Optimization Challenges in Transistor Sizing

Chandu Visweswariah

IBM Thomas J. Watson Research Center

Yorktown Heights, NY

Acknowledgments

- Thank you for inviting me
- The work I am presenting today is that of the extended EinsTuner team at five different sites of IBM
- Special thanks to Phillip Restle for the animations you are about to see

Preliminary remarks

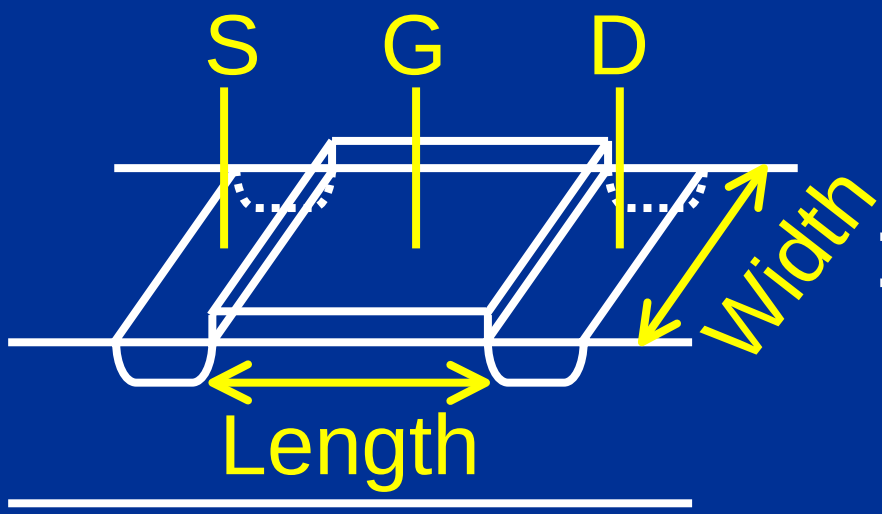
- I am not a mathematician!
- There is nothing multi- about this talk!
- All chip design problems, in some sense, are
  - optimization problems
  - huge
  - mixed discrete/continuous
  - riddled with redundancies and degeneracies
  - semi-infinite
  - statistical
  - multi-disciplinary, multi-objective, multi-…
  - expensive function and gradient evaluations
  - numerically noisy...

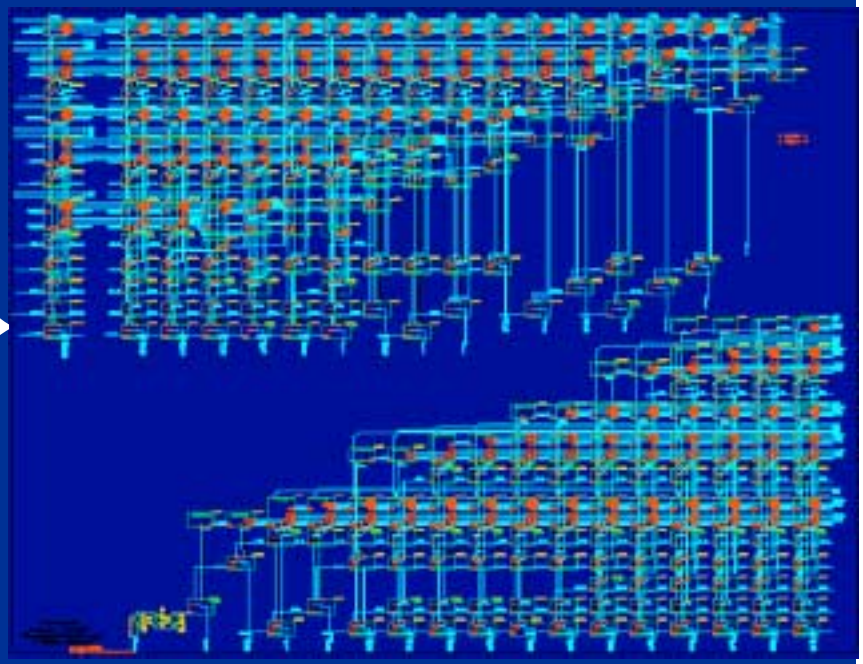
Two acts focusing on optimization

- Act I: what we have done with existing nonlinear optimization techniques
  - what is EinsTuner?
  - how does it work?
  - what is its impact?
- Act II: what we would love to do and currently cannot
  - numerical noise
  - capacity
  - robustness, scaling, weighting, degeneracy
  - recursive minimax support
  - mixed integer continuous problems

Act I: What is EinsTuner?

- A CAD tool that determines the optimal size of each transistor in a digital design





- Wider transistors switch faster; wider transistors load the previous stage more
- The timing of millions of paths interact

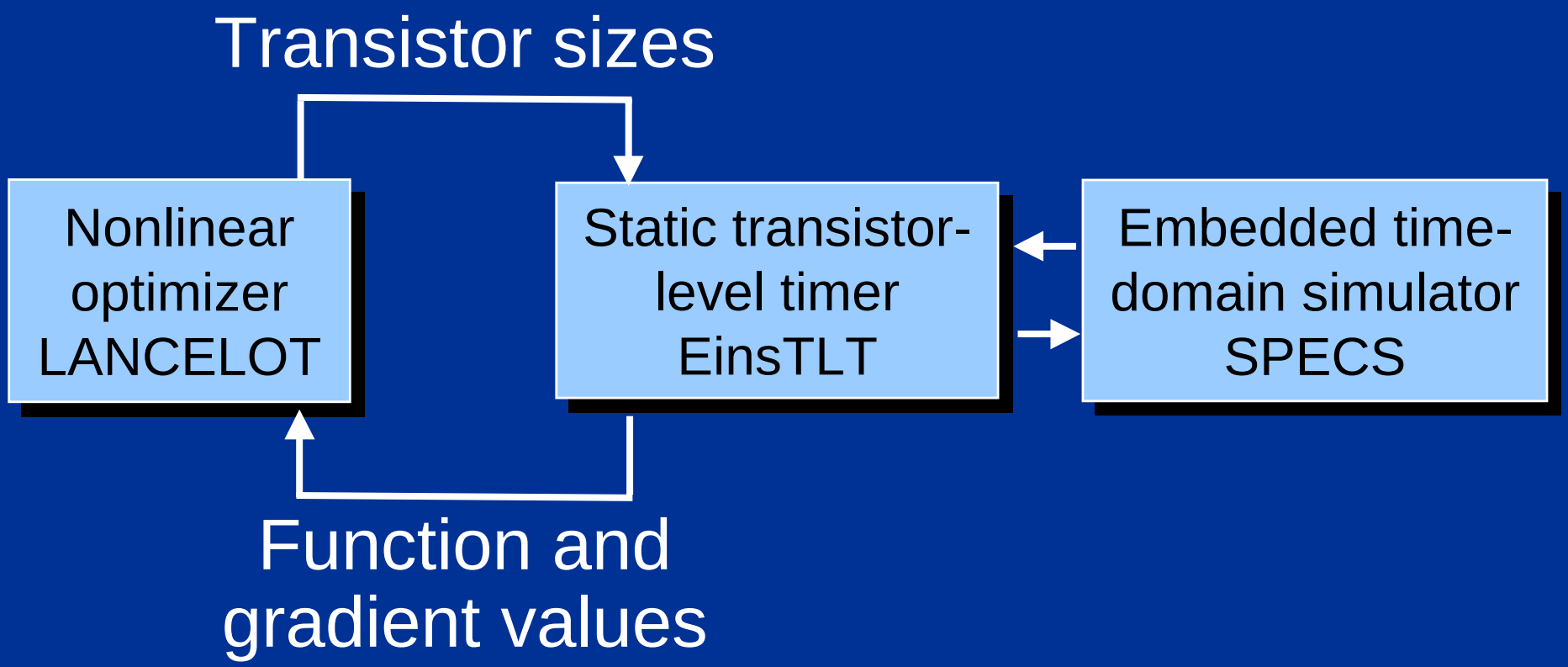
Static?

- A static, formal, transistor sizing tool geared towards custom high-performance digital circuits
  - static: based on static transistor-level timing analysis, so implicitly takes into account all paths through the logic
  - custom: timing based on real time-domain simulation of all possible transitions through each clump of strongly-connected transistors
  - formal: guaranteed optimal
    - “local” according to engineers
    - “global” according to mathematicians
  - transistor-level, therefore inherently flat
  - see DAC ’99, ICCAD ’99, SIAM J. Opt 9/99

Features of EinsTuner

- Extensive (sweeping and fine-grained) grouping, ratio-ing and no-touch commands
- 50K transistor capacity (≈140K variables)
- Allows delay, area, β ratio, input loading, slew and transistor width constraints
- Allows minimization of delay, area or linear combination thereof
- Easy-to-use; full GUI support; good fit into (semi-custom and custom) methodologies
- Timer benefits such as accuracy for custom circuits, pattern-matching, state analysis

EinsTuner: formal static optimizer



Components of EinsTuner

1 Read netlist; create timing graph (EinsTLT)

Formulate 3 pruned optimization problem

Feed problem to nonlinear optimizer (LANCELOT) 2

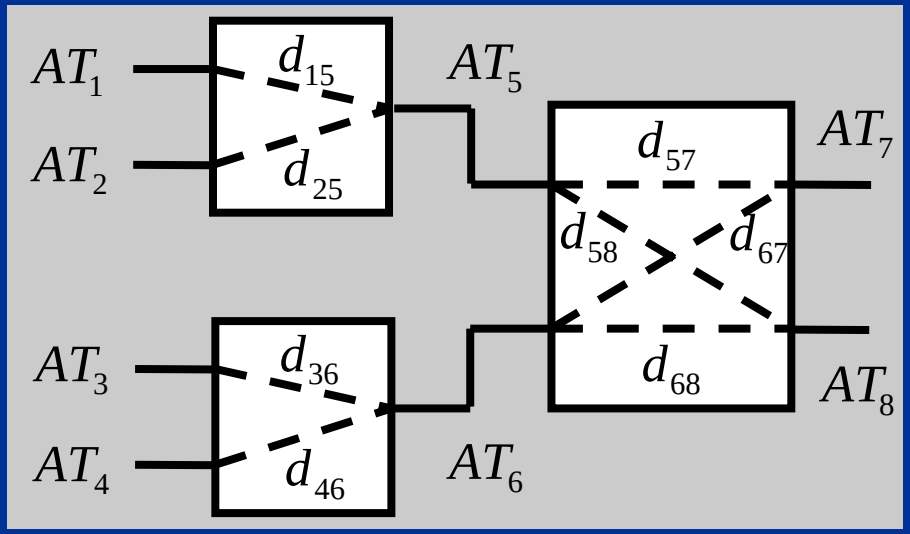
Solve optimization problem, call simulator for delays/slews and gradients thereof

Fast simulation and incremental sensitivity computation (SPECS)

Obtain converged solution

Snap-to-grid; back-annotate; re-time

Static optimization formulation



$$\min\ [\max\ (AT_7, AT_8)]$$

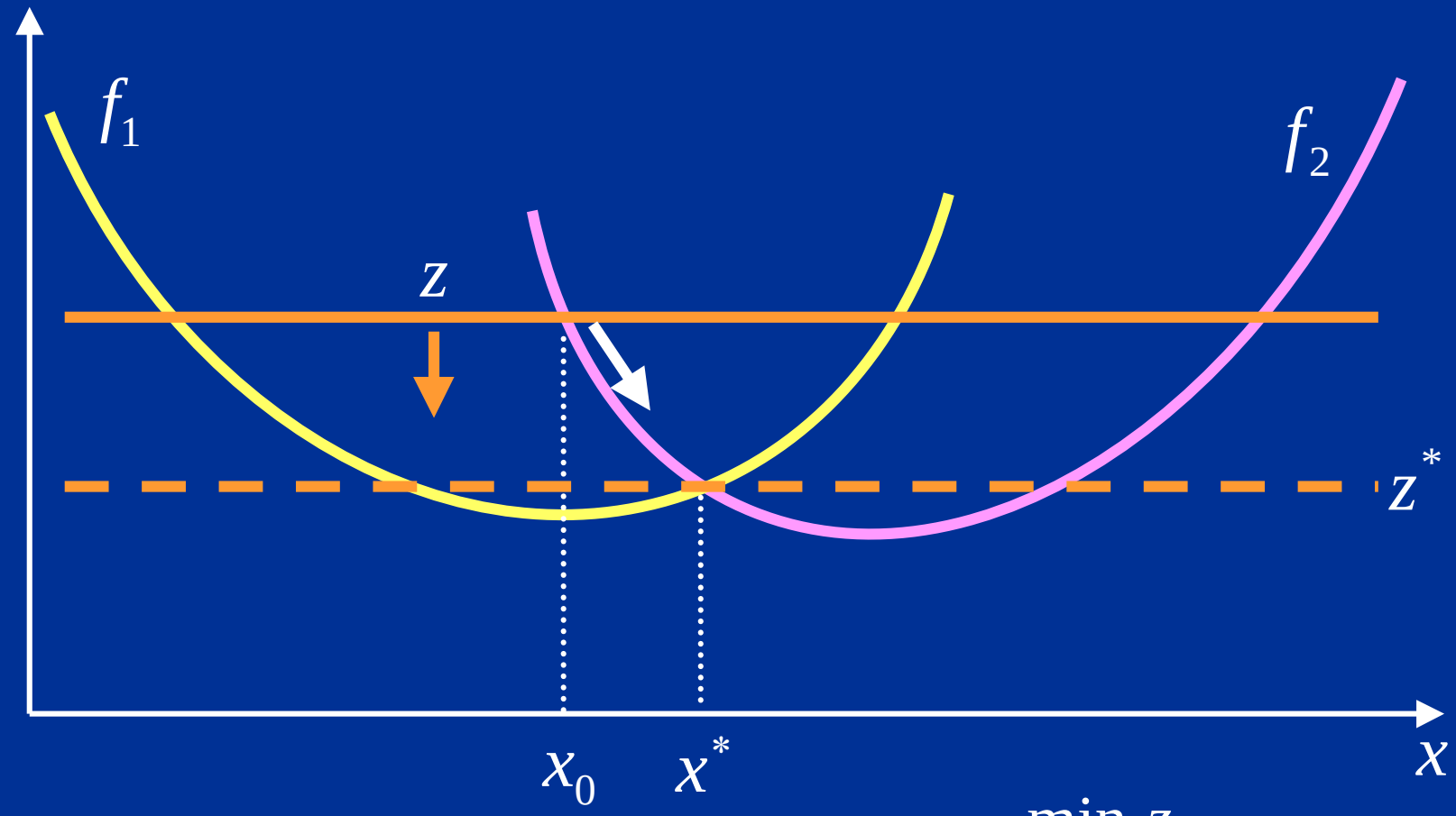
$$\text{s.t. } AT_7 = \max\ [AT_5 + d_{57},\ AT_6 + d_{67}]$$

$$\text{s.t. } AT_8 = \max\ [AT_5 + d_{58},\ AT_6 + d_{68}]$$

$$\text{s.t. } AT_5 = \max\ [AT_1 + d_{15},\ AT_2 + d_{25}]$$

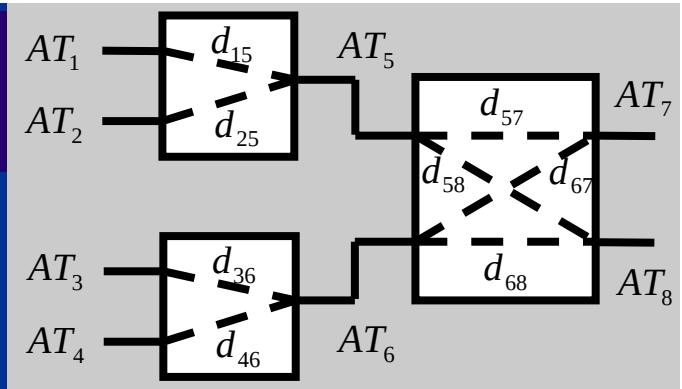
$$\text{s.t. } AT_6 = \max\ [AT_3 + d_{36},\ AT_4 + d_{46}]$$

Digression: minimax optimization



$$\min_x [\max\{f_1(x), f_2(x)\}] \quad \Rightarrow \quad \min_{x,z} z \quad \text{s.t. } z \geq f_1(x), \quad \text{s.t. } z \geq f_2(x)$$

Remapped problem



$$\min[\max(AT_7, AT_8)] \Rightarrow \begin{cases} \min z \\ \text{s.t. } z \geq AT_7 \\ \text{s.t. } z \geq AT_8 \end{cases}$$

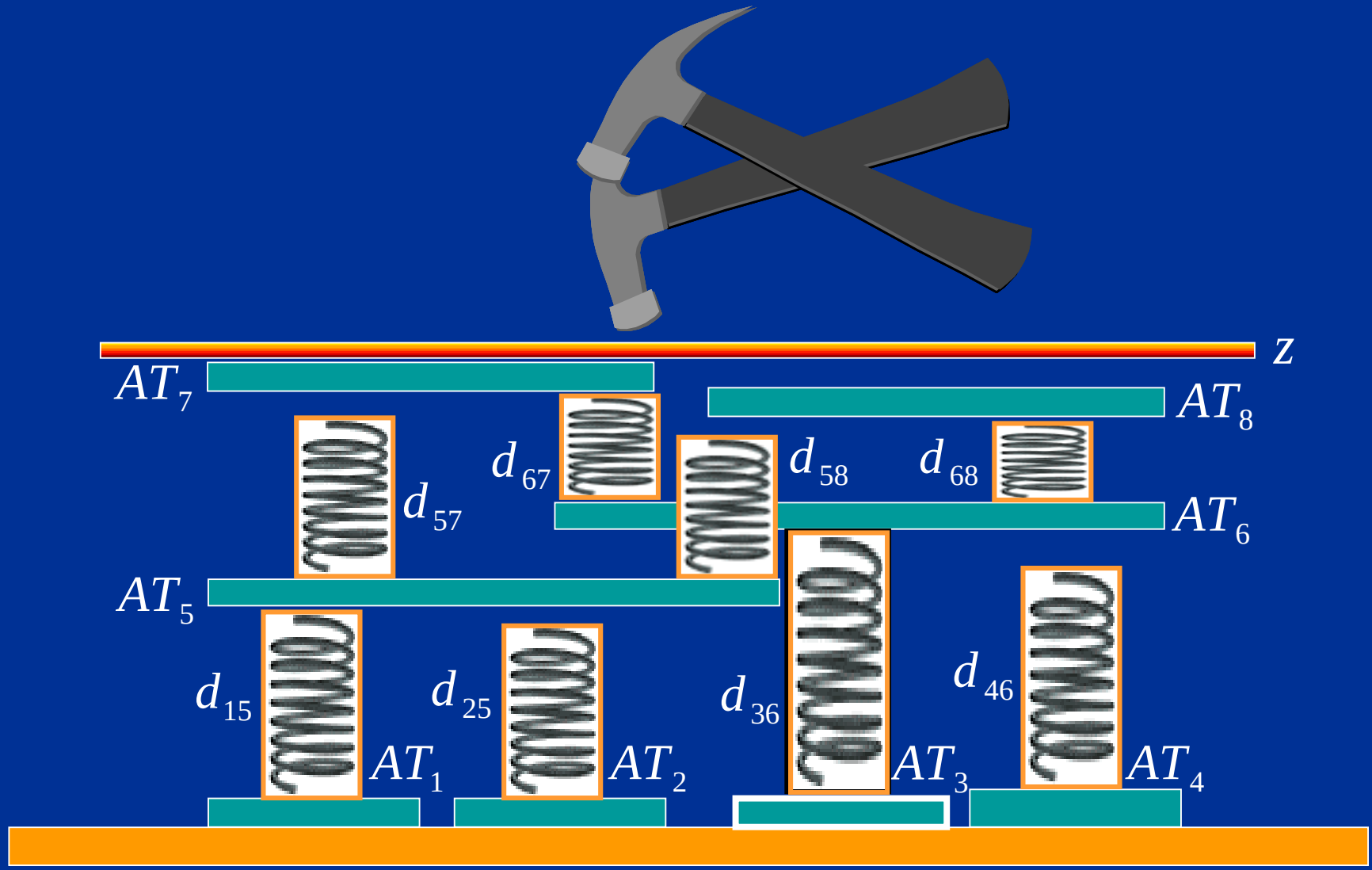
$$AT_7 = \max[AT_5 + d_{57}, AT_6 + d_{67}] \Rightarrow \begin{cases} AT_7 \geq AT_5 + d_{57} \\ AT_7 \geq AT_6 + d_{67} \end{cases}$$

$$AT_8 = \max[AT_5 + d_{58}, AT_6 + d_{68}] \Rightarrow \begin{cases} AT_8 \geq AT_5 + d_{58} \\ AT_8 \geq AT_6 + d_{68} \end{cases}$$

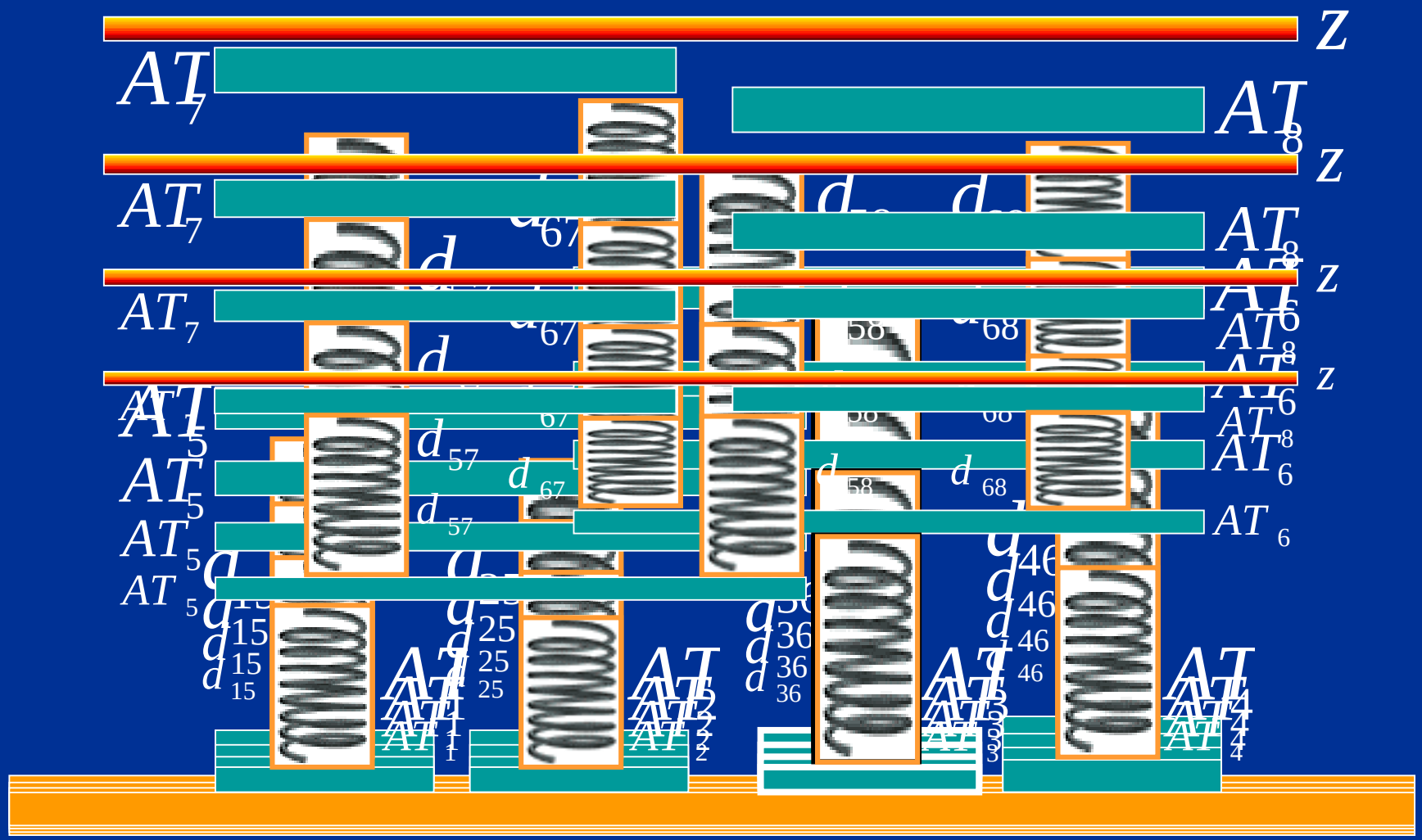
$$AT_5 = \max[AT_1 + d_{15}, AT_2 + d_{25}] \Rightarrow \begin{cases} AT_5 \geq AT_1 + d_{15} \\ AT_5 \geq AT_2 + d_{25} \end{cases}$$

$$AT_6 = \max[AT_3 + d_{36}, AT_4 + d_{46}] \Rightarrow \begin{cases} AT_6 \geq AT_3 + d_{36} \\ AT_6 \geq AT_4 + d_{46} \end{cases}$$

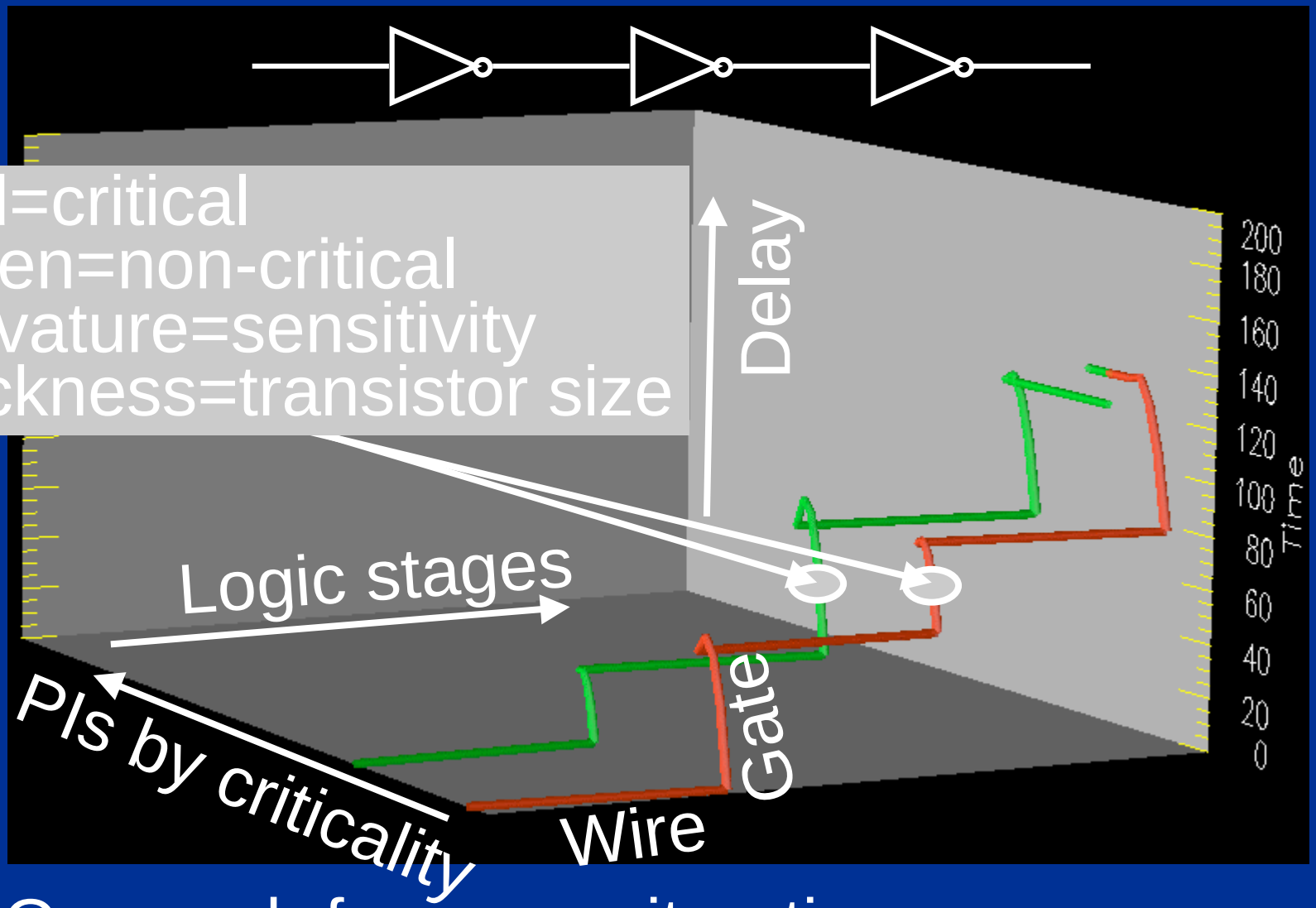
Springs and planks analogy



Springs and planks analogy

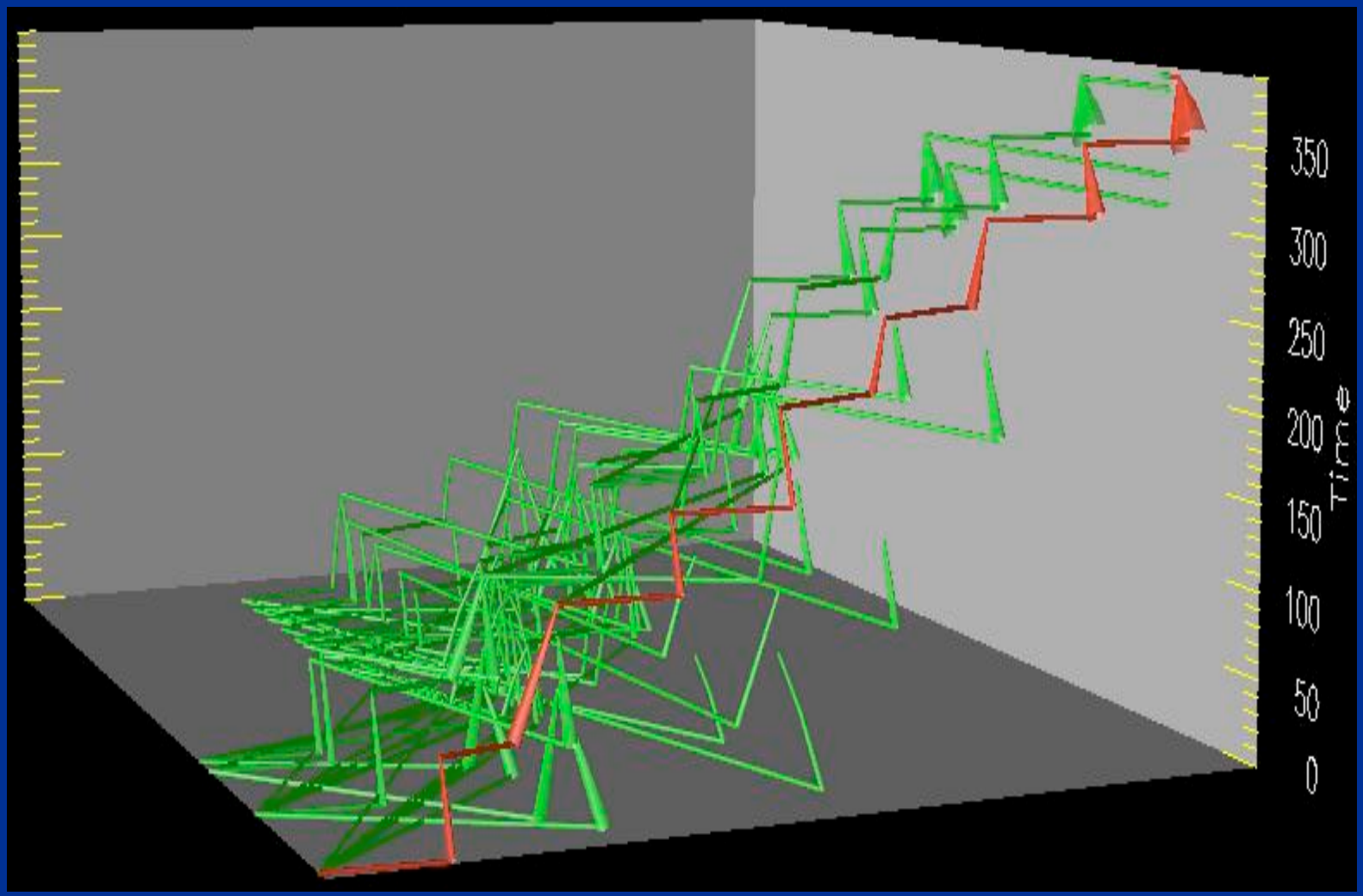


Algorithm animation: inv3

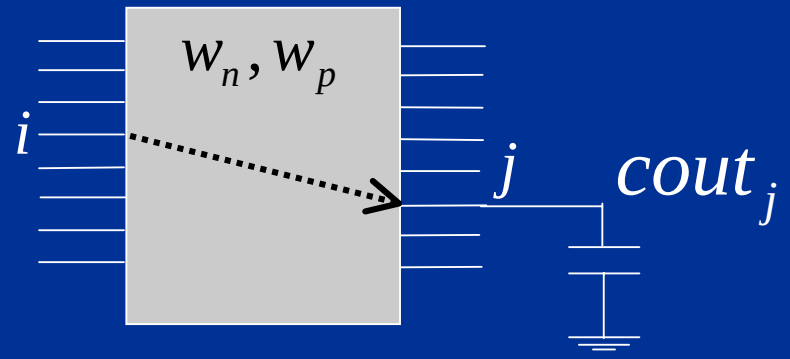


- One such frame per iteration

Algorithm demonstration: 1b ALU



Constraint generation



$$AT_j^r \geq AT_i^f + d_{ij}^r(w_n, w_p, cout_j, slew_{in}^f)$$

$$AT_j^f \geq AT_i^r + d_{ij}^f(w_n, w_p, cout_j, slew_{in}^r)$$

$$slew_j^r \geq s_{ij}^r(w_n, w_p, cout_j, slew_{in}^f)$$

$$slew_j^f \geq s_{ij}^f(w_n, w_p, cout_j, slew_{in}^r)$$

Statement of the problem

$$\min \; z$$

$$\text{s.t.} \quad z \geq AT_i + RAT_i \qquad \text{for all POs}$$

$$\text{s.t.} \quad AT_j \geq AT_i + d_{ij}(W,S) \qquad \text{for each timing arc}$$

$$\text{s.t.} \quad s_j \geq s_{ij}(W,S) \qquad \text{for each timing arc}$$

$$\text{s.t.} \quad \Sigma W \leq \text{area target}$$

$$\text{s.t.} \quad \text{pincap}_i(W) \leq \text{target}_i \qquad \text{for all PIs}$$

$$\text{s.t.} \quad \beta_{\min} \leq \beta_i(W) \leq \beta_{\max} \qquad \text{for all gates}$$

$$\text{s.t.} \quad W_{\min} \leq W_i \leq W_{\max} \qquad \text{for all FETs}$$

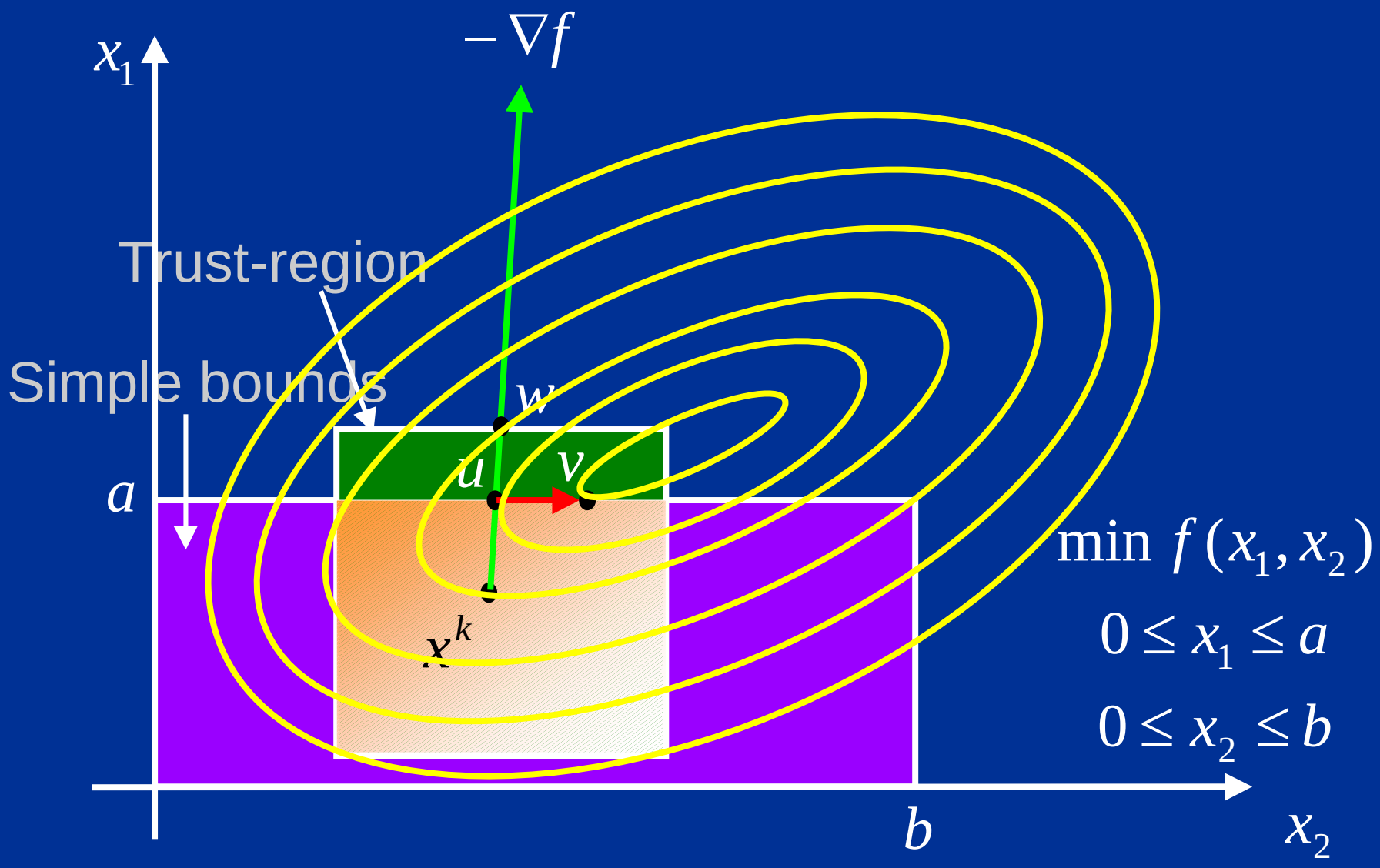
$$\text{s.t.} \quad S_i \leq S_{\max}^{\text{internal}} \qquad \text{for all internal nets}$$

$$\text{s.t.} \quad S_i \leq S_{\max}^{\text{PO}} \qquad \text{for all POs}$$

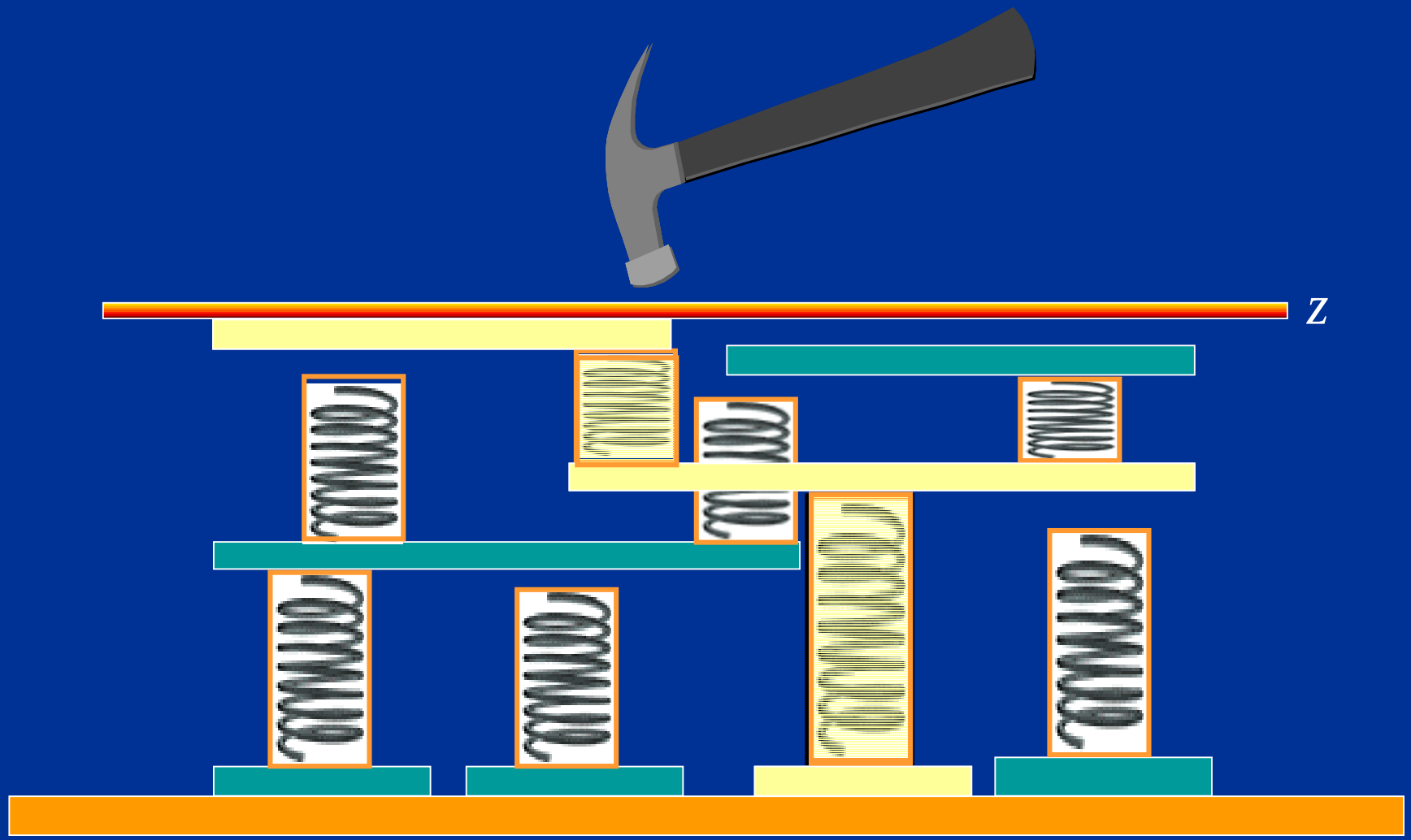
LANCELOT

- State-of-the-art general-purpose nonlinear optimizer
- Trust-region method
- Exploitation of group partial separability; augmented Lagrangian merit function
- Handles general linear/nonlinear equalities/ inequalities and objective functions
- Simple bounds handled by projections
- Several choices of preconditioners, exact/inexact BQP solution, etc.

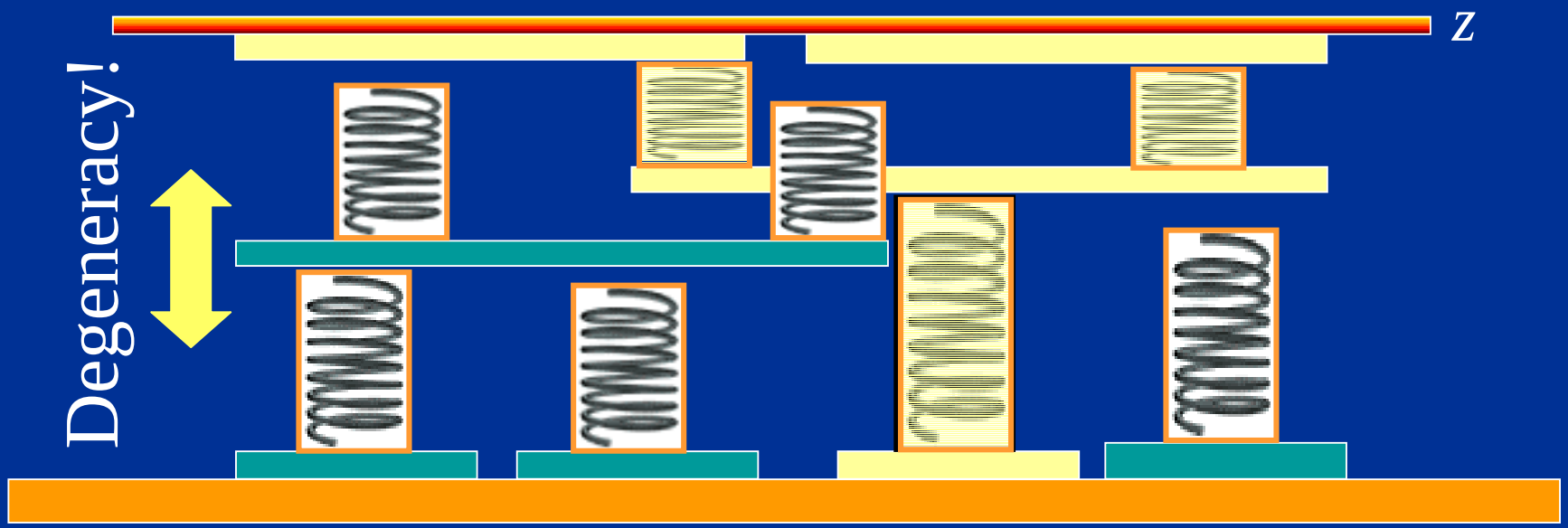
LANCELOT algorithms



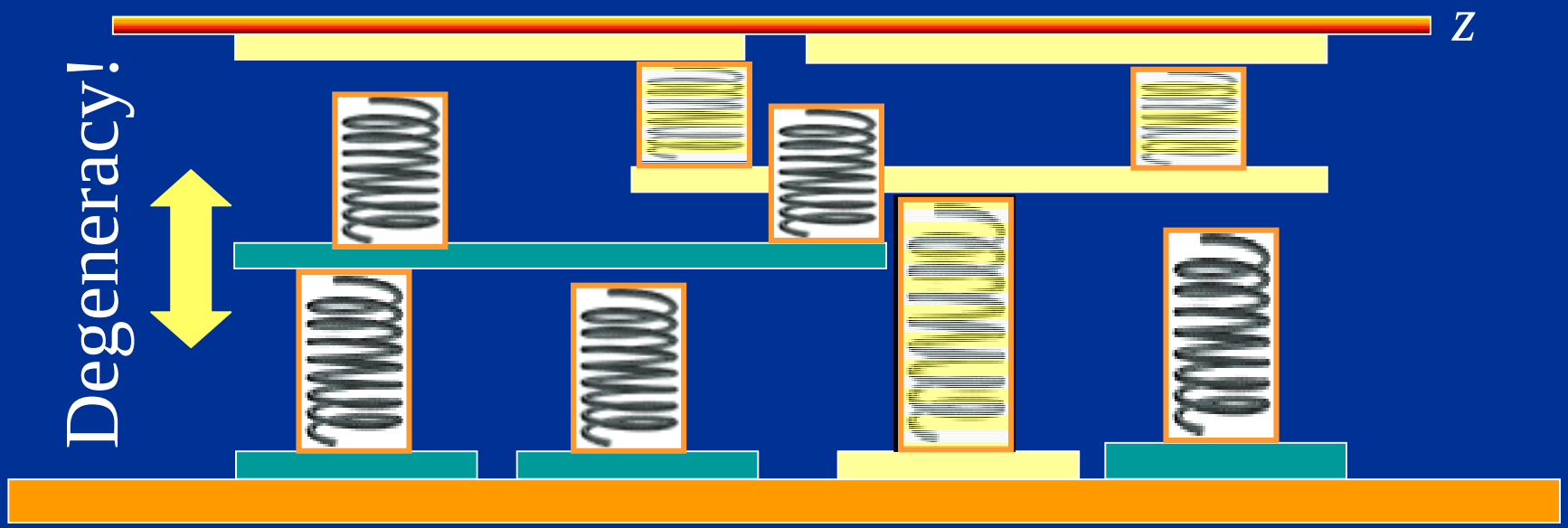
Demonstration of degeneracy



Demonstration of degeneracy

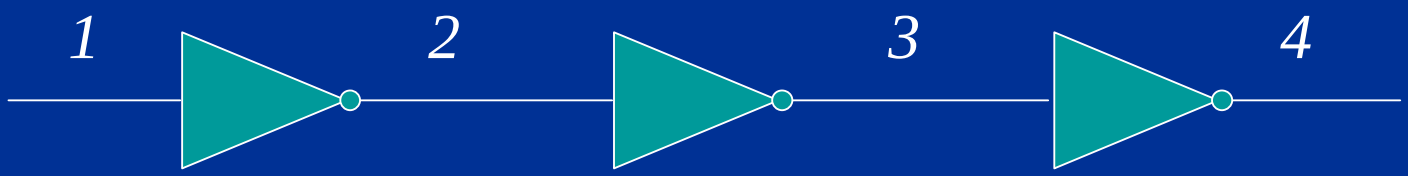


Demonstration of degeneracy



Why do we need pruning?

Pruning: an example

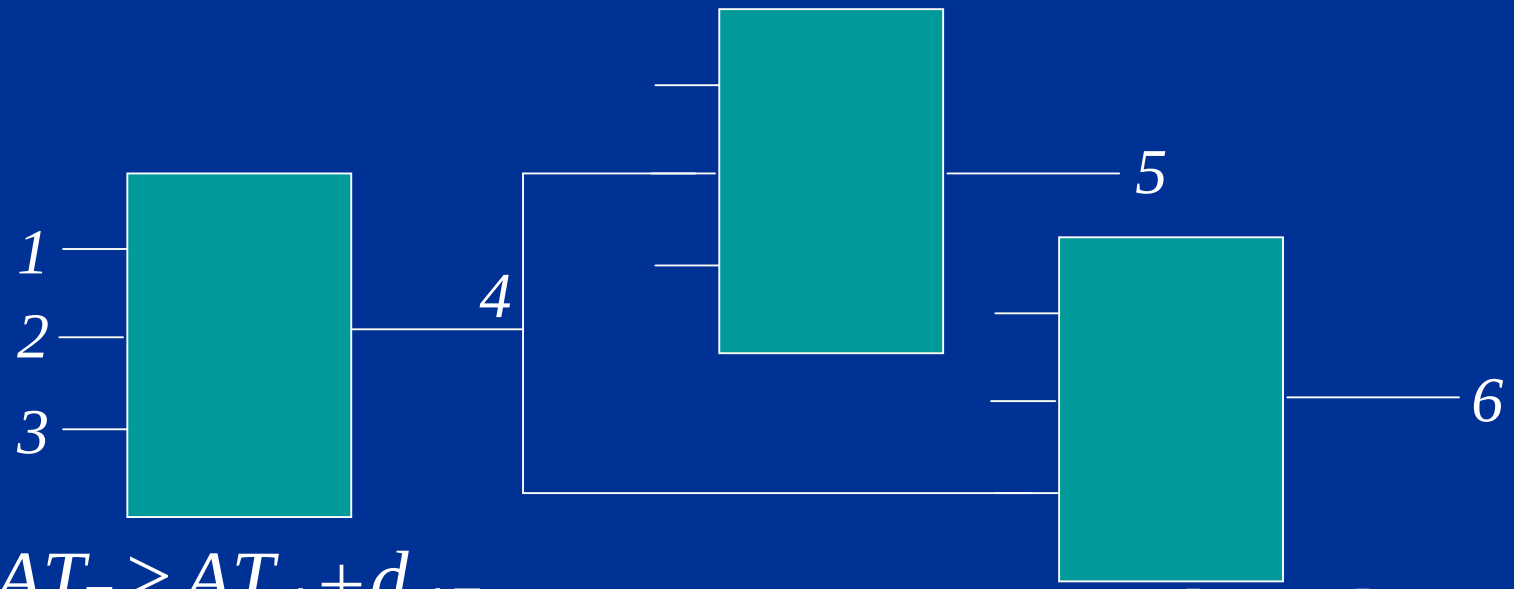


$$\min z$$
$$\text{s.t. } z \geq AT_4^r$$
$$\text{s.t. } z \geq AT_4^f$$
$$\text{s.t. } AT_4^r \geq AT_3^f + d_{34}^r$$
$$\text{s.t. } AT_4^f \geq AT_3^r + d_{34}^f$$
$$\text{s.t. } AT_3^r \geq AT_2^f + d_{23}^r$$
$$\text{s.t. } AT_3^f \geq AT_2^r + d_{23}^f$$
$$\text{s.t. } AT_2^r \geq AT_1^f + d_{12}^r$$
$$\text{s.t. } AT_2^f \geq AT_1^r + d_{12}^f$$

$$\Rightarrow$$

$$\min z$$
$$\text{s.t. } z \geq AT_1^r + d_{12}^f + d_{23}^r + d_{34}^f$$
$$\text{s.t. } z \geq AT_1^f + d_{12}^r + d_{23}^f + d_{34}^r$$

Block-based vs. path-based timing



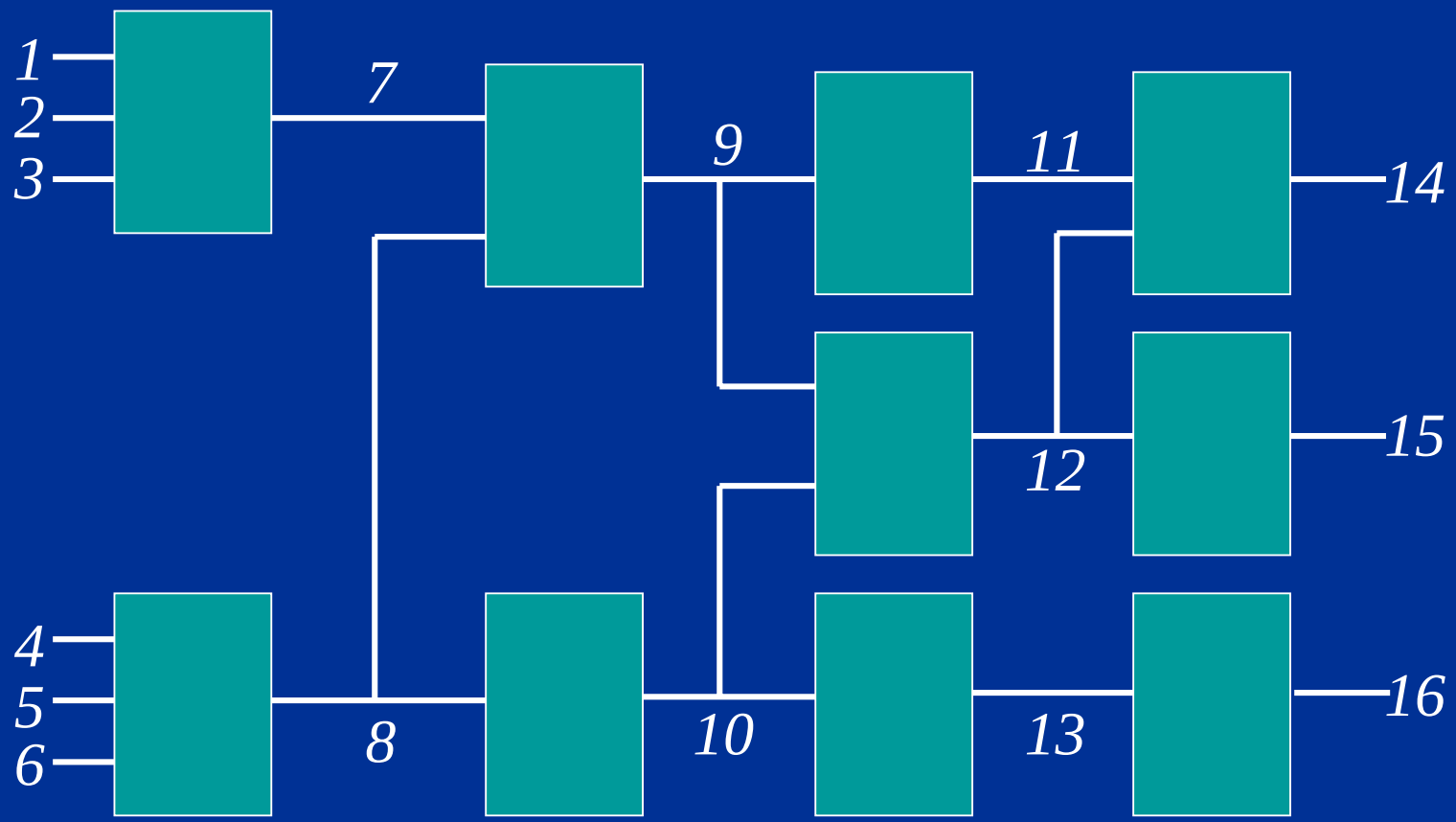
Block-based

$$AT_5 \geq AT_4 + d_{45}$$
$$AT_6 \geq AT_4 + d_{46}$$
$$AT_4 \geq AT_1 + d_{14}$$
$$AT_4 \geq AT_2 + d_{24}$$
$$AT_4 \geq AT_3 + d_{34}$$

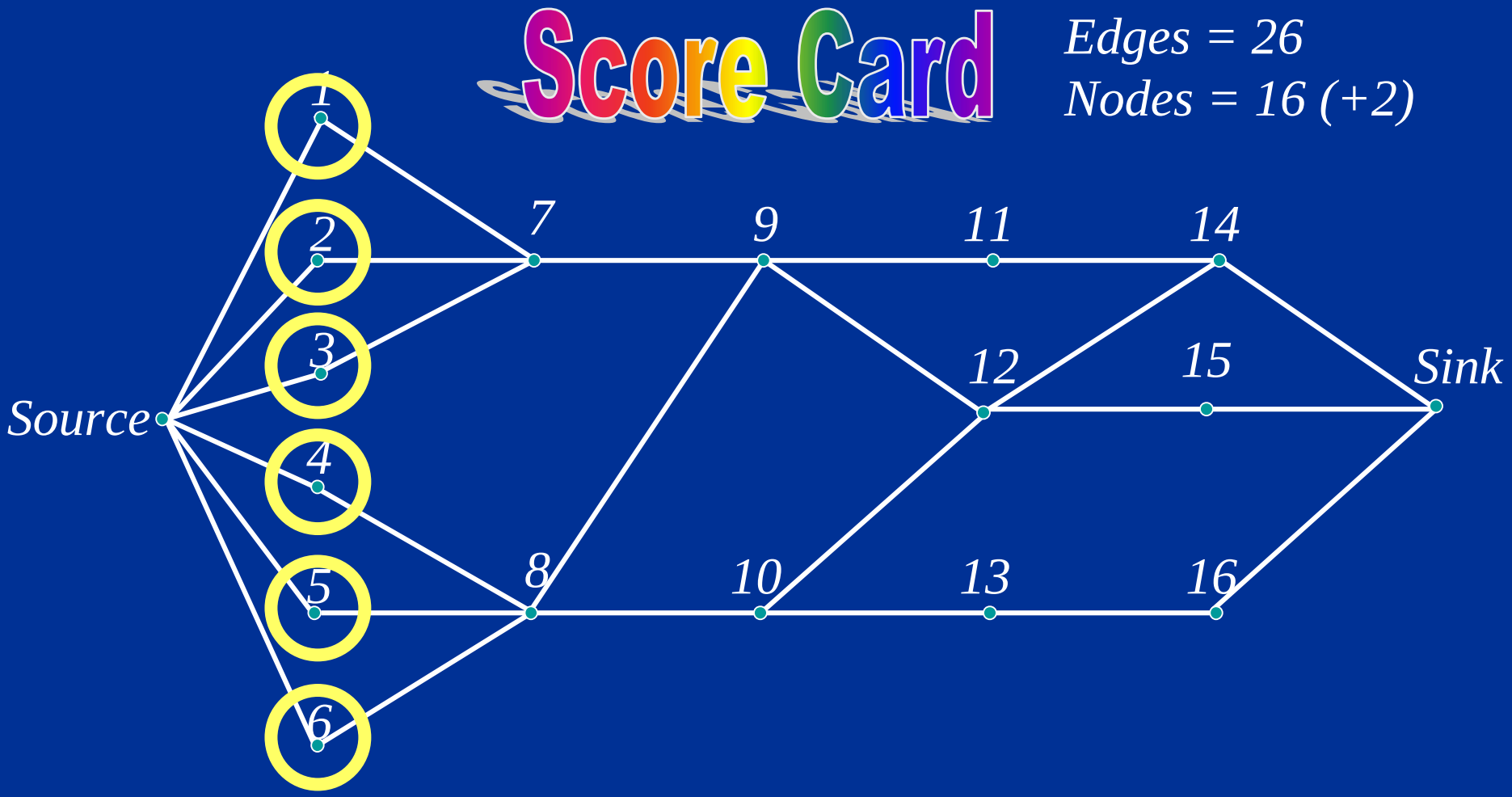
$$AT_5 \geq AT_1 + d_{14} + d_{45}$$
$$AT_5 \geq AT_2 + d_{24} + d_{45}$$
$$AT_5 \geq AT_3 + d_{34} + d_{45}$$
$$AT_6 \geq AT_1 + d_{14} + d_{46}$$
$$AT_6 \geq AT_2 + d_{24} + d_{46}$$
$$AT_6 \geq AT_3 + d_{34} + d_{46}$$

Path-based

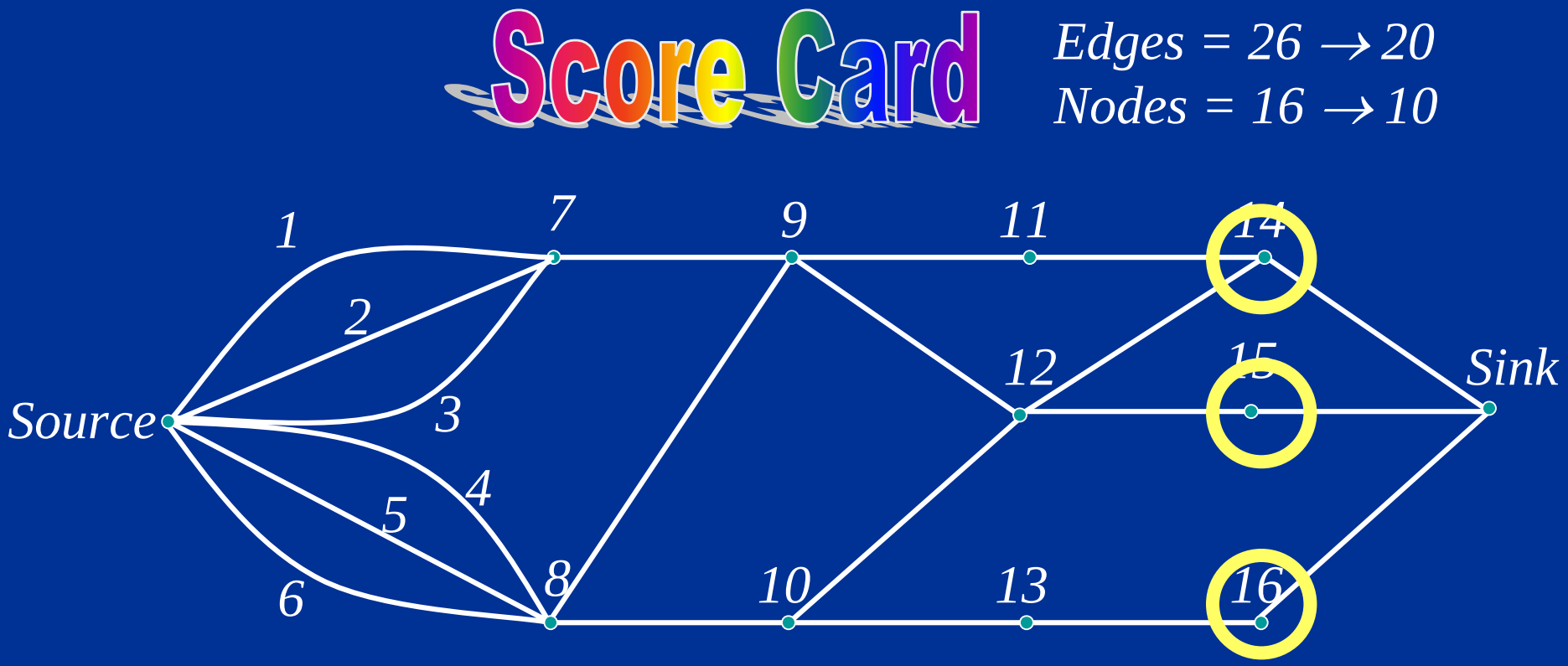
Detailed pruning example



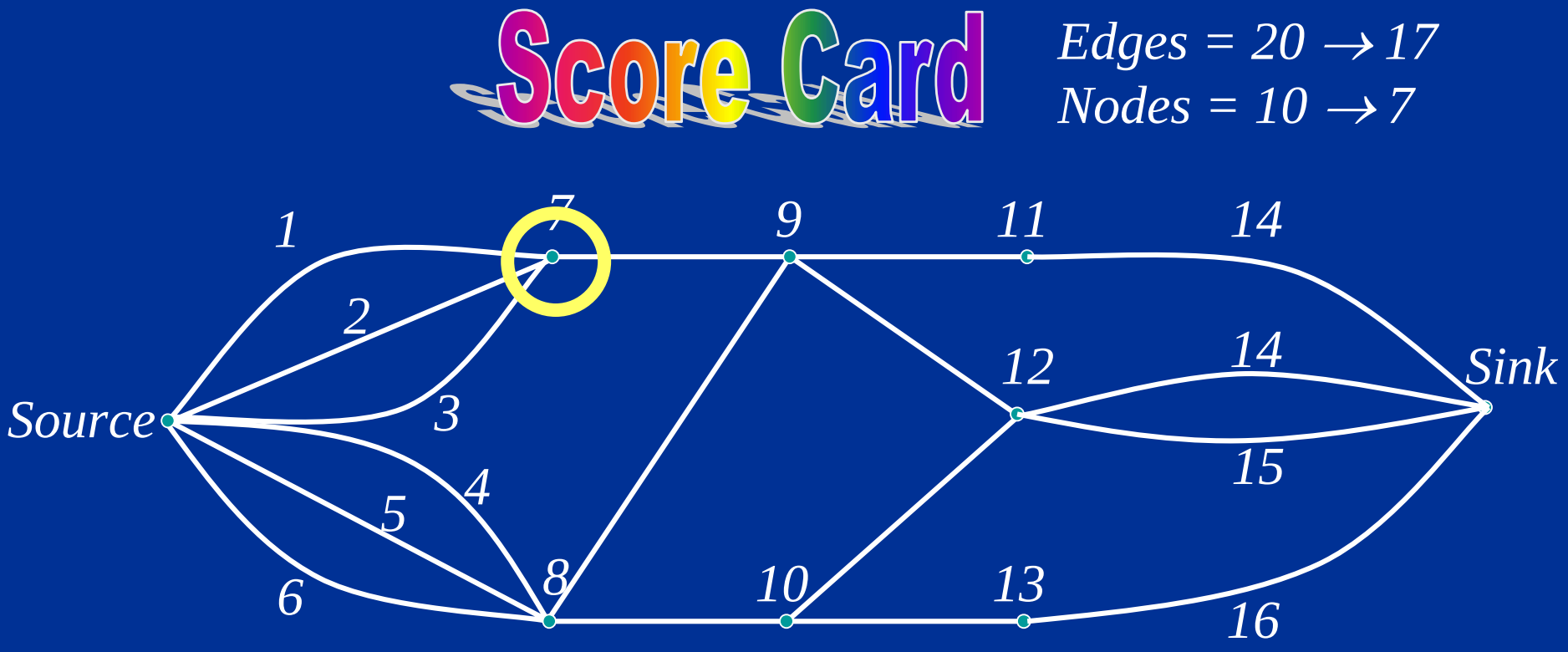
Detailed pruning example



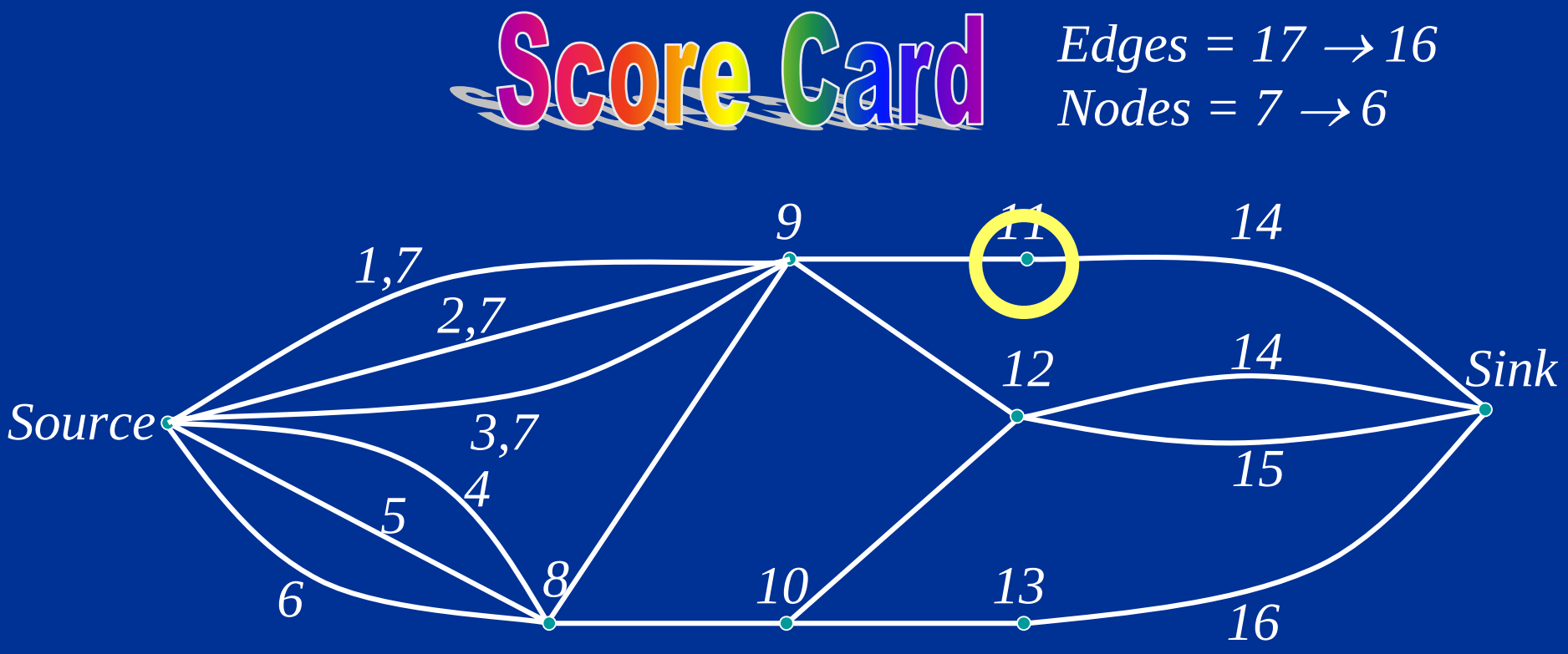
Detailed pruning example



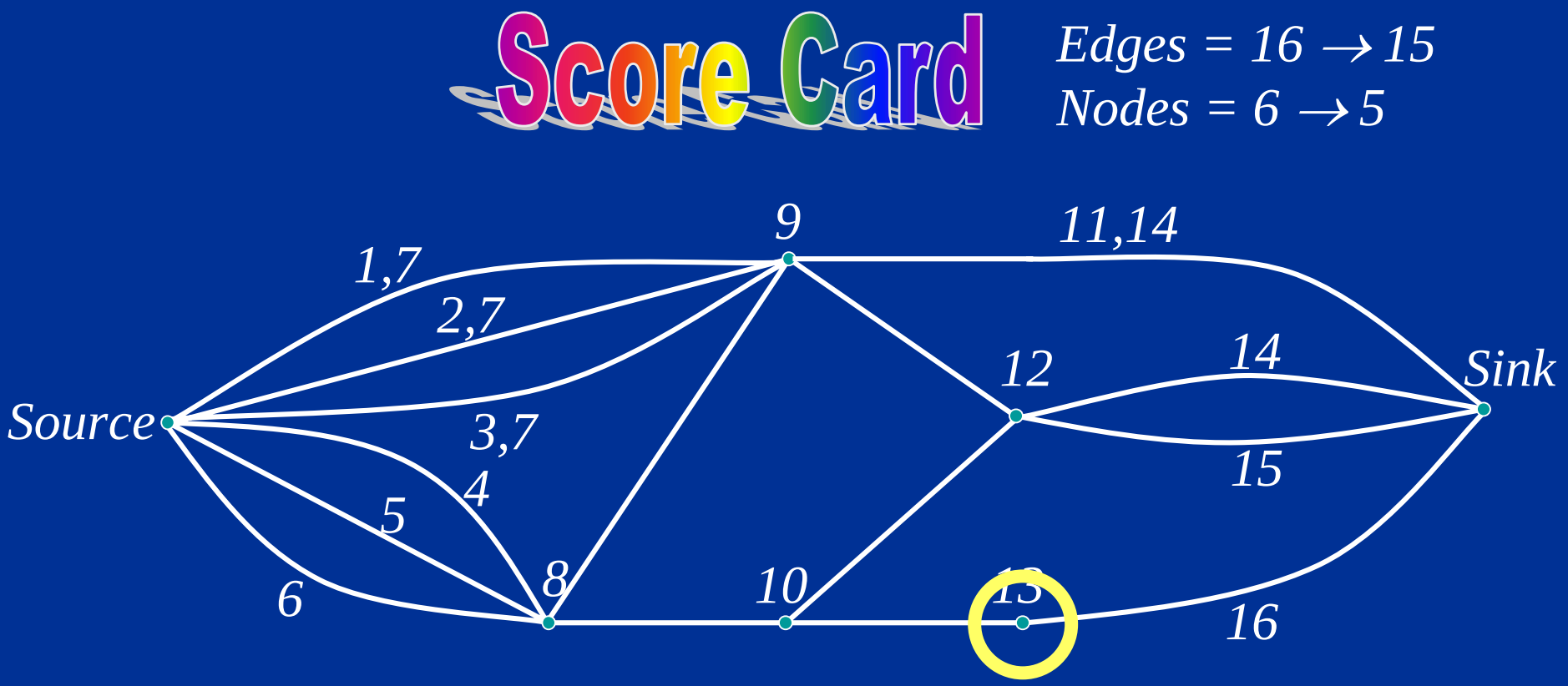
Detailed pruning example



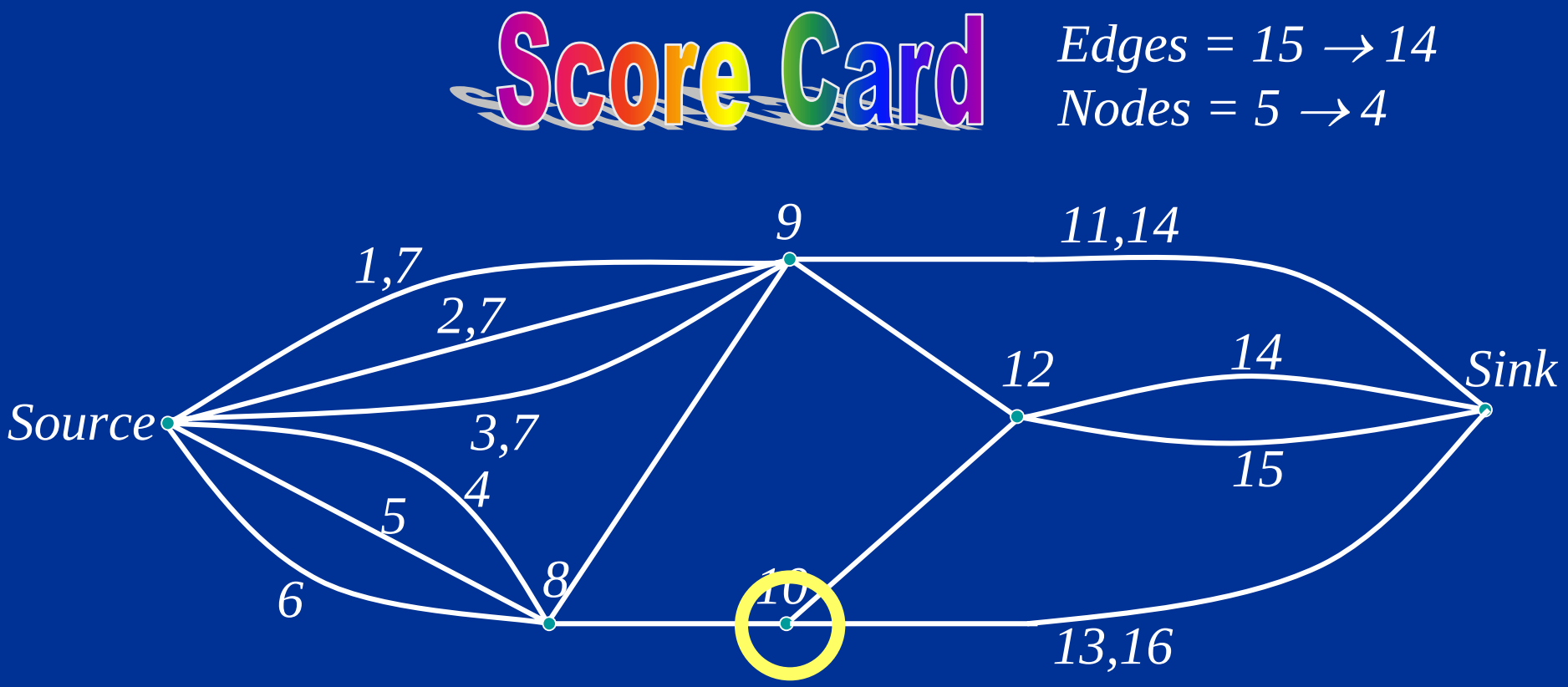
Detailed pruning example



Detailed pruning example



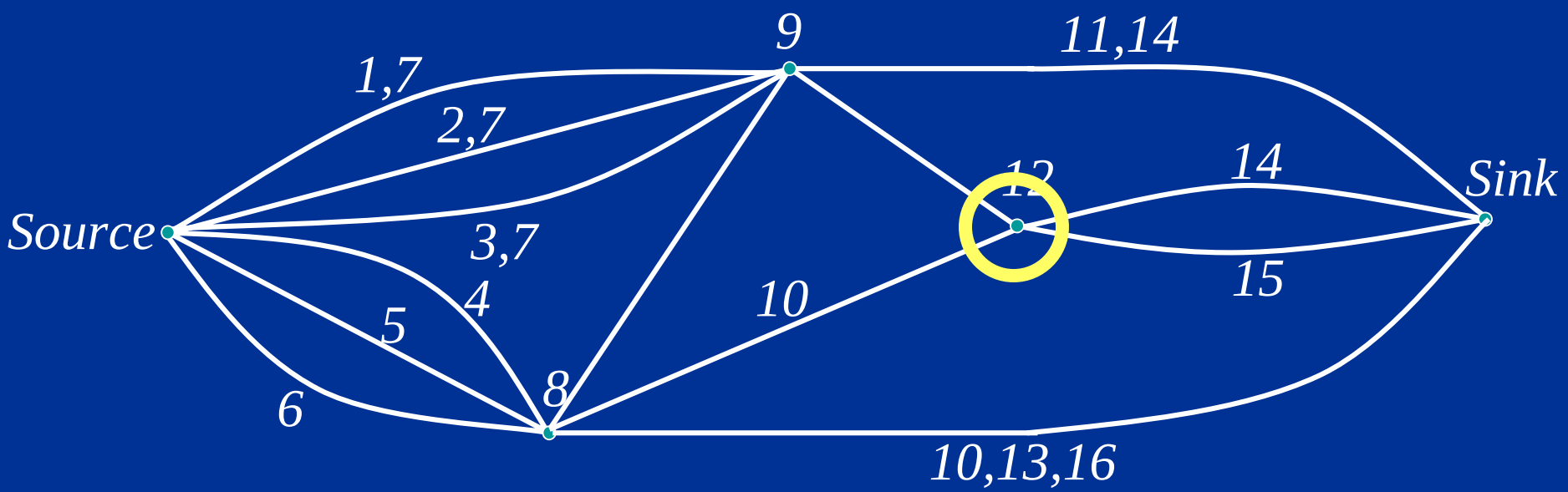
Detailed pruning example



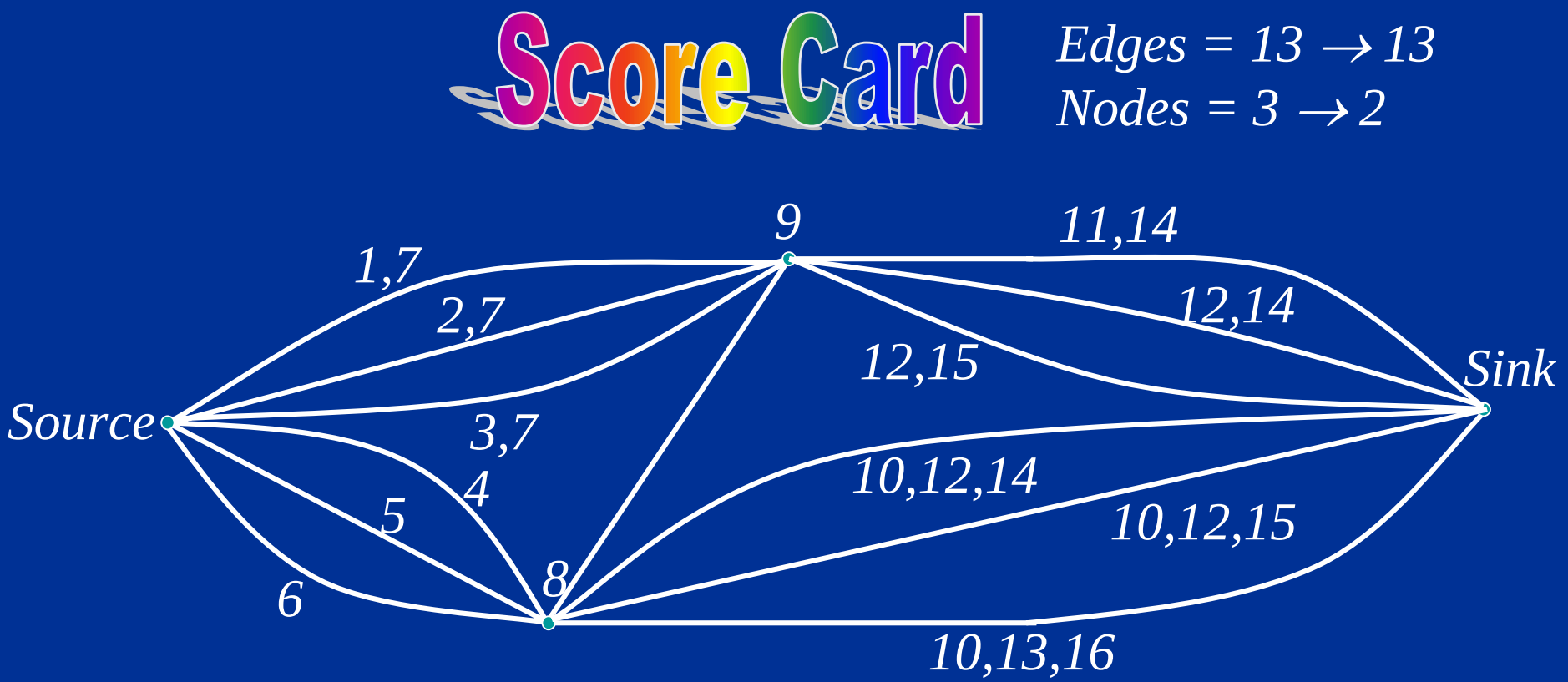
Detailed pruning example

Score Card

Edges = 14 → 13
Nodes = 4 → 3



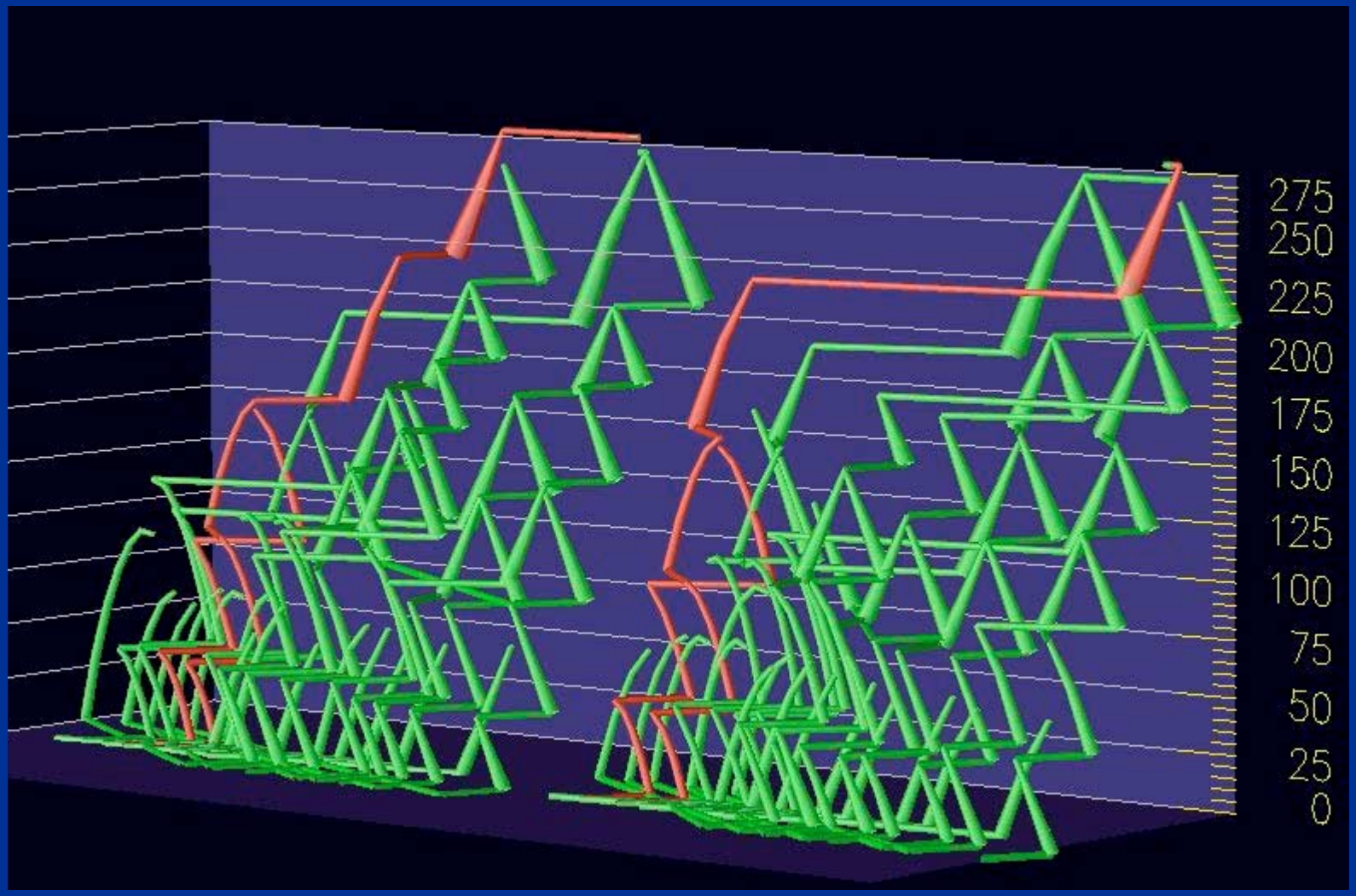
Detailed pruning example



Edges: 26 to 13 (2x)

Nodes: 16 to 2 (8x)

Pruning vs. no pruning



No time to mention...

- Simultaneous switching
- Tuning of transmission-gate MUXes
- Adjoint sensitivity computation
- The role of pattern matching
- Strength calculations/symmetrized stacks
- State analysis
- Benefits of animation (www.opendx.org, www.research.ibm.com/dx)
- Latches
- Uncertainty-aware tuning

EinsTuner impact

- Better circuits
  - 15%+ on previously hand-tuned circuits!
- Improved productivity in conjunction with semi-custom design flow (see Northrop et al, DAC ’01)
- Better understanding of tradeoffs
  - lifts designers’ thinking to a higher level
- Currently being applied to third generation of z-series (S/390) and PowerPC microprocessors to benefit from EinsTuner
- Recently applied to synthesized logic, too

Application to synthesized logic

- Synthesis tools good at combinatorial and discrete optimizations
- Tuners good at continuous optimization with a “global view”
- We would like the best of both worlds
- Solution: flip-flopping between the two optimizations with a “best guess” ordering of “transforms” based on engineering intuition!

Act II: If only...

Rich source of research problems

What we are able to do

What we could do with a robust, fast, high-capacity, noise-immune, degeneracy-immune, easy-to-use, non-finicky nonlinear optimizer

NAND delay variation in cmos9

- PFET width changed 1.0 micron at a time

NAND delay variation in cmos9

- PFET width changed 0.1 micron at a time

NAND delay variation in cmos9

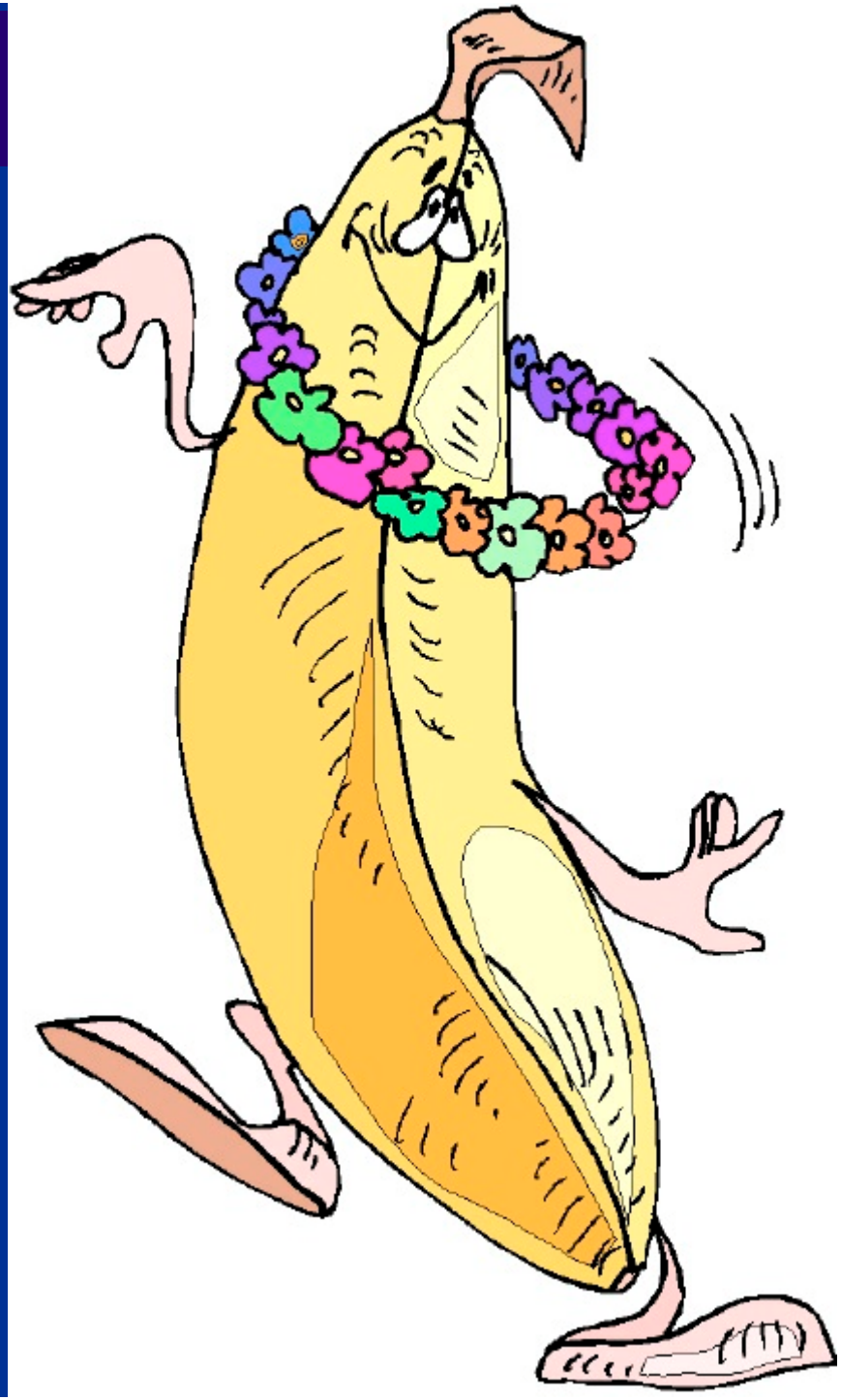
- PFET width changed 0.01 micron at a time

Numerical noise

- Need a formal basis for optimization in the presence of inevitable numerical noise
- Formal methods for measuring noise level
- Tie all tolerances to a multiple of noise level
  - tolerances for bounds
  - tolerances for updating multipliers, etc.
  - stopping criteria
- Currently our optimization is a race against noise and the conclusion is not pleasing!
- All optimization decisions noise-based

Numerical noise

- Inject various types of noise (correlated and uncorrelated) with different amplitude into function and gradient evaluation of analytic optimization problems
- Easy experimentation thus possible to test theories of noise-immune optimization



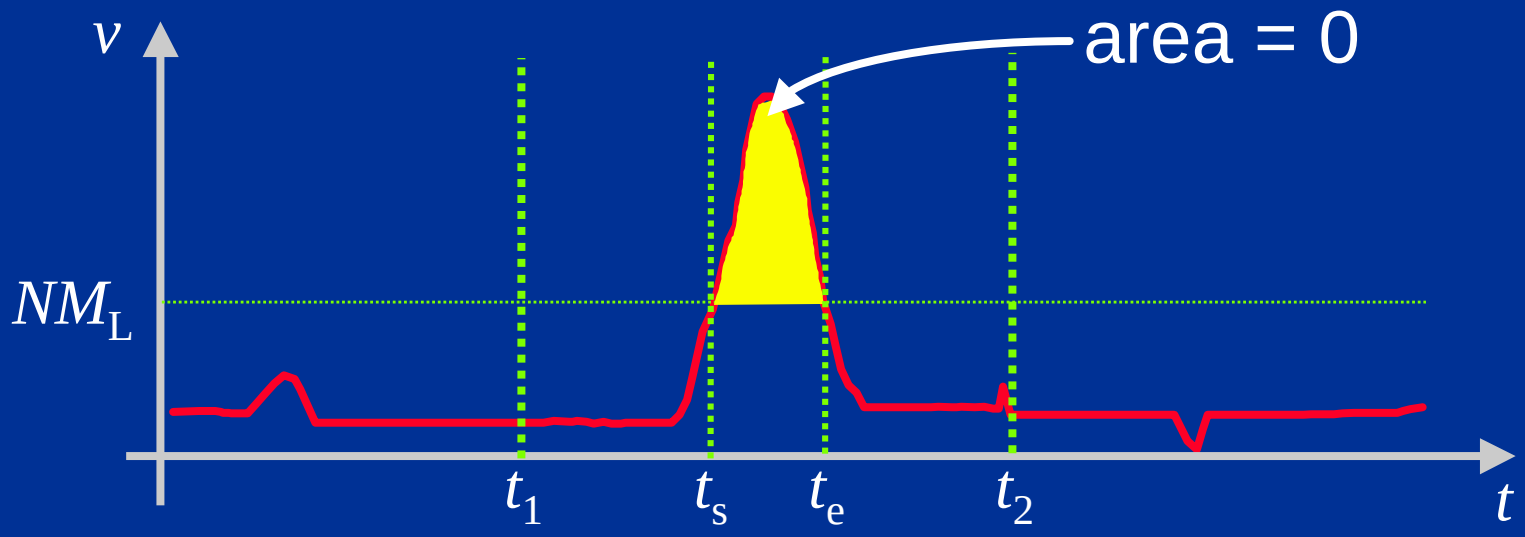
Capacity

- Today, our biggest problem has about 48K transistors (29K tunable), 140K variables (102K slacks), and 111K constraints
- Runs for 4.3 days!
- We go to great lengths to reduce problem size
  - in the tuner code
  - in the pre-processor
  - in our methodology

What higher capacity would permit

- Tuning of larger macros
- Early mode considerations
- Manufacturability
  - replicate constraints and variables at each process corner
  - rich possibilities in choice of objective function
- Explicit (circuit) noise constraints
- Benefits of explicit grouping constraints in post-optimality analysis

Digression: noise constraints



$$v(x,t) \le NM_L \text{ for all } t \text{ in } [t_1, t_2]$$

$$\int_{t_s}^{t_e} \{v(x,t) - NM_L\}\, dt = 0$$

- Combine Harmony and EinsTuner
  - essential for dynamic circuits (see TCAD 6/00)
  - replace rules-of-thumb by actual noise constraints for static circuits

Robustness

- Scaling
  - requires unit change in all variables to have roughly same order of magnitude of impact
  - need non-uniform trust-region; would subsume 2-step updating [Conn et al, SIAM J. Opt. 9/99]
  - very difficult in general
- Weighting
  - essential in any large-scale optimizer
  - need problem-size-dependent weights
- Sensitivity to choices should be minimized
  - finicky behavior not helpful in engineering application

Ease of use

- Nonlinear optimizers should stop assuming they are the “master” program
- There is no standard API for nonlinear optimization packages
- Aspects such as error/message handling and memory management are important; FORTRAN does not help
- Not easy to help the optimizer with domain-specific knowledge
  - e.g., availability of information about Lagrange multipliers in recursive minimax problems

Mixed integer/continuous problems

- Assignment of low threshold voltage devices
- Swapping pin order in multi-input gates (combinatorial problem)
- Working from a discrete gate library
- Discrete choice of taper ratios
- In general, the type of re-structuring that a logic synthesis program may consider

Recursive minimax support

$$\min z$$
$$\text{s.t. } z \geq AT_i + d_{iz}(x), \quad i = 1,2,\cdots,I$$
$$\text{s.t. } AT_i \geq AT_j + d_{ji}(x), \quad j = 1,2,\cdots,J$$
$$\text{s.t. } AT_j \geq AT_k + d_{kj}(x), \quad k = 1,2,\cdots,K$$
$$\vdots$$

- Need a way to force “timing correctness”
  - arrival times and slews at lowest feasible value
  - one of them tight at every stage
  - product of slacks = 0?

Degeneracy and redundancy

- Nonlinear optimizers easy to mislead
- Most engineering problems have plenty of inherent degeneracy and redundancy
- Can optimizers be made less sensitive to degeneracy?
- Example:

$$\min f(x)$$
$$\text{s.t. } c(\mathrm{x}) = 0$$
$$\text{s.t. } c(\mathrm{x}) = 0$$

is degenerate because

$$\nabla_x f(x) = \nabla_x c(x)[\lambda_1 + \lambda_2]$$

Practical considerations

- Detection of impossible problems
  - bound problems
  - obviously impossible constraints
  - not-so-obviously impossible constraints that need a little function evaluation to detect
- Post-optimality analysis
  - final values of Lagrange multipliers can be used to give the designer guidance on obtaining further improvement
  - may need to re-solve for multipliers at end
  - information needs to be collected and presented to designer in nice form
  - formulation of problem may change to take advantage of this analysis

Conclusion (Act I)

- Can robustly carry out static circuit tuning
- Consistent and high-quality results
- Improvement in designers' productivity
- Moves designers' thinking to a higher level
- Presently impacting third generation of z-series (S/390) and PowerPC microprocessors

Conclusion (Act II)

- Open problems in nonlinear optimization whose solution will have a real impact
  - numerical noise
  - capacity
  - robustness, scaling, weighting
  - ease of use, common API
  - discrete variables
  - support for recursive minimax problems
  - degeneracy
  - post-optimality analysis
  - detection of impossible problems

Nuggets

- Optimality and elegance don’t count; improvement does
- Improvement doesn’t count, productivity does
- Productivity doesn’t count, the level of thinking does
- Automatic optimization lifts the thinking of designers to a higher level!

Nuggets

- Lots of extra points for an intuitive graphical user interface
- %age improvement is proportional to the acceptance of the tool
- Inaccurate (convex?) models do not help
- Start points are messy and hopelessly infeasible; problem statements, too!
- Optimizers are forgiving of poor gradients; visualization is a great debugging aid
- Never use optimization as a black box!