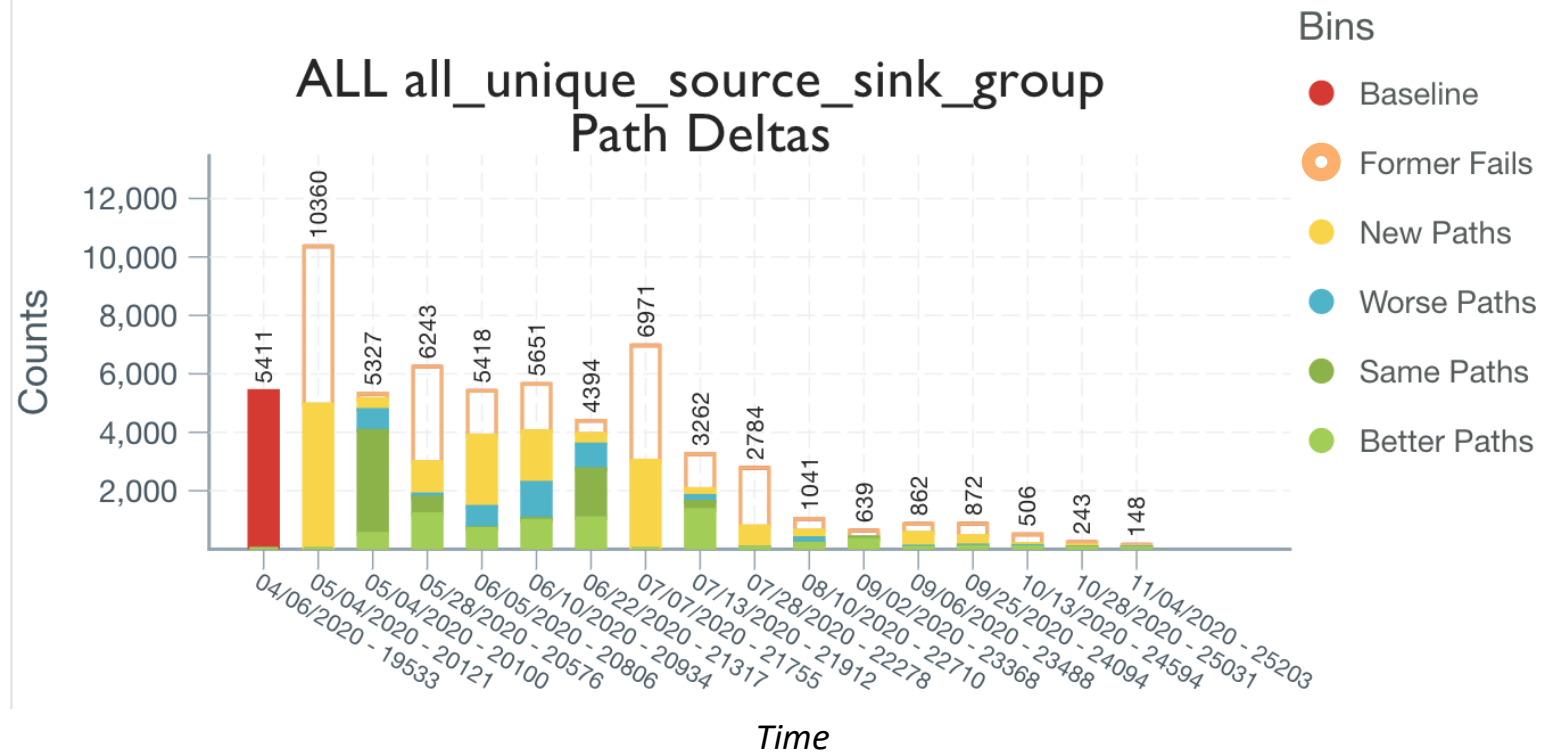


Tellum floor plan

# Step 1: What happened?

Questions:

- Is this the trend I expect?
- How many paths are new?
- How many got worse? Better?
- Why did all of these disappear?



Goal: Form a mental model of the “whole picture”

# Step 2: Why did it happen?

- Questions:
- What do the boundary paths look like?
- Which categories are the high fliers?
- How did these architecture paths change?

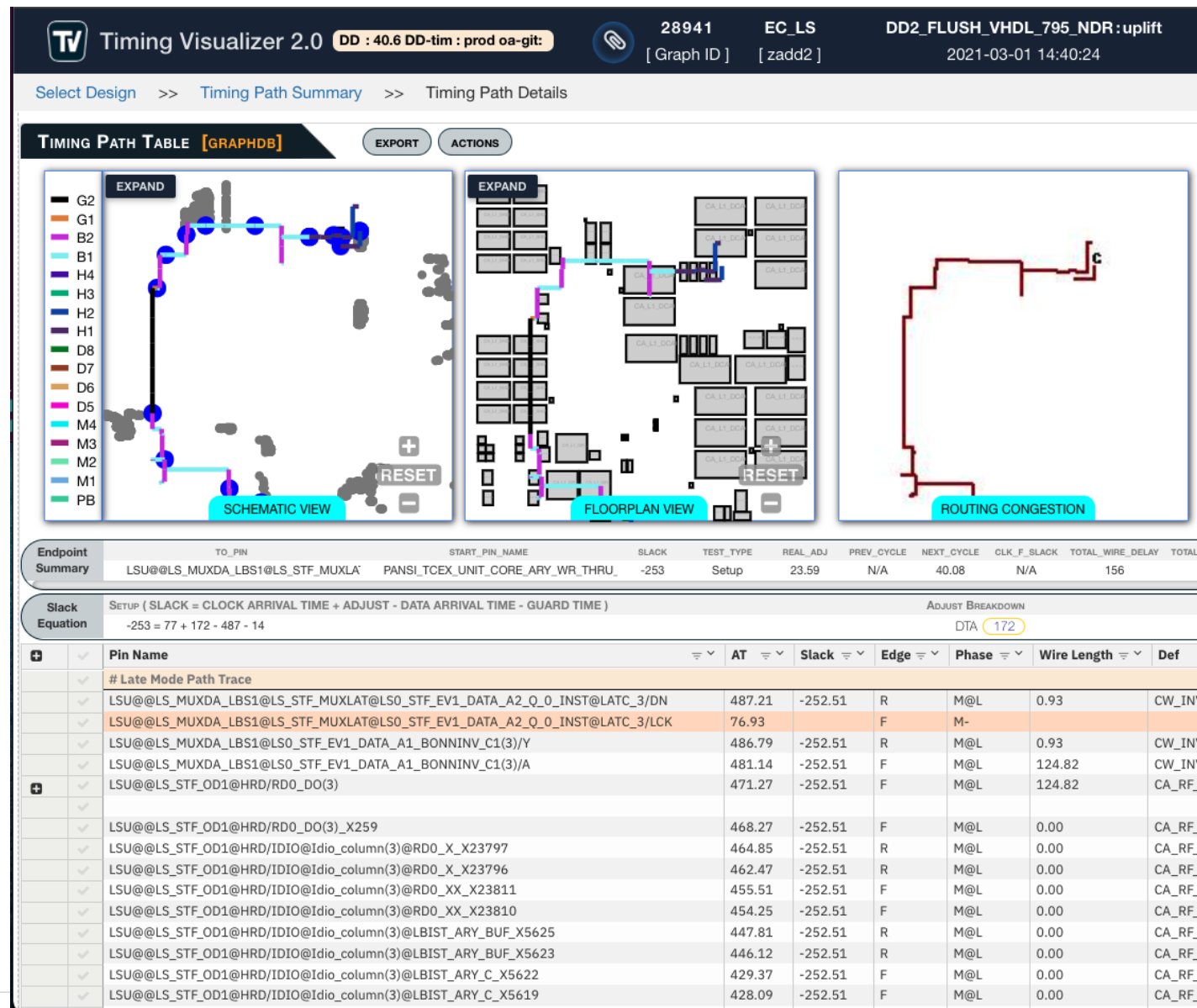
*Goal: Dive into specific design aspects*



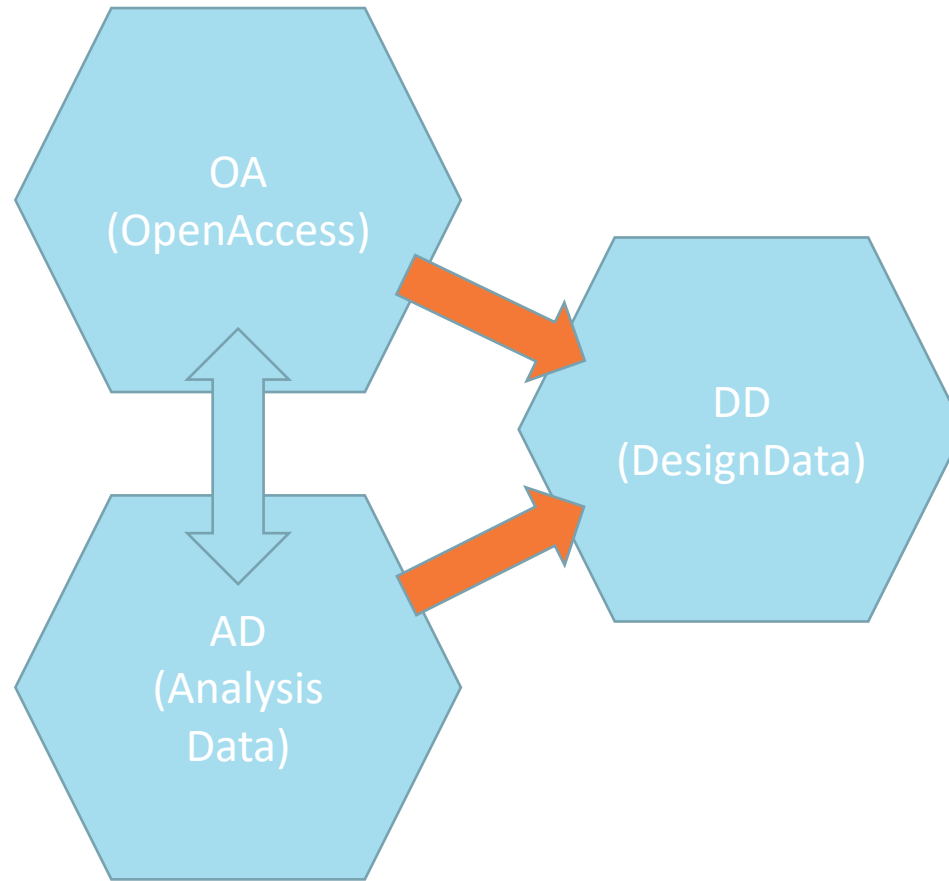
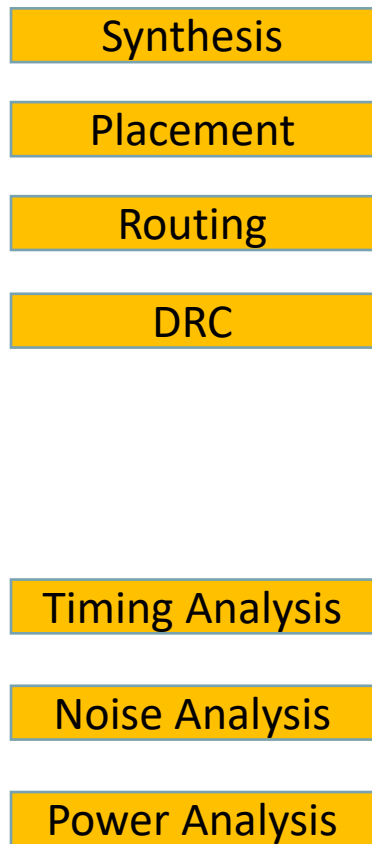
# Step 3: Deep Dive

- Questions:
- Does this require intervention?
- Does this make sense?
- How can we fix this problem?
- Which aspects need to be changed?

*Goal: Dive into specific design aspects*



# Physical Design IA



- DD collects data from multiple disciplines and **links** them together in a manner **easy** to explore
- DD stores all of its data in a **single file** for convenient management
- DD memory/disk footprint is **over 10x** smaller than the memory consumed by the construction flows
- DD load/init times are **over 100x** faster
- DD can load netlist, timing, placement, routing of a one **million** net design in less than one **minute**

# Summary

- The post exponential era is upon us
  - Single threaded performance gains being replaced by large scale integration of multiple cores
  - At the same time, we need to deal with increasing levels of SoC integration, impacts of process variability, etc.
- To succeed at advanced technology nodes, STA must
  - Be highly parallelizable
  - Leverage hierarchy and reuse
  - Account for process variability in a sustainable way (use SSTA)
  - Do all the above in an incremental framework
- Once analysis is complete, these results must be available in a manner which enables design team to see the big-picture, dive into details on-demand, and derive actionable insight
  - Analogy: Google/Waze/Apple-Maps like interface for reviewing any timing run at any level of hierarchy at any point in time, near instantaneously!
- (Speaker's Opinion) Any timing analyzer which cannot satisfy the above will eventually be **doomed**