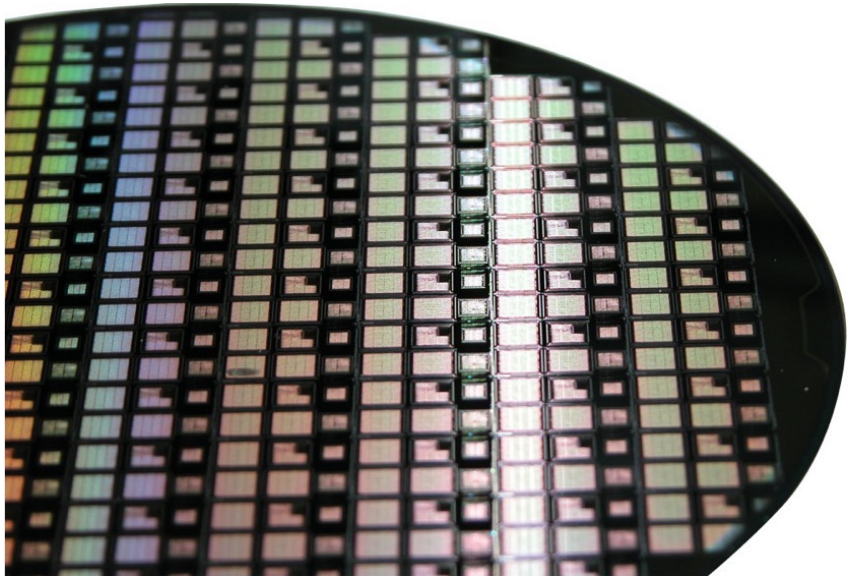
ETH zürich

Netlist: Turning ideas in HDL into physical gates



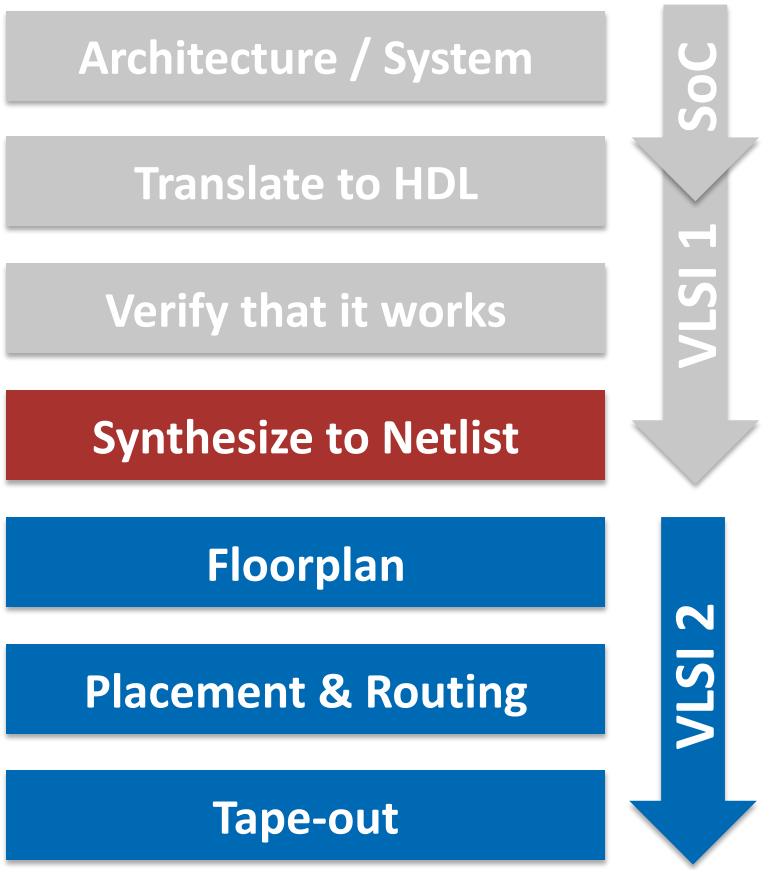
19 February 2026

Luca Benini

Frank K. Gürkaynak

Back to physical design, our main topic

- **Front-end design**
  - Starts from an idea, specification
  - Finds a suitable architecture (SoCDAML)
  - Translates it to HDL (VLSI1)
  - Verifies that it does what it is supposed to
- **So far independent of technology**
  - Synthesis maps RTL description onto a library of cells in a technology
  - Output is mostly a Verilog netlist
- **The rest of the design flow uses this**

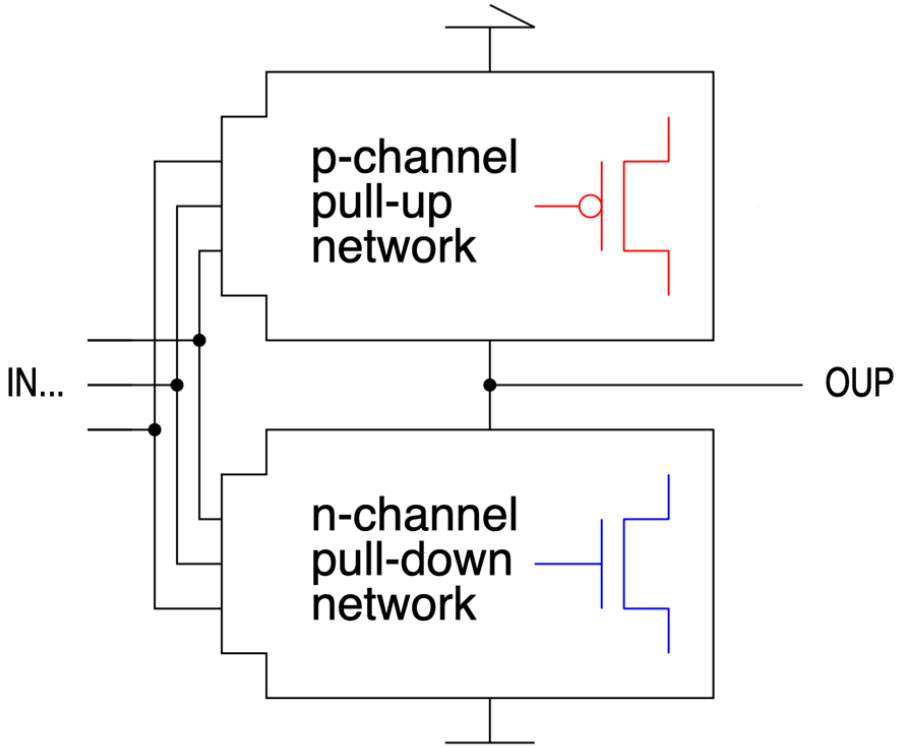


CMOS digital design is key to modern IC design

- **Boolean algebra allows us to map any functionality to 1s and 0s**
  - Establishes rules to transform and optimize (key job of synthesis)
- **Set of simple combinational gates and sequential elements needed**
  - Basically with a 2-input NAND/NOR and a Flip-flop you can do anything
- **Digital CMOS gives us an efficient way of implementing simple gates**
  - Use transistor as a switch
    - When ON current flows
    - When OFF no current flows (well a tiny leakage current flows, but more on that later)
  - Combine nMOS and pMOS for efficiency
    - At a given time either nMOS pulls down or pMOS pulls up
  - Relatively simple to manufacture, easy to define electrical parameters

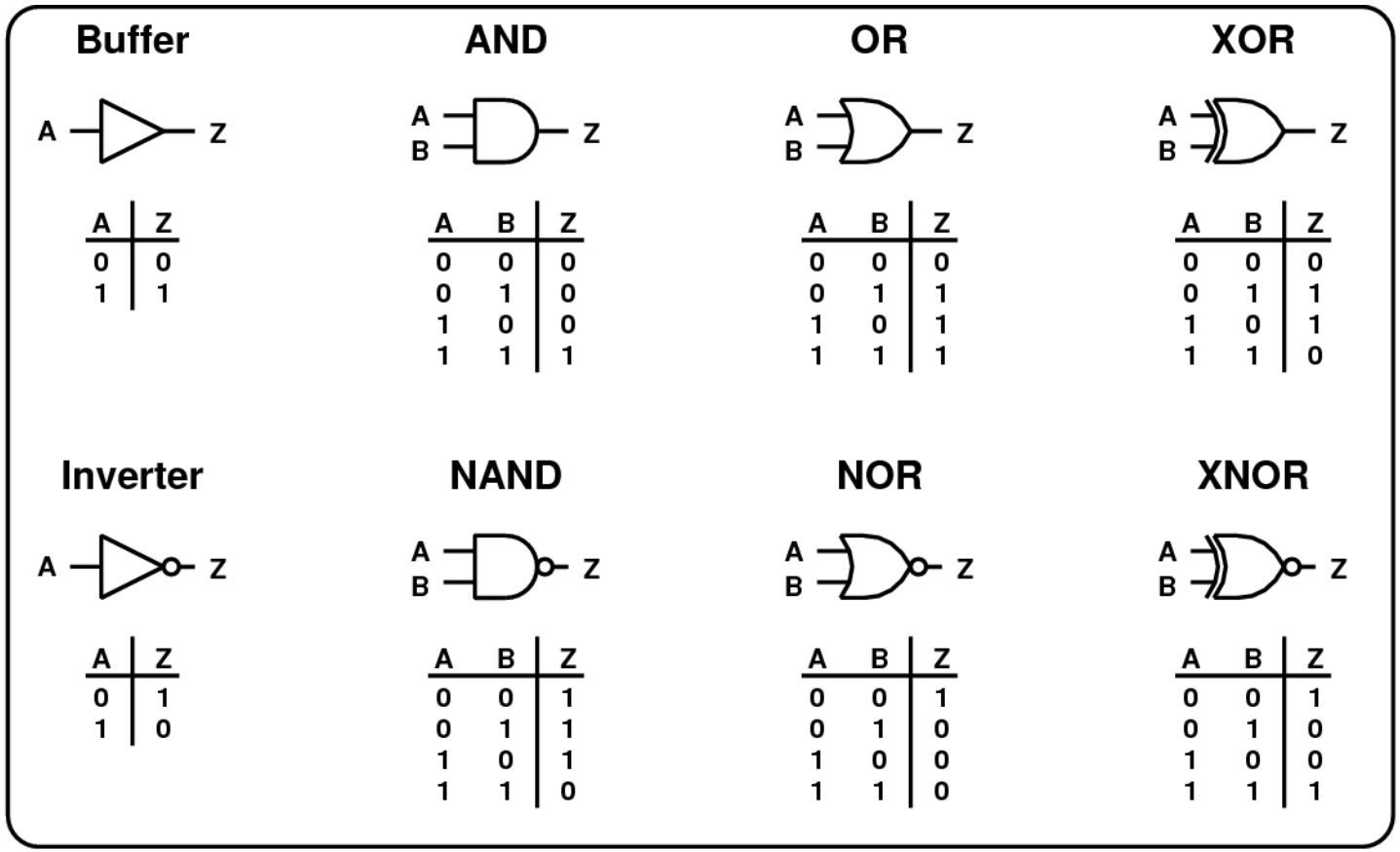
You can construct any Boolean logic function in CMOS

- **You can construct the nMOS pull-down network**
  - AND is a **series** connection
  - OR is a **parallel** connection
- **The pMOS pull-up network will then be constructed in a complementary fashion**
  - What is **parallel** in nMOS will be in **series** in pMOS
  - What is in **series** in nMOS will be **parallel** in pMOS

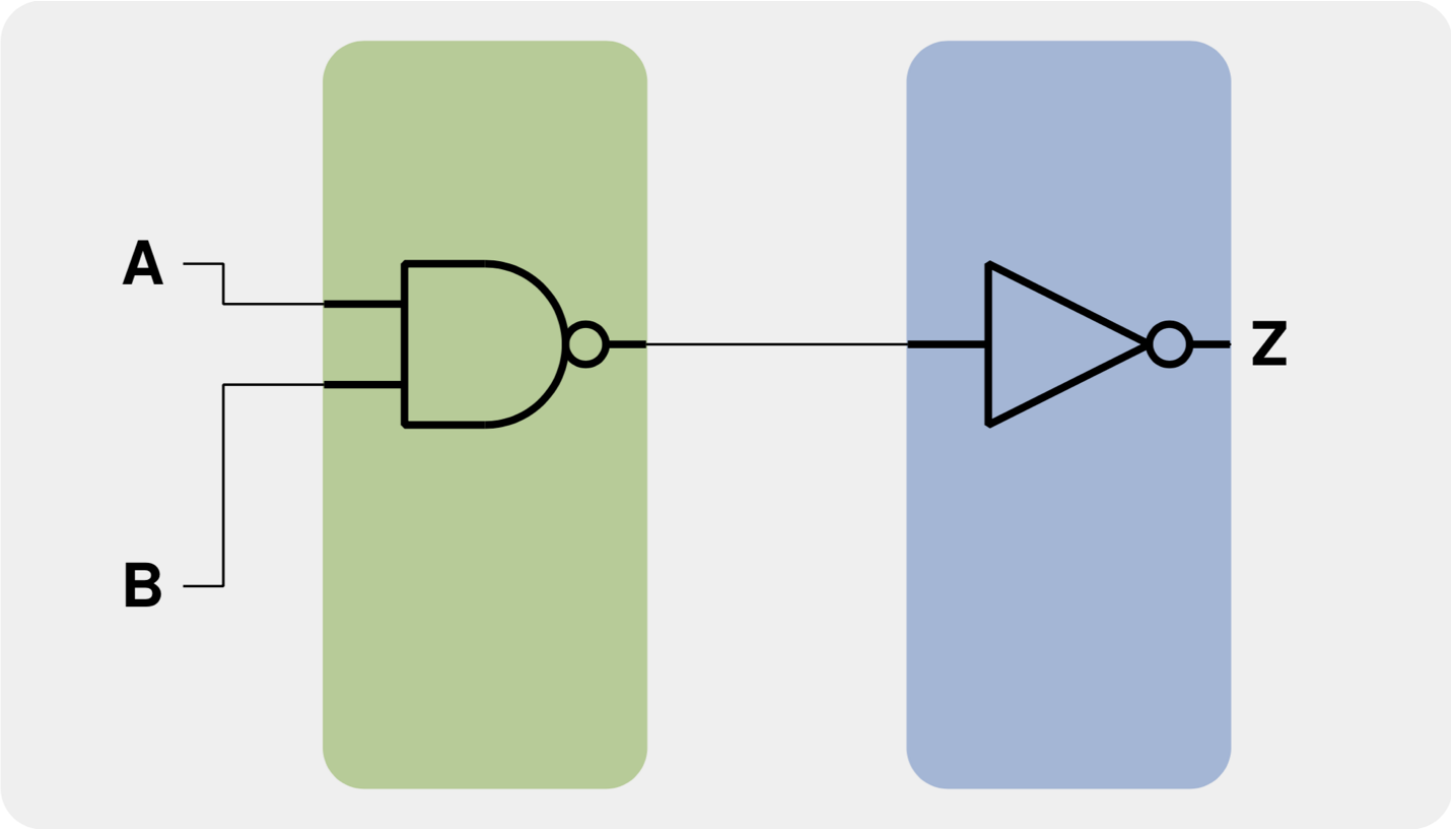


Slide adapted from Hubert Kaeslin, "Top-Down Digital VLSI Design Vol2: From Gate-Level Circuits to CMOS Fabrication"

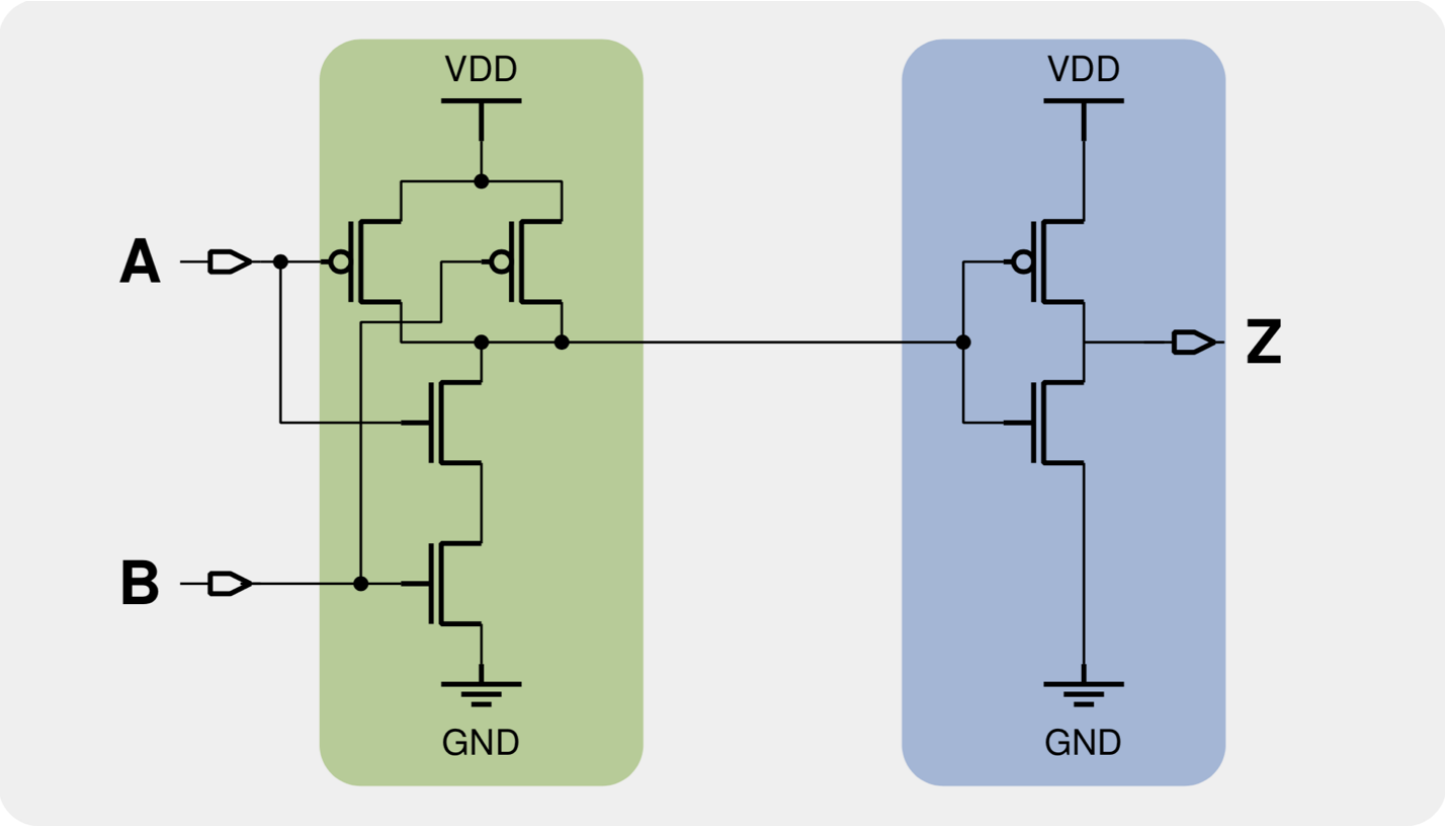
Most common logic cells



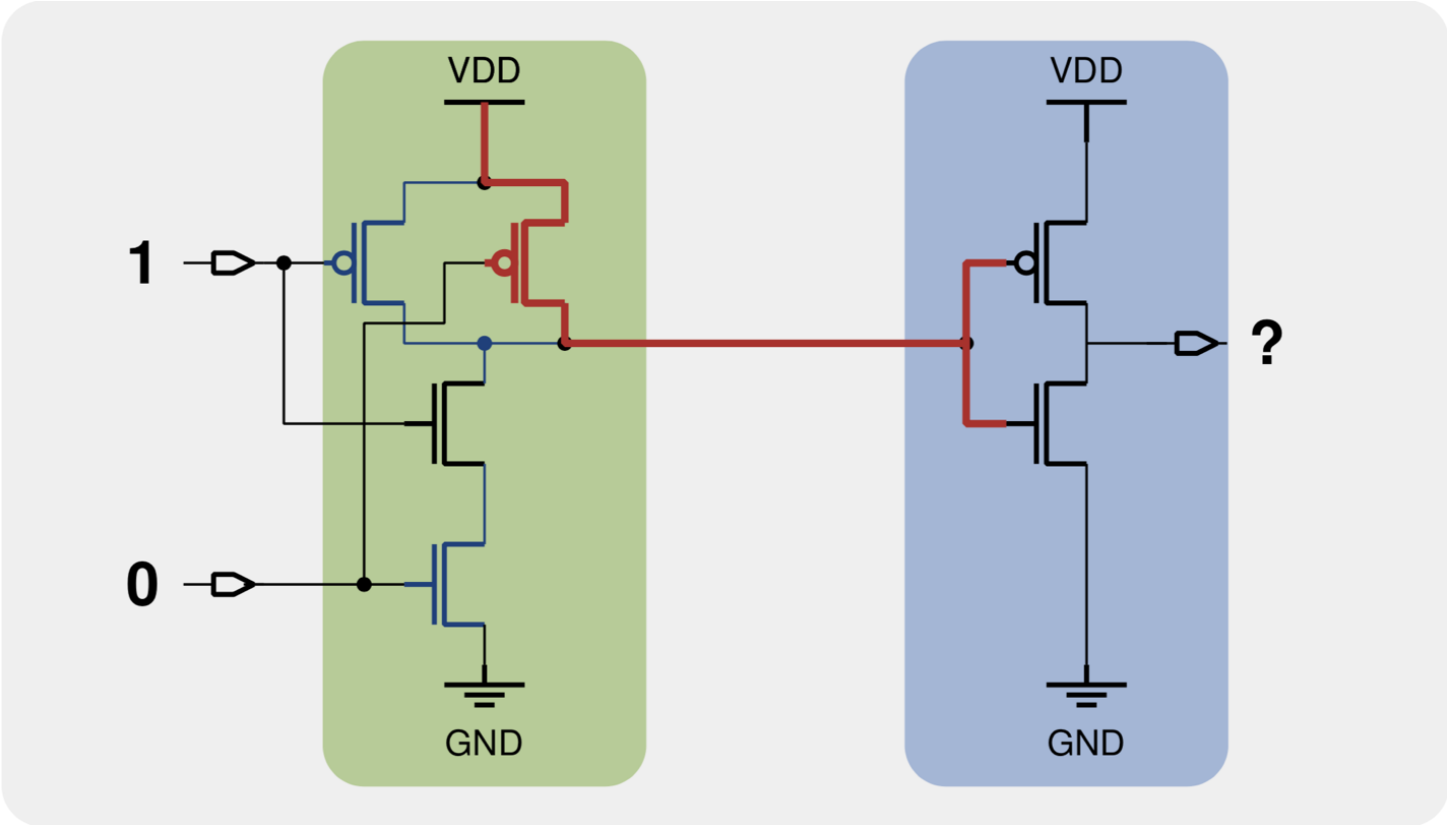
Timing of CMOS gates simplified



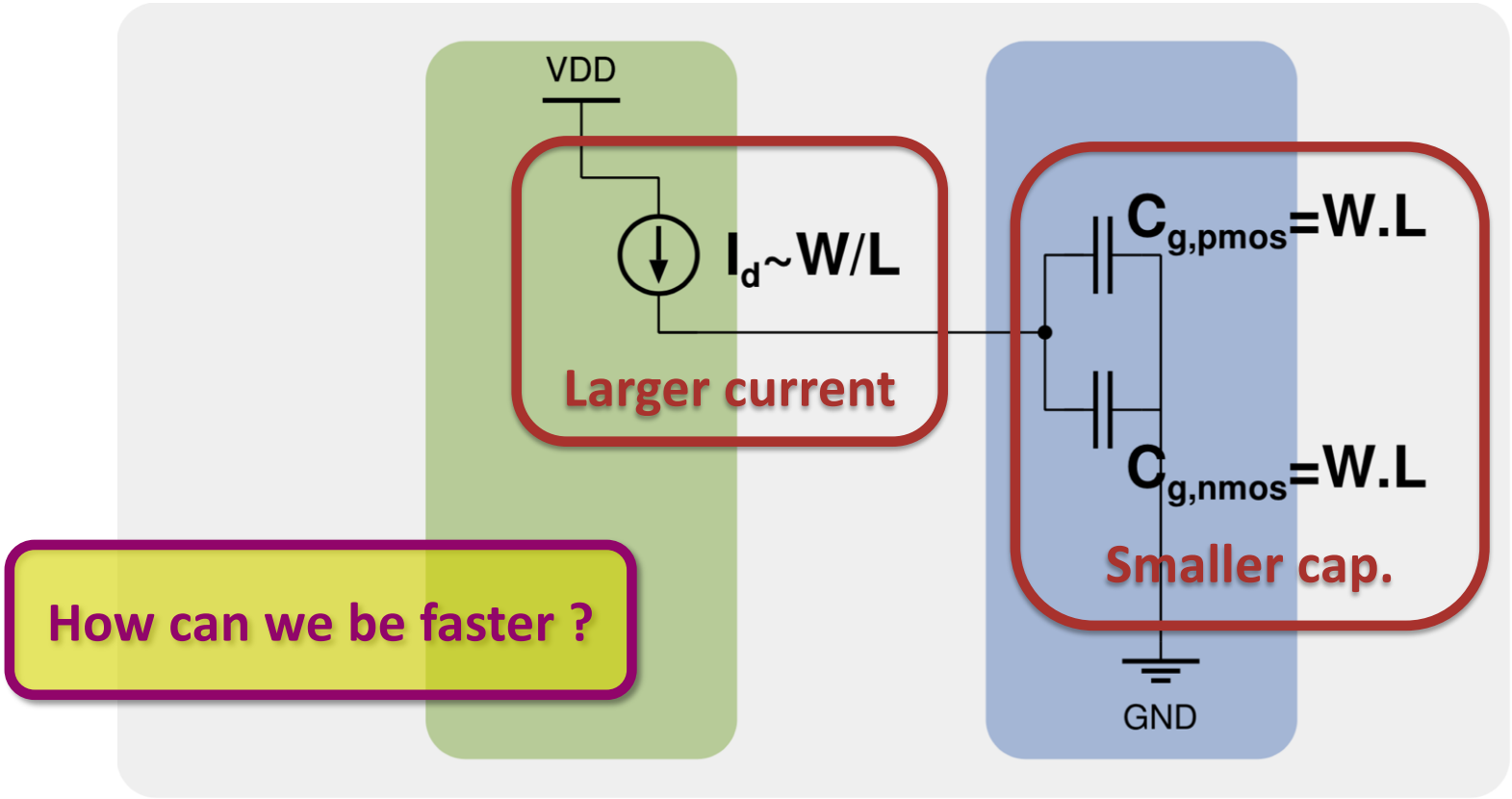
These connected gates at transistor level



Assume we are charging the output



Simplified model of charging the output to high voltage



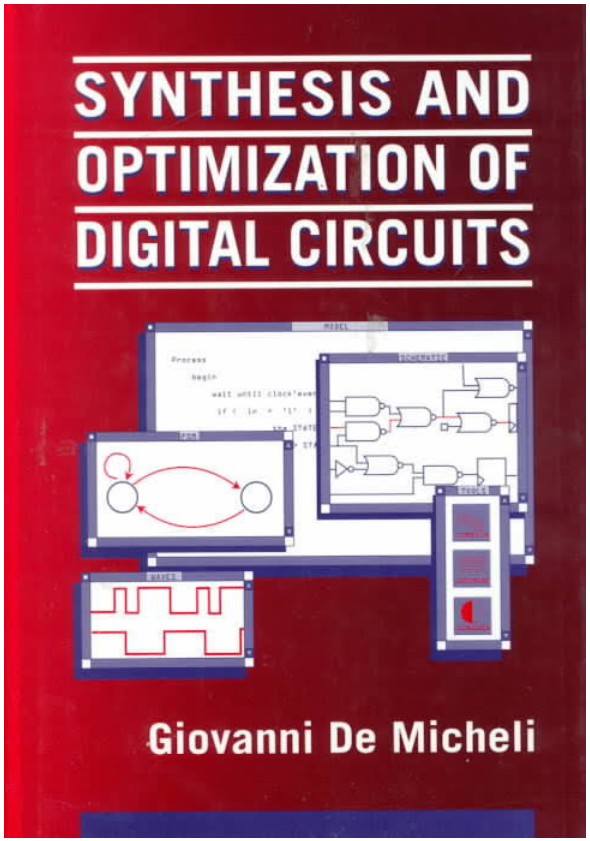
Timing of CMOS gates is to find a balance

- **Larger transistors can speed things up**
  - More current can switch the output faster
  - Trading off area/power against speed
- **But you pay the price with increased input capacitance**
  - The previous gate now has a larger problem
- **Parasitic effects add to our problem**
  - In modern technologies, wire capacitance dominates over gate capacitance
- **Goal: Finding a good balance between drive and input capacitance**
  - Libraries include the same function with several DRIVE variations
  - Libraries include buffers and inverters to support long connections
  - EDA tools can work out which combinations will be best to realize a function

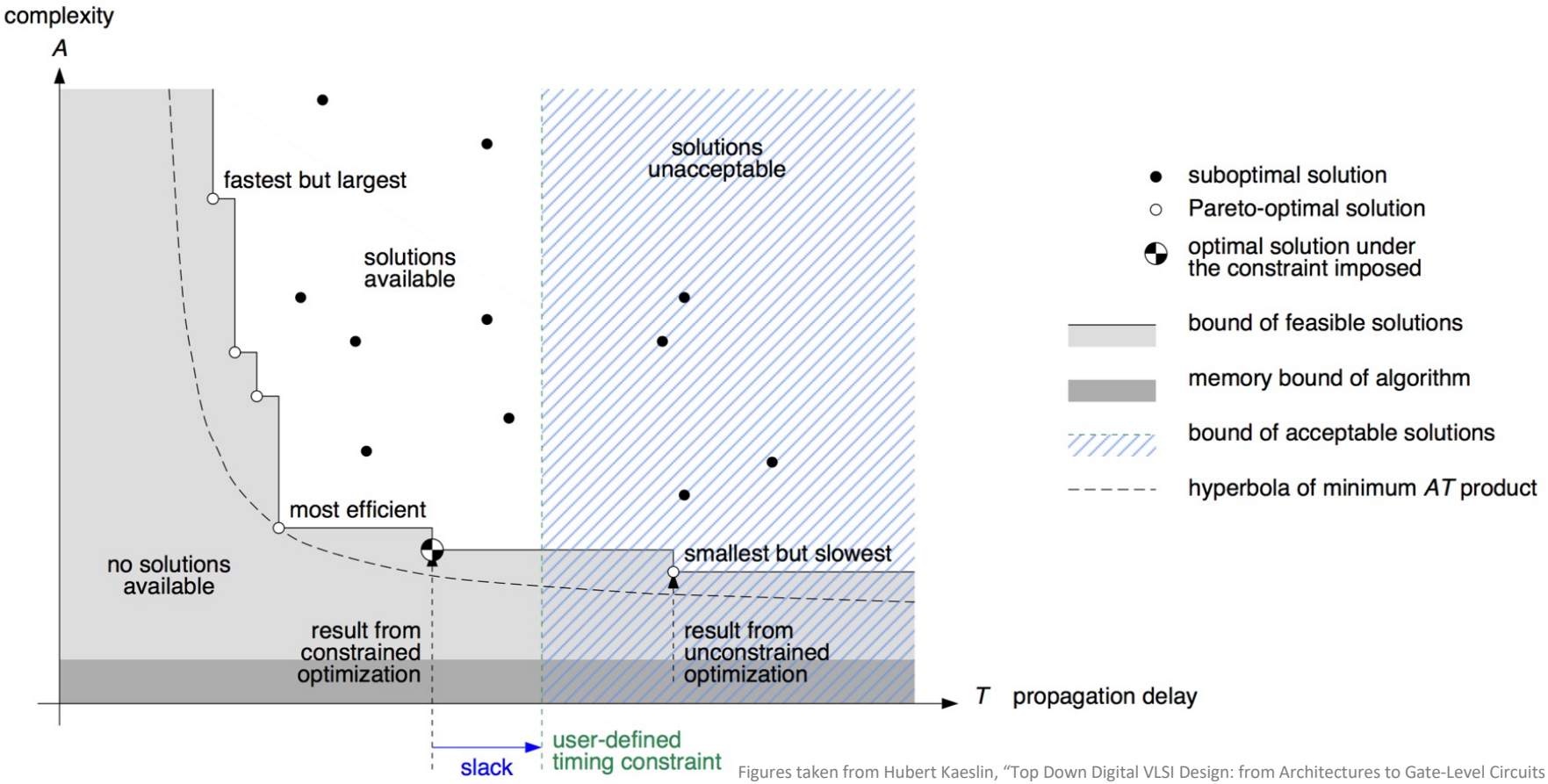
Moving from HDL to Synthesis

- Logic synthesis algorithms can map the HDL description into a library of standard cells using Boolean Algebra resulting in a **gate-level netlist**.
  - The tool can help optimize speed, area, power
  - It just needs to know what is more important
  - **Constraints** are used to explain our wishes
- This netlist is what we use for next steps

- Giovanni De Micheli (Prof. @EPFL) wrote one of the first books on logic synthesis.

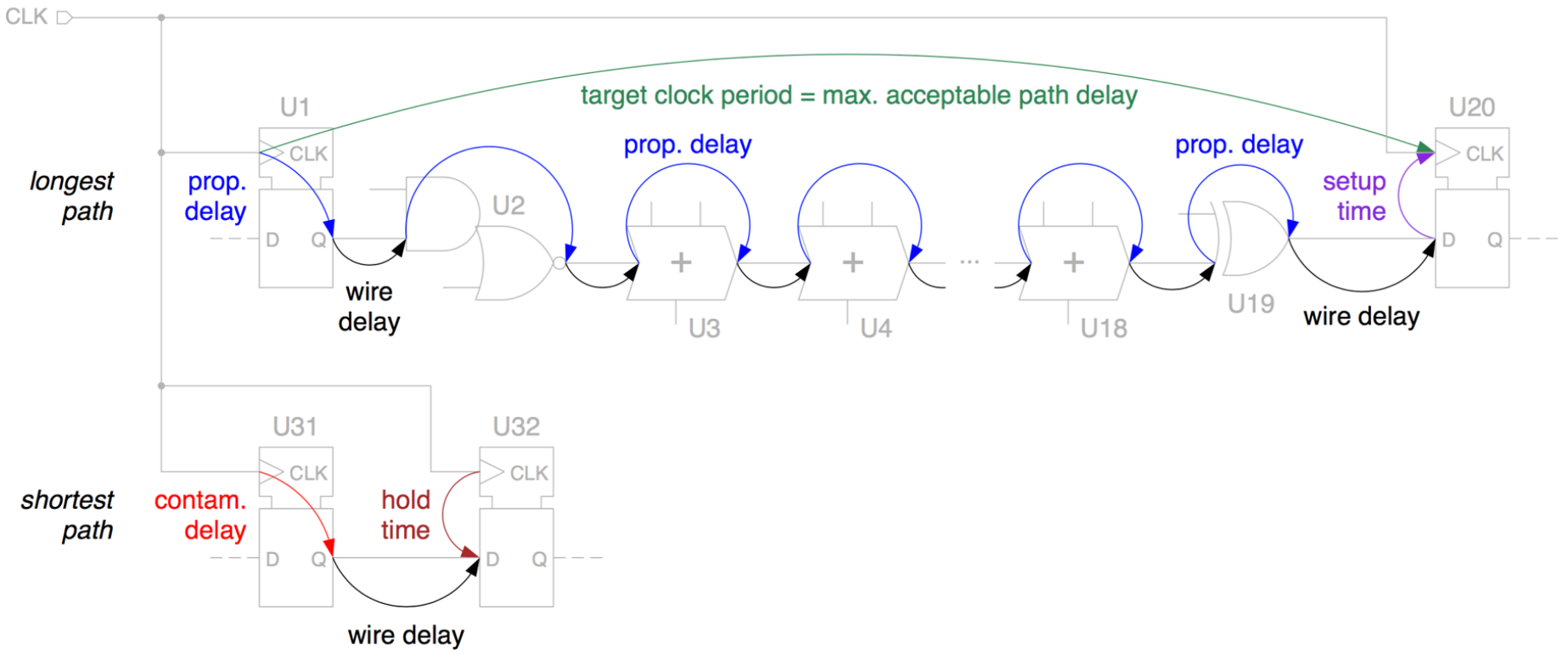


Optimization goals during synthesis



Figures taken from Hubert Kaeslin, "Top Down Digital VLSI Design: from Architectures to Gate-Level Circuits and FPGAs"

Propagation/contamination delay, hold/setup time



Figures taken from Hubert Kaeslin, "Top Down Digital VLSI Design: from Architectures to Gate-Level Circuits and FPGAs"

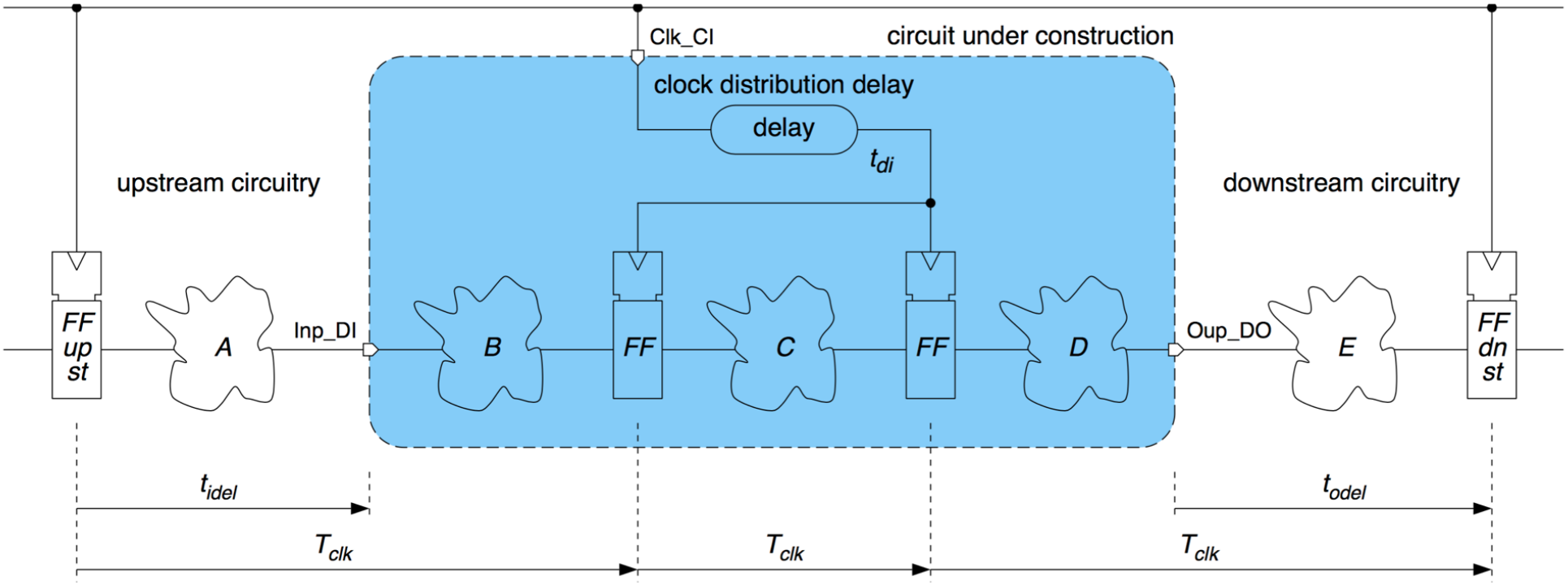
Telling what you want == Constraints

- **Timing determines many of the properties (indirectly)**
  - A fast circuit will be larger
  - A large circuit will consume more power
  - Therefore timing is the main constraint to tell what kind of circuit you want
- **The clock period determines all internal timing**
  - We will only consider circuits with single clock domains where this is simple
  - Larger real circuits are a bit more involved.
- **What about the boundary conditions, input and outputs of your circuit**
  - We need to know about the environment
  - In industrial designs, these are the most important constraints
    - It defines how your circuit will work together with the circuits of your customer

Input/Output delays: Boundary conditions of IC design

- **Once you specify the clock period, all timing inside circuit is known**
  - We can calculate all setup/hold, propagation/contamination delays
  - If necessary optimize the circuit
- **Problem is what happens at the boundary of your circuit**
  - Compared to a reference clock, when do the inputs arrive latest/earliest
  - What is the earliest/latest time the outputs will appear every cycle
  - We use timing constraints for input and inputs to complete the timing description
- **Specified by stating the delay of the environment**
  - Input delay of 1ns means that, inputs will come 1ns **after** the clock edge
  - Output delay of 2ns means that, the outputs have to appear 2ns **before** the next clock edge

Input and output timing seen from the circuit



Figures taken from Hubert Kaeslin, "Top Down Digital VLSI Design: from Architectures to Gate-Level Circuits and FPGAs"

Netlist

Structured Verilog code instantiating standard cells

HDL can be used for many different things (Yesterday)

- **Behavioral HDL code**
  - Common for describing hardware
  - Explaining the synthesizer how our architecture looks like
- **Structural HDL code**
  - Describes modules and interconnections
  - A 1:1 translation of a block diagram
- **Gate-level netlist**
  - A subset of structural HDL where at lowest level the modules are standard cells
  - Since only very basic functionality is used, netlists are usually simple Verilog
  - Interfaces, connections are all simple logic types

Some small notes about netlists

- **Generated (predominantly) automatically**
  - (Usually) one large file
  - Meant to be processed through tools, no manual edits foreseen
- **Uses only Verilog syntax (SystemVerilog additions not needed)**
  - **NO** `always`, `always_ff`, `always_comb`, `always_latch` statements
  - **NO** `logic`, `int` types used, typically only `wire`
  - Can make use of hierarchy (multiple `module` defintions) or be flat (single `module`)
- **Sometimes there will be an additional power netlist as well**
  - Includes connections for power nets (VDD, GND, substrate)
  - Used for LVS (Exercise 11)
  - Simulation with multiple power domains

Anatomy of a netlist

```
module croc_chip (
        clk_i,
// […]
   wire net5157;
   wire i_croc_soc_i_croc__0389_;
   wire i_croc_soc_i_croc__0390_;
// […]
 sg13g2_a21oi_1 i_croc_soc_i_croc__3715_ (.Y(i_croc_soc_i_croc__0117_),
        .A1(i_croc_soc_i_croc__0107_),
        .A2(i_croc_soc_i_croc__0112_),
        .B1(i_croc_soc_i_croc__0116_));
   sg13g2_buf_4 fanout5182 (.X(net5182),
        .A(net5185));
// [ … ]
endmodule
```

Croc from exercises

Single module

43’191 wire definitions

40’091 standard cells

182’348 lines

Some tips for browsing netlists

- **Do not open a file in an editor, if you do not plan to edit it!**
  - Use less
    ```
    less -S netlist.v
    ```
- **Navigation in `less` is simple (i.e. find the last module defined)**
  - **SHIFT+G** (go to end)
  - **`?module`** (search backwards for module)
- **Learn how to use basic UNIX commands to help you**
  - `grep wire netlist.v | wc -l`
  - `awk '/sg13g2/{print $1}' netlist.v | sort -u`
- **You will notice most 'older' UNIX tools have similar parameters**

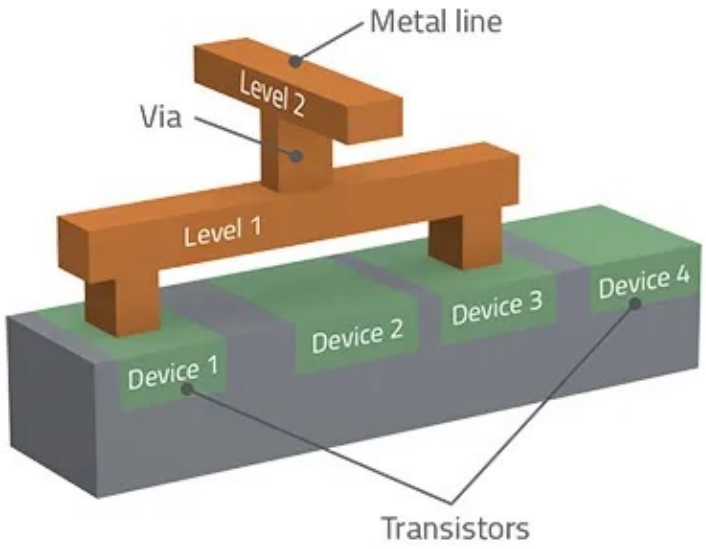
Routing layers

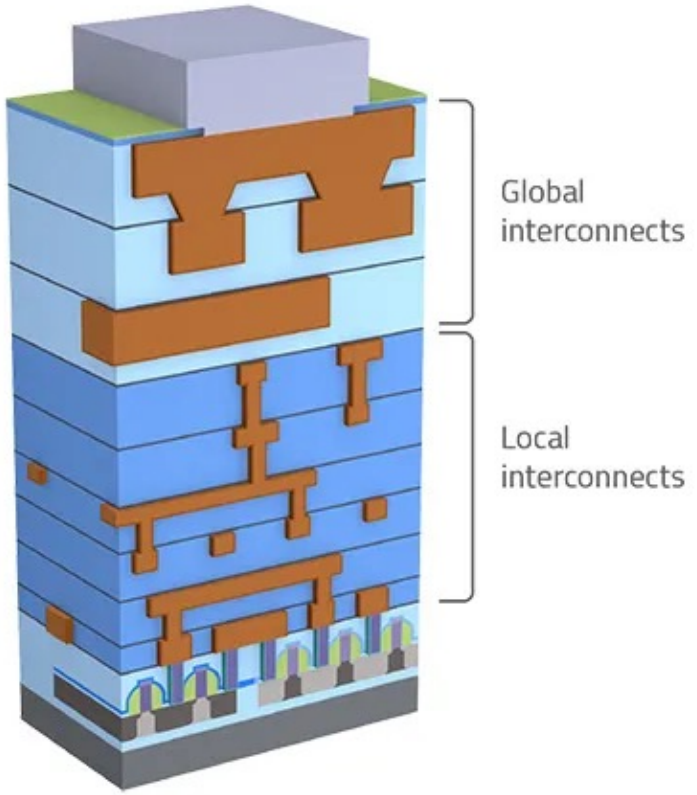
Front-end and back-end for IC Design

- **Back-end**
  - The interconnect between transistors
  - Metal lines
  - Vias/contacts
  - Connection to outside
- **Our interest in VLSI2**
- **Front-end**
  - The transistors
  - Not part of this lecture

Slide adapted from Hubert Kaeslin, "Top-Down Digital VLSI Design Vol2: From Gate-Level Circuits to CMOS Fabrication"

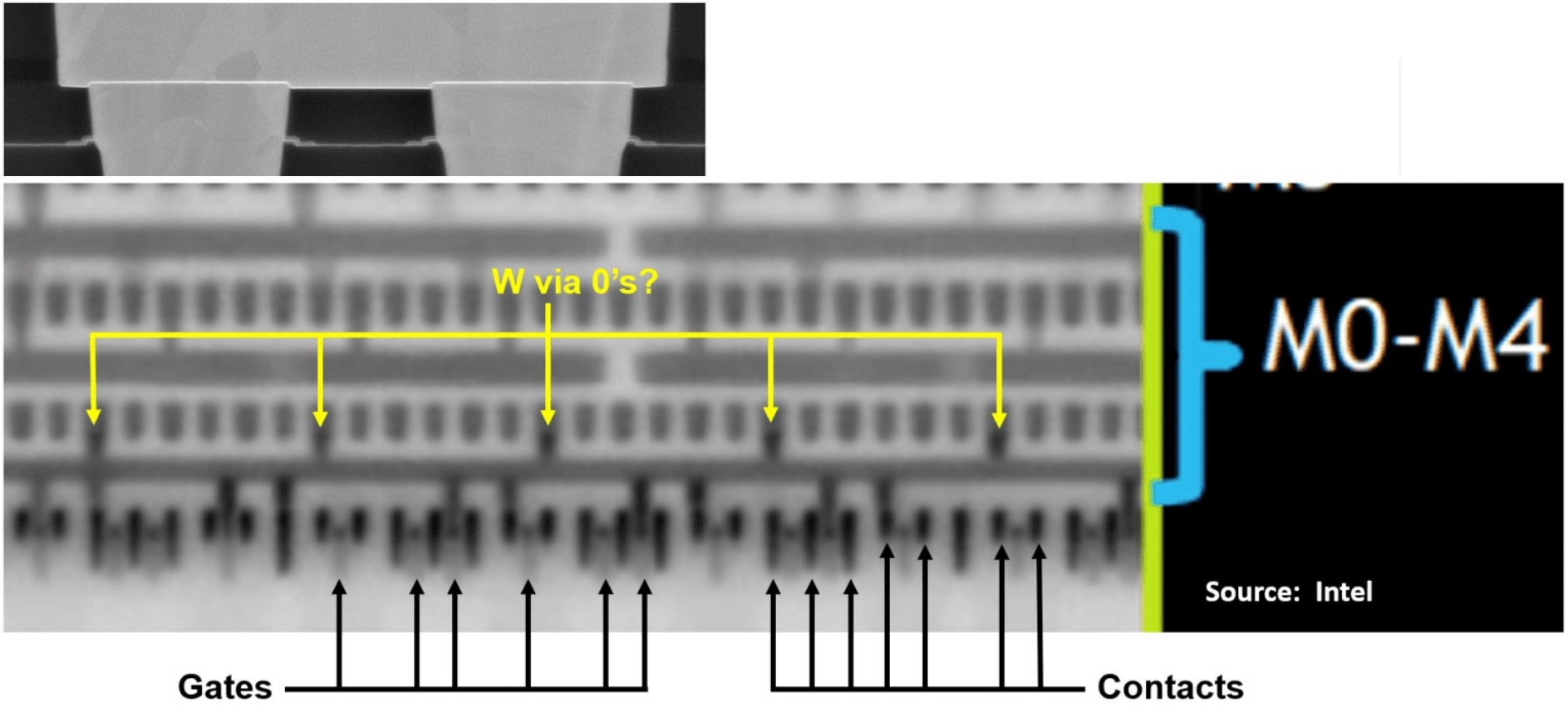
Interconnects in IC design





https://semiengineering.com/all-about-interconnects/

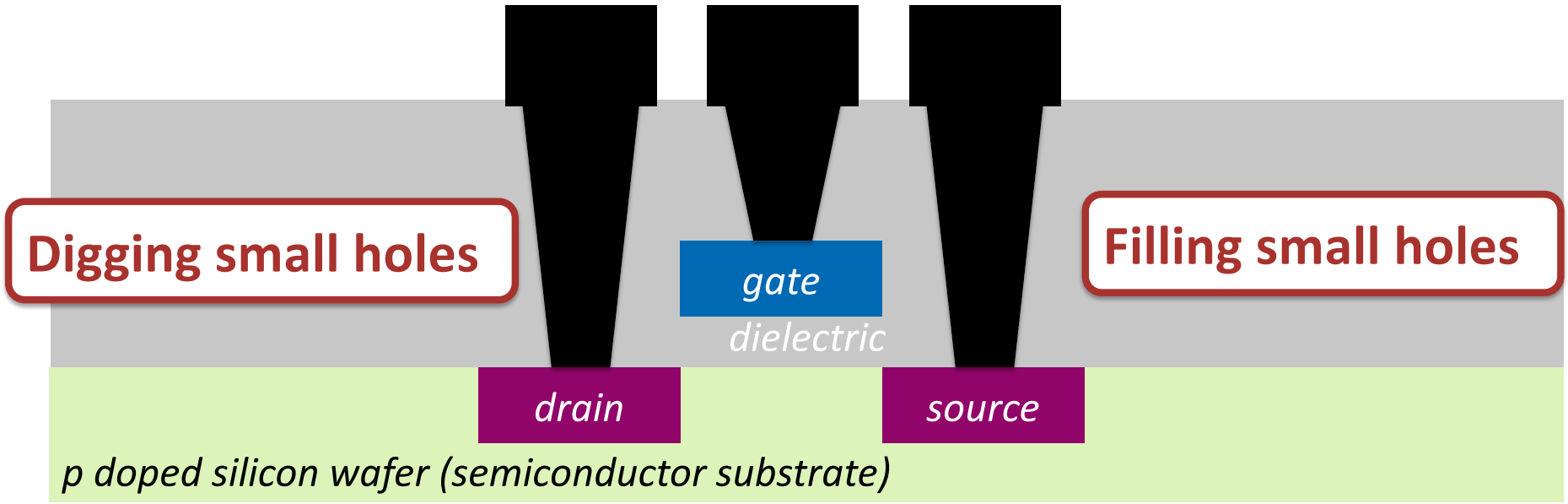
A look into state of the art metal stack (Intel 4)



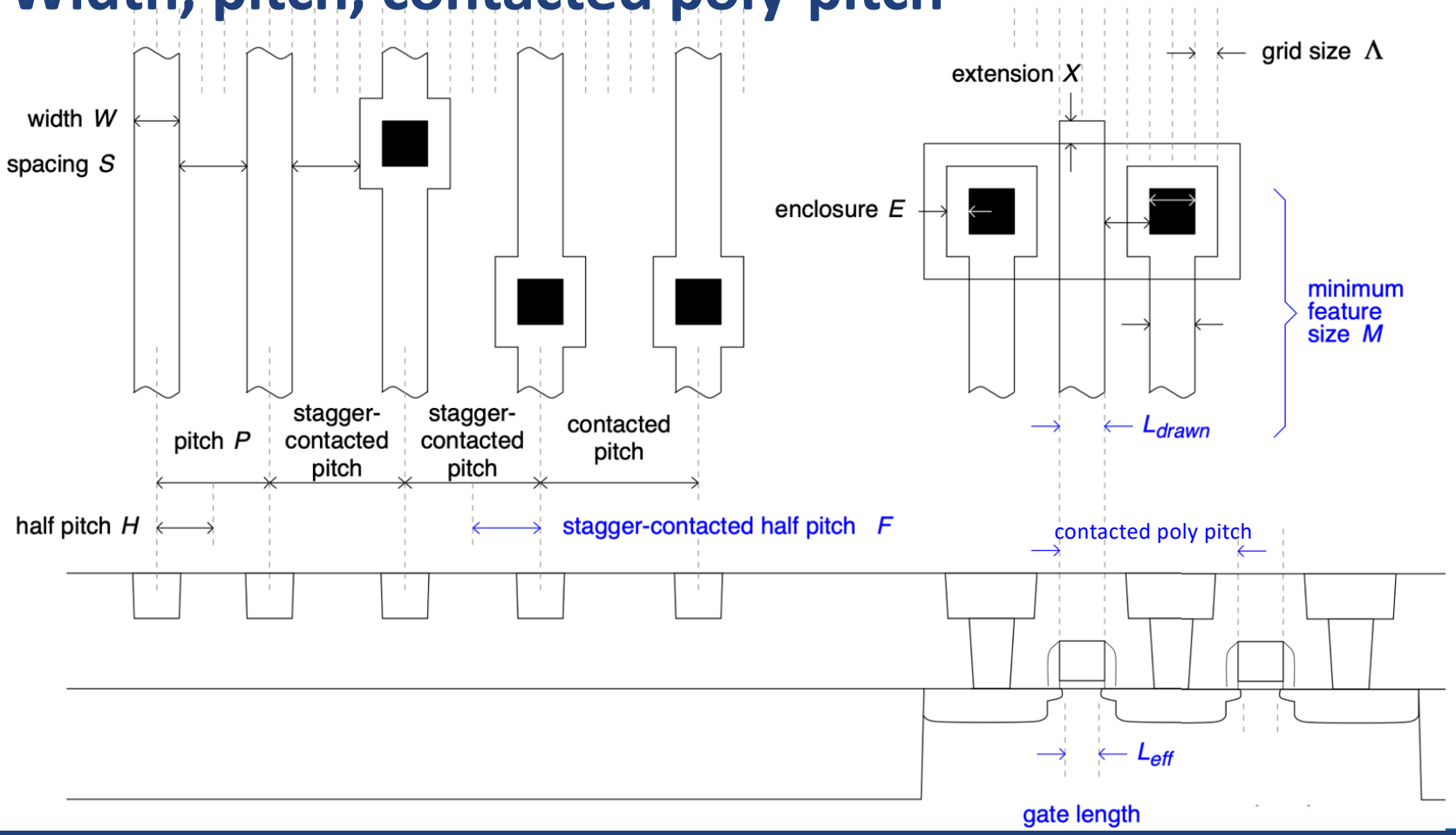
https://www.semiconductor-digest.com/intel-4-process-drops-cobalt-interconnect-goes-with-tried-and-tested-copper-with-cobalt-liner-cap/

Issues in manufacturing

Patterning dense structures



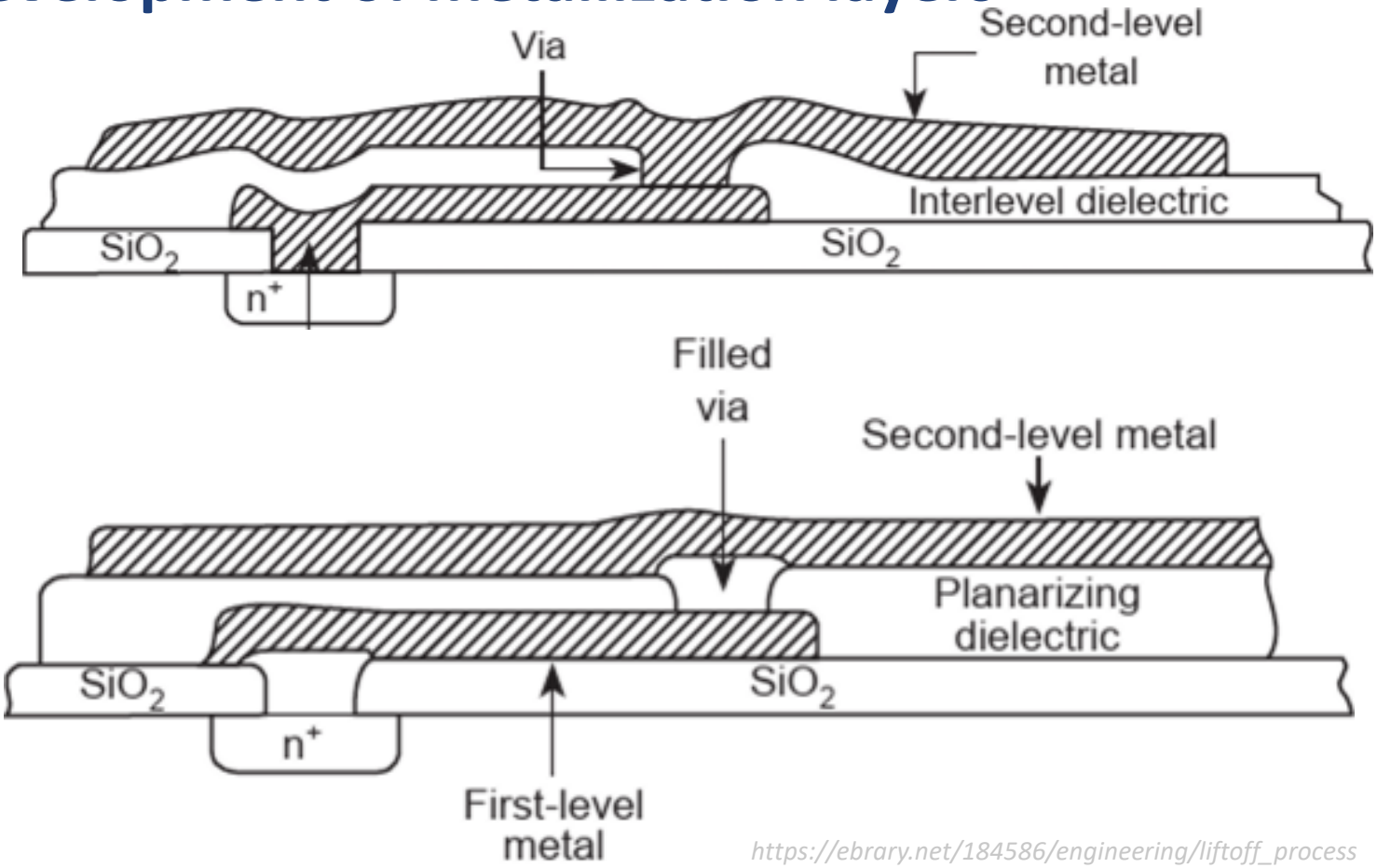
Width, pitch, contacted poly-pitch



All wires have to come down to transistors

- **It is the transistors that realize the logic functions**
  - We use the wires to connect drains, gates and sources of the transistors
- **We started with modest number of interconnect capability**
  - The polysilicon that forms the gate was also used for short interconnect
    - It is not the best conductor, so only for short distances
  - Metal lines were used to connect drains and sources
- **Early technologies had 1 (or 2) polysilicon and 2-3 metal layers**
  - Every layer adds complexity to the process
  - Surface gets bumpy, aligning the next lithography step gets harder
  - Adding vias between layers is cumbersome

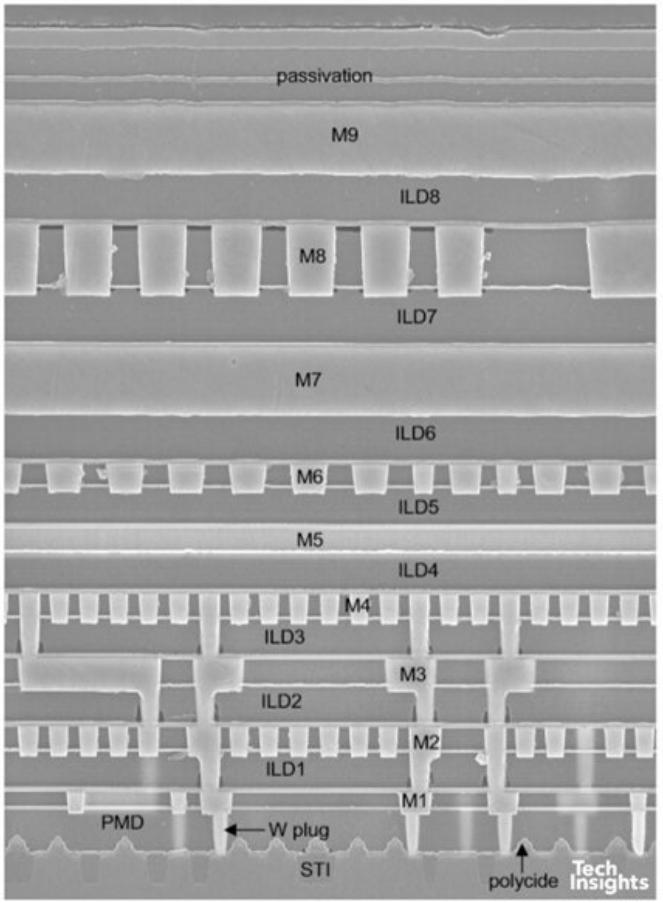
Development of metallization layers



https://ebrary.net/184586/engineering/liftoff_process

Planarization helped us to get more metal layers

- **Jumped quickly from 2-3 layers to 5-6 then to 8-13 layers**
  - As usual, the first change brought the biggest impact
  - But it gets crowded as you go down close to the transistors
- **Current use is more specialized**
  - **First few layers**: cells, macros
  - **Lower layers**: inter-block routing
  - **Middle layers**: routing between blocks
  - **Upper layers**: Long distance connections, power, clock
- **Upper layers usually 'thicker'**

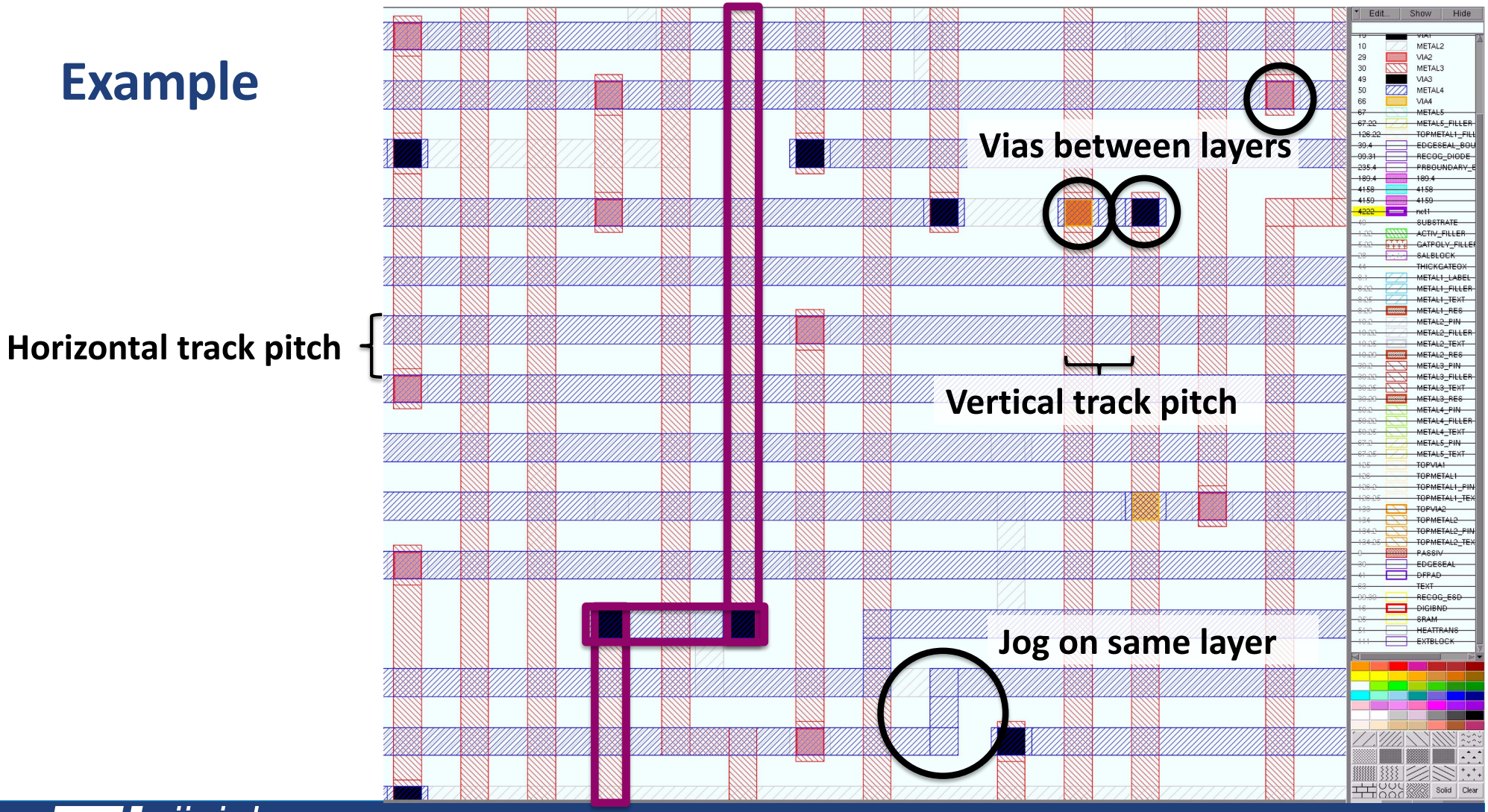


https://www.techinsights.com/blog/trip-down-tsmc-memory-lane-part-2

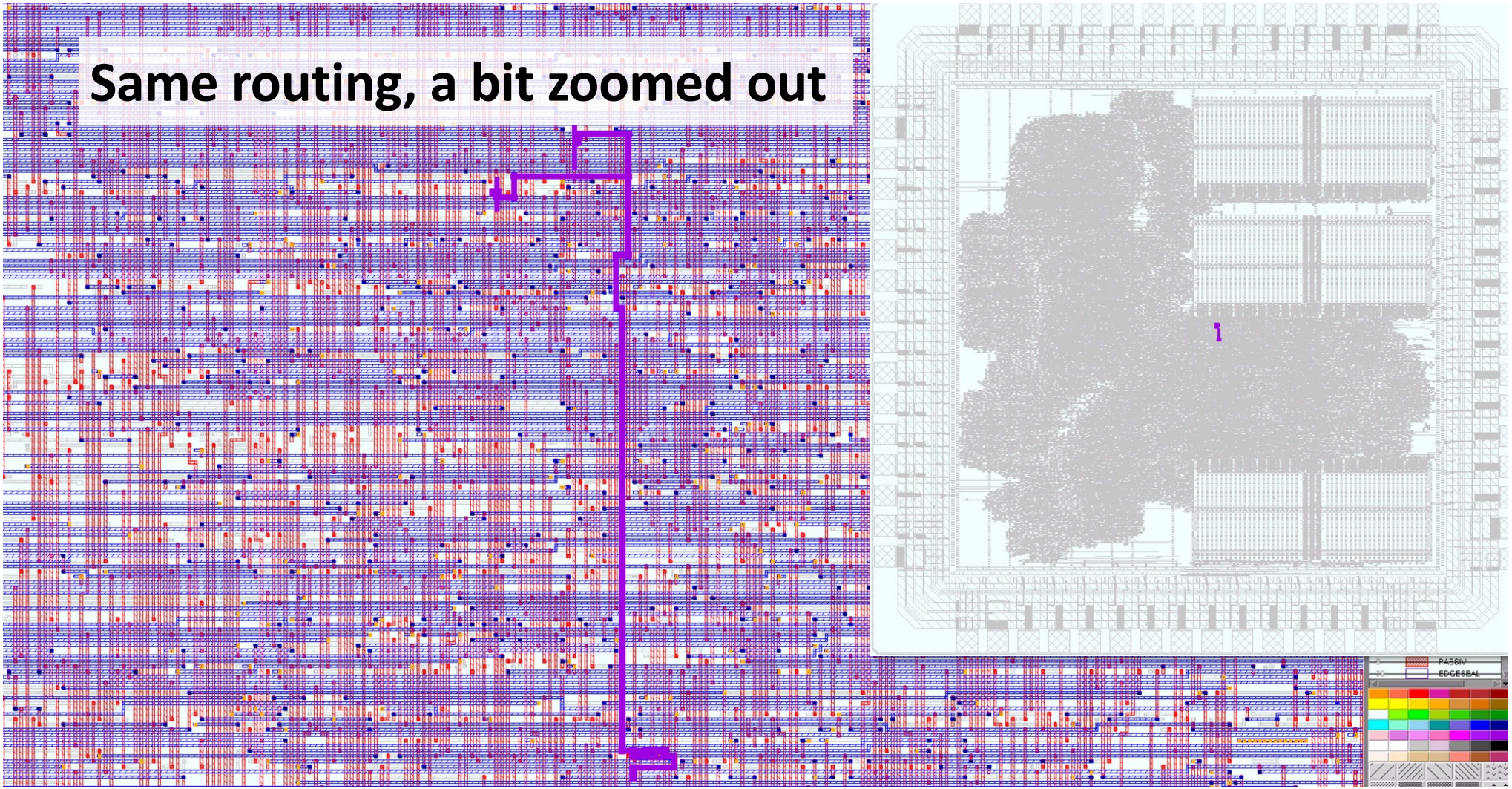
Manhattan routing

- **Comes from the grid like map of Manhattan Island in NYC**
  - Every layer has a preferred direction (horizontal / vertical)
  - To go from Point A to Point B
    - Travel horizontal on one layer
    - Use a via to the next layer (up or down)
    - Travel vertical on the next layer
- **Greatly Simplifies routing**
- **The preferred direction is not technology specific**
  - Current standard cells in IHP use M1/M2 horizontal, M3 vertical
  - EZlibrary for IHP designed in VLSI5 uses M1 horizontal, M2 vertical
- **Possible to allow 'jogs' short changes in direction**
  - Parameter of routing tool

Example

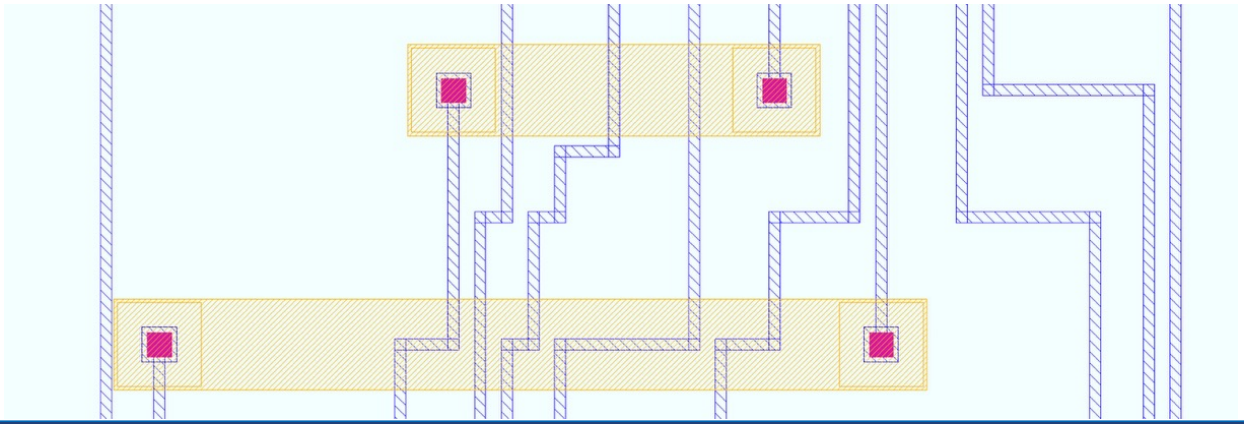

Horizontal track pitch
Vias between layers
Vertical track pitch
Jog on same layer

Same routing, a bit zoomed out

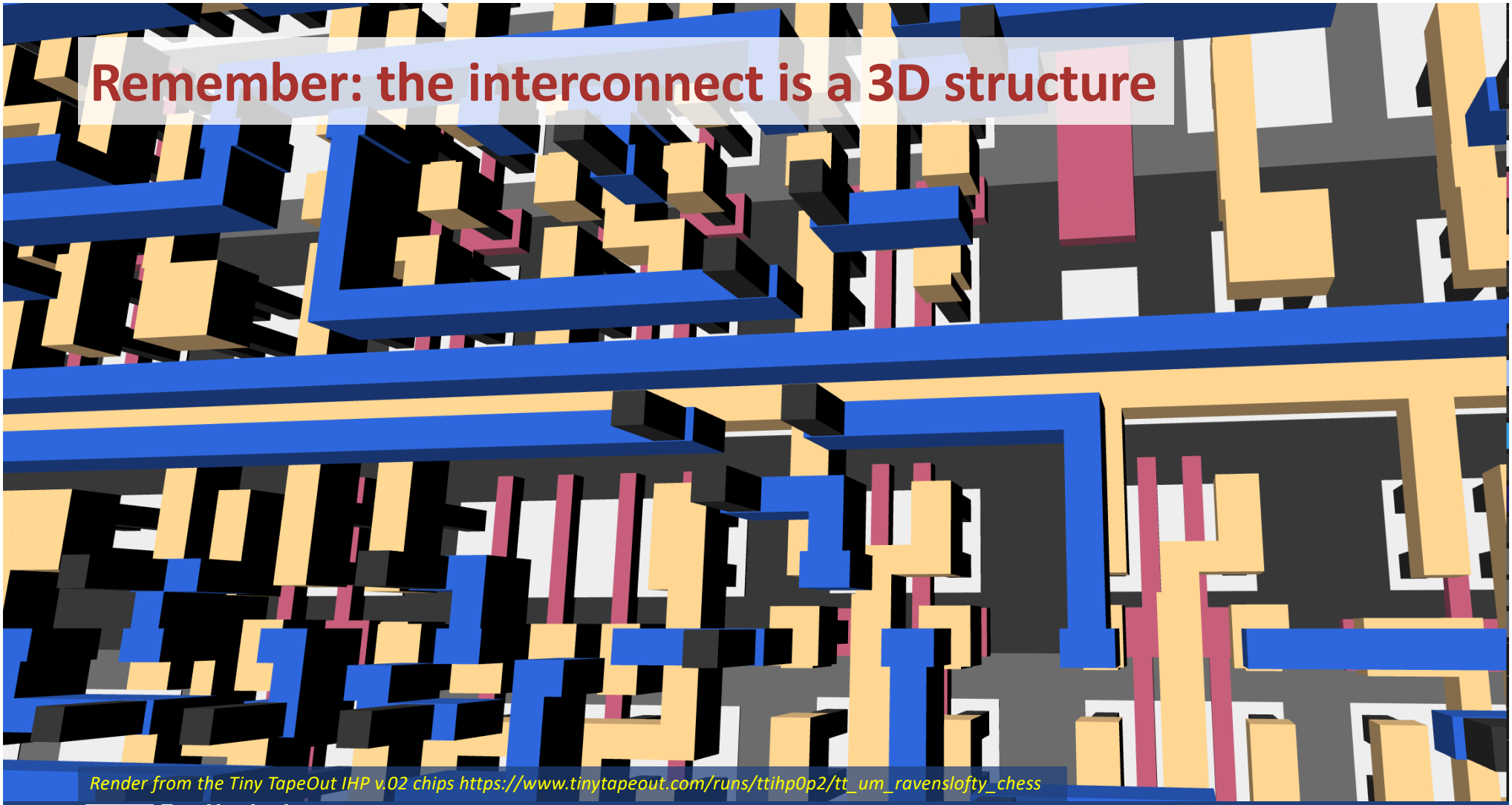


Track pitch is used commonly for back-end design

- **Designates how close two adjacent metal lines can run**
  - Needs to also account for placing vias
  - May end up being a bit more relaxed than what the technology limits are
- **Usually upper layers are thicker → have a wider pitch**
  - In most cases the upper metal pitch is a direct multiple of lower pitch
  - So that signals from upper layers can come down easily



Remember: the interconnect is a 3D structure

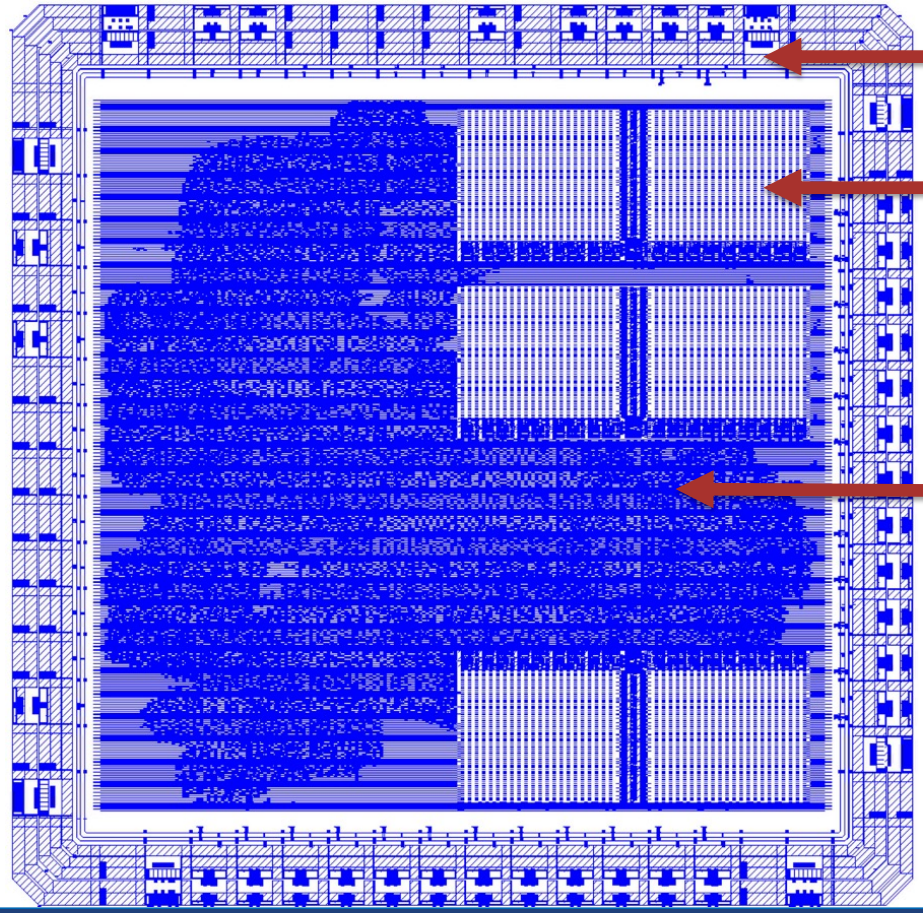


Render from the Tiny TapeOut IHP v.02 chips https://www.tinytapeout.com/runs/ttihp0p2/tt_um_ravenslofty_chess

Standard Cells

Pieces that make up digital chips

How do we organize logic cells logic cells?

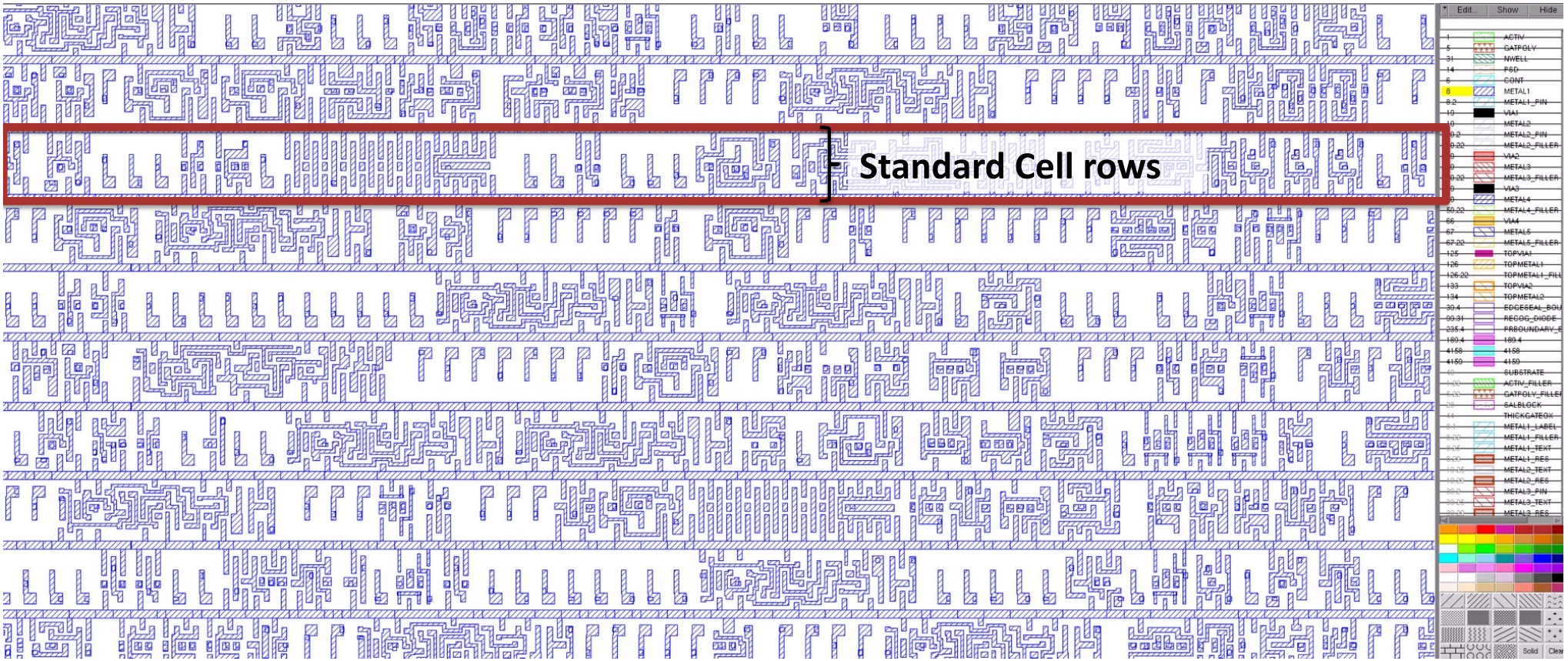


I/O ring (a bit later)

Macro cells (in a moment)

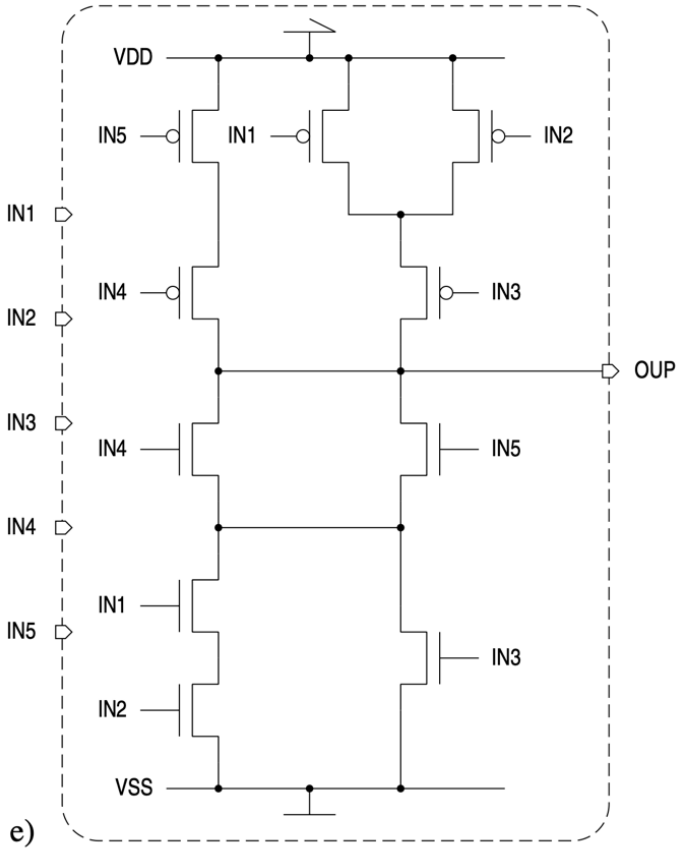
Standard cells

Zooming in for the standard cells organized in rows



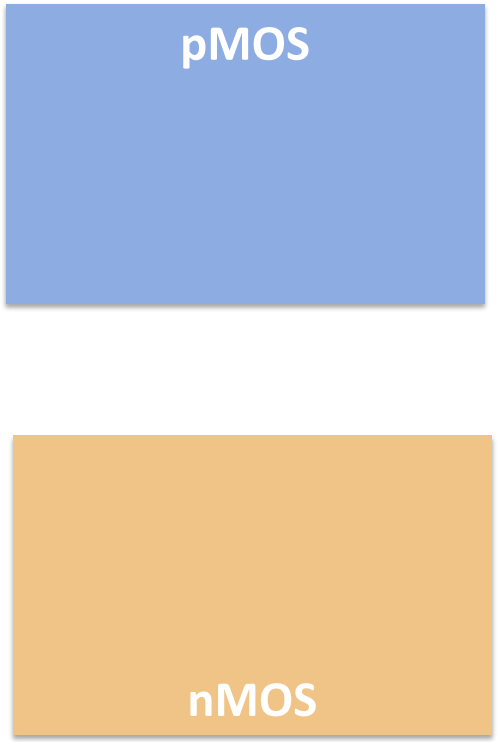
Some observation about CMOS digital gates

- **We have two complementary sides**
  - pMOS for pulling up (towards VDD)
  - nMOS for pulling down (towards GND)
- **Same number of transistors on both sides**
  - True for most, there are some exceptions
- **Power connections**
  - VDD is connected to pMOS
  - GND is connected to nMOS
- **Inputs connect to both pMOS and nMOS**



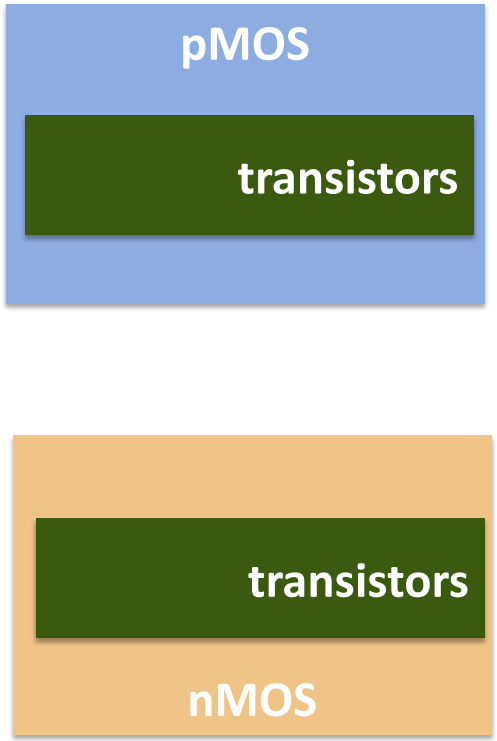
Physical observations on these cells

- **We have two complementary sides**
  - pMOS for pulling up (towards VDD)
  - nMOS for pulling down (towards GND)
- **Same number of transistors on both sides**
  - True for most, there are some exceptions
- **Power connections**
  - VDD is connected to pMOS
  - GND is connected to nMOS
- **Inputs connect to both pMOS and nMOS**



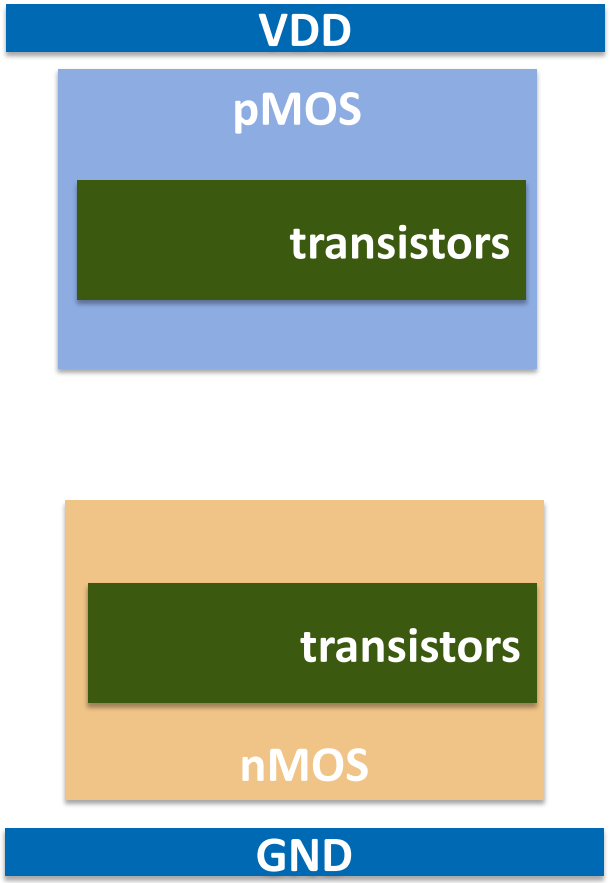
Physical observations

- We have two complementary sides
  - pMOS for pulling up (towards VDD)
  - nMOS for pulling down (towards GND)
- **Same number of transistors on both sides**
  - True for most, there are some exceptions
- Power connections
  - VDD is connected to pMOS
  - GND is connected to nMOS
- Inputs connect to both pMOS and nMOS



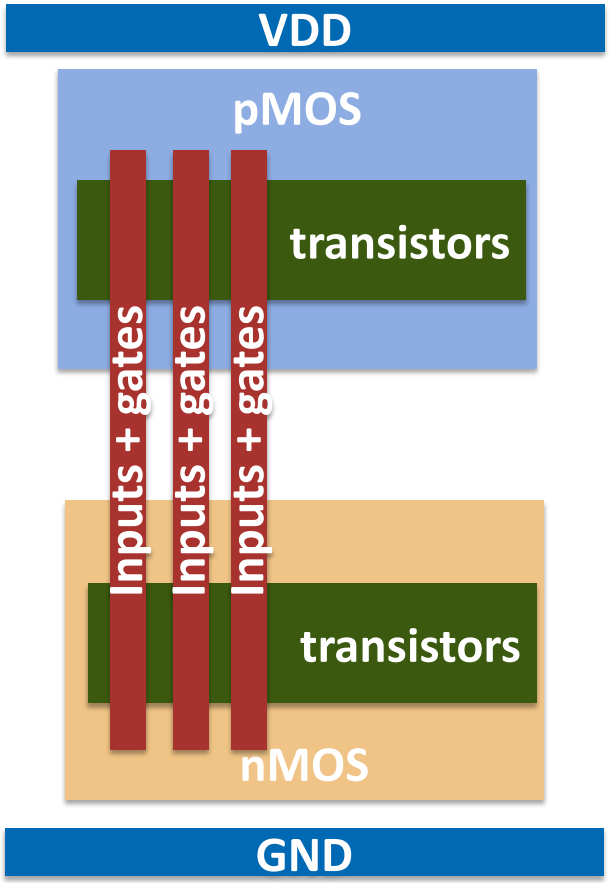
Physical observations

- We have two complementary sides
  - pMOS for pulling up (towards VDD)
  - nMOS for pulling down (towards GND)
- Same number of transistors on both sides
  - True for most, there are some exceptions
- **Power connections**
  - VDD is connected to pMOS
  - GND is connected to nMOS
- Inputs connect to both pMOS and nMOS

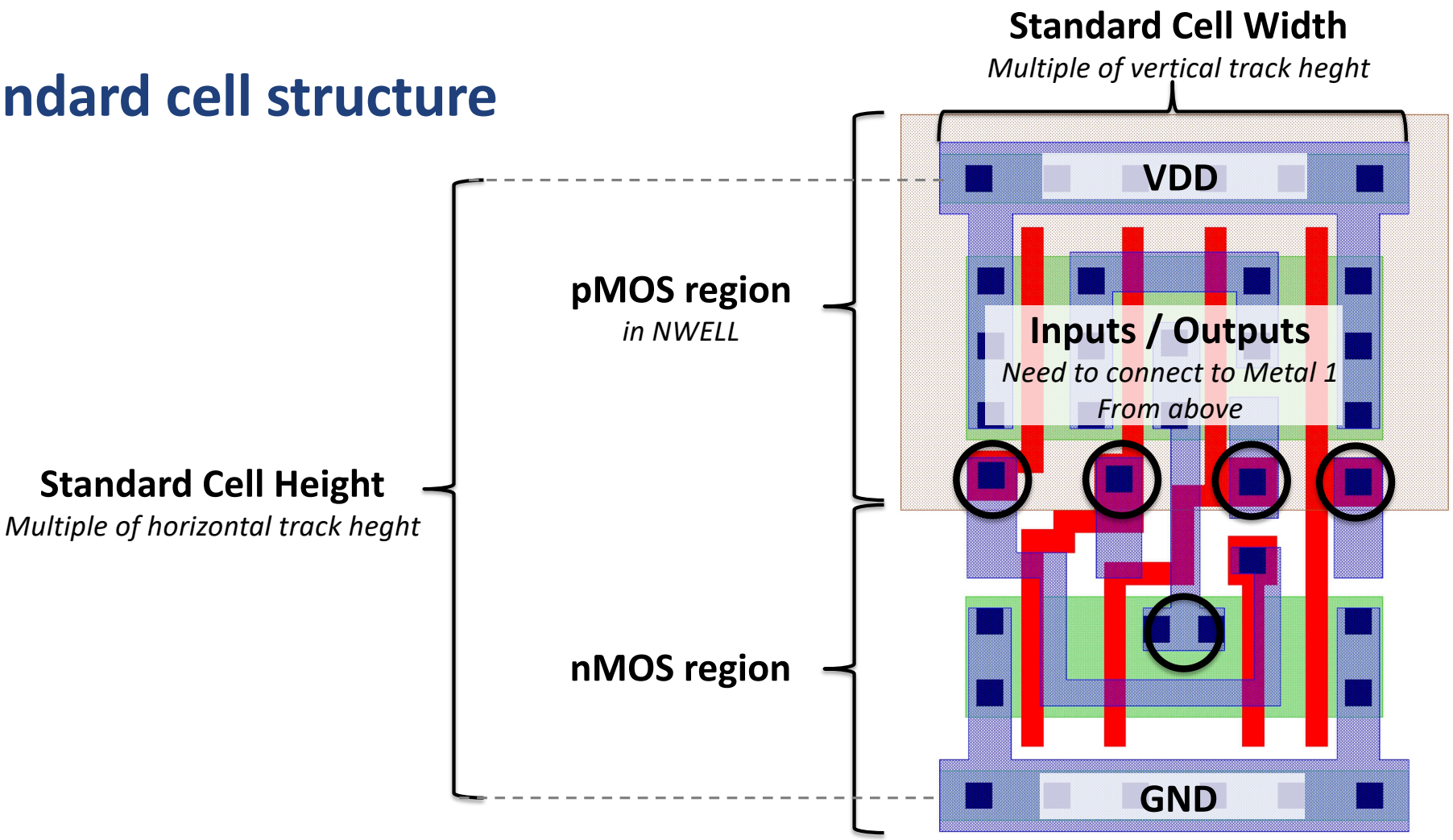


Physical observations

- We have two complementary sides
  - pMOS for pulling up (towards VDD)
  - nMOS for pulling down (towards GND)
- Same number of transistors on both sides
  - True for most, there are some exceptions
- Power connections
  - VDD is connected to pMOS
  - GND is connected to nMOS
- **Inputs connect to both pMOS and nMOS**

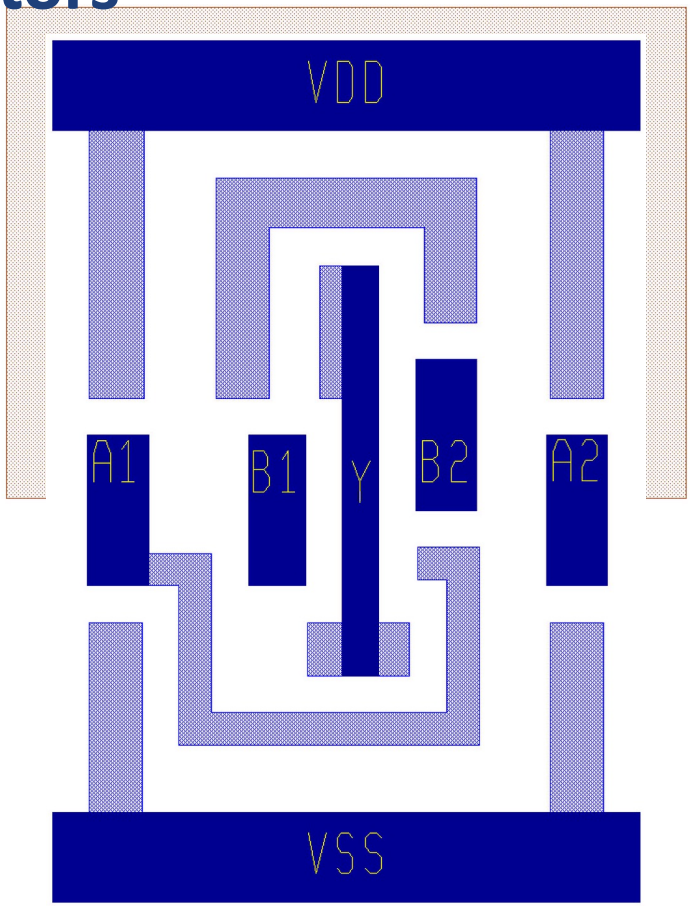


Standard cell structure



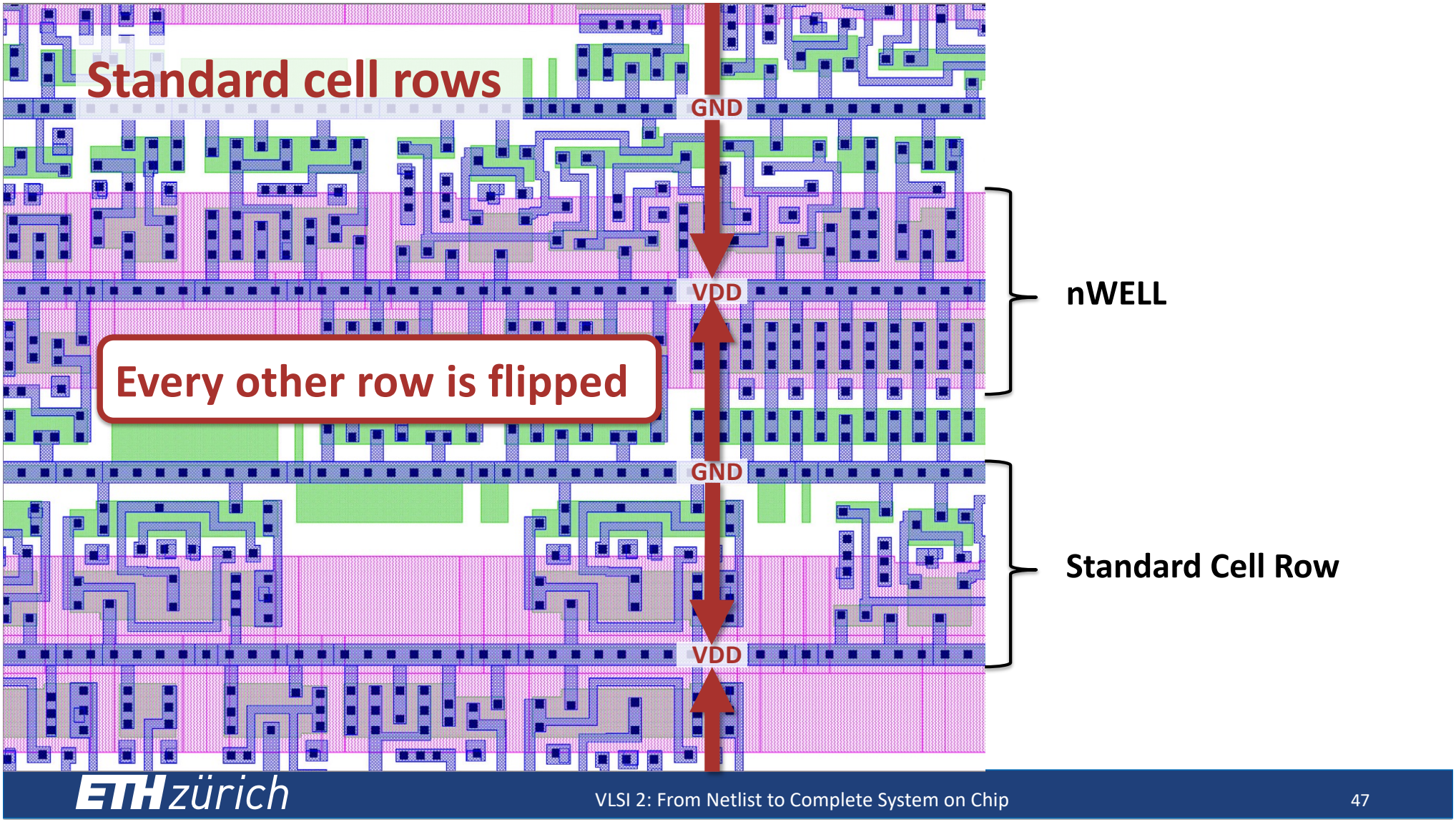
Back-end design does not need transistors

- **Verilog netlist tells us**
  - Type of gates
  - Which inputs are connected to what
  - Where the output is connected
- **Tool does not need more details**
  - For every cell we need
    - Location and layer of all pins (in and out)
    - If there are some restrictions for routing (blockages)
  - We call these views abstracts
    - Defined in the LEF file (we will talk about that later today)

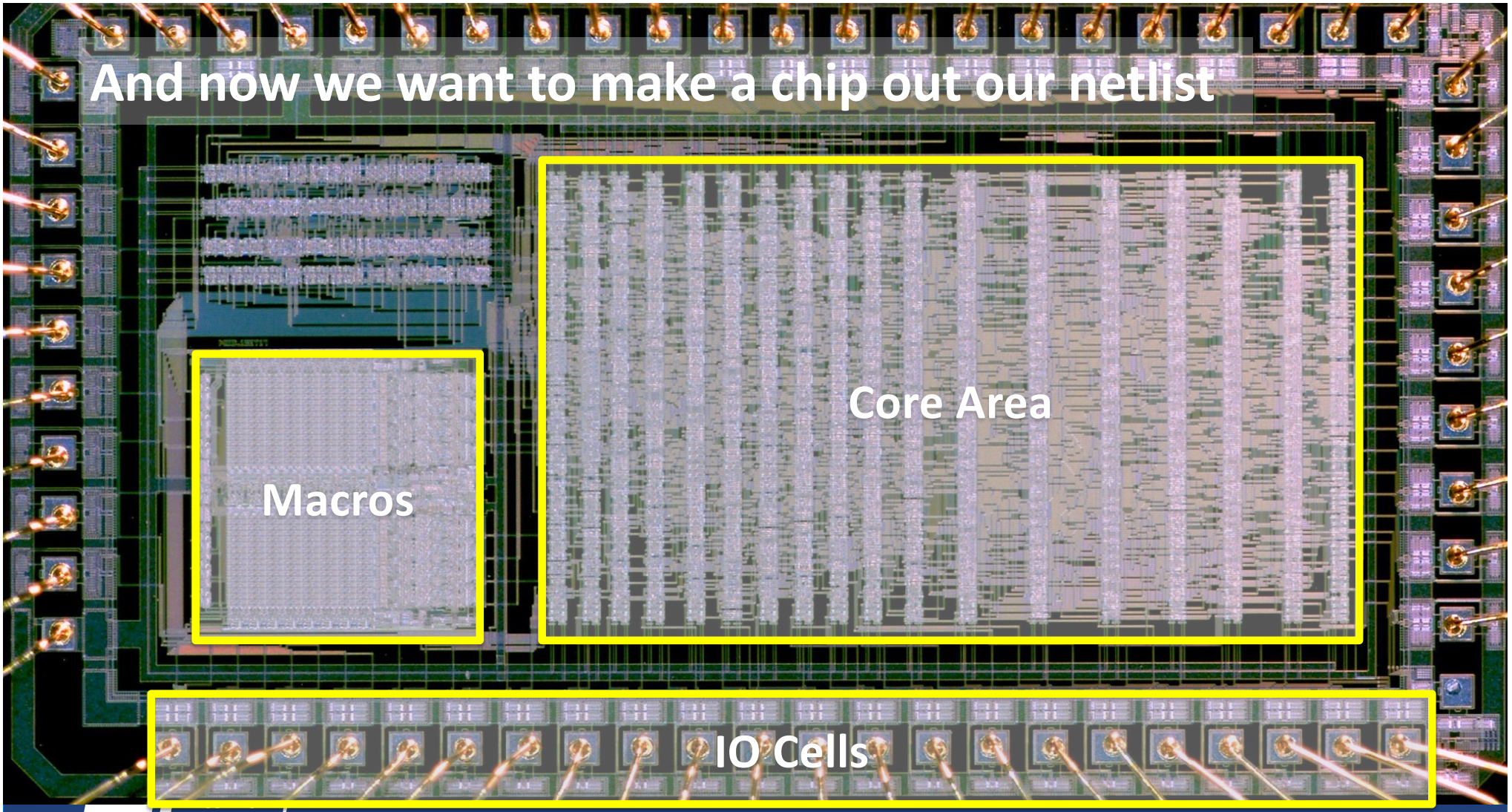


Standard cell rows

The only thing we need to know is where to connect to the cell


Standard cell rows
GND
VDD
Every other row is flipped
GND
VDD
nWELL
Standard Cell Row

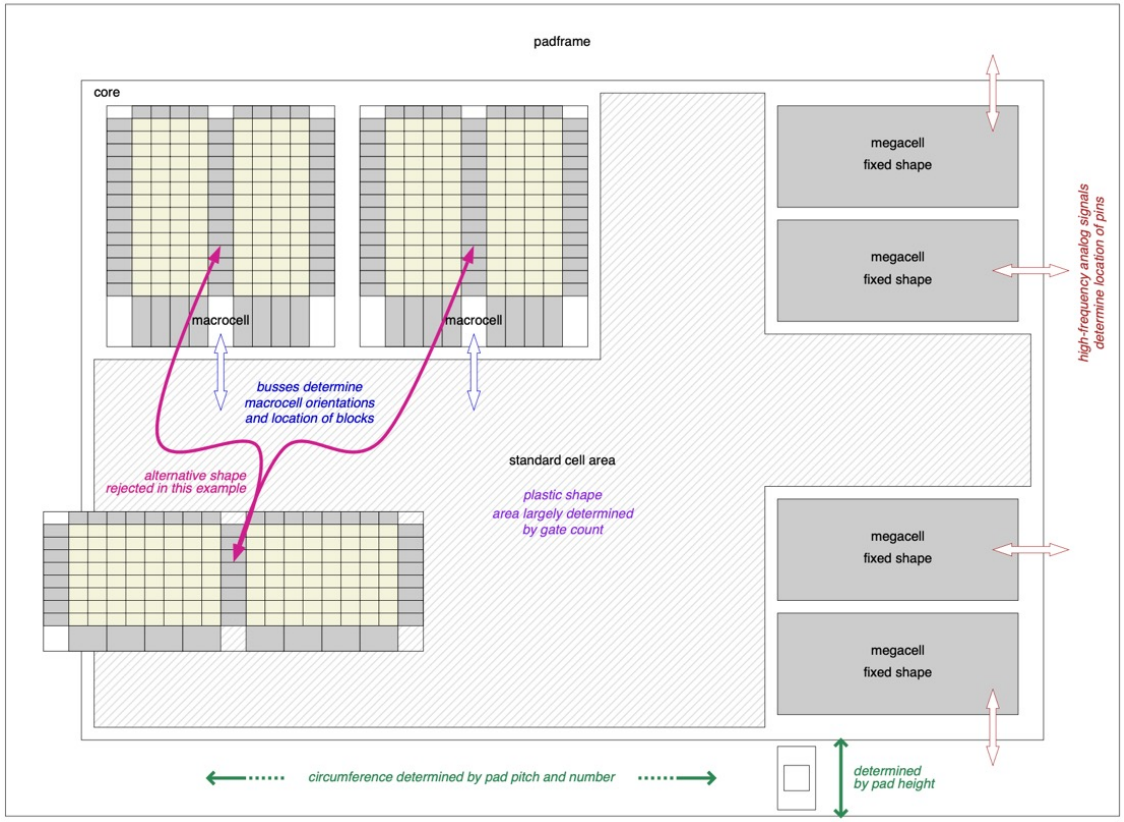
And now we want to make a chip out our netlist



We need a floorplan to determine the circuit area/shape

- **The unit cost of IC is directly proportional to the silicon area**
  - Reducing the die size, reduces the cost
  - Goal: Find the minimum viable silicon area to realize our circuit!
    - Without increasing the engineering cost beyond what we are saving in terms of unit cost.
- **The netlist (lecture 3) gives us the net cell area needed, but:**
  - The cells can not be placed next to each other with no space
    - Additional area required to bring power, solve practical issues (antenna cells, tap cells)
    - Timing constraints will pull parts of the circuit together, causing congestion
    - Space needed for further optimizations (clock tree, buffers for time fixing)
  - Space around macros, I/O cells that are not aligned in standard cell rows
- **I/O and packaging constraints (later this lecture) can also set limits**
  - The net area needed for your chip will be larger than the net cell area

Initial considerations for a floorplan



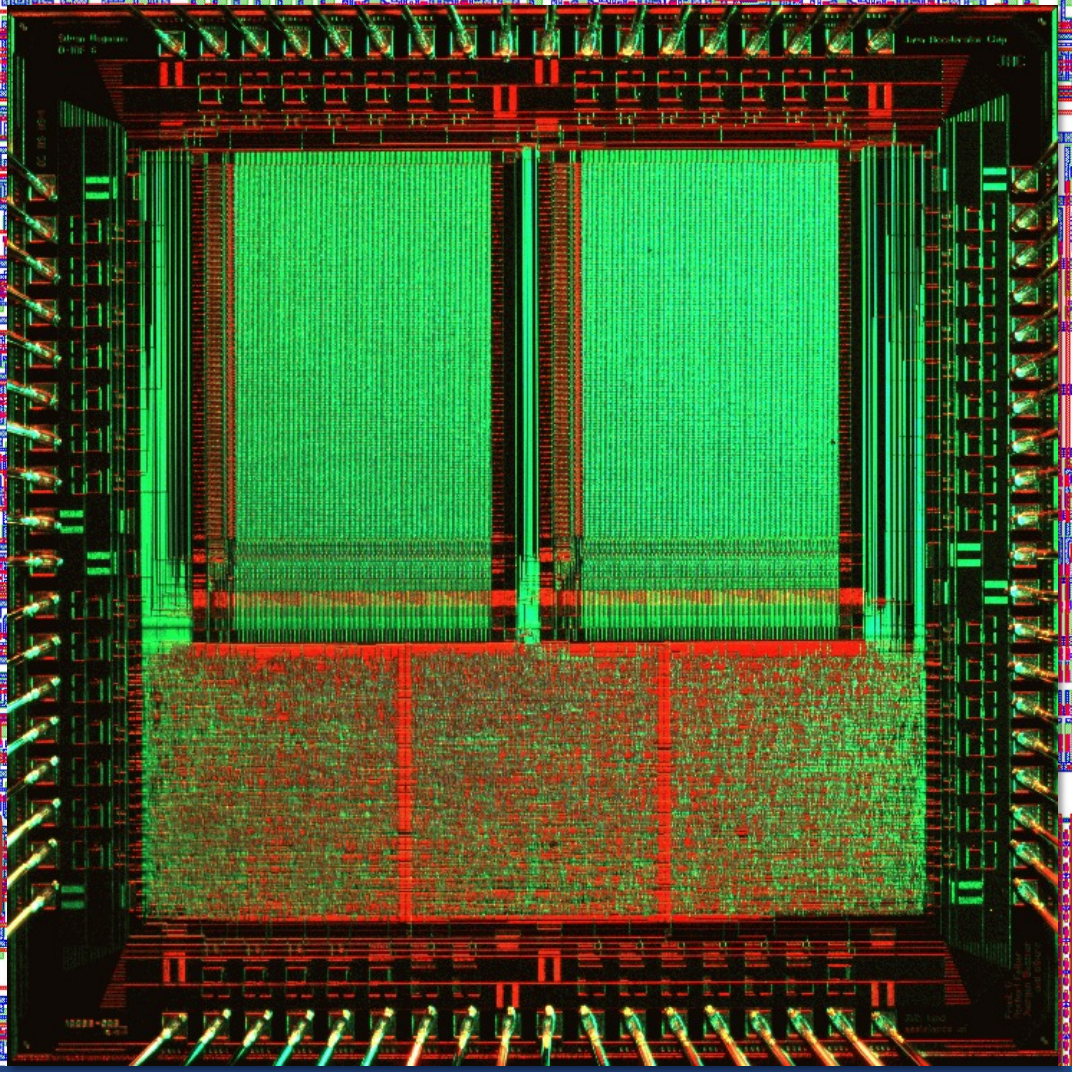
- **Area for the I/O pads**
- **Place macro cells**
  - These are rigid blocks that are 'dropped' in.
  - Make sure they connect well to rest of the chip
- **Define the core area**
  - Where we will have the standard cells

Slide adapted from Hubert Kaeslin, "Top-Down Digital VLSI Design Vol2: From Gate-Level Circuits to CMOS Fabrication"

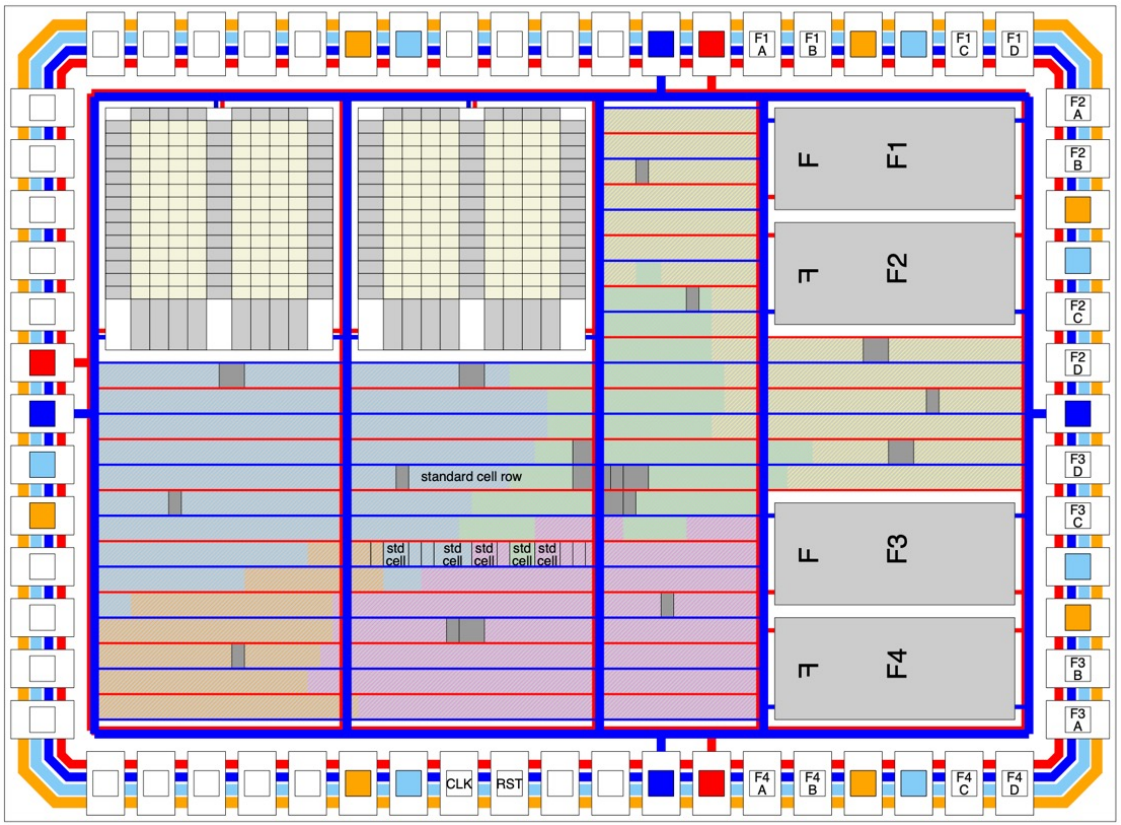
Story about macro cells

Pre-designed cells that do not fit into standard cell rows

In this case a memory macro



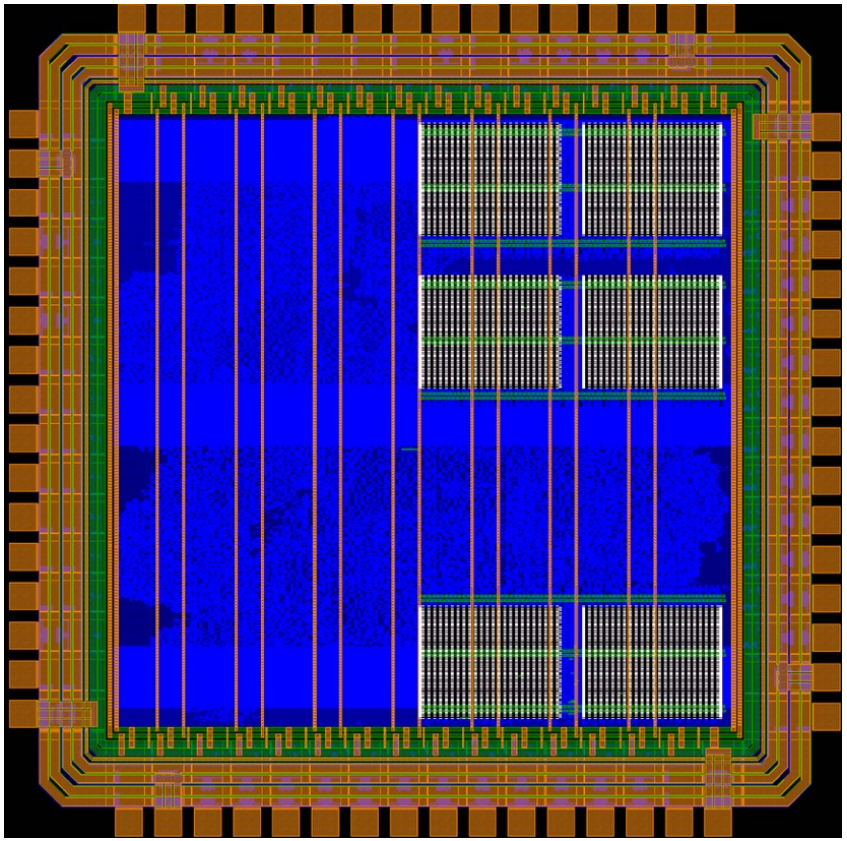
Initial considerations for a floorplan



- **Area for the I/O pads**
- **Place macro cells**
  - These are rigid blocks that are ‘dropped’ in.
  - Make sure they connect well to rest of the chip
- **Define the core area**
  - Where we will have the standard cells
- **Connect power**

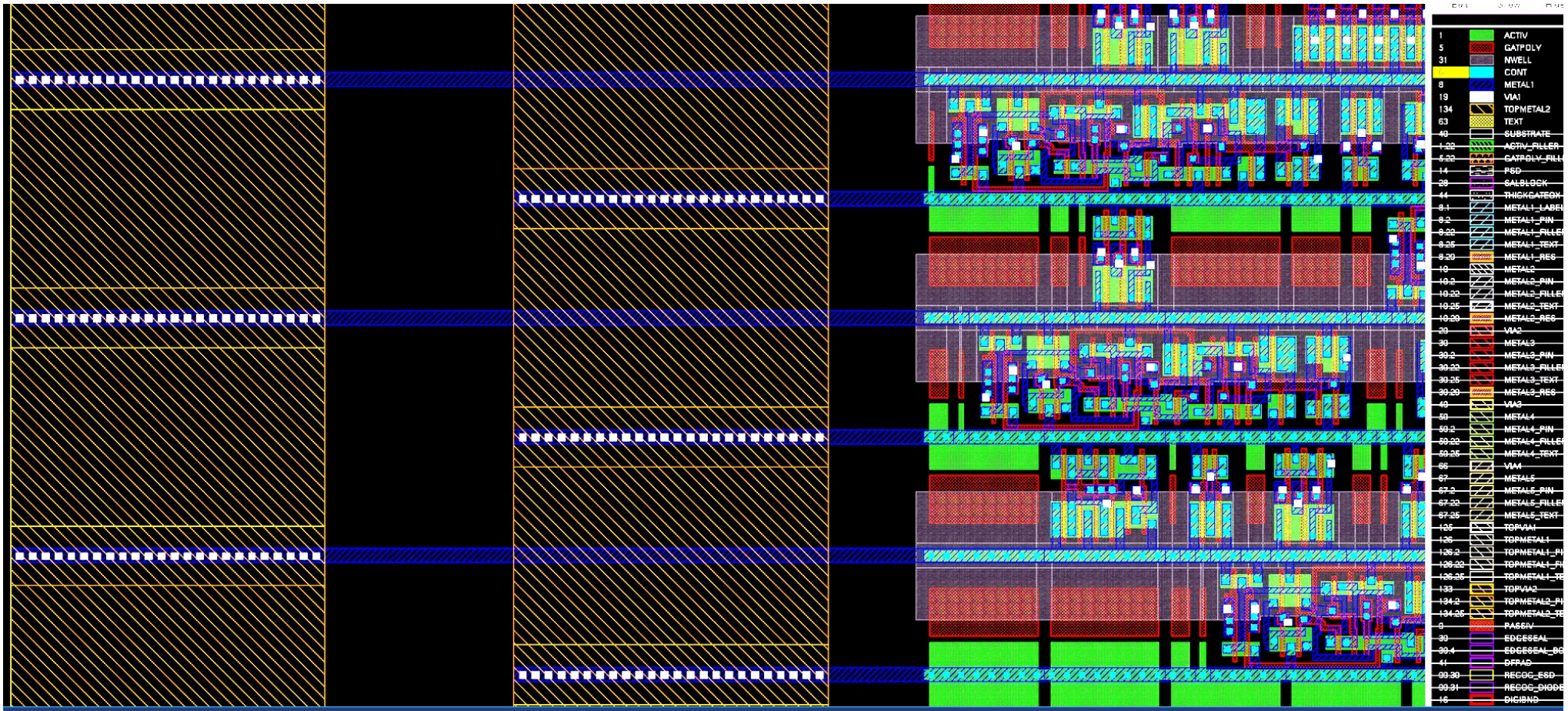
Slide adapted from Hubert Kaeslin, “Top-Down Digital VLSI Design Vol2: From Gate-Level Circuits to CMOS Fabrication”

Connecting power to the chip

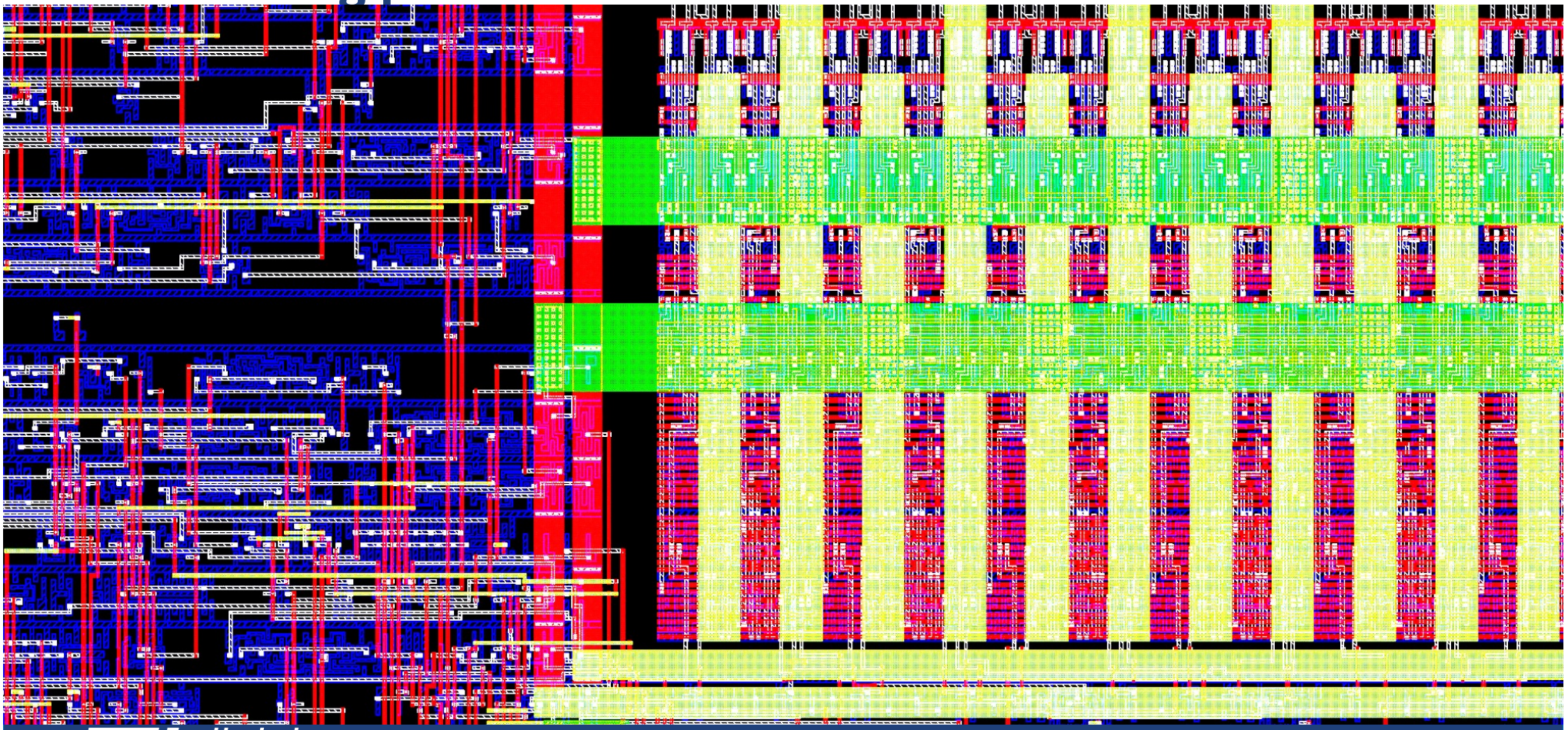


- **The power plan is implemented in floorplan**
  - What metal layers
  - How wide
  - Connect how

Connecting power: Cells



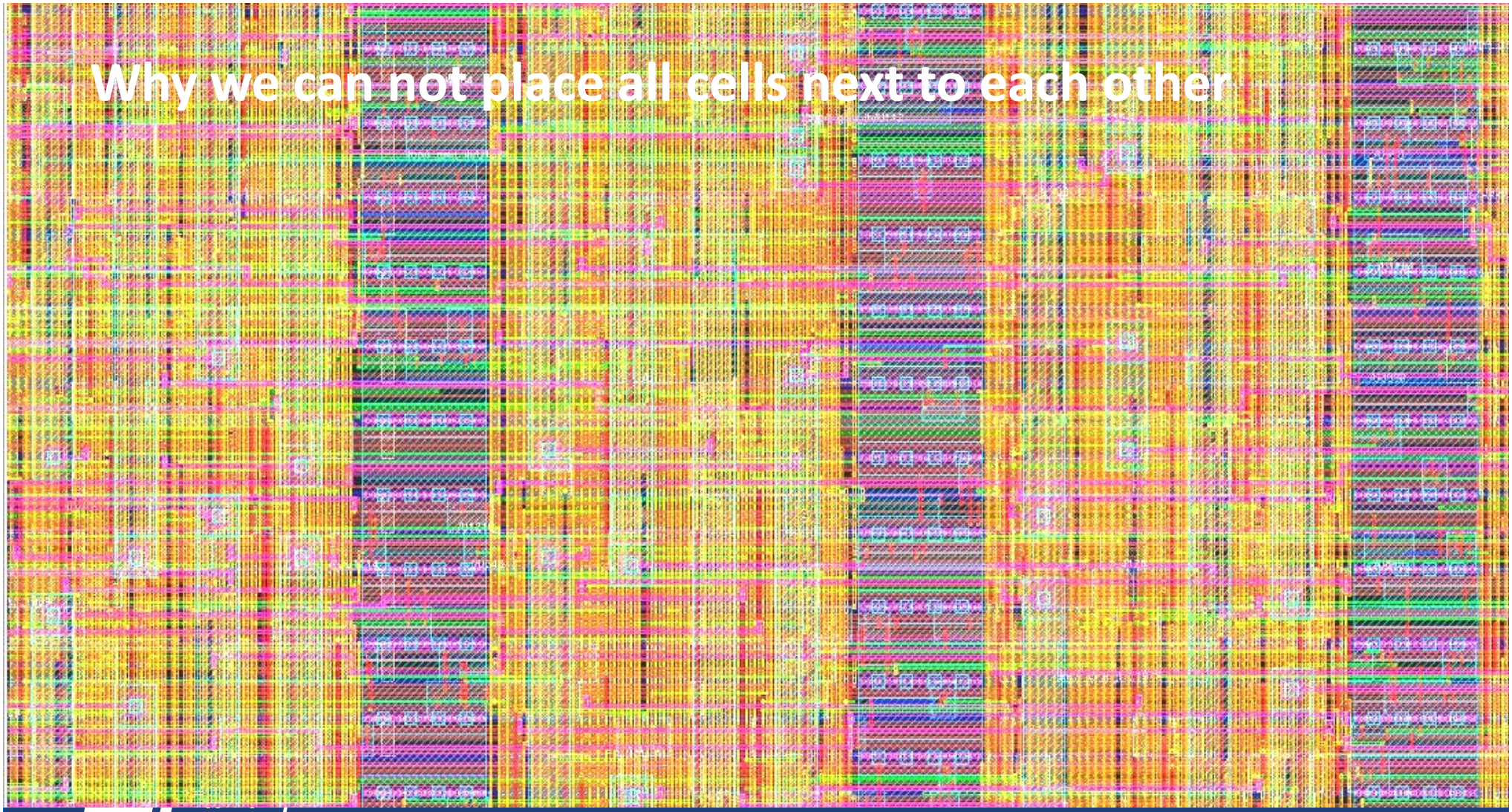
Connecting power: Macros



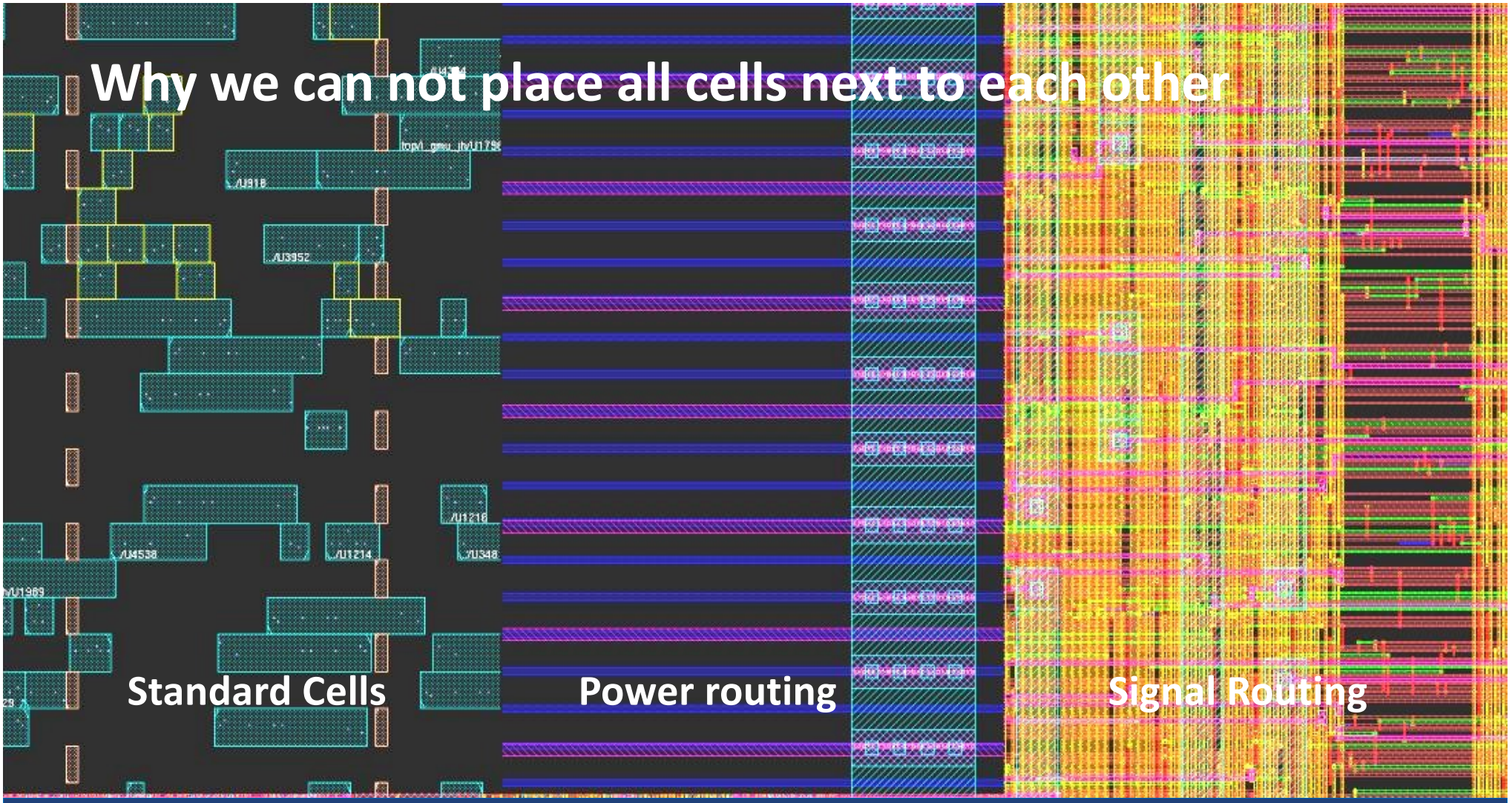
How much area do we need?

- **The hard macros are (relatively) easy**
  - They are already pre-designed, have a fixed shape/area
  - They need a bit of space around so that they can made to fit with the rest
- **I/O cell rules are relatively straightforward**
  - The types and size of all I/O cells are already known
  - Packaging, bonding guidelines will determine spacing and placement
- **But cell area is not that easy**
  - We need at least the net cell area from the netlist
  - And maybe quite a bit more
  - Just how much more depends on technology as well as the design
  - Requires experience and iterations

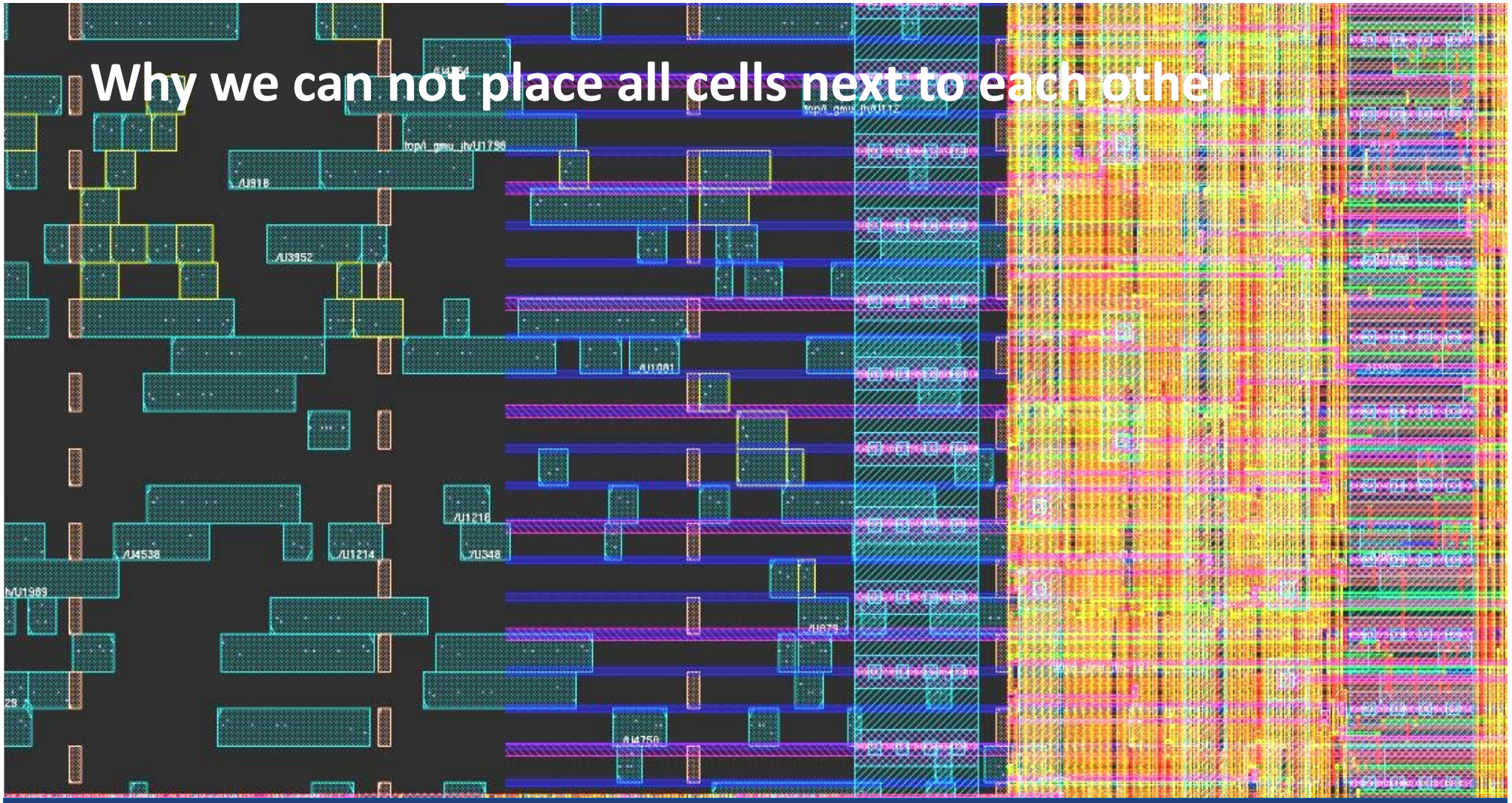
Why we can not place all cells next to each other



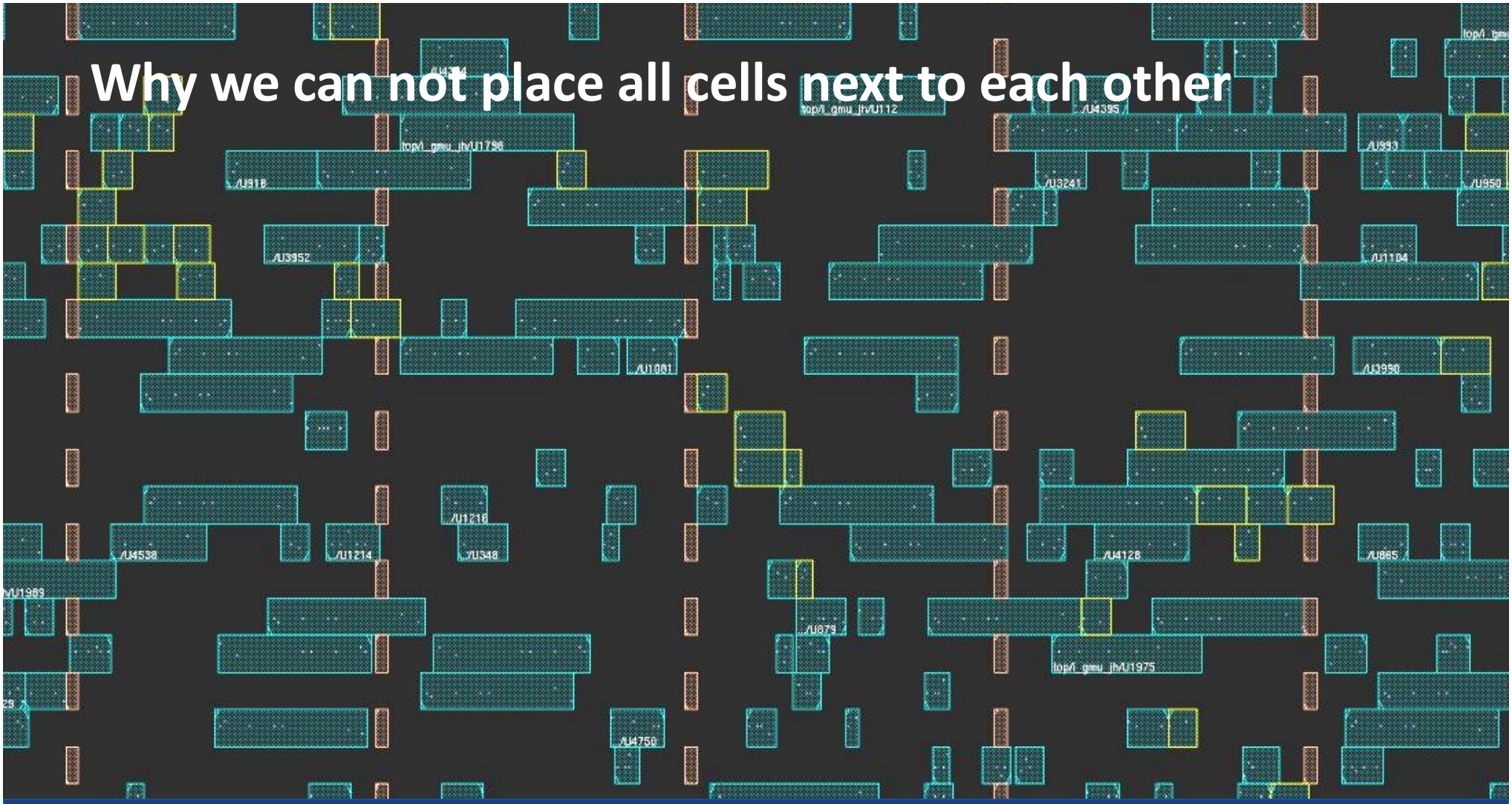
Why we can not place all cells next to each other



Why we can not place all cells next to each other



Why we can not place all cells next to each other

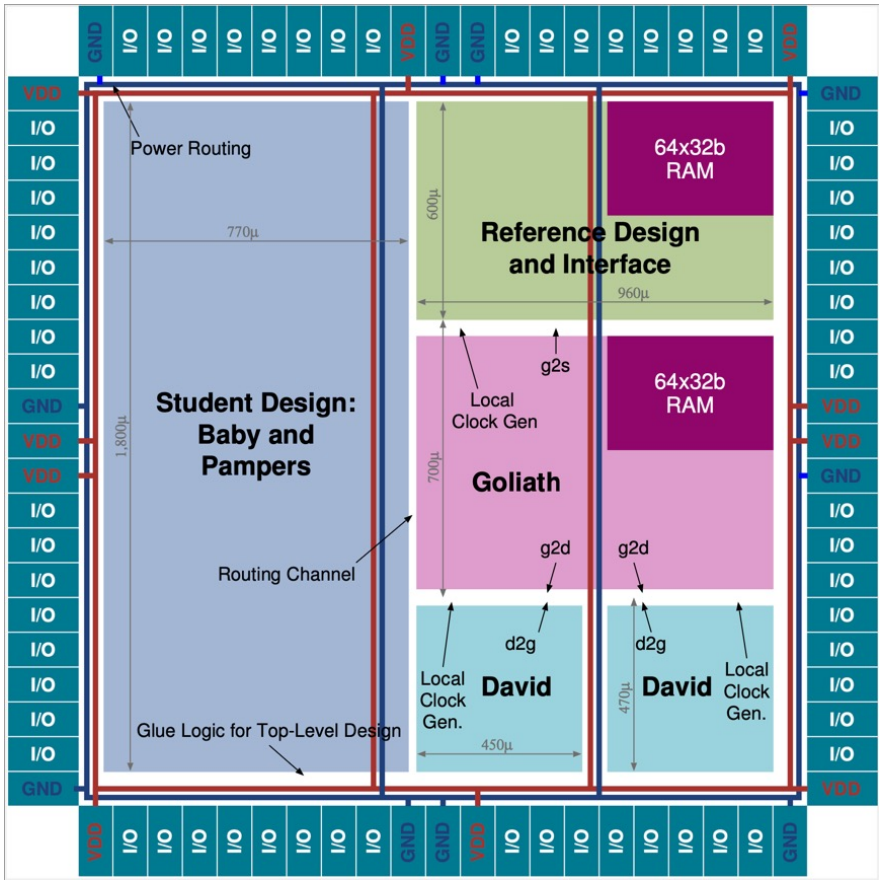


Core utilization: a measure of how well we fill the chip

- **We need some extra space in core area as explained**
  - Signal routing congestion
  - Room for optimization
  - Additional physical cells, (tap cells, antenna cells)
  - Keep out regions around macros
- **Core utilization tells us the percentage**
  - Core utilization = net cell area / reserved core area
- **The core area is determined by *estimating* the core utilization**
  - You continue through the back-end process and see if this is a workable solution
  - Highly iterative and experience based
  - For a given technology and type of design you get good initial estimations

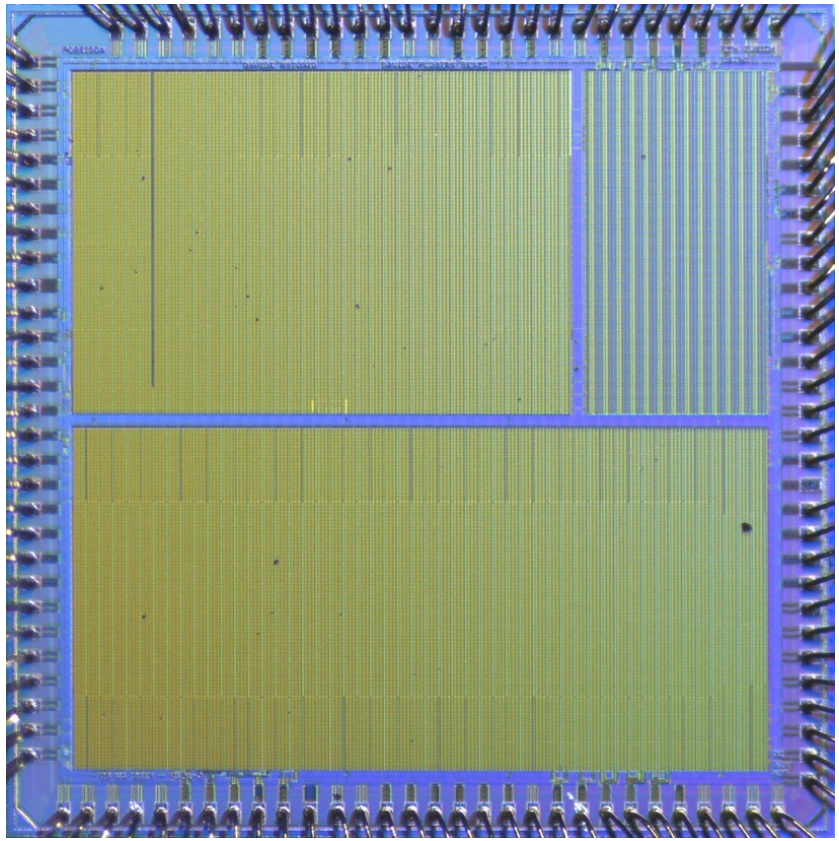
Hierarchical design, partitioning

- **Divide and conquer**
  - Complex designs are broken down to smaller pieces
  - Each block is then independently placed and routed
- **Flexibility requires tool support**
  - One can see subdesigns as a macro block, finish them and include on top
  - A tool can keep the boundaries fluid allowing more optimization during the process
- **Goal is to reduce effort!**



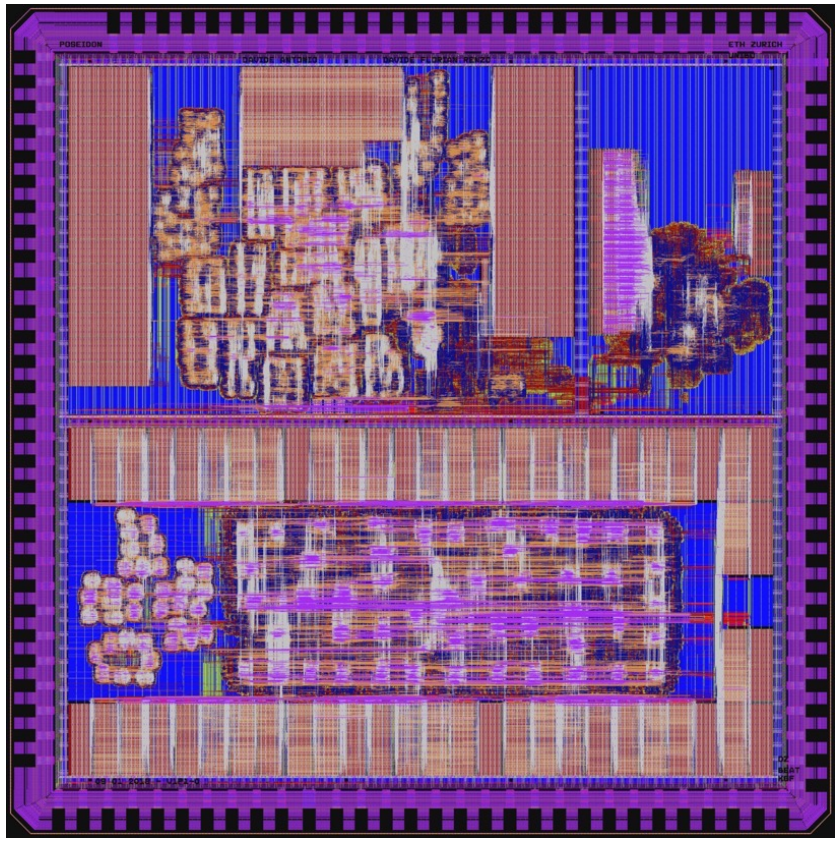
Example hieararchical design: Poseidon

- **Clearly see three different parts**
  - Designed independently
  - Timing constraints for interfaces between modules
- **This design is bottom-up**
  - Initially area is allocated to each three designs based on estimations (and margins)
  - Designs placed & routed separately
  - At top level designs are macro blocks



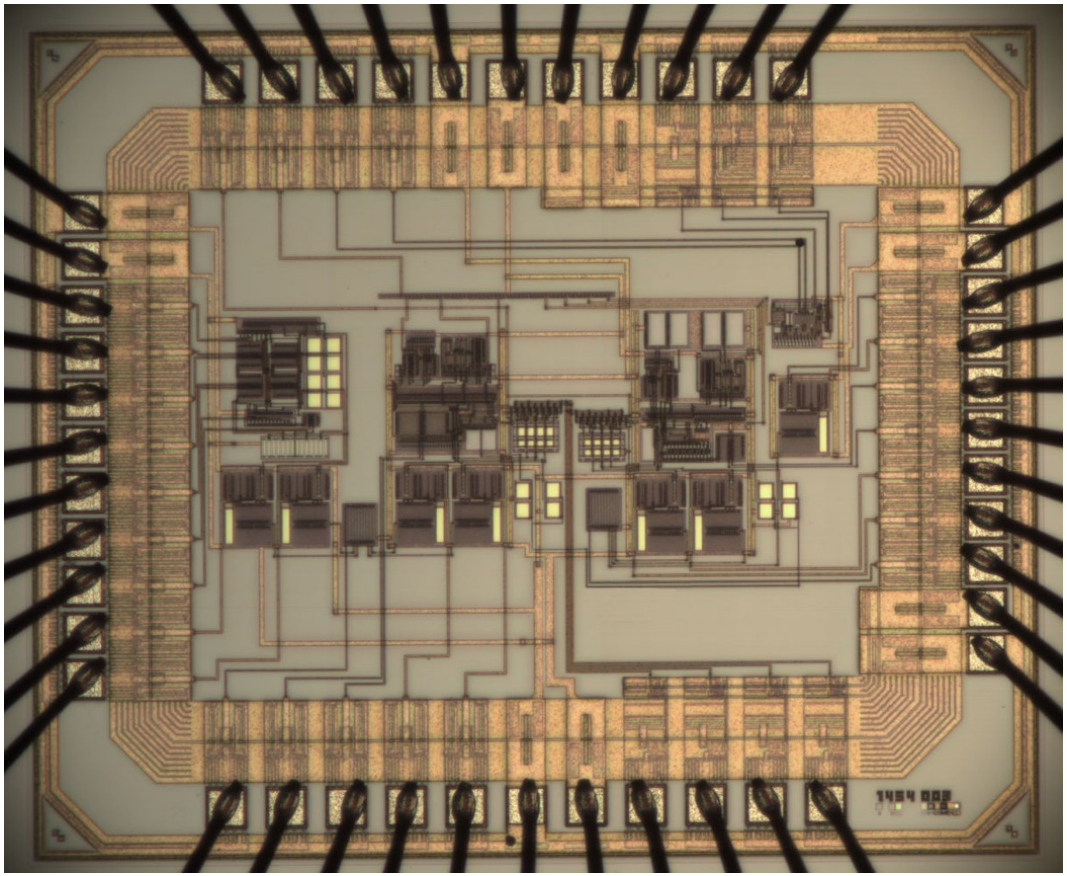
Example hieararchical design: Poseidon

- **Clearly see three different parts**
  - Designed independently
  - Timing constraints for interfaces between modules
- **This design is bottom-up**
  - Initially area is allocated to each three designs based on estimations (and margins)
  - Designs placed & routed separately
  - At top level designs are macro blocks
- **Alternative is top-down**
  - Requires more tool support



We use specialized I/O transistors to in modern chips

- **We use different power supplies for core and I/O.**
  - 5.0V, 3.3V, 2.5V, 1.8V, 1.2V…
  - 14nm chip does not have I/O transistors that are 14nm
- **I/O drivers are able to drive loads on the PCB**
- **Isolate outside from inside**
  - Provide protection against Electro-Static Discharge (ESD), an issue during handling



Signals on the outside and inside of a microchip

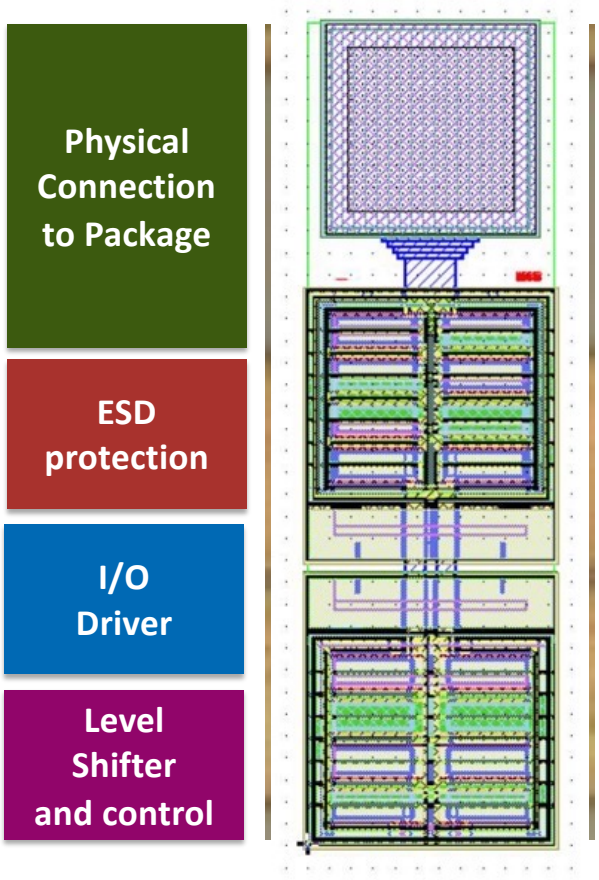
Outside the chip

- **Larger loads**
  - PCB tracks (pF)
  - Input capacitances of chips (pF)
- **Longer distances**
  - PCBs can be 30cm or larger
- **More noise sources**
  - More exposed

Inside the chip

- **Tiny transistors (Moore's Law)**
  - Very small capacitances (fF)
  - …and very thin oxides
- **Shorter distances**
  - Largest chips 20mm on one side
  - Mostly smaller 2mm – 8mm
- **More controlled environment**
  - Package is sealed

Specialized I/O cells are needed for the connections



- **Physical connection to package**
  - We will need someway to make connection
  - Wire bonds or solder balls
- **Electro Static Discharge (ESD) protection**
  - Make sure that electrical discharge does not destroy sensitive internal circuitry
- **I/O driver**
  - Buffers to drive external (pF) loads
- **Level Shifter & Control**
  - Move from the internal supply voltage to I/O supply
  - Enable, pull-up/down, drive strength control

But why I should I care about this during IC design

- **Selecting correct pad instances part of IC design**
  - A chip that is not powered, does not receive anything from outside and does not communicate its outputs is not very useful
  - Cost of packages is one of the major contributors to overall cost
- **The number and type of pad instances affect our floorplan**
  - I/O pads are not really small, and they bring additional constraints (power ring etc).
  - Pad limited vs core limited, determines area, shape.
- **Proper power distribution and signal quality depends on pad locations**
  - This is not the only criteria, but one of the important aspects
- **Different signals need different special interface circuits (PHYs)**
  - DDR, USB, Ethernet al have specific electrical properties on the interface
  - These PHYs (Physical layer) are designed to comply with these requirements