# Logistics

- I hope you are making progress toward final project
- Lab 5 will be assigned on Thursday
  - A Calibre tutorial in class over Zoom

# Lecture 15: Parasitic Extraction

ECE201A

Some notes adopted from
Andrew B. Kahng

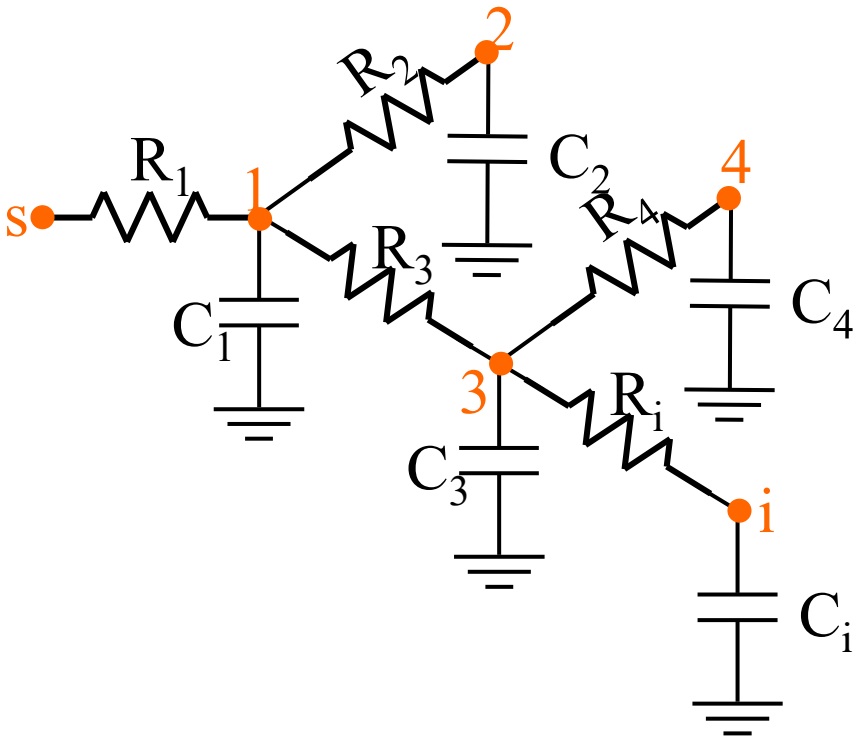
# Layout Parasitic Extraction

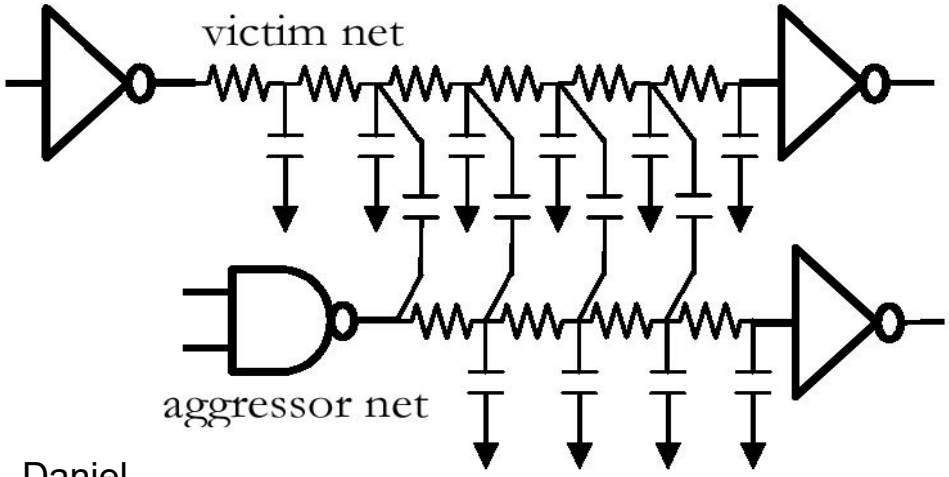
- GOAL: Generate “real” RC model of interconnect
- Necessary step after routing
- Extraction can generate various types of data:
  - RC (dspf/rspf/spef/set_load)
  - Custom Wire Load Model
  - LEF capacitance coefficients
- Account for non-ideal nature of interconnect
  - Wire capacitance
  - Wire and via resistance
- Parasitic information is used in post-layout verification
  - Timing verification of synchronous circuits
  - Functional verification of asynchronous circuits
- Design performance is ultimately limited by parasitics

# Uses of Parasitic Estimation

Example: to produce RC tree network for Elmore or other moment-based delay analysis

Example: to produce RC network for crosstalk analysis



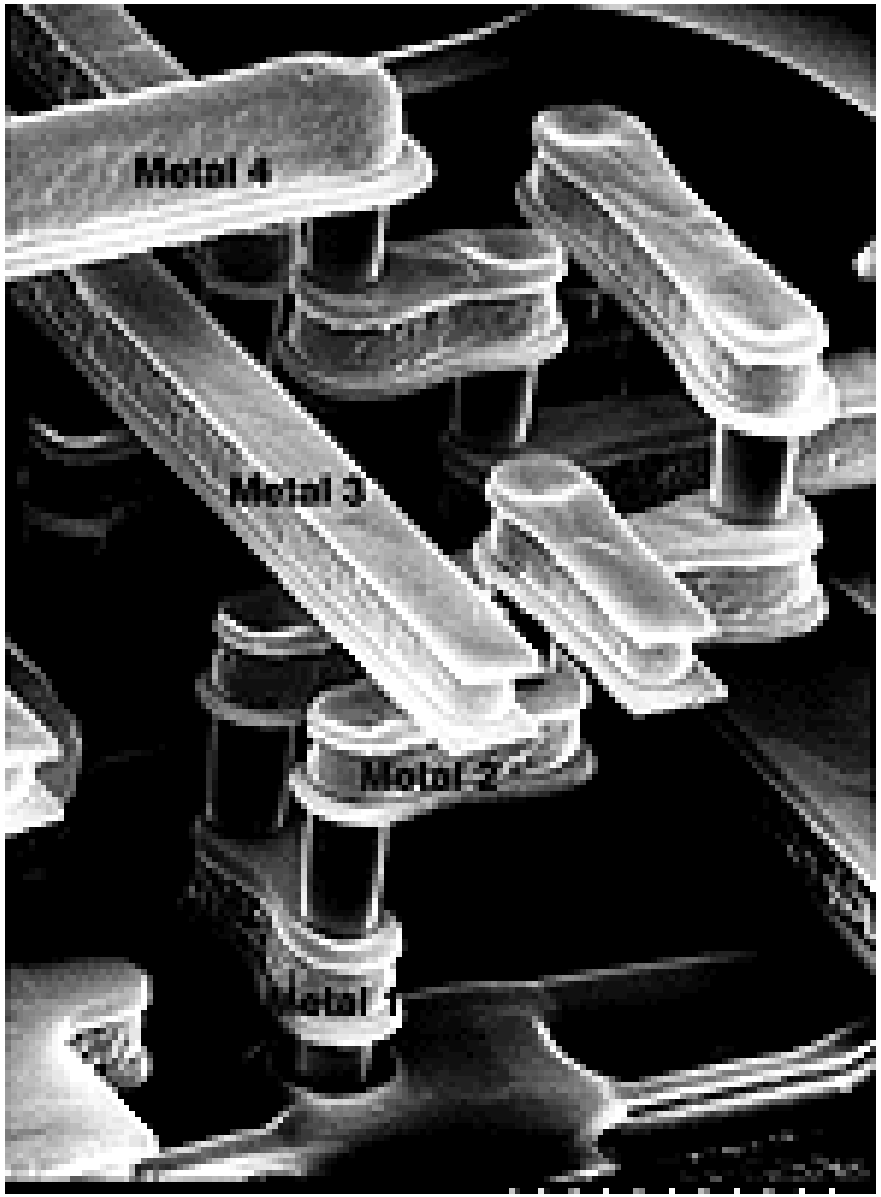




Slide courtesy L. Daniel

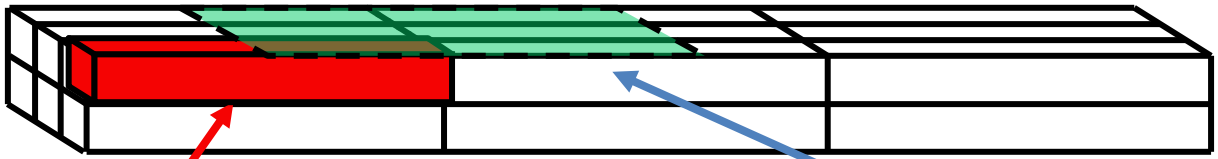
# Post-Layout Parasitic Extraction

- Must perform after routing
- Account for non-ideal nature of interconnect
  - Wire capacitance
  - Wire and via resistance
- Parasitic information is used in post-layout verification
  - Timing verification of synchronous circuits
  - Functional verification of asynchronous circuits

**Design performance is ultimately determined by parasitics**



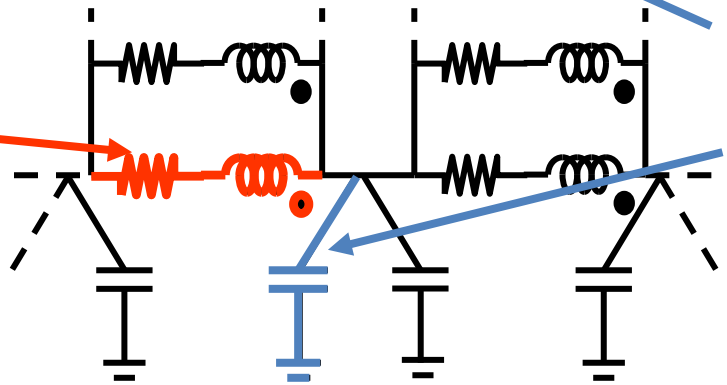
# Parasitic Estimation (Two Basic Steps)



Electromagnetic Analysis

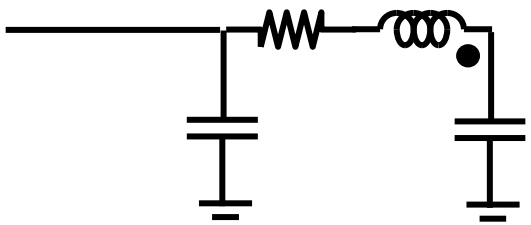
thin volume filaments with constant current

small surface panels with constant charge



millions of elements

Model Order Reduction



tens of elements

Slide courtesy L. Daniel

# Capacitance

http://learn.tsinghua.edu.cn:8080/2003990088/papers/RLC_extraction.ppt

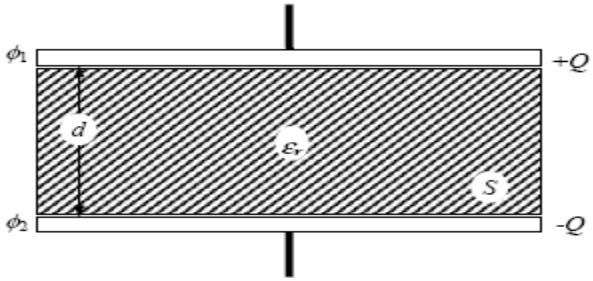
## Parallel-plate capacitor

Voltage: $V = \phi_1 - \phi_2$

Q and –Q are induced on both plates;

proportional to V

This is a ratio : C = Q/V




If plate dimension is large compared to spacing *d*:

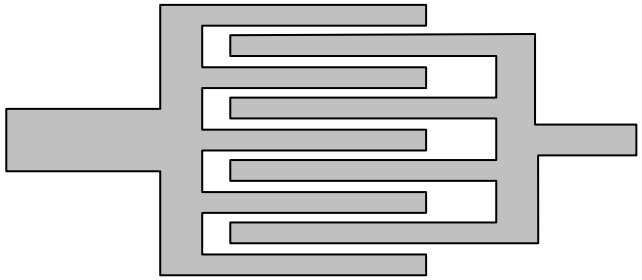
Permittivity (how much charge a given material can store in a unit volume) of free space (vacuum) (F/m)

Relative permittivity of material, e.g., 3-7-3.9 for SiO2

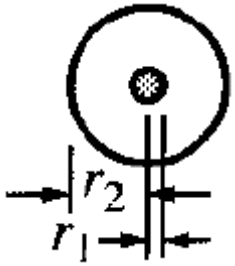
$$C = \frac{\varepsilon_0 \varepsilon_r S}{d},$$

$$\varepsilon_0 = \frac{1}{4\pi \times 9 \times 10^9} = 8.85 \times 10^{-12} C^2 / N \cdot m^2$$

## Other familiar capacitors



interdigitated

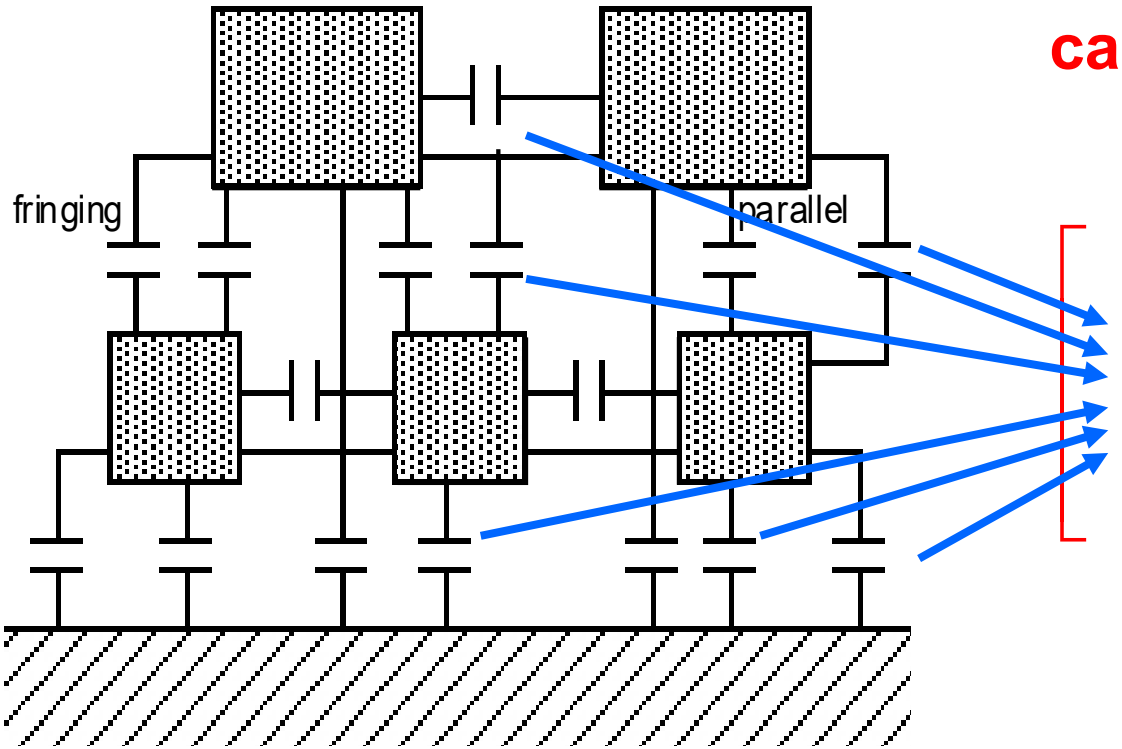



coaxial

# Capacitance Matrix

Given a collection of N conductors (of any shape and dimension), Q = CV

- "3D" RCX: Set voltages on conductors, solve for charge to find C values. Use Finite Element or other methods
  - E.g., Synopsys Raphael




**Find the coupling capacitance matrix C**

$$C = ?$$

$$\begin{bmatrix} v \end{bmatrix} = \begin{bmatrix} q \end{bmatrix}$$

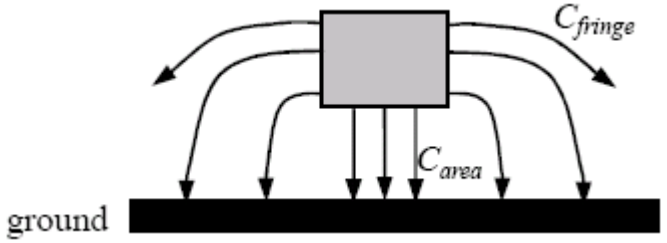
Slide courtesy L. Daniel

# Capacitance Extraction in Practice

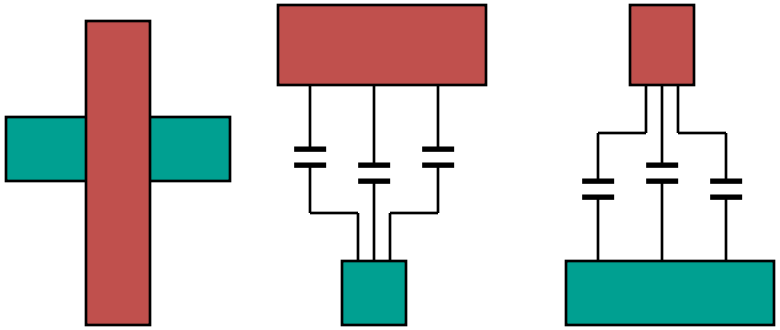
- Simple structures can have accurate analytical formulae
- Unlike resistance, capacitance is a function not only of a wire's own geometry, but of its environment as well
- Errors with respect to full-chip analysis helped by locality of electrostatics
- Methods: analytical, 1D/2D, 2.5D, 3D
  - 1D: Cap function of length (other lateral or z direction geometries need to be "averaged")
  - 2D: Cap accounts for lateral geometries, z needs to be averaged
    - $C = k_1\, Area + k_2\, Perimeter + k_3\, Coupling\_length / Coupling\_spacing$
  - 3D: true 3D solution

Cap/unit length $C = C_{area} + C_{fringe} = \frac{\varepsilon \cdot w}{d} + \frac{2\pi\varepsilon}{\log(d/H)}$

2D methods use fitted approximations to capture 3D effects



# 2.5D Capacitance Extraction



- Compromise: 2.5D extraction
  - Compromise between speed and accuracy
  - Models 3D effects by a combination of *two orthogonal 2D structures*
  - E.g., two cross-section views on the *x*-*z* and *y*-*z* planes, *z* is the vertical axis going through layers
  - Compose the two 2D solutions to construct the 3D solution

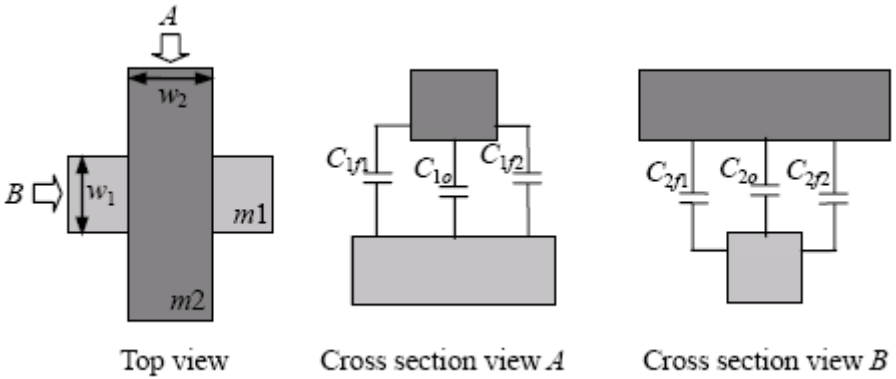
# Capacitance Extraction in Practice

http://learn.tsinghua.edu.cn:8080/2003990088/papers/RLC_extraction.ppt

Commercial tools

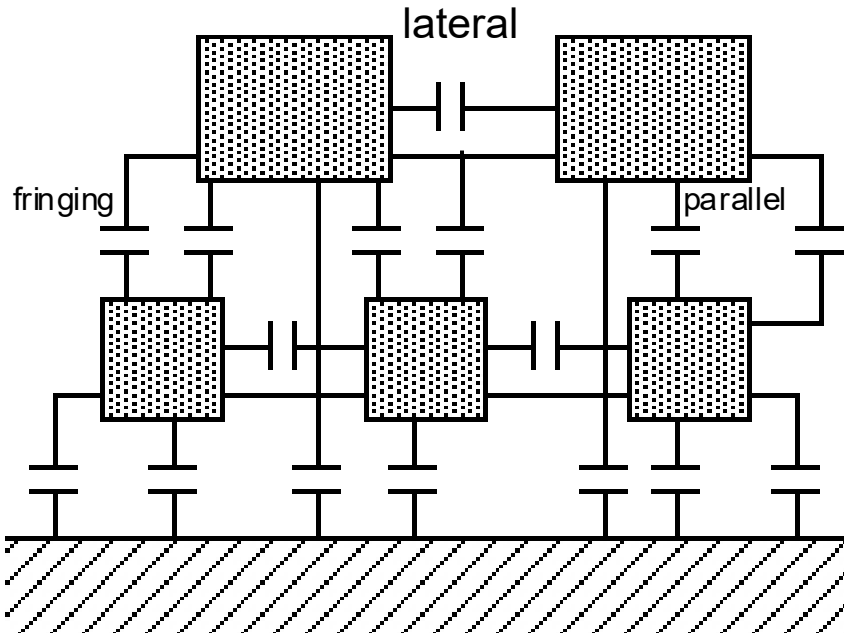
Task: full-chip, full-path extraction

Goal: error ≤10%, runtime ~ overnight for given process




$$C_{m1,\,m2} = C_A \times w_1 + (C_B - C_{2o}) \times w_2$$

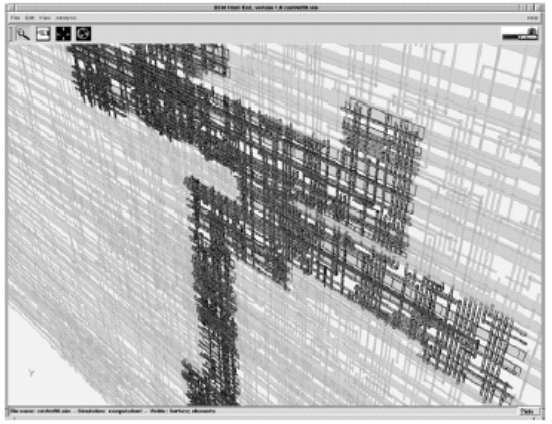
Error > 10%




From "Digital Integrated Circuits", 2nd Edition, Copyright 2002 J. Rabaey et al.

# How Modern Chip-Scale (quasi) 3D Capacitance Extractor Works

- Technology pre-characterization
  - generate coefficients with 3D field solver for "representative" sample of patterns
  - Patterns = cross-sections through "tunnel" that contains a section of the victim net
  - For given process, generate geometric patterns
    - Reduce number of geometric parameters and patterns: symmetry, shielding effects, etc.
      - How big can the pattern library be ?
  - Create a big look-up table
  - Time consuming, but only done once
  - Each layer of interconnect added to the cross-section roughly doubles time for coefficient generation
- Extraction
  - Chop layout into pieces
  - Match patterns to lookup table entries
  - Combine capacitances
- Example commercial tools:
  - Cadence Quantus
  - Mentor Calibre XRC
  - Synopsys StarRC



# LEF Coefficients

- LEF Resistance is sheet resistance: RPERSQ
  - RPERSQ: *any* square wire would have resistance of RPERSQ Ohms
  - To calculate resistance of a wire R = RPERSQ * (aspect ratio of wire)
- LEF capacitance values are 2D
  - CPERSQDIST: overlap cap per unit micron^2
  - EDGECAPACITANCE: fringe cap per unit micron
  - C = (CPERSQDIST x wire width x wire length) + (EDGECAPACITANCE x 2 (wire width + wire length)
- Capacitance coefficients are statistical in deep-submicron
  - Effective area and edge capacitance dependent on surrounding routing
  - Congested blocks have higher effective capacitance

# Formats: SPEF

```
*D_NET *2 6.58027e-05

*CONN
*I *26:A I *C 4 5 *L 0 *D INV_X1
*P *2 I *C 4 6 *L 0

*CAP
1 *26:A 2.82188e-06
2 *2:1 2.82188e-06
3 *2:2 3.00794e-05
4 *2 3.00794e-05

*RES
1 *2:2 *2 2.625
2 *2:1 *2:2 1
3 *26:A *2:1 0.409231
*END
```

# Physical Verification

ECE201A

Some notes adopted from
Andrew B. Kahng
Lei He
Igor Markov
Mani Srivastava
Mohammad Tehranipoor

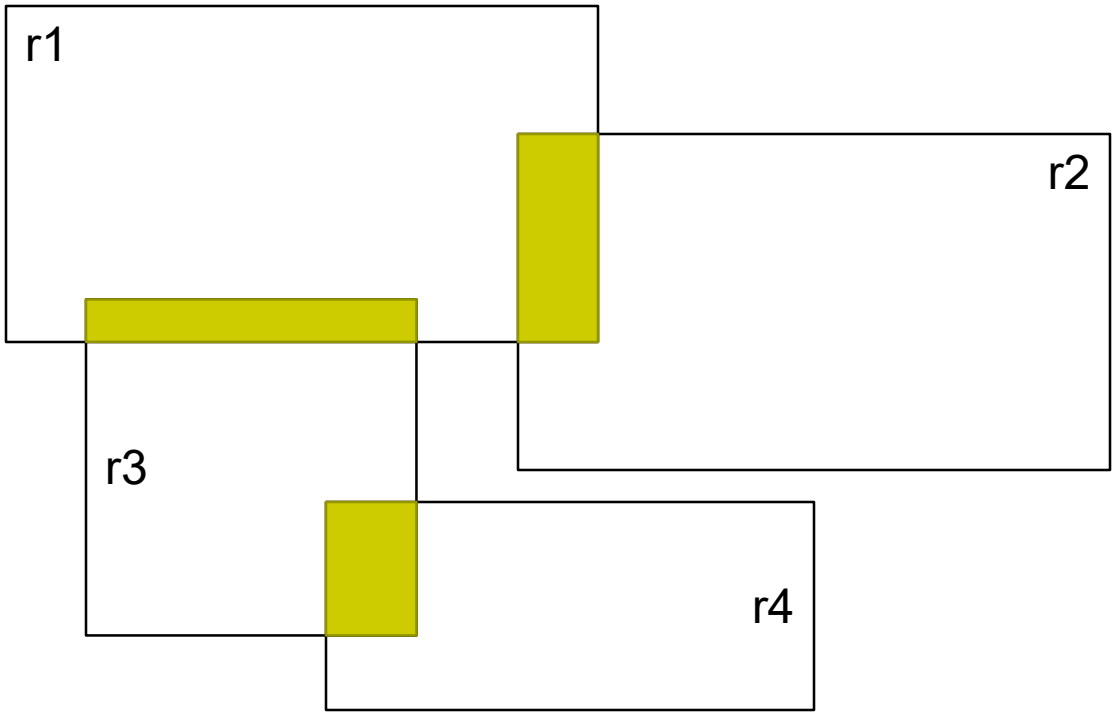
# Polygon Representation

- How to store polygons (really want integers to avoid catastrophic rounding errors)
  - E.g., DBU = 0.5nm; max chip size = 5cm → max x/y coordinate = 1cm/1nm → 18 bits → 32 bit "int" storage = 4 bytes
  - Lets limit ourselves to rectangles → 16 bytes per rectangle
    - Assume layout is packed with 100nmx100nm rectangles in a 1cmx1cm chip → 10B rectangles → 160GB!
    - 10 metal layers → 1.6TB memory to operate on a layout!
- Need a more compact polygon storage
  - What can we do ?

# Reducing Memory

- Leverage hierarchy: create a “master” and then instantiate it just with a “pointer” everywhere else.
  - Downside: reconstructing requires traversal of the hierarchy.
- Most polygons are “small” → it is better to store “deltas” rather than absolute coordinates relative to origin of the polygon.
  - May compress further knowing that most rectangles have a major axis → less storage for delta (or extent) in the minor axis direction

# How to Do Boolean Operations on Polygons ?: A Rectangle Intersection Example



# Intersecting Two Rectangles

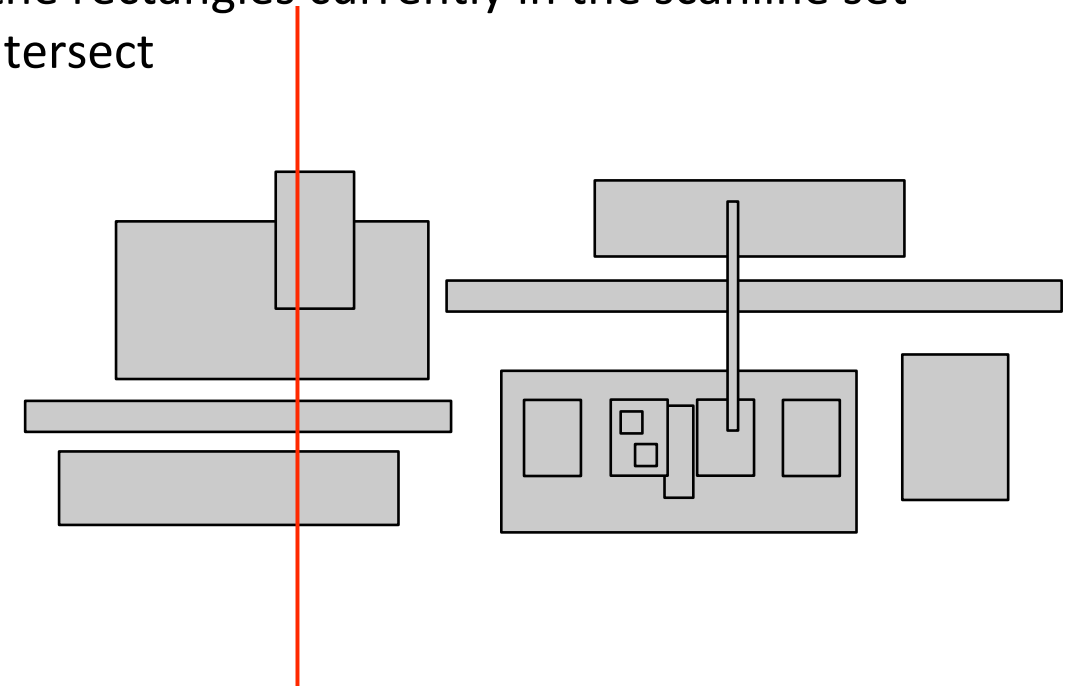
- A rectangle is represented by {LL, TR}:
- Two rectangles overlap if
  - TR_x(1) >= LL_x(2) && TR_x(2) >= LL_x(1)
  - TR_y(1) >= LL_y(2) && TR_y(2) >= LL_y(1)
- Intersection rectangle is
  - TR = min(TR_1, TR_2)
  - LL = max(LL_1, LL_2)

# Overall Algorithm

- Naieve method: Check all rectangle pairs
  - $O(n^2)$
  - Impractically slow (n ~ 1B)
- How do we speed things up ?
  - Do we need to check all pairs of rectangles ?

# Scanline Algorithm: An Event-Based Approach

- Move a vertical “scanline” left to right
  - Sort rectangles by x-coordinate and process in this order stopping at left & right endpoints
  - Add rectangles when they “enter” and delete them when they “exit”
  - Only the rectangles currently in the scanline set can intersect



# Improving Simple Scanline

- Why do we expect scanline algorithm to help ?
- Which direction should we scan ?
  - Left to right vs. top to down
- Improve further by storing the rectangle intervals in an interval search tree → O(n log n) search for intersections
- Overall algorithm is O(n log n) as well
  - Sorting before starting scanline is O(n log n)

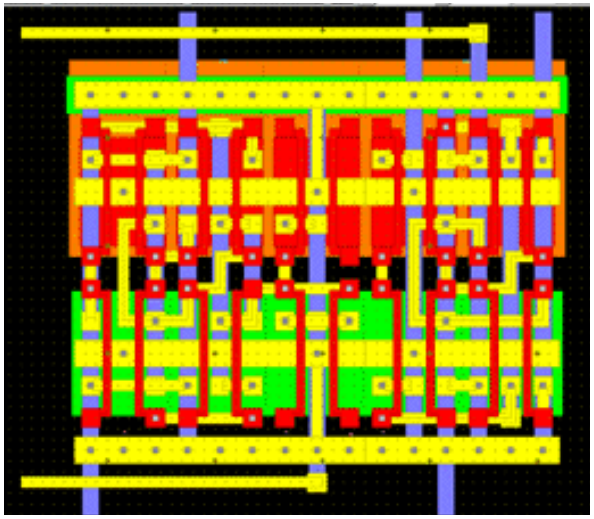
# 5 min break

# Hierarchical Layout Operations

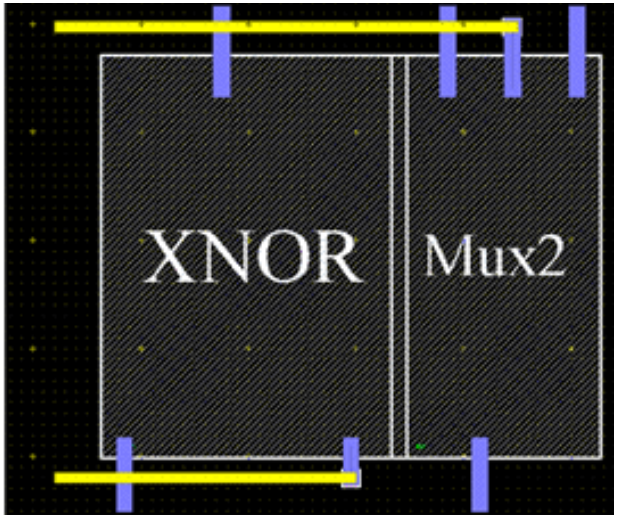
# Hierarchical vs. Flat Layout

UCLA

**Flat**



**Hierarchy**



# Why Hierarchy?

- Less memory
  - Same cell instantiated multiple times
- Less processing time
  - Repeated structure verified “once”
- Better results in verification (designer perspective)
  - E.g. DRC : reporting a much smaller set of violations ➔ easier debugging

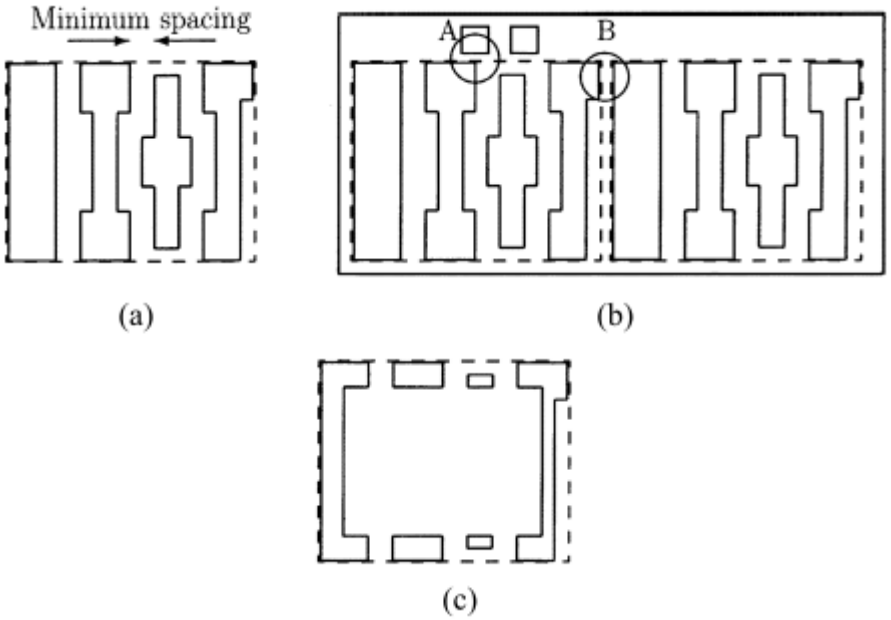
# Challenges in Hierarchical Processing

- Cell neighborhood/instantiation affect results
  - Also Cells may overlap
  - Different orientations of cells
- Migration/Compaction:
  - Multiple instances of same cell ➔ different sets of constraints BUT ONE output cell
  - Working on **hierarchical** view ➔different results from working on **flat** view of same layout
- Hierarchical DRC is NP-complete

# Hierarchical Verification Flow

1. Check all leaf cells.
2. For each cell
   build an abstract:
   a (hopefully) simpler version of the cell that only contains features that are needed for checking cell interactions.
3. Start at hierarchy level 1 (from leaf)
4. Verify cells of current hier. level:
   a. Substitute with cell abstracts
   b. Run **flat** verification algorithm on resulting data
5. Prepare abstract for the next higher level
6. Repeat till top of hierarchy

# Example




a: cell
b: hierarchical layout
c: Abstract of cell

# LVS Flow

LVS: Layout vs. Schematic

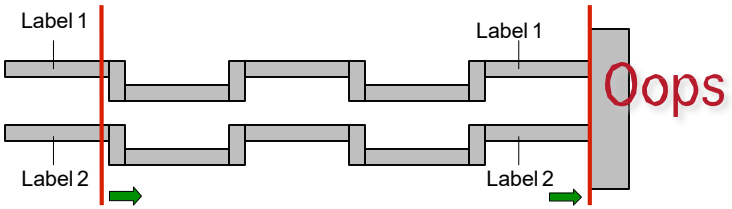
1. Extract transistor-level netlist (SPICE) from polygonal layout
2. Convert post P&R Verilog/DEF into a transistor level netlist
3. Compare the two netlists
    1. Convert them to graphs
    2. Are the two graphs isomorphic (i.e., twisted versions of each other) ?

# Netlist extraction

- Identifies connected components
  - Abutting shapes in same layer
  - Wires connected through vias
  - Terminal connections for MOS transistors
- Connections can be identified during forward scanline processing
  - New label is started with a new (unconnected) shape
  - Two labels are merged when an intersection is detected as scanline goes along

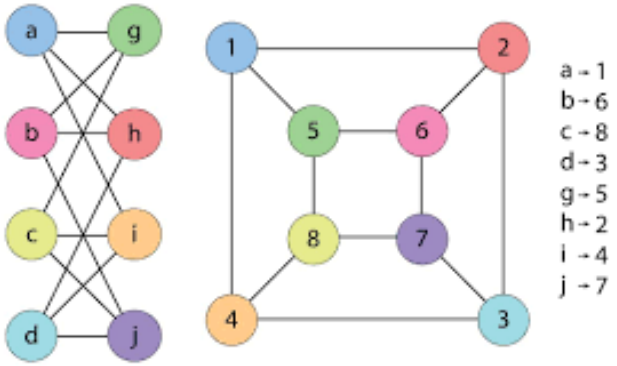
# Netlist extraction (contd.)

- Connections on far end cannot be predicted
  - Problem especially for power/ground connections




  - Deal with this issue by having a backward pass or relabeling after the forward scanline pass
- How to handle multiple layers
  - A scanline for each layer
  - Merge labels when a via is detected

# Graph Isomorphism




- Circuit comparison is equivalent to testing of graph-isomorphism
  - Graph isomorphism: is there any mapping of vertices and edges from Graph 1 to Graph 2 which makes them the same
  - No efficient deterministic algorithm
    - Limited degree of graph nodes except few, e.g. VDD, GND, Clock helps.
    - Can be solved by canonical labeling method (both graphs if isomorphic will have same labeling).

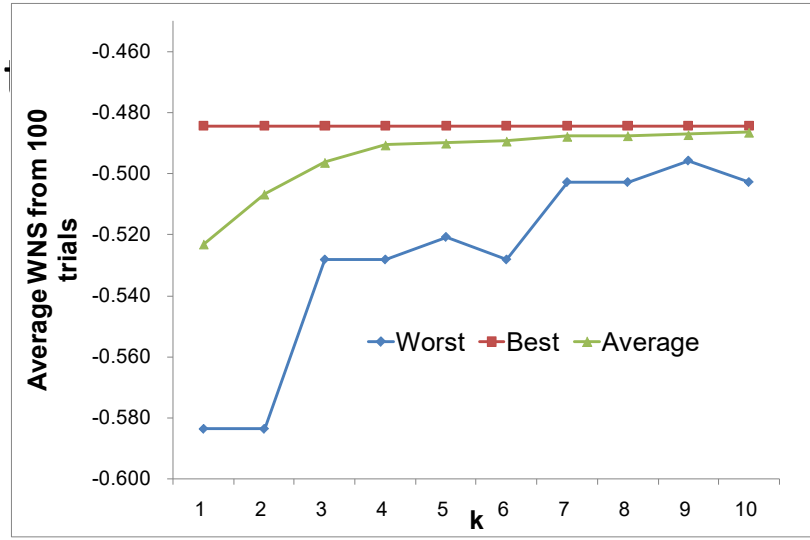
# Some Points about EDA Tools

Q: Do tools give different answers when you run them multiple times?

A: Maybe, but why would that be bad?

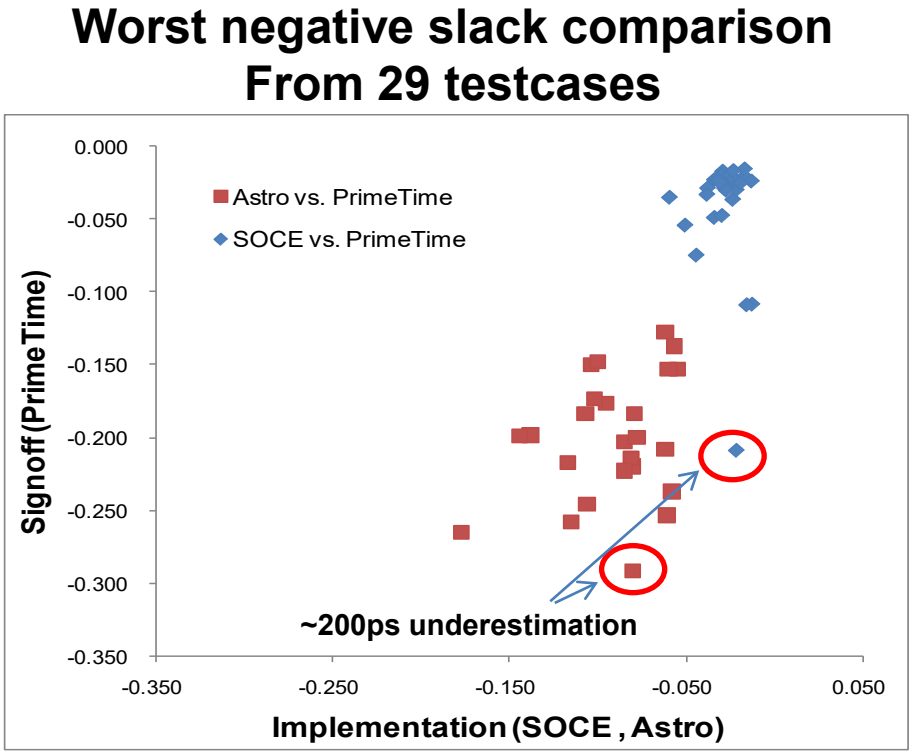
# Noise in Production Design Flow

- Two major sources of noise
  - Miscorrelation in parasitics extractor and timer
  - Suboptimality of heuristic optimization engines
    - Most design optimization problems are NP-hard → Heuristic approaches have been used
    - Heuristics lead to "NOISE" that creates variability in solution quality
- Exploting noise in design flow
  - Use idle machines: we can choose
  - best solution among N different solu
    - Example: Best-of-k method



# Miscorrelation: Implementation vs. Signoff

- Experiment setup
  - Testcases:
    - aes_cipher_top
    - jpeg_encoder
  - Tools
    - SOCE / Astro
- Results
  - Most cases, P&R tools underestimate timing slack; increasing TAT needed to fix violations at signoff
  - There is no clear trend – i.e., not clear what factors cause miscorrelation
- Conventional approaches
  - RC derating in implementation tools to have pessimistic delay



# Inherent Noise: Ignorable Perturbation vs. Results

UCLA

- Slight changes in design constraints can make significant difference in final timing
- Possible knobs to perturb in design constraints
  - Clock cycle time
  - Clock uncertainty
  - IO delay constrants
  - RC values
- Loose timing constraints do not always improve timing
  - 0.1ps change in constraint → > 50ps change in signoff timing
- Noise is really random! → Difficult to predict

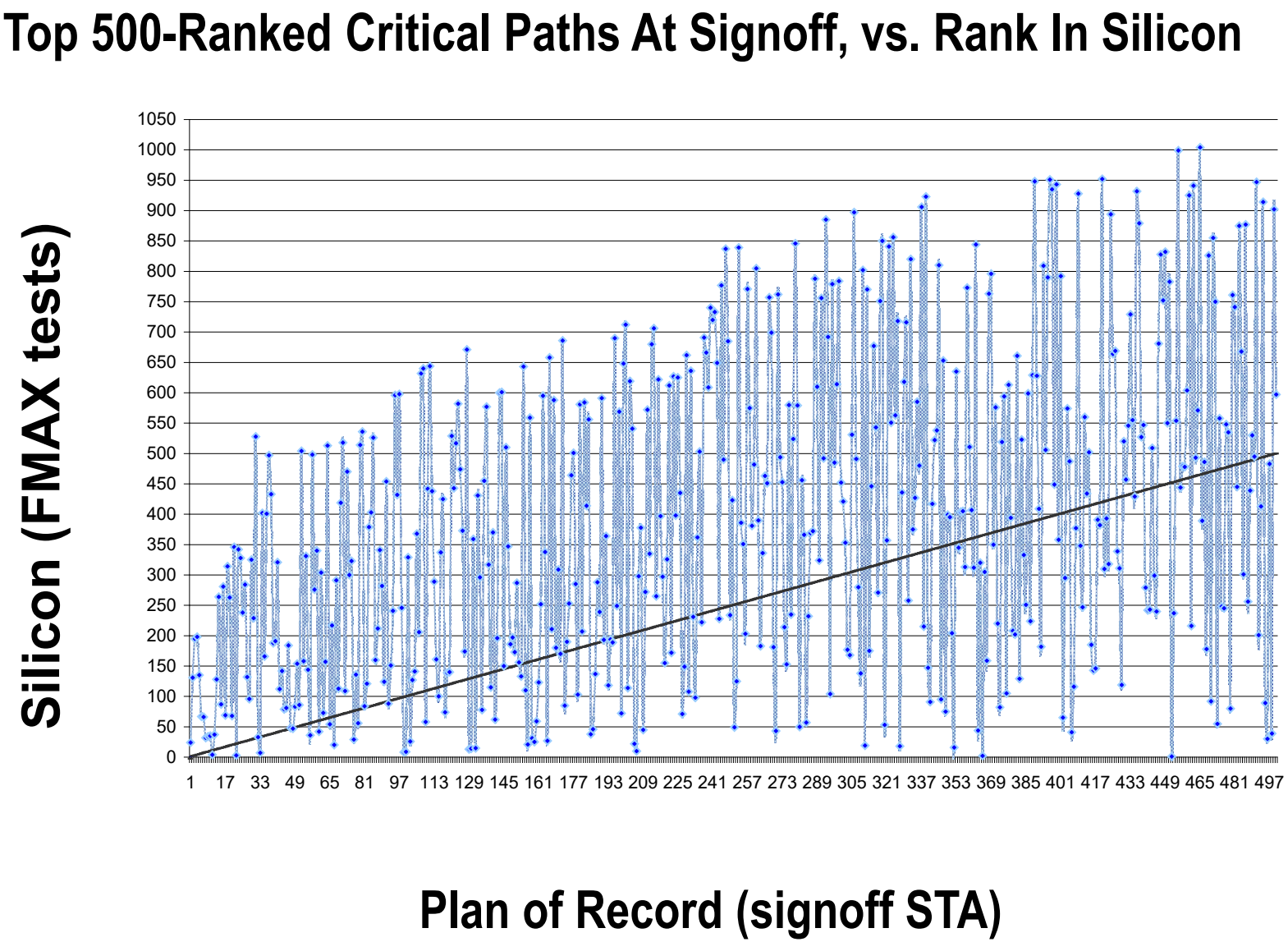
# Inherent Noise: Example Results

| Design | Criticality | Clock (ns) | "S" With original Clock Setup | | | "A" With original Clock Setup | | | "B" With original Clock Setup | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | WNS (SOCE) (ns) | WNS (PT) (ns) | TNS (PT) (ns) | WNS (Astro) (ns) | WNS (PT) (ns) | TNS (PT) (ns) | WNS(BF) (ns) | WNS (PT) (ns) | TNS (PT) (ns) |
| AES | Tight clock (original 2.2ns) | 2.1998 | -0.407 | -0.430 | -81.124 | -0.241 | -0.487 | -94.822 | -0.077 | -0.391 | -60.156 |
| | | 2.1999 | -0.392 | -0.420 | -73.533 | -0.218 | -0.512 | -89.316 | -0.067 | -0.397 | -58.728 |
| | | 2.2000 | -0.399 | -0.457 | -85.641 | -0.255 | -0.569 | -100.956 | -0.081 | -0.331 | -59.985 |
| | | 2.2001 | -0.436 | -0.439 | -82.053 | -0.280 | -0.535 | -110.341 | -0.074 | -0.442 | -61.048 |
| | | 2.2002 | -0.406 | -0.441 | -82.576 | -0.246 | -0.490 | -92.196 | -0.067 | -0.384 | -51.980 |
| | Loose clock (original 3.0ns) | 2.9998 | -0.026 | -0.119 | -1.965 | 0.040 | -0.280 | -35.482 | 0.000 | -0.342 | -44.778 |
| | | 2.9999 | -0.091 | -0.095 | -2.137 | 0.064 | -0.325 | -34.699 | 0.001 | -0.469 | -46.154 |
| | | 3.0000 | -0.046 | -0.096 | -3.499 | 0.049 | -0.346 | -36.565 | -0.001 | -0.448 | -48.369 |
| | | 3.0001 | -0.049 | -0.112 | -1.972 | 0.083 | -0.239 | -23.040 | -0.008 | -0.373 | -44.683 |
| | | 3.0002 | -0.061 | -0.078 | -1.718 | 0.057 | -0.287 | -31.985 | 0.000 | -0.421 | -48.042 |
| JPEG | Tight clock (original 1.3ns) | 1.2998 | -0.294 | -0.315 | -625.434 | -0.265 | -0.352 | -744.637 | -0.228 | -0.324 | -501.295 |
| | | 1.2999 | -0.263 | -0.281 | -566.317 | -0.240 | -0.418 | -701.361 | -0.166 | -0.266 | -410.594 |
| | | 1.3000 | -0.257 | -0.258 | -537.580 | -0.256 | -0.395 | -733.841 | -0.244 | -0.338 | -567.228 |
| | | 1.3001 | -0.249 | -0.303 | -561.013 | -0.239 | -0.321 | -719.196 | -0.202 | -0.304 | -475.253 |
| | | 1.3002 | -0.298 | -0.514 | -757.272 | -0.229 | -0.346 | -731.566 | -0.197 | -0.277 | -471.392 |
| | Loose clock (original 2.0ns) | 1.9998 | -0.005 | -0.011 | -0.011 | 0.101 | -0.140 | -0.520 | 0.000 | -0.216 | -11.407 |
| | | 1.9999 | 0.008 | -0.068 | -0.068 | 0.101 | -0.140 | -0.520 | 0.000 | -0.167 | -12.021 |
| | | 2.0000 | -0.007 | -0.093 | -0.137 | 0.101 | -0.131 | -1.240 | -0.002 | -0.196 | -15.189 |
| | | 2.0001 | -0.001 | -0.010 | -0.010 | 0.096 | -0.098 | -0.449 | 0.001 | -0.181 | -16.782 |
| | | 2.0002 | 0.008 | -0.004 | -0.006 | 0.099 | -0.066 | -0.279 | -0.006 | -0.178 | -12.220 |

Q: Why do chips pass timing signoff in design, but then fail to yield in the fab?

A: Signoff criteria are different from manufacturing criteria.

# Problem: Path Index Migration/Miscorrelation



# No Two Chips are Identical

- Manufacturing variation is the reality.
  - “Corners” (FF, SS, typical) try to approximate them
- Do NOT expect models to be “accurate”
  - Manufacturing process is always a random variable with a distribution
  - Corollary: wasting time on 1ps improvement is useless
  - Corollary: wasting time on 1nm dimension change is useless and often not permitted by design rules

Q: Can I do better than the tool?

A: Very unlikely.

# EDA tools are (usually) well optimized

- Experience
  - You: 0-10 designs
  - EDA tool: 1000s of designs over years/decades
- Correctness
  - You: prone to making mistakes
  - EDA tool: any bugs ironed out over experience
- Quality of results
  - You: may be can do a better job with 100 gate designs
  - EDA tool: Optimized algorithms to deal with millions of gates
- Cost and Effort
  - You: a graduate engineer paid costing a company $200K+/year
  - EDA tool: hardware cost: $10K/year for 32 processor server which can churn million gate SP&R overnight + tool cost ($10K-$500K/year amortized over many designers)
- Weird, strange constraints or objectives
  - This is where you *may* have an edge. Tools are optimized to handle the common case and some not so common cases (through a large number of visible and hidden switches)

Q: Can I get away with no “programming” being a designer?

A: Not really, at least if you want to be an effective digital designer.

# Managing Complex Designs requires Methodologies → Scripting

UCLA

- You may not need to write complex C++ code but scriptware is *very* common
  - Timers, SP&R, most EDA tools: TCL is the defacto standard scripting language.
    - Industry-strength tool flows often have 1000s of lines of TCL scripts
  - Running PV (e.g., Calibre): its own SVRF scripting language
  - Managing design databases (OpenAccess, Milkyway, etc) using TCL, Python, SKILL,….
  - Parsing reports, automating tool flows, managing files: Shell scripts, Perl, Python, TCL…
- Opening tool GUIs is more of an exception than norm
  - Its preferred to launch noGUI scripts and wait for runs to complete
  - May be use the GUI or the tool shell to debug
- Unix/Linux is the near-universal standard (Windows/MAC support is minimal): Learn how to use Linux and Linux shell utilities effectively!
- Jobs are launched often on large server farms → learn how to use compute cluster tools (e.g., LSF)

Q: Can I just ask somebody if I get stuck using a tool?

A: Not always, learn to debug yourself!

# Debugging yourself

- "Big" companies *may* have internal tool support and external application engineering support which *may* be sufficient
  - But no one appreciates "trivial" questions
- "Small" companies, universities get little tool support
  - Universities get near zero
- Debugging yourself
  - Google!
  - Tools have extensive documentation
    - User guides, reference manuals, man/info pages, application notes
  - Message boards on EDA company websites
  - Resist the urge to post on Piazza (or CAD support team) the first instant you see an error. Most tool errors are informative which help you debug.

Q: How do I search for prior art ?

A: Google Scholar

# Literature search 101

- Very few things are truly “new”
- Best (current) way of searching literature: Google Scholar (searches books, papers, patents)
- Think of keywords on the topic and what might be one hop away
  - Remember *“i”* in Math is *“j”* in EE!
  - My TSP is your scan chain ordering!
- Everything in Google Scholar has “cites” and “cited by”. This allows you to systematically trace literature.