



Nanoelectronic Architectures

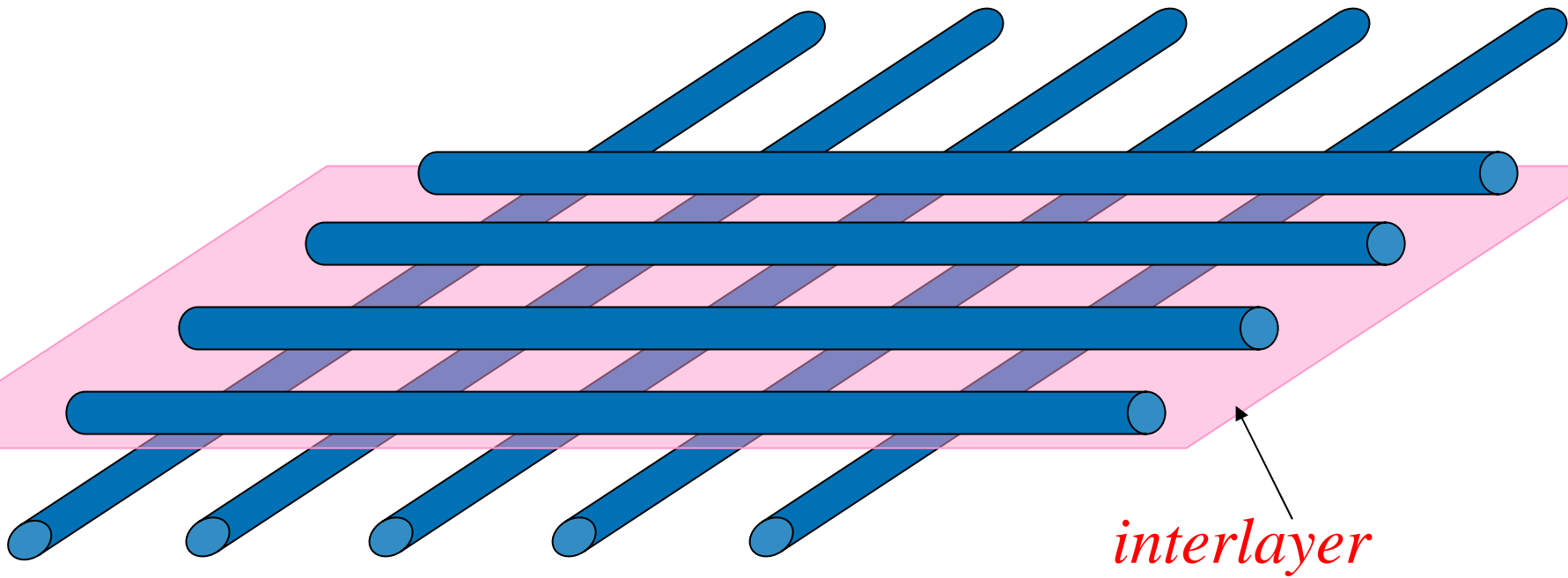
Greg Snider
QSR, Hewlett-Packard Laboratories



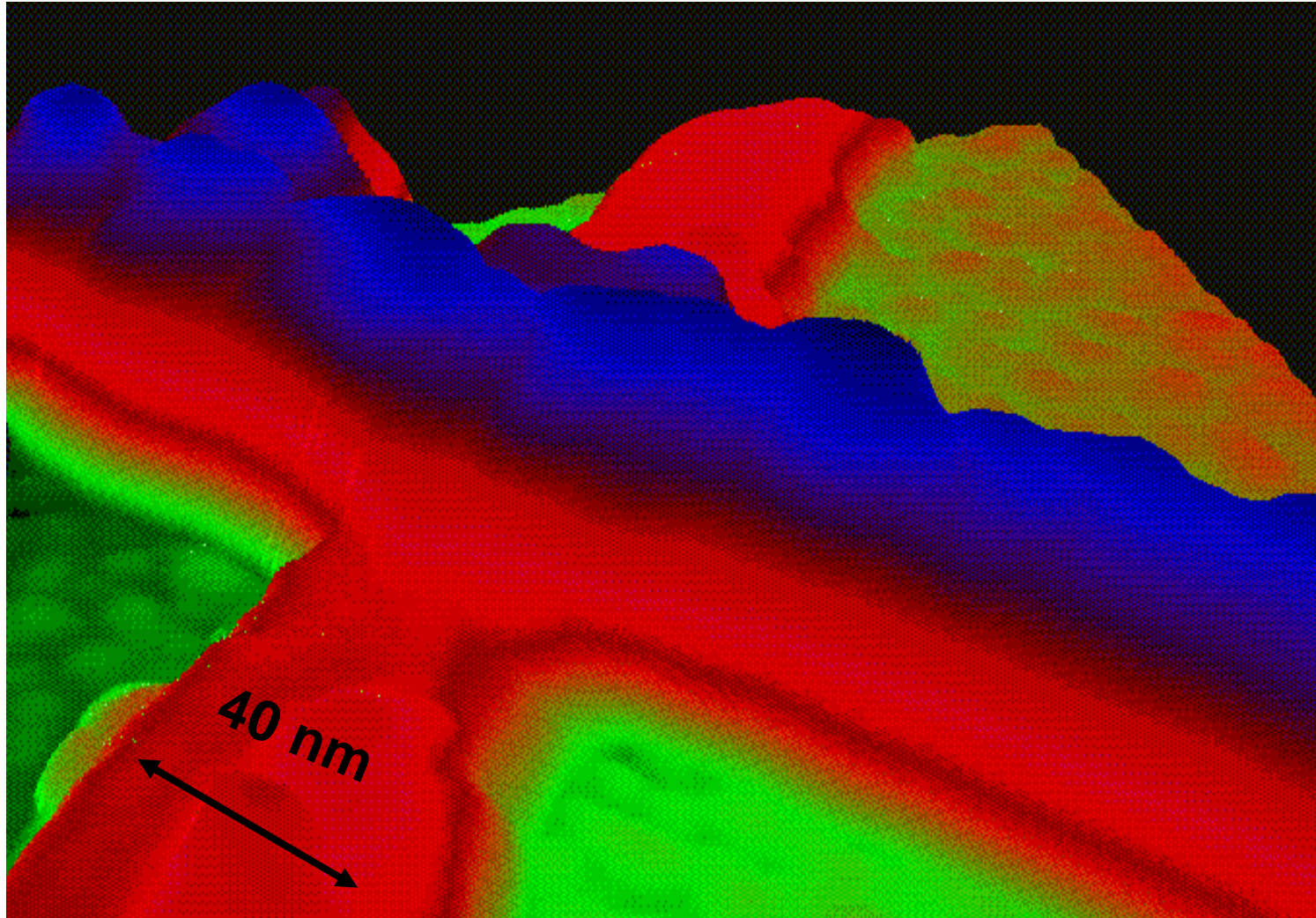
Overview

- Crossbars
 - Conventional logic
 - Defects and faults
-
- Unusual latches
 - Strange logic and weird state machines
 - Demo—compilation and simulation

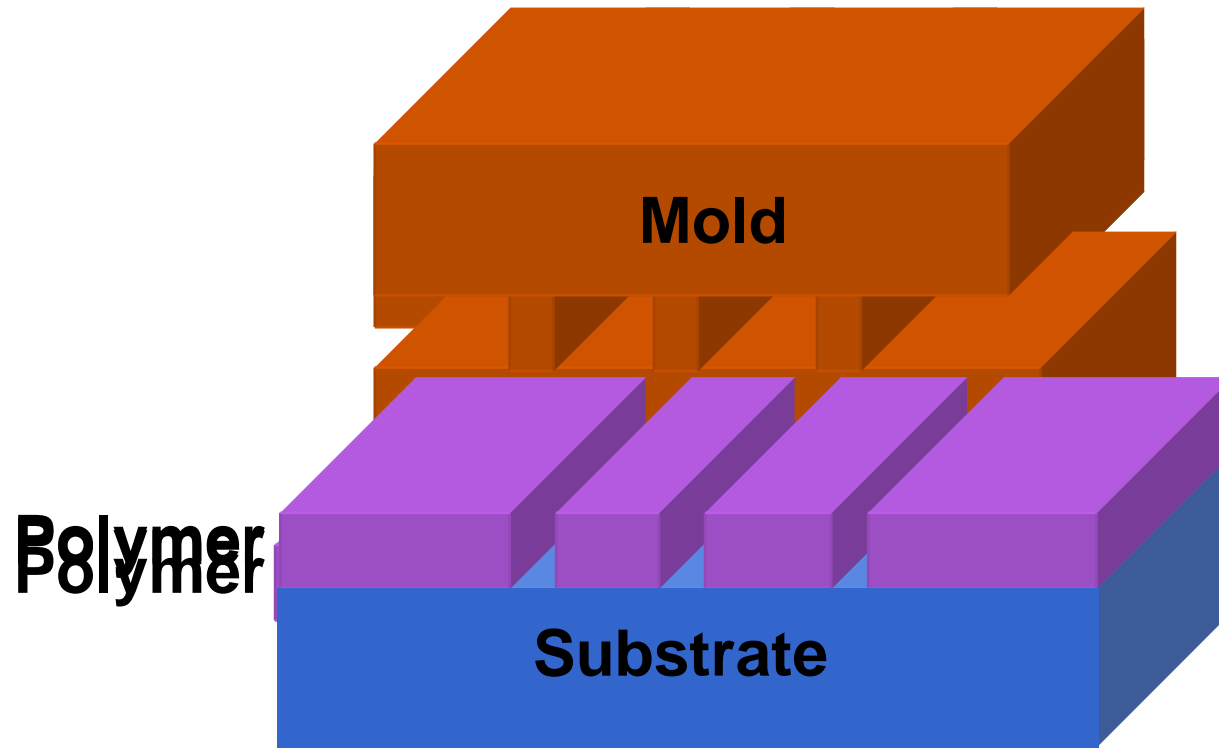
Crossbars



Crossbars

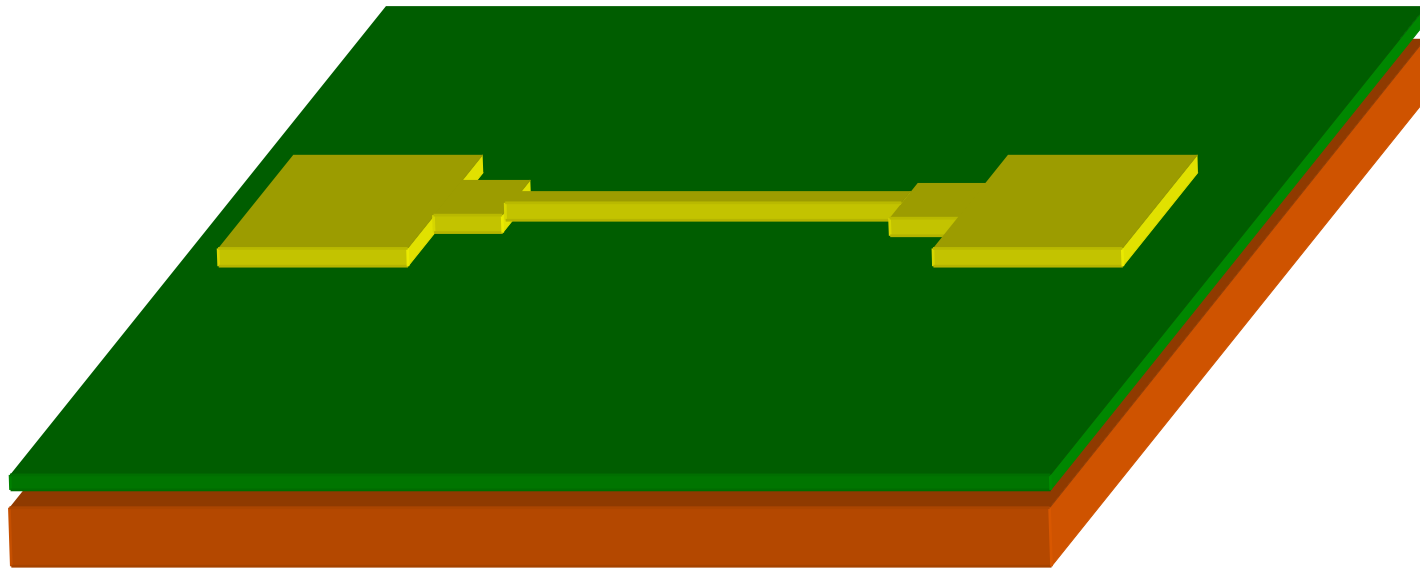


Imprint Lithography

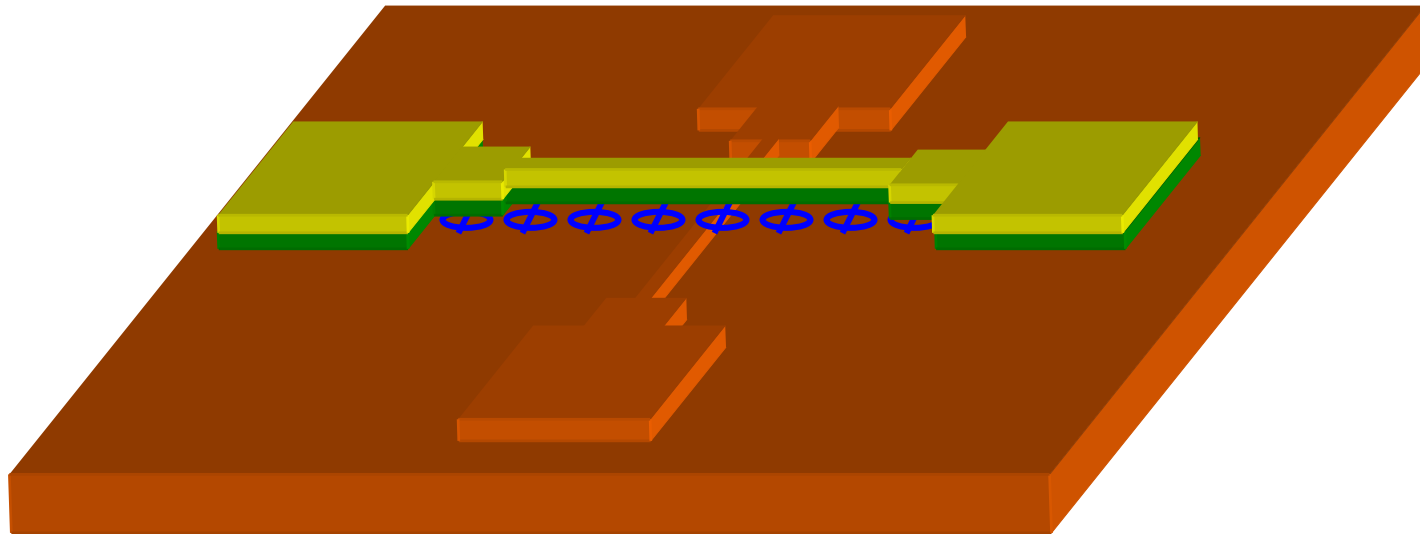


US Patent 6432740

Nanoscale Molecular Devices

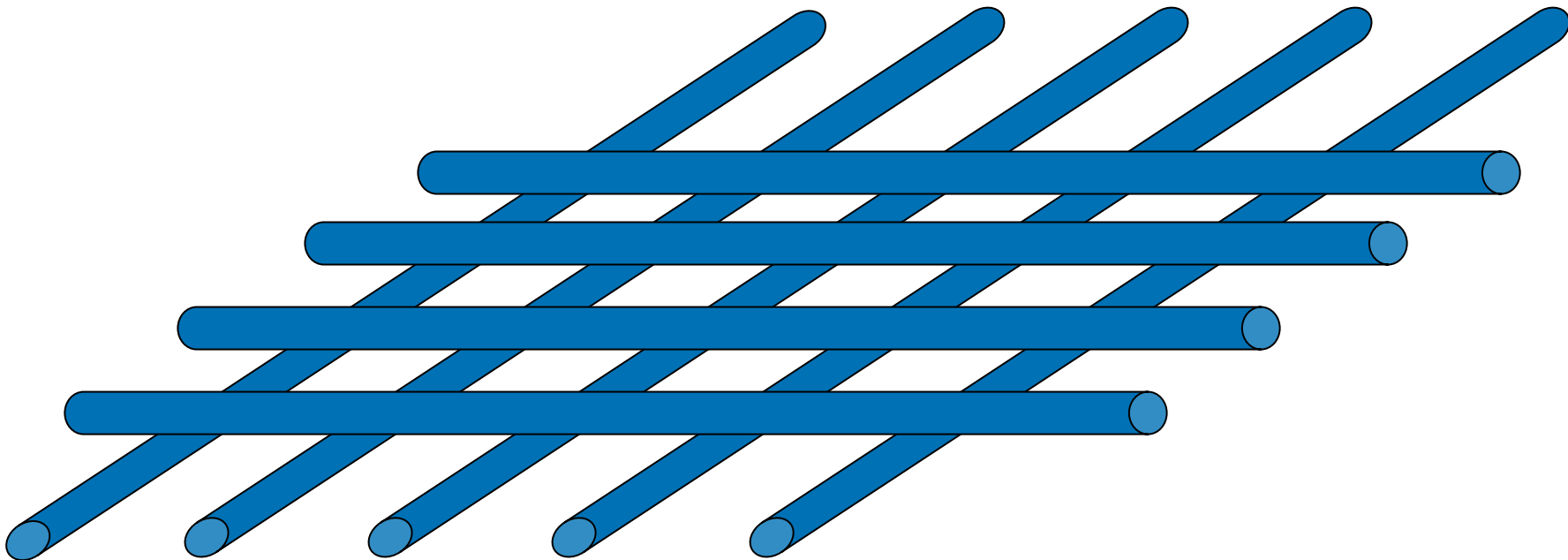


Nanoscale Molecular Devices

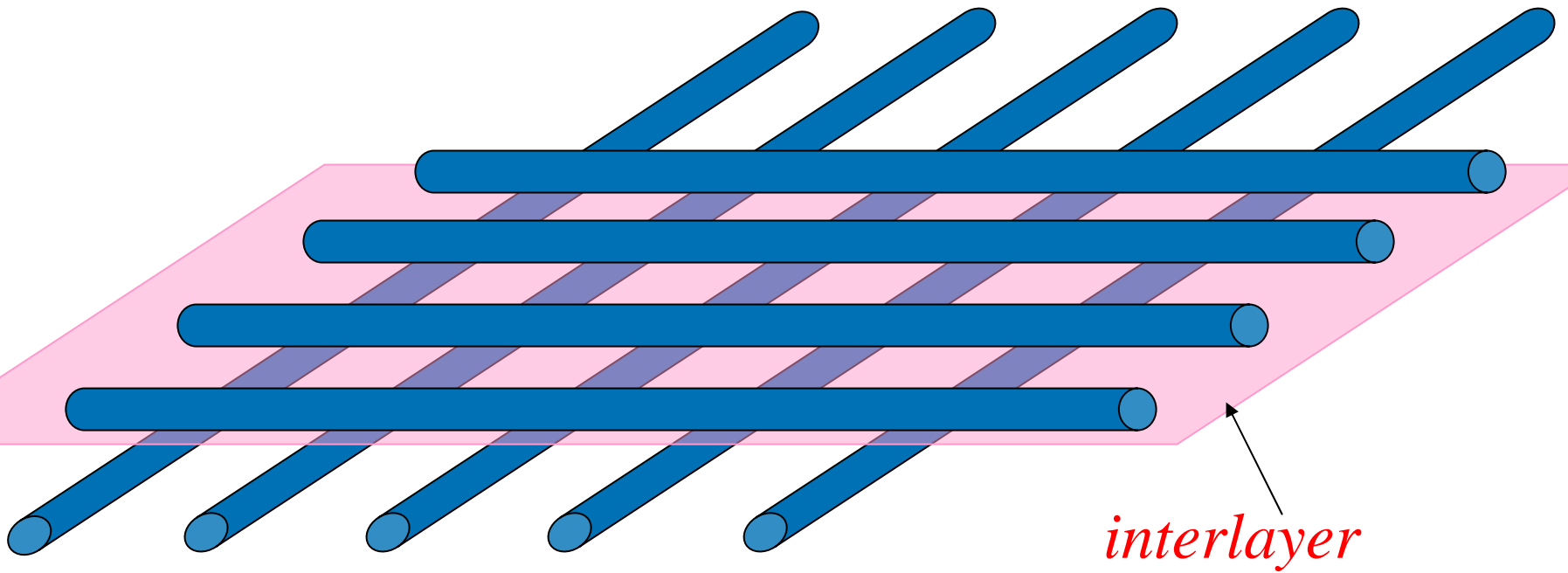


Configurable Crossbar = *Tile*

Crossbar

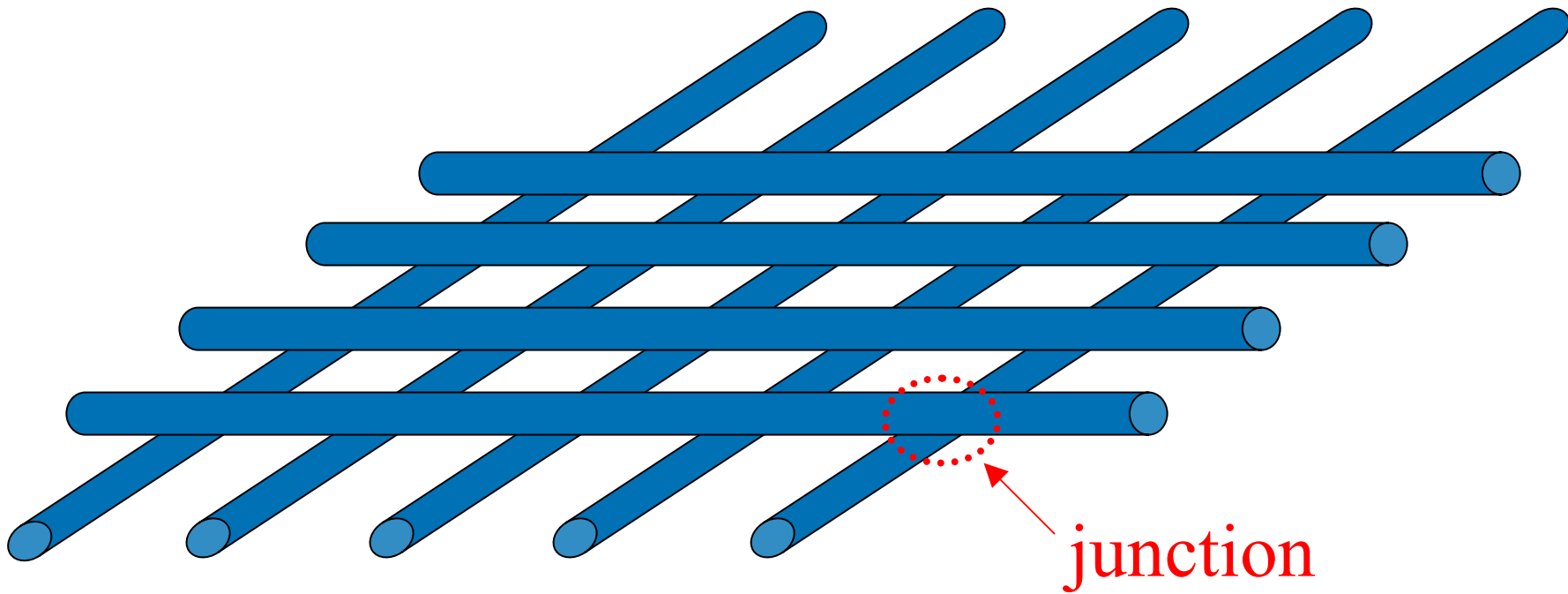


Configurable Crossbar = *Tile*

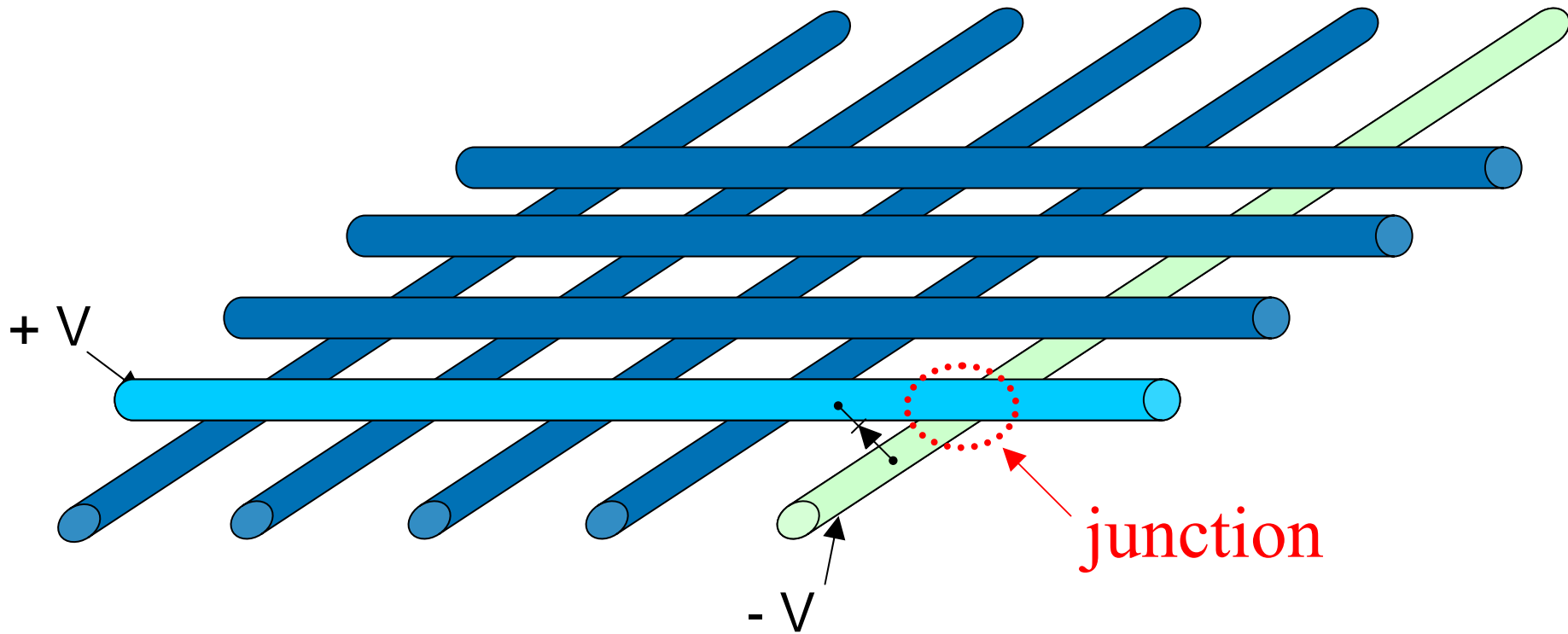


Configurable Crossbar = *Tile*

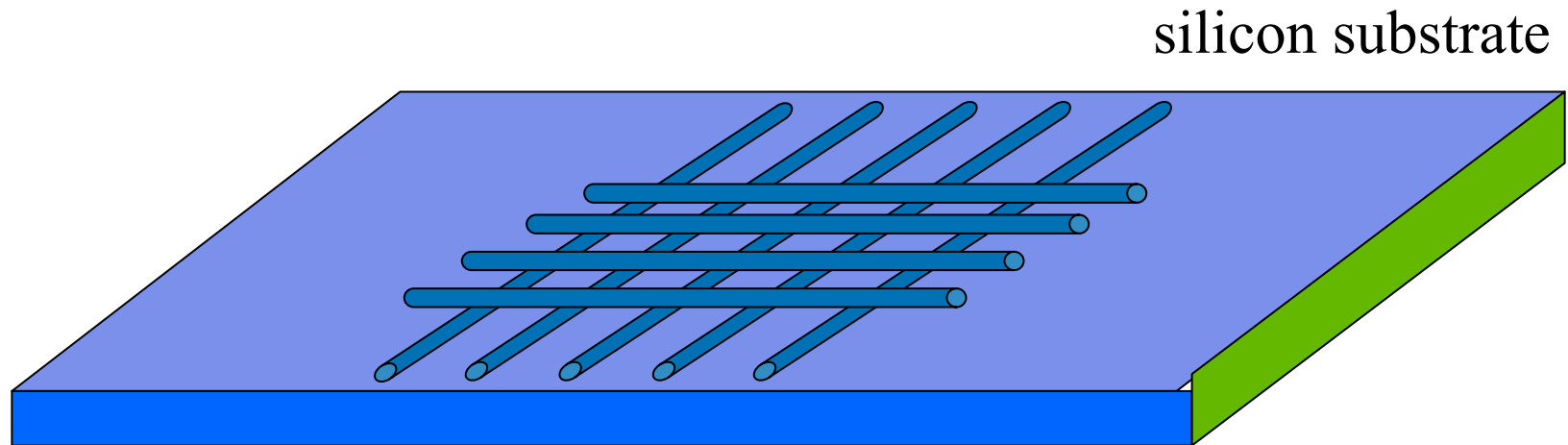
Crossbar



Configuring a junction



Interfacing: one scenario



Substrate provides through microwires:

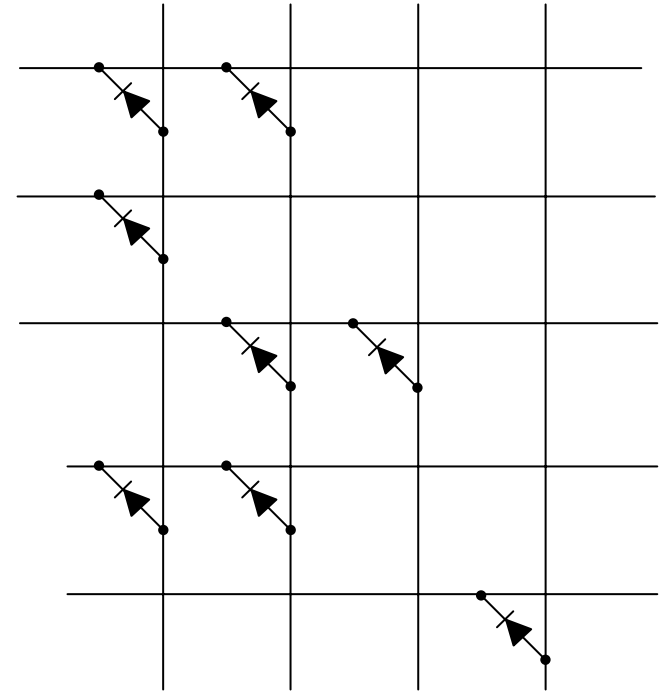
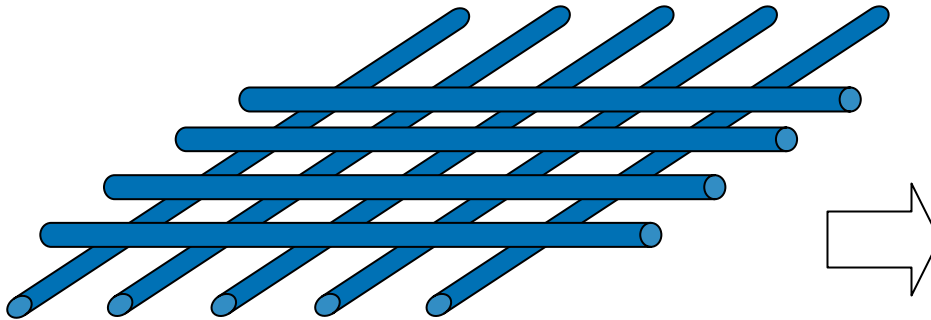
power

clock

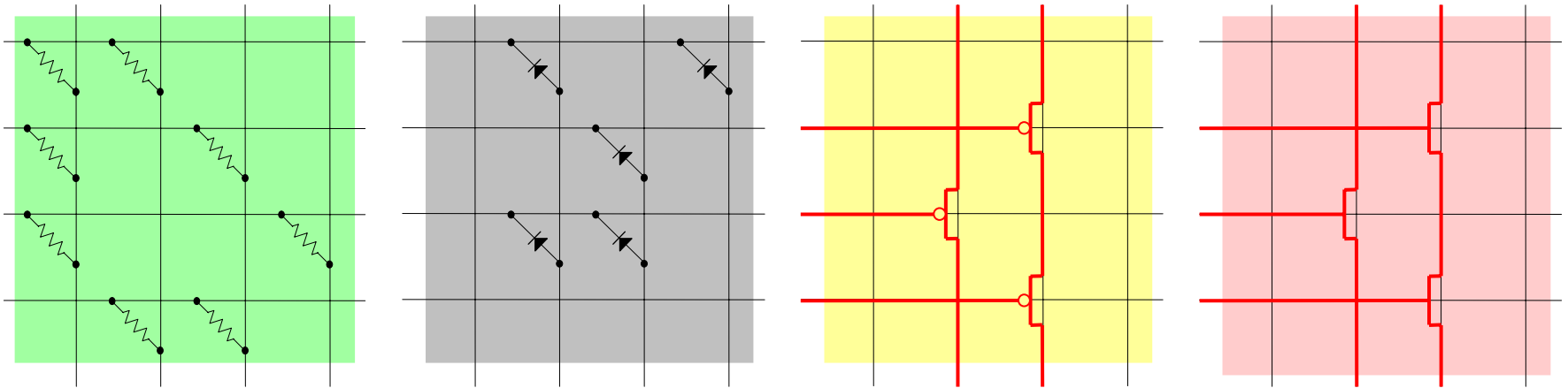
data I/O

configuration I/O

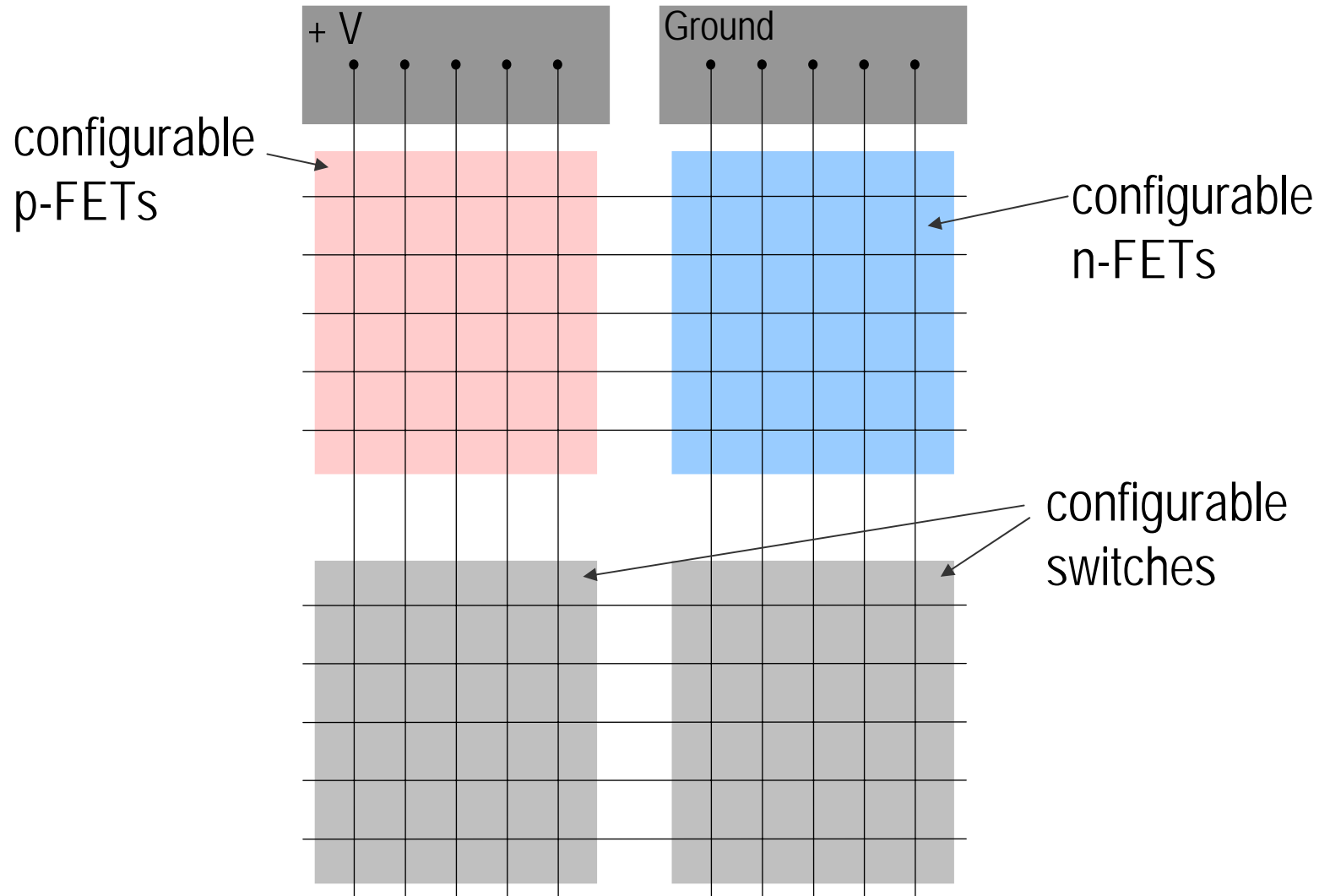
Tiles \rightarrow Circuits



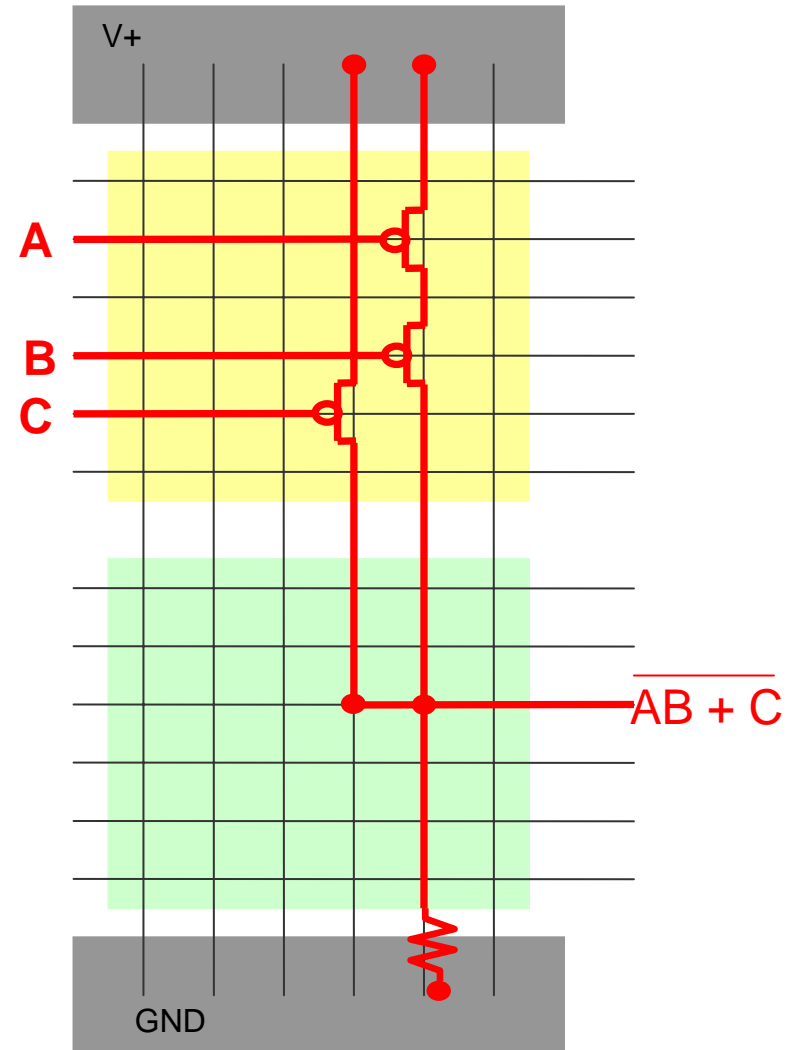
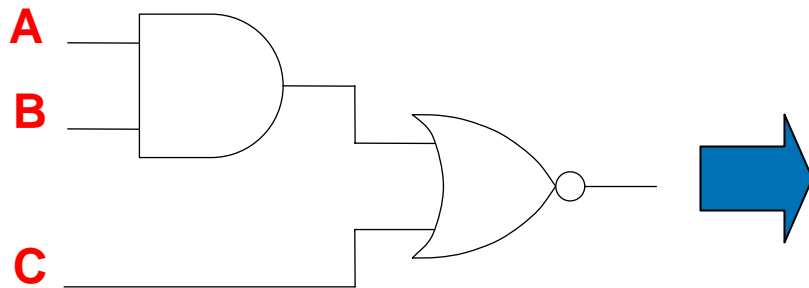
Tile Types



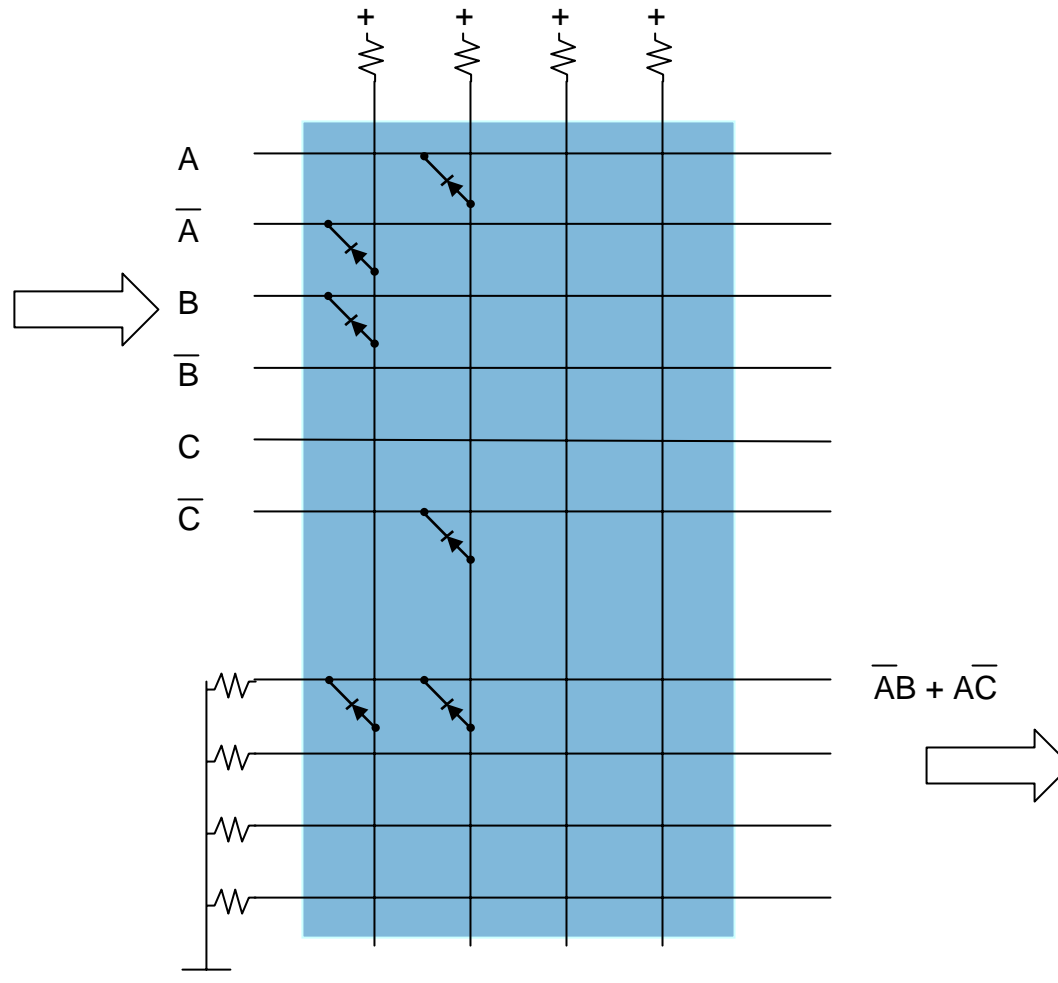
Mosaics = tiles of different types



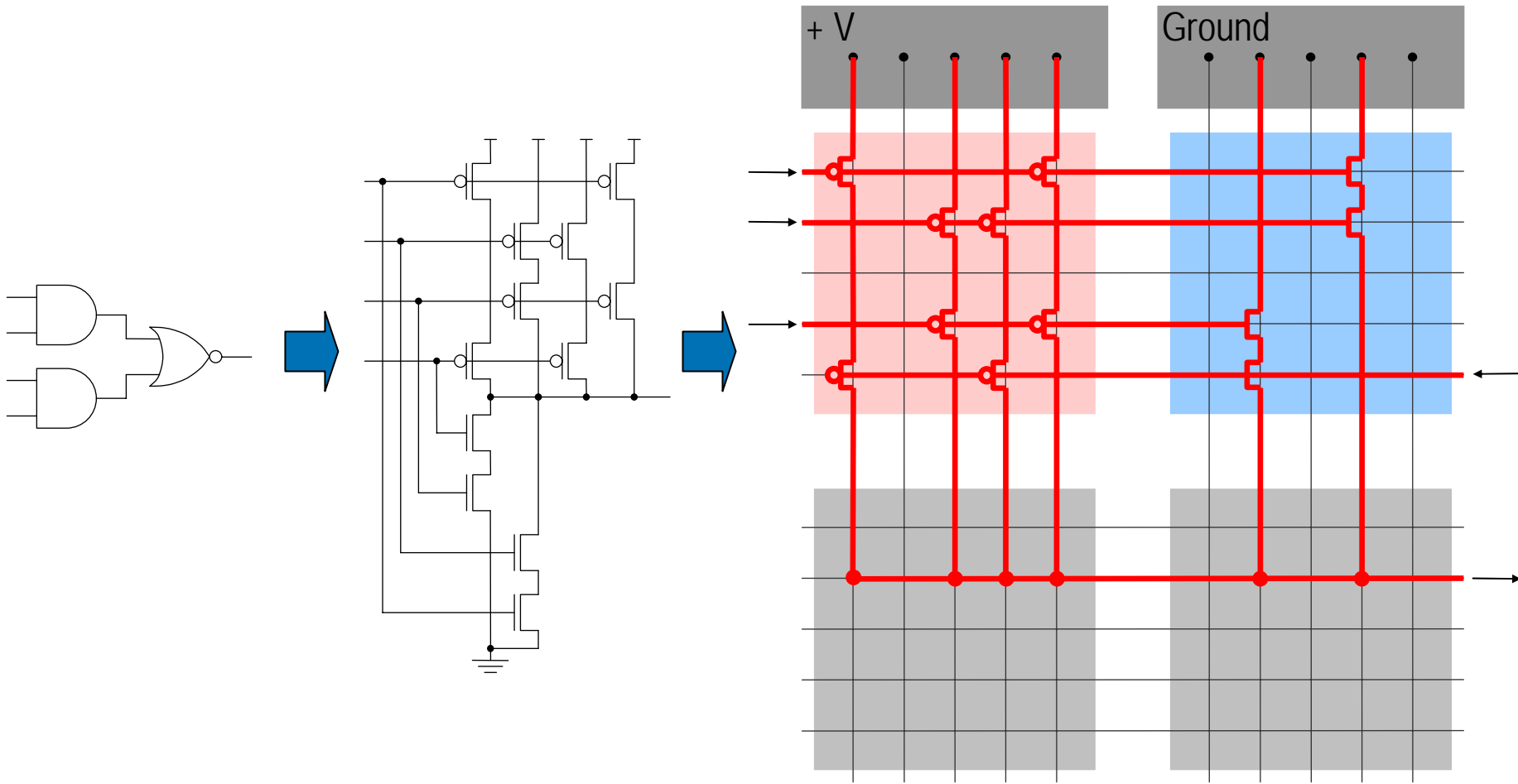
Mosaic: p-FET / resistor logic



Mosaic: Diode / resistor Logic



Mosaic: n-FET / p-FET logic



Are Tiles / Mosaics necessary?

- Adds complexity to manufacturing.
- Can we simplify?

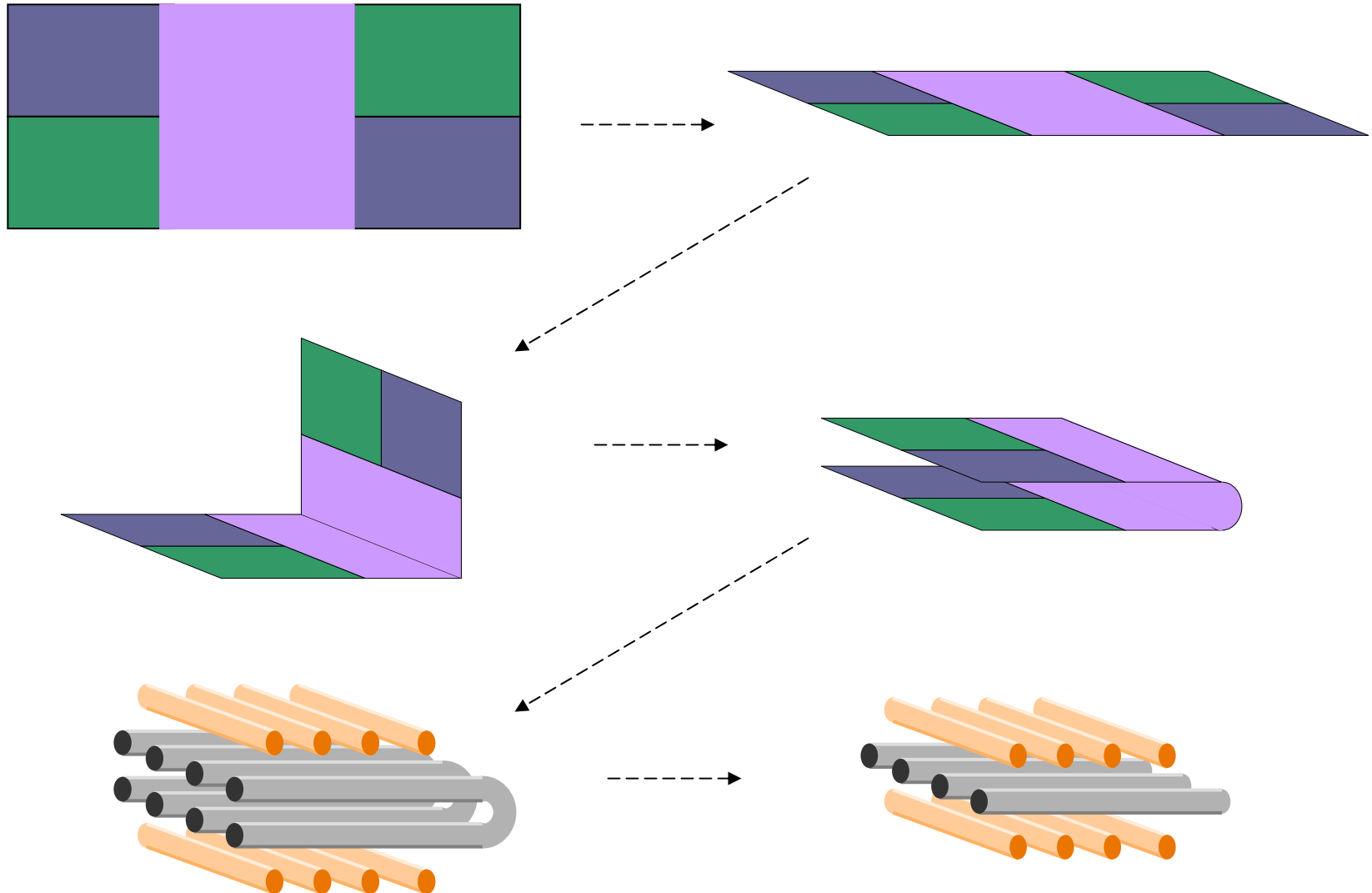
Are Tiles / Mosaics necessary?

- Adds complexity to manufacturing.
- Can we simplify?

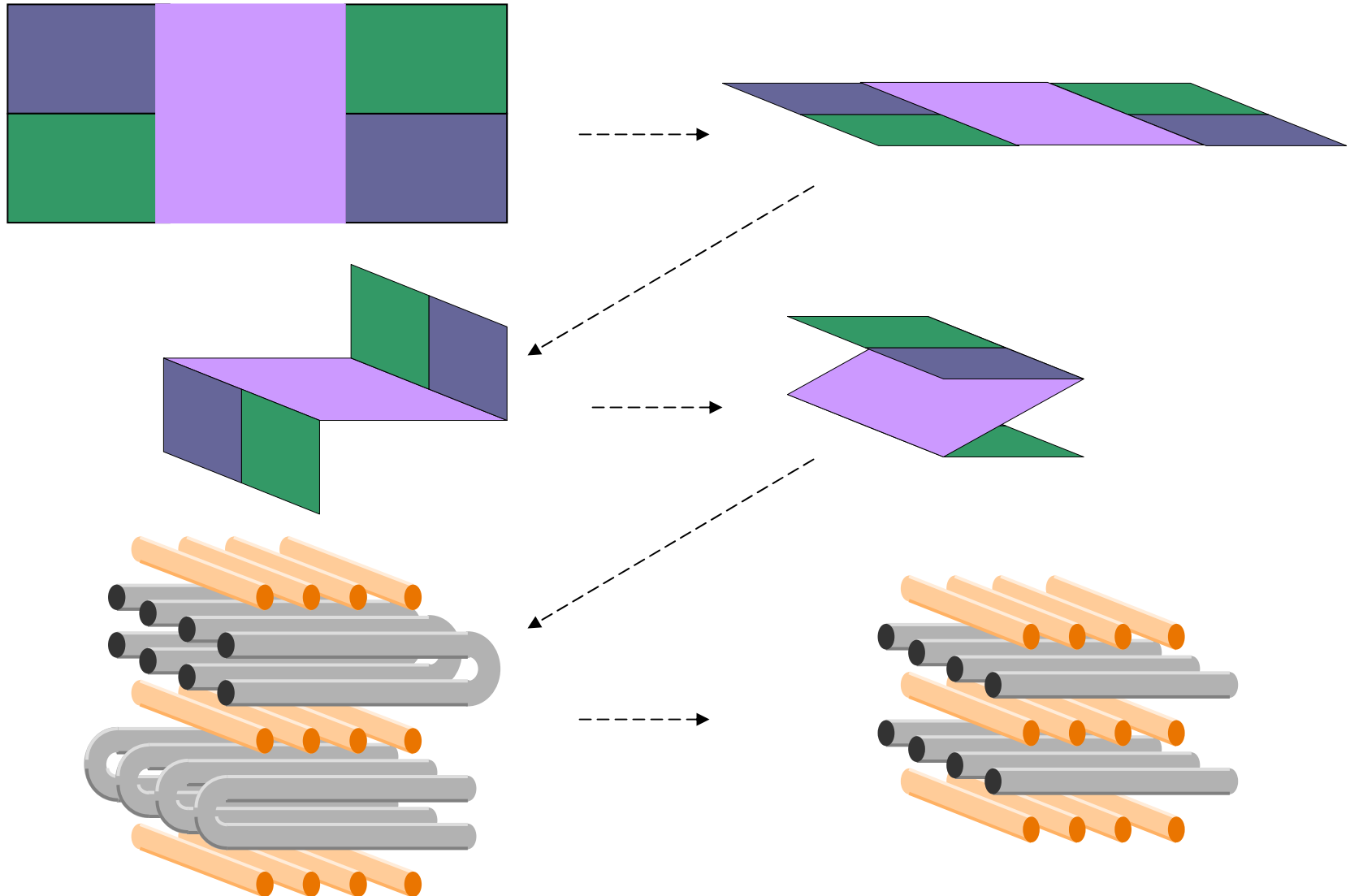
Strategy:

Fold mosaics... *tile* → *layer* !!!

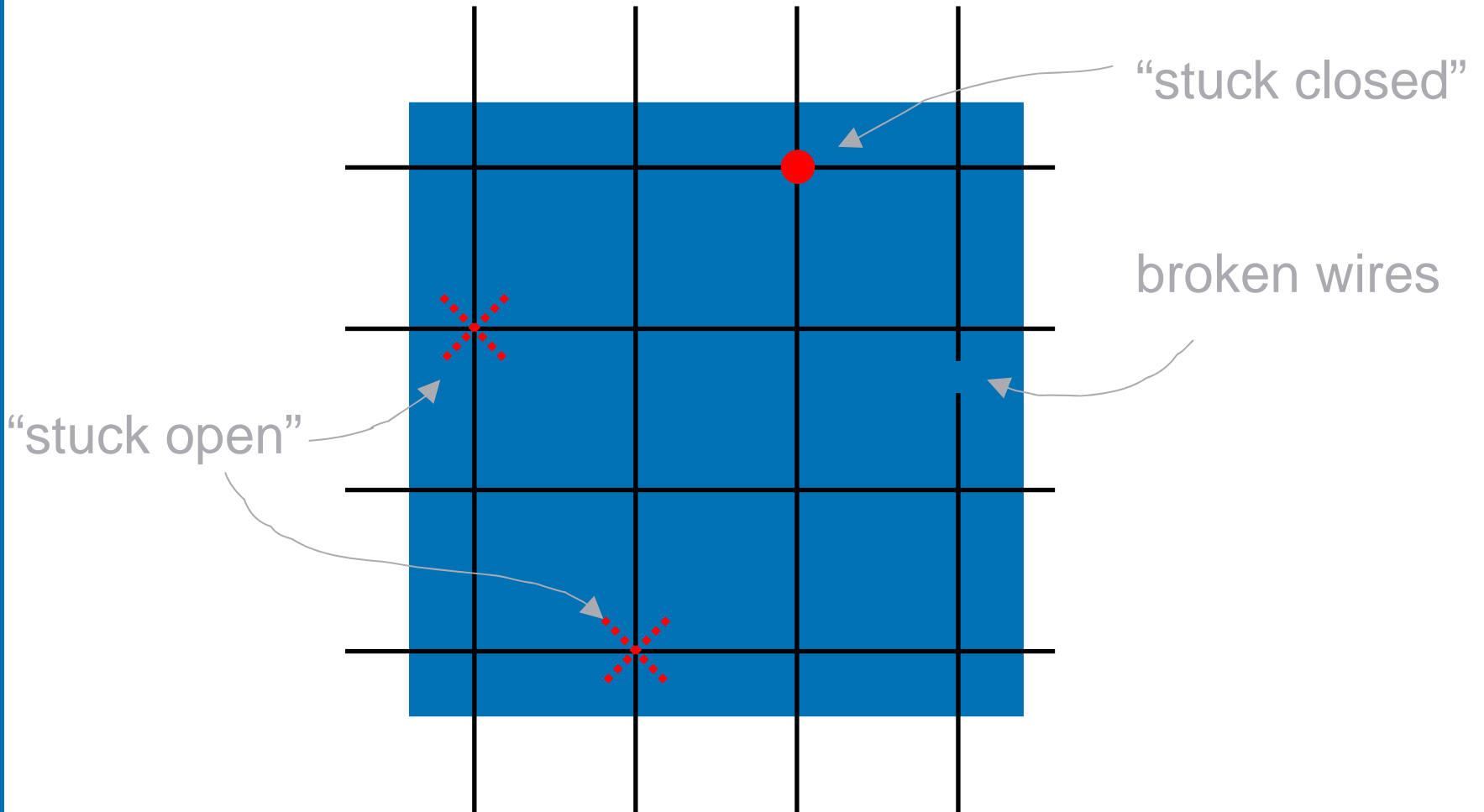
Folded Mosaics



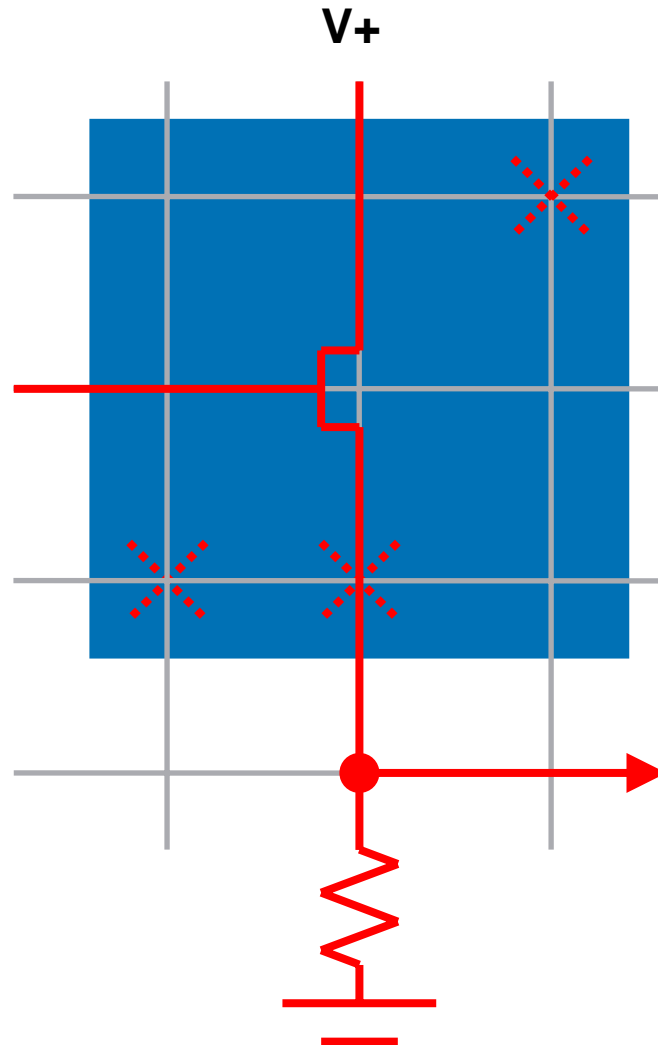
Multi-folded Mosaics



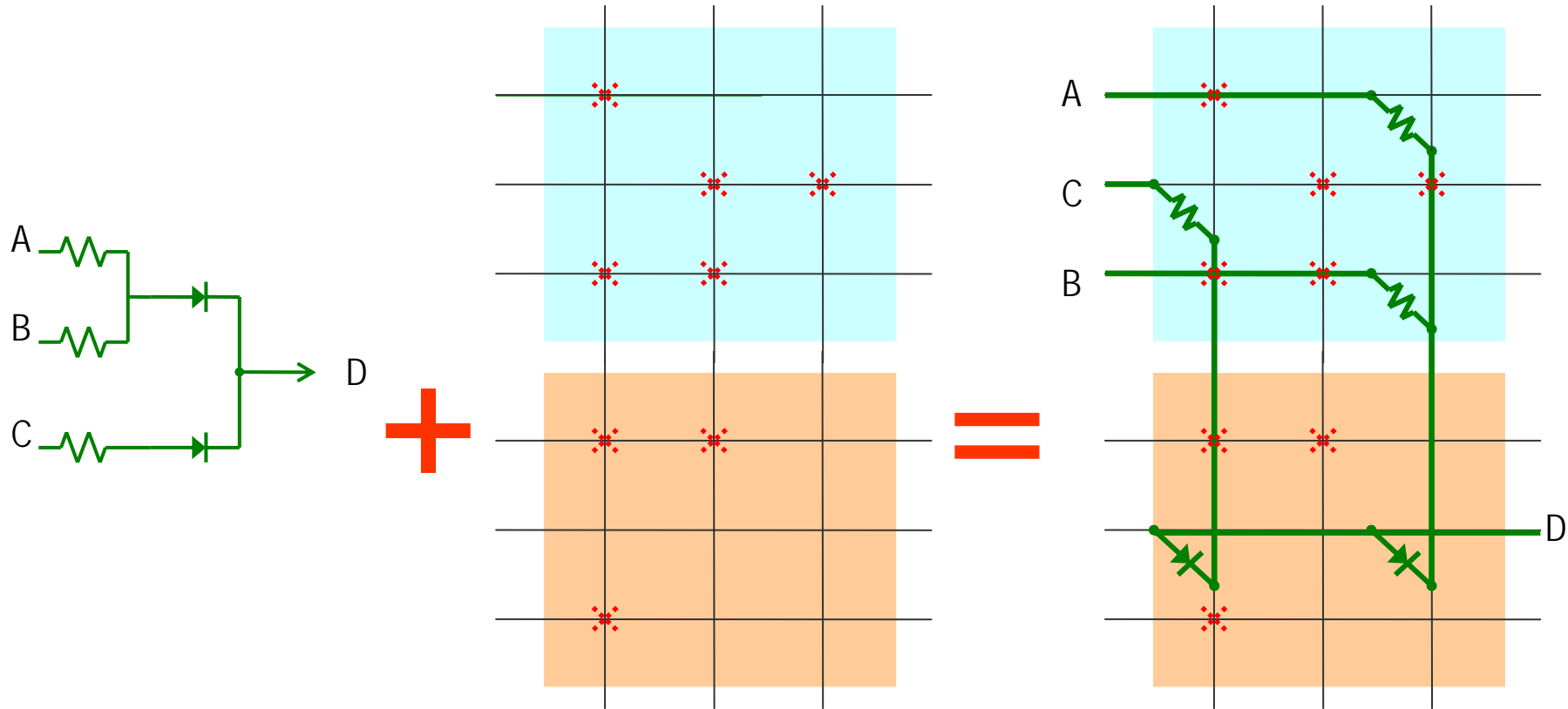
Defects



Defect Avoidance



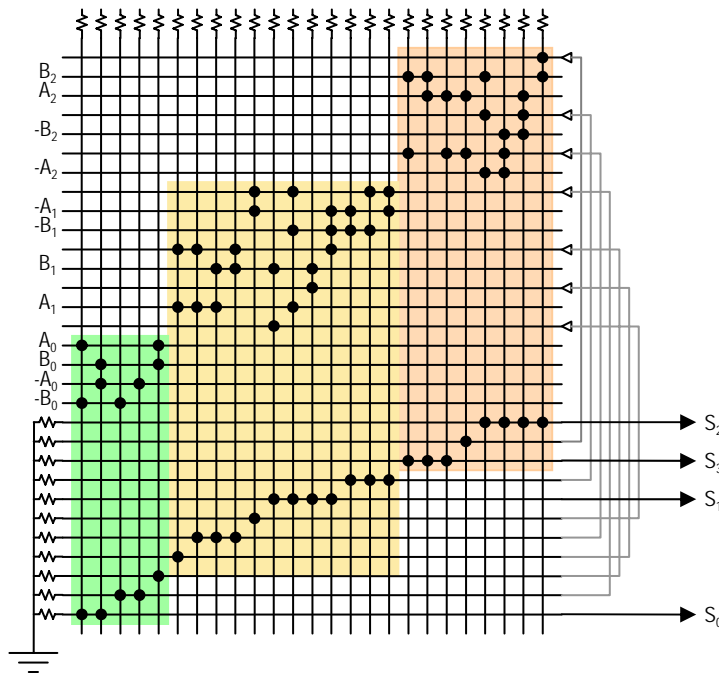
Defect Avoidance as a Resource Allocation Problem



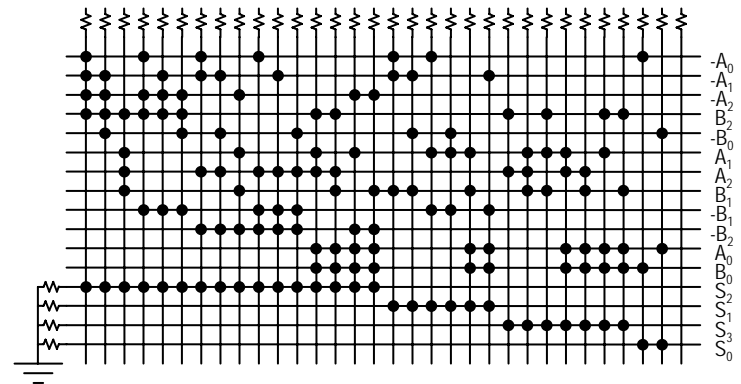
Embedding problem (graph monomorphism)

But, many ways to create circuits...

3-bit adder



Multi-level diode logic



2-level diode logic

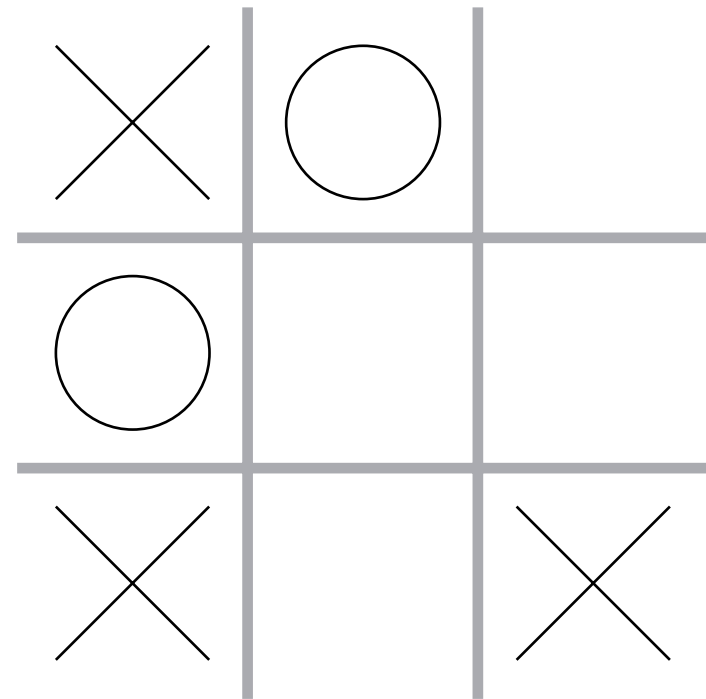
Example: *tic-tac-toe*

```

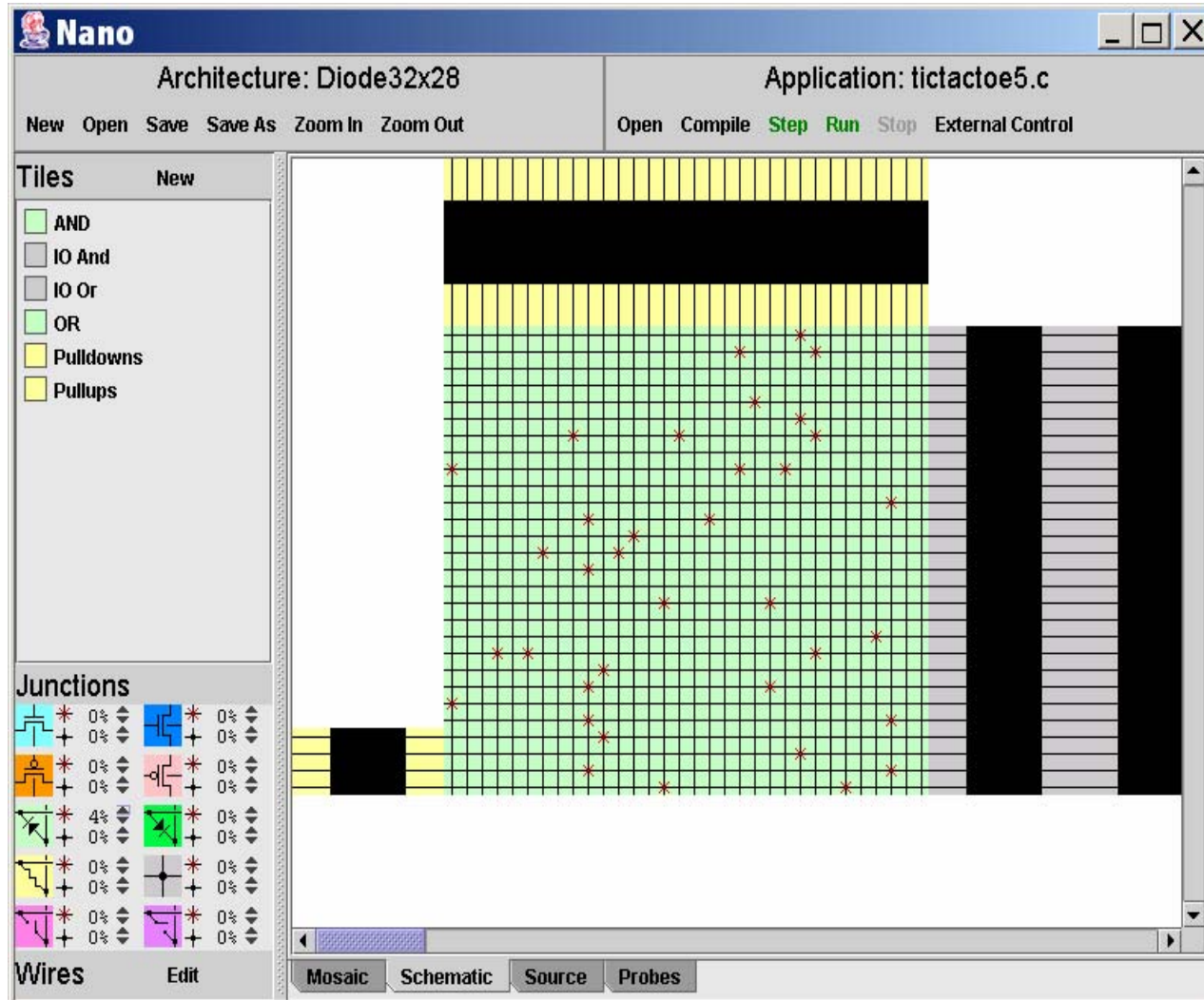
• int game3Response(int moveNumber,
• int humanMove)
• {
• int response;
• if (moveNumber == 1)
• response = I;
• else if (moveNumber == 2) {
• if (humanMove == E)
• response = G;
• else
• response = E;
• } else if (moveNumber == 3) {
• if (humanMove == D)
• response = H;
• else
• response = D;
• }
• return response;
• }

```

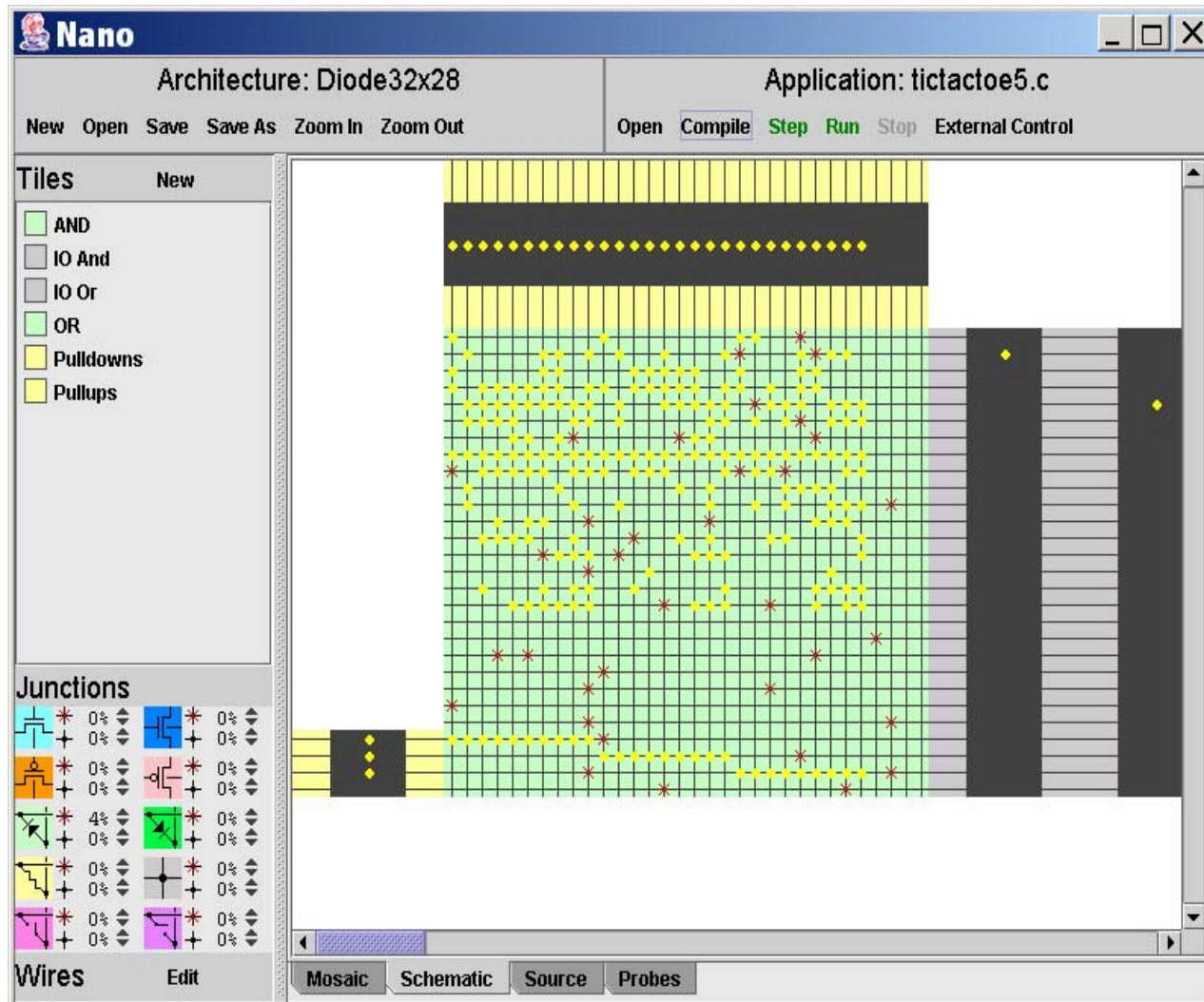
-
-
-



Target: defective diode crossbar



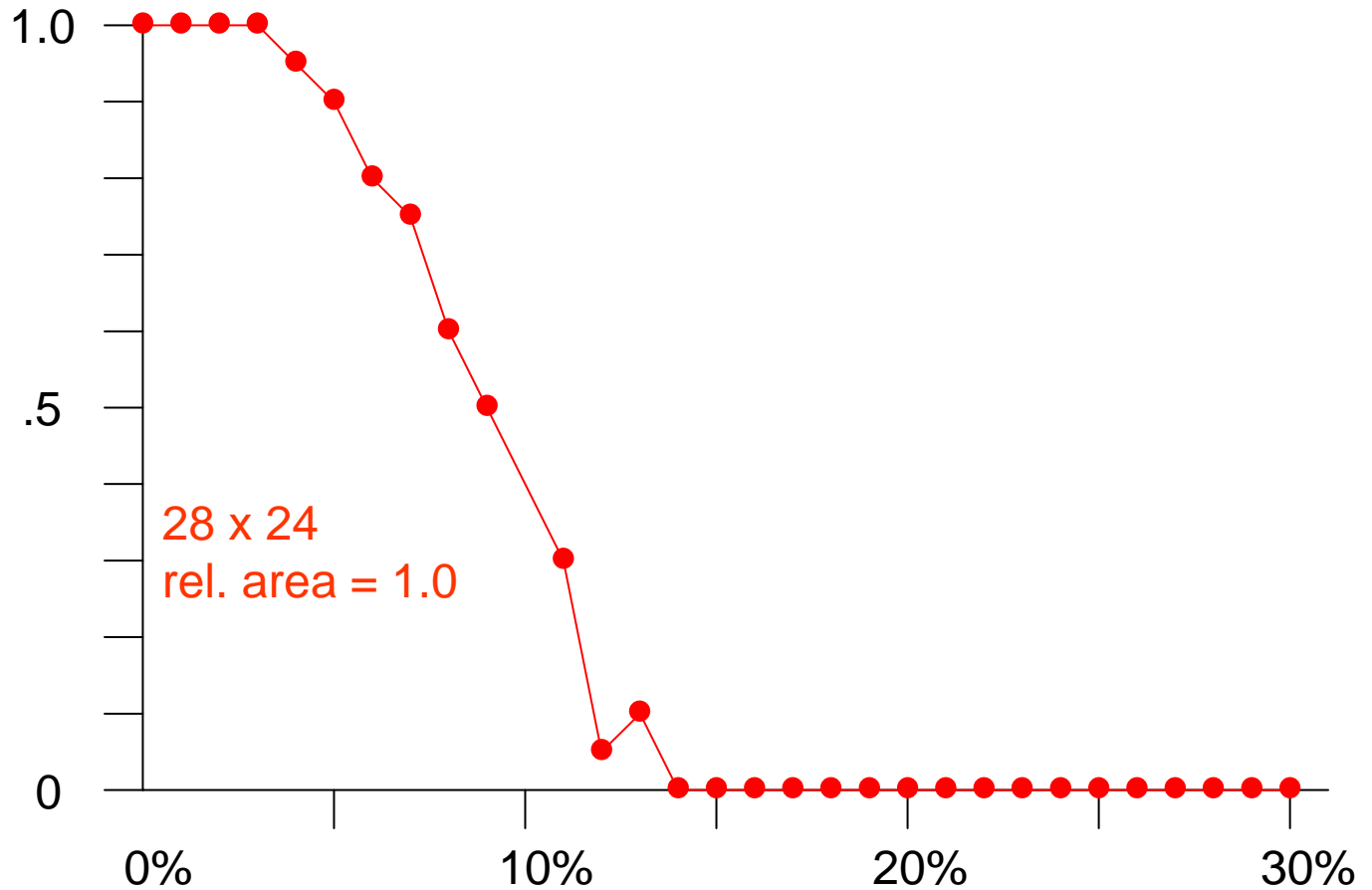
Tic-tac-toe compiled onto target



2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

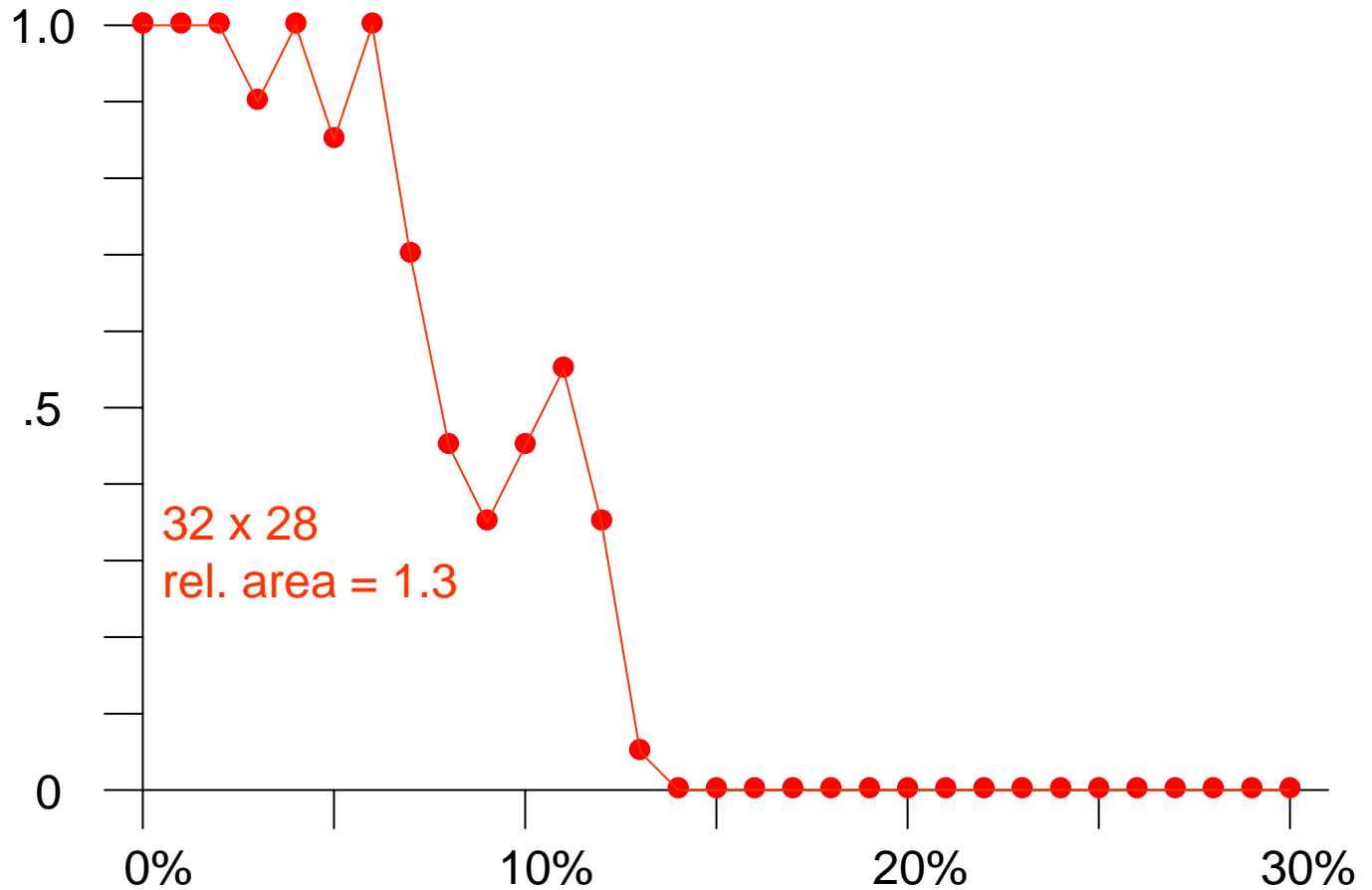


Defective junctions (stuck open)

2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

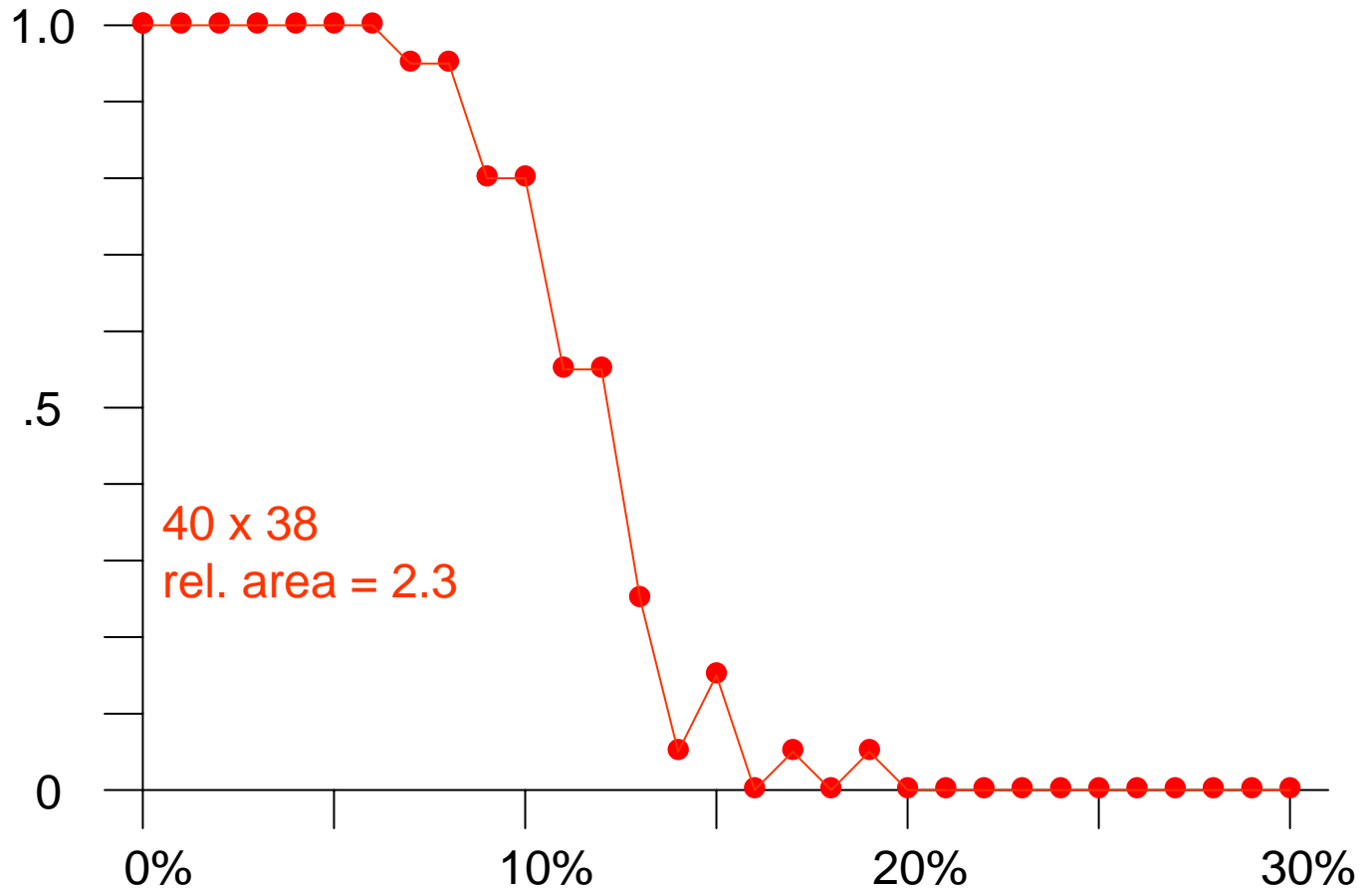


Defective junctions (stuck open)

2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

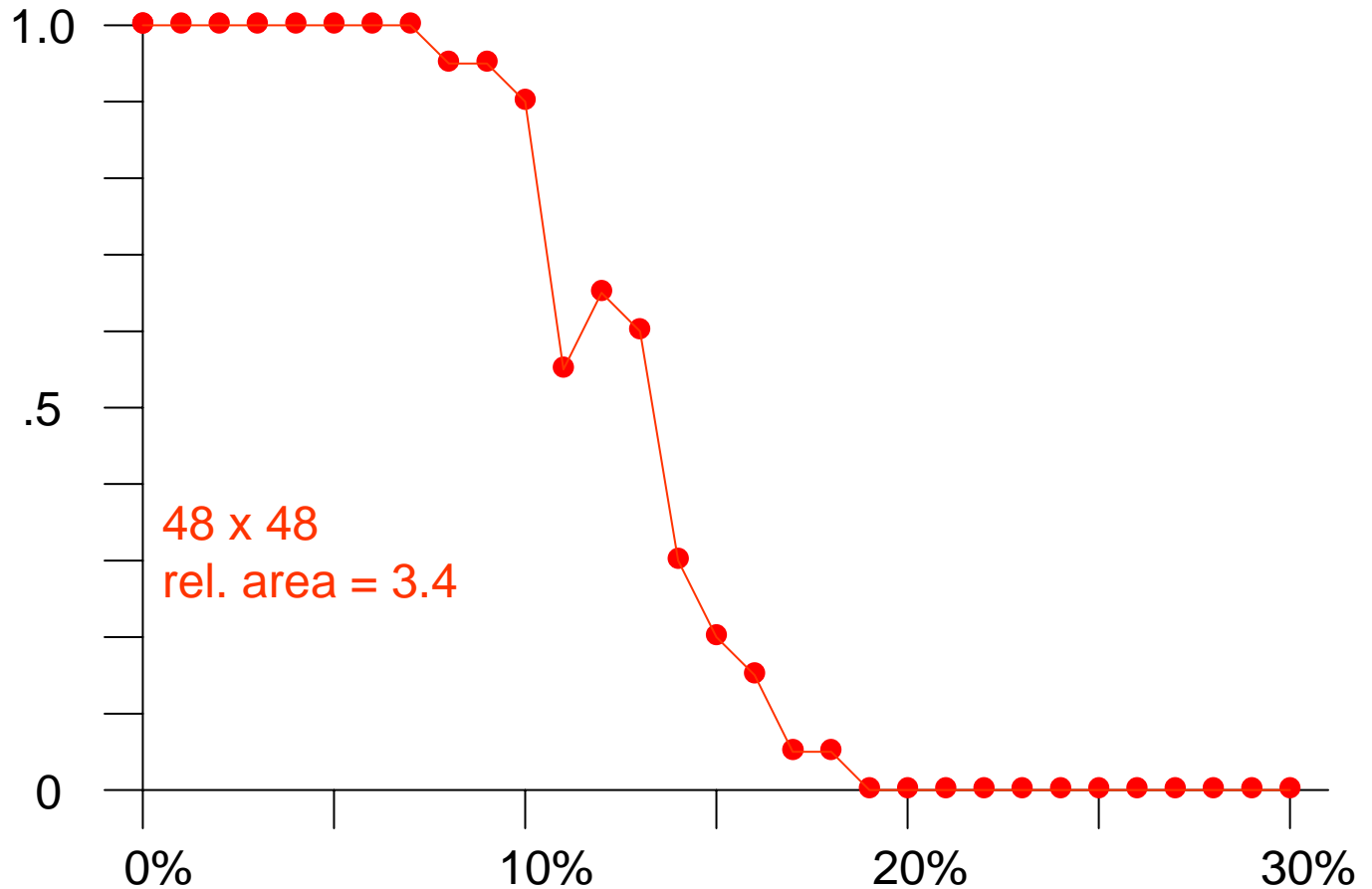


Defective junctions (stuck open)

2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

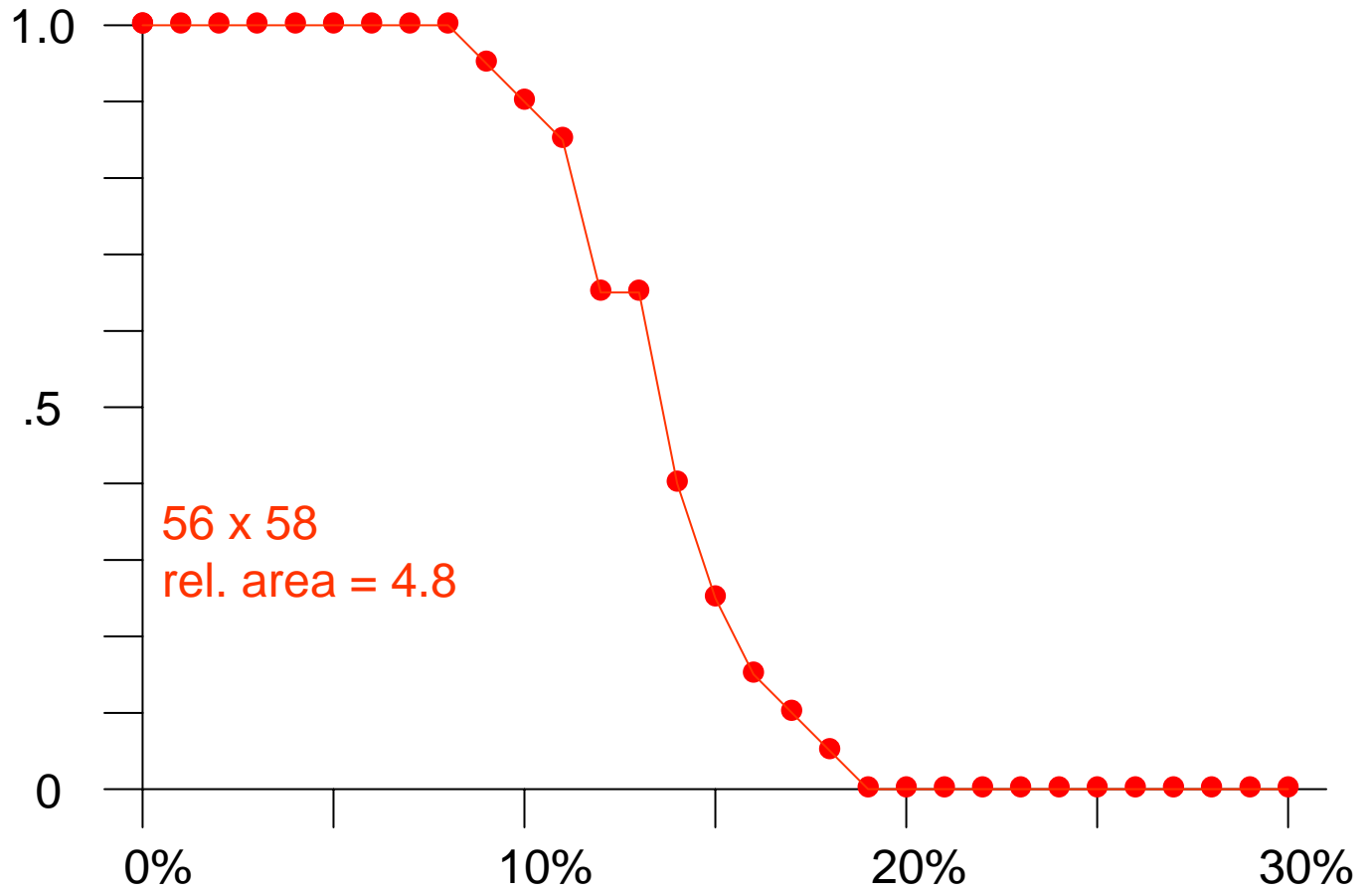


Defective junctions (stuck open)

2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

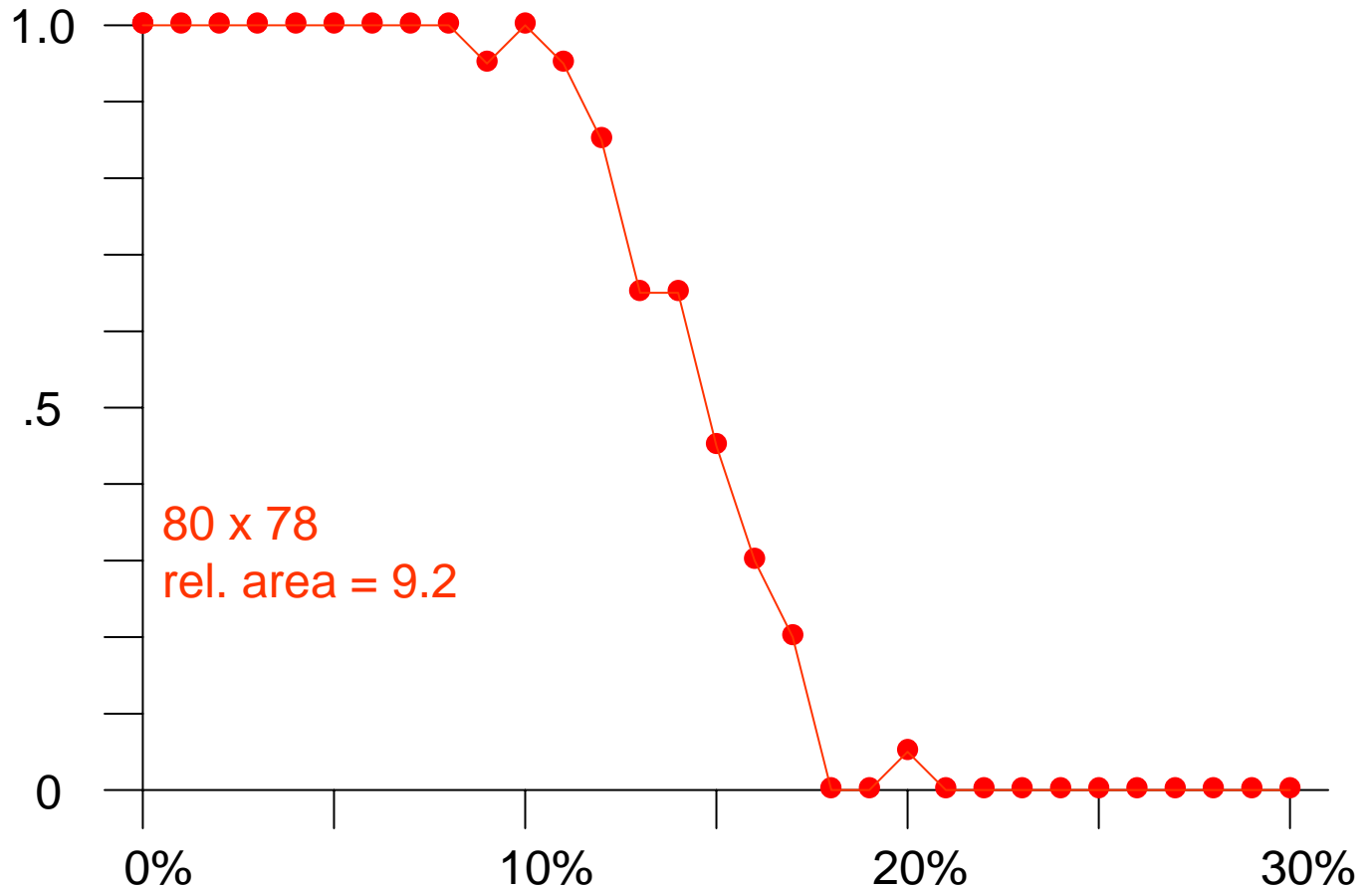


Defective junctions (stuck open)

2-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

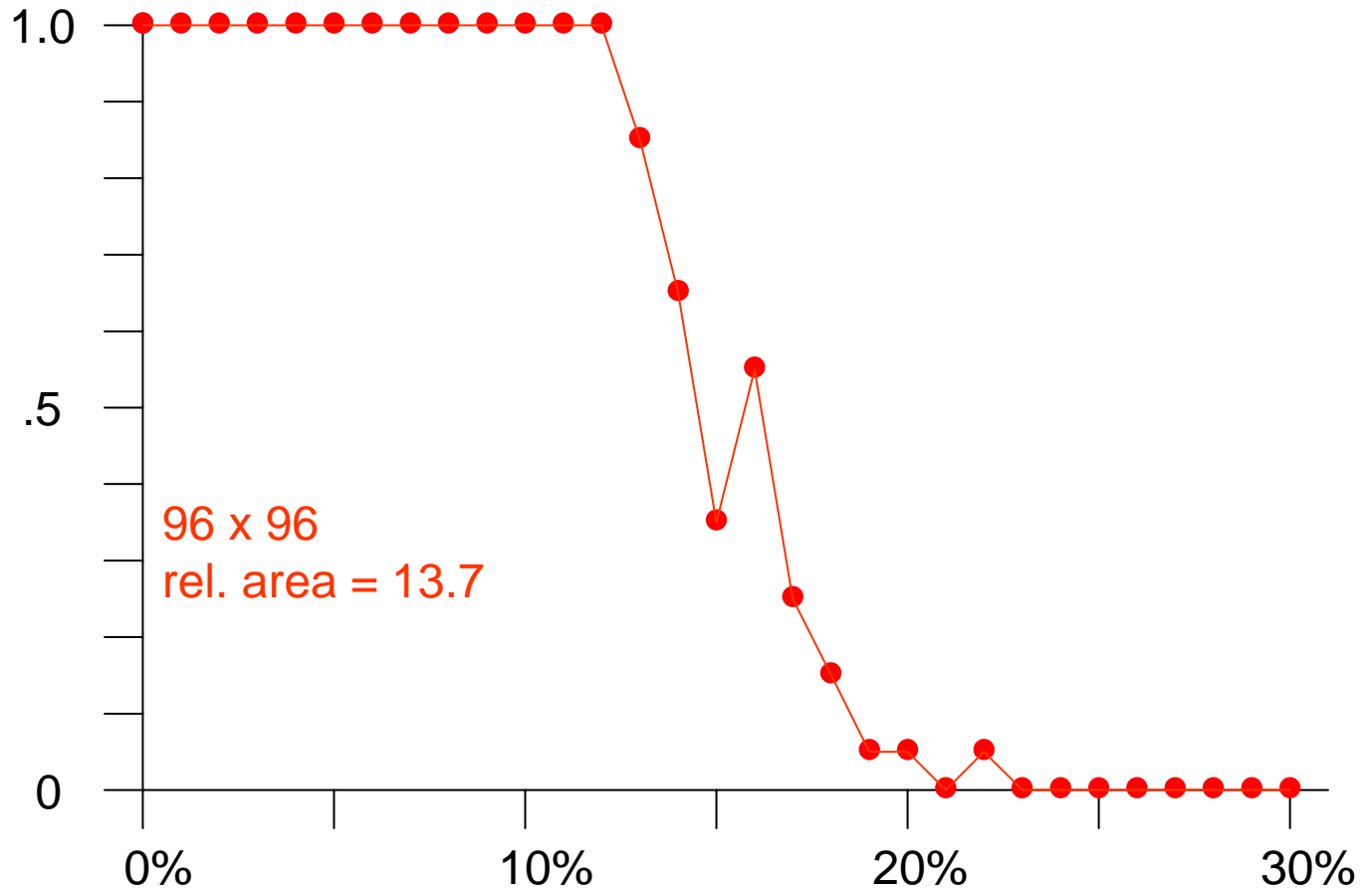


Defective junctions (stuck open)

2-level logic

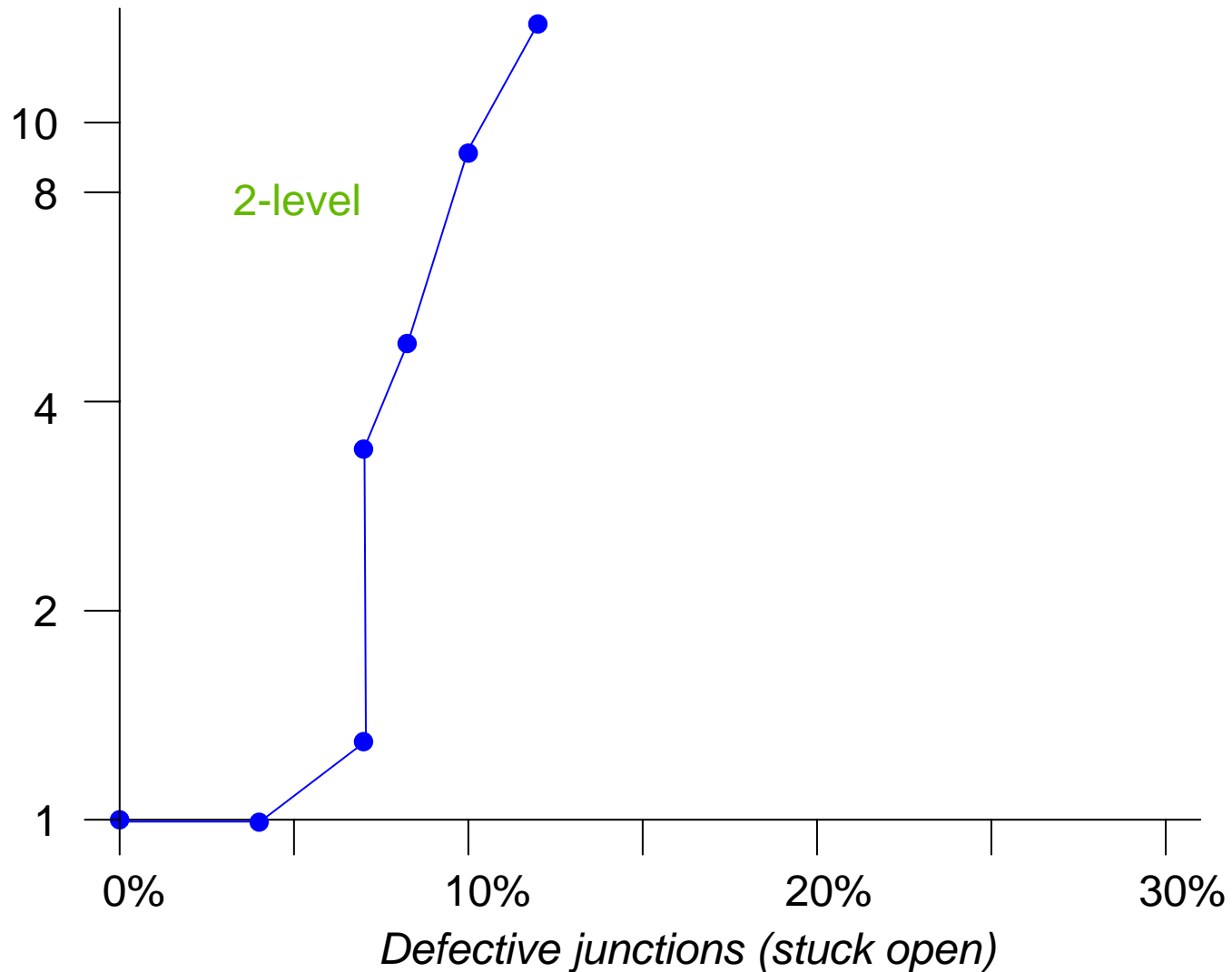
*Prob. of
successful
allocation*

(20
compiles
per point)



Defective junctions (stuck open)

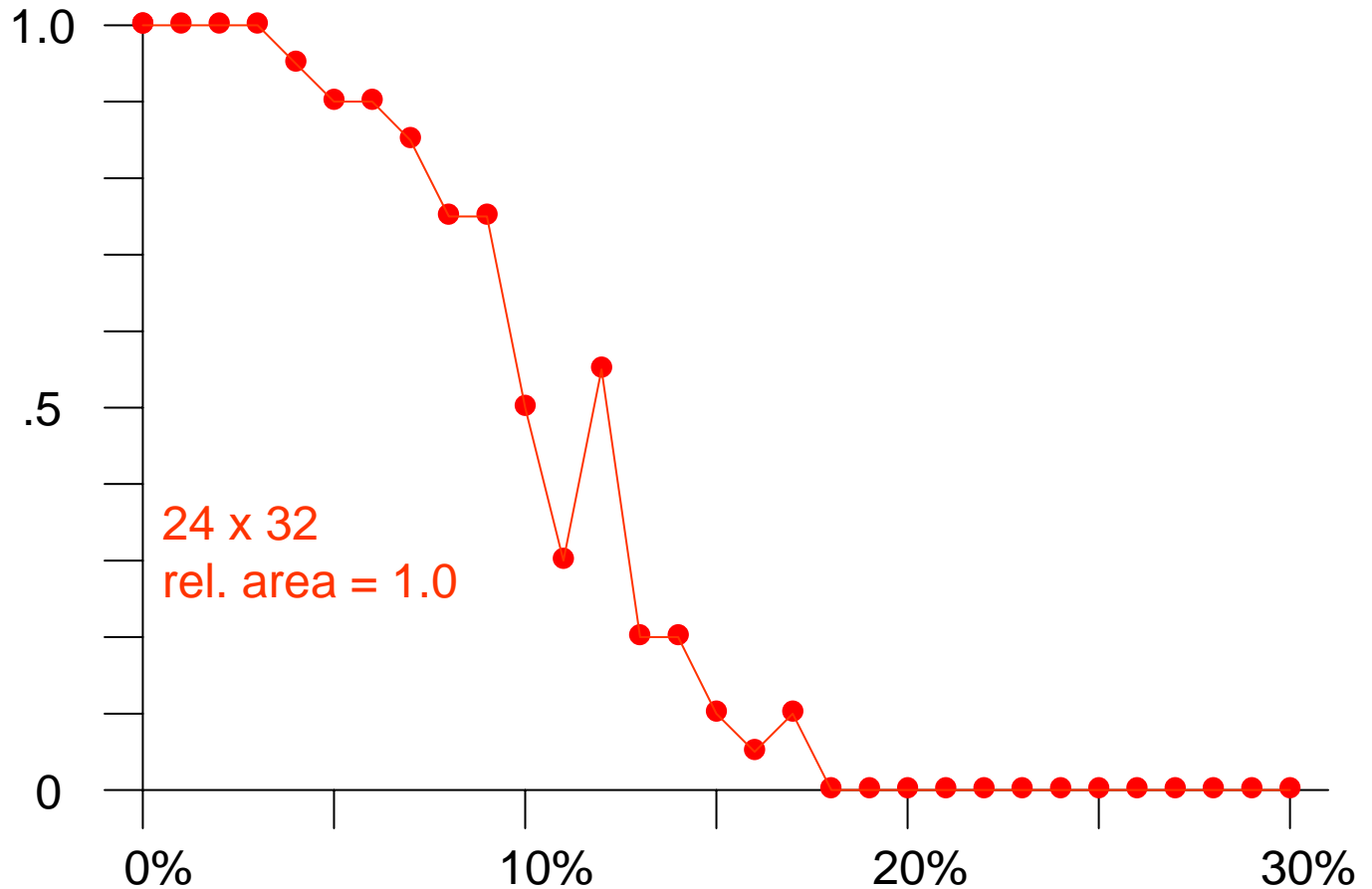
Area = f(defect rate)



multi-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

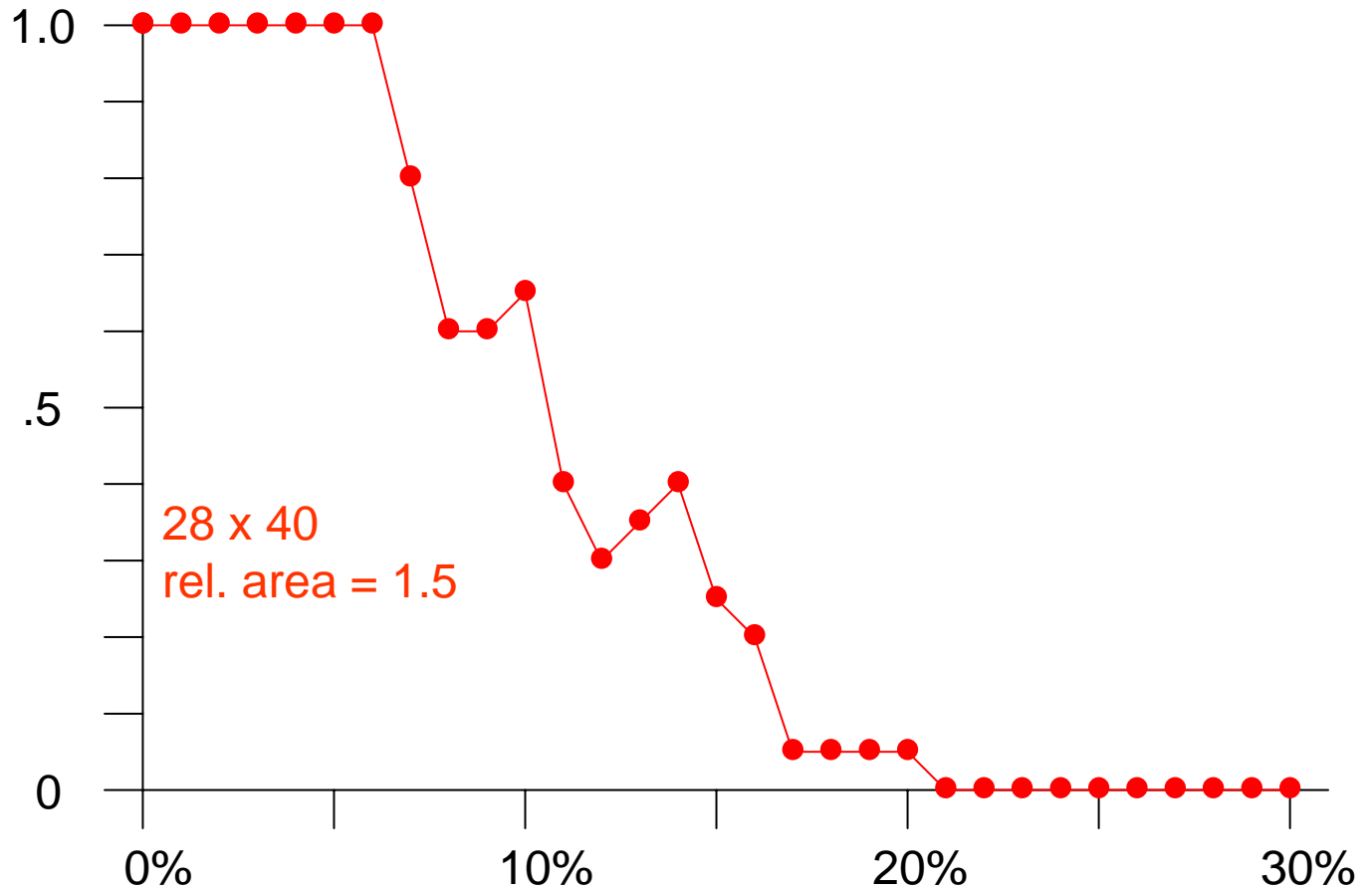


Defective junctions (stuck open)

multi-level logic

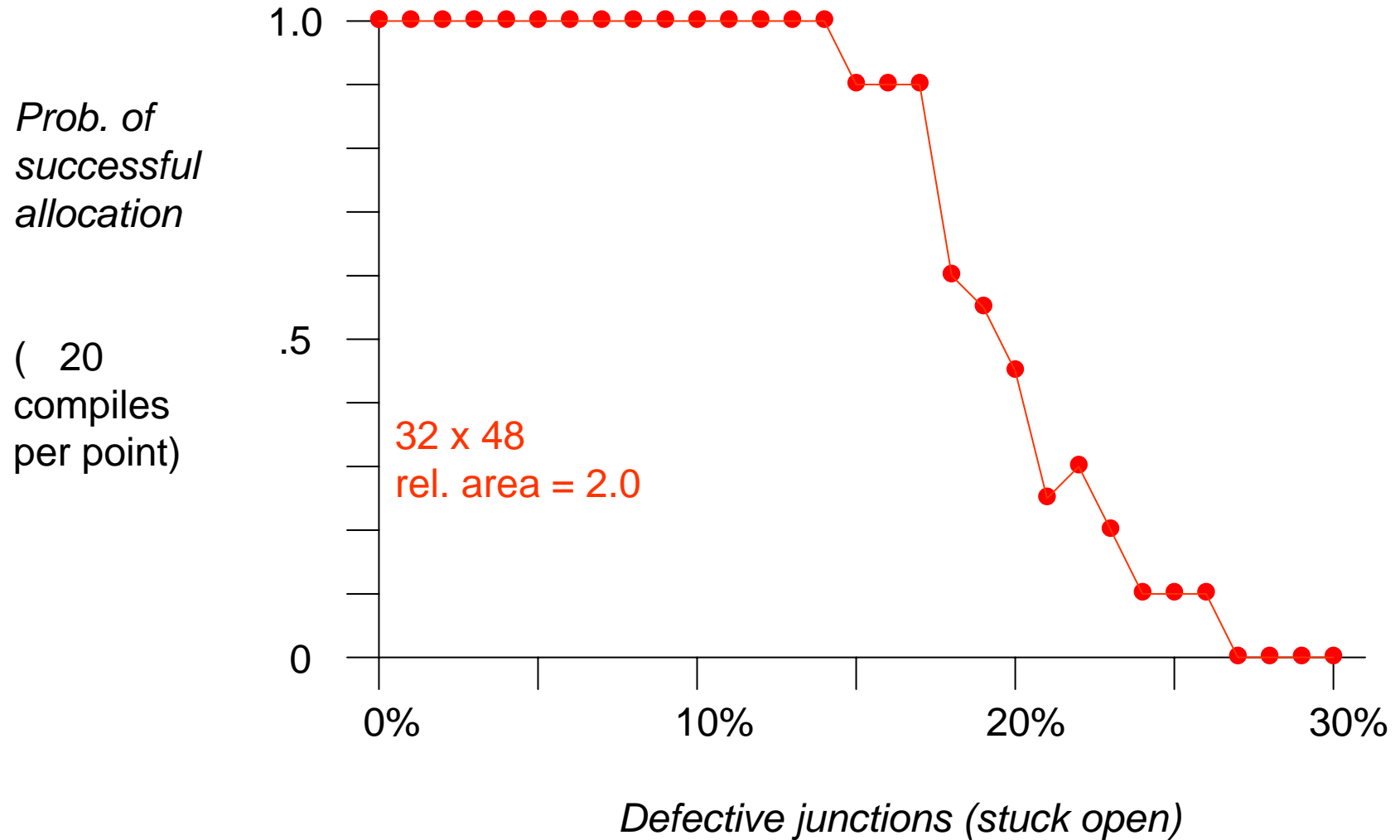
*Prob. of
successful
allocation*

(20
compiles
per point)



Defective junctions (stuck open)

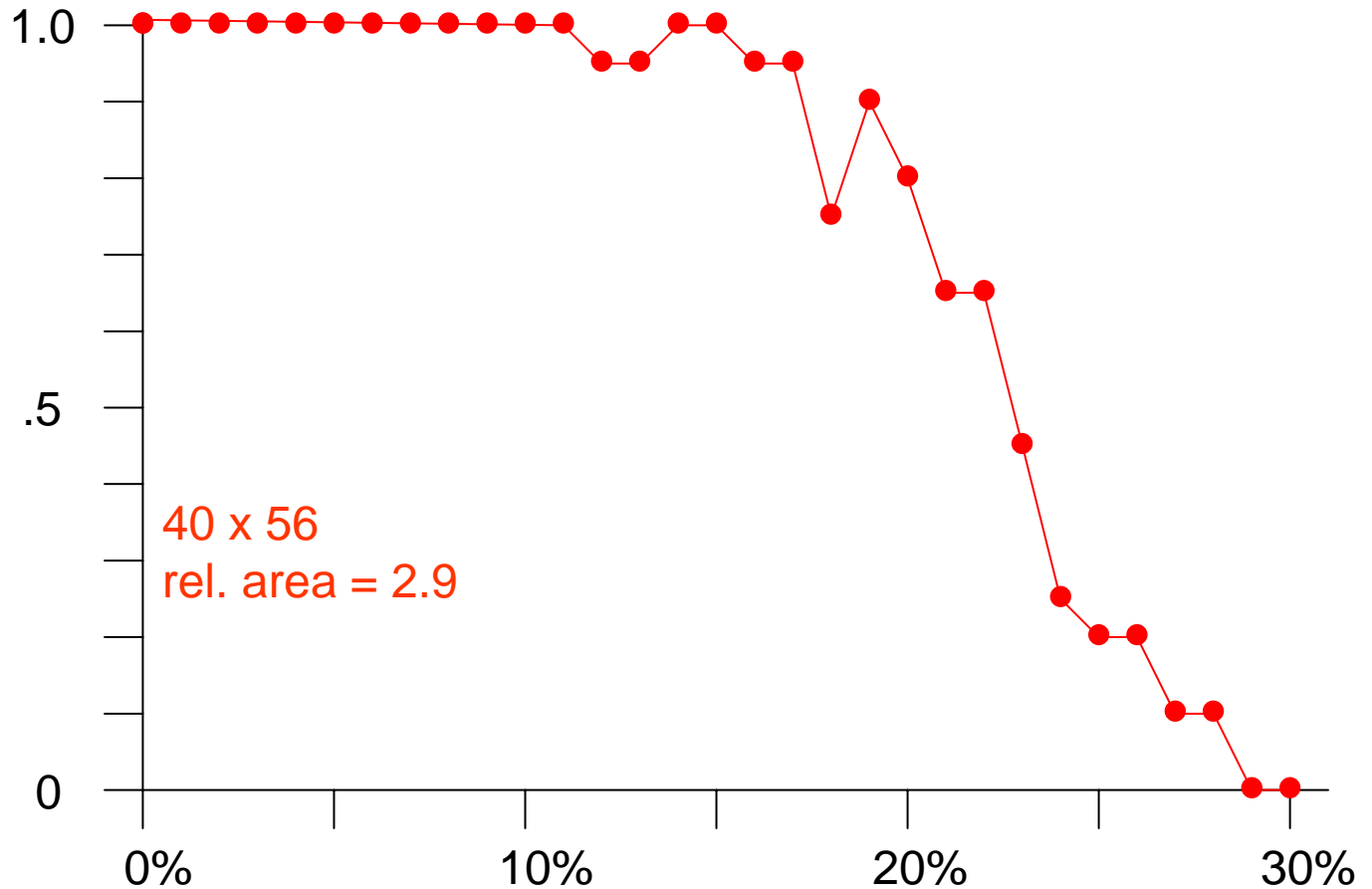
multi-level logic



multi-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

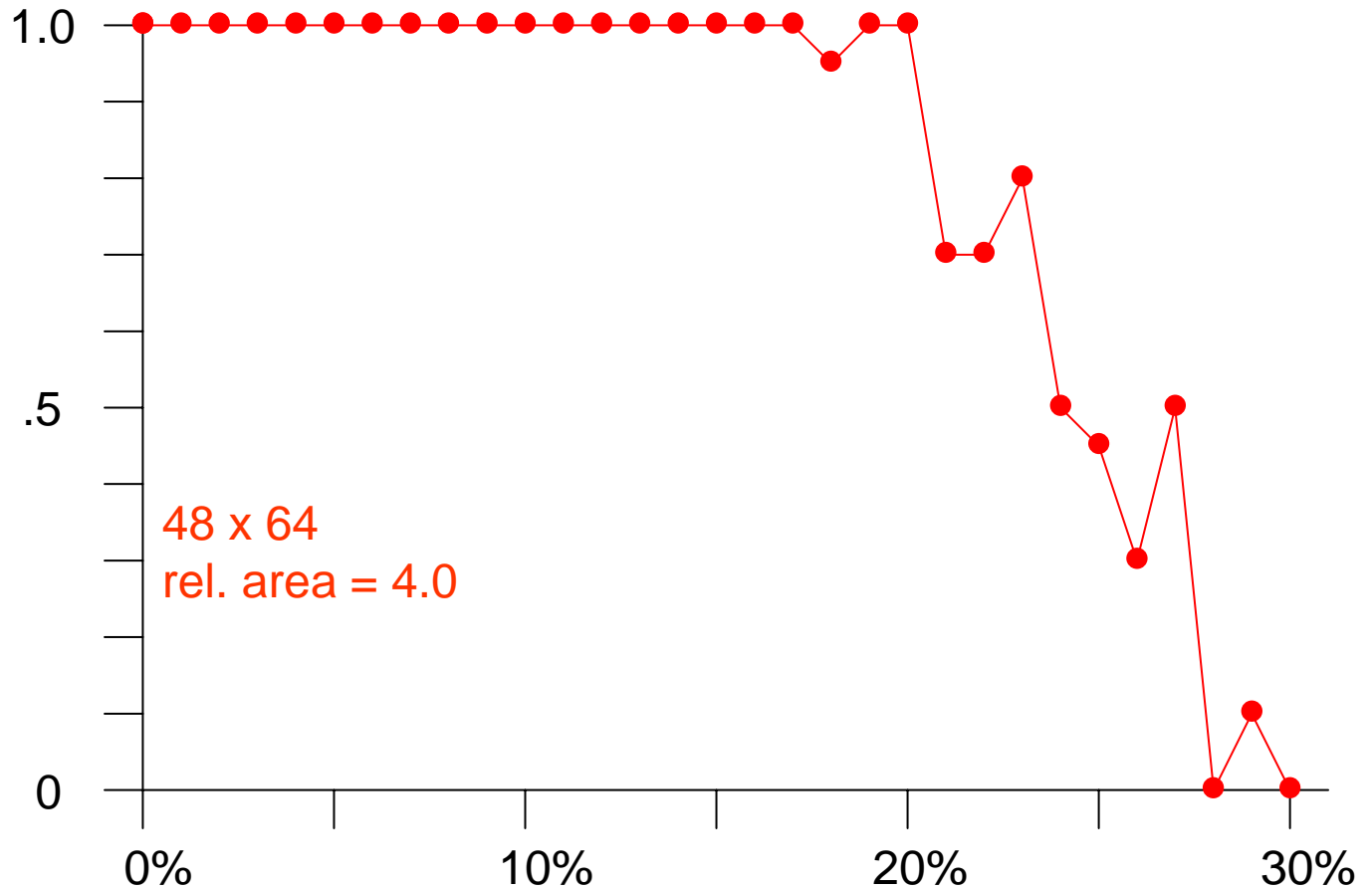


Defective junctions (stuck open)

multi-level logic

*Prob. of
successful
allocation*

(20
compiles
per point)

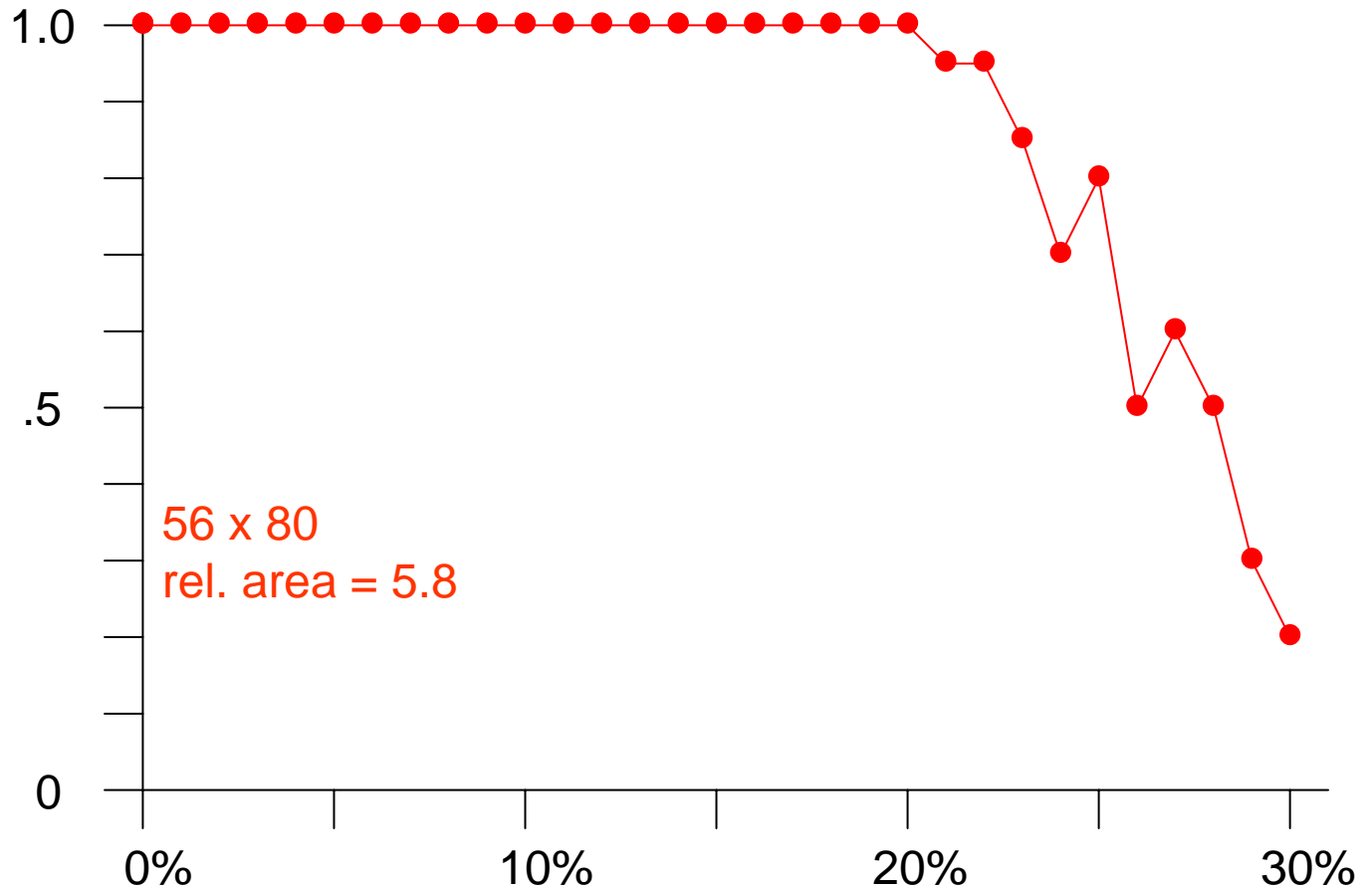


Defective junctions (stuck open)

multi-level logic

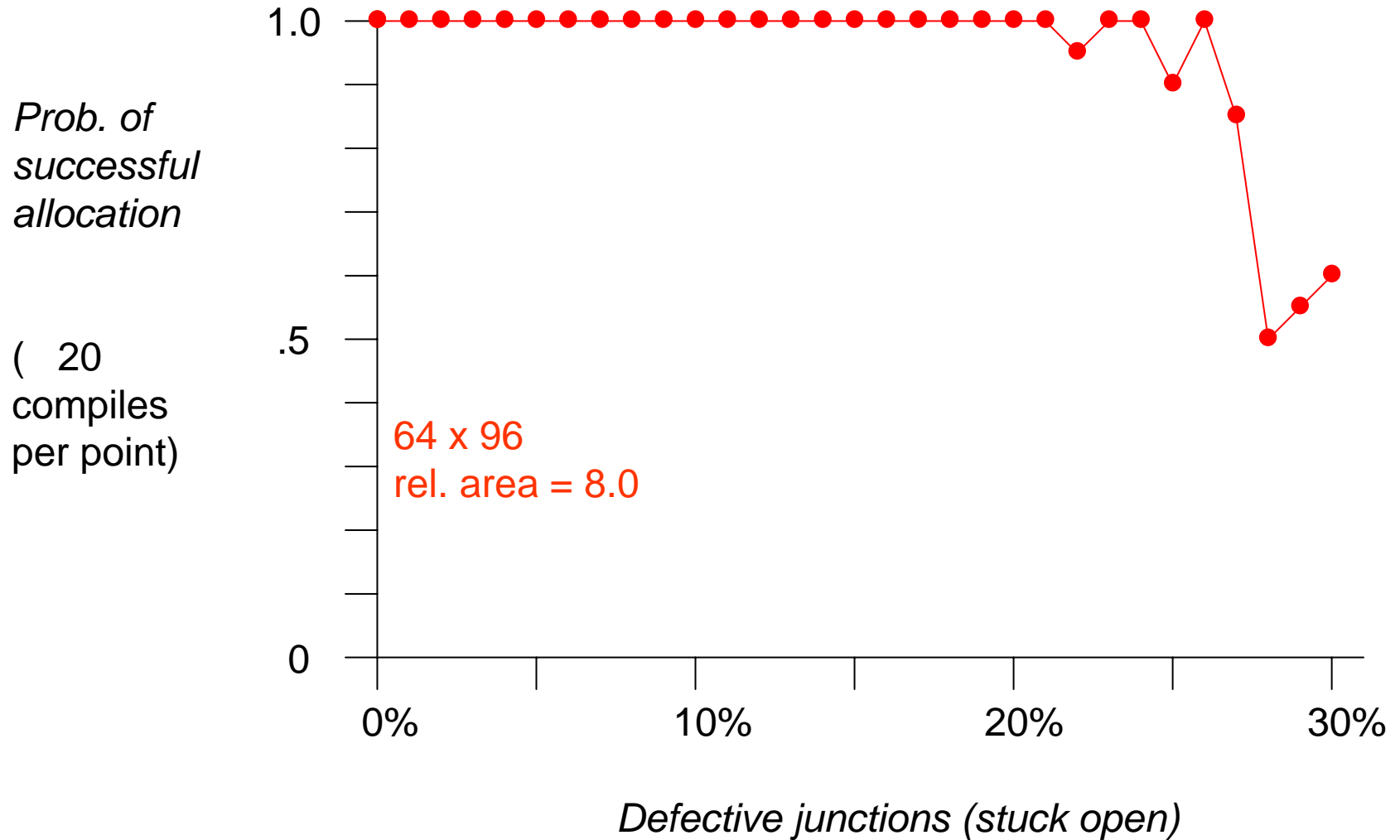
*Prob. of
successful
allocation*

(20
compiles
per point)



Defective junctions (stuck open)

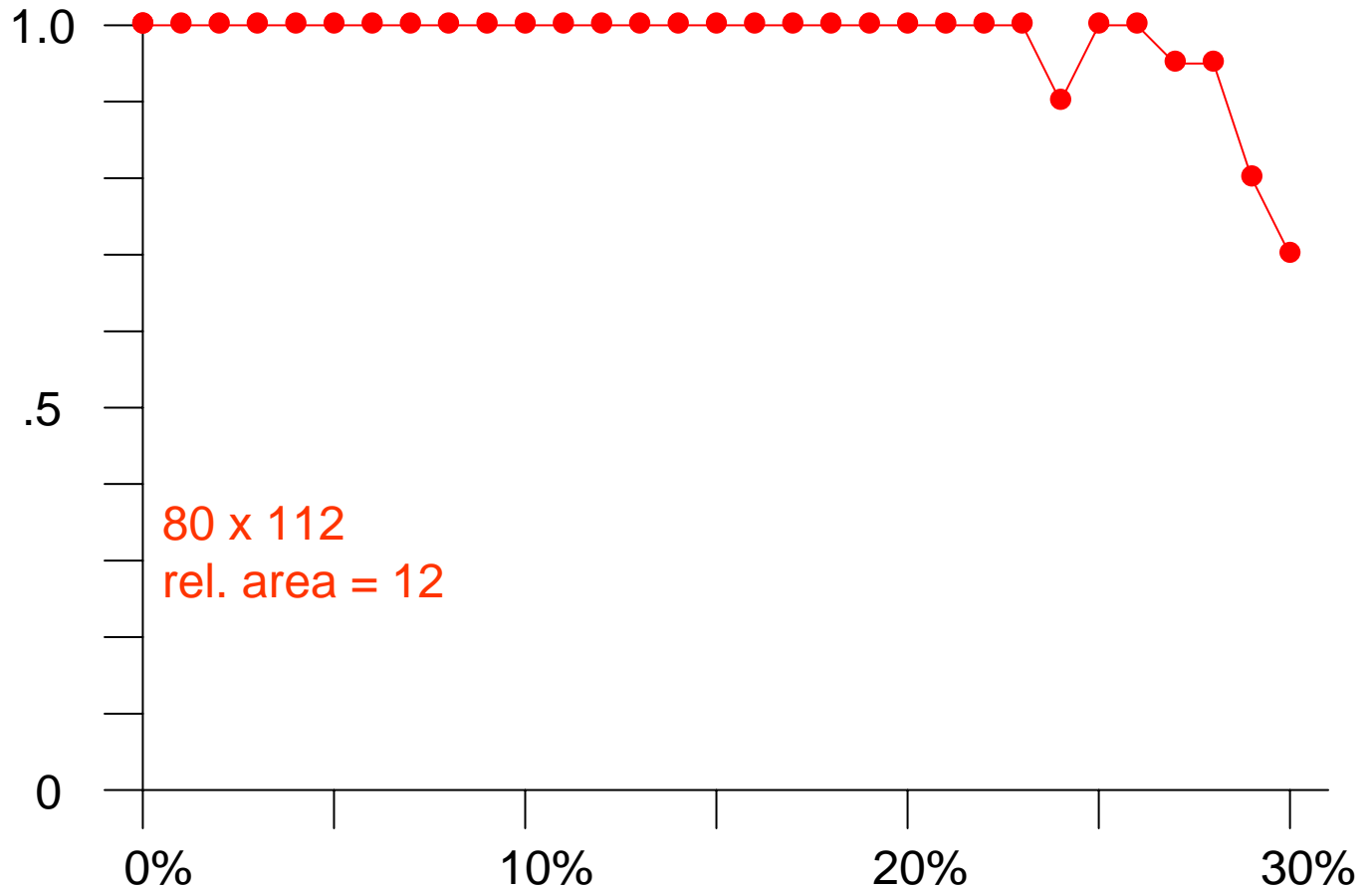
multi-level logic



multi-level logic

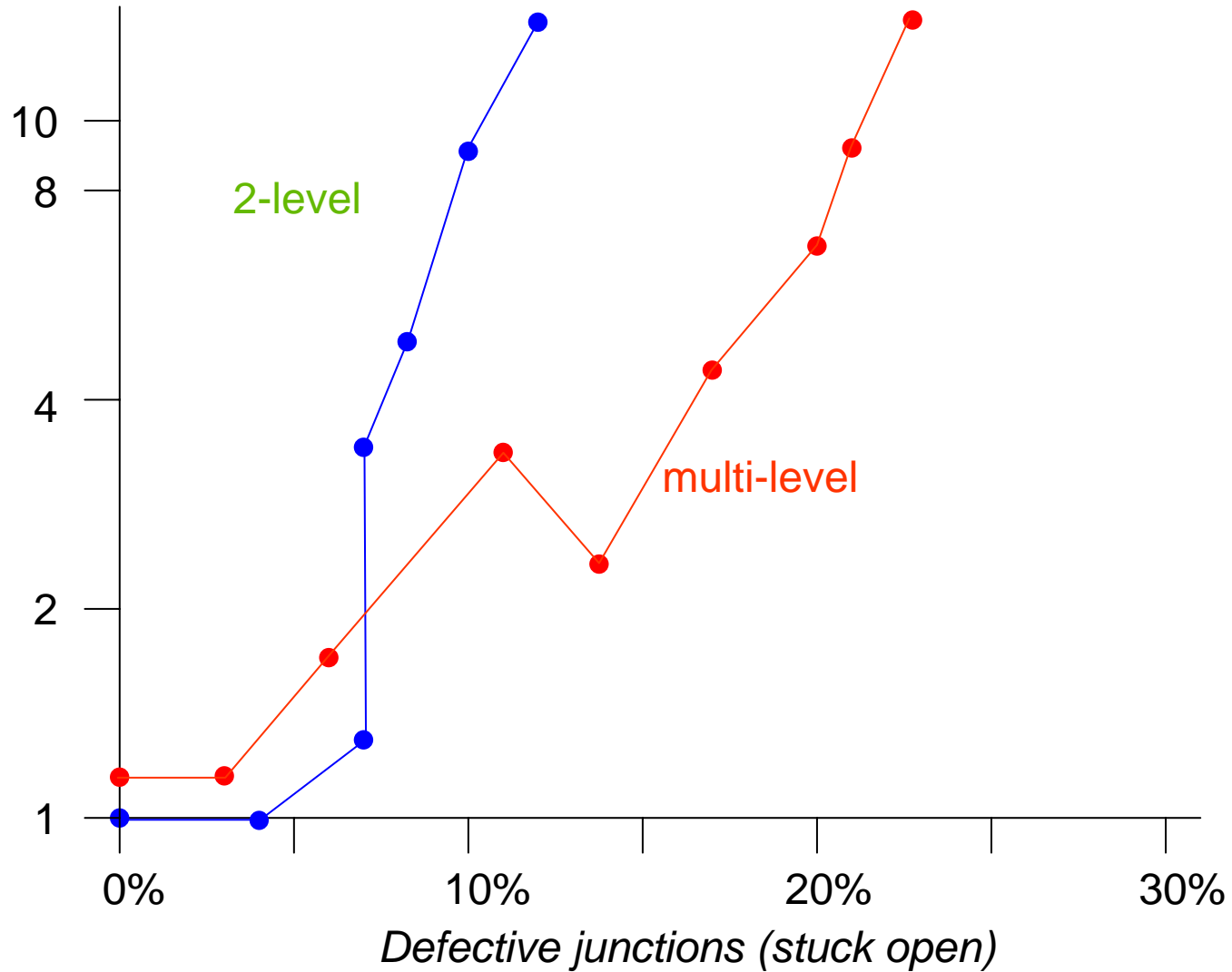
*Prob. of
successful
allocation*

(20
compiles
per point)



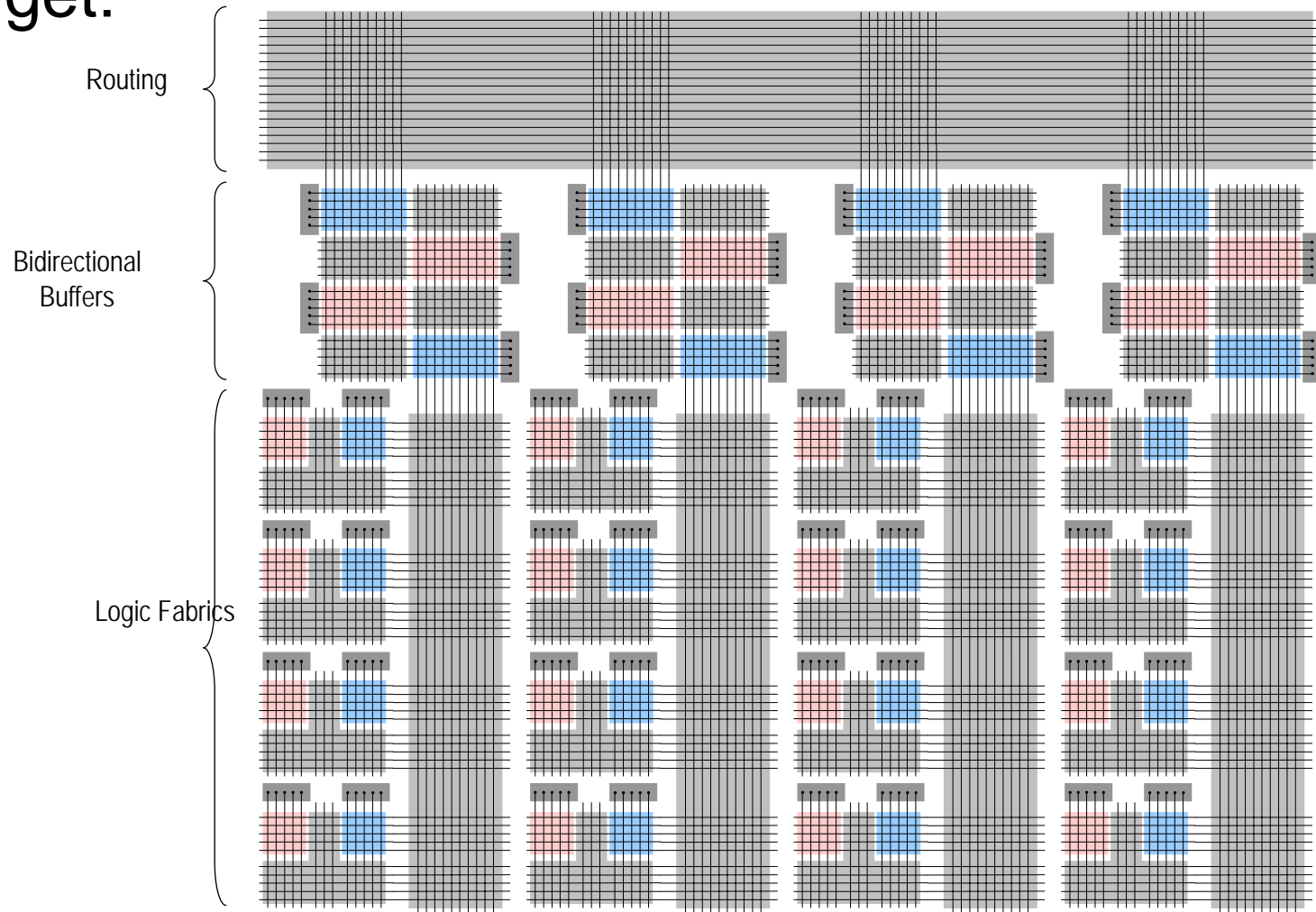
Defective junctions (stuck open)

Area = f(defect rate)

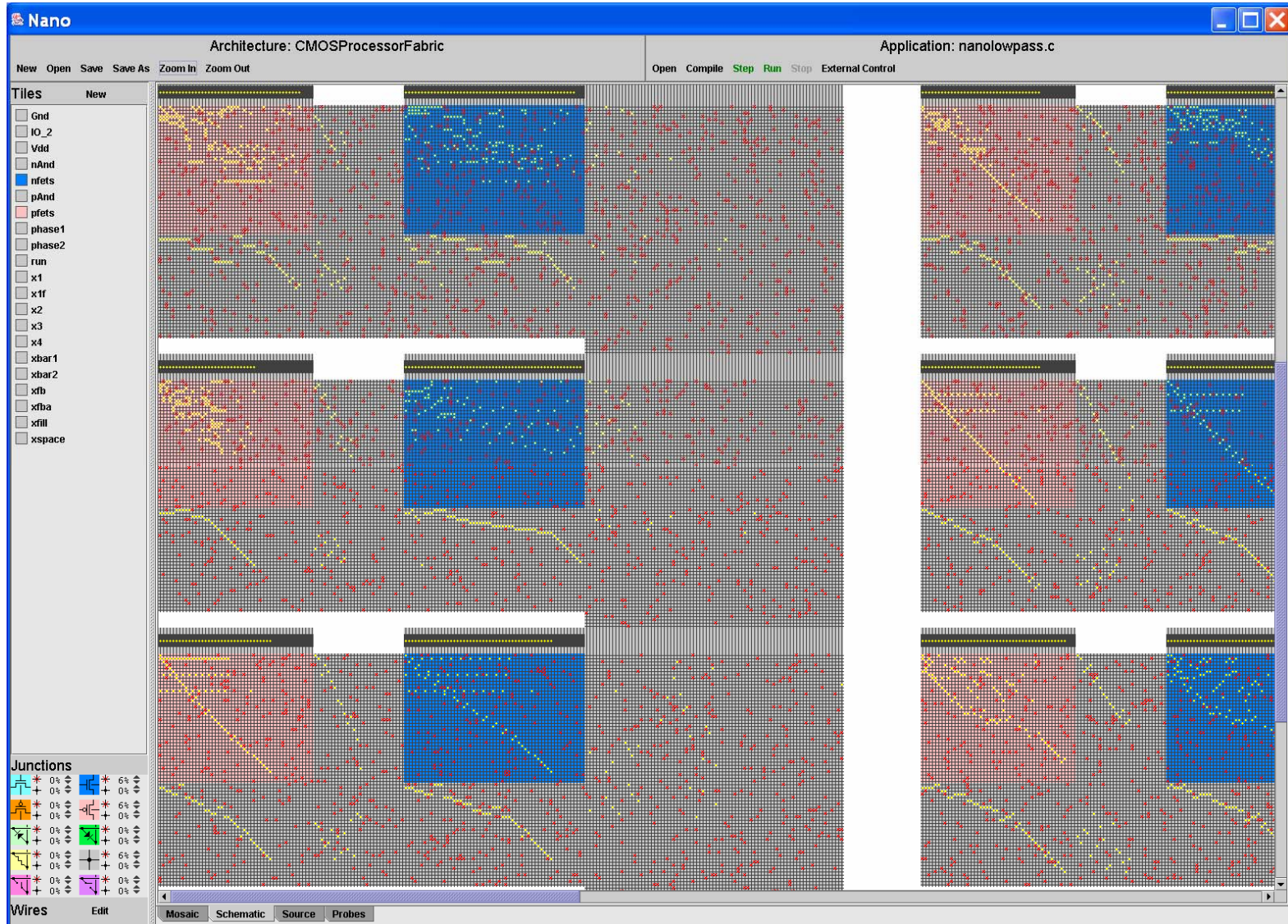


4-bit microprocessor...

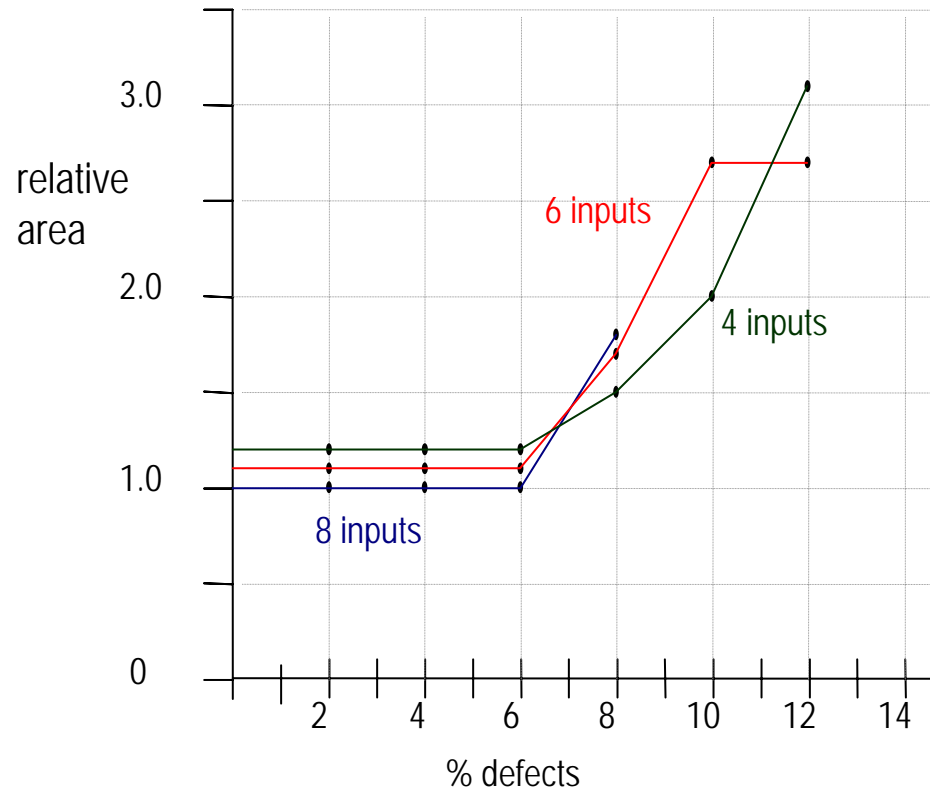
target:



...compiled onto defective mosaic:



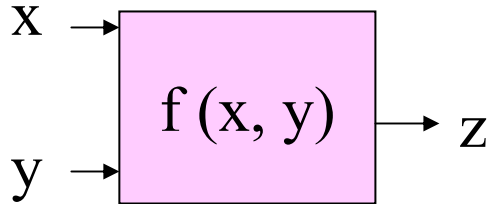
4-bit microprocessor mapping algorithms



Fault Tolerance

- Error correction
- Error detection

Triple Modular Redundancy (von Neumann)

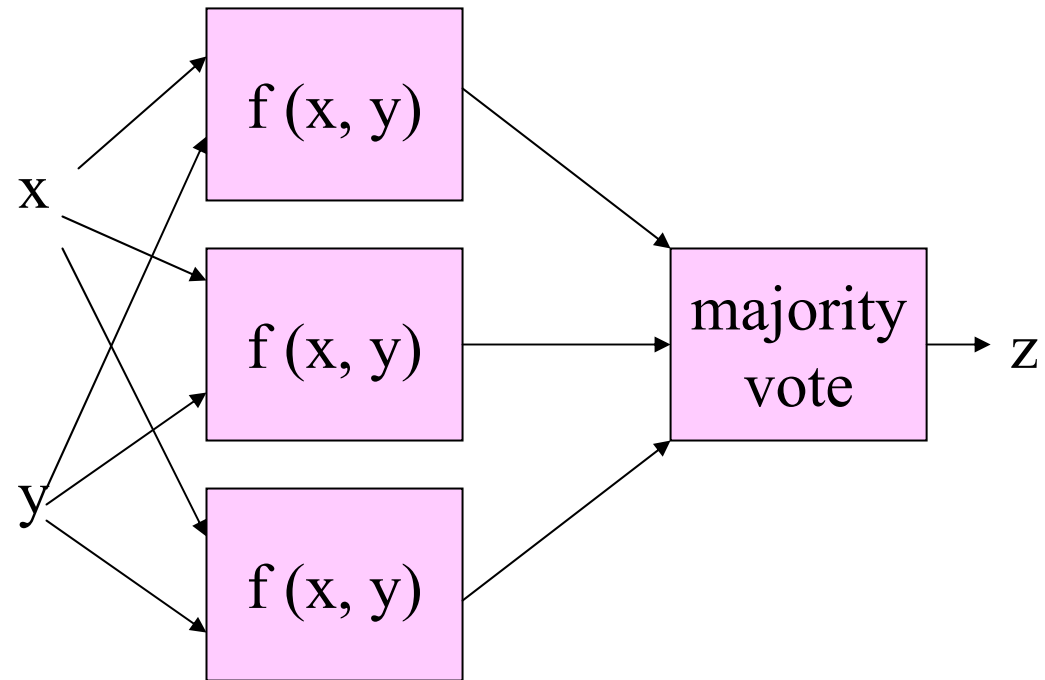


Triple Modular Redundancy (von Neumann)

Voter assumed reliable!

\Rightarrow voter small

\Rightarrow *coarse-grained*



What if voters are flaky?

What if voters are flaky?

- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)



What if voters are flaky?

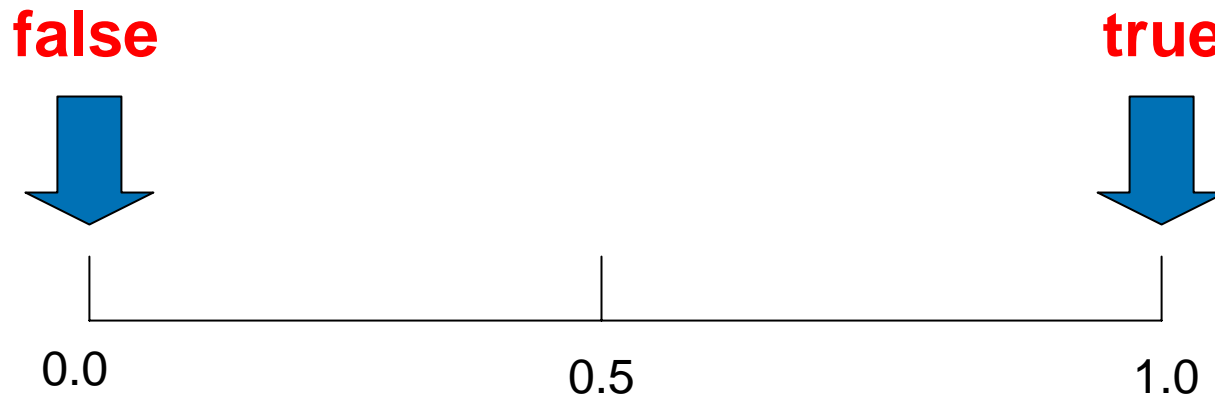
- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)

false



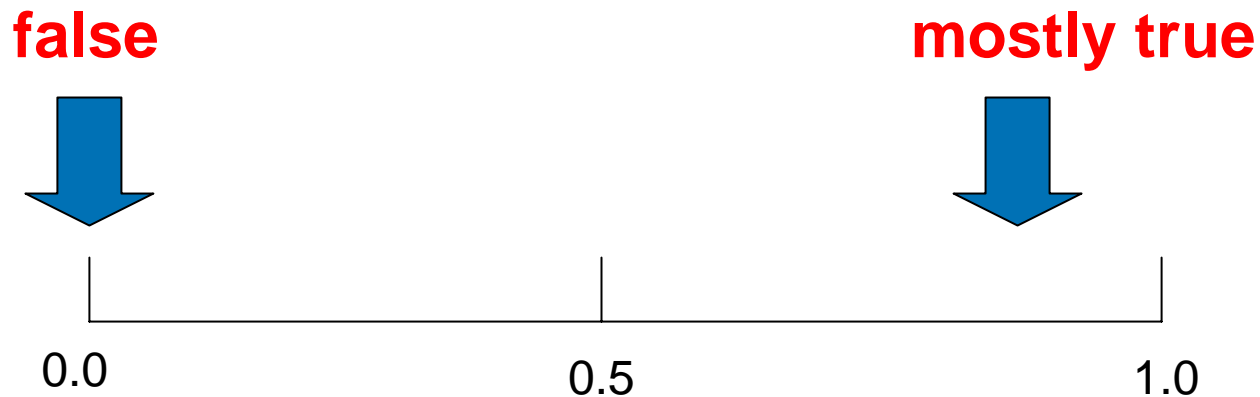
What if voters are flaky?

- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)



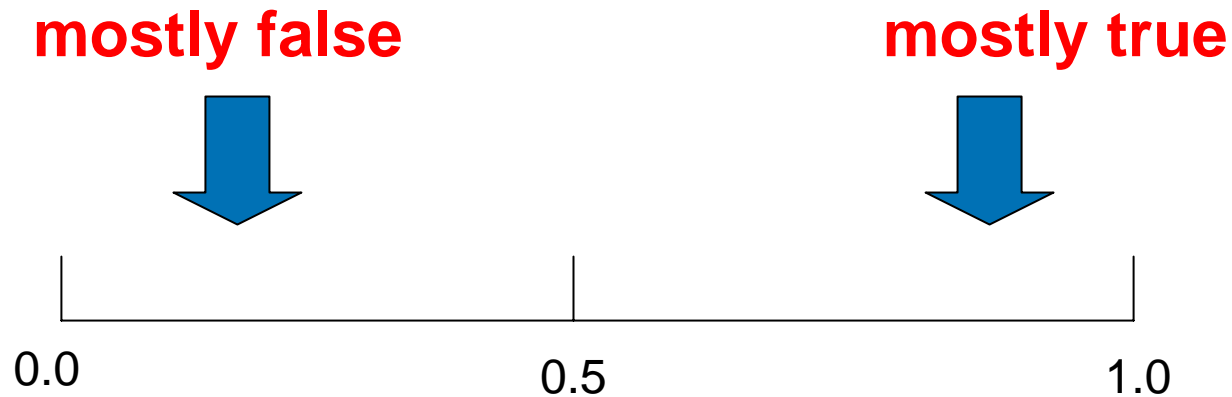
What if voters are flaky?

- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)



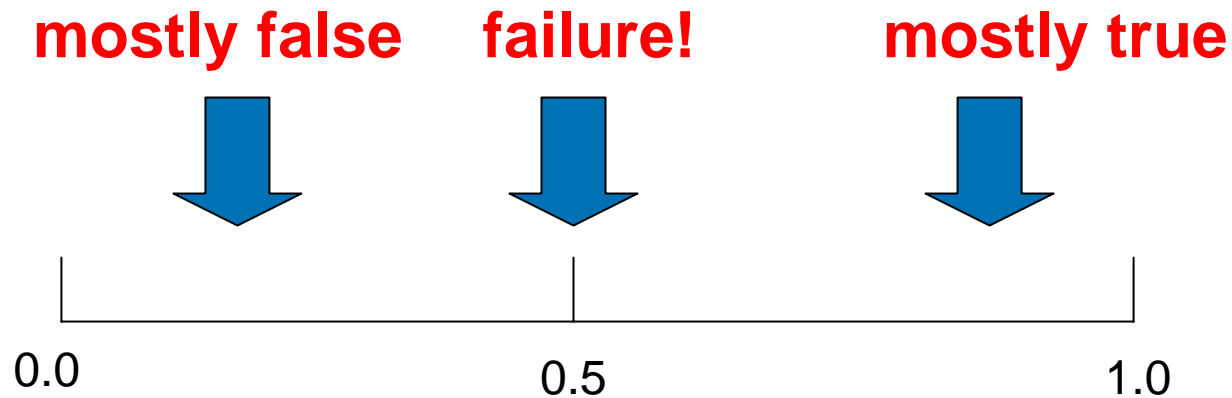
What if voters are flaky?

- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)

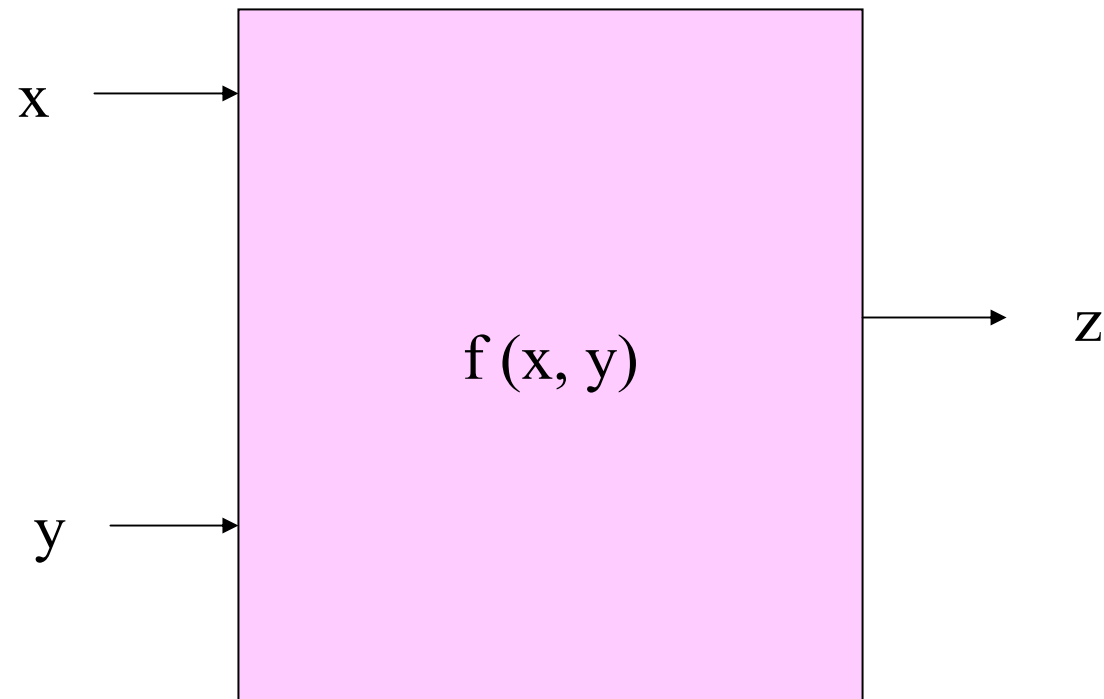


What if voters are flaky?

- Probabilistic approach
- Each logic signal \rightarrow “fuzzy” value (0...1)

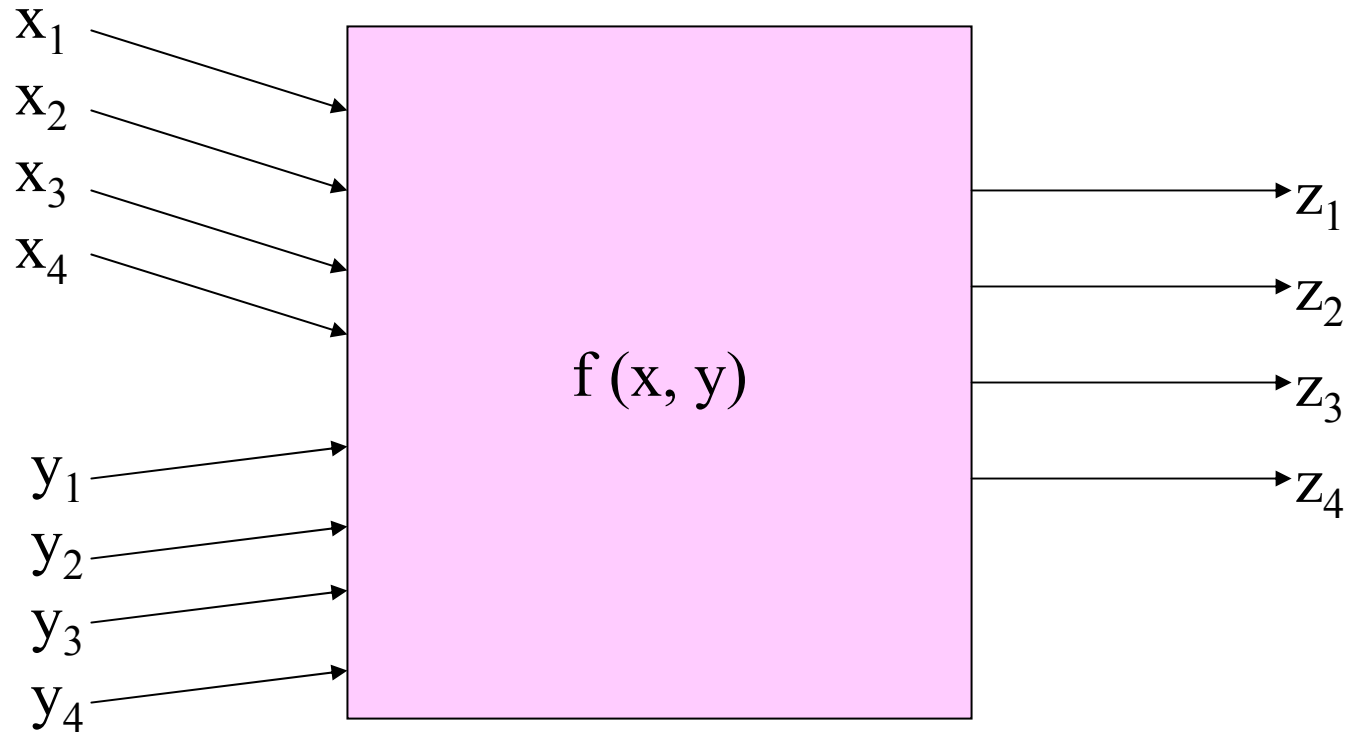


Parallel Restitution (von Neumann)



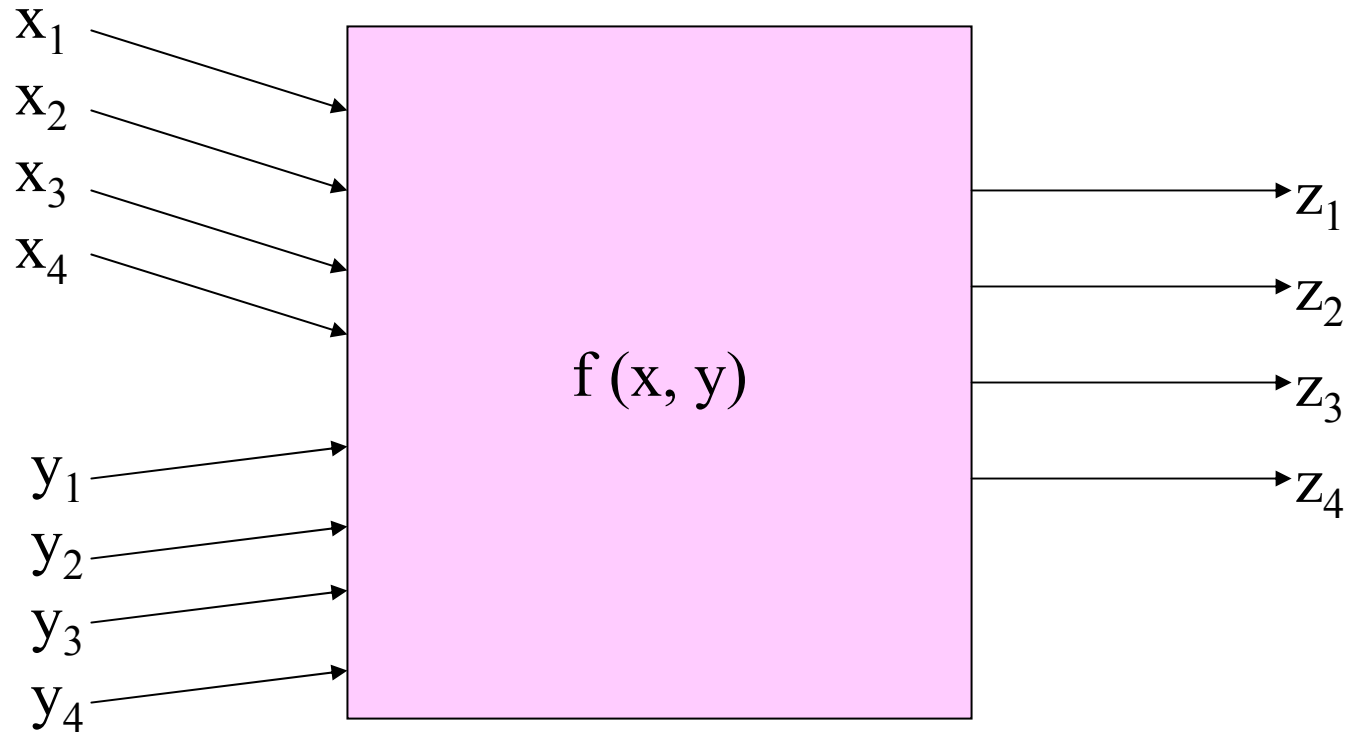
1. Replace each wire with “bundle”

Parallel Restitution (von Neumann)



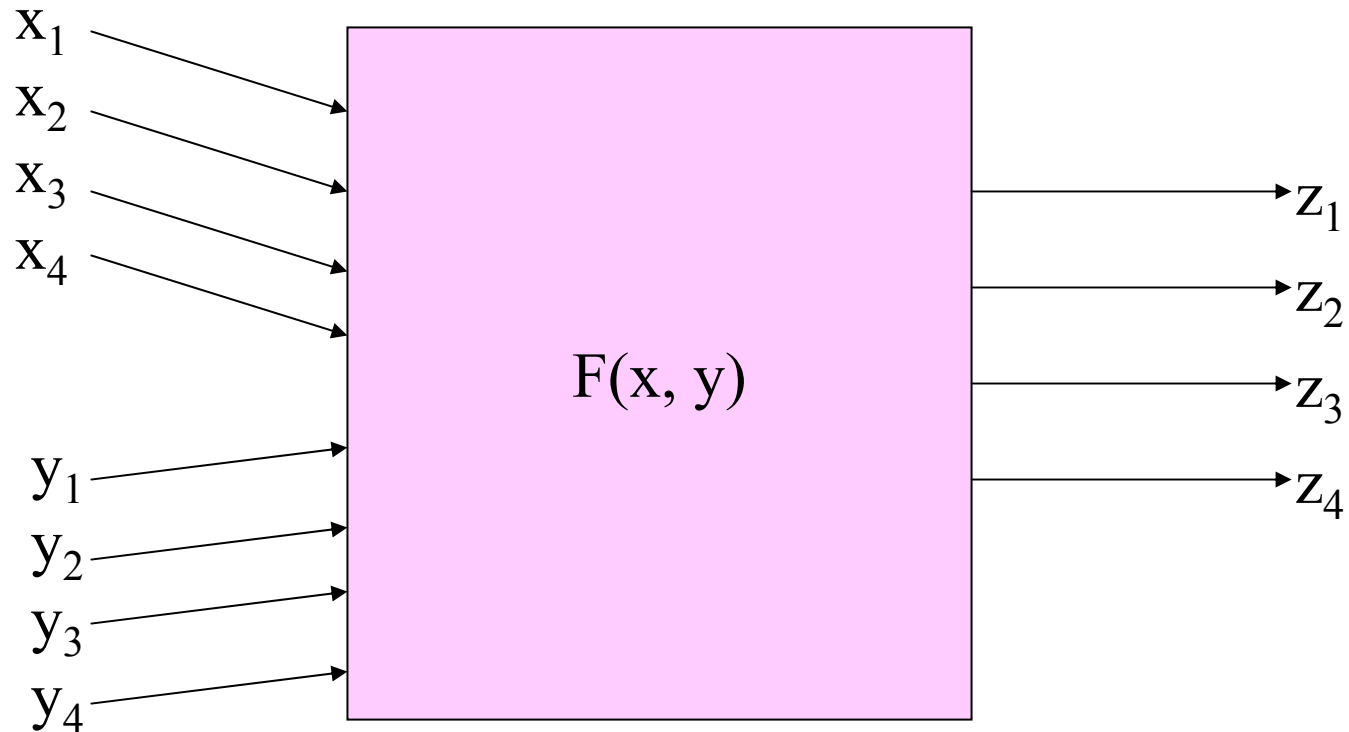
1. Replace each wire with “bundle”

Parallel Restitution (von Neumann)



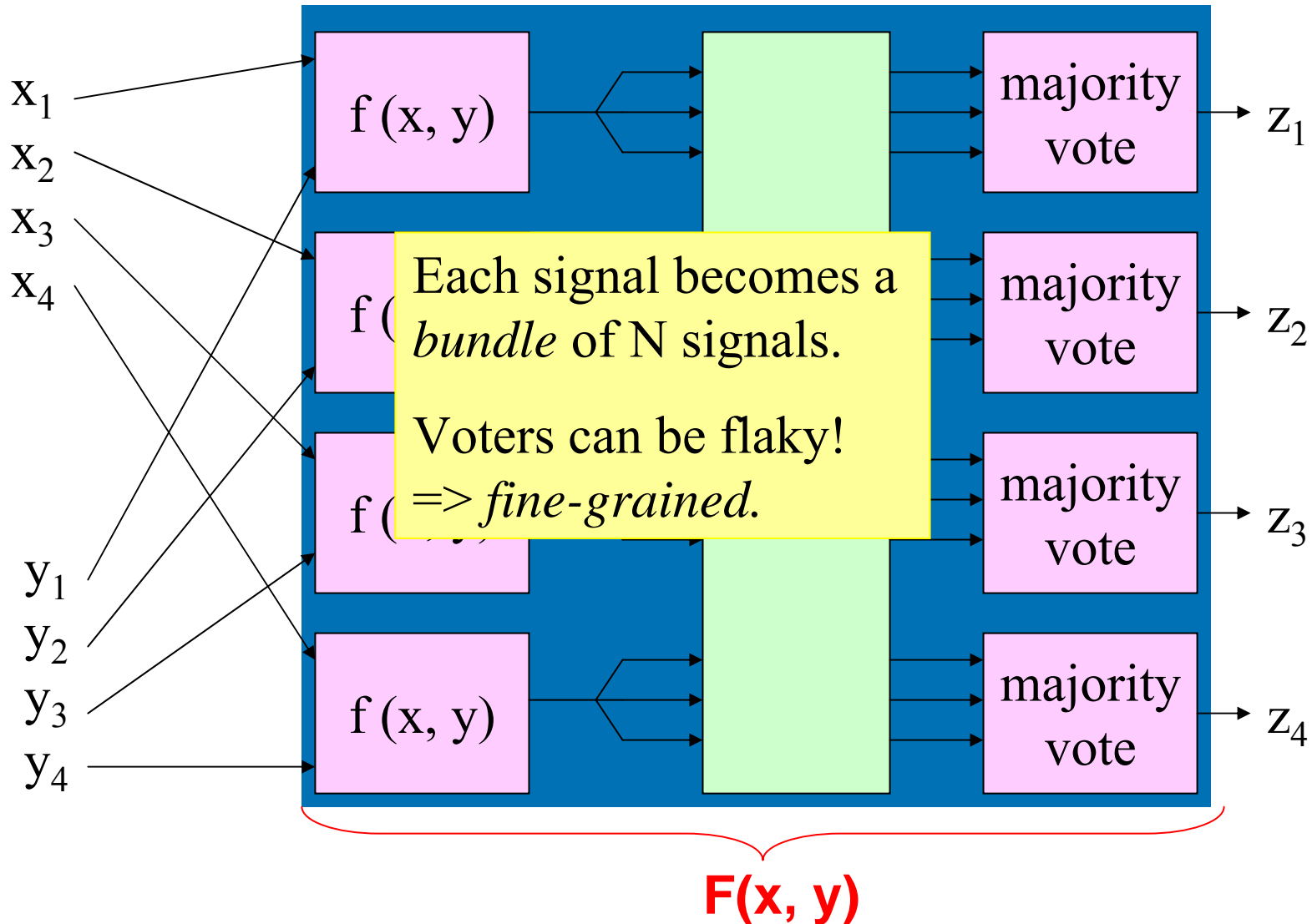
2. Replace function with redundant version, F

Parallel Restitution (von Neumann)

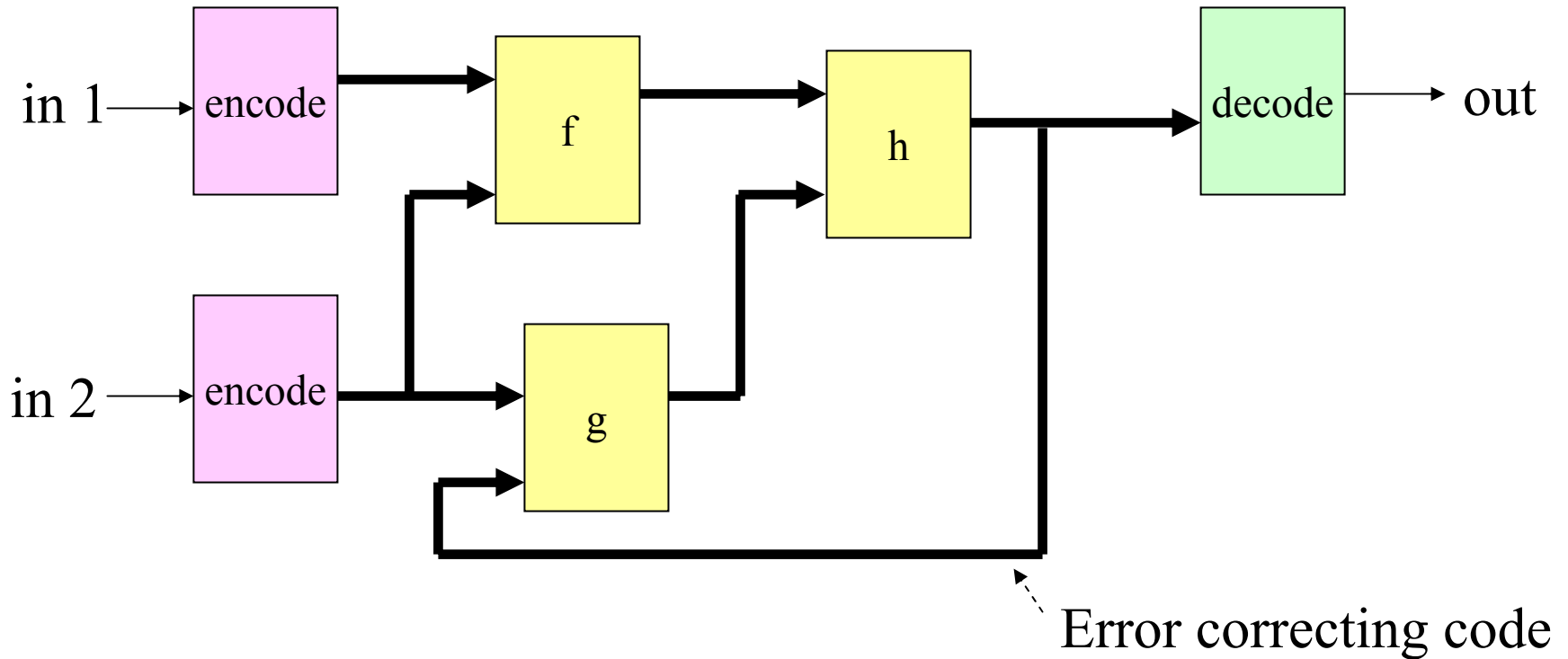


2. Replace function with redundant version, F

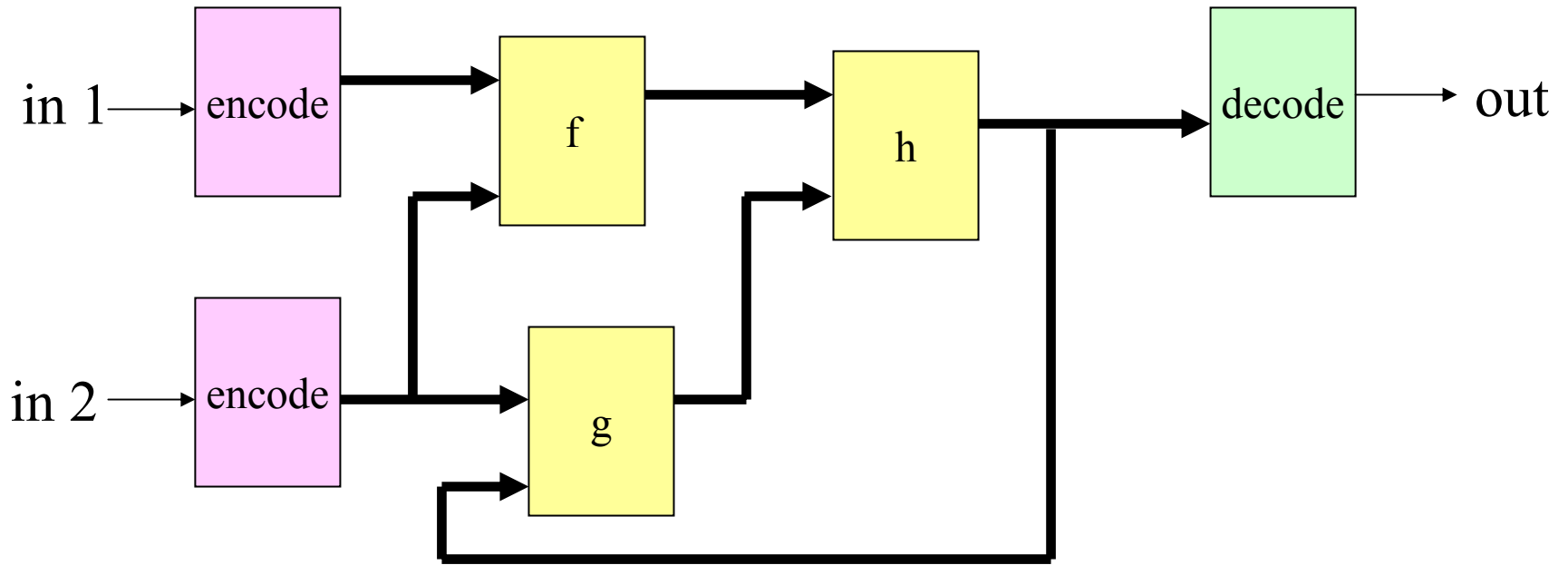
Parallel Restitution (von Neumann)



Self-correcting circuits



Self-correcting circuits



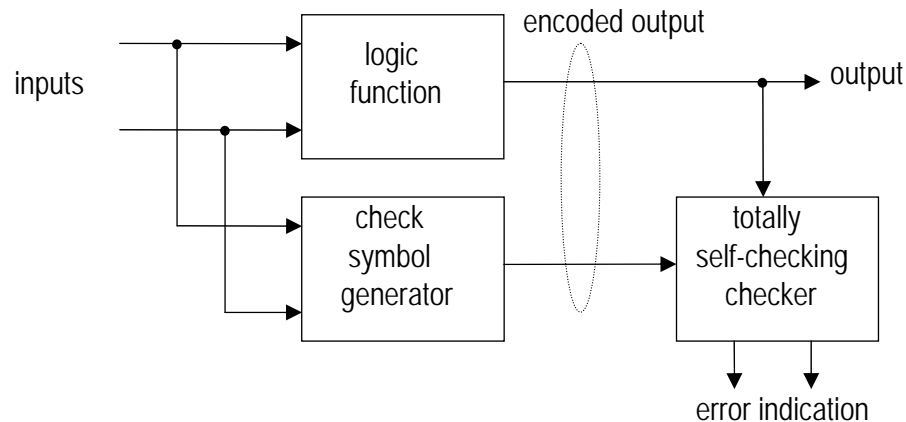
Do efficient codes exist?

Add, multiply, shift → yes!

Others → no!

Error correcting code

Self-checking circuits

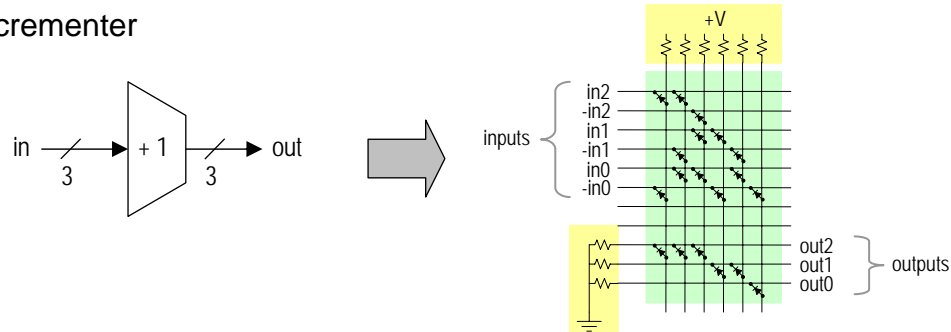


Error detection is cheaper than correction:

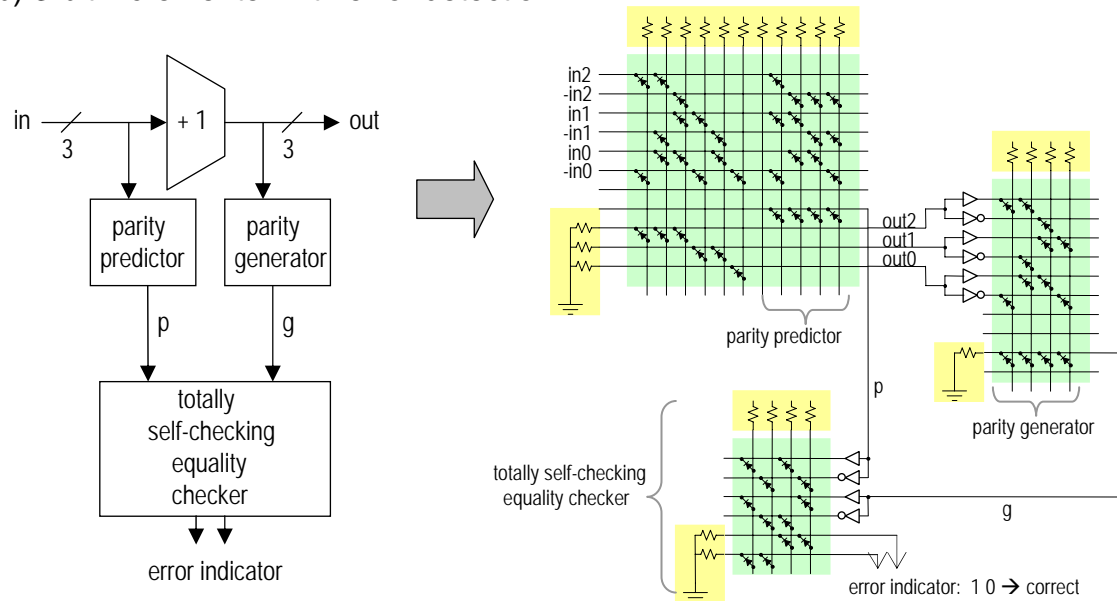
- Execute machine cycle.
- If no errors: latch results, advance state machine,
- Otherwise restart current cycle.

Example: self-checking circuit

(a) 3-bit incrementer

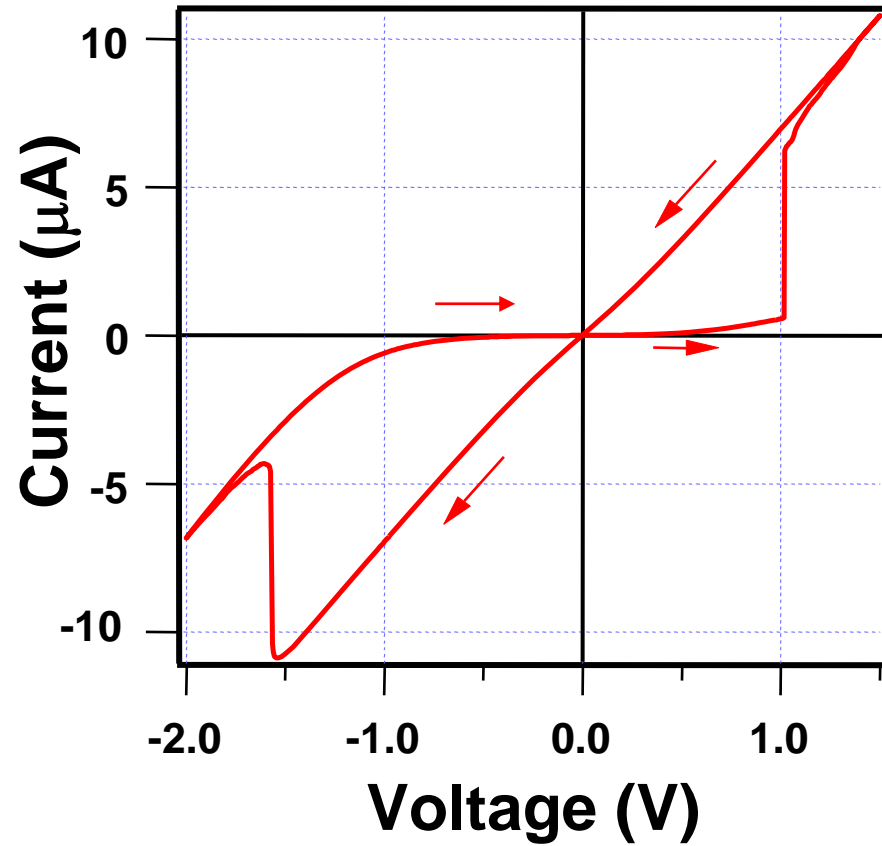
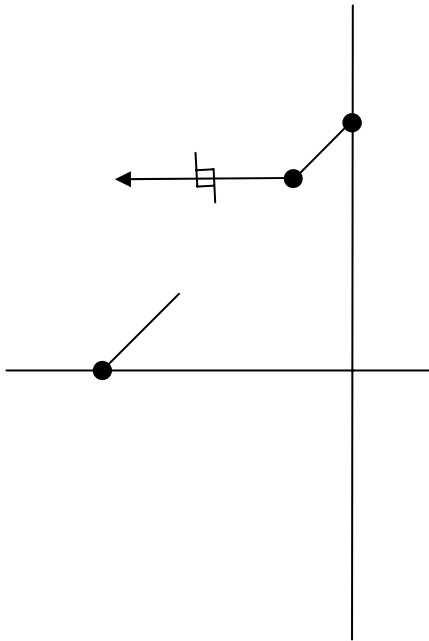


(b) 3-bit incrementer with error detection

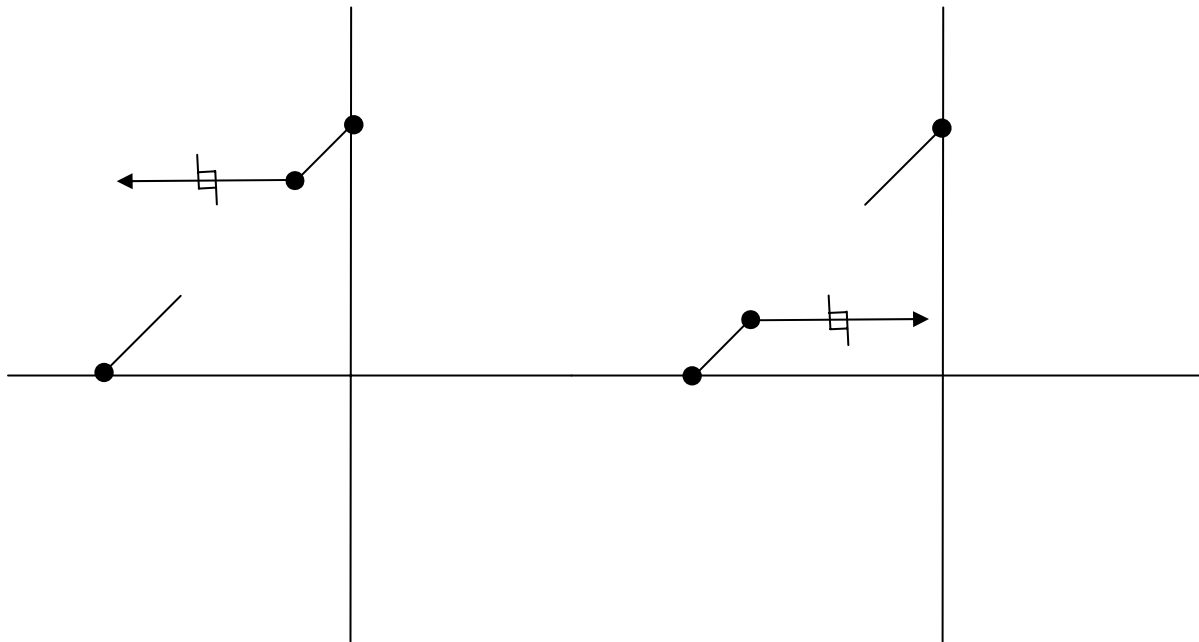


Unconventional Stuff

Hysteretic Resistors

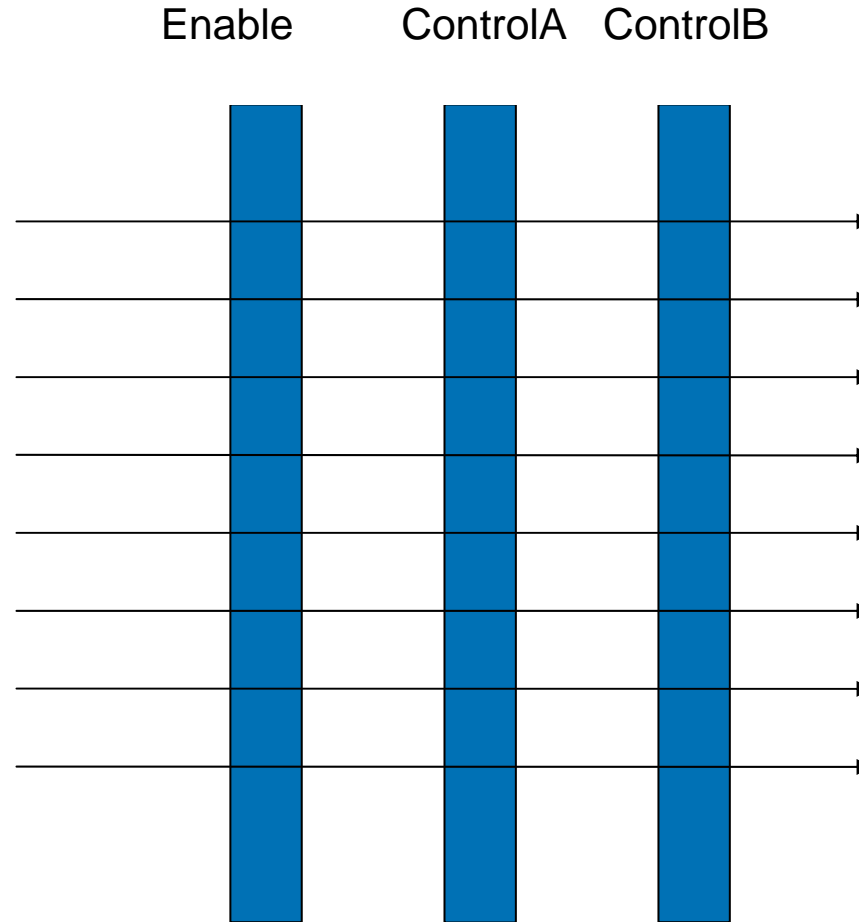


Hysteretic Resistor Latch (voltage)

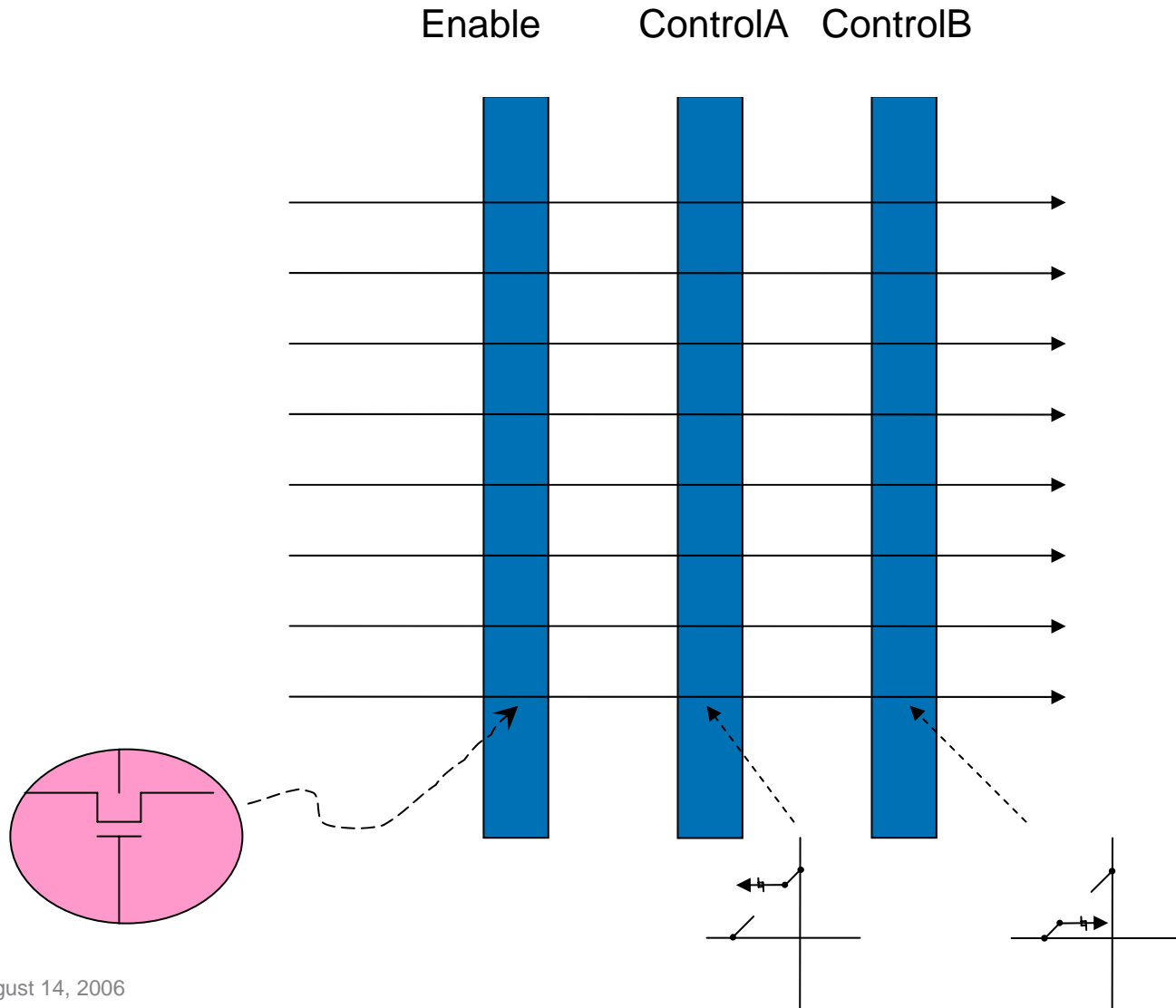


Opposite “polarities”

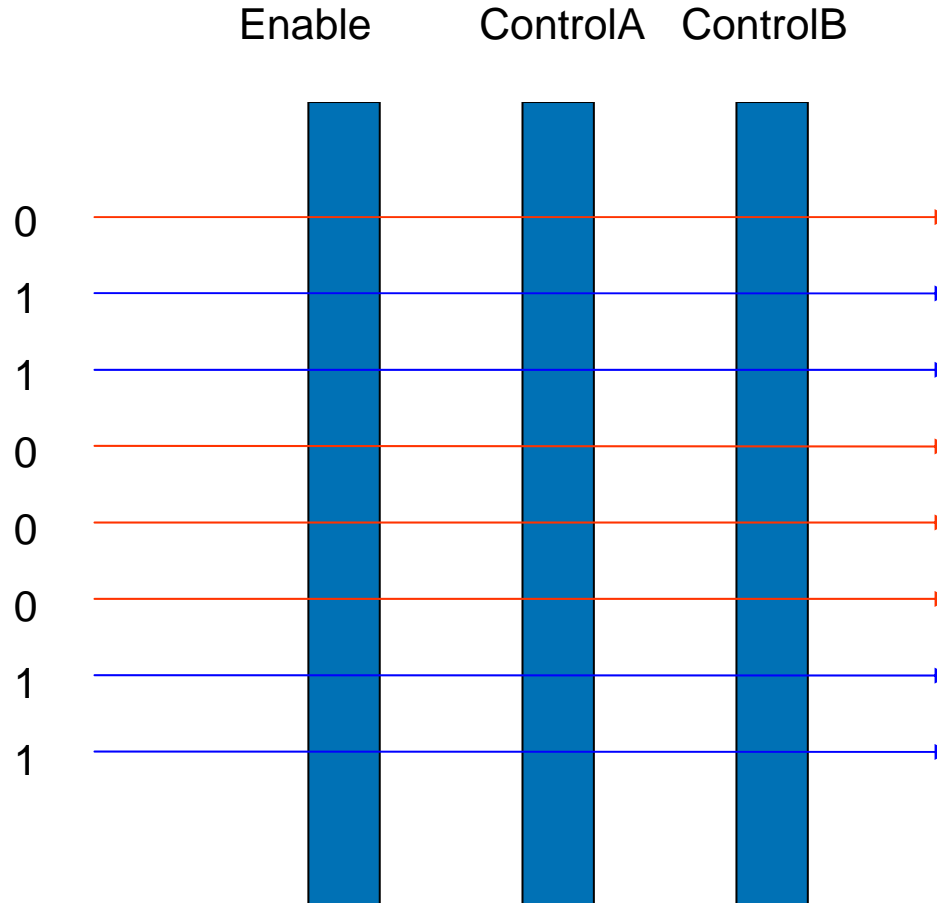
Hysteretic Resistor Latch (voltage)



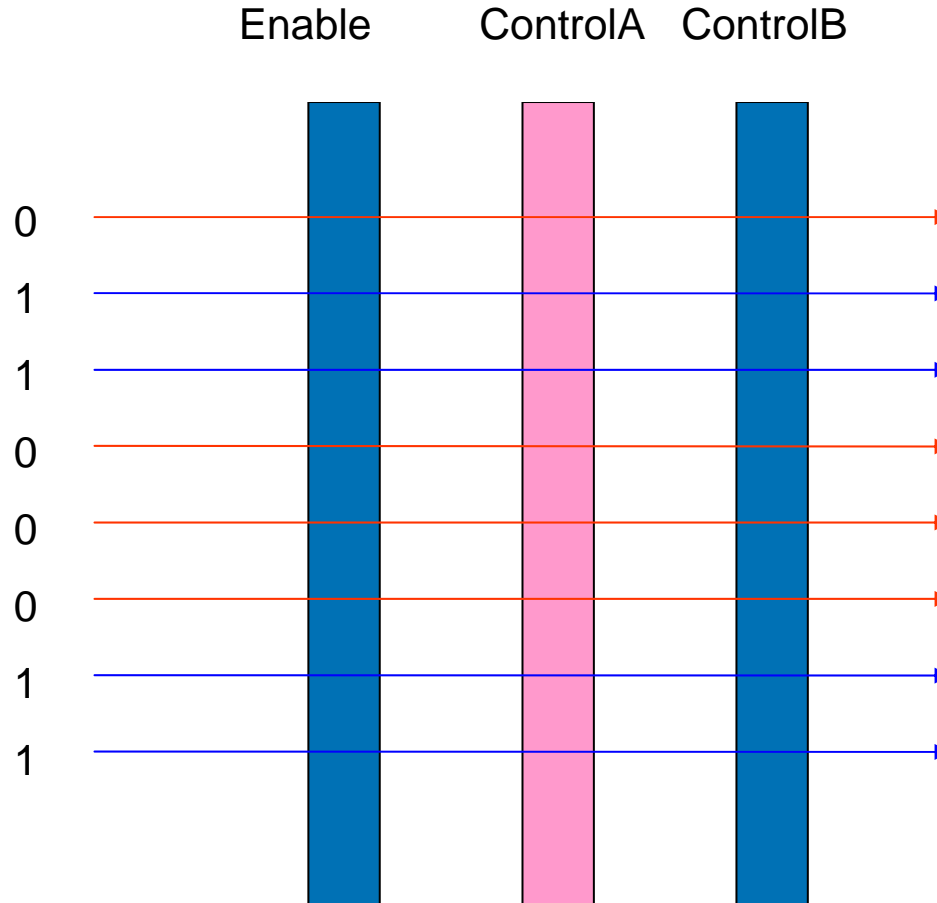
Hysteretic Resistor Latch (voltage)



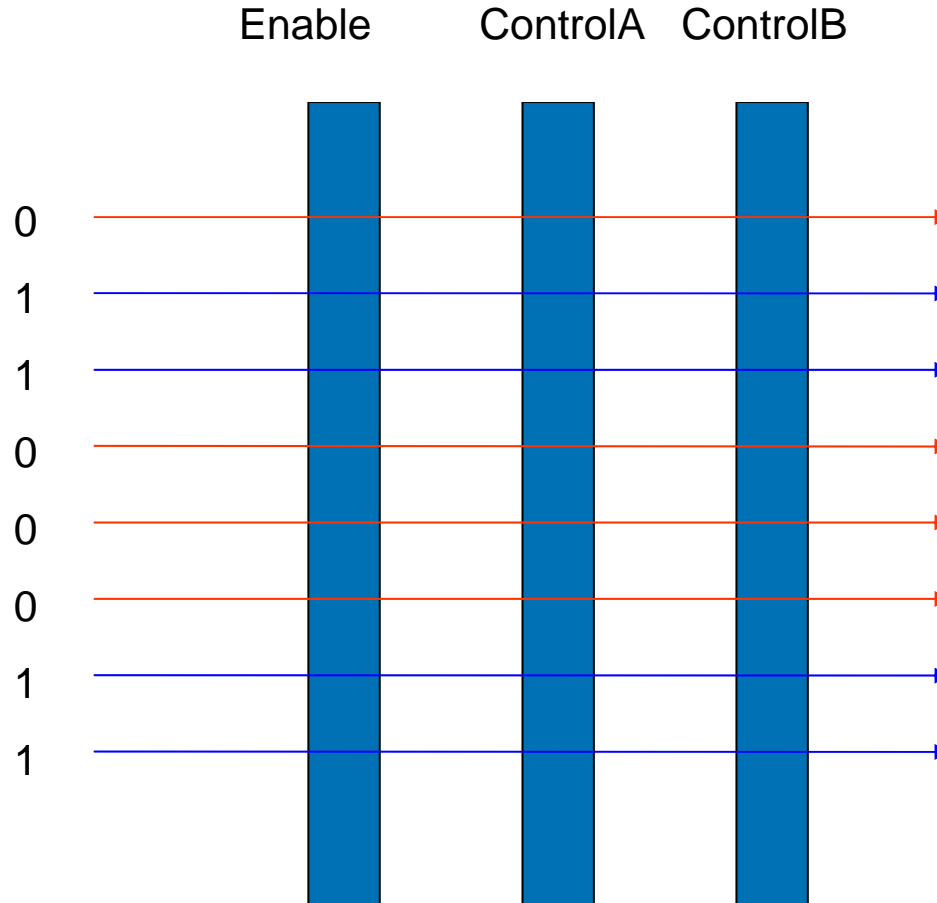
Programming Mode



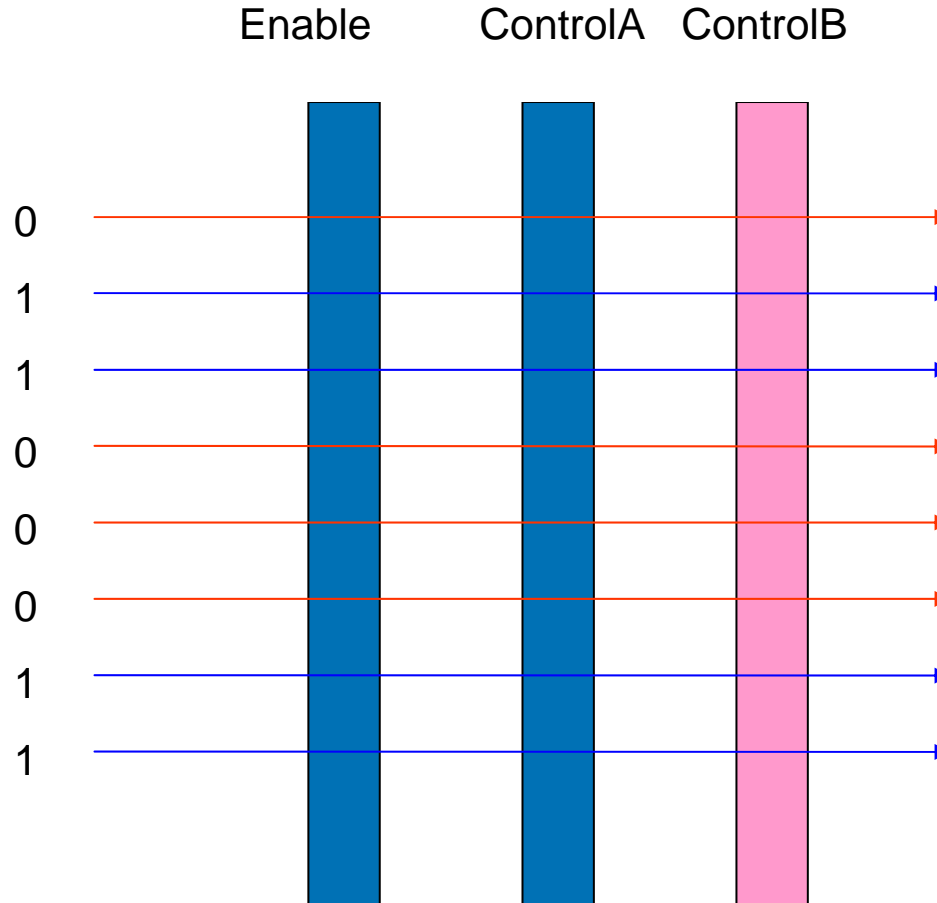
Programming Mode



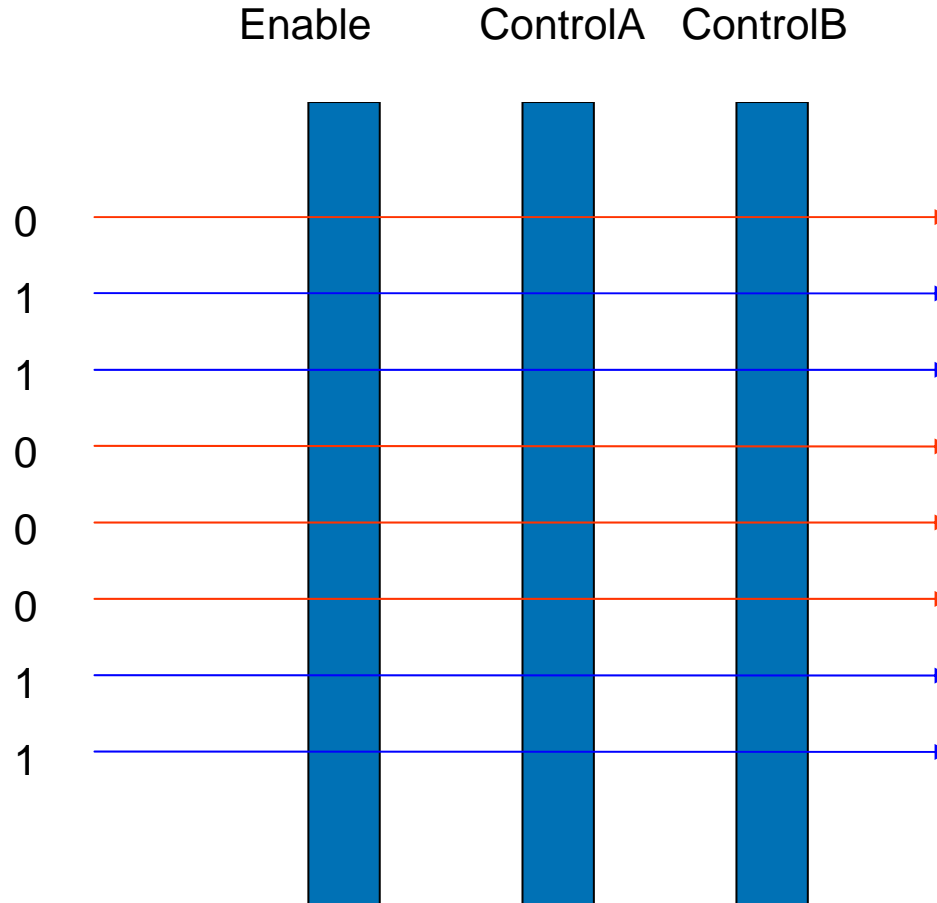
Programming Mode



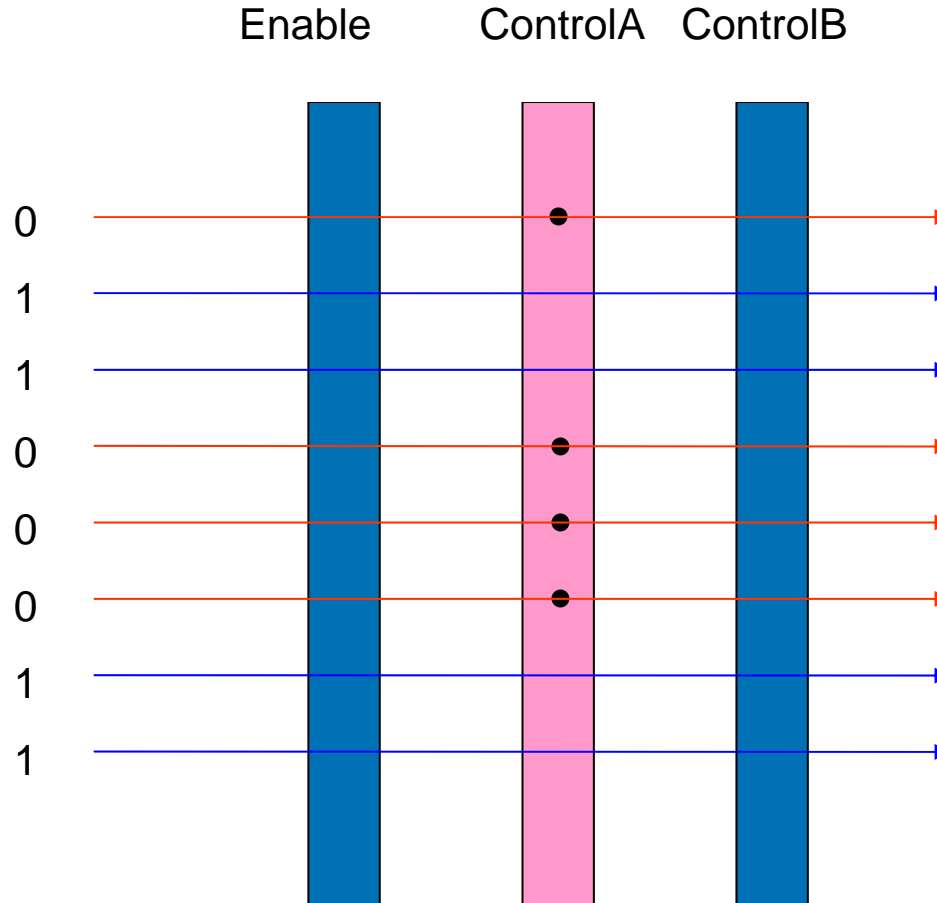
Programming Mode



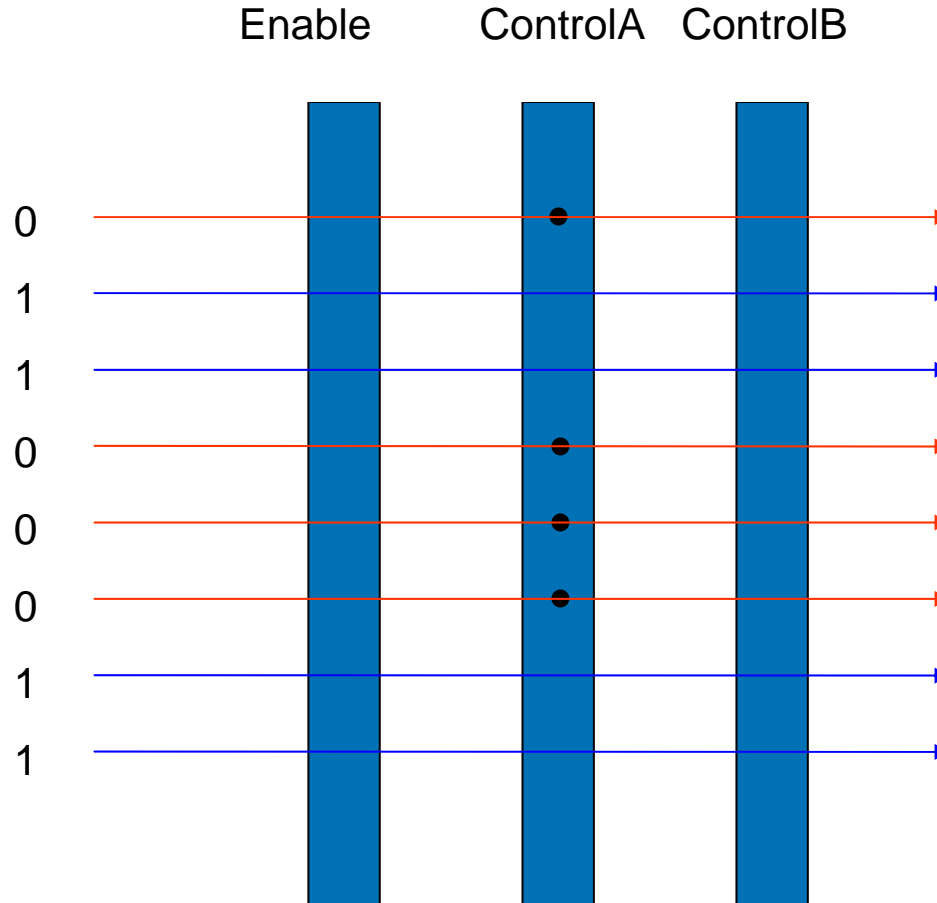
Programming Mode



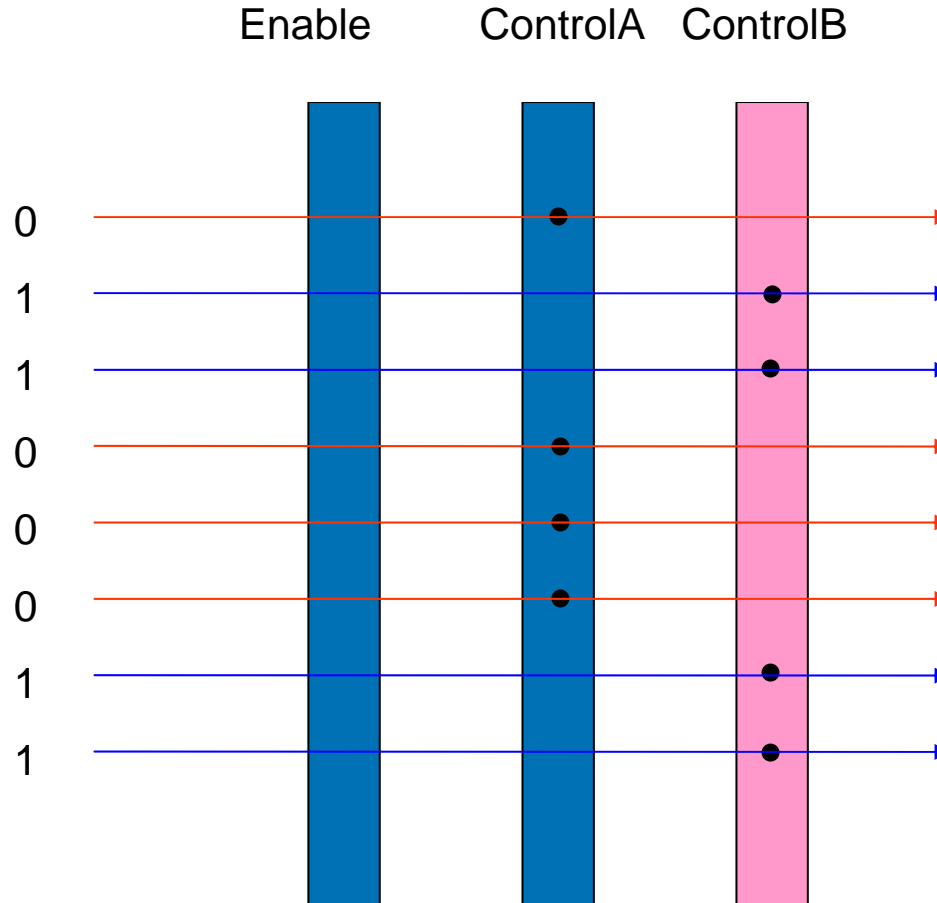
Programming Mode



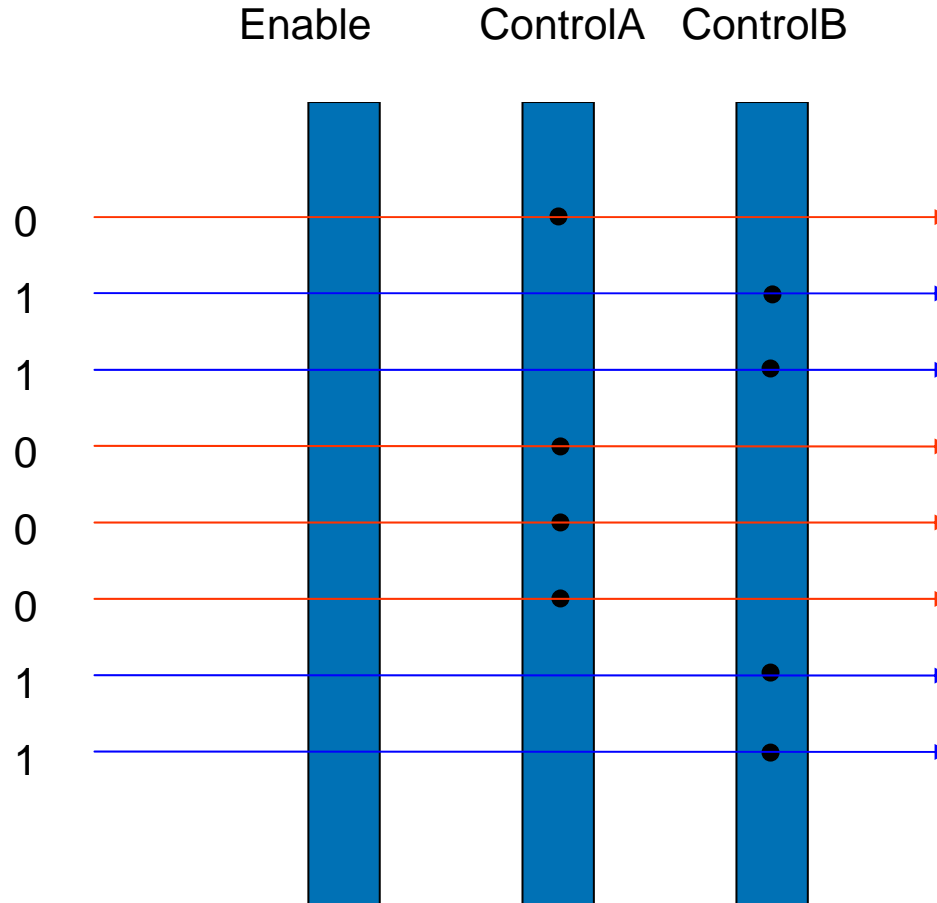
Programming Mode



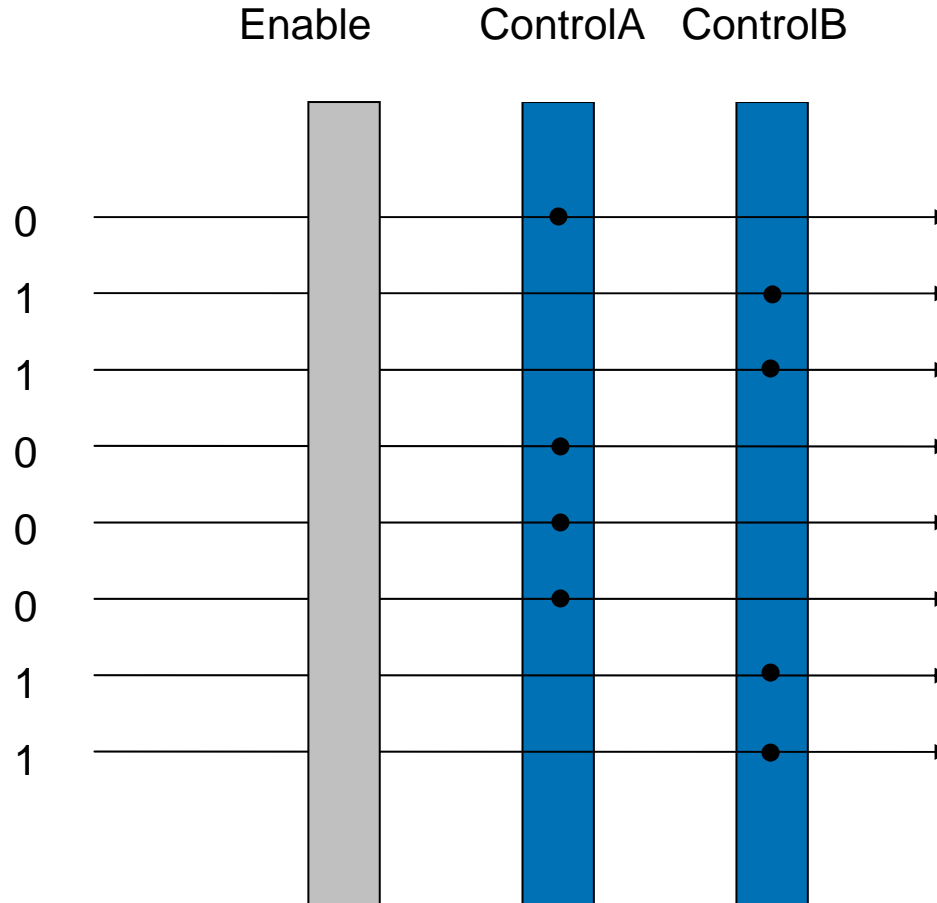
Programming Mode



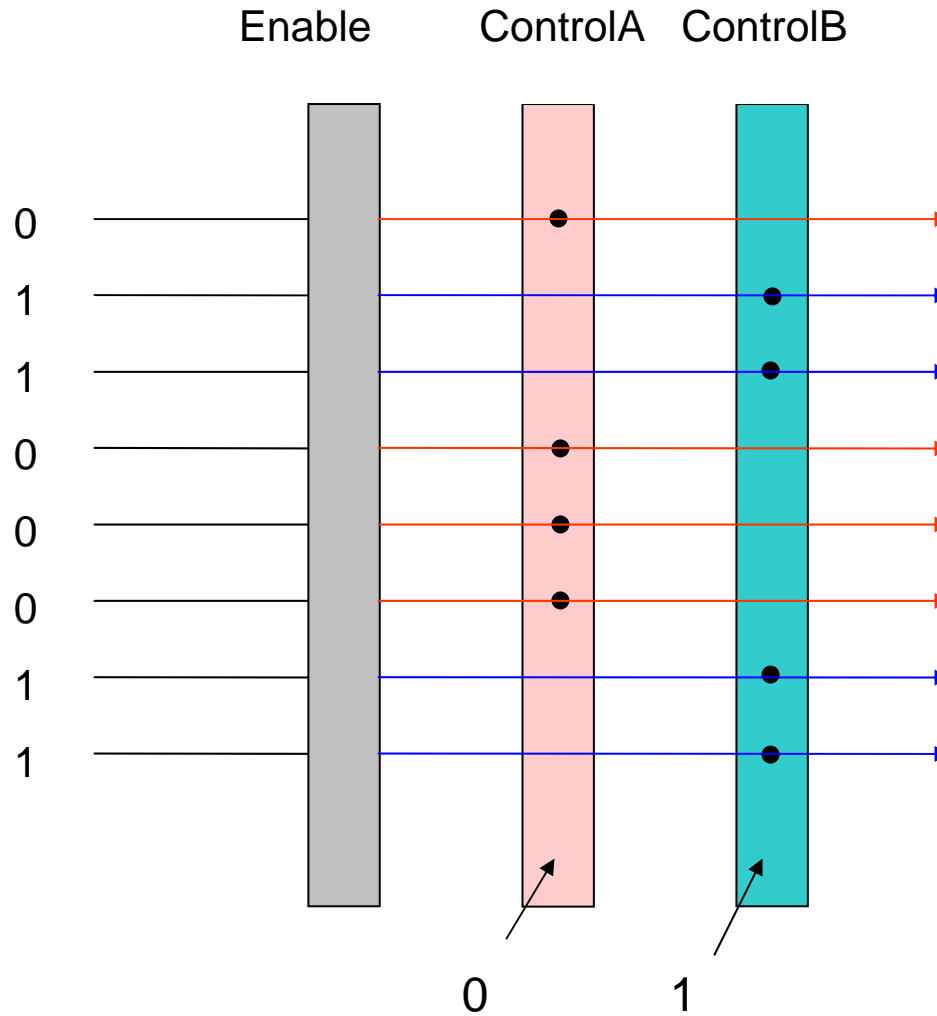
Programming Mode



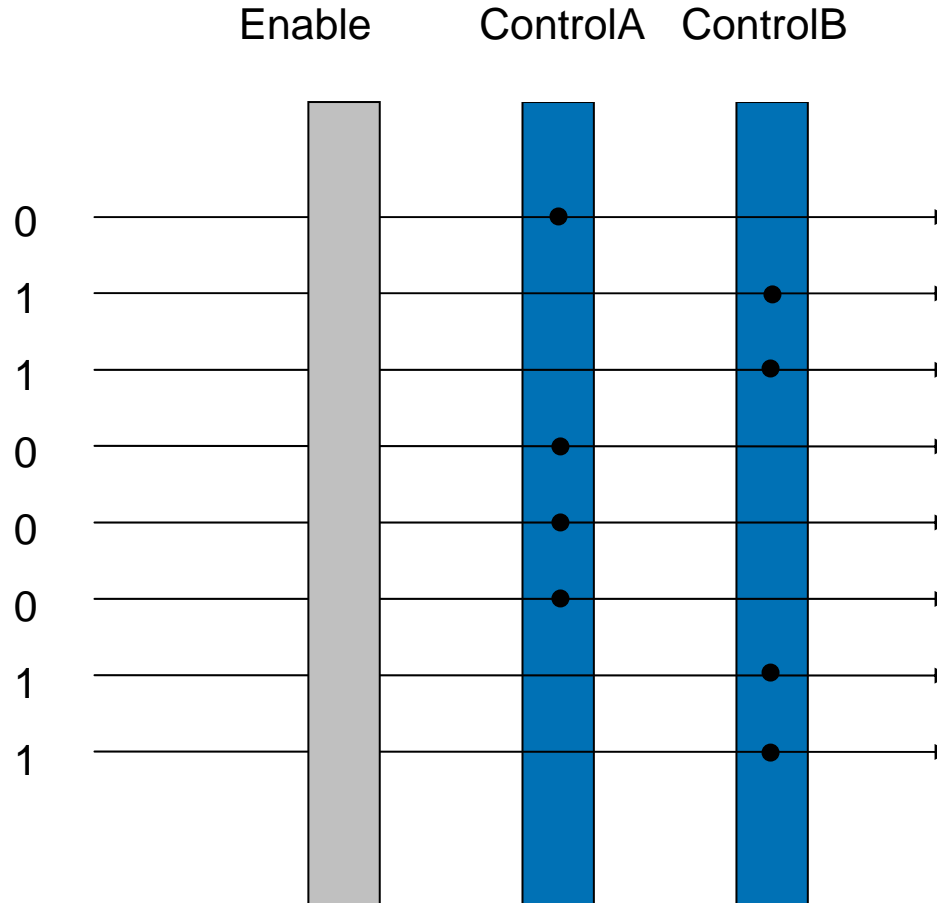
Output Mode



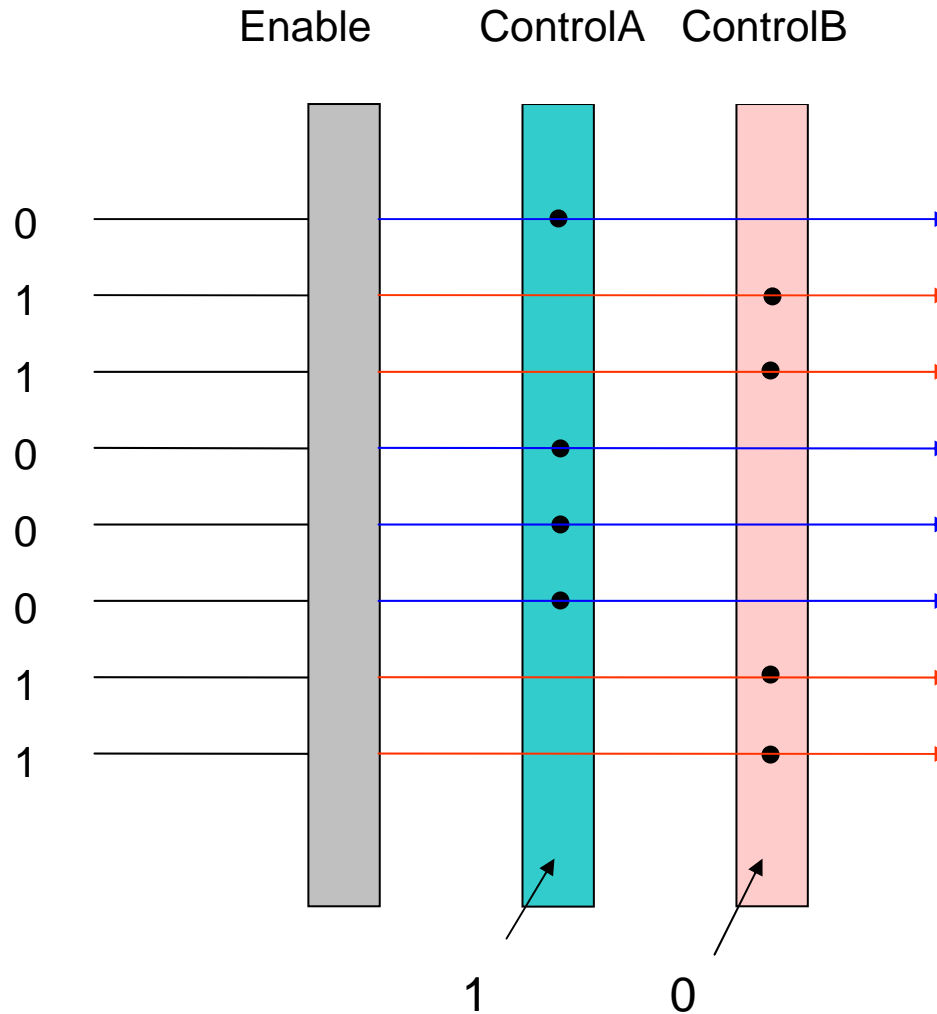
Output Mode (non-inverting)



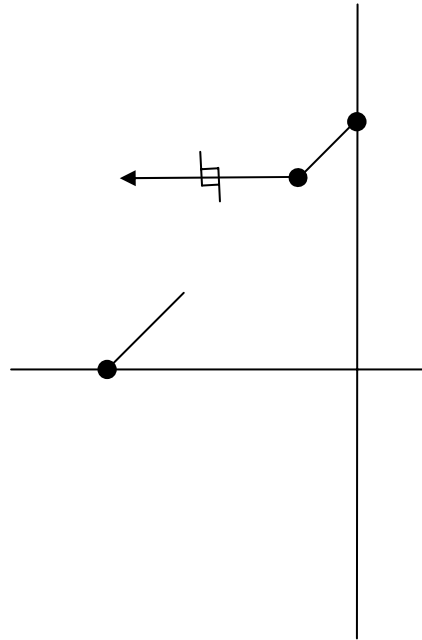
Output Mode



Output Mode (inverting)



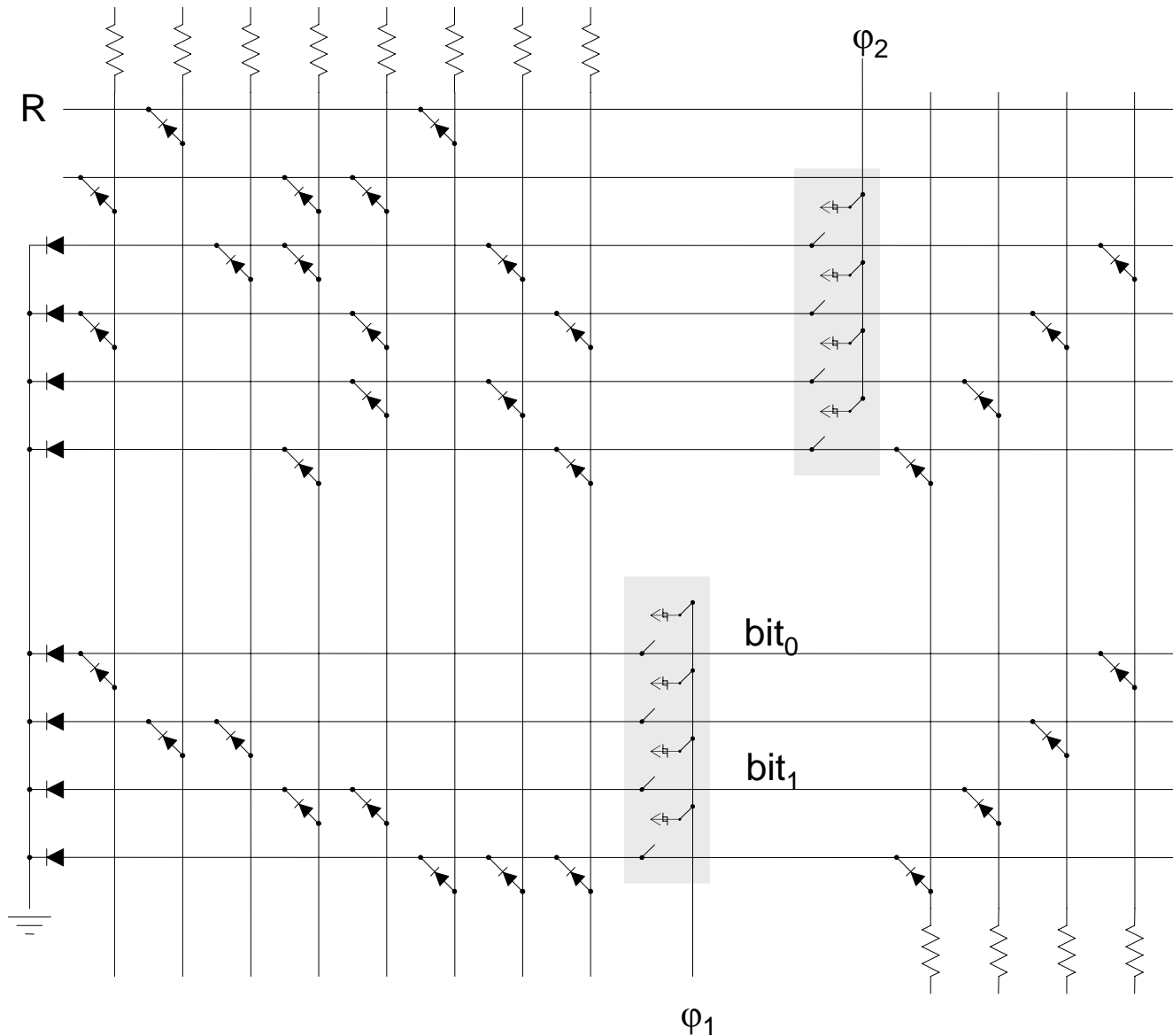
Hysteretic Resistor Latch (impedance encoding)



But wait... only need one hysteretic resistor:

- Open \rightarrow logic 1
- Closed \rightarrow logic 0

2-bit counter: strange diode-logic, impedance-encoded latches



Logic with Hysteretic Resistors?

What if all you had were hysteretic resistors?
Can you do logic as well as latches?

Logic with Hysteretic Resistors?

What if all you had were hysteretic resistors?
Can you do logic as well as latches?

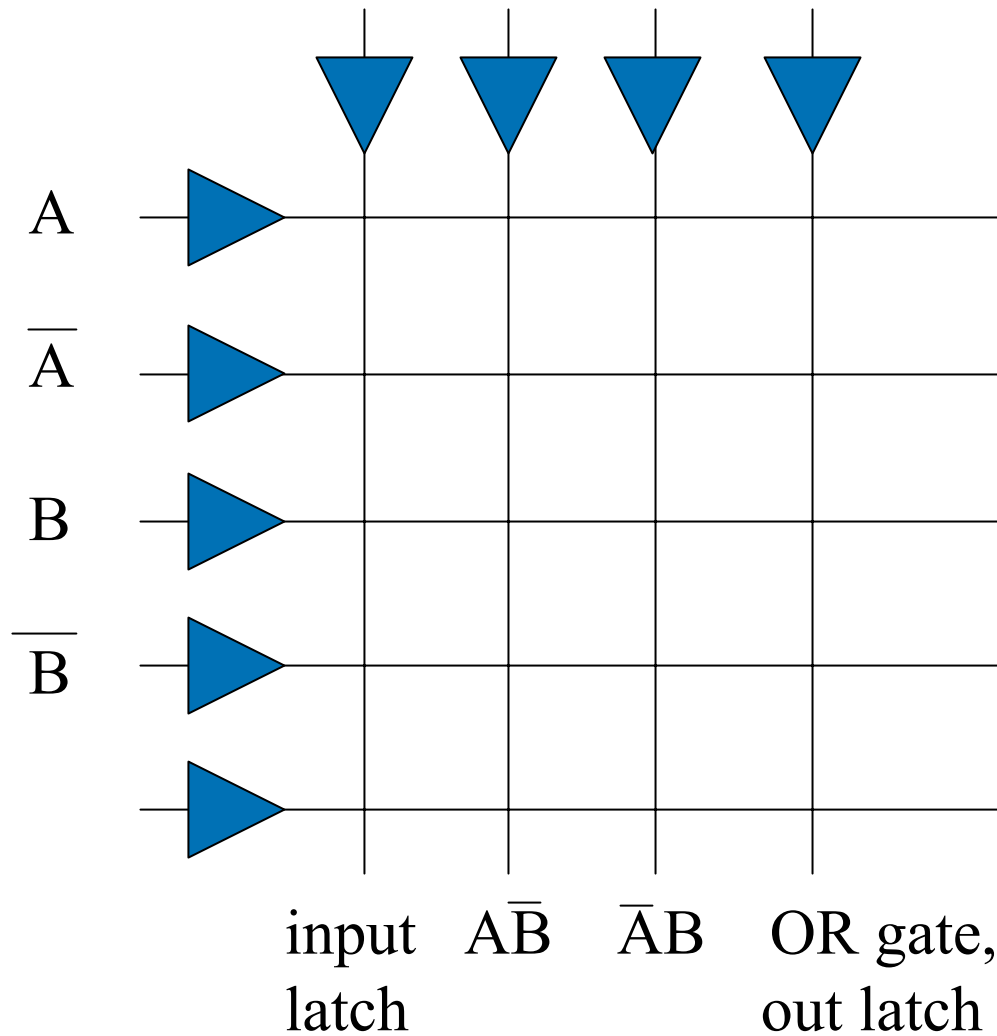
YES

Logic with Hysteretic Resistors?

Catches

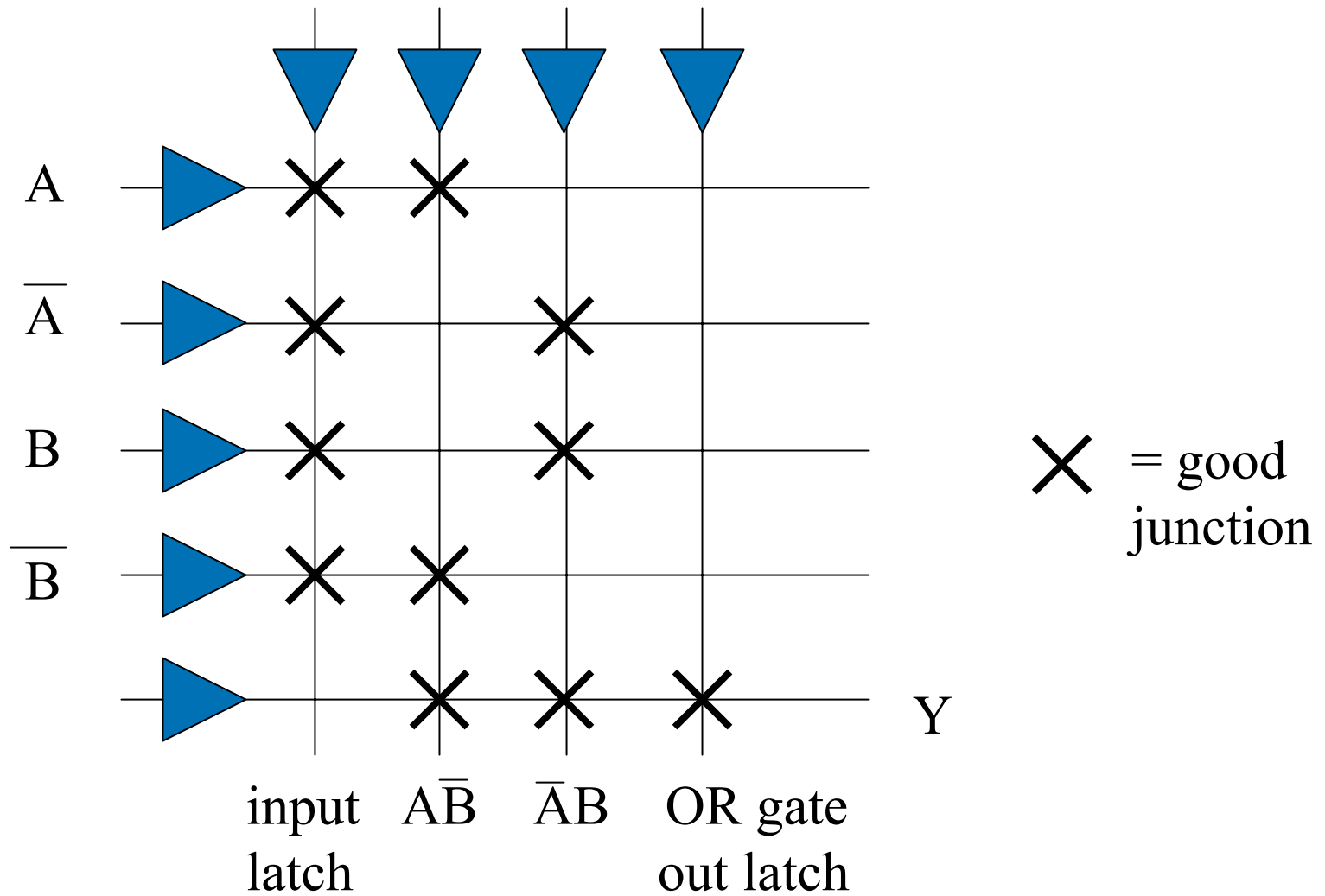
- Must destroy some junctions to “stuck open”
- Complex driving signals

Circuit

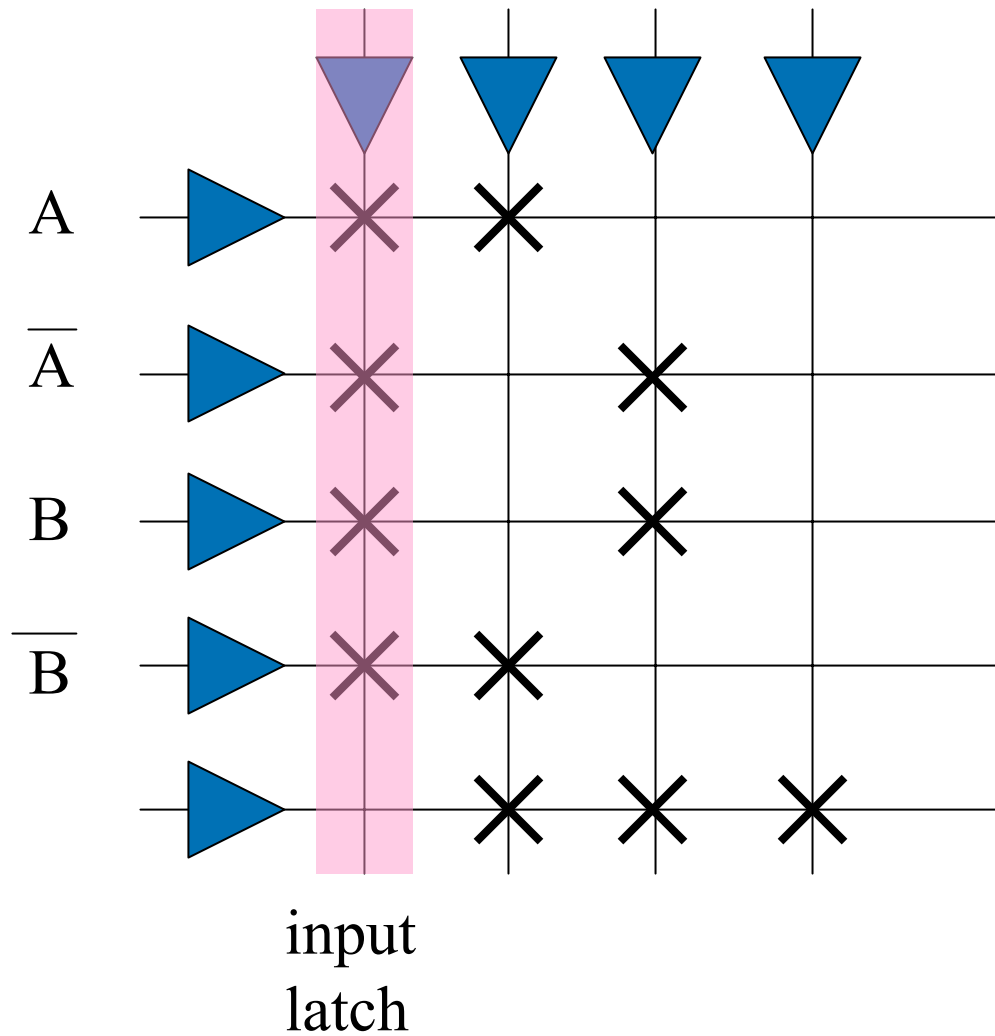


$$Y = \bar{A}\bar{B} + \bar{A}B$$

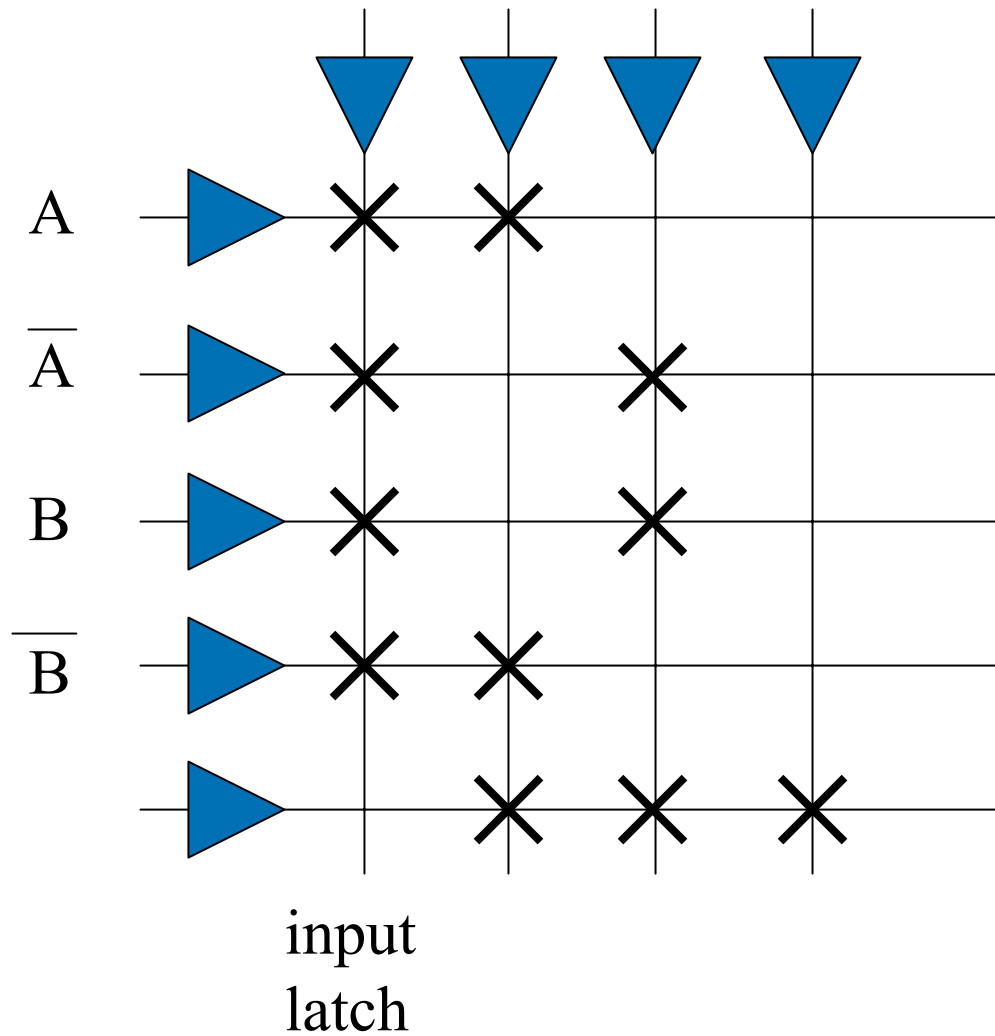
Circuit



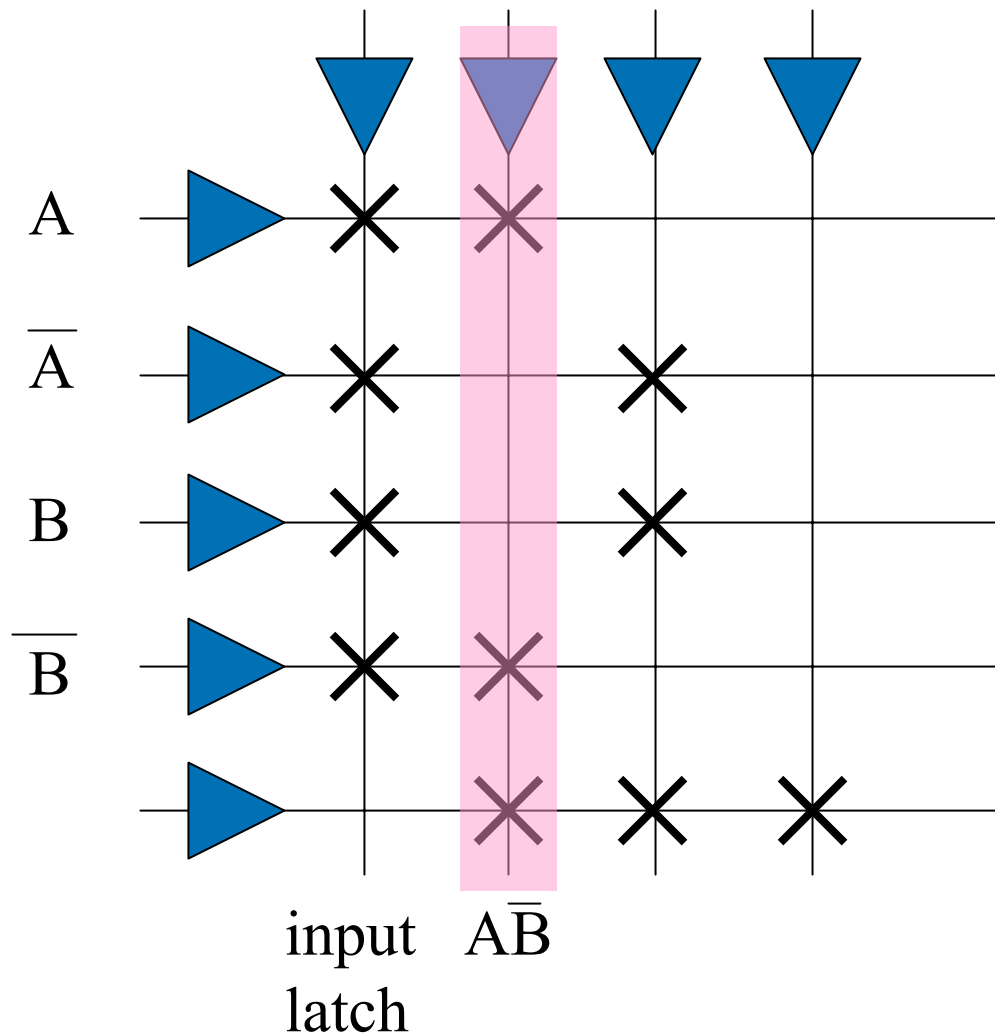
Latch inputs



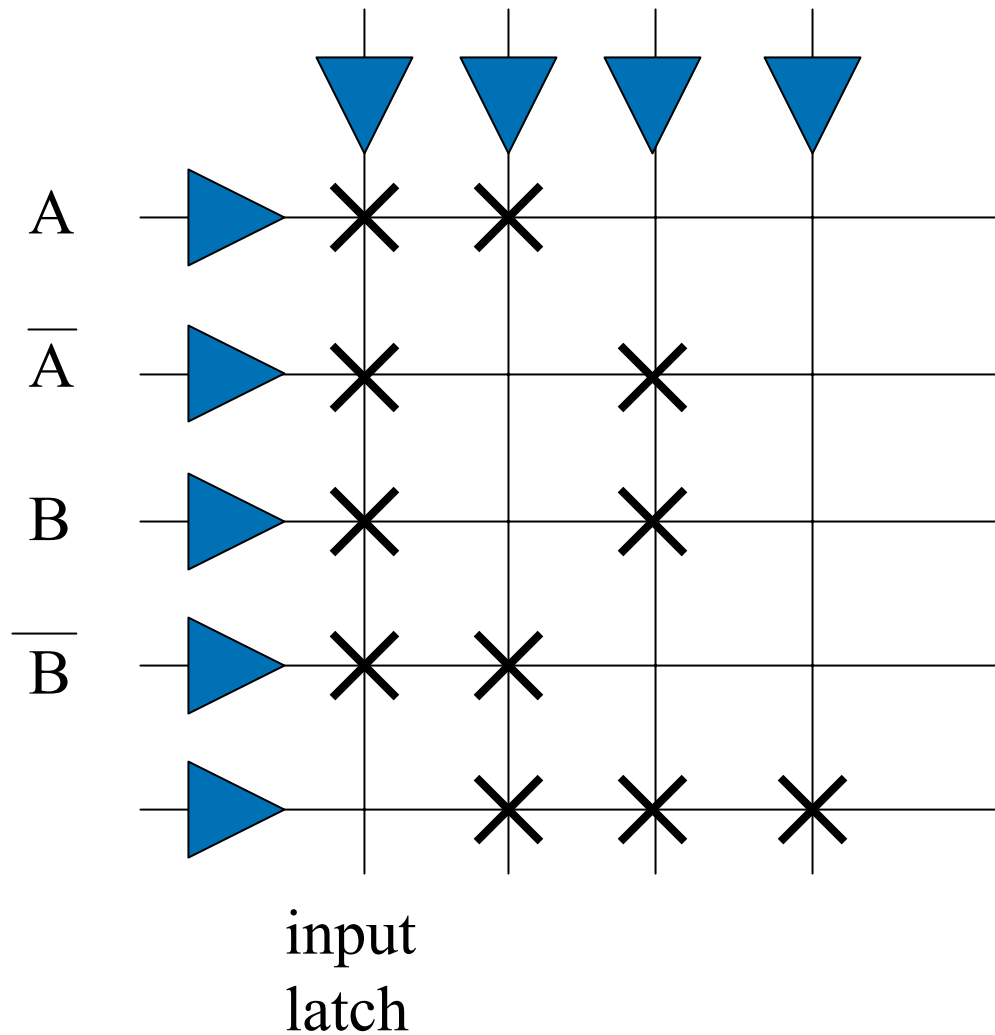
Latch inputs



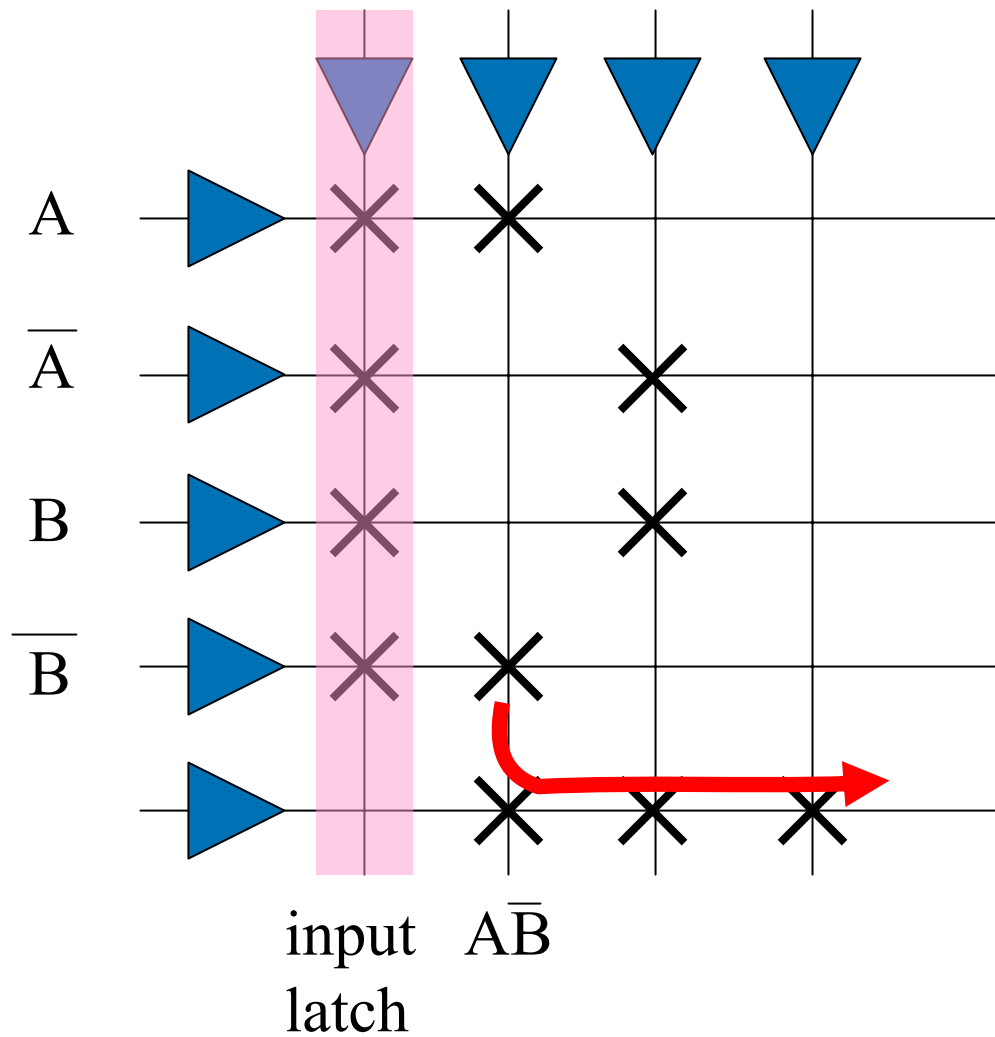
Close First Minterm Switches



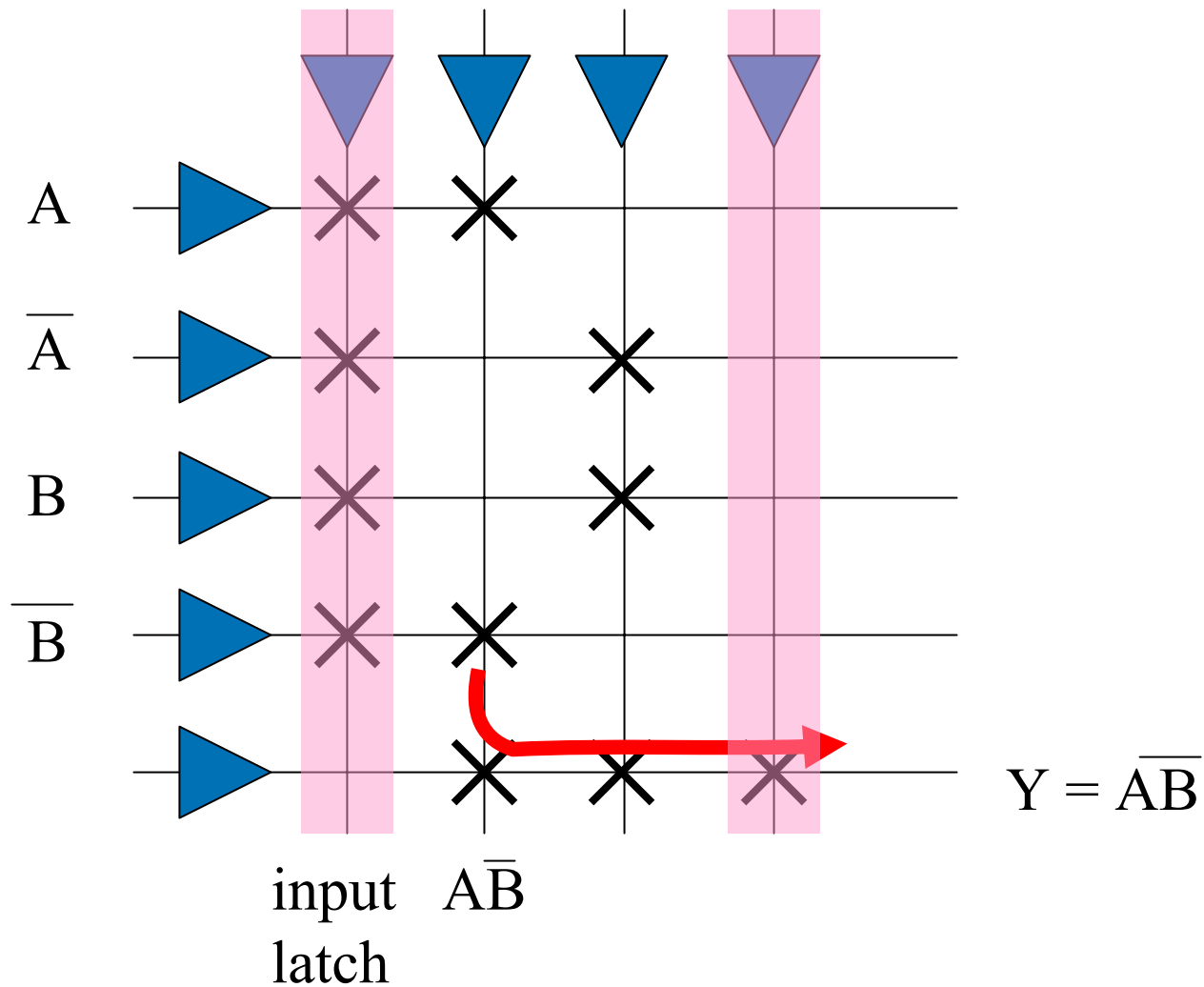
Close First Minterm Switches



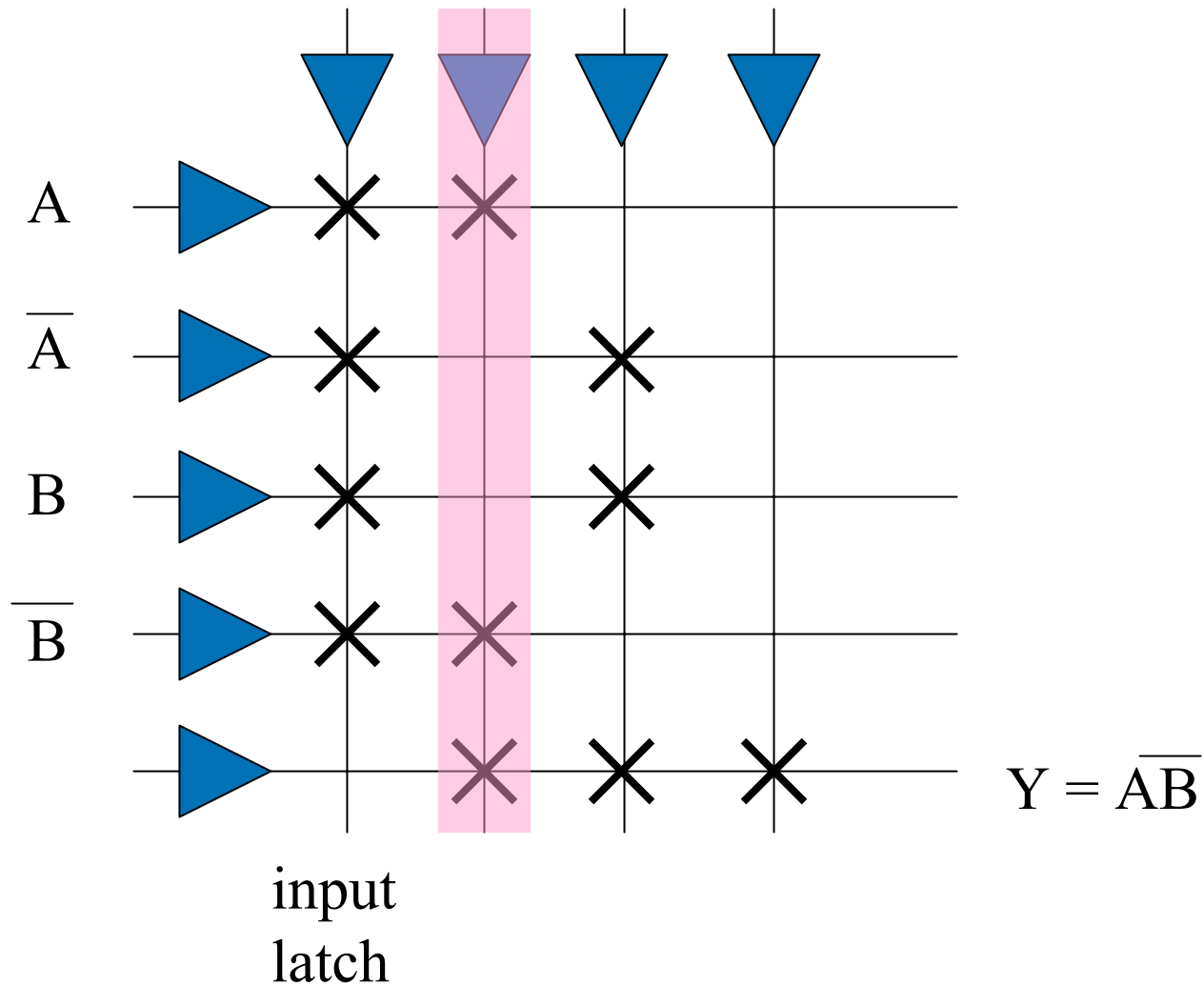
Compute minterm 1



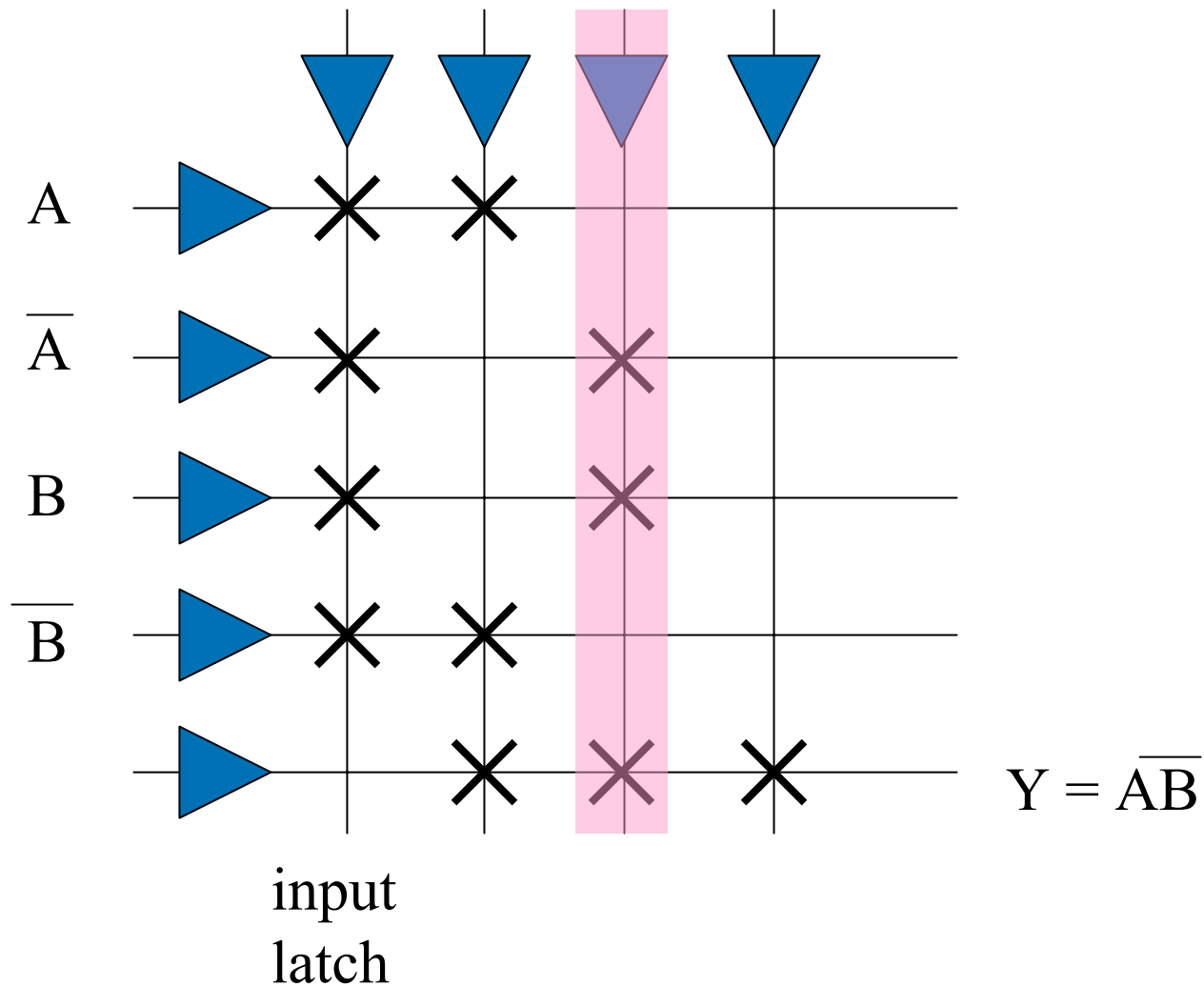
Latch minterm 1 result



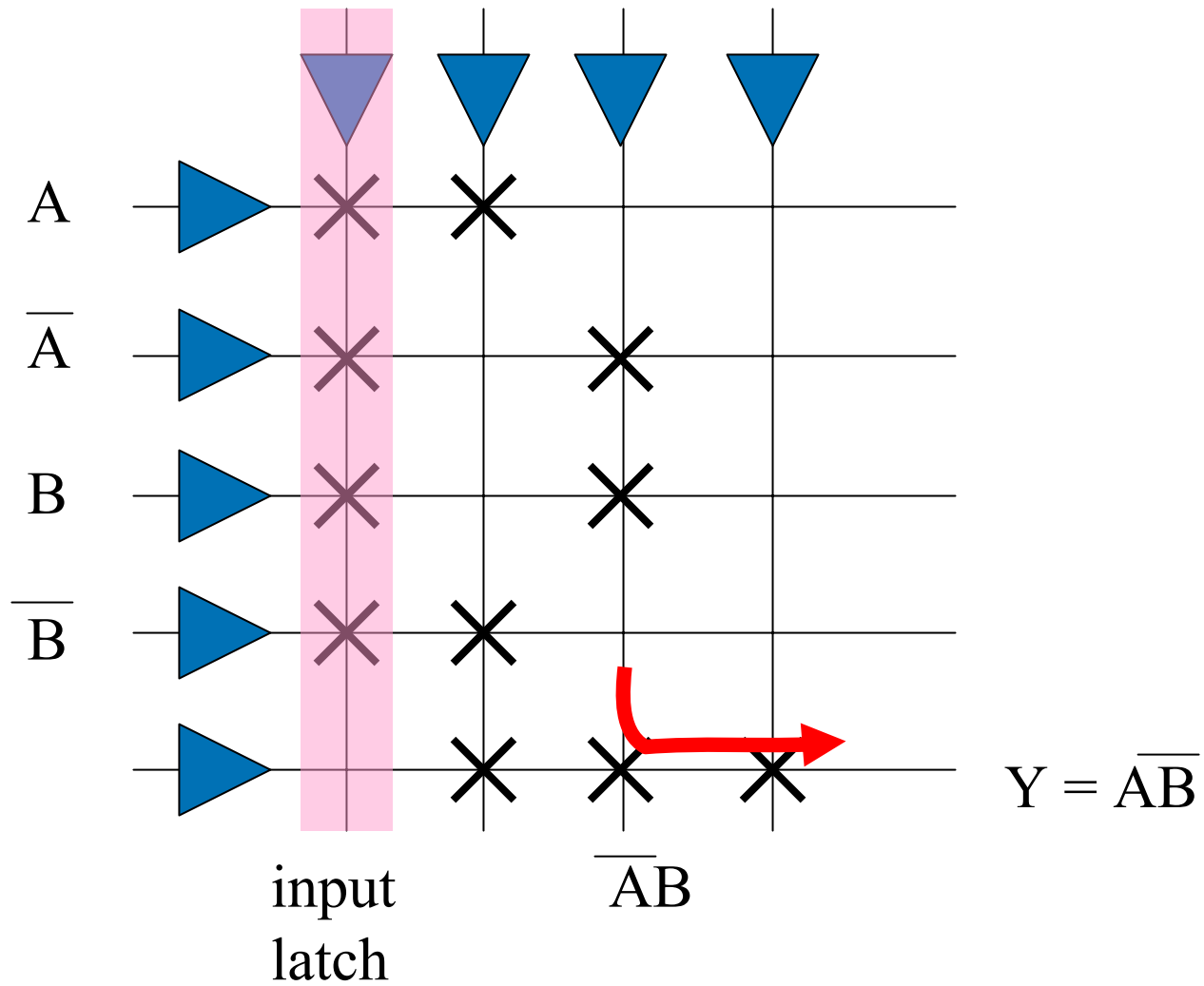
Open minterm 1 switches



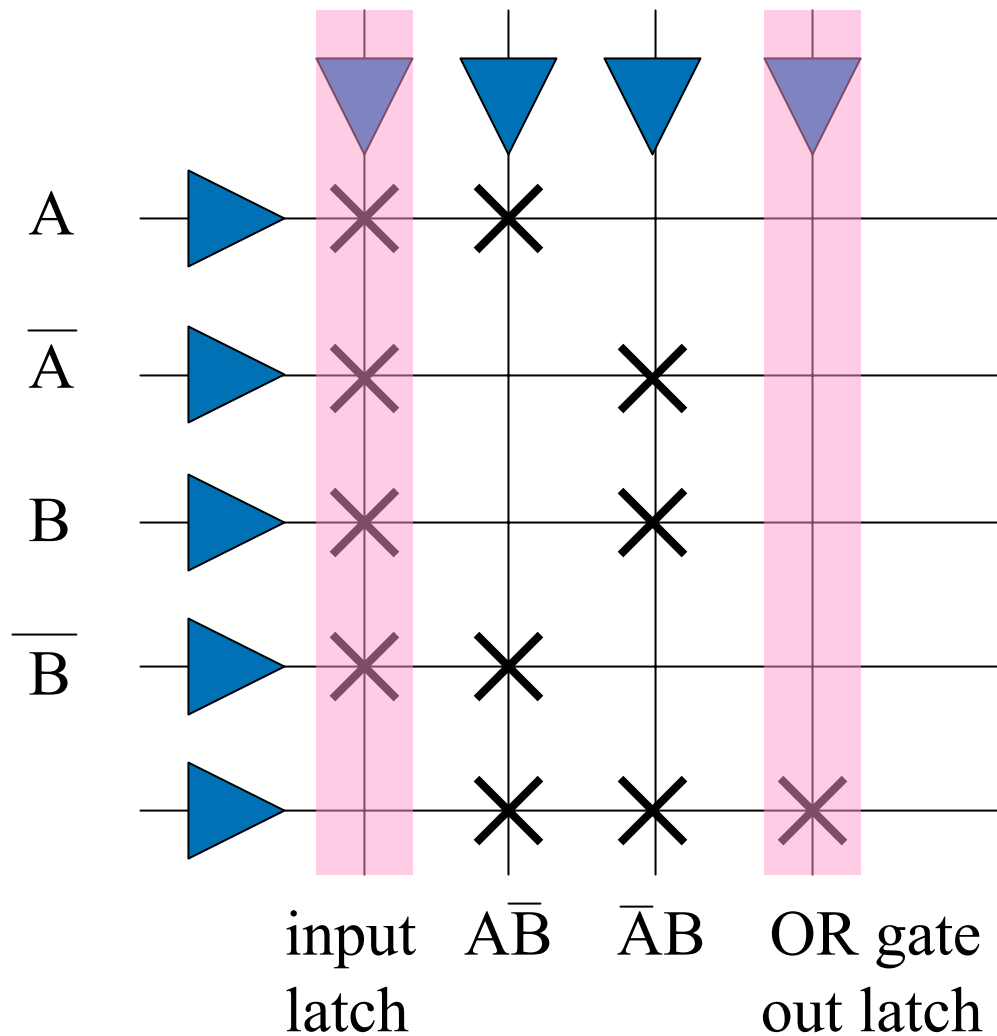
Close minterm 2 switches



Compute minterm 2

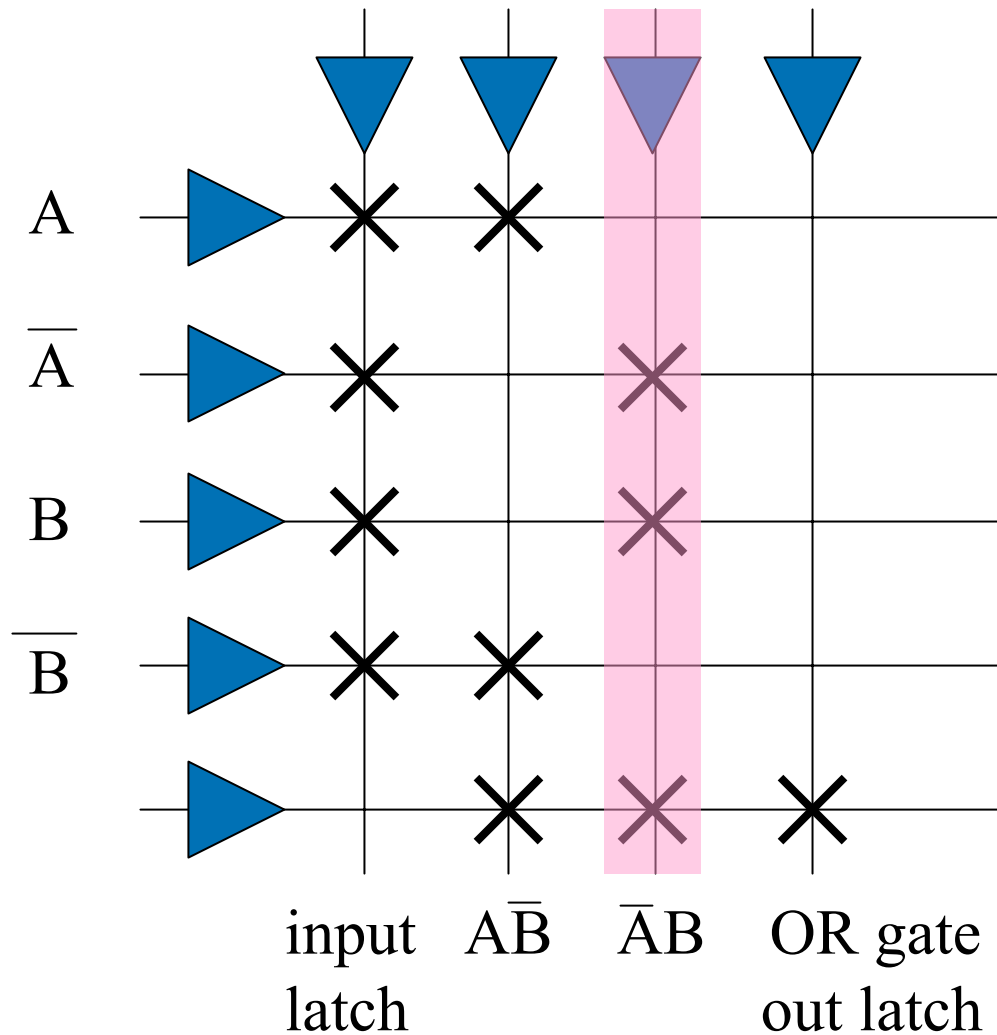


Accumulate Results



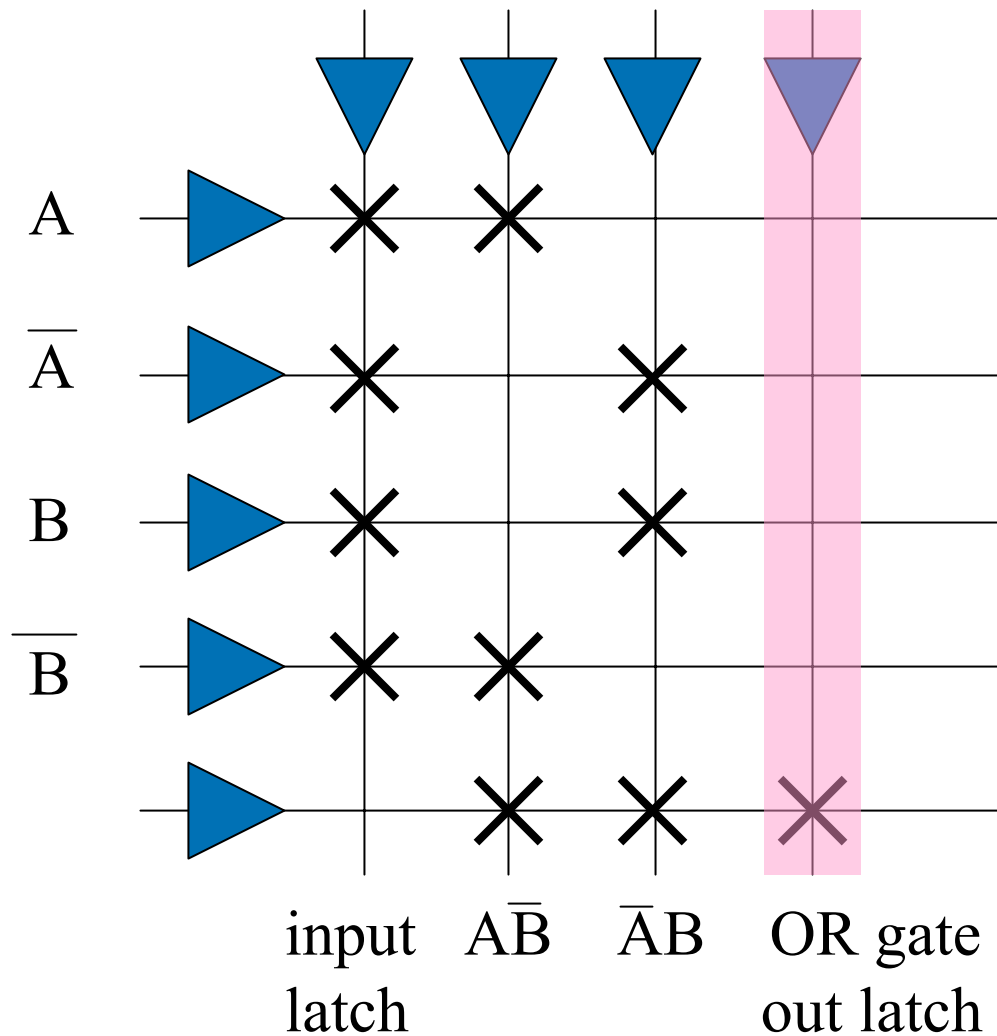
$$Y = \bar{A}\bar{B} + \bar{A}B$$

Open minterm 2 switches



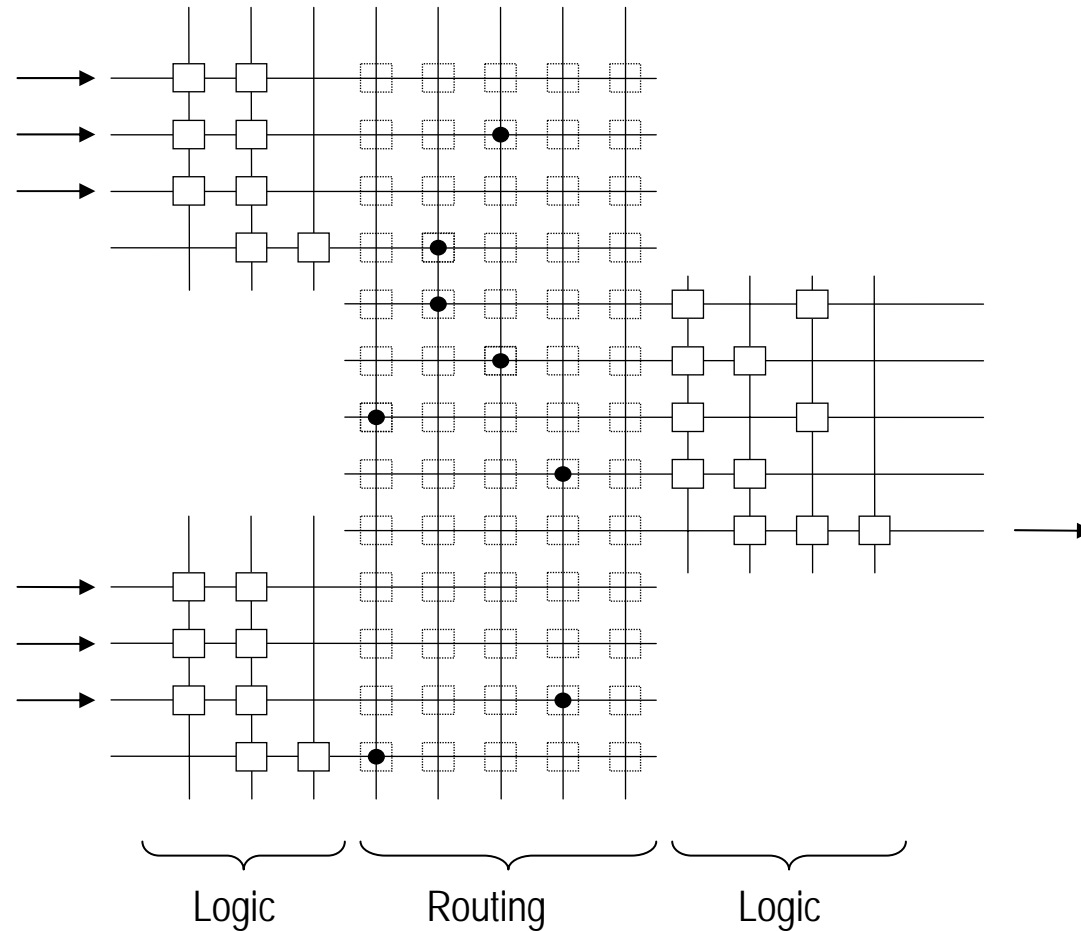
$$Y = A\bar{B} + \bar{A}B$$

Read output latch



$$Y = \bar{A}\bar{B} + \bar{A}B$$

Hysteretic Resistor Systems



Demo—software tools

- Development environment
- Compiler (C code → nanocircuits)
- Logic simulator
- SPICE simulator

These will be open sourced



i n v e n t

Questions and Answers