Lecture 1: Introduction to Physical Design

Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Logistics

- **Instructor: Dr. Tsung-Wei Huang (TW)**
  - tsung-wei.huang@utah.edu
  - MEB 2124
- **Time: MoWe / 3 PM – 4:20 PM (excluding holidays)**
  - In person: WEB L114
  - Zoom (back-up plan): https://utah.zoom.us/j/2468214418
- **Webpage**
  - GitHub: https://github.com/tsung-wei-huang/ece5960-physical-design/
- **Office hour: by appointment**

Scoring

- **Total 100 points**
  - Four programming assignments of key physical design algorithms
  - Class presentation for important physical design algorithms
- **Academic integrity**
  - **You should always finish assignments by yourself!**
  - I trust you but don't take it for granted (it's your career not mine…)
  - https://regulations.utah.edu/academics/6-400.php
- **Textbook**
  - No need to buy any textbook
  - Slides will be available and are enough

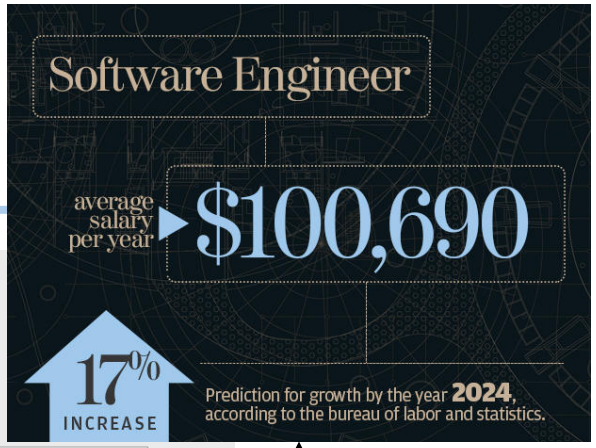
Class Philosophy

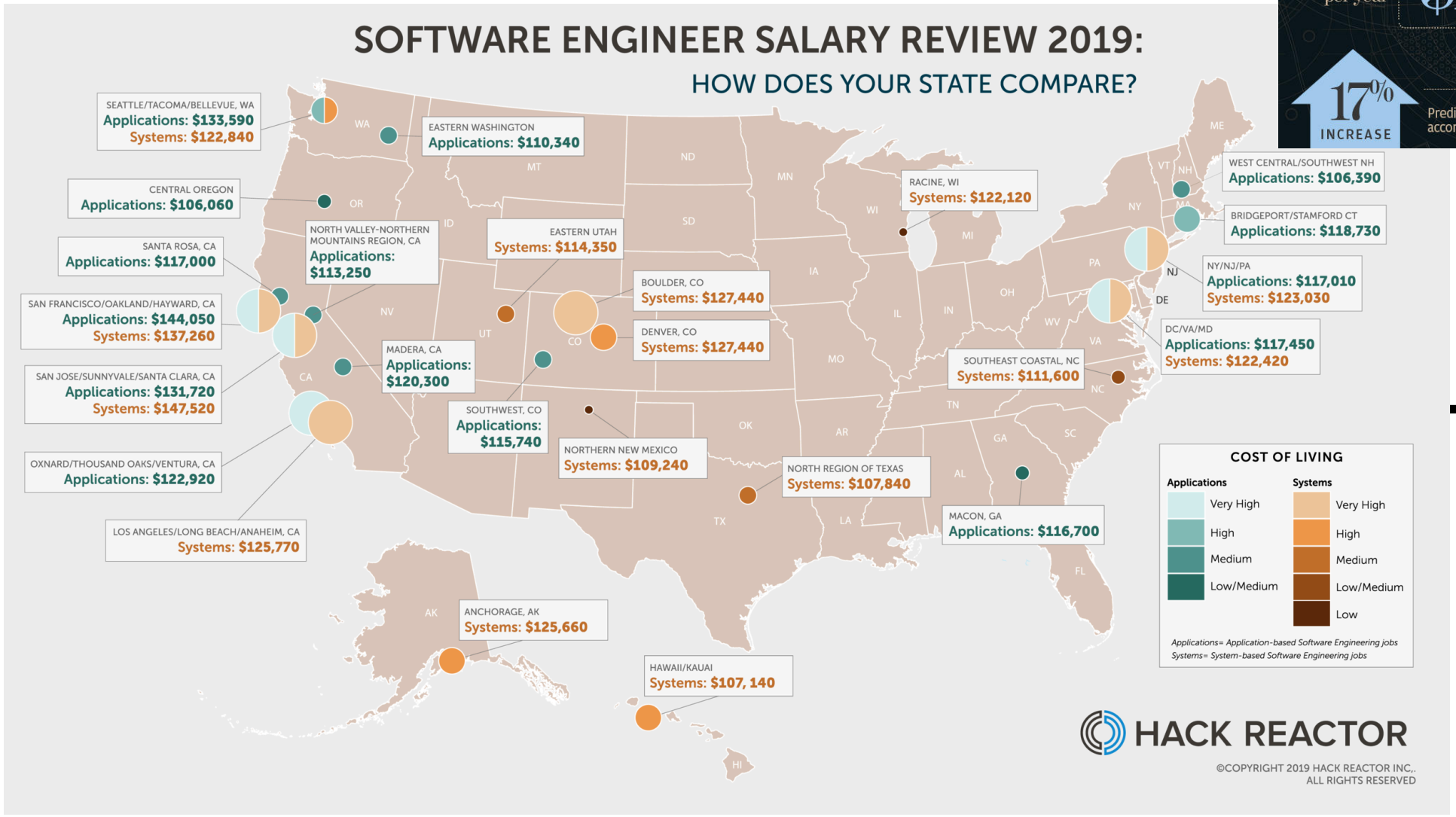
- **Learn important physical design algorithms**
  - Combinatorial optimization problems (e.g., graph, discrete math)
  - Analytical methods (e.g., linear programming)
- **Program important physical design algorithms**
  - Write C/C++/Python code to solve physical design problems
  - Implement various data structures and algorithms
- **At the end of the class, I want you to**
  - Understand important computer design problems
  - Improve your coding skills and algorithm knowledge
  - Have more job opportunities in software companies (high pay today!)

Programming Machines

- **Linux programming environment**
  - Utah CADE Linux machine: https://www.cade.utah.edu/
  - Your personal MacBook, Linux desktop, etc.
- **Evaluation environment**
  - Instructor's server: twhuang-server-01.ece.utah.edu
  - Email the instructor (tsung-wei.huang@utah.edu) for an account with
    - Your account name
    - Your default password
- **Programming languages**
  - No restriction but highly recommend C/C++

Why Programming…?

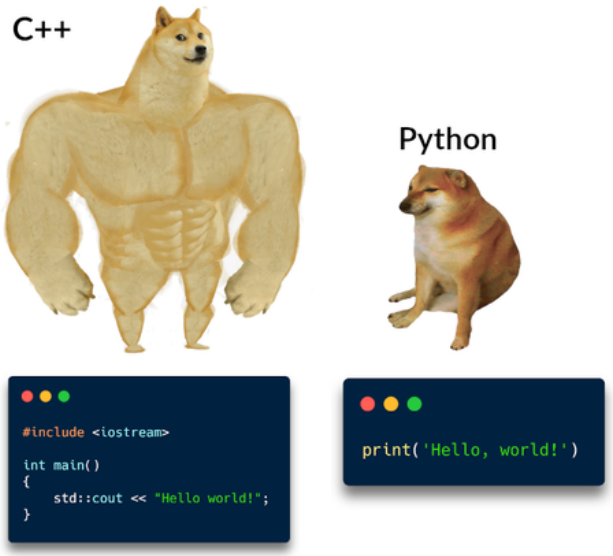




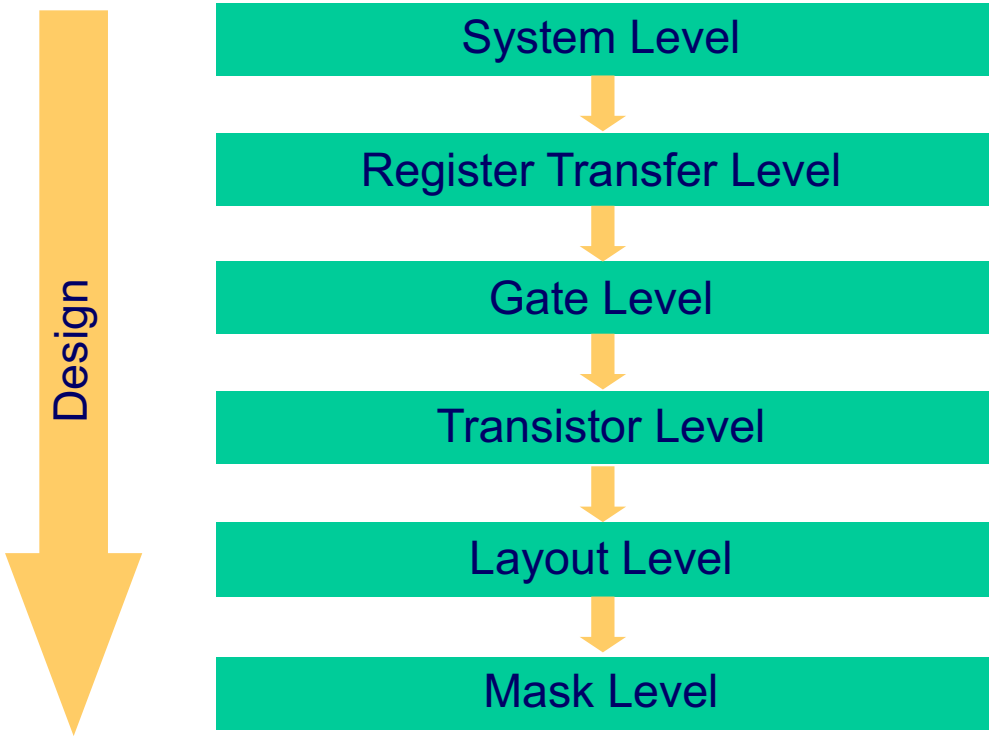
WHICH PROGRAMMING LANGUAGE TO LEARN FIRST?

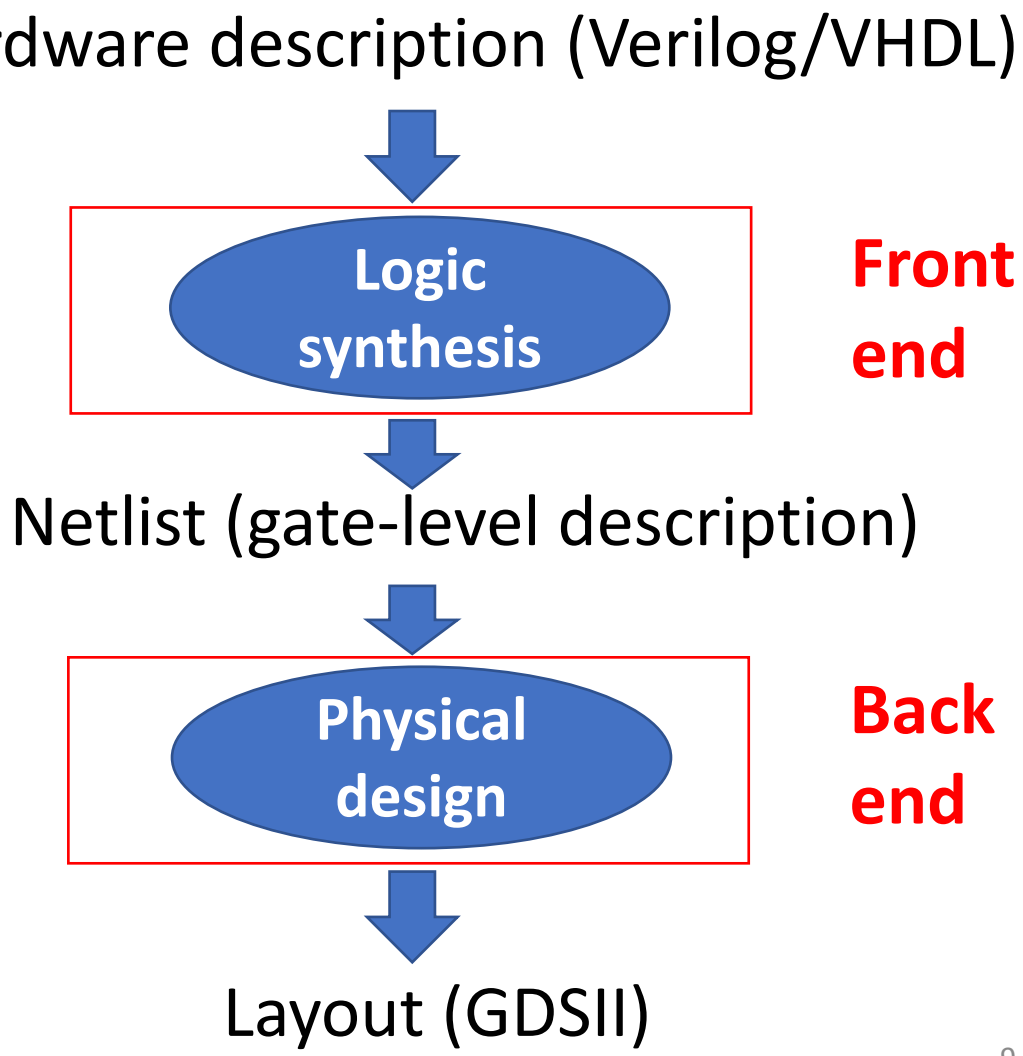
From my 10-year Programming Perspective

- **In general, your logic thinking matters the most**
  - How to formulate a problem computationally?
  - How to solve a computational problem efficiently?
  - How to improve the solution quality programmatically?
- **However, as a CE/CS student, I highly recommend C++**
  - C/C++ let you know how performance works
    - Software design patterns
    - Computer hardware and memory hierarchies
  - Python let you know how productivity works
  - Java let you know how portability works

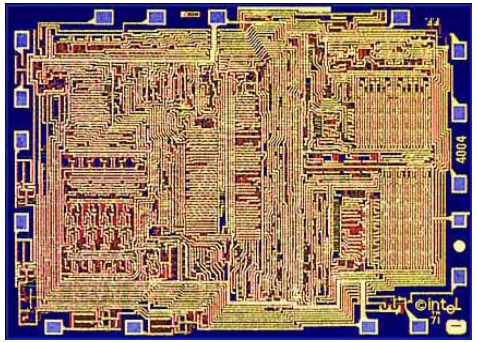


Silicon Compiler (Design Automation)

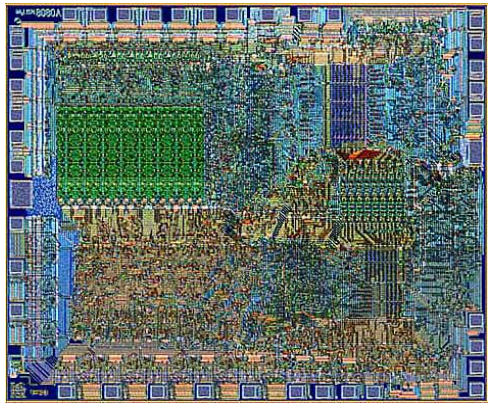




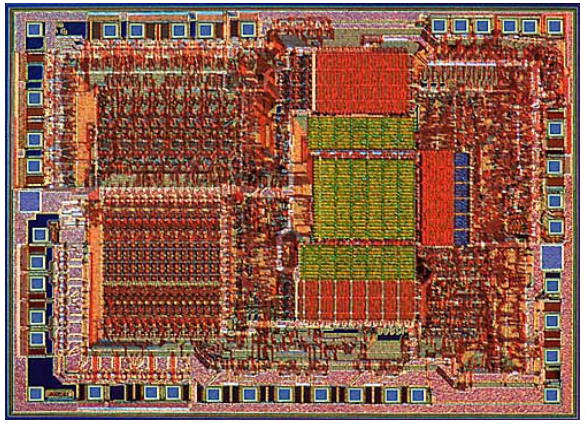
Why Silicon Compiler?



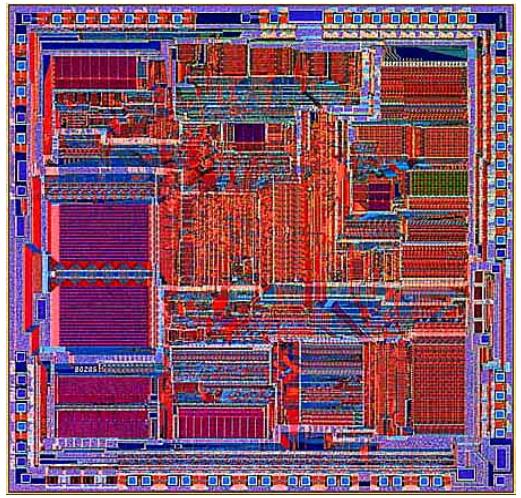
Intel 4004 (‘71)



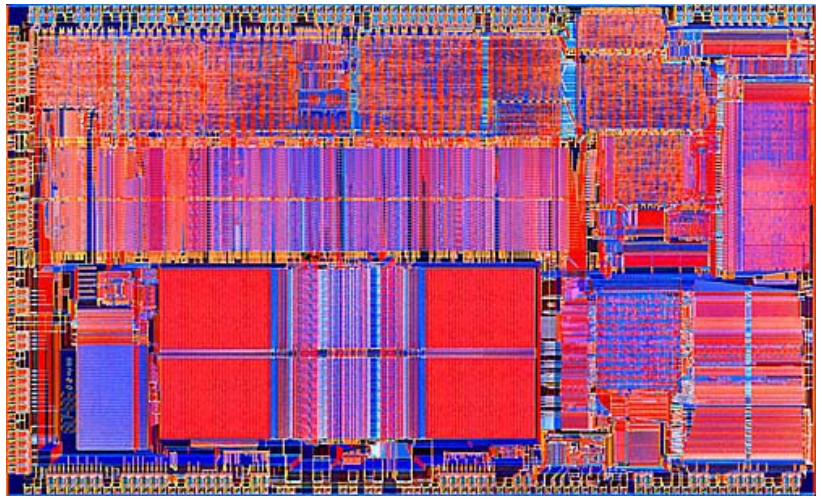
Intel 8080



Intel 8085

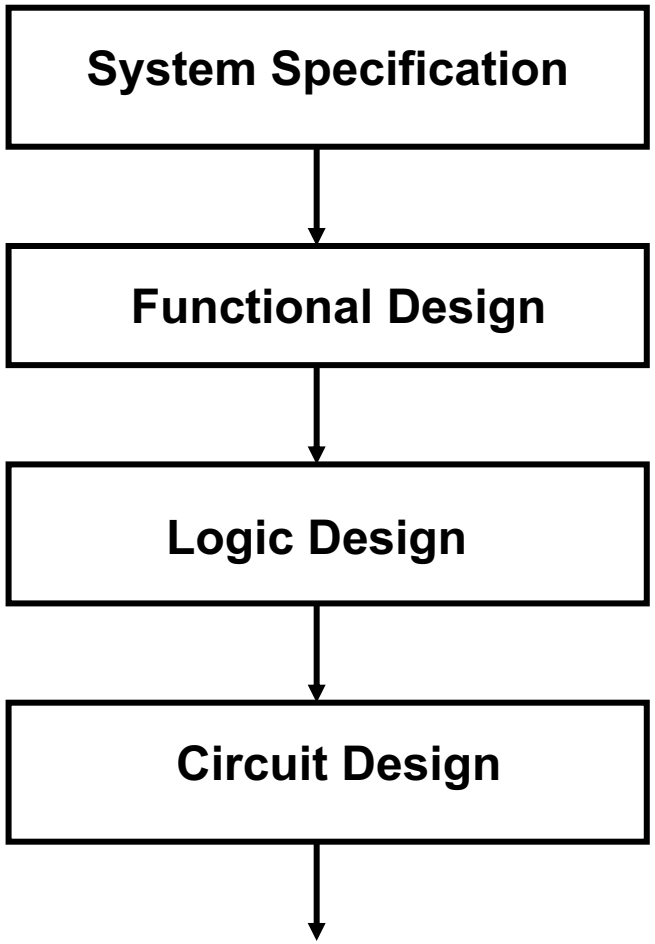


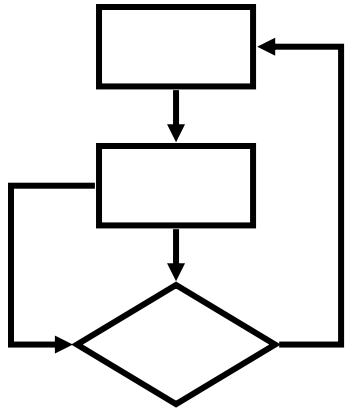
Intel 8286



Intel 8486

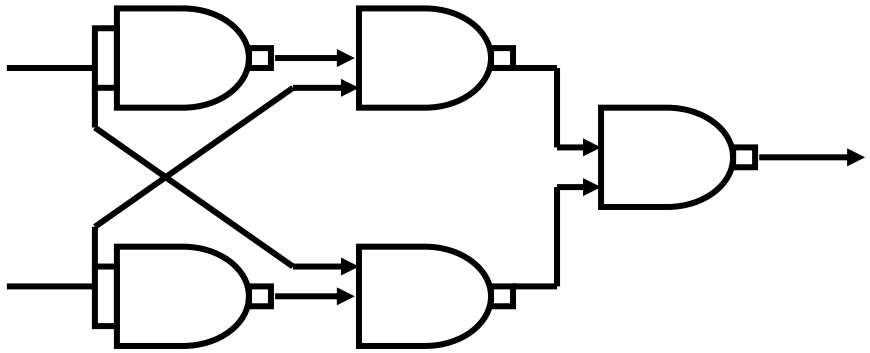
Silicon Compiler: Front-end



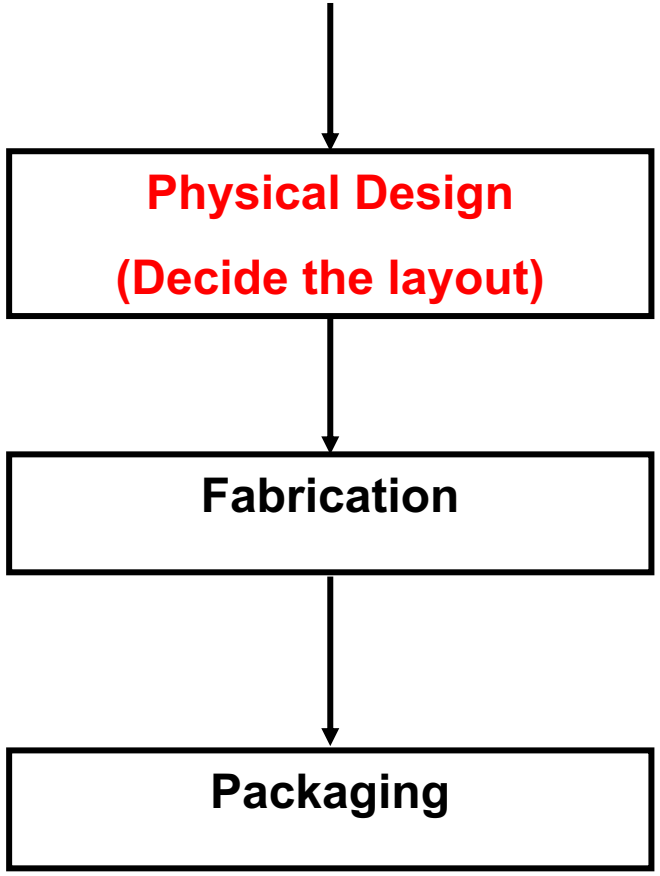


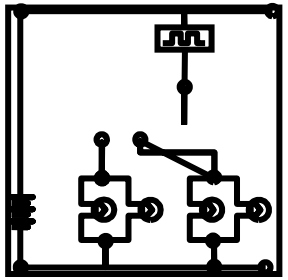
X=(AB*CD)+(A+D)+(A(B+C))

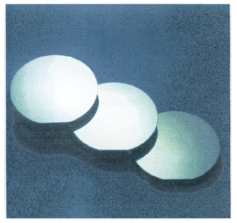
Y=(A(B+C))+AC+D+A(BC+D))

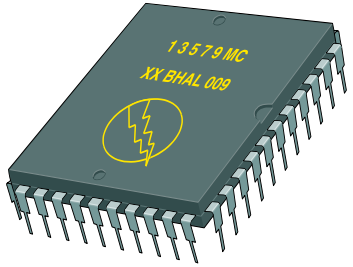


Silicon Compiler: Back-end

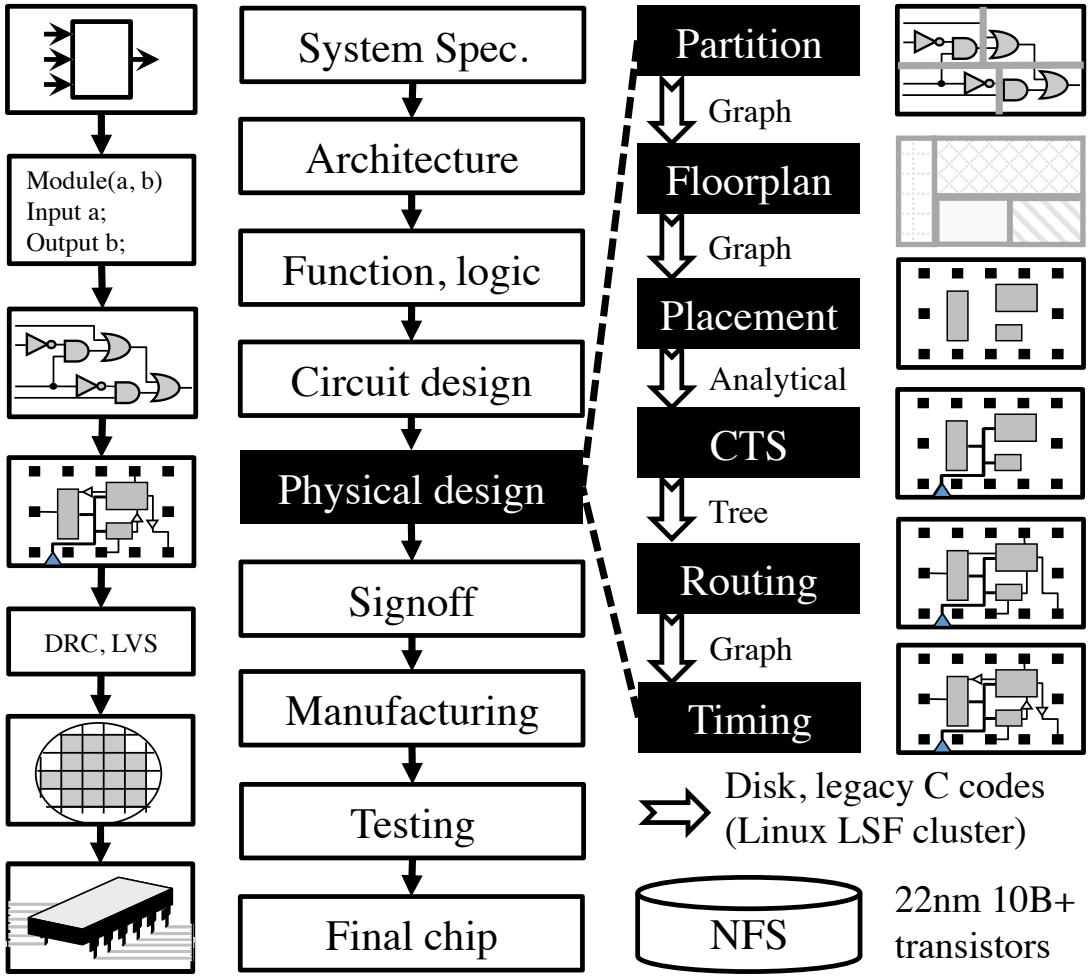






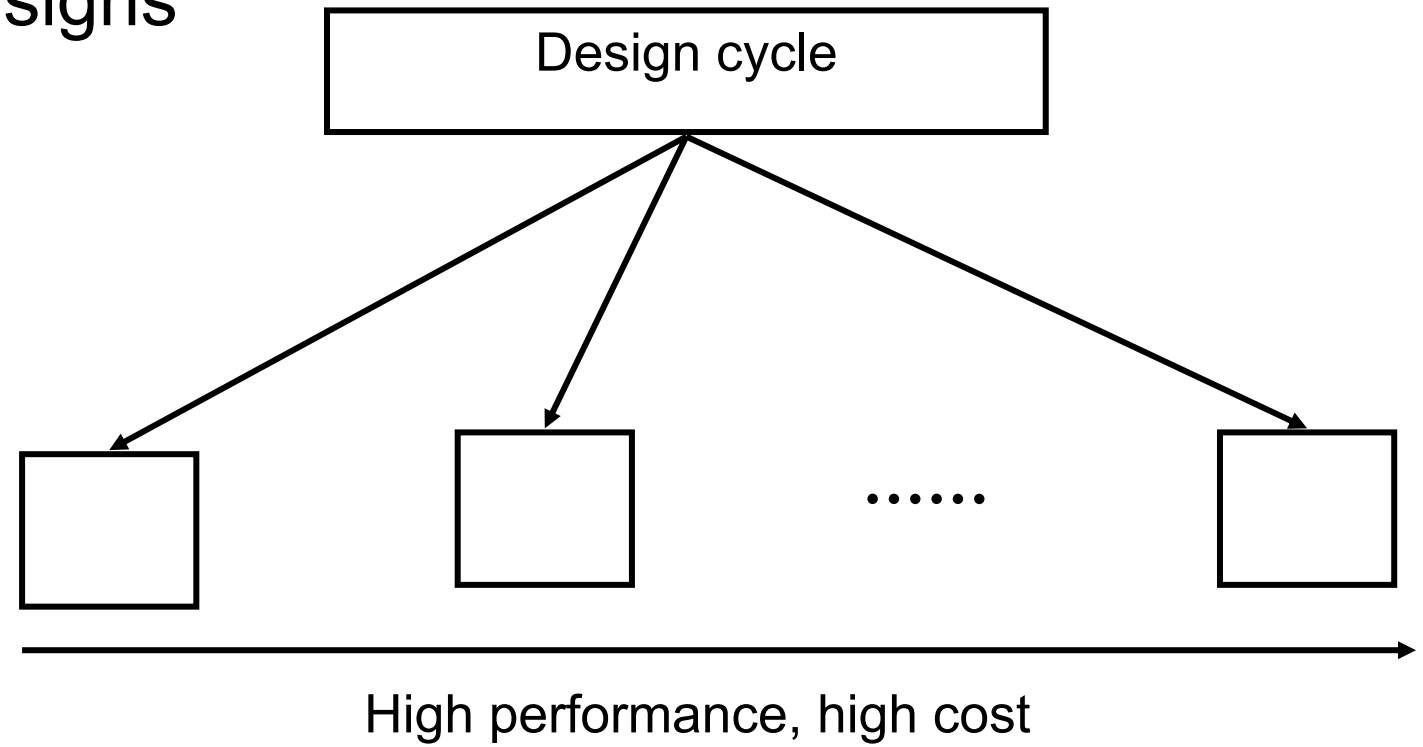


Physical Design Flow



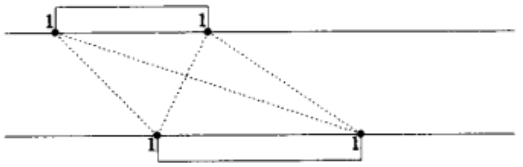
Complexity of Physical Design

- More than **10 billion** transistors
- Performance driven designs
- Power-constrained designs
- Time-to-Market



History of Physical Design

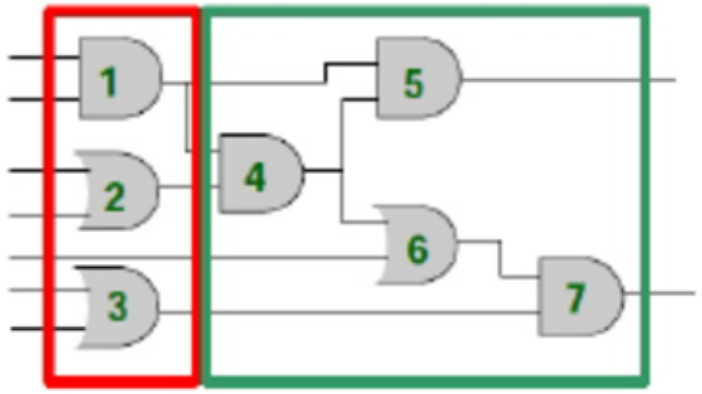
- **Born in early 60's (board layout)**
- **Passed teenage in 70's (standard cell place and route)**
- **Entered early adulthood in 80's (over-the-cell routing)**
- **Declared dead in late 80's !!!**
- **Found alive and kicking in 90's**

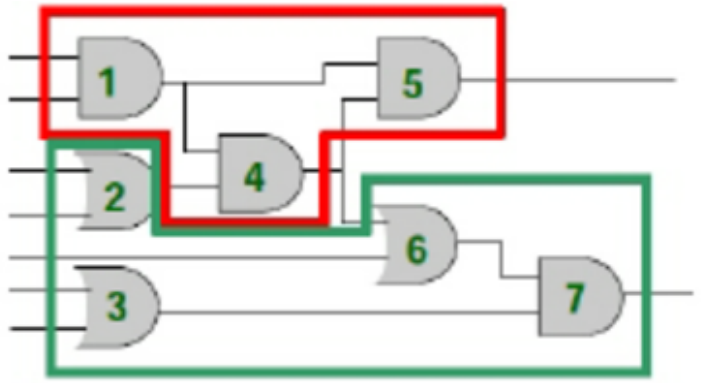


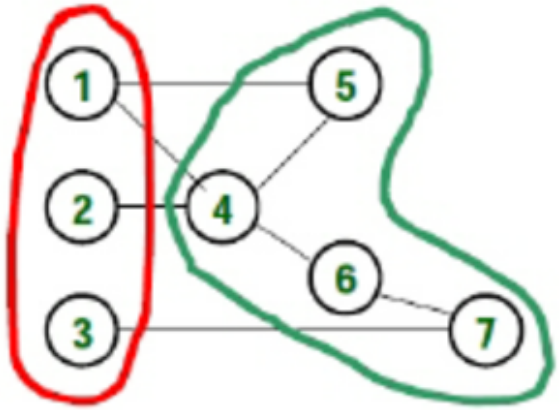
- **Physical Design has become a key force in design cycle**
  - Thanks to the deep submicron scaling! We really need automation!
  - Thanks to algorithm innovations in computer science
  - Thanks to computer hardware advancement

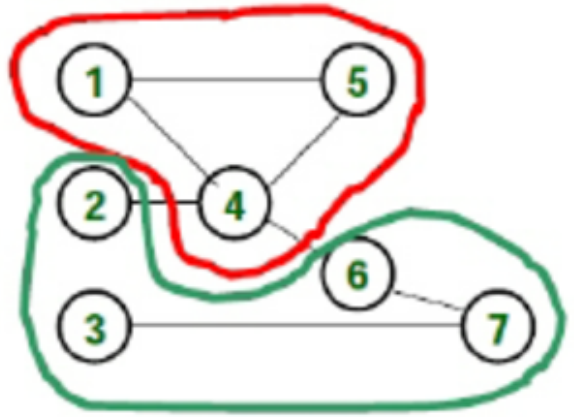
Circuit Partition Example

- Which partition result is better?



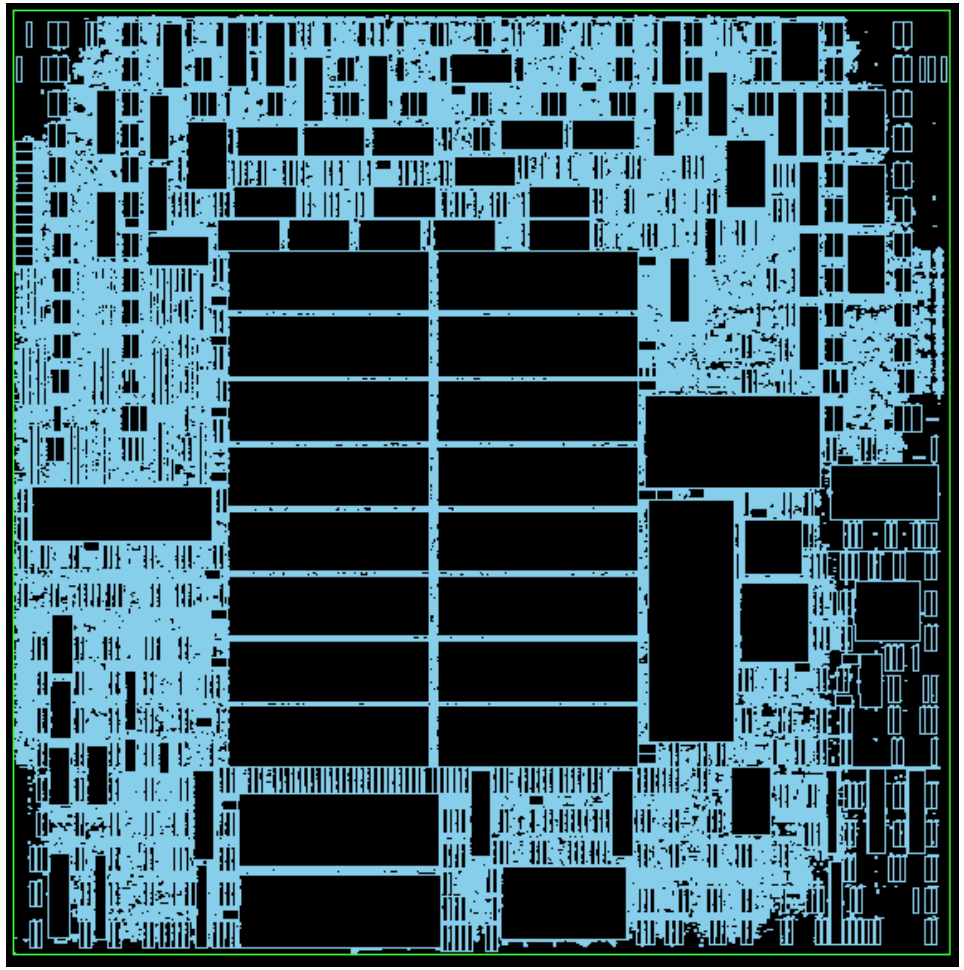






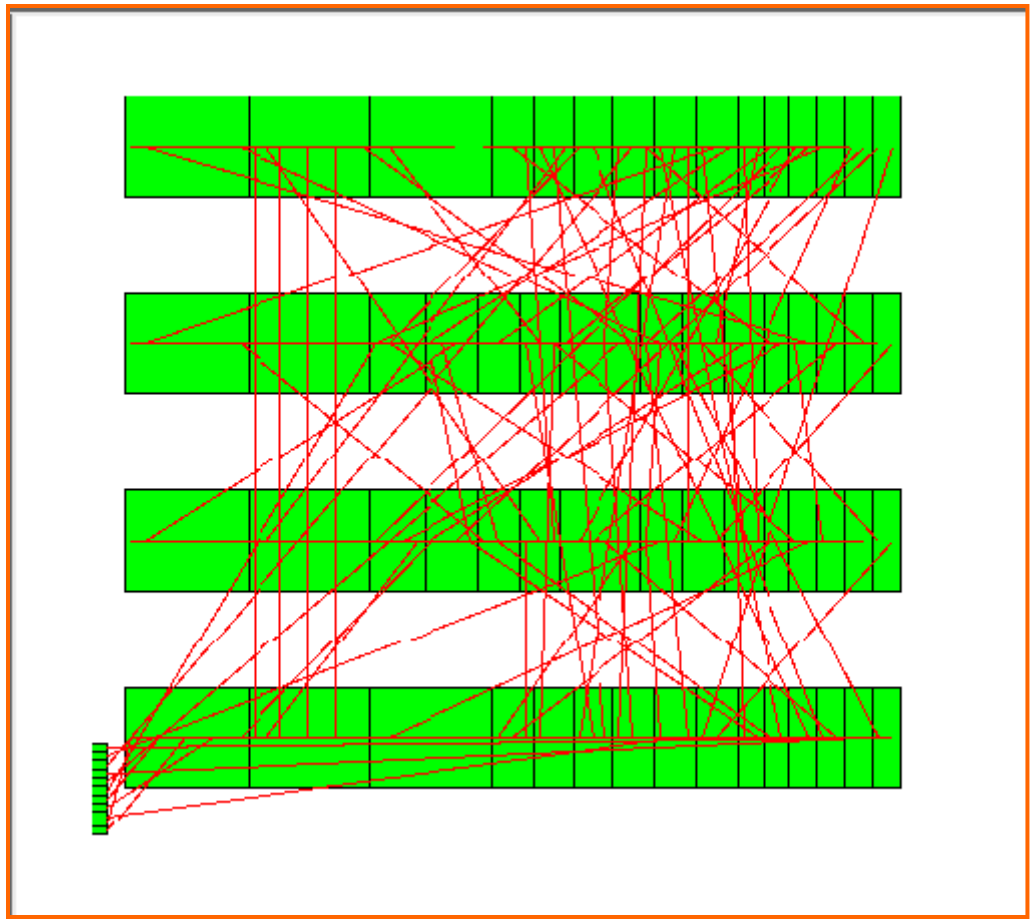
Floorplan Example

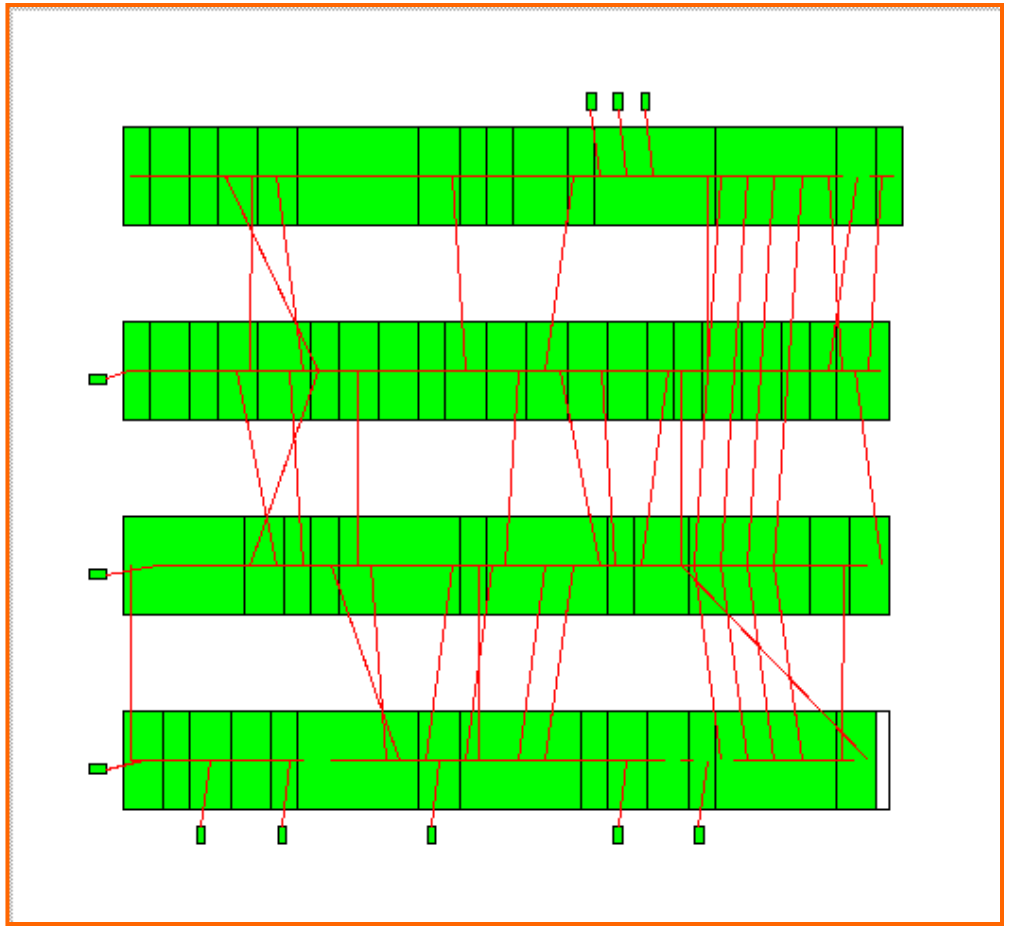
- Many macros
- Data paths + logic elements
- I/O constraints
  - Area
  - Wire bond
  - Connection (e.g., gate 1 links gate 2)
  - …



Placement Example

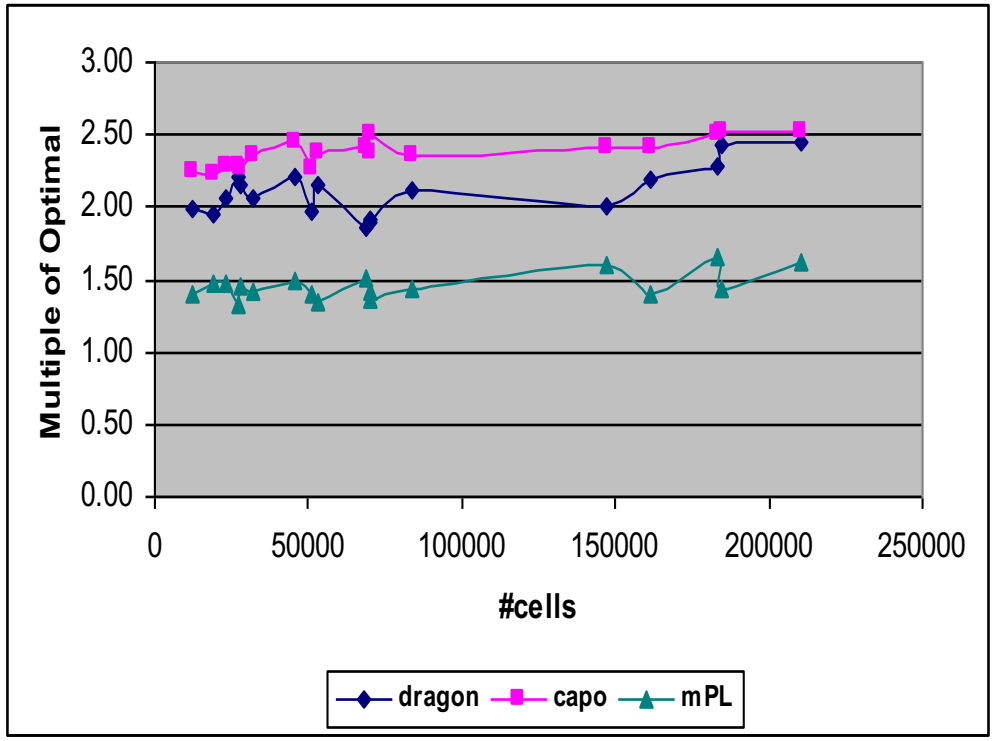
- Which placement result is better?

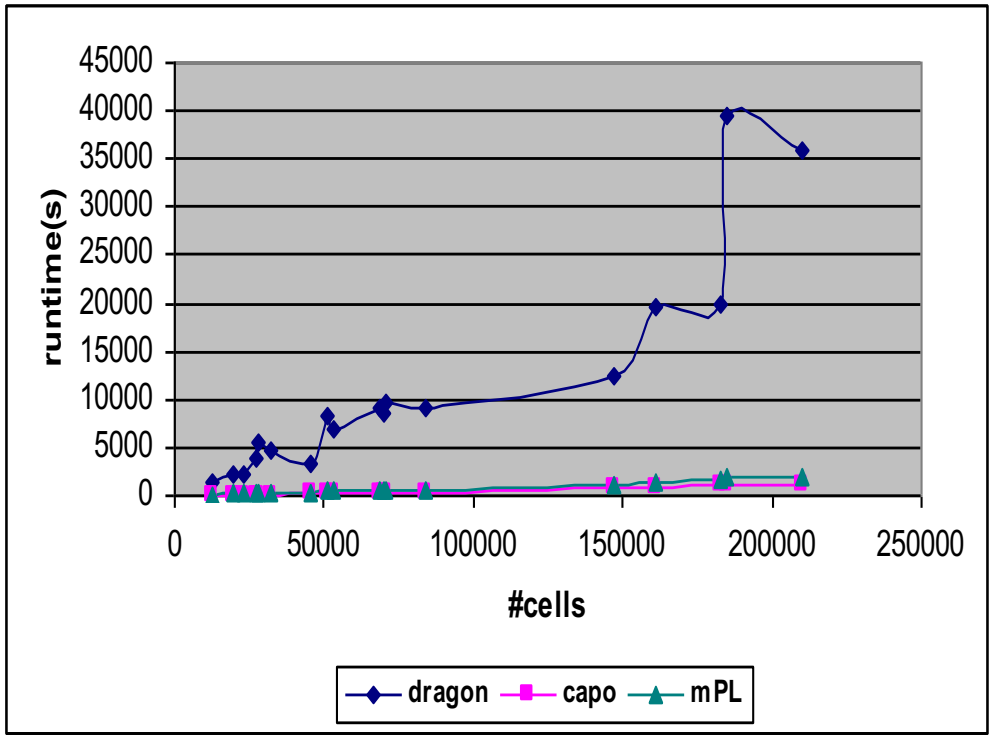




Placement Example (cont'd)

- Commercial tools are still away from optimal solutions …

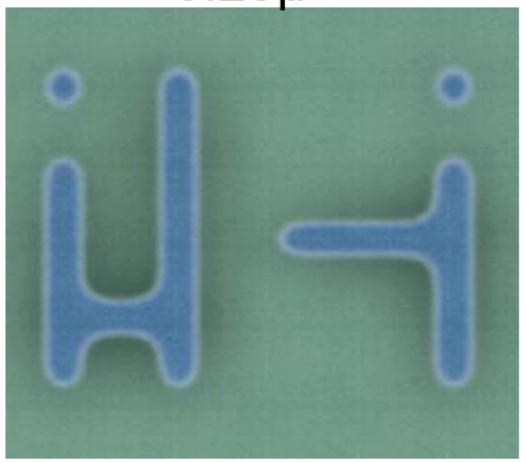




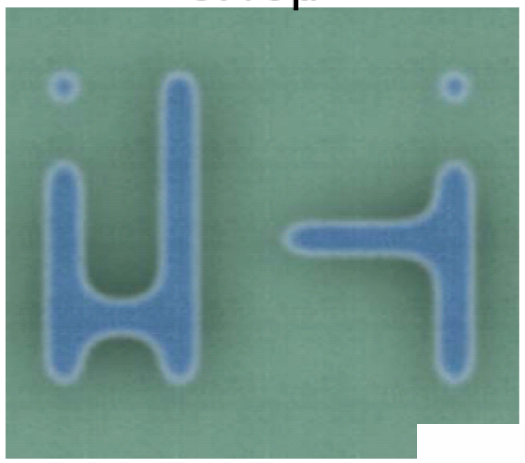
Lithography Example

Layout

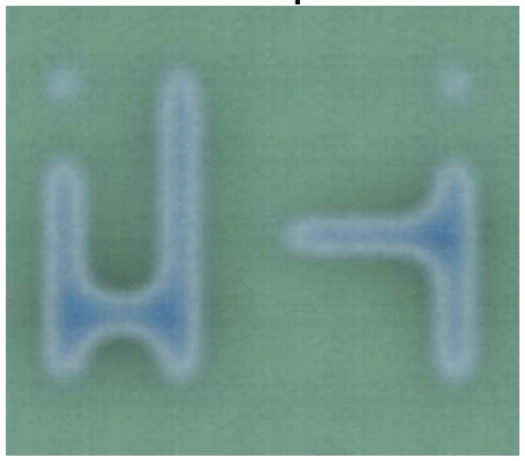
0.25μ



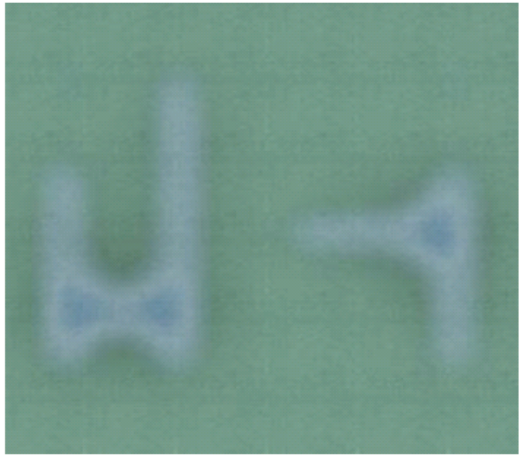
0.18μ



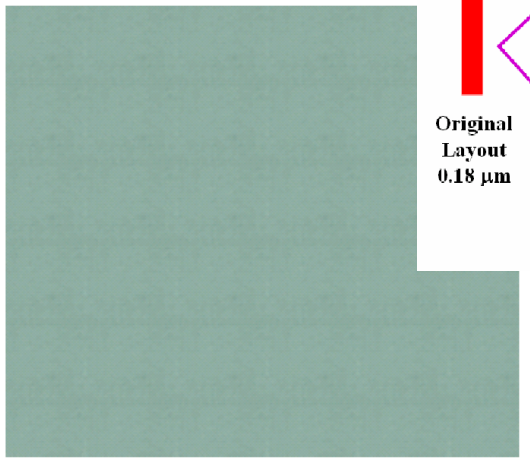
0.13μ

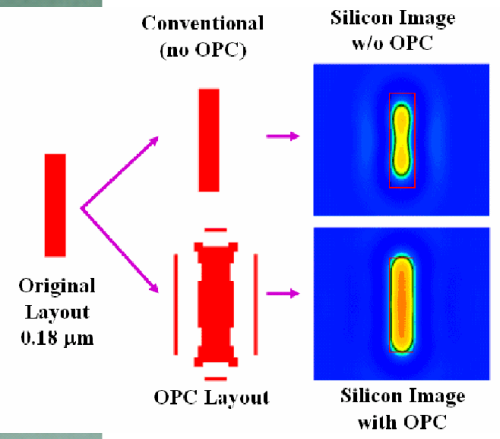


90-nm



65-nm





Lithography Example (cont'd)

Market Summary > ASML Holding NV

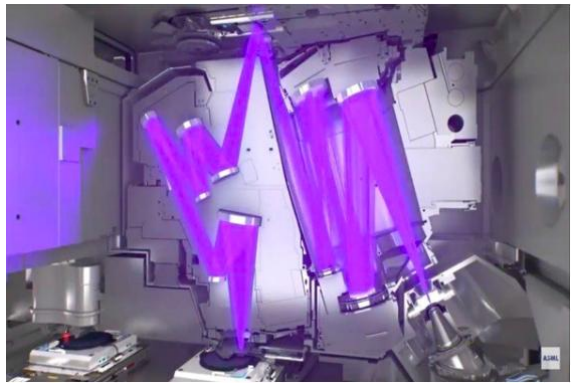
577.31 USD

+426.12 (281.84%) ↑ past 5 years

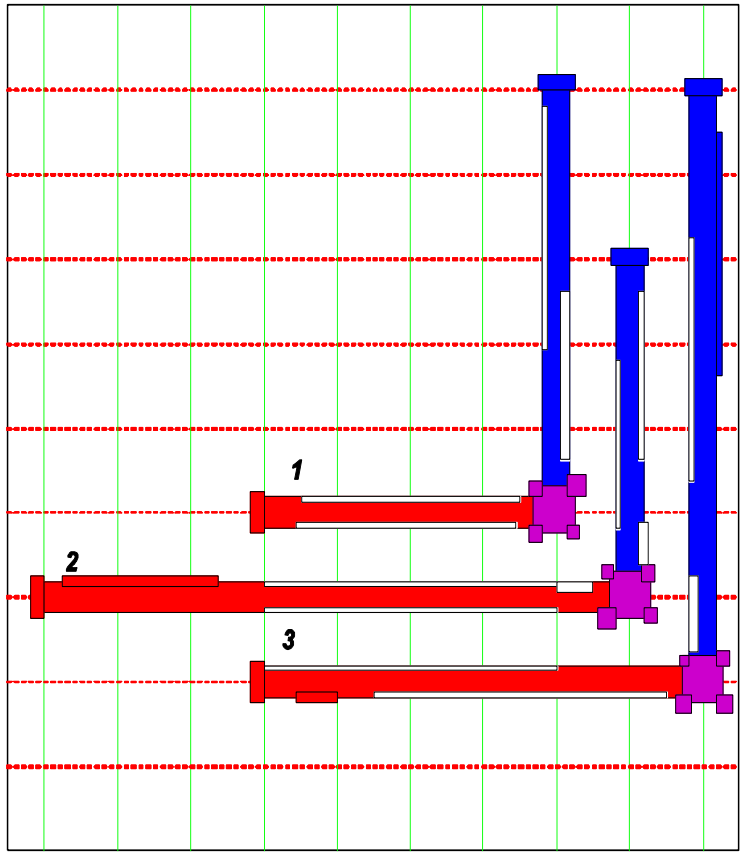
Closed: Aug 5, 5:04 PM EDT • Disclaimer

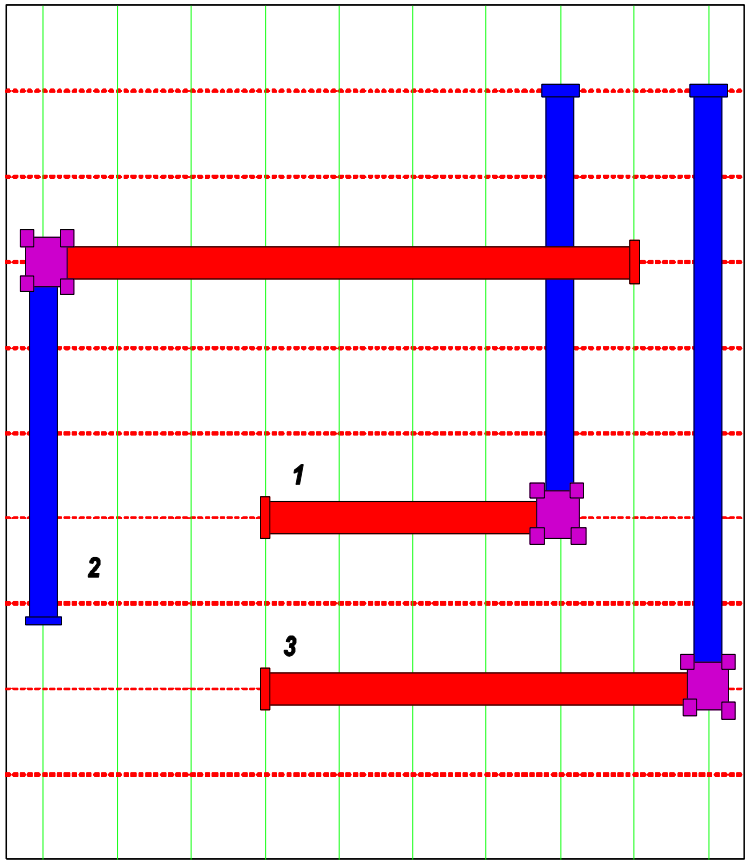
After hours 577.31 0.00 (0.00%)

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max



Optical Proximity Correction (OPC)





More OPC friendly

Solving Physical Design Problems

- To optimize design among different objectives, area, power, performance, and etc.
- **Fundamental questions: How to do it smartly?**
- Definition of algorithm in a board sense: A step-by-step procedure for solving a problem. Examples:
  - Cooking a dish
  - Making a phone call
  - Sorting a hand of cards
- Definition for computational problem
  - A well-defined computational procedure that takes some value as input and produces some value as output

Computational Problems

- **Decision problem**
  - Given a positive integer n, determine if n is prime
- **Search problem**
  - Find all even numbers in X=[1,2,3,4,5]
- **Counting problem**
  - Given a positive integer n, count its number of nontrivial prime factors
- **Optimization problem**
  - Given a graph G, find an independent set of G of maximum size
- …

Course objective: How do we solve these problems smartly?

Example: Big Mod Problem

- **Computational problem description**
  - Input: a, b, c (3<a, c<5000000, 1<b<2147483647)
  - Output: $x = a^b \pmod c$
- **A simple solution**

```
for(i=1, x=1; i<=b; i++)
    x = (x*a)%c;
std::cout << “a^b % c = ” << x << std::endl;
```

Example: Big Mod Problem

- **Computational problem description**
  - Input: a, b, c (3<a, c<5000000, 1<b<2147483647)
  - Output: $x = a^b \pmod c$
- **A simple solution**

```
for(i=1, x=1; i<=b; i++)
    x = (x*a)%c;
std::cout << “a^b % c = ” << x << std::endl;
```

- **Is the solution smart enough?**
  - We need a total of “b” iterations
  - Very slow execution for large b

Refined Solution (Recursive)

- Input: a=2, b=15, c; Output: $x = a^b \pmod c$
- **Simple solution needs a total of b=15 iterations**
- Let's refine the iteration count recursively

$2^{16} = (\ 2^8\ )^2$ total 1 calculation
$2^8 = (\ 2^4\ )^2$ total 1 calculation
$2^4 = (\ 2^2\ )^2$ total 1 calculation
$2^2 = (\ 2^1\ )^2$ total 1 calculation
$2^1 = (\ 2^0\ )$ total 1 calculation

Refined Solution (Recursive) (cont'd)

```
#define SQUARE(x) (x*x)
int bigmod(int a, int b, int c) {
    if(b==0) return 1;
    else if(b%2==0)
        return SQUARE(bigmod(a,b/2,c)%c)%c;
    else return ((a%c)*bigmod(a,b-1,c))%c;
}
int main() {
    int a,b,c;
    while(std::cin >> a >> b >> c)  std::cout<<bigmod(a,b,c)<<endl;
    return 0;
}
```

$2^{16} = (2^8)^2$	total 1 calculation
$2^8 = (2^4)^2$	total 1 calculation
$2^4 = (2^2)^2$	total 1 calculation
$2^2 = (2^1)^2$	total 1 calculation
$2^1 = (2^0)$	total 1 calculation

Refined Solution (Iterative)

```
int bigmod(int a, int b, int c) {
  int result = 1;
  while(b > 0) {
    if(b & 1) {
      result = (result * a) % c;
    }
    b = b >> 1;
    a = a*a%c;
  }
  return result;
}
```

Runtime Comparison

- https://www.onlinegdb.com/

Runtime (with –O2) – Enabled Optimization			
(a, b, c)	Naive	Refined (recursive)	Refined (iterative)
(2, 1000000, 3)	13720 us	24 us	22 us
(2, 10000000, 3)	136148 us	9 us	**4 us**
(2, 100000000, 3)	1320049 us	8 us	**4 us**

Code Analysis

- https://godbolt.org/z/6zT4cK3Wz

	Naive	Refined (recursive)	Refined (iterative)
Lines of C++ Code	7	11	11
Lines of Assembly (-O0)	55	99	**59**
Lines of Assembly (-O2)	27	57	**32**

Magic behind the Refined Solution

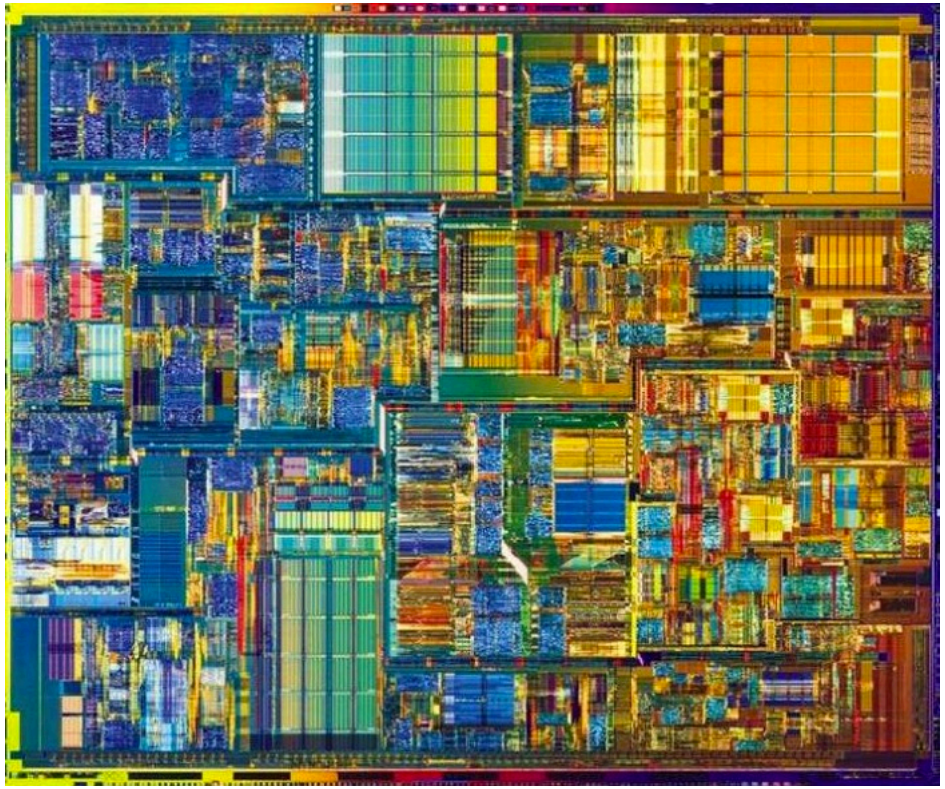
In fact, there is a name for the refined method:

Divide and Conquer (D&C) algorithm

(used to solve >50% large and complex computational problems, e.g., Google Map, sorting, etc.)

D&C is Widely used in Physical Design

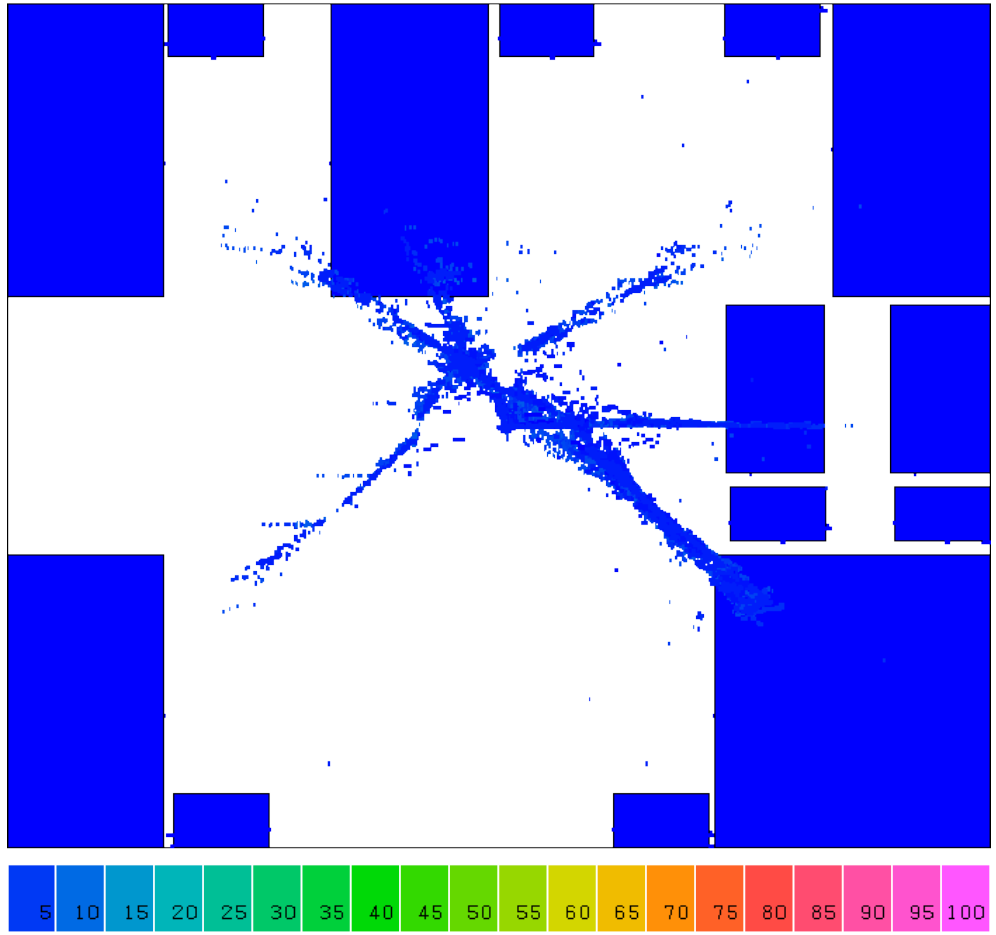
- Modern circuits sizes are too large to handle in a flat fashion



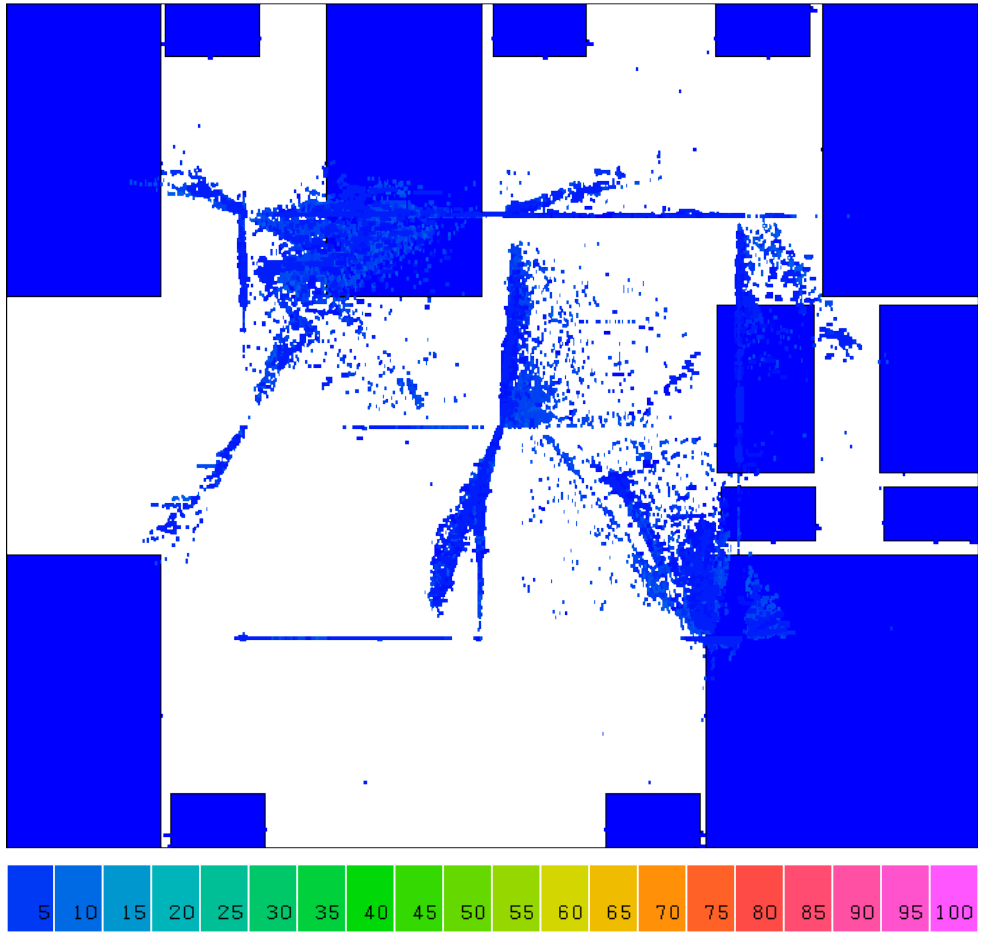
42M transistors
1.5MHz; 224 mm^2

Directly solving the original problem takes forever to finish …

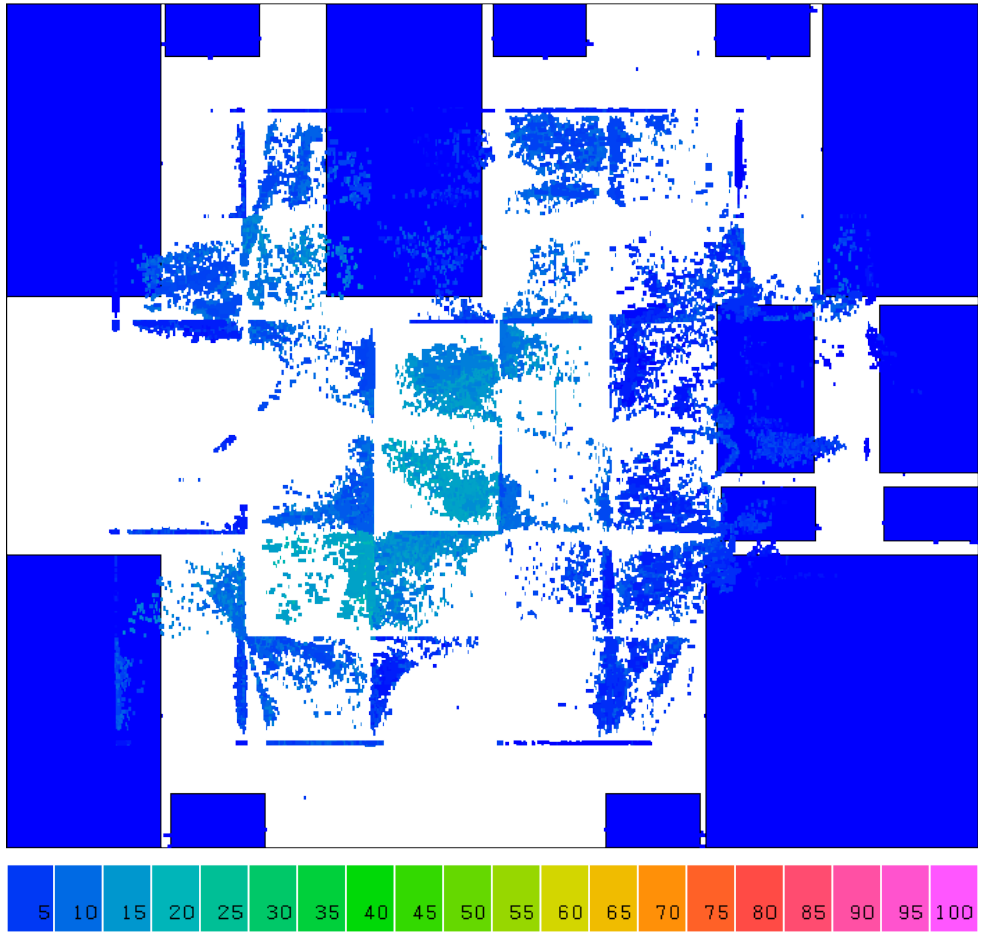
Example: D&C-based Placement



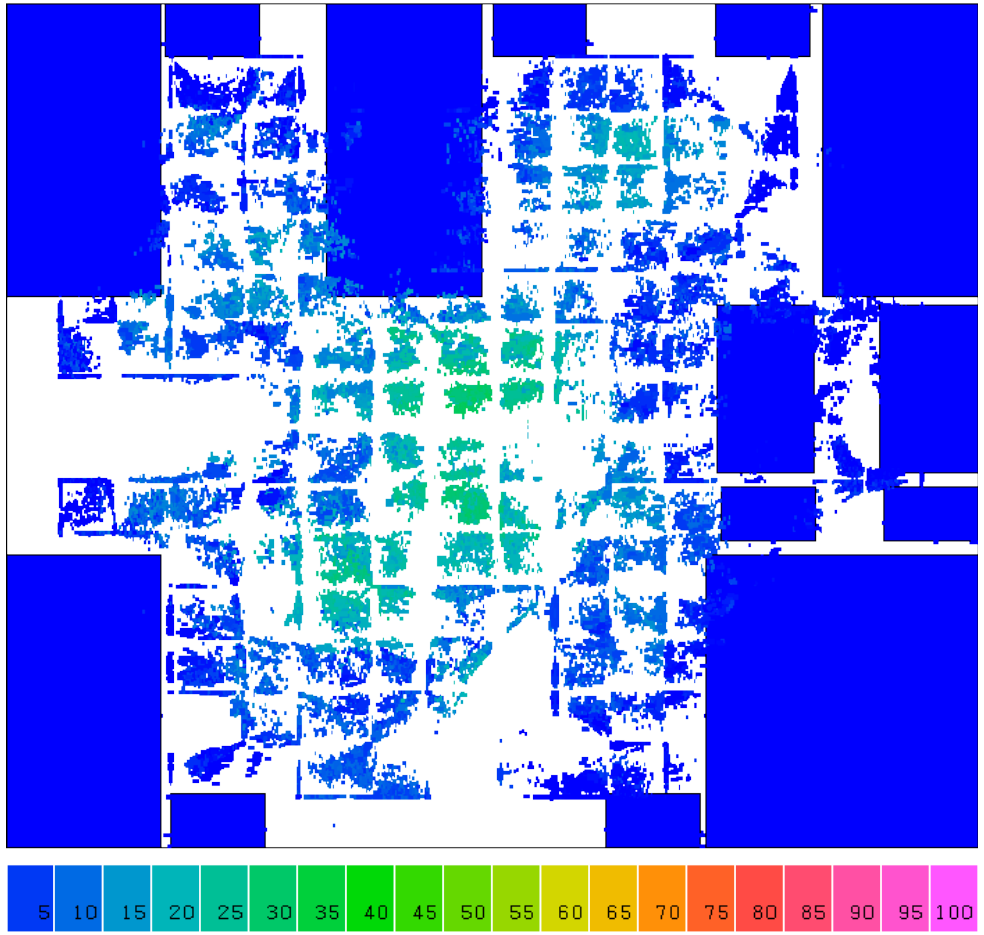
Example: D&C-based Placement



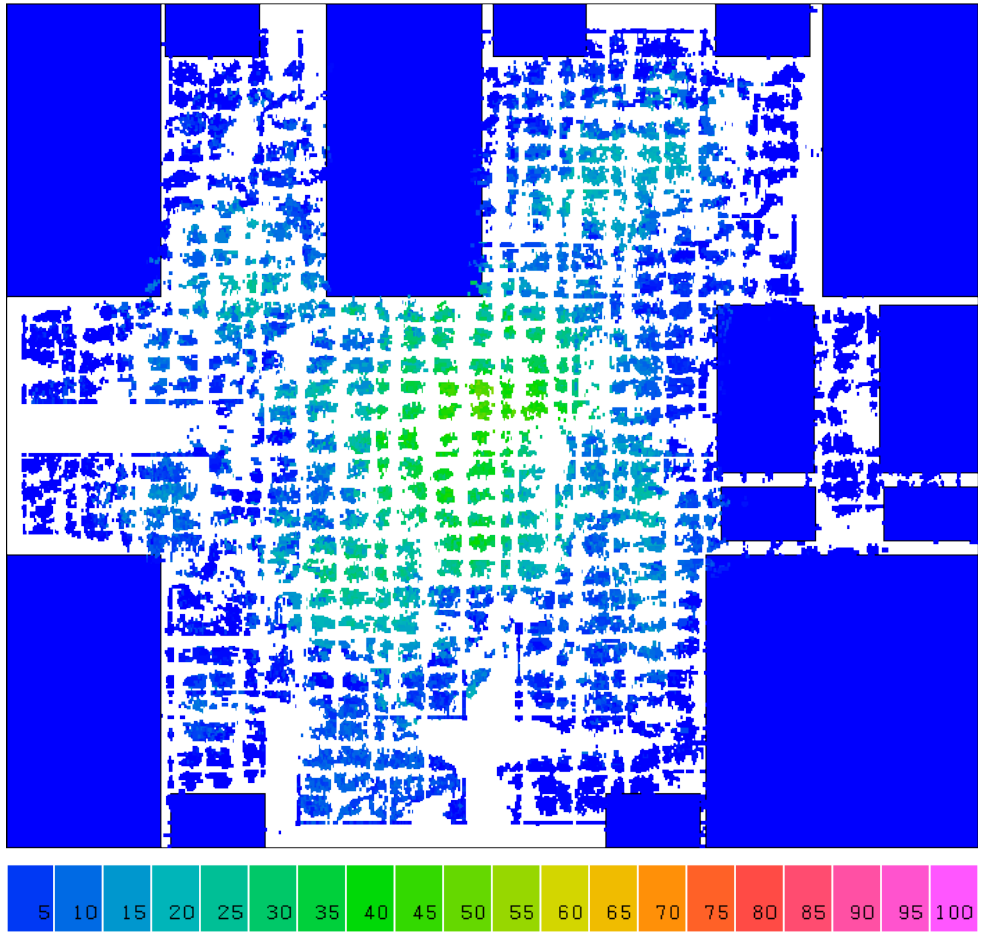
Example: D&C-based Placement



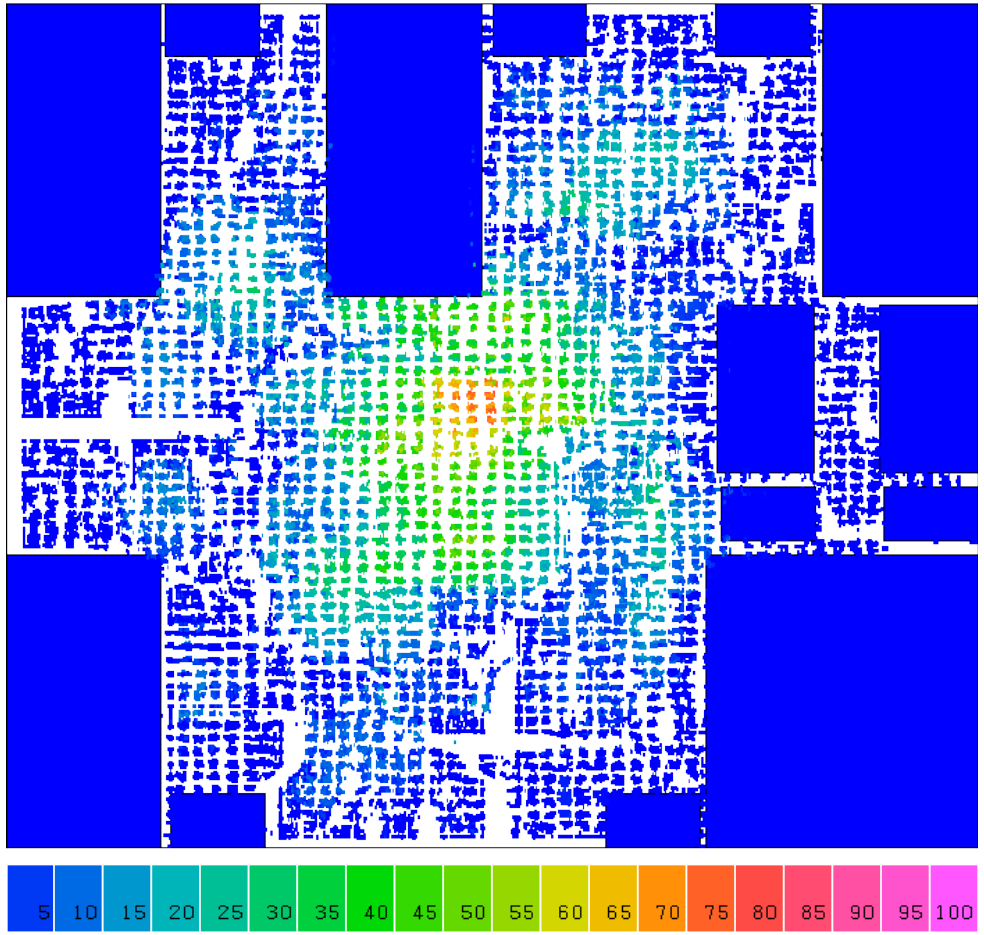
Example: D&C-based Placement



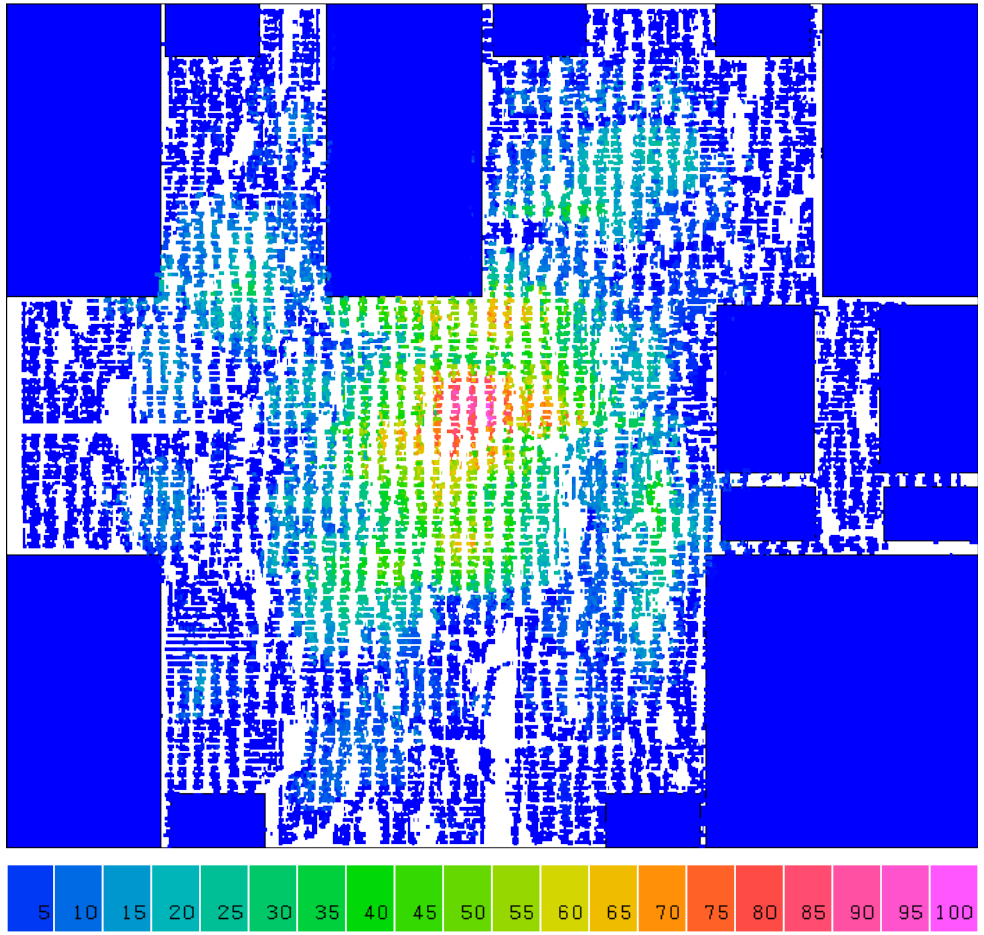
Example: D&C-based Placement



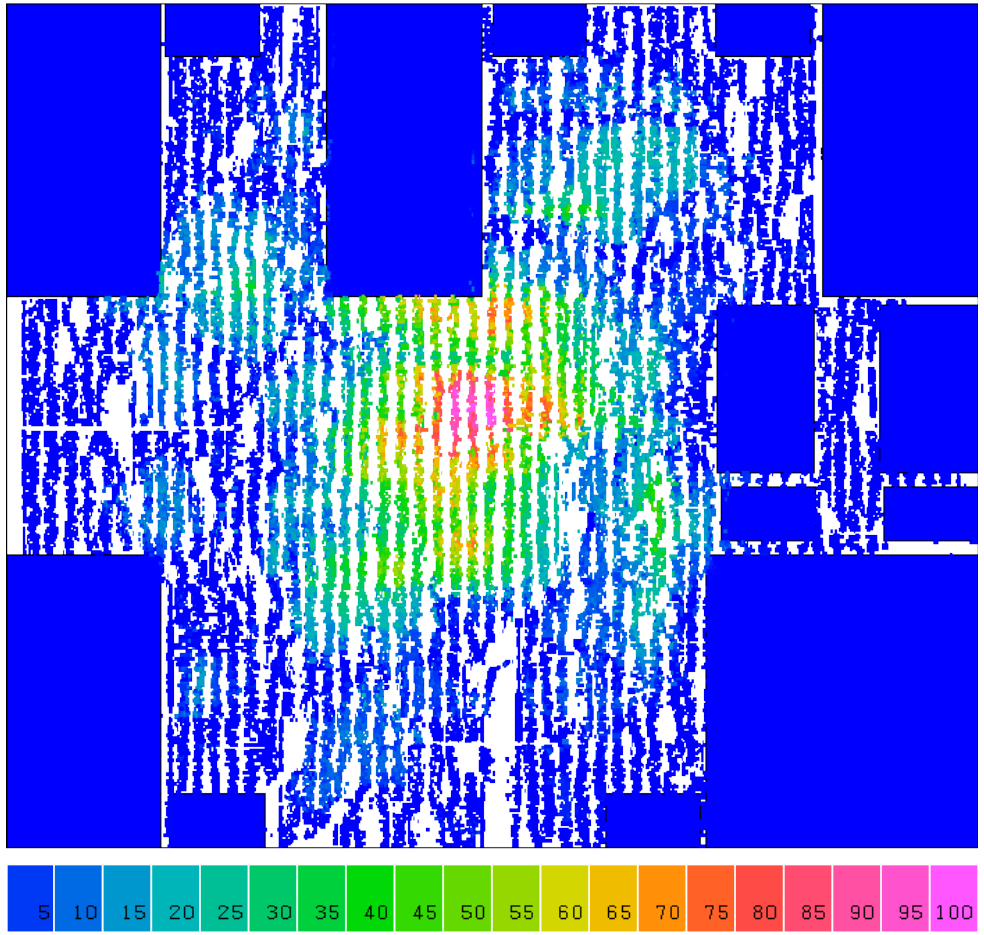
Example: D&C-based Placement



Example: D&C-based Placement



Example: D&C-based Placement



Summary

- **We have discussed class logistics**
- **We have discussed design automation flow**
- **We have discussed physical design**
- **We have discussed concept of computational problems**
- **We have discussed a computational problem example**
  - Problem formulation
  - Naïve solution
  - Refined solution
  - Practical code analysis techniques