# HARDWARE IMPLEMENTATION OF A REAL-TIME GENETIC ALGORITHM FOR ADAPTIVE FILTERING APPLICATIONS

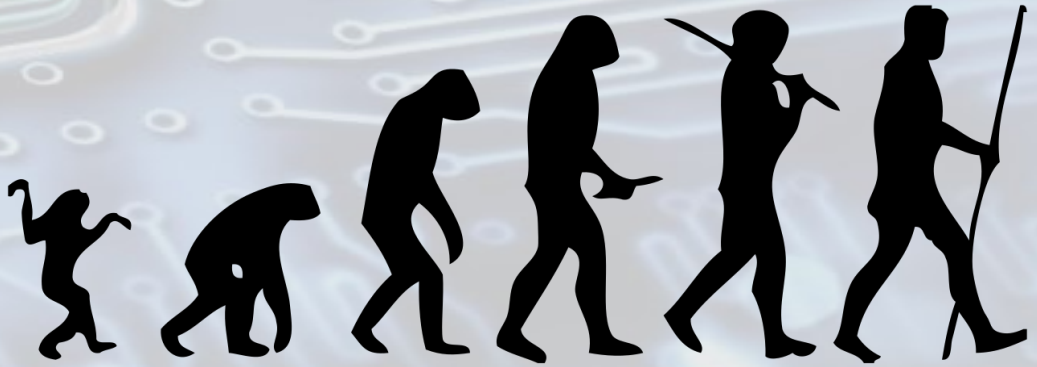
May Buzaglo and Judit Ben Ami

Under the supervision of: Shahar Gino

Jan. 2022

# Outline

- Background
- Project goals
- Alternative solutions
- Selected solution
- Our work
  - Project stages
  - Architectural design
  - Innovations
  - Results
    - Verification
    - Synthesis
    - Performance
- Summary
- Next steps



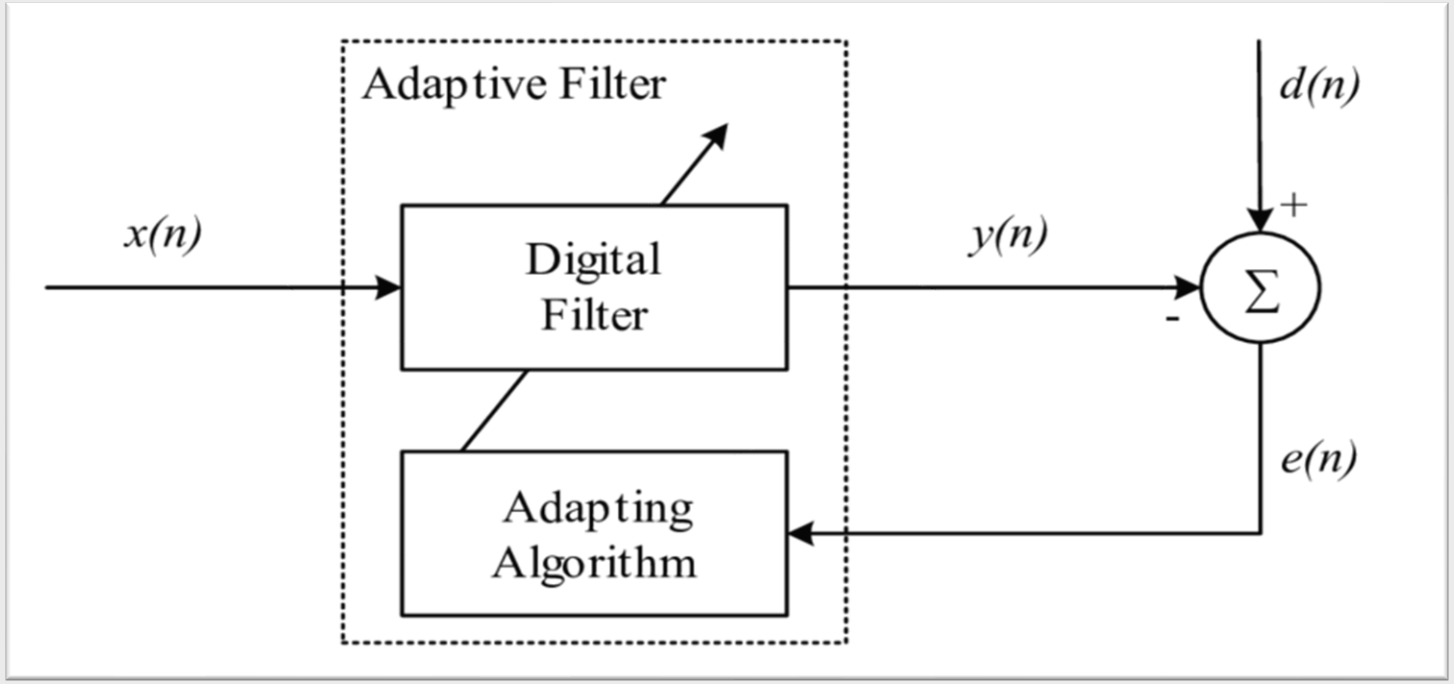
# Background

*Hardware Implementation of a Real-time Genetic Algorithm for Adaptive Filtering Applications*

- **Adaptive filtering**
  - **ARMA Model**
- **Genetic algorithms**

# Adaptive Filtering

- **Adaptive filter models the relationship between its input and output signals.**




$$e[n] = d[n] - y[n]$$

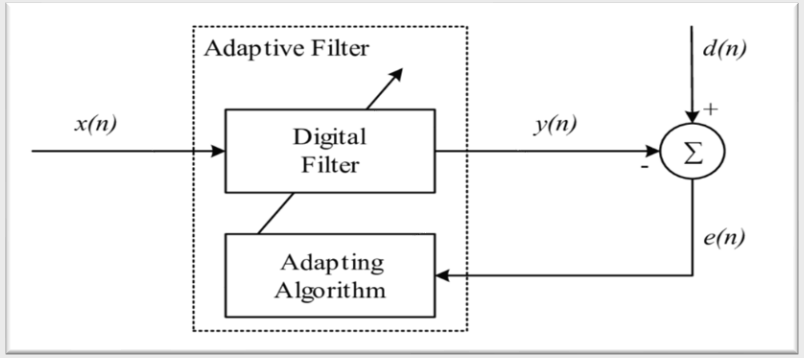
# Adaptive Filtering

- **Adaptive filter models the relationship between its input and output signals.**

$$X[n] = (x[n], x[n-1], \ldots, x[n-M+1])^T$$

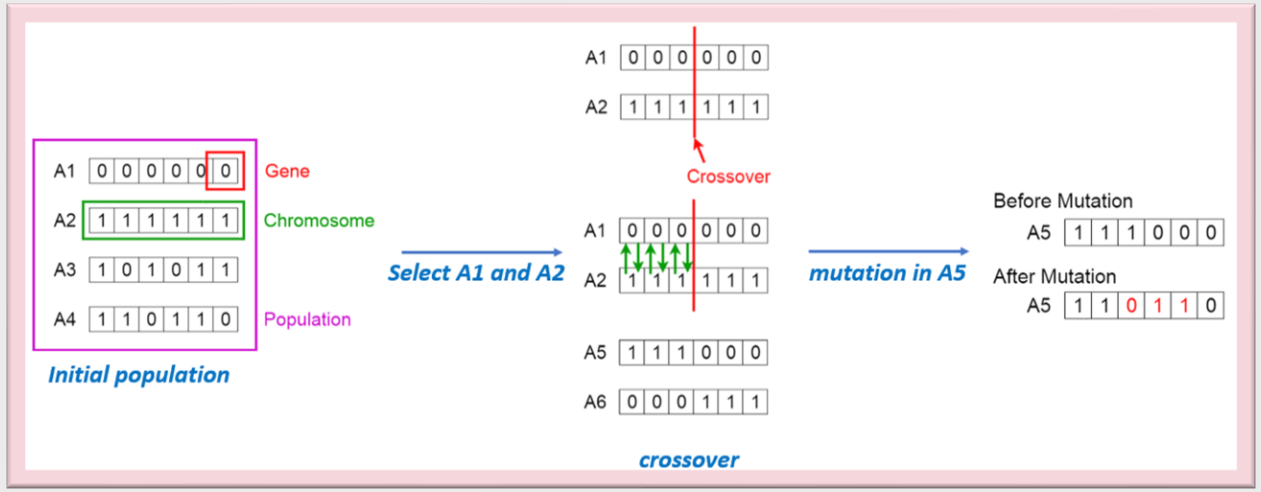
$$W[n] = (w_1[n], w_2[n], \ldots, w_M[n])^T$$

$$y[n] = W[n]X[n]$$

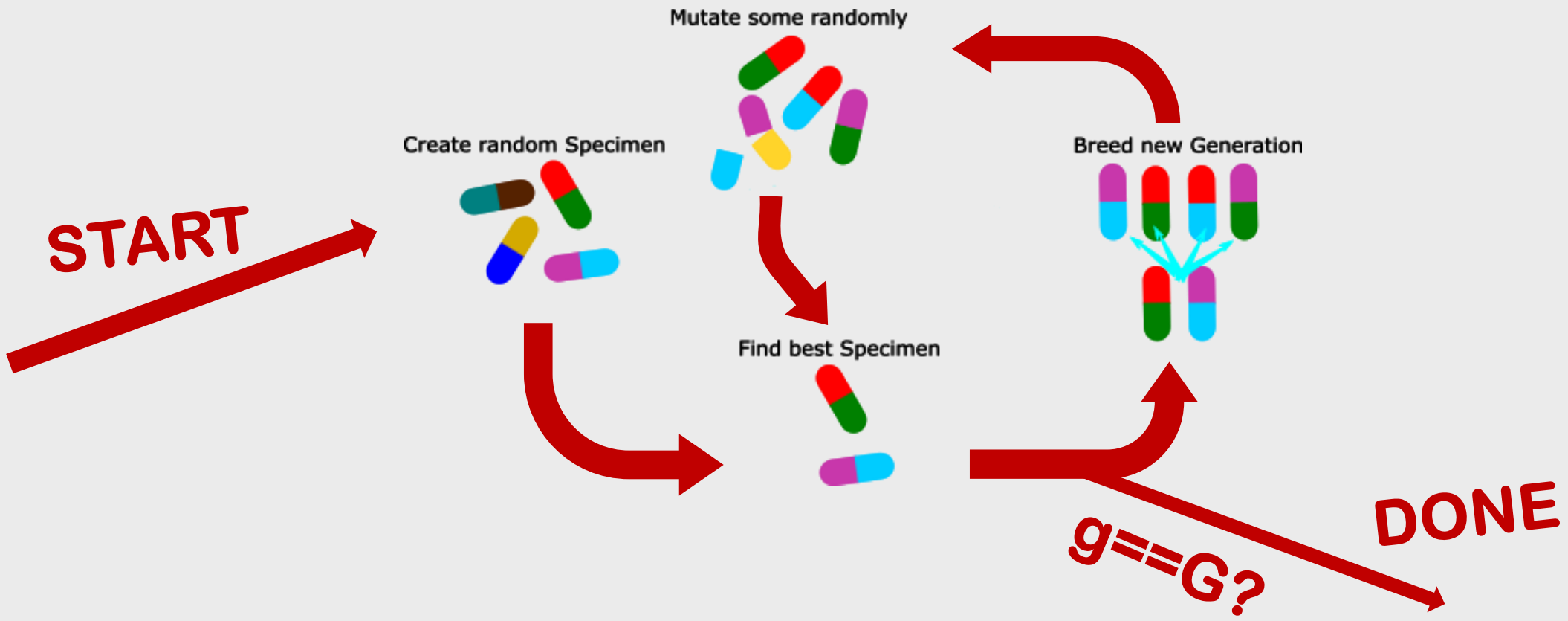


# Genetic Algorithms

- Genetic algorithm (GA) is a search heuristic that is inspired by Charles Darwin's theory of natural evolution.
- Has the ability to find the global solutions for linear and nonlinear.
- Has 5 main stages
  - Create initial population
  - Fitness
  - Selection
  - Crossover
  - Mutation



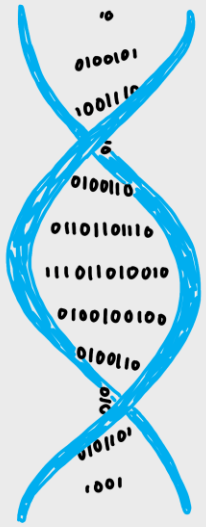
# Genetic Algorithms



# Goal

**Implementing real-time Genetic Algorithm hardware accelerator, based on the article:**

***"Hardware Implementation of a Real-time Genetic Algorithm for Adaptive Filtering Applications"***

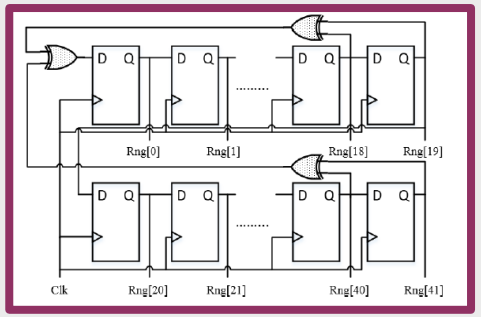


# Alternative Solutions

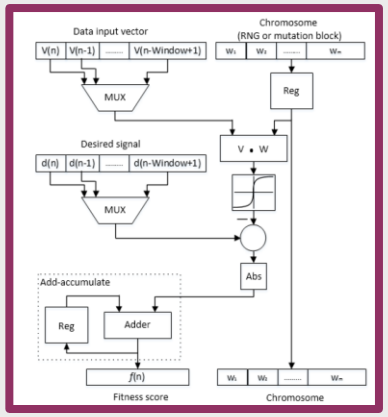
- Software solution
- Different hardware implementations

# Selected Solution

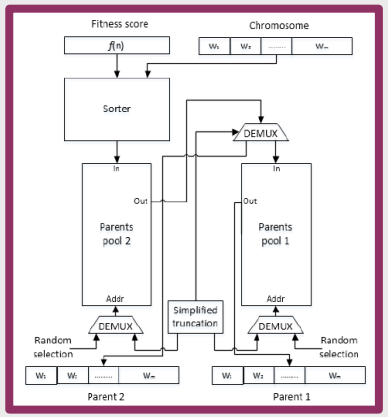
- **Main algo blocks from the article:**



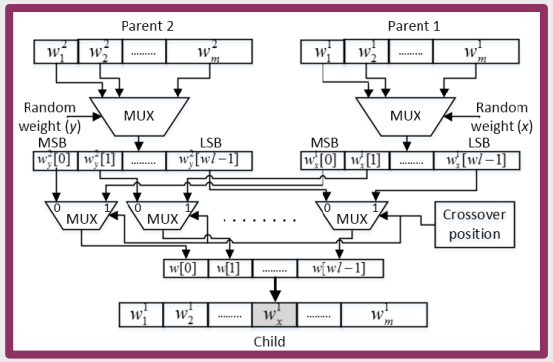

**Create initial population**



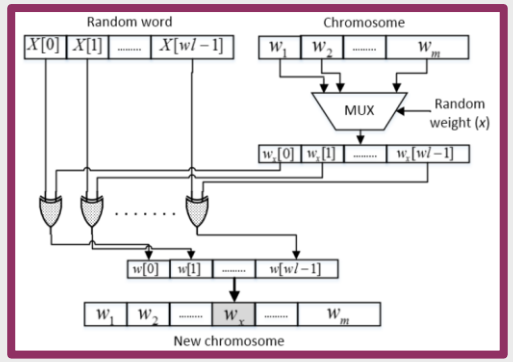

**Fitness**




**Selection**




**Crossover**




**Mutation**

# OUR WORK

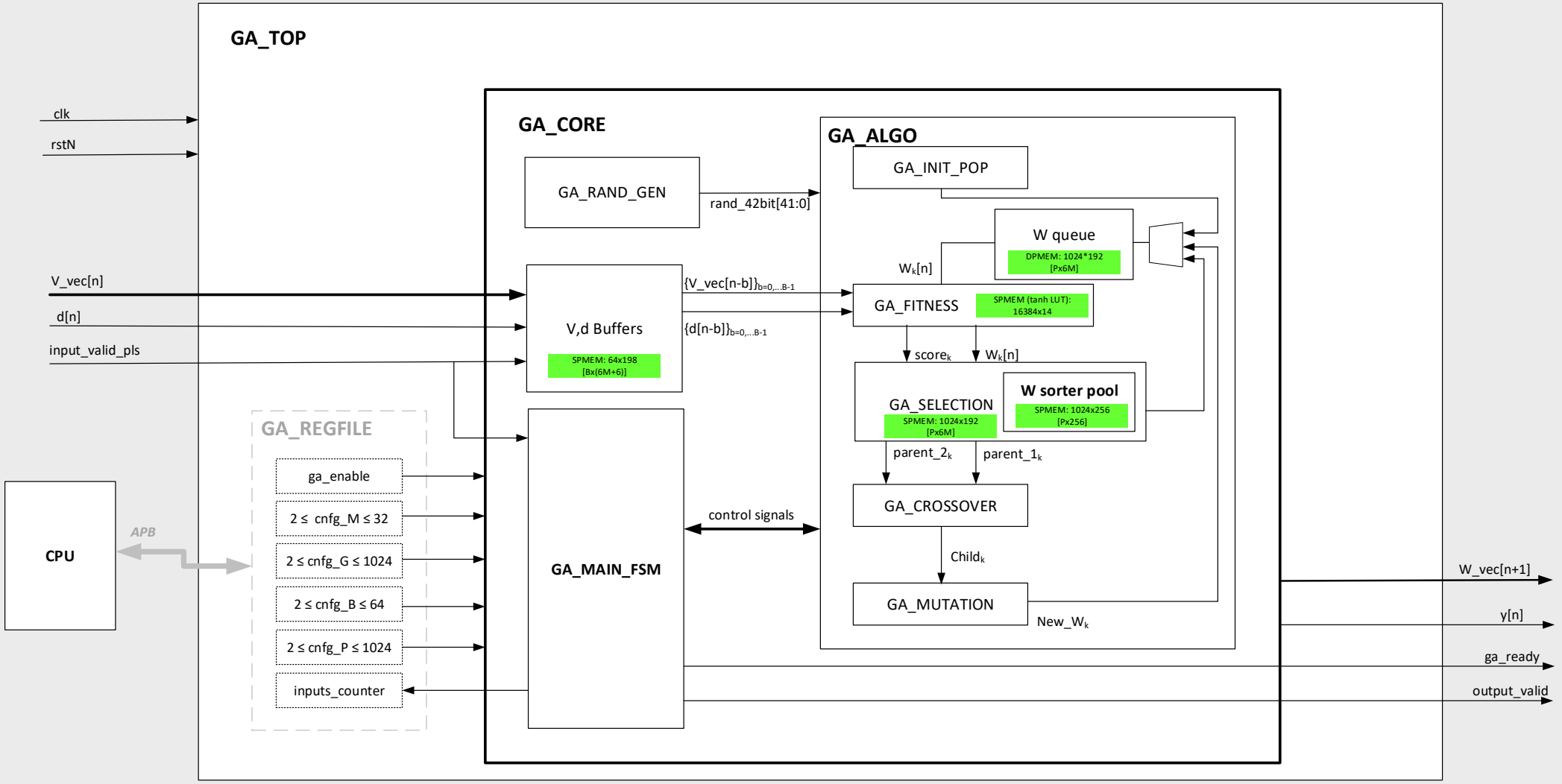
# Steps

- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

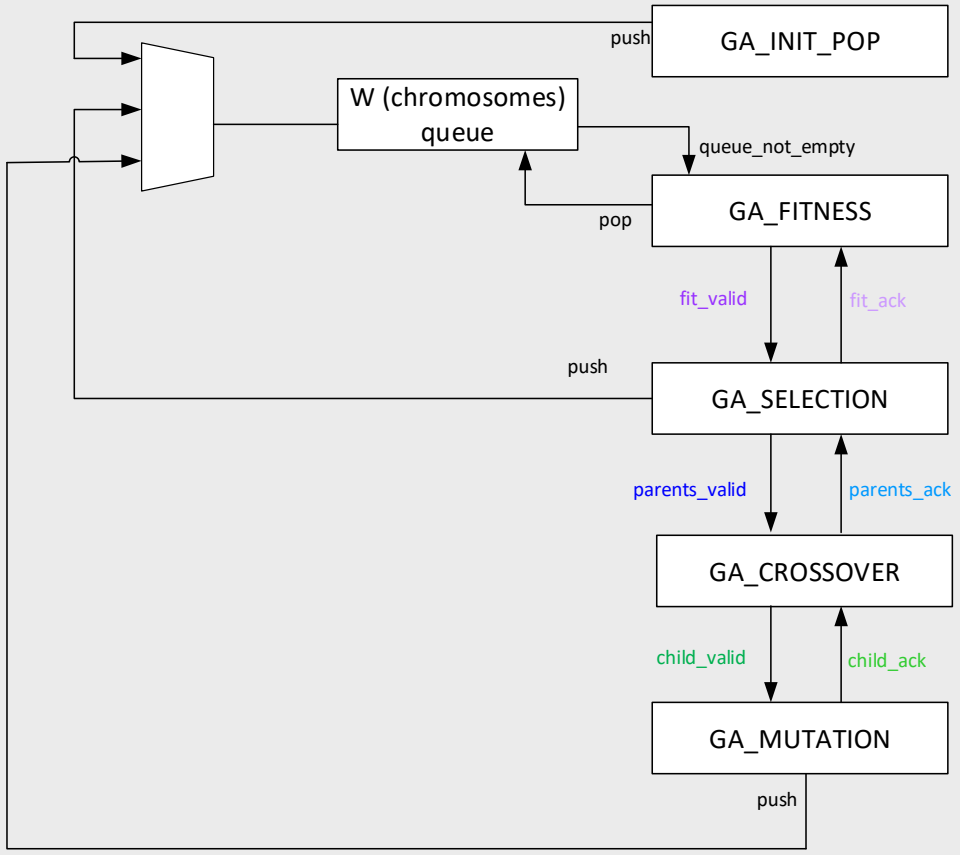
# Steps

- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

# Architectural Design – Top level Diagram



# GA_ALGO




## MAIN FUNCTION

Gets 42 bit random number at every cylce. Outputs all of the chromosomes one-by-one every time a chromosome is ready. Finish when reaches P chromosomes.

Gets a single individual and outputs it with it's fitness function

Gets inidividuals cycle by cycle and start to sort them. After getting all the individuals, outputs cycle-by-cycle pairs of paraents

Gets a pair of parents and creates a child

Gets a child and makes mutation and output the new individual

## MAIN INTERFACE

[relevant regs: cnfg_P, cnfg_M]
**Inputs**: start_pls, rand_42bit
**Outputs**: init_pop_chromosome, push

[relevant regs: cnfg_B]
**Inputs**: fit_enable, queue_not_empty, queue_chromosome, V_vec, d ,fit_ack
**Outputs**: cnfg_max_fit_score, pop, vd_buff_rd_req, vd_buff_rd_idx, fit_chromosome, fit_score, fit_valid

[relevant parameters: cnfg_P]
**Inputs**: selection_enable, rand_42bit[18:0], fit_chromosome, fit_score, fit_valid, parents_ack
**Outputs**: gen_created_pls, fit_ack, parent1, parent2, parents_valid, gen_best_chrom, gen_best_score, selection_chromosome, push

[relevant parameters: cnfg_M]
**Inputs**: cnfg_max_fit_score, gen_best_score, rand_42bit[28:19], parent1, parent2, parents_valid, child_ack
**Outputs**: parents_ack, child+child_valid

[relevant parameters: cnfg_M]
**Inputs**: cnfg_max_fit_score, gen_best_score, rand_42bit[39:29], child+child_valid
**Outputs**: child_ack, chromosome, push

# Architectural Design – Pipeline Diagram

Random created generation

Sorted generation

New generation

| GA_ INIT_POP [Time per chromosome: M/7] | W queue | GA_ FITNESS [Time per chromosome: 13*B] | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection [Time per chromosome: 4] | GA_ CROSSOVER [Time per chromosome: 1] | GA_ MUTATION [Time per chromosome: 1] | |
|---|---|---|---|---|---|---|---|
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (1) |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (2) |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (3) |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (4) |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (5) |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (6) |
| ... | | | | | | | |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (7) |
| ... | | | | | | | |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (8) |
| ... | | | | | | | |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (9) |
| ... | | | | | | | |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (10) |
| ... | | | | | | | |
| GA_ INIT_POP | W queue | GA_ FITNESS | GA_SELECTION – W sorted pool | GA_SELECTION – parents selection | GA_ CROSSOVER | GA_ MUTATION | (11) |

Gen 1 created!

**Chosen parameters:**

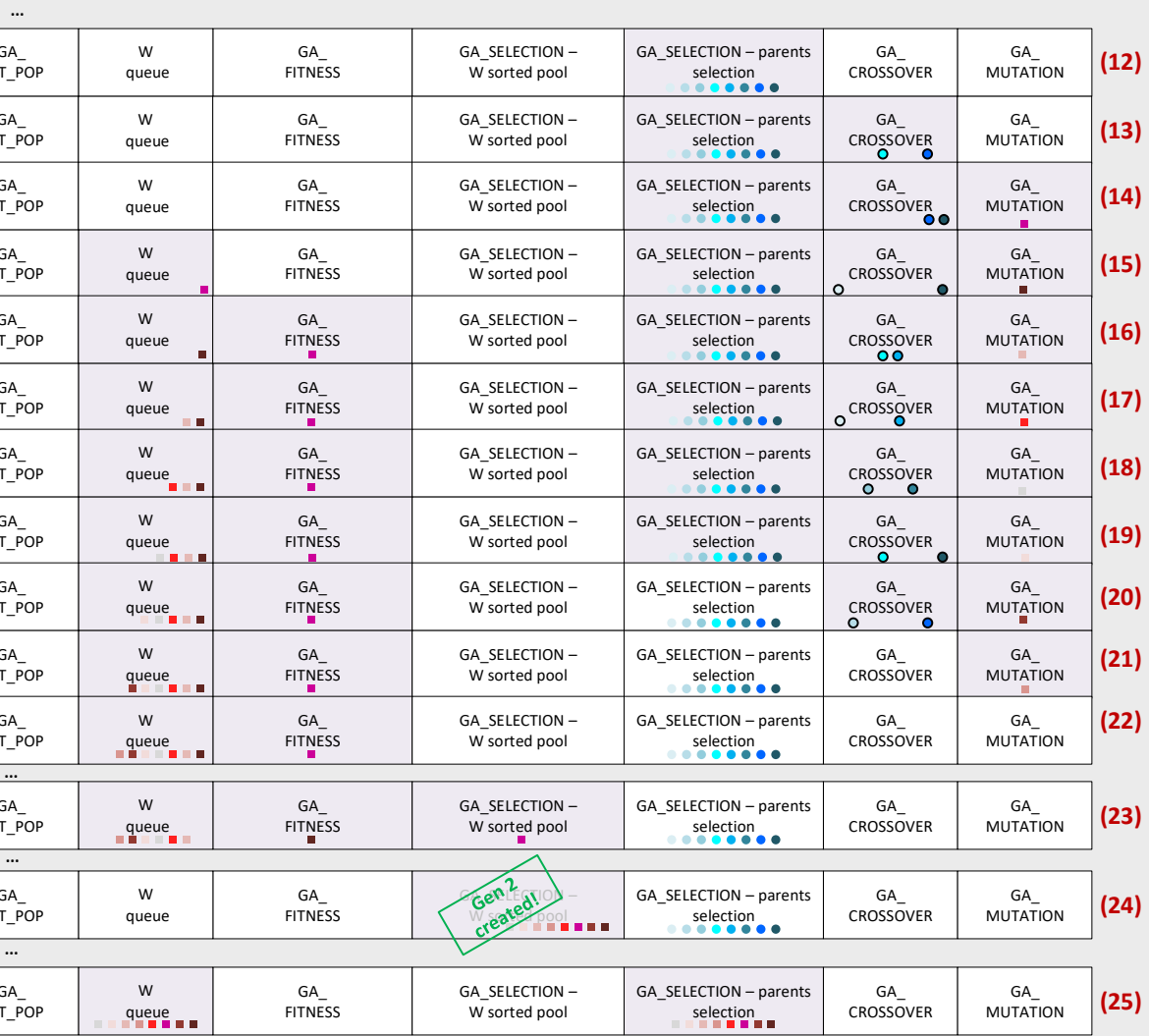
- cnfg_M = 7
- cnfg_B = 4
- cnfg_P = 8
- cnfg_G =2

# Architectural Design – Pipeline Diagram

Random created generation

Sorted generation

New generation



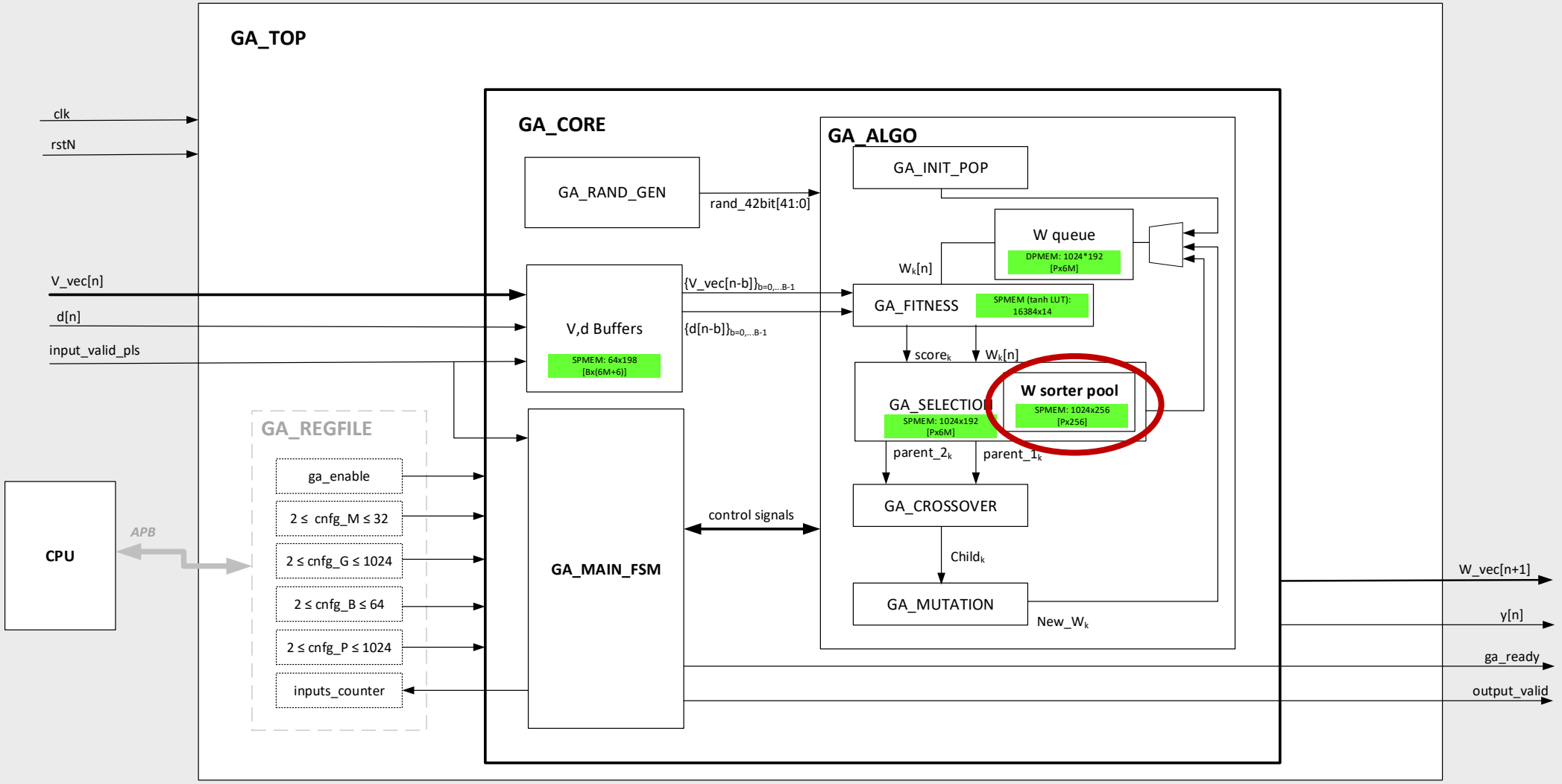

**Chosen parameters:**

- cnfg_M = 7
- cnfg_B = 4
- cnfg_P = 8
- cnfg_G =2

# Innovations

- **Binary-search-tree sorter**
- **Pseudo-modulus**

# Innovations – BST Sorter



# Innovations – BST Sorter

- Functionality
  - insert and ordered traversal
  - serial sorter

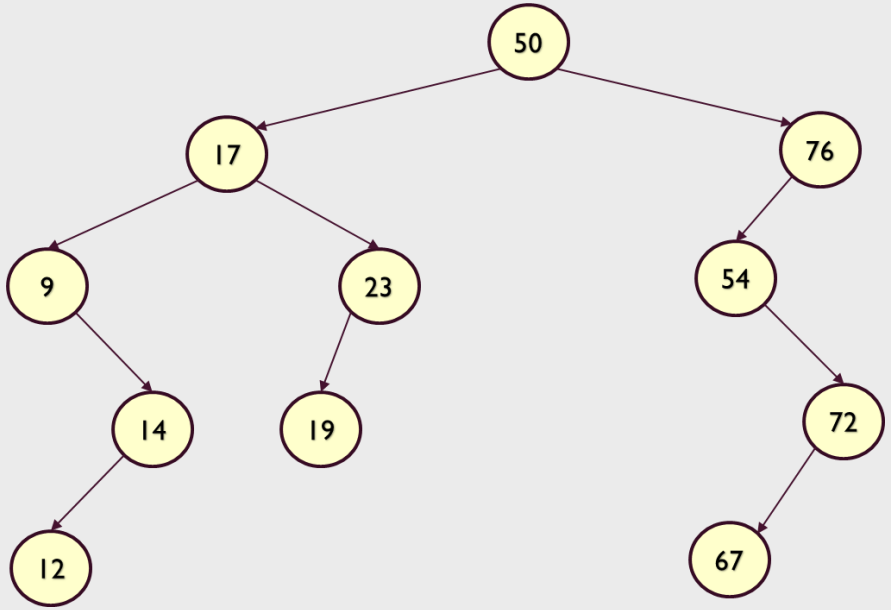
- Performance
  - Fast sorter (1 clk latency) requires a lot of area
    - FF array

  - Area efficient sorter is required
    - Memory

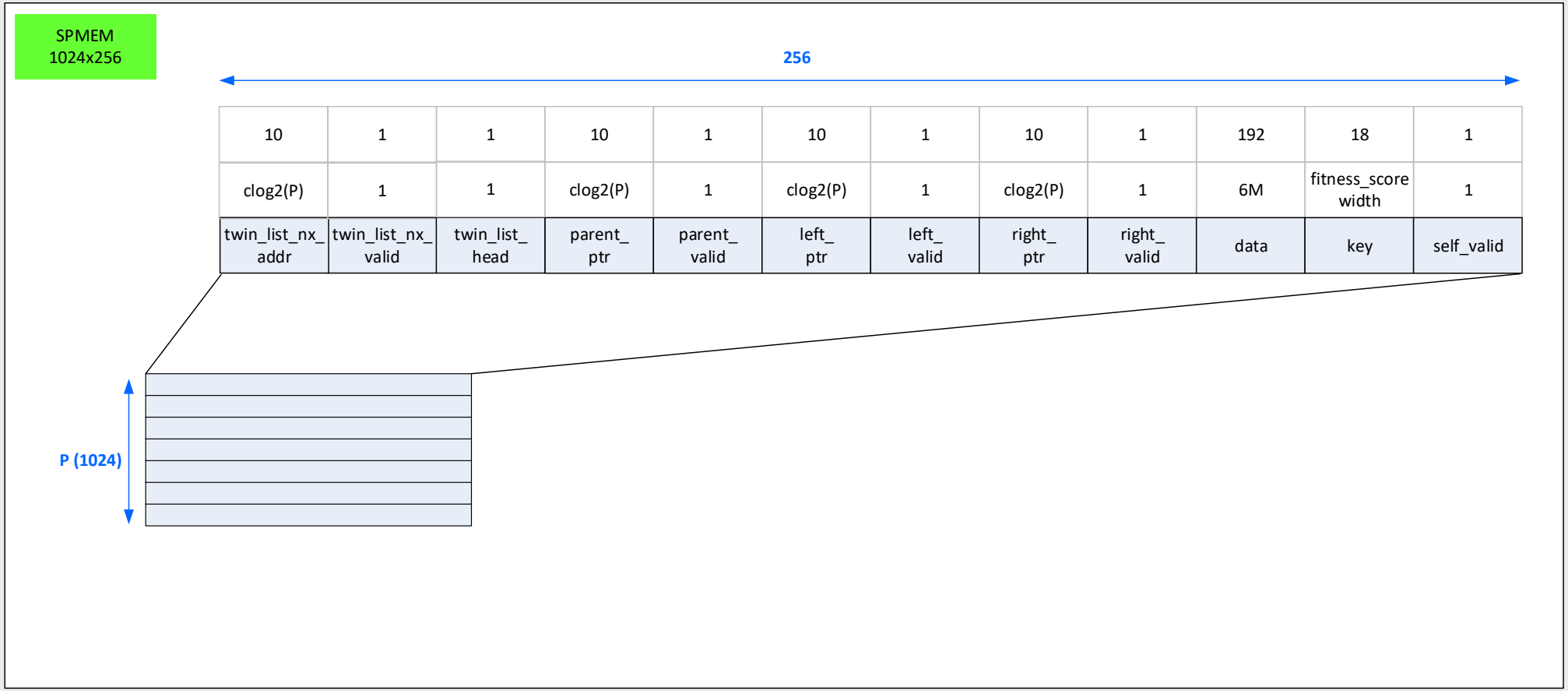
***Chosen solution: binary search tree sorter***

# Innovations – BST Sorter

- **BST sorter:**
  - **Operations:**
    - Insert
    - In-order traversal
  - **For each node:**
    - Right child key > node key
    - Left child key < node key

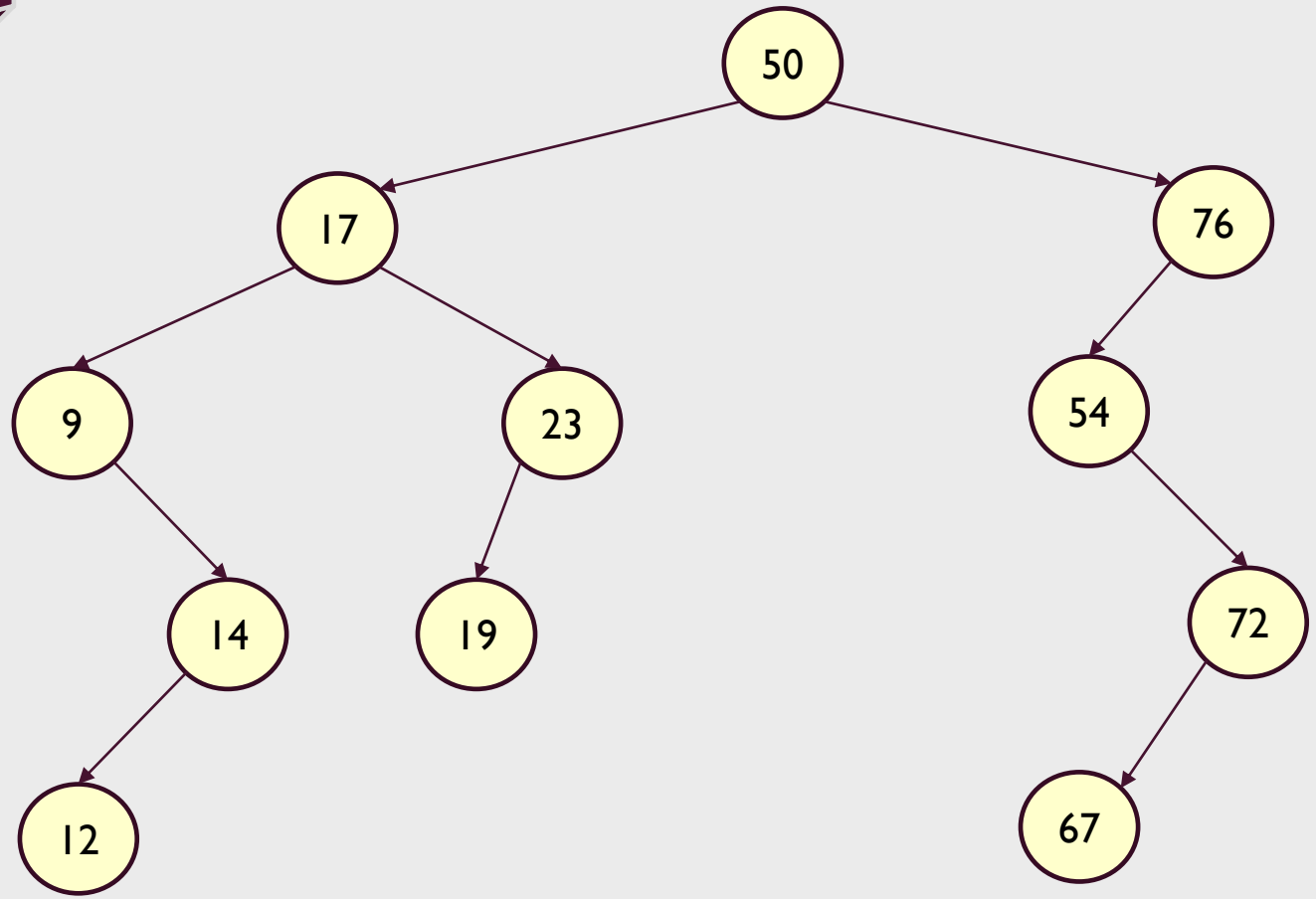


# BST Sorter – Memory



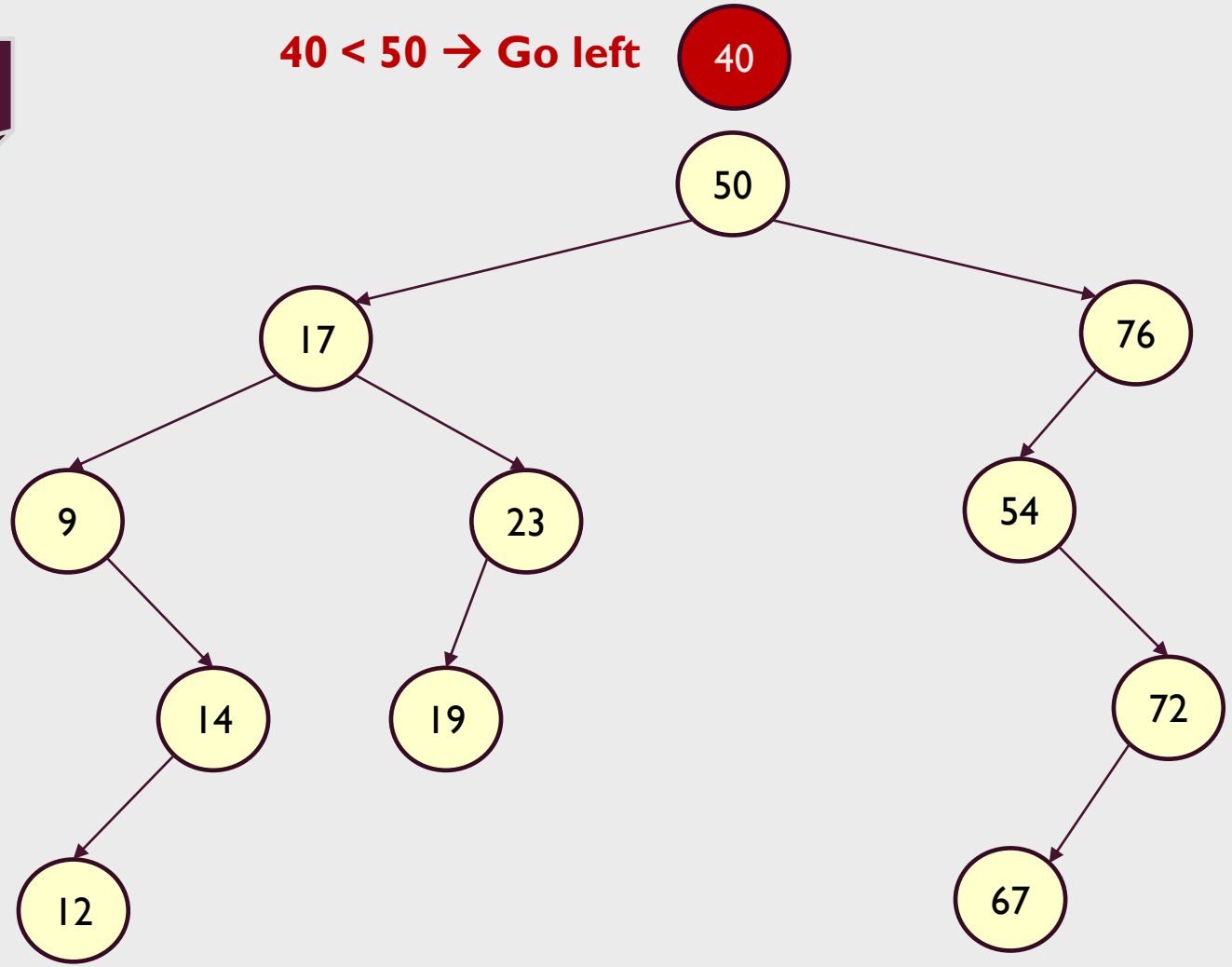
# Innovations – BST Sorter

Insert Example



# Innovations – BST Sorter

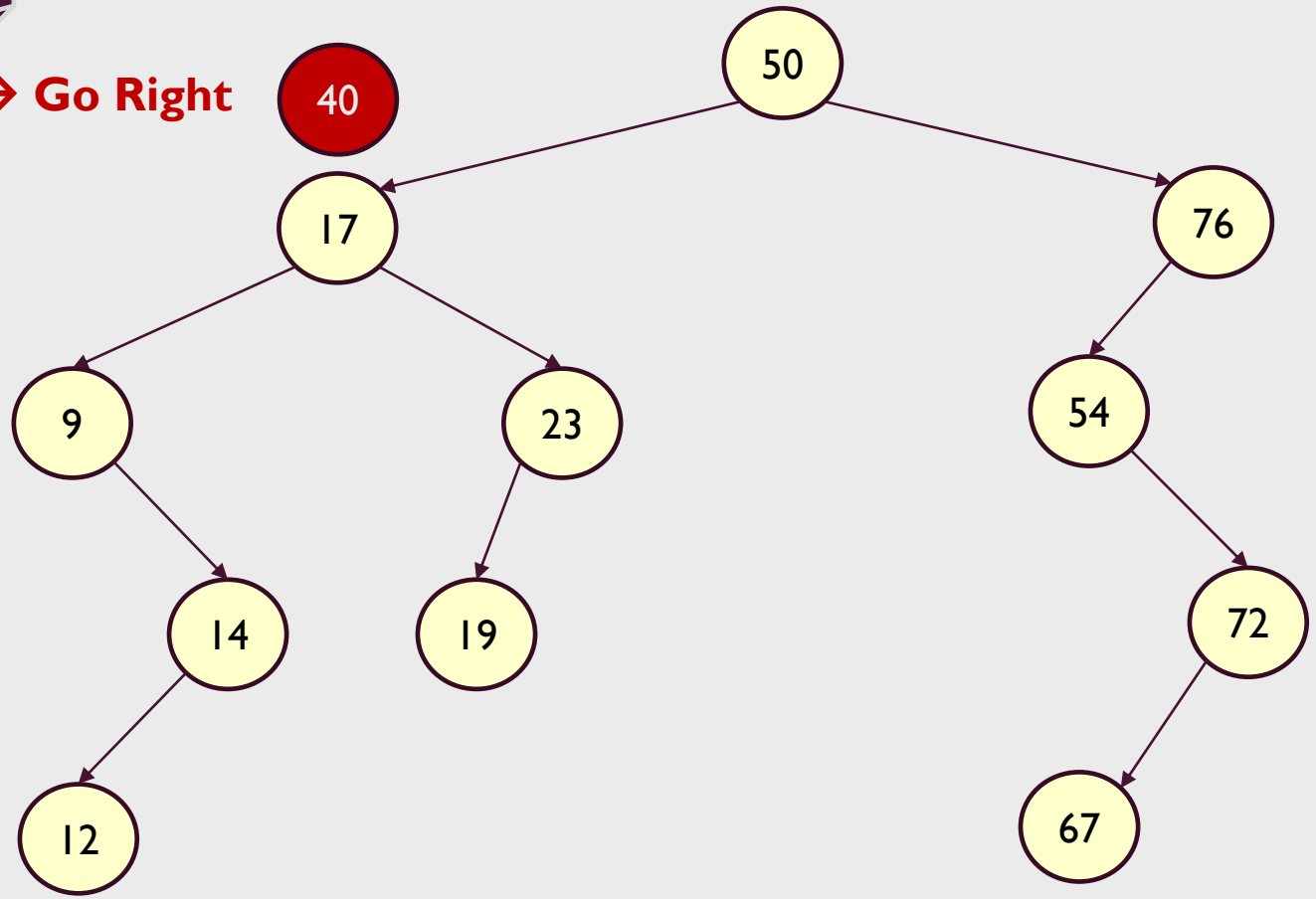
Insert Example



# Innovations – BST Sorter

## Insert Example

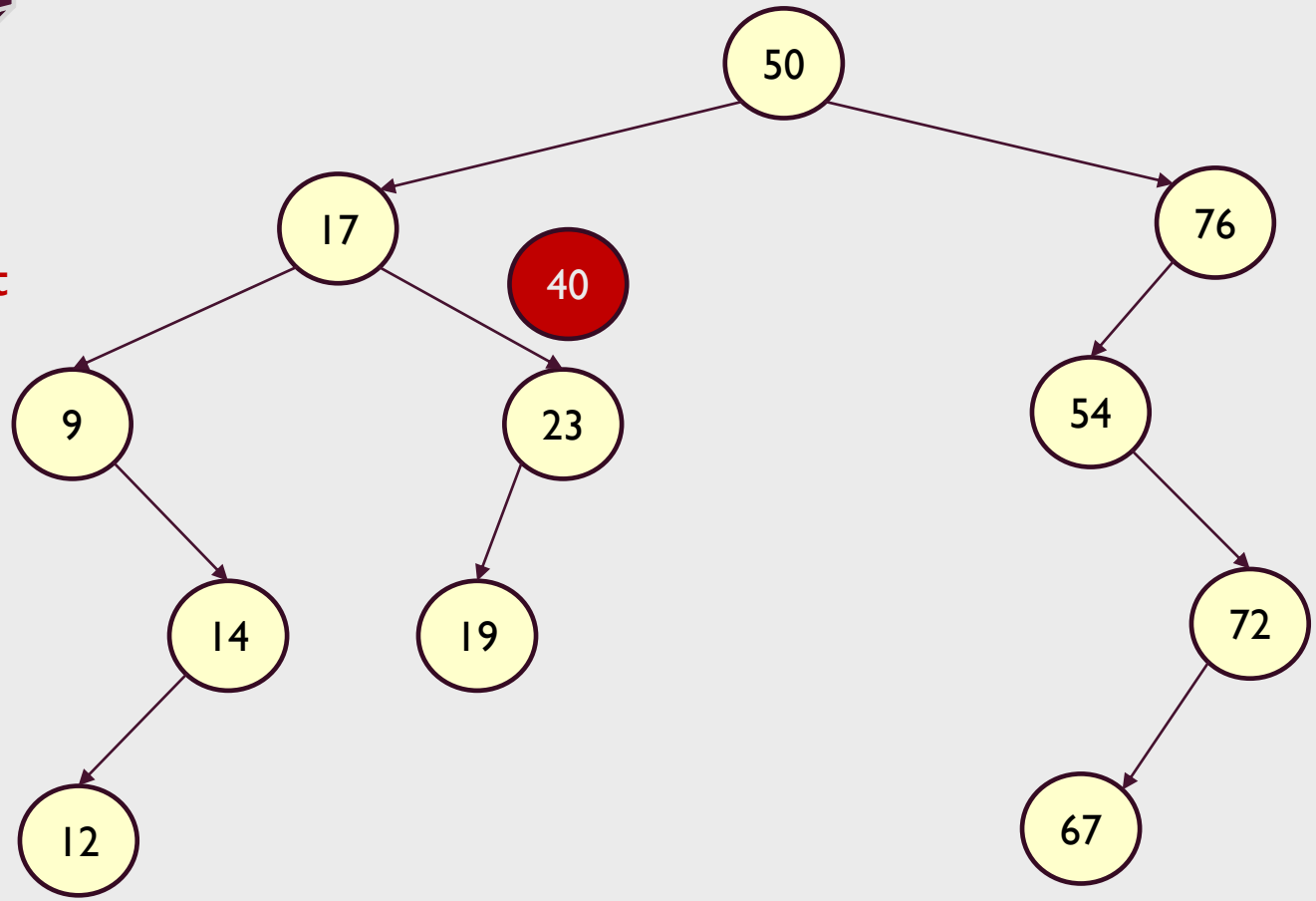
**40 > 17 → Go Right**



# Innovations – BST Sorter

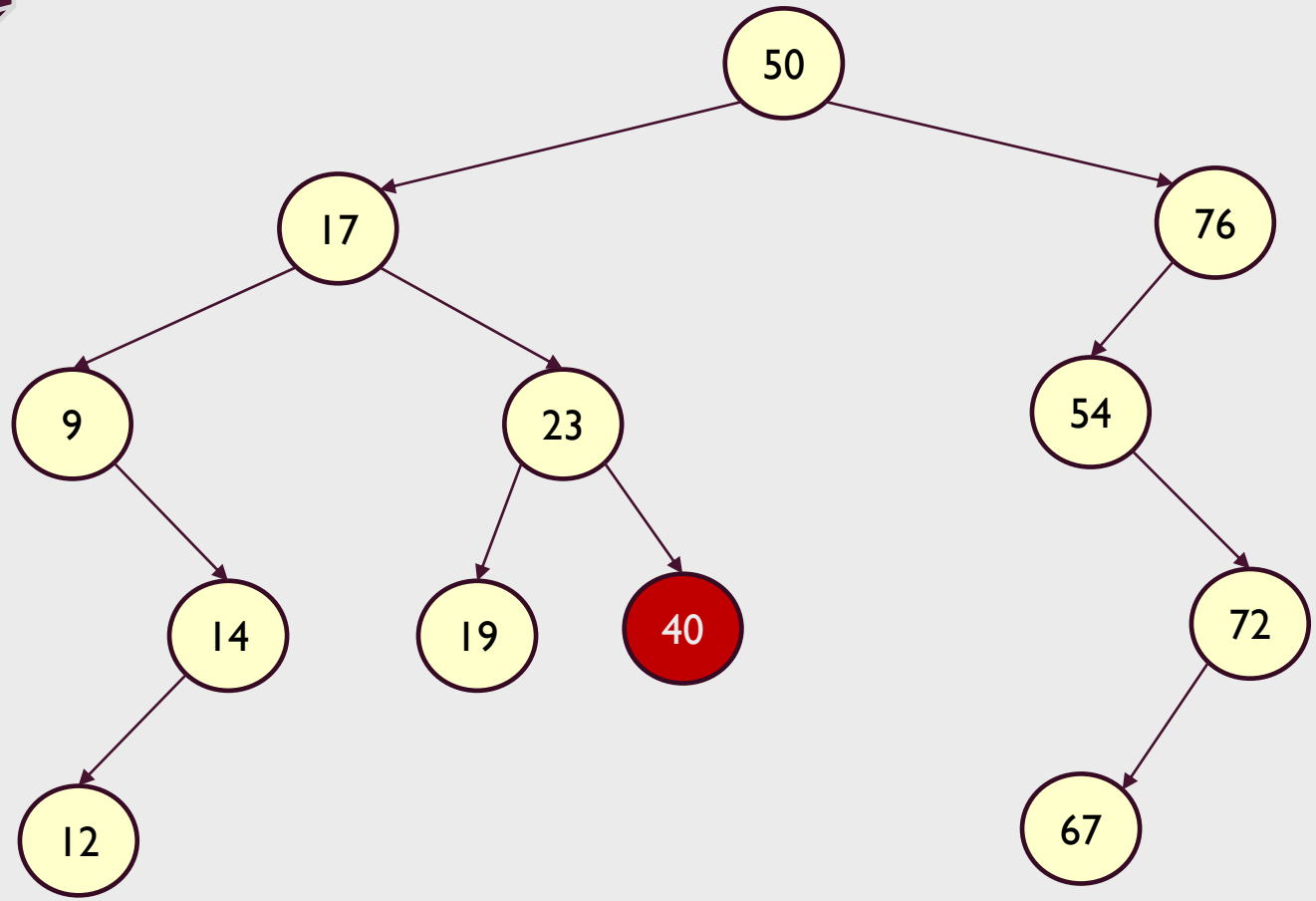
## Insert Example

**40 > 23 → Go Right**



# Innovations – BST Sorter

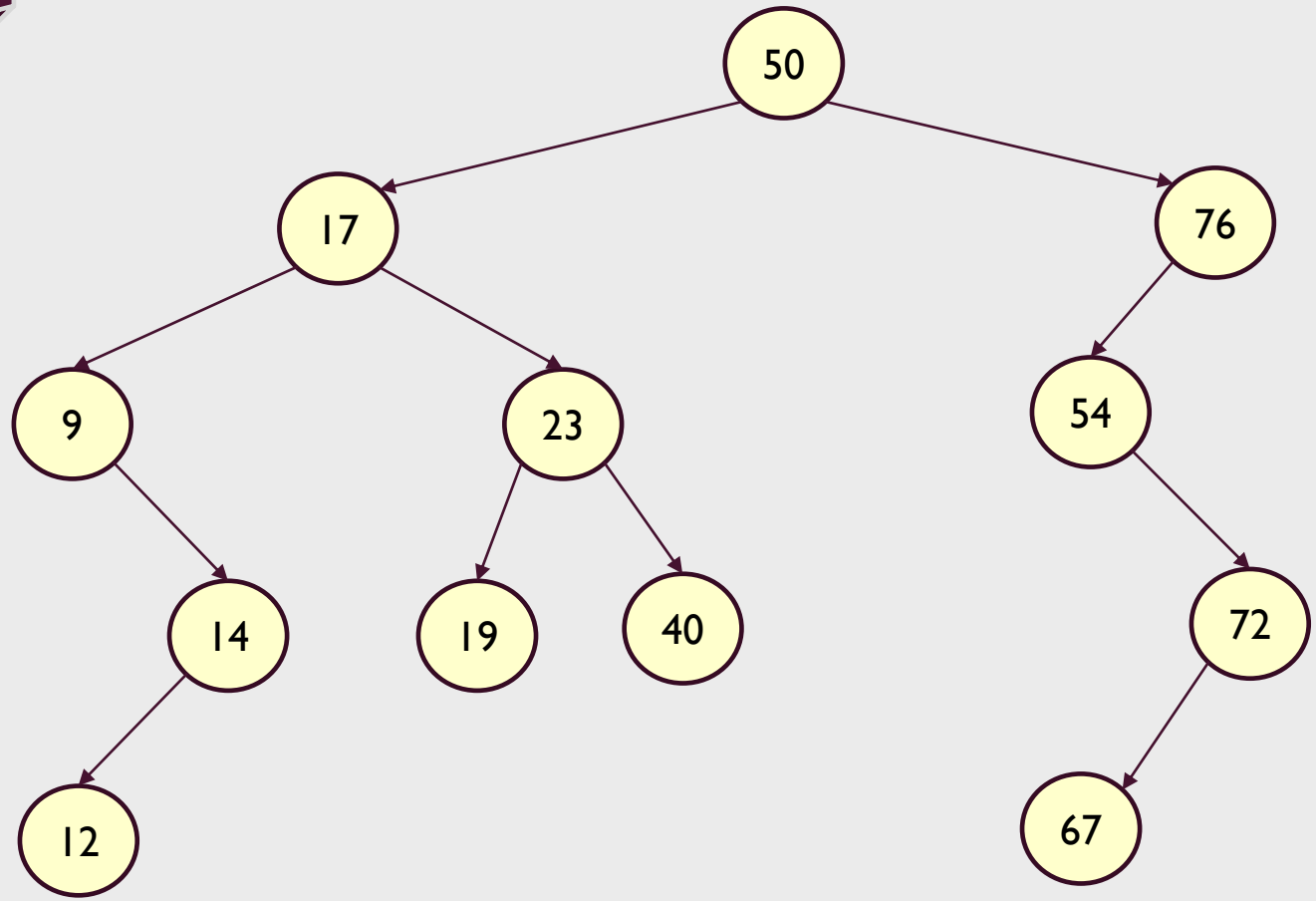
Insert Example




**Found place!**

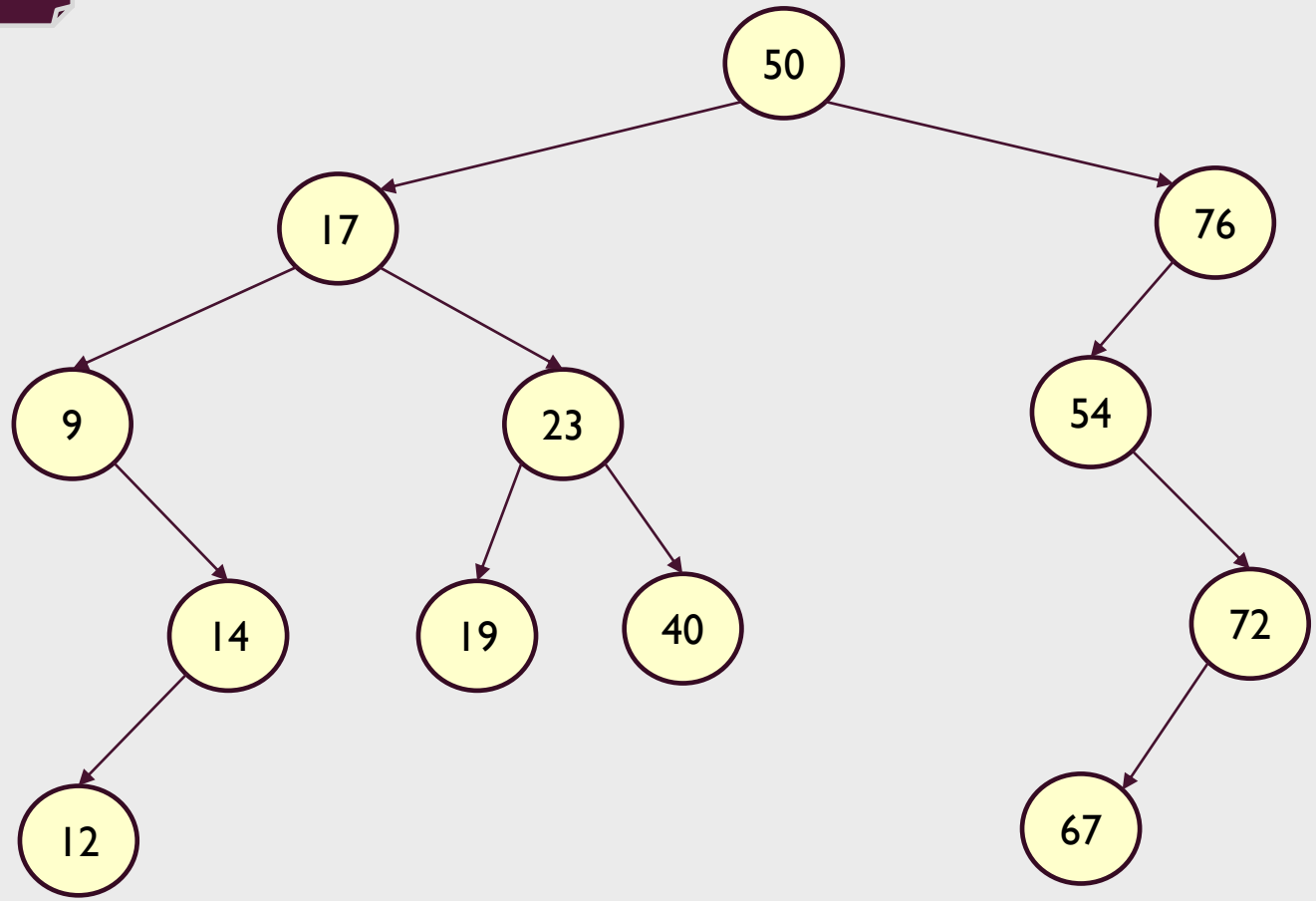
# Innovations – BST Sorter

Insert Example



# Innovations – BST Sorter

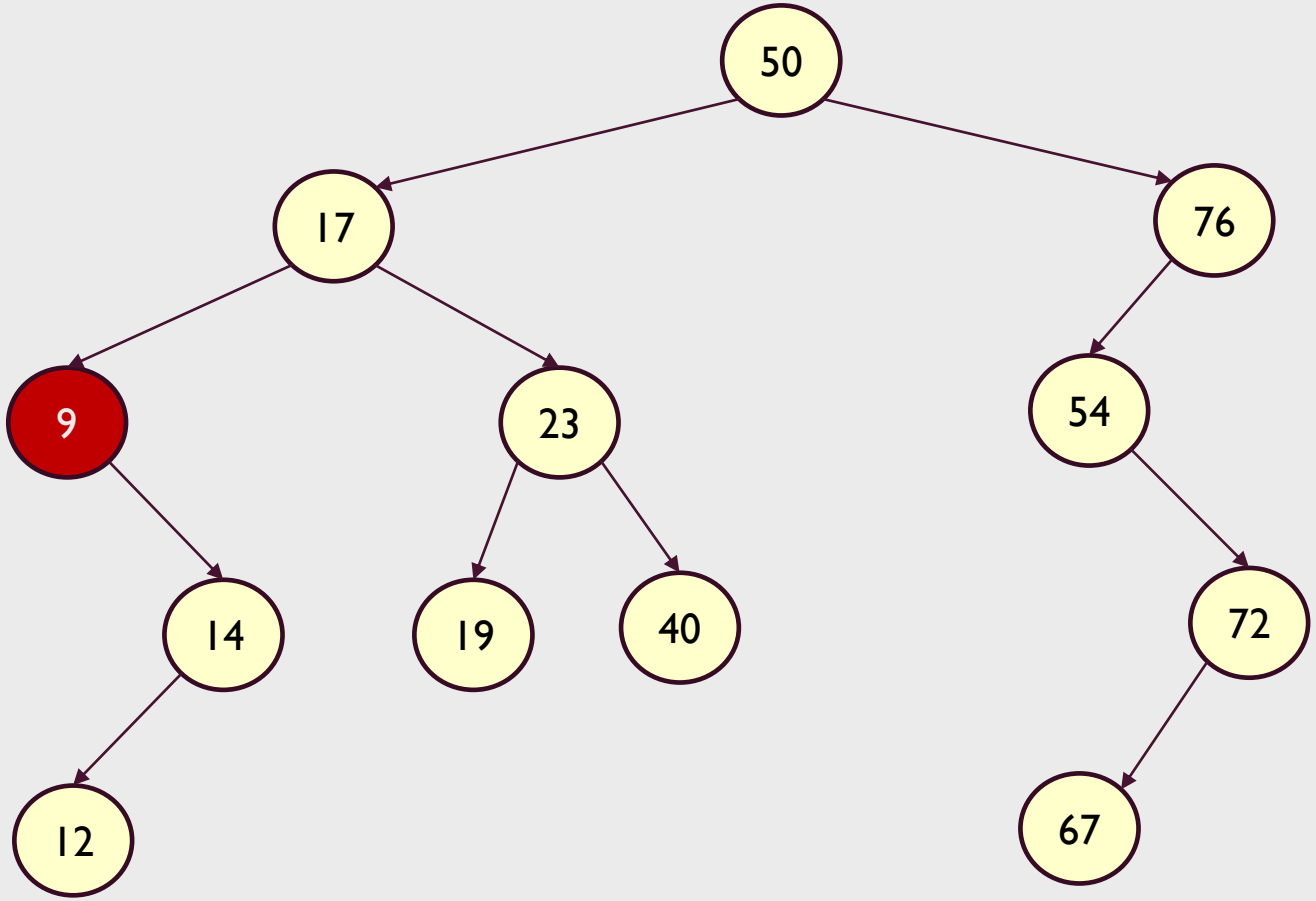
## Traversal Example



# Innovations – BST Sorter

## Traversal Example

9

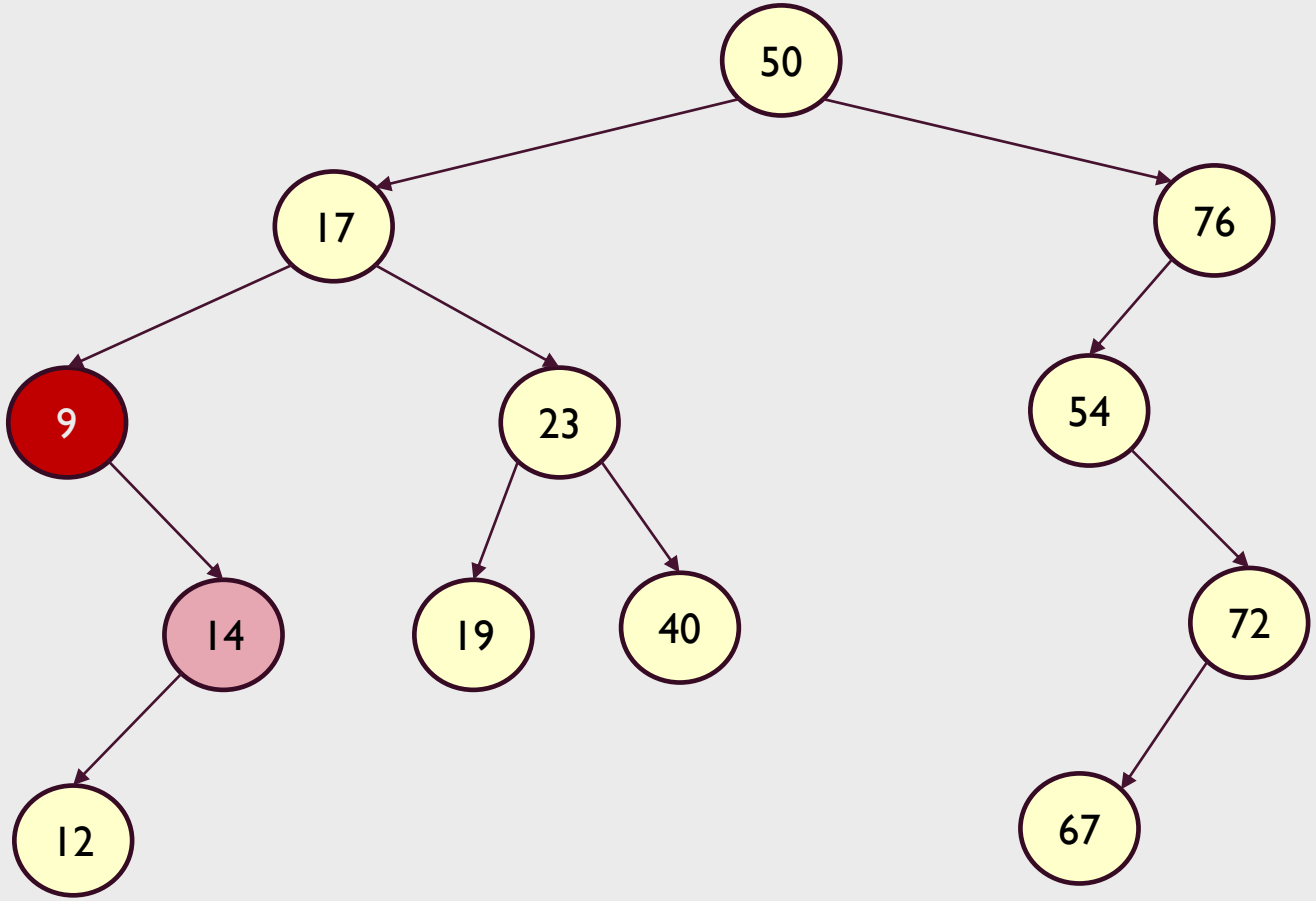



**Inorder traversal:**
**Start with min element.**
**Has right sub tree → find its min**

# Innovations – BST Sorter

## Traversal Example

9

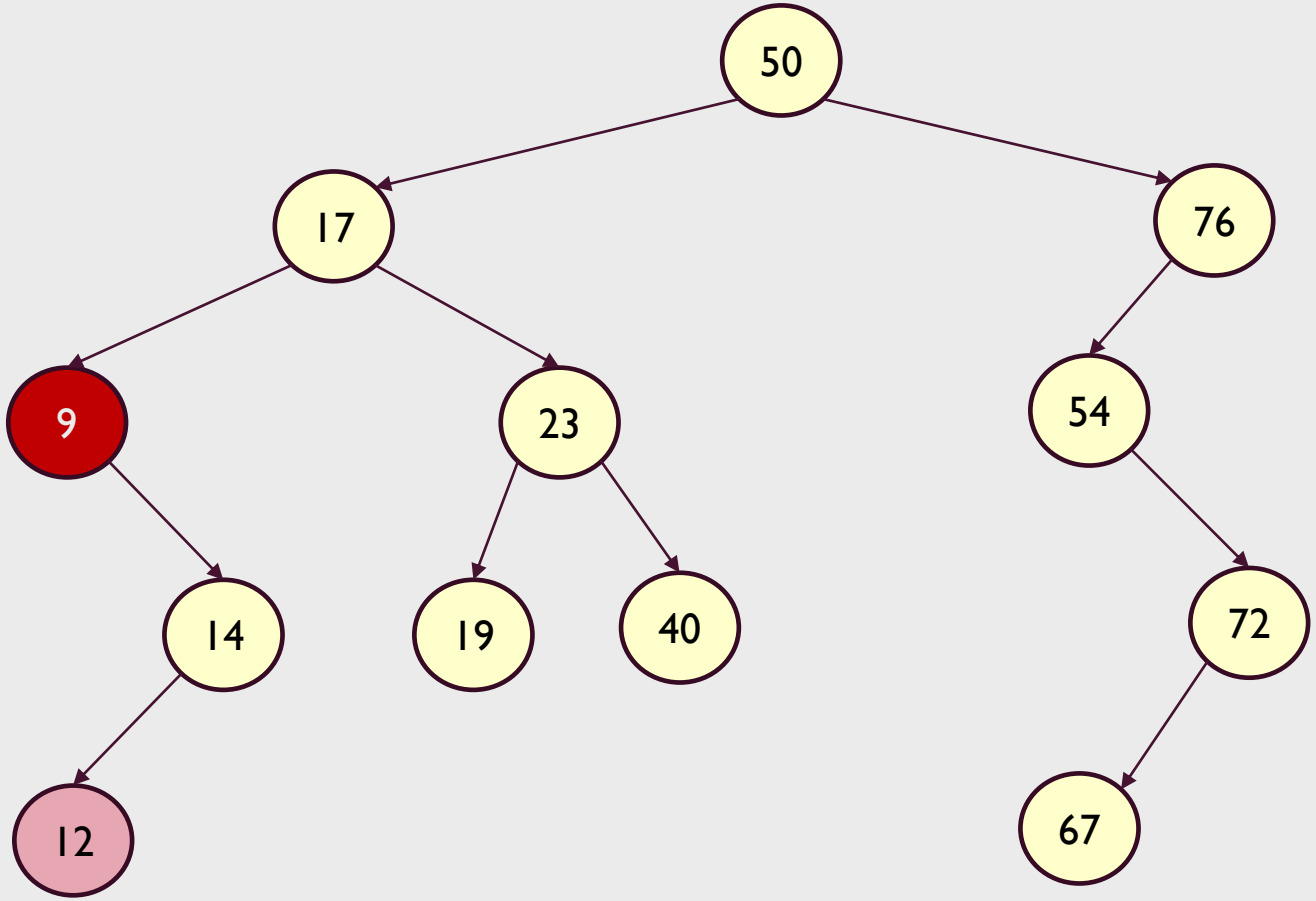



**Founding min: go down left**

# Innovations – BST Sorter

## Traversal Example

9

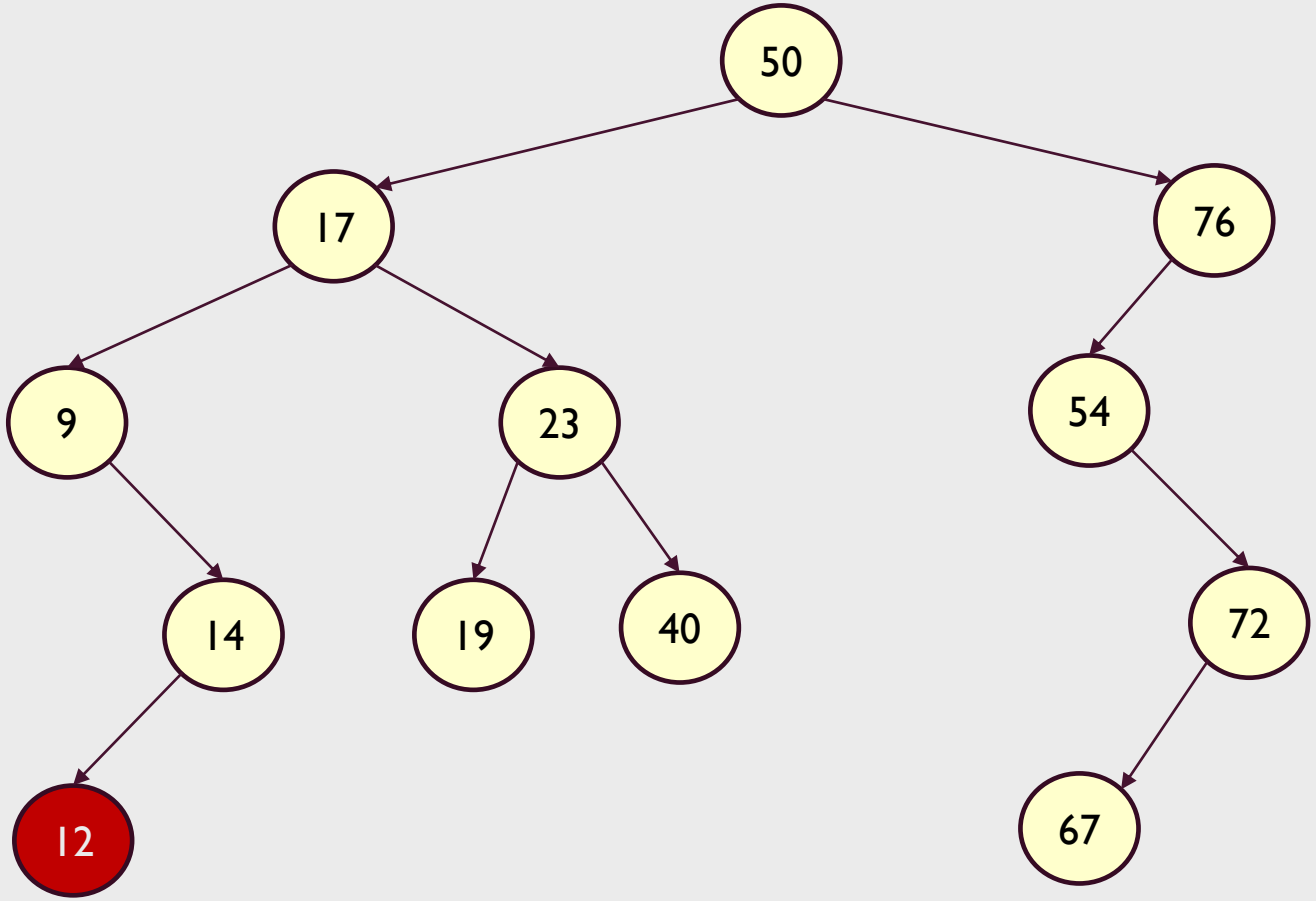



**Founding min: go down left.**
**no more left → min**
**found → found nx!**

# Innovations – BST Sorter

## Traversal Example

9 12

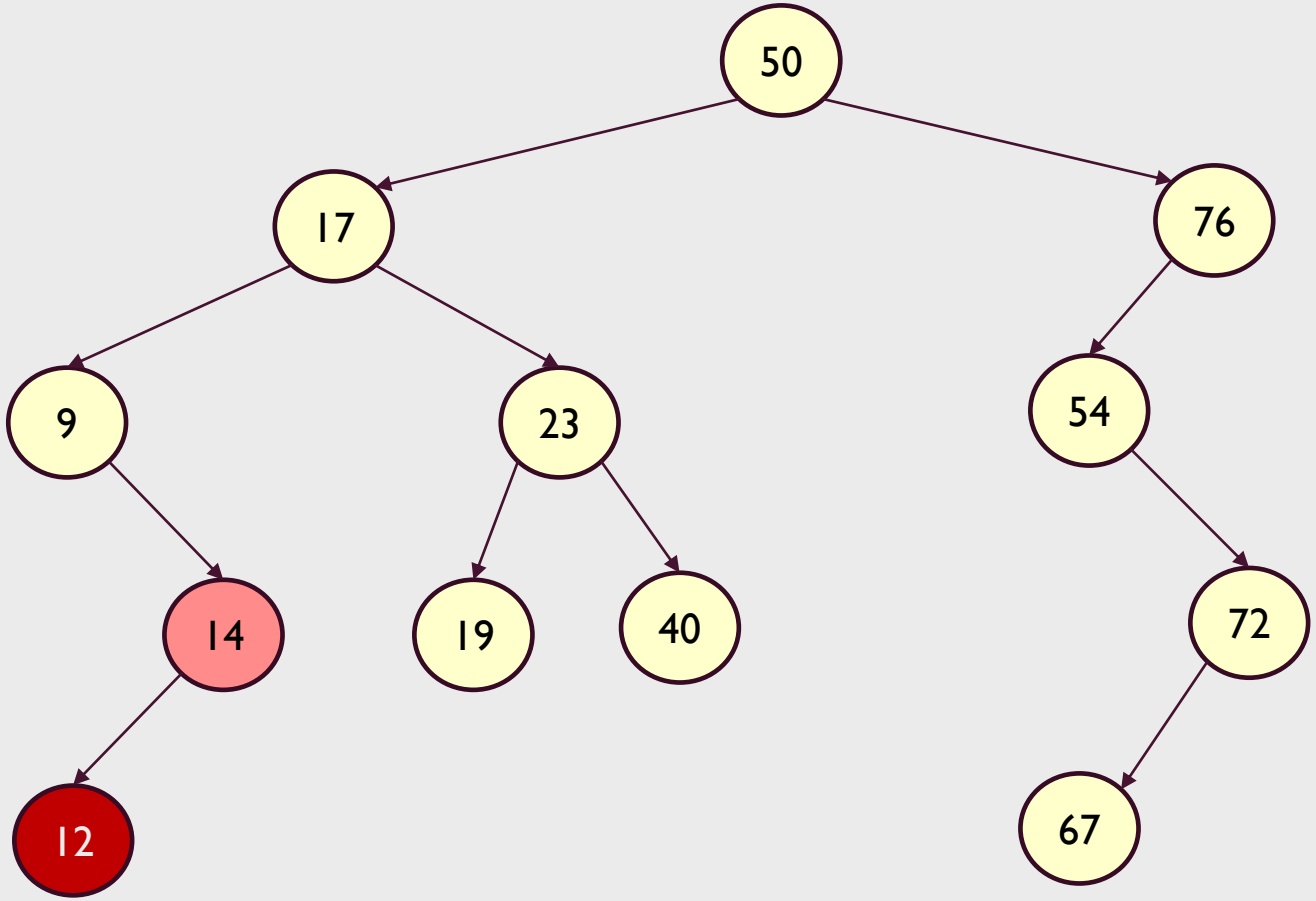



**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

## Traversal Example

9 12

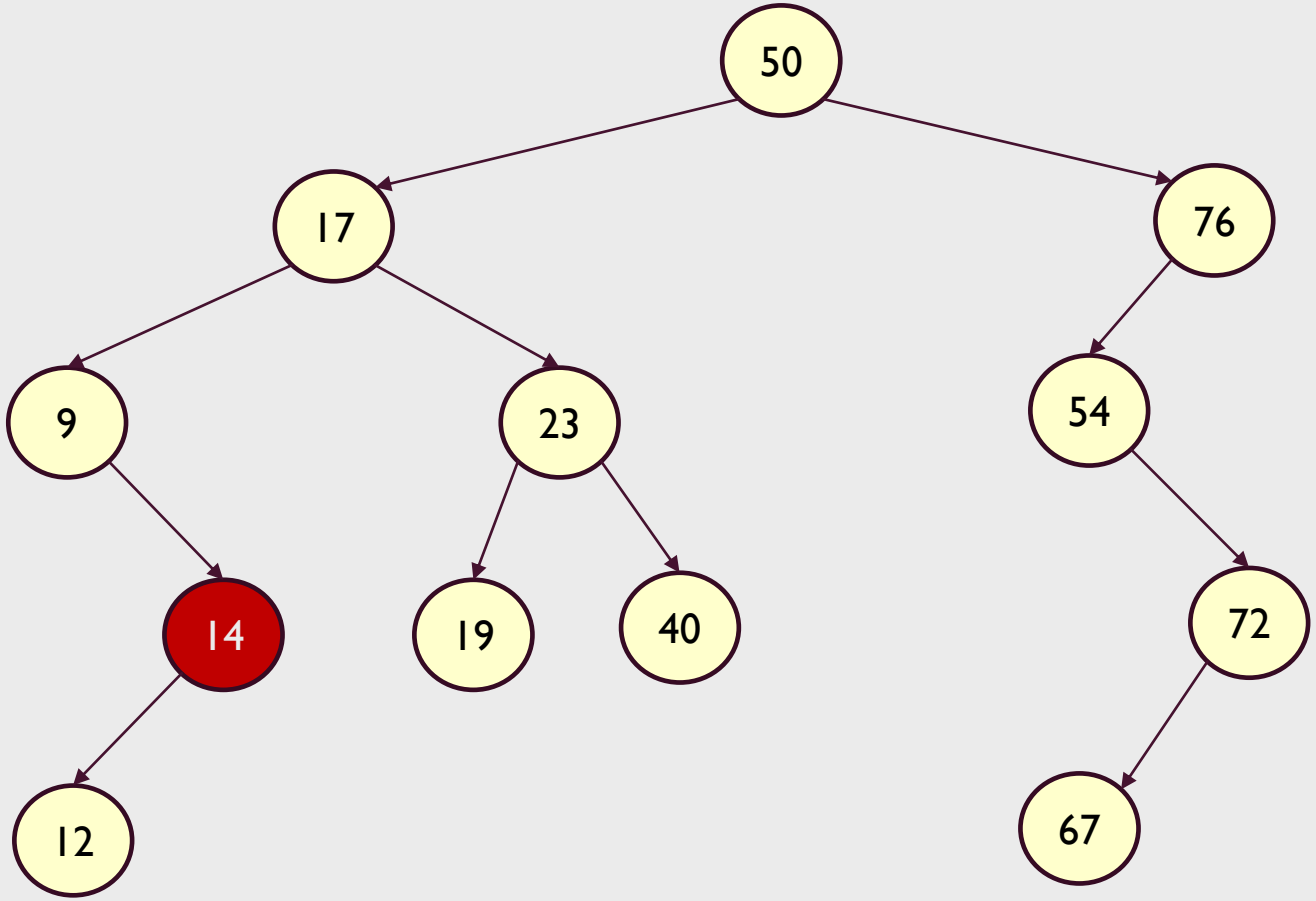



**We are not right child → found nx!**

# Innovations – BST Sorter

## Traversal Example

9 12 14

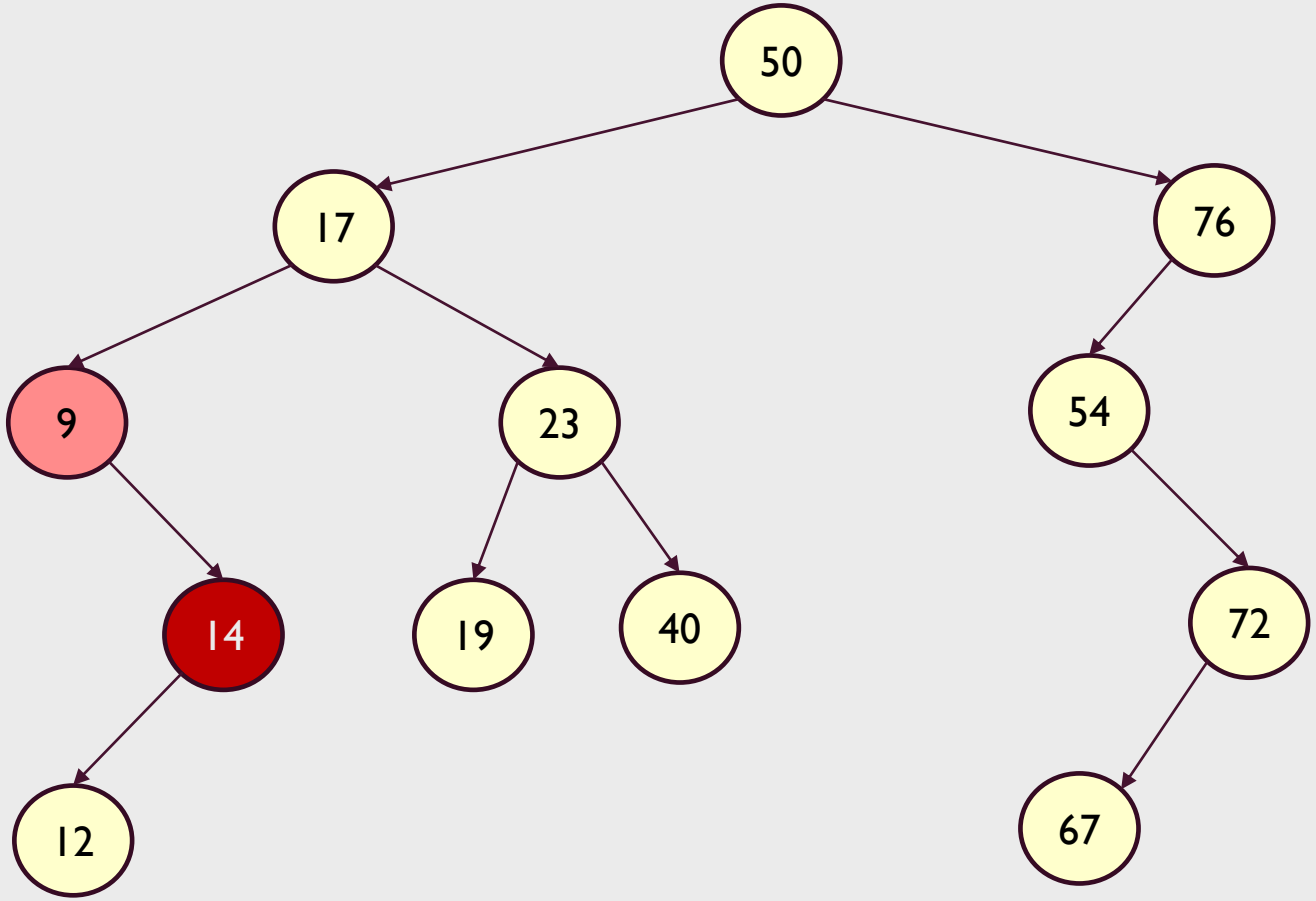



**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

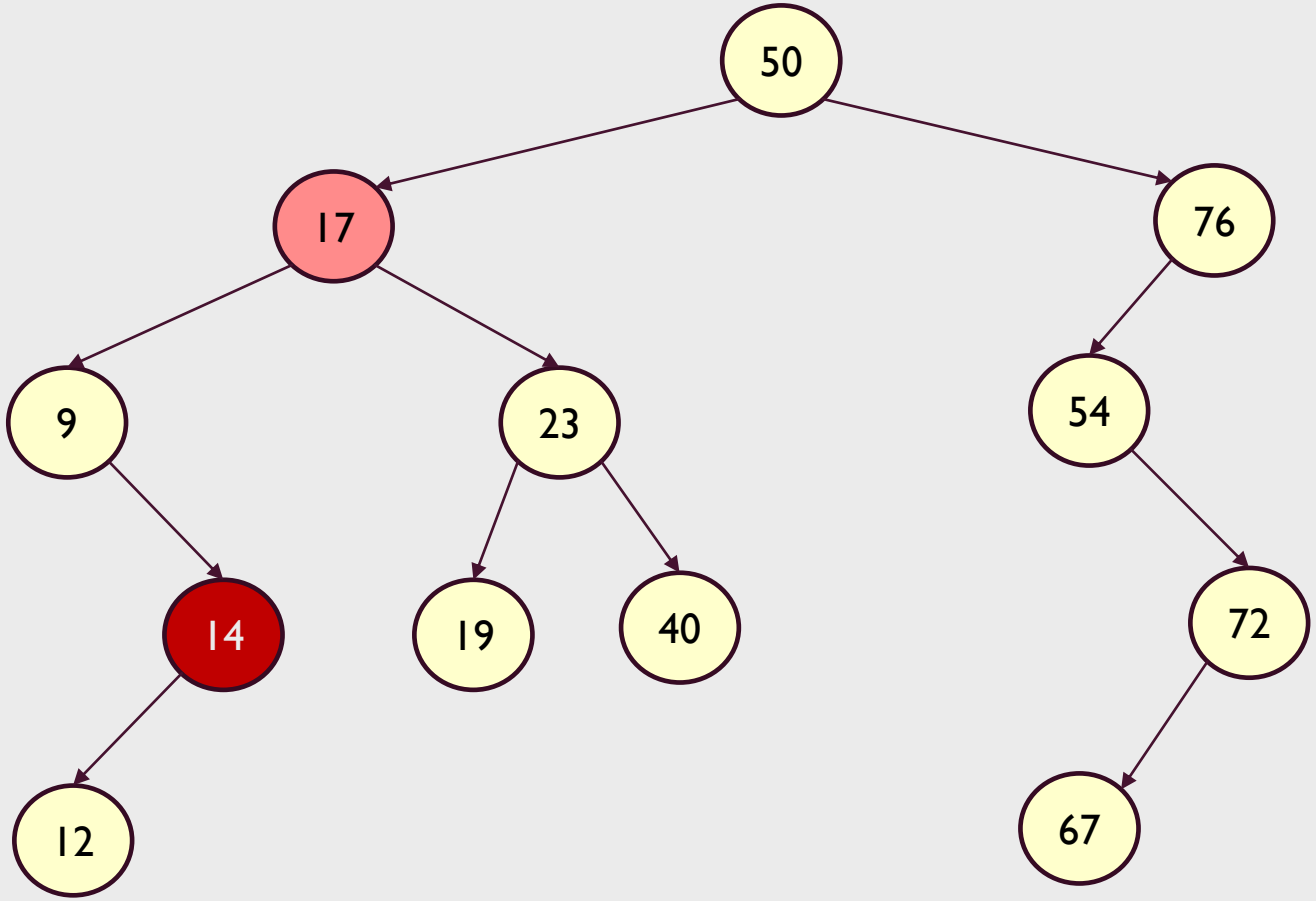
## Traversal Example


**Keep going up**



# Innovations – BST Sorter

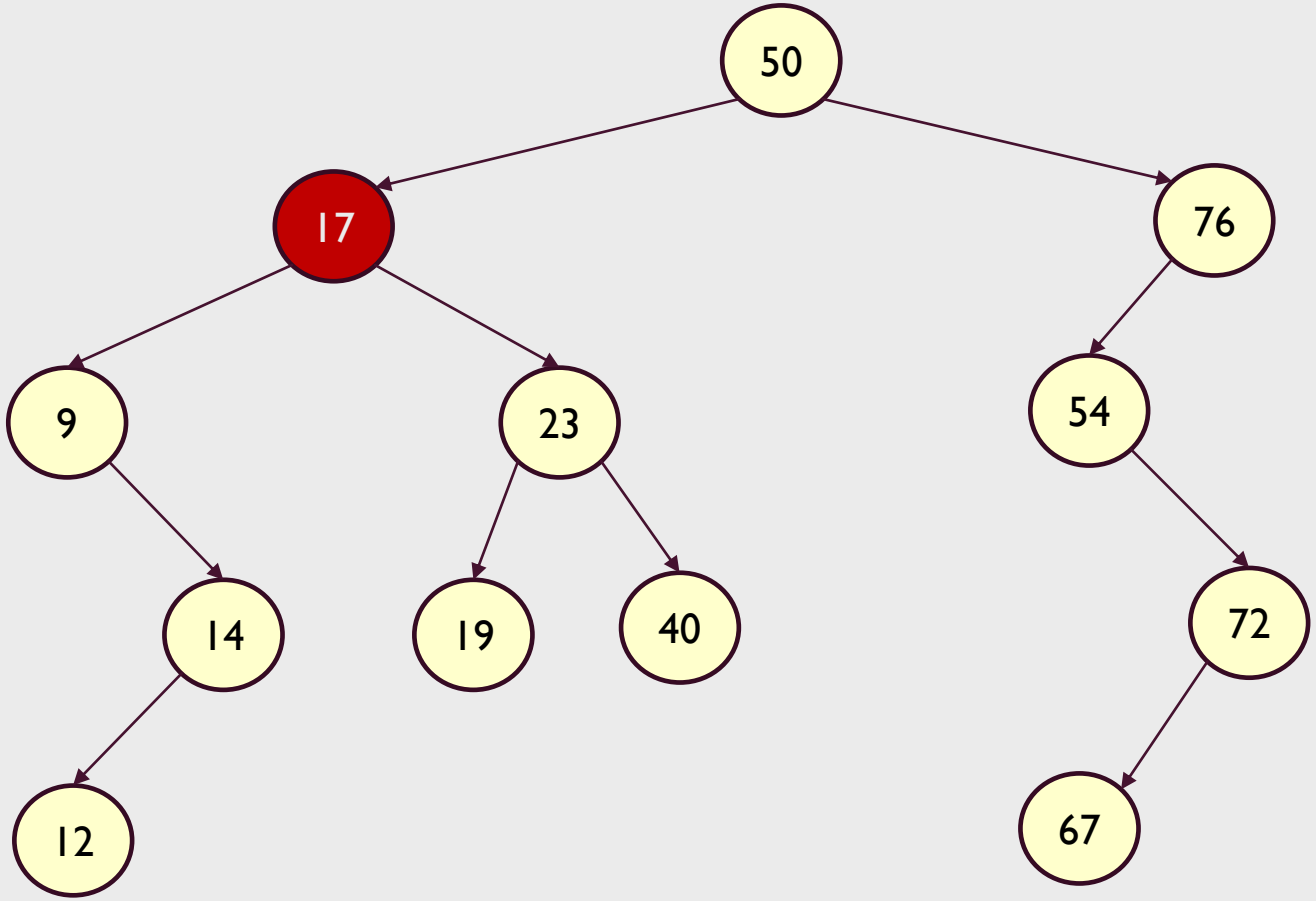
## Traversal Example


**We are not right child → found nx!**

# Innovations – BST Sorter

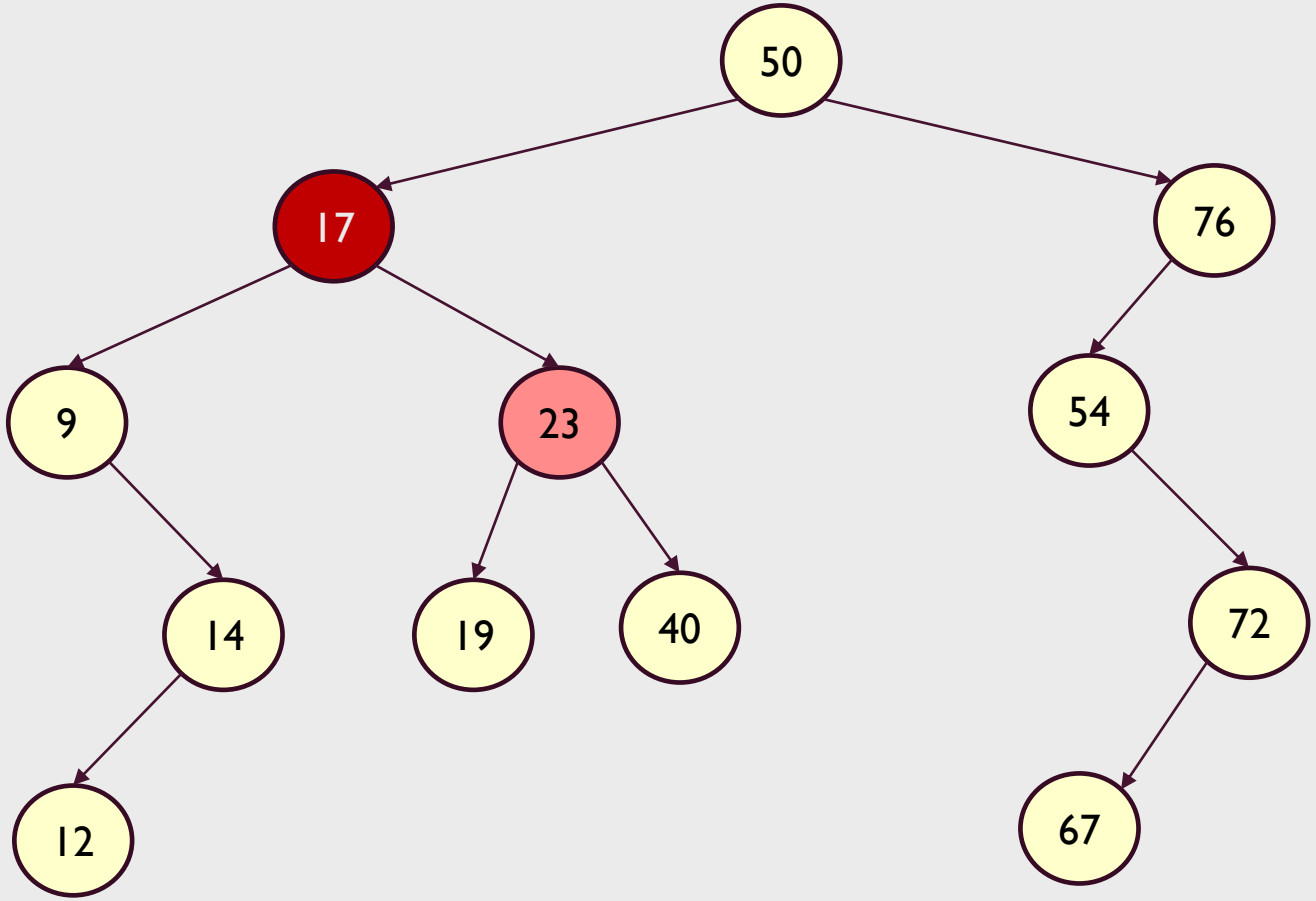
## Traversal Example


**Has right sub tree → find its min**

# Innovations – BST Sorter

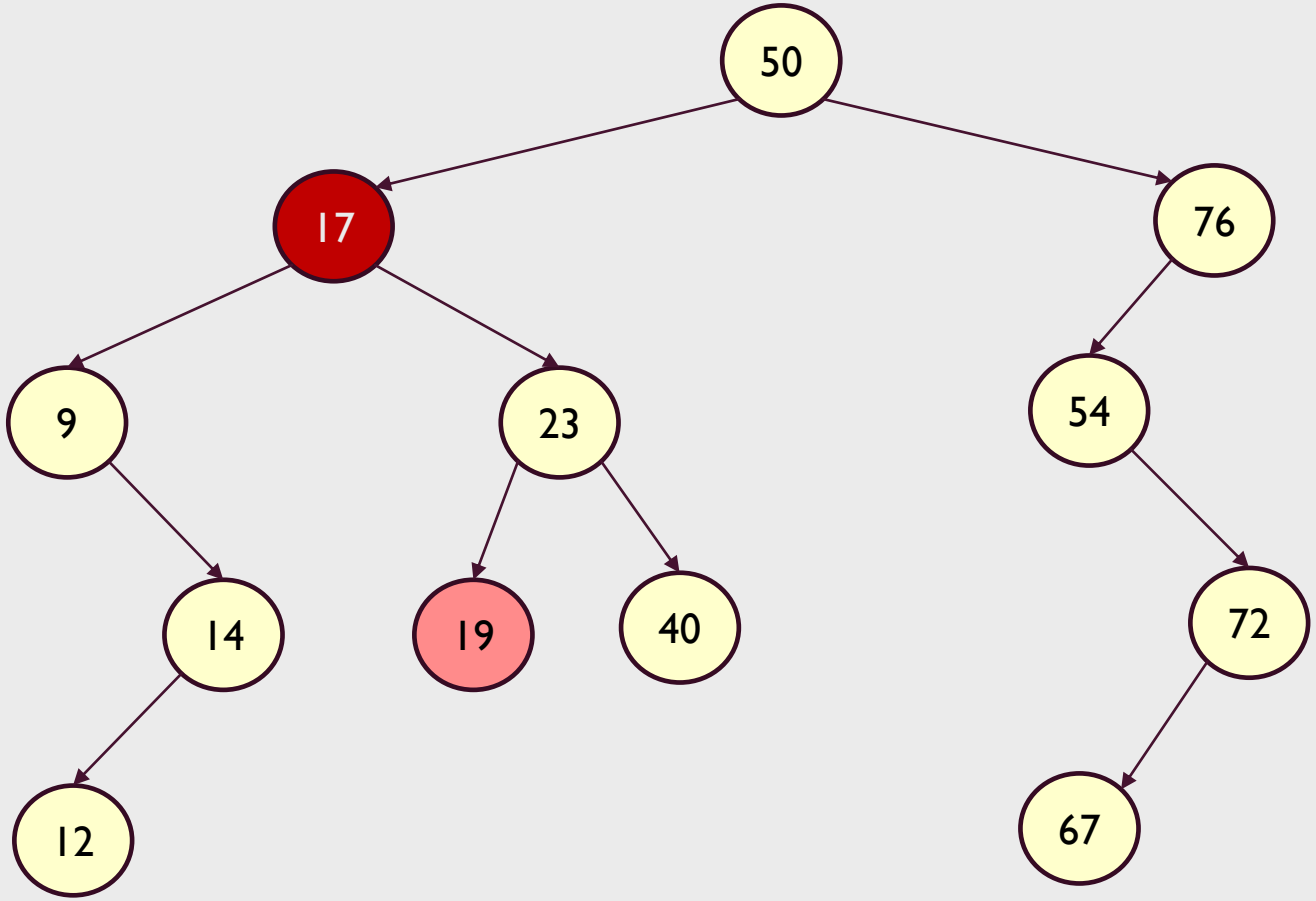
## Traversal Example


**Founding min: go down left.**

# Innovations – BST Sorter

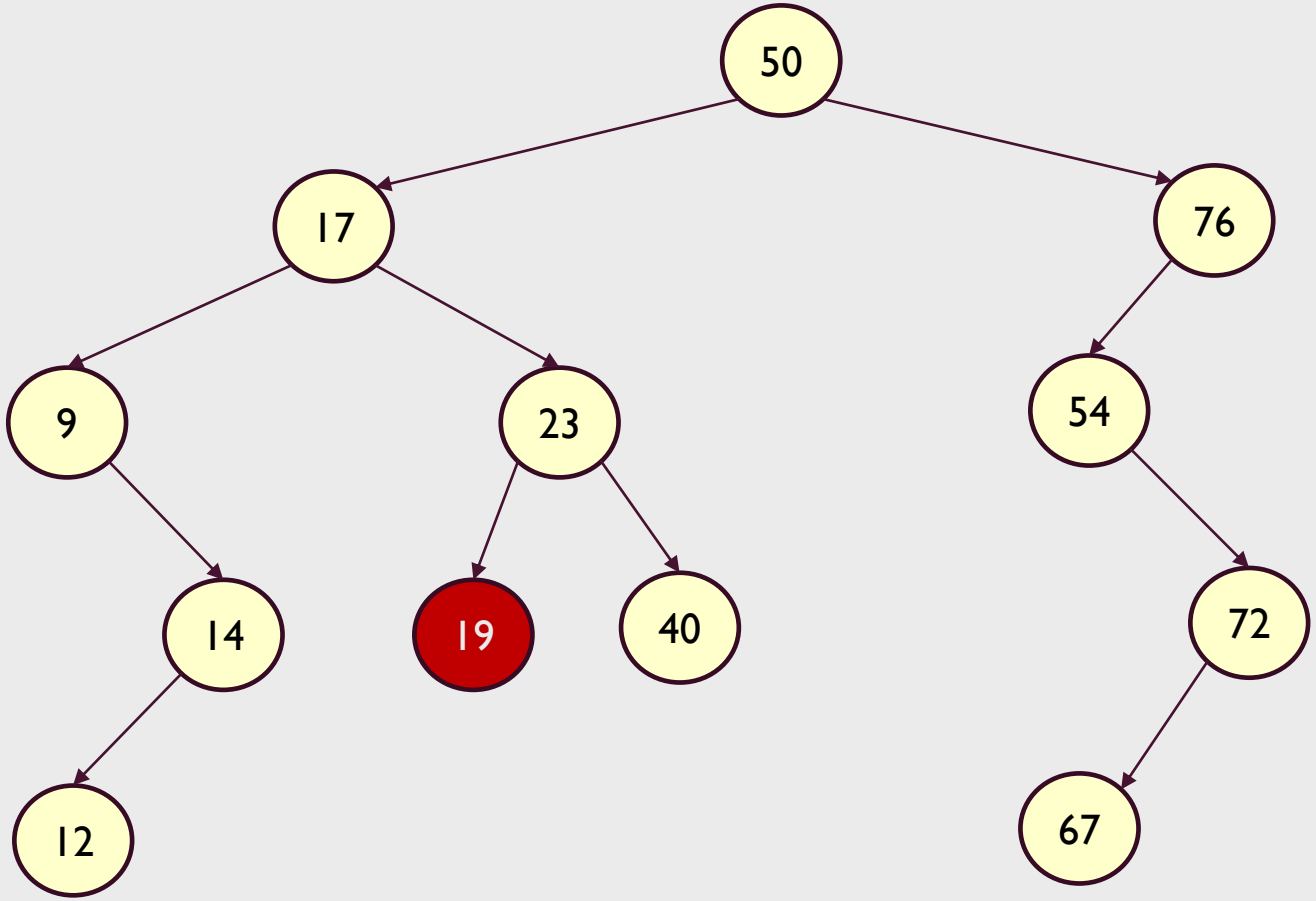
## Traversal Example


**Founding min: go down left.**
**no more left → min**
**found → found nx!**

# Innovations – BST Sorter

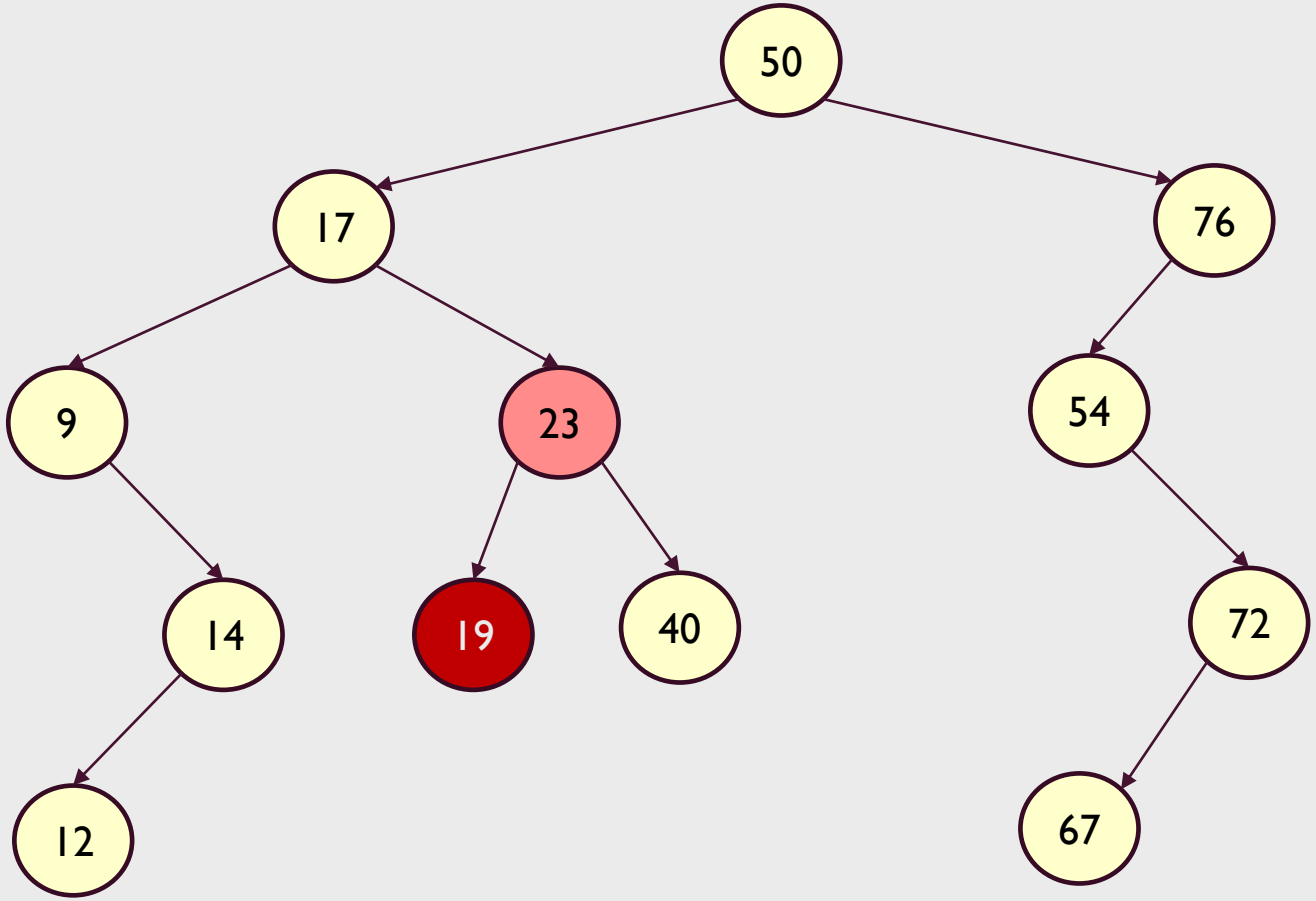
## Traversal Example


**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

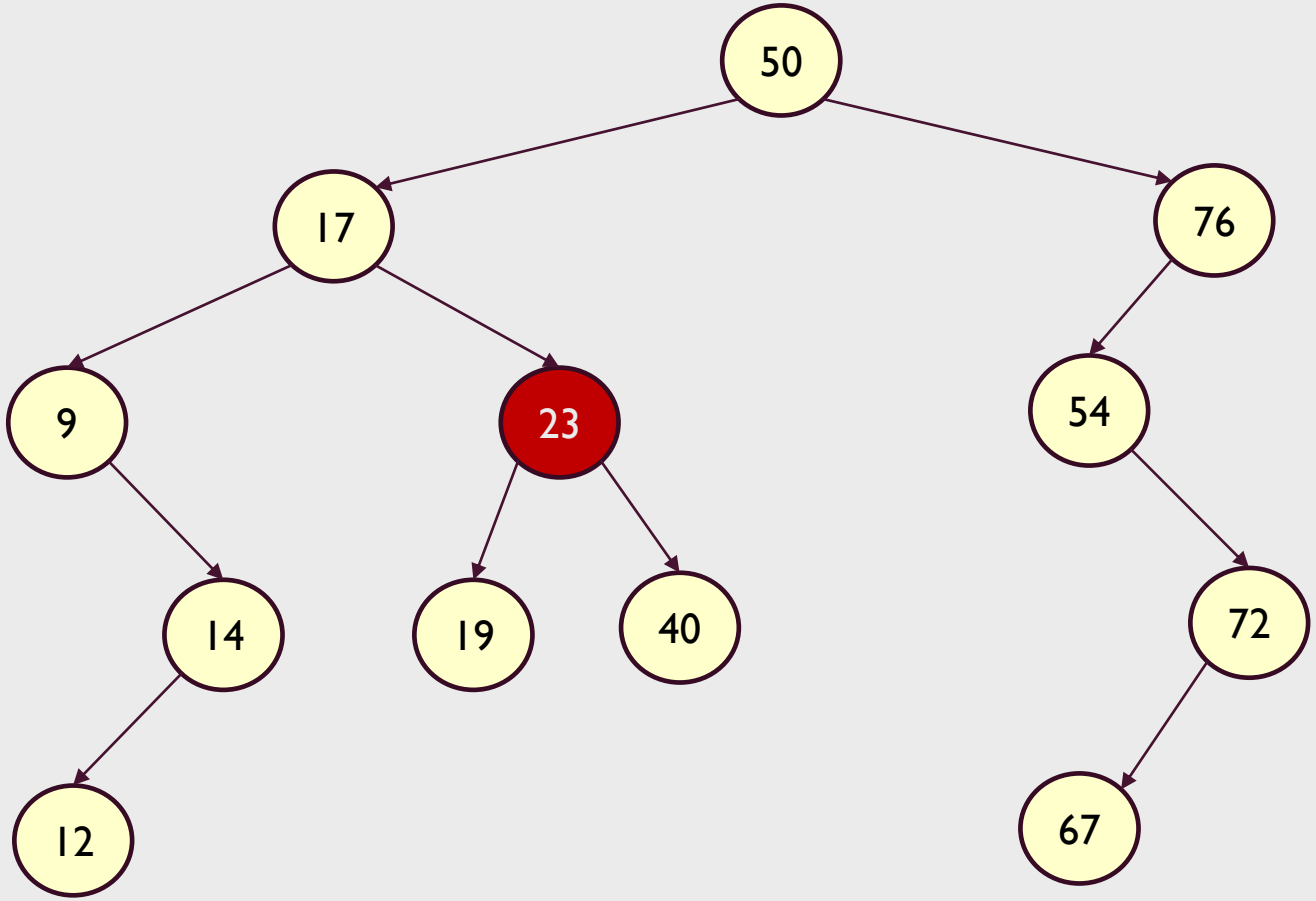
## Traversal Example


**We are not right child → found nx!**

# Innovations – BST Sorter

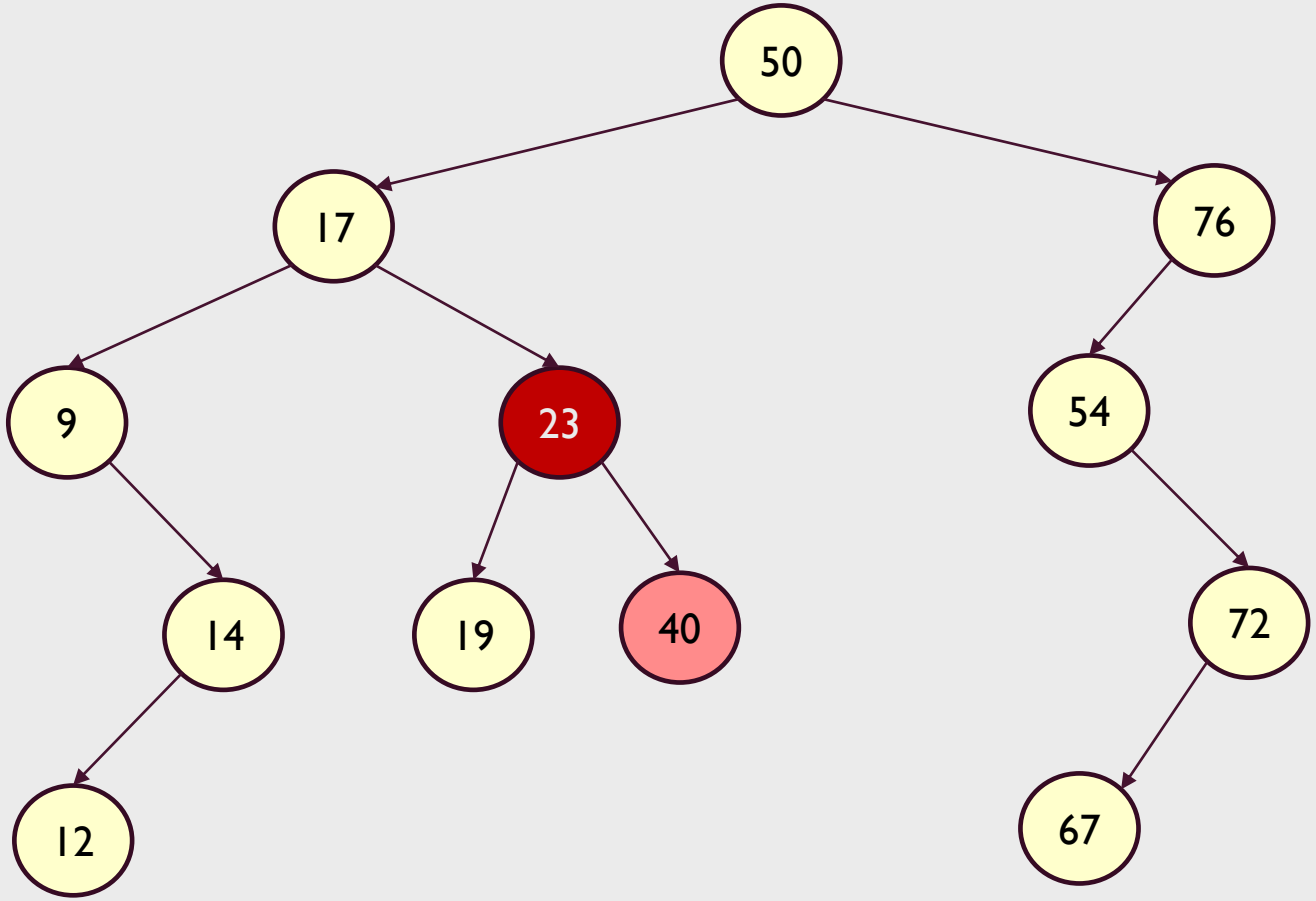
## Traversal Example


**Has right sub tree → find its min**

# Innovations – BST Sorter

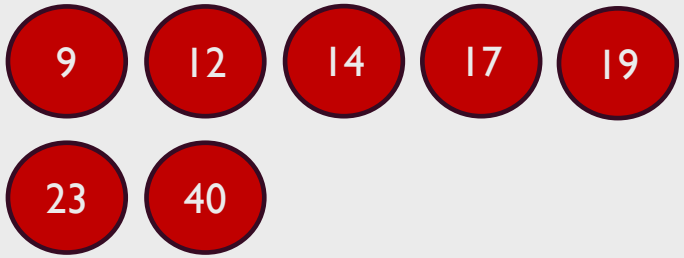
## Traversal Example

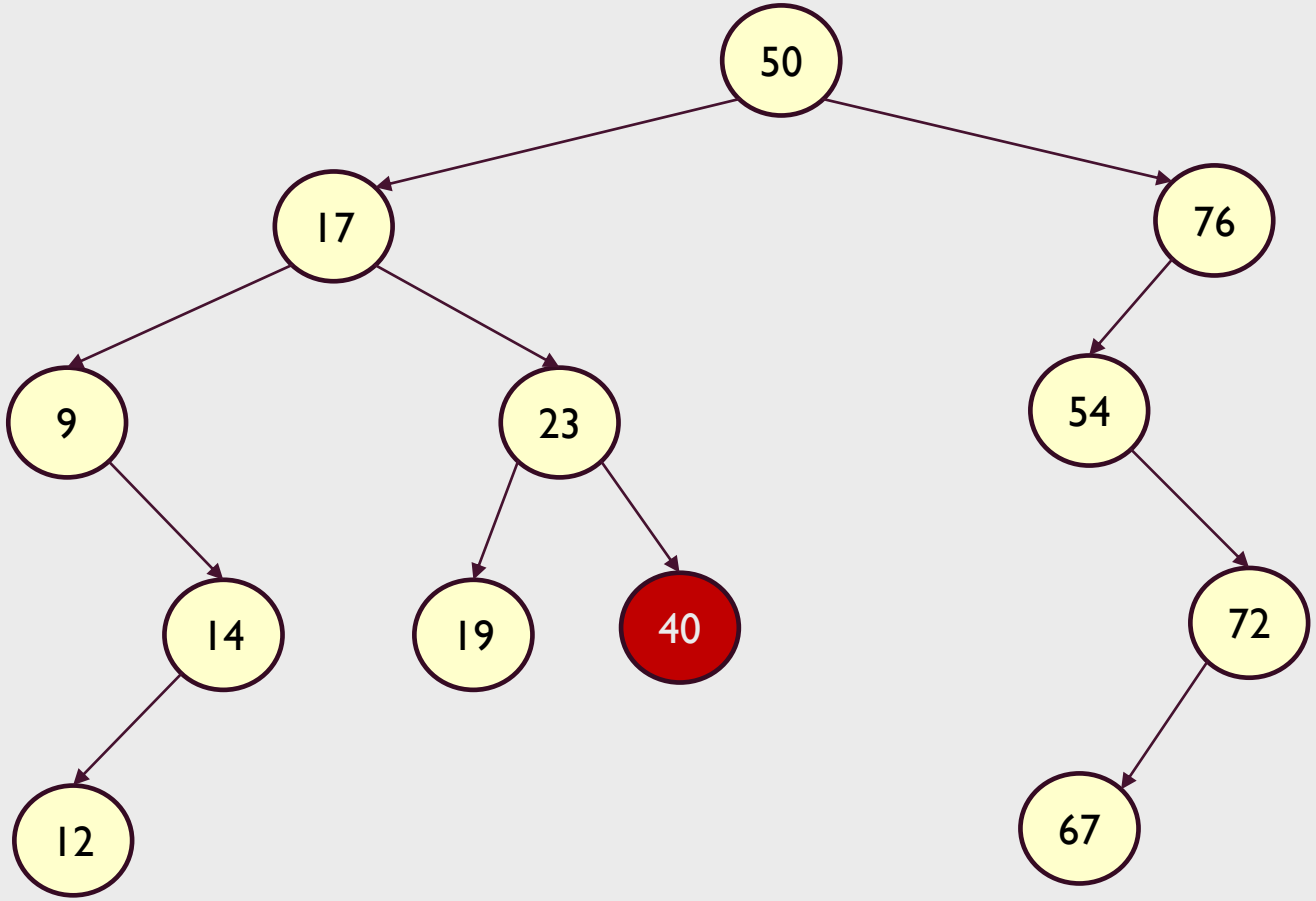




**Founding min: go down left.**
**no more left → min**
**found → found nx!**

# Innovations – BST Sorter

## Traversal Example

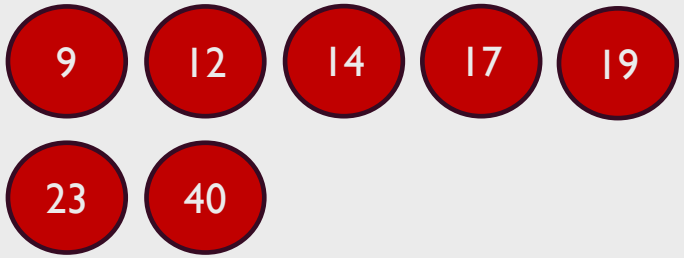


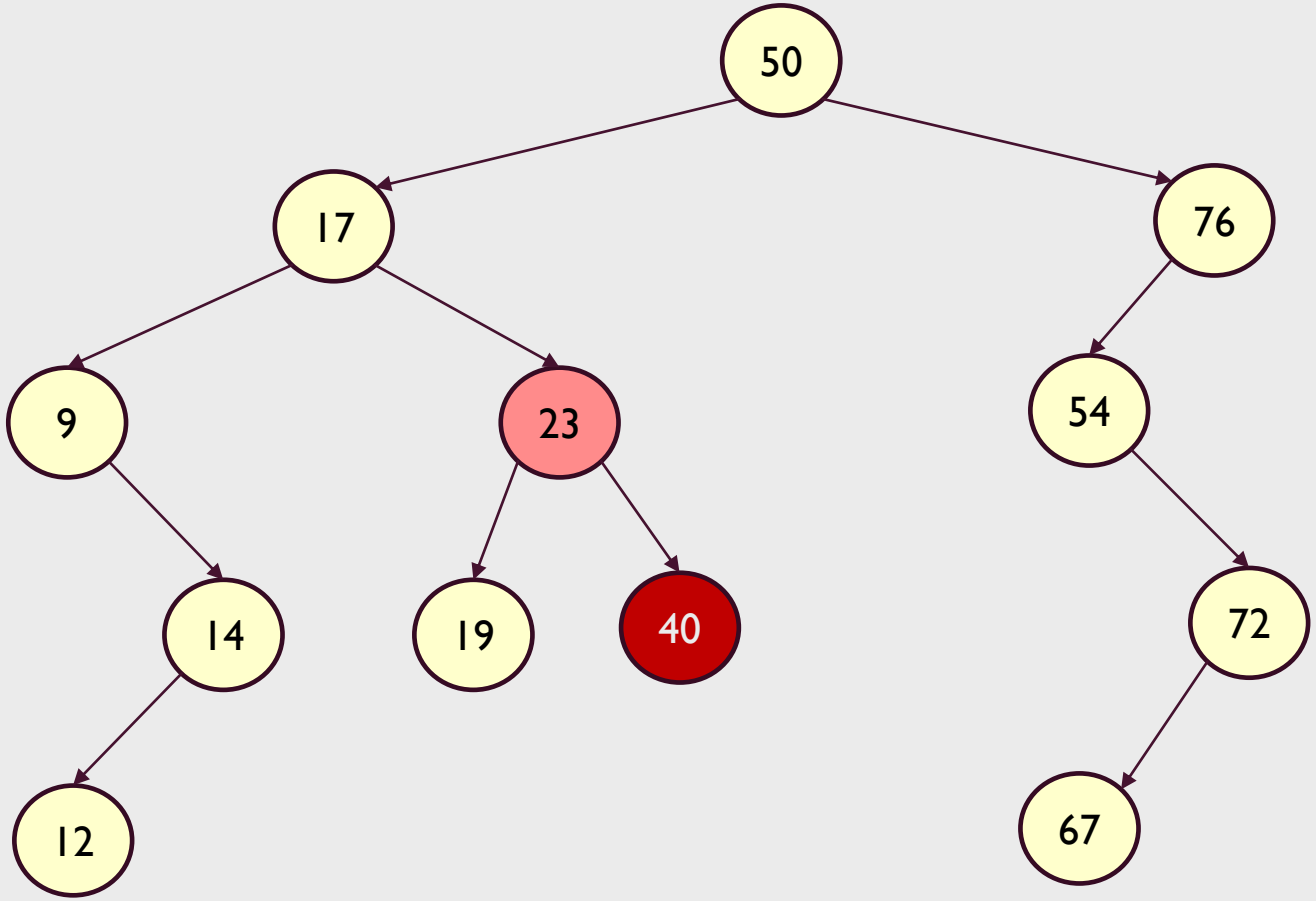




**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

## Traversal Example




**Keep going up**

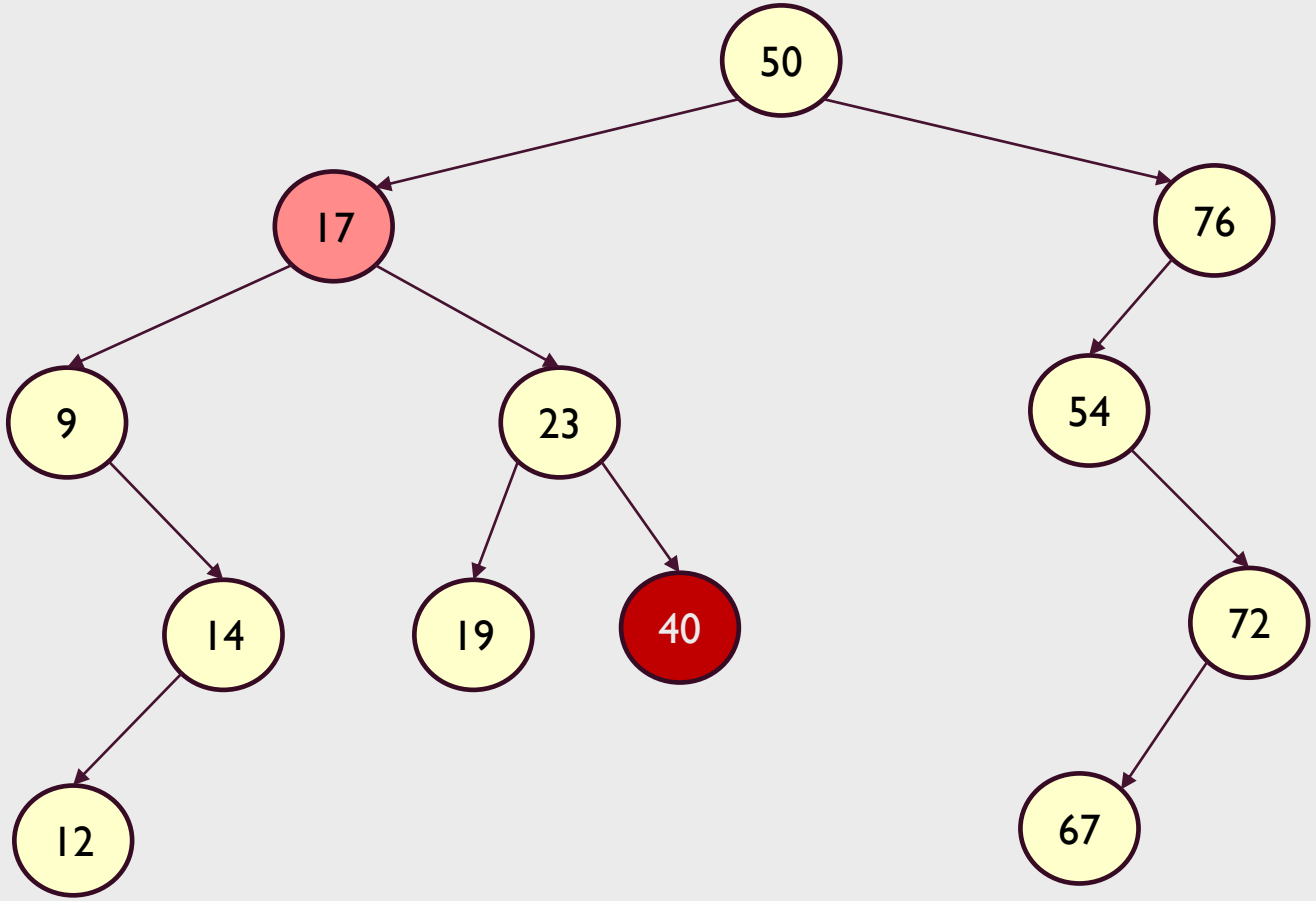
# Innovations – BST Sorter

## Traversal Example

9 12 14 17 19

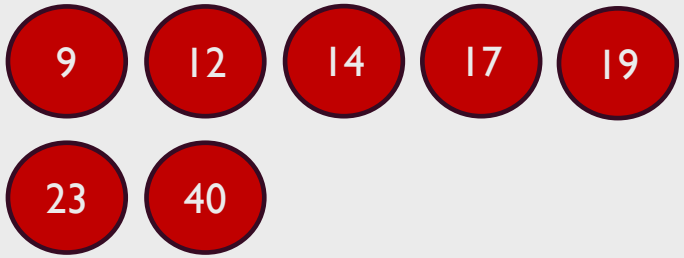
23 40

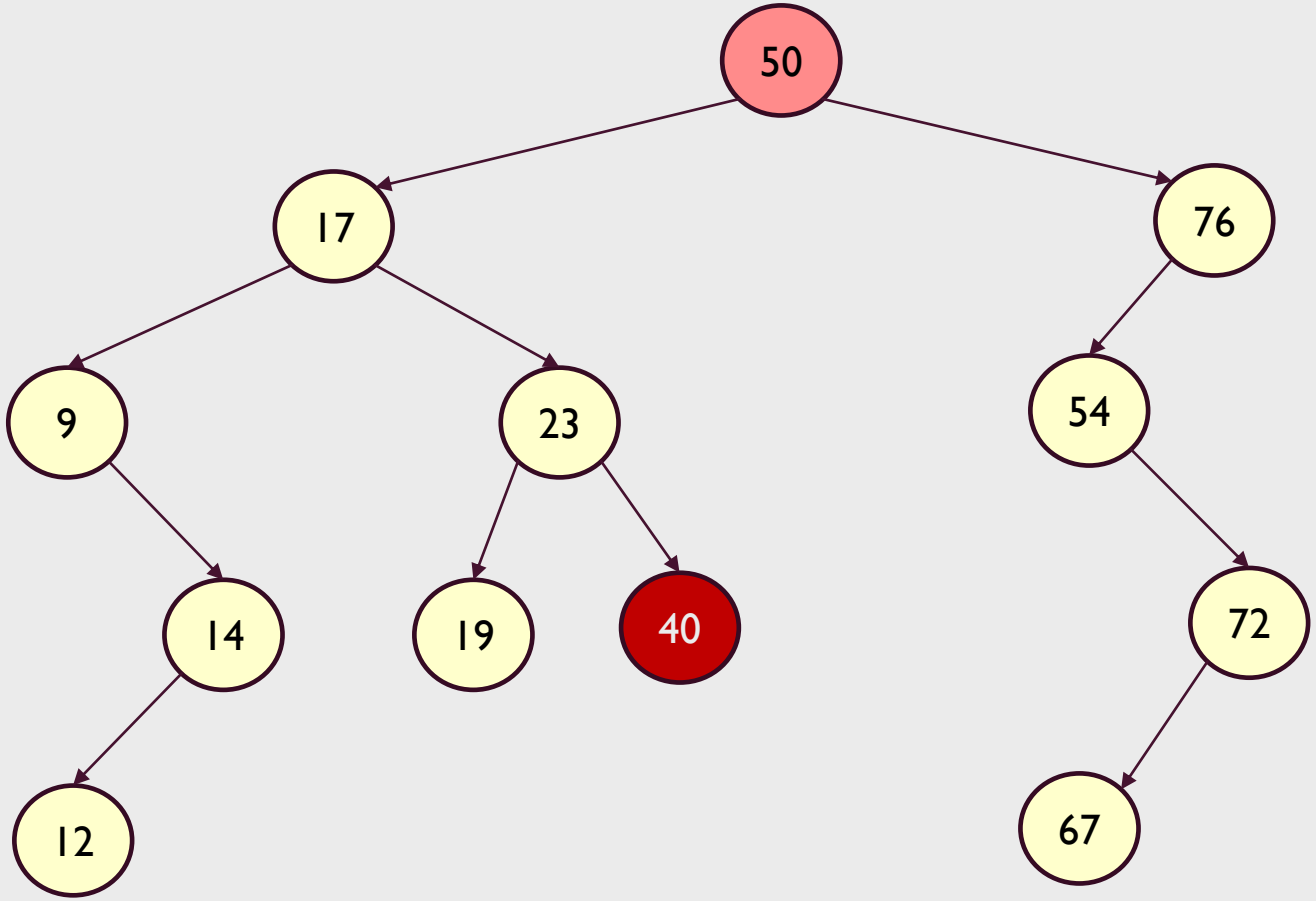
**Keep going up**



# Innovations – BST Sorter

## Traversal Example

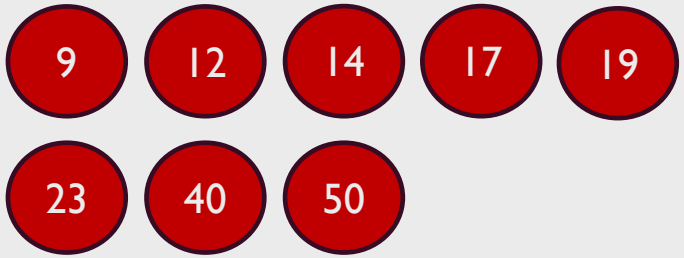


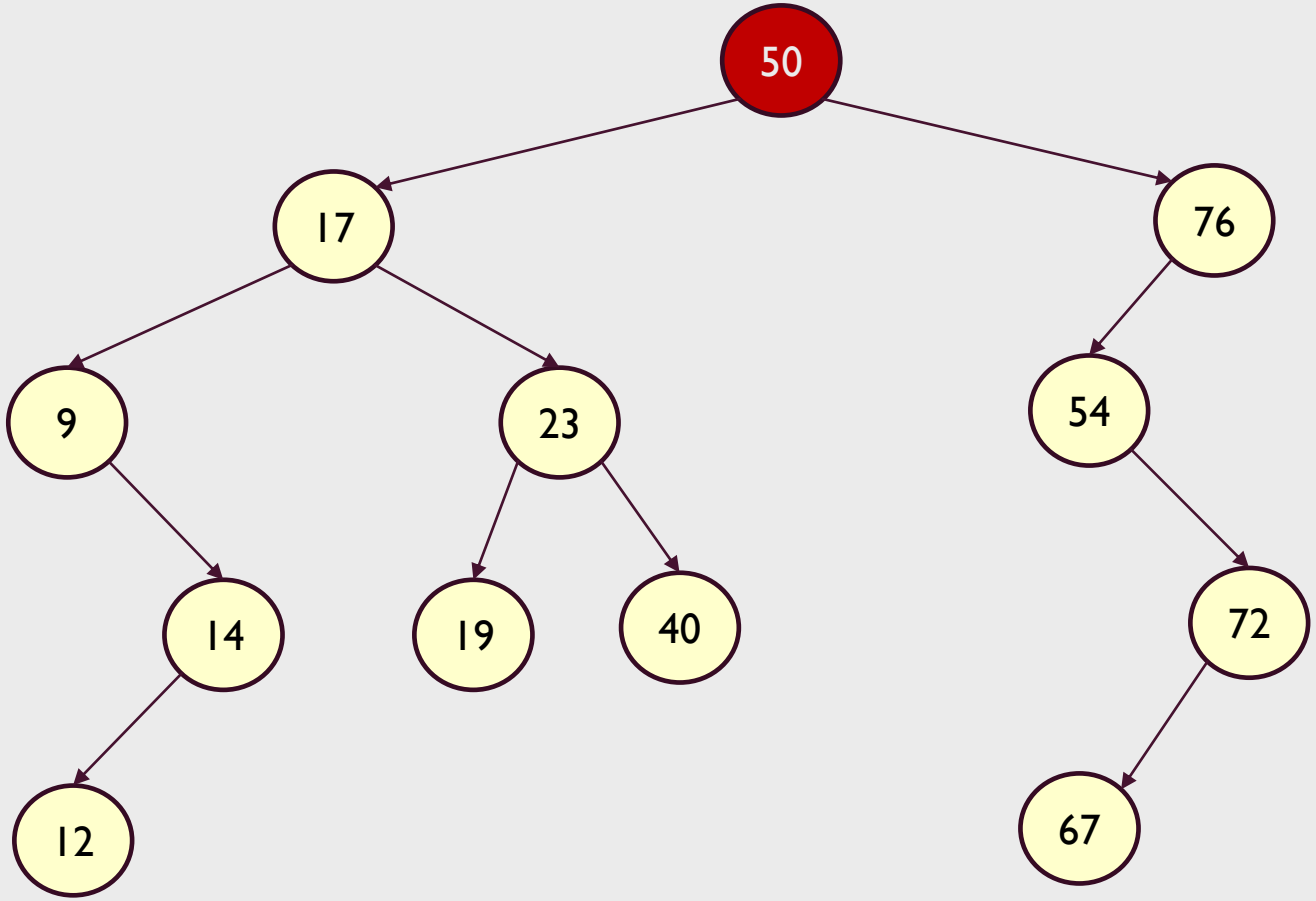




**We are not right child → found nx!**

# Innovations – BST Sorter

## Traversal Example

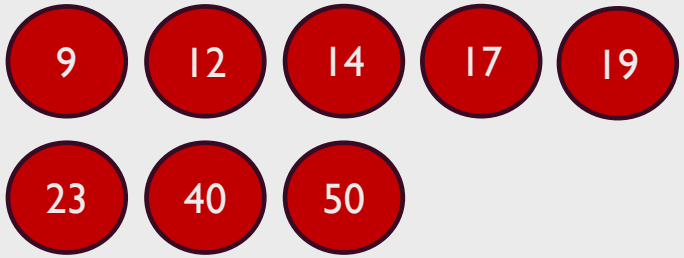


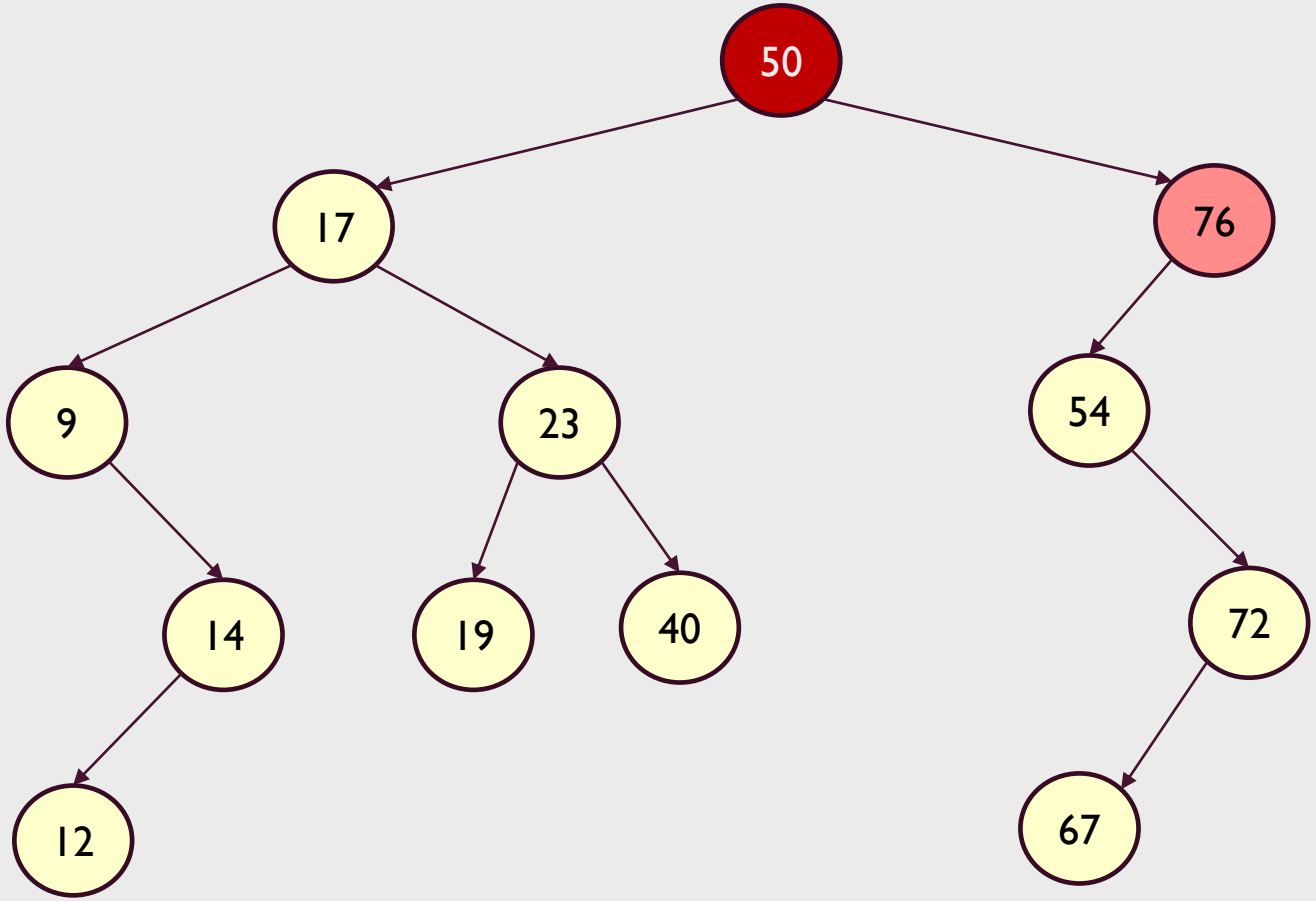




**Has right sub tree → find its min**

# Innovations – BST Sorter

## Traversal Example

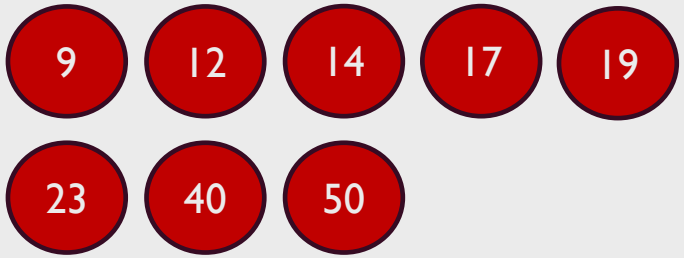


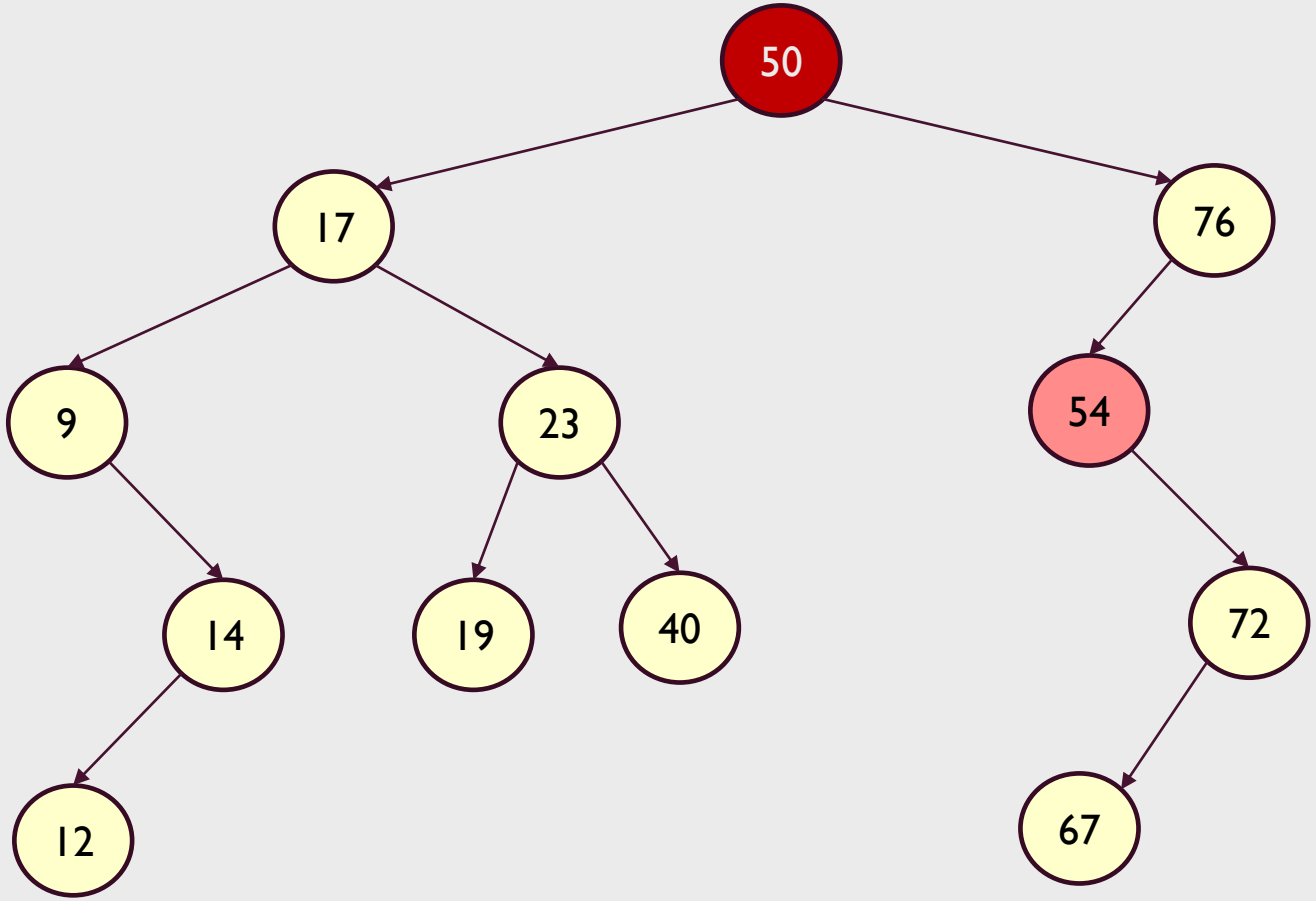




**Founding min: go down left.**

# Innovations – BST Sorter

## Traversal Example

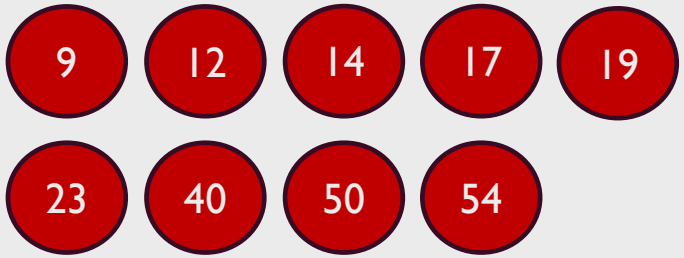


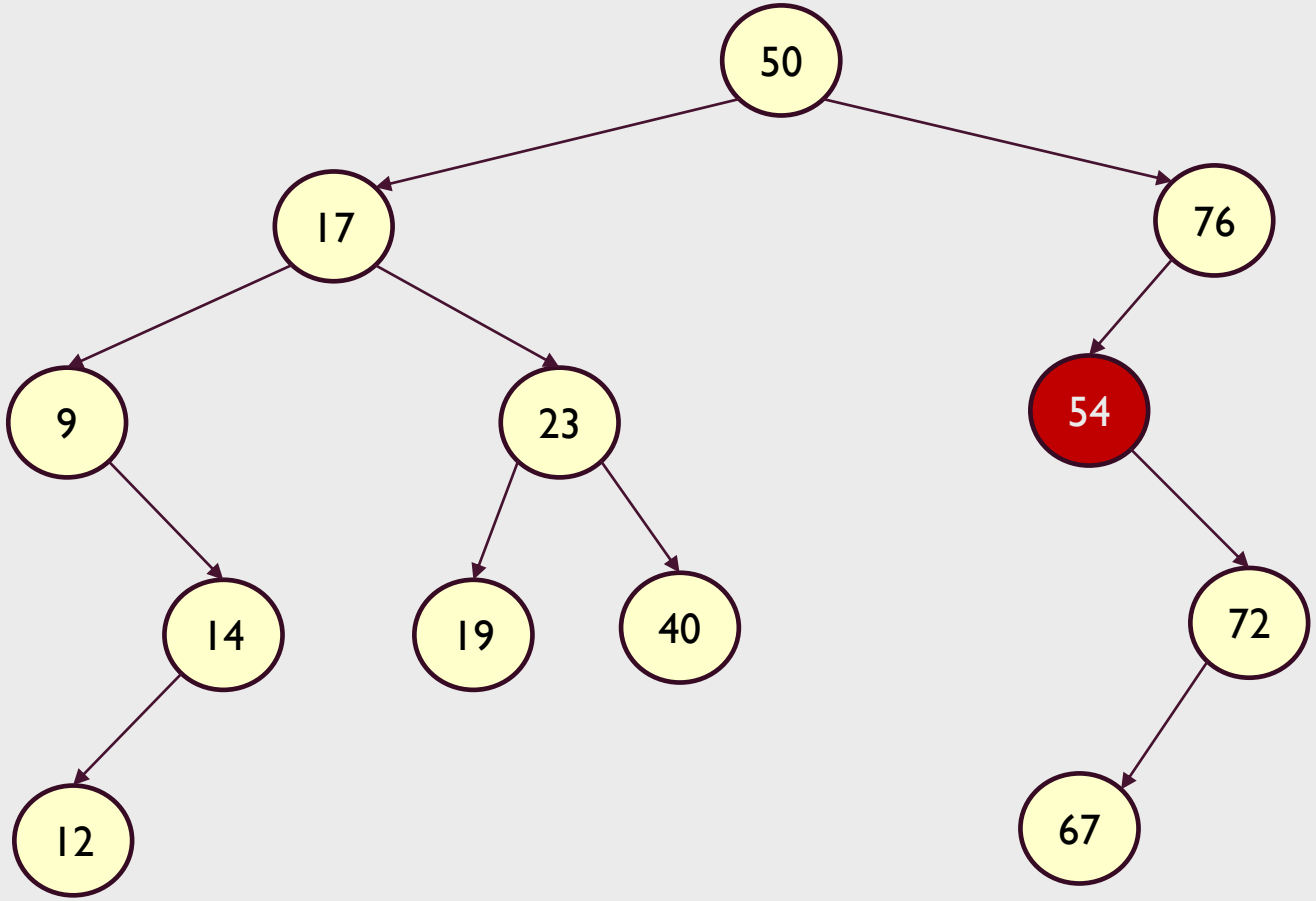




**Founding min: go down left.**
**no more left → min**
**found → found nx!**

# Innovations – BST Sorter

## Traversal Example

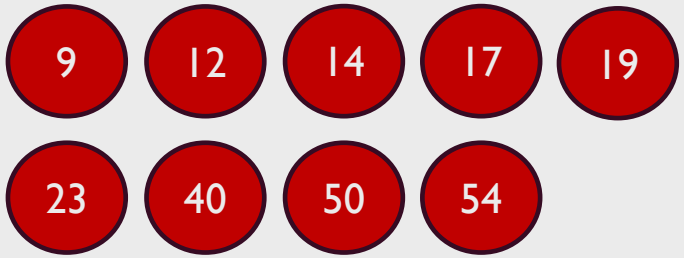


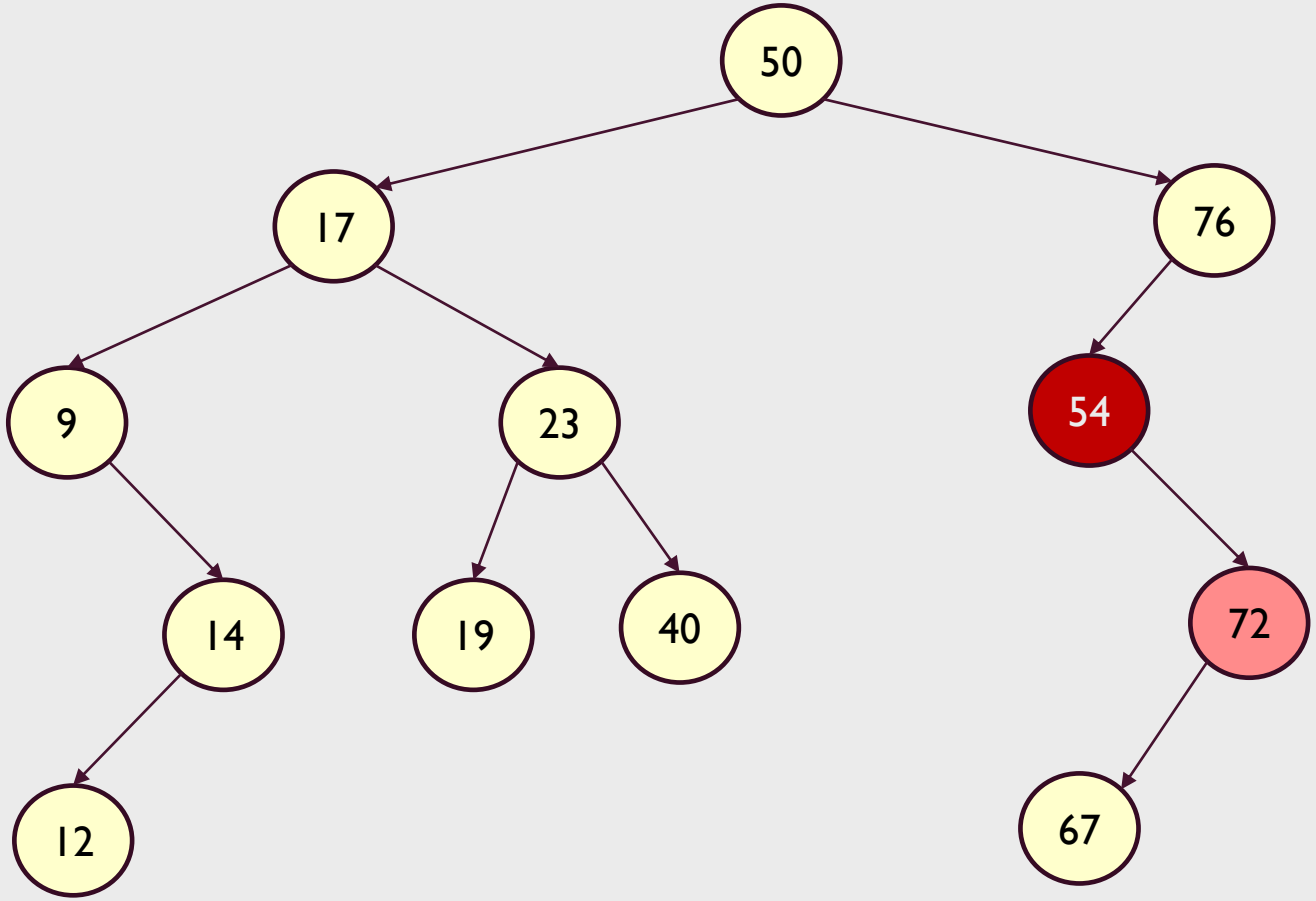




**Has right sub tree → find its min**

# Innovations – BST Sorter

## Traversal Example

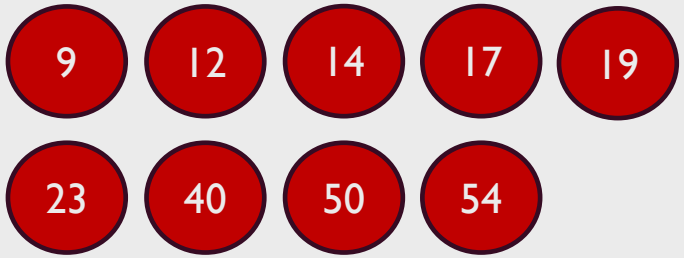


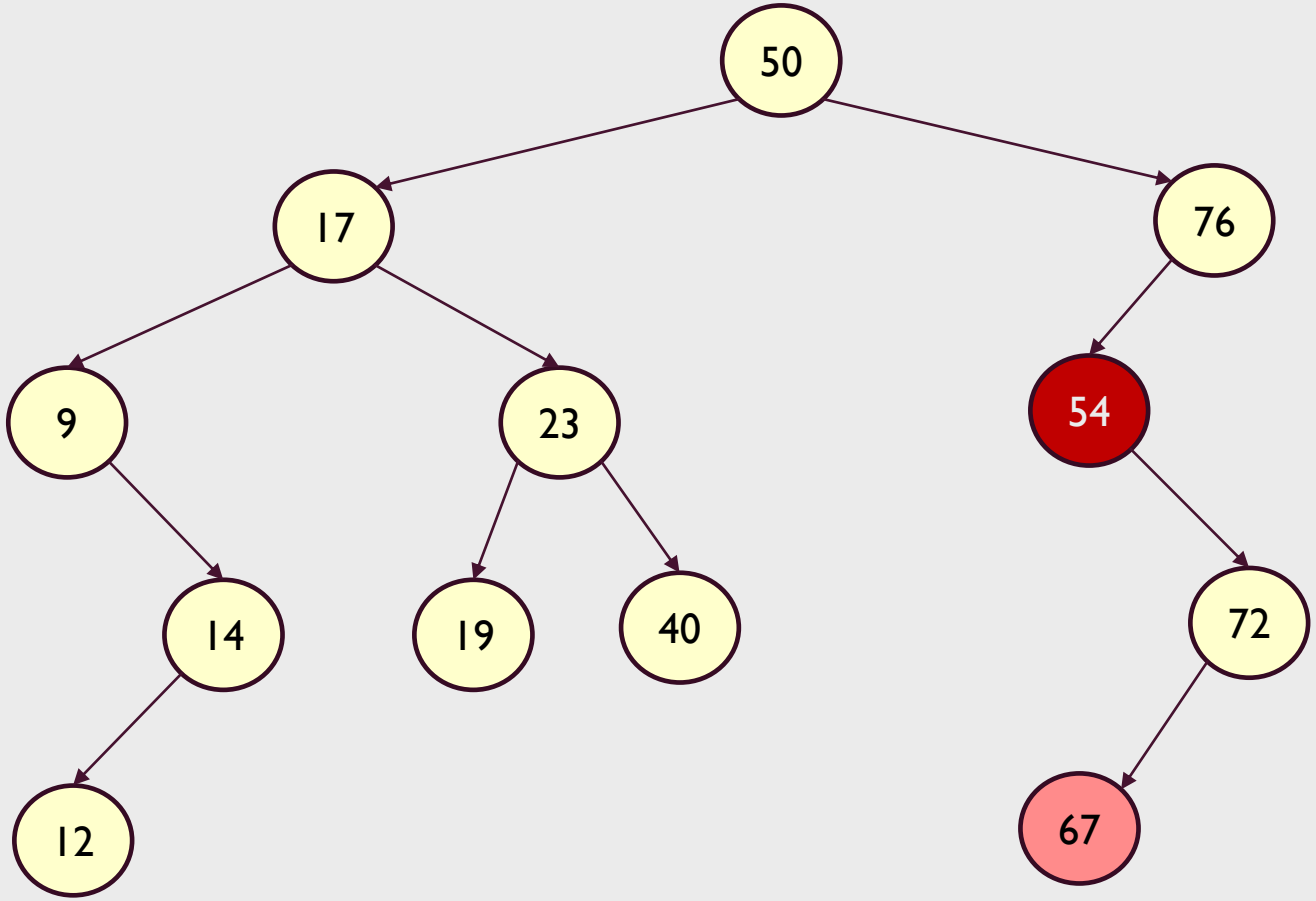




**Founding min: go down left.**

# Innovations – BST Sorter

## Traversal Example

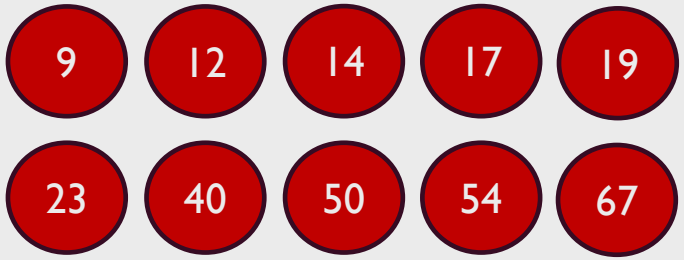


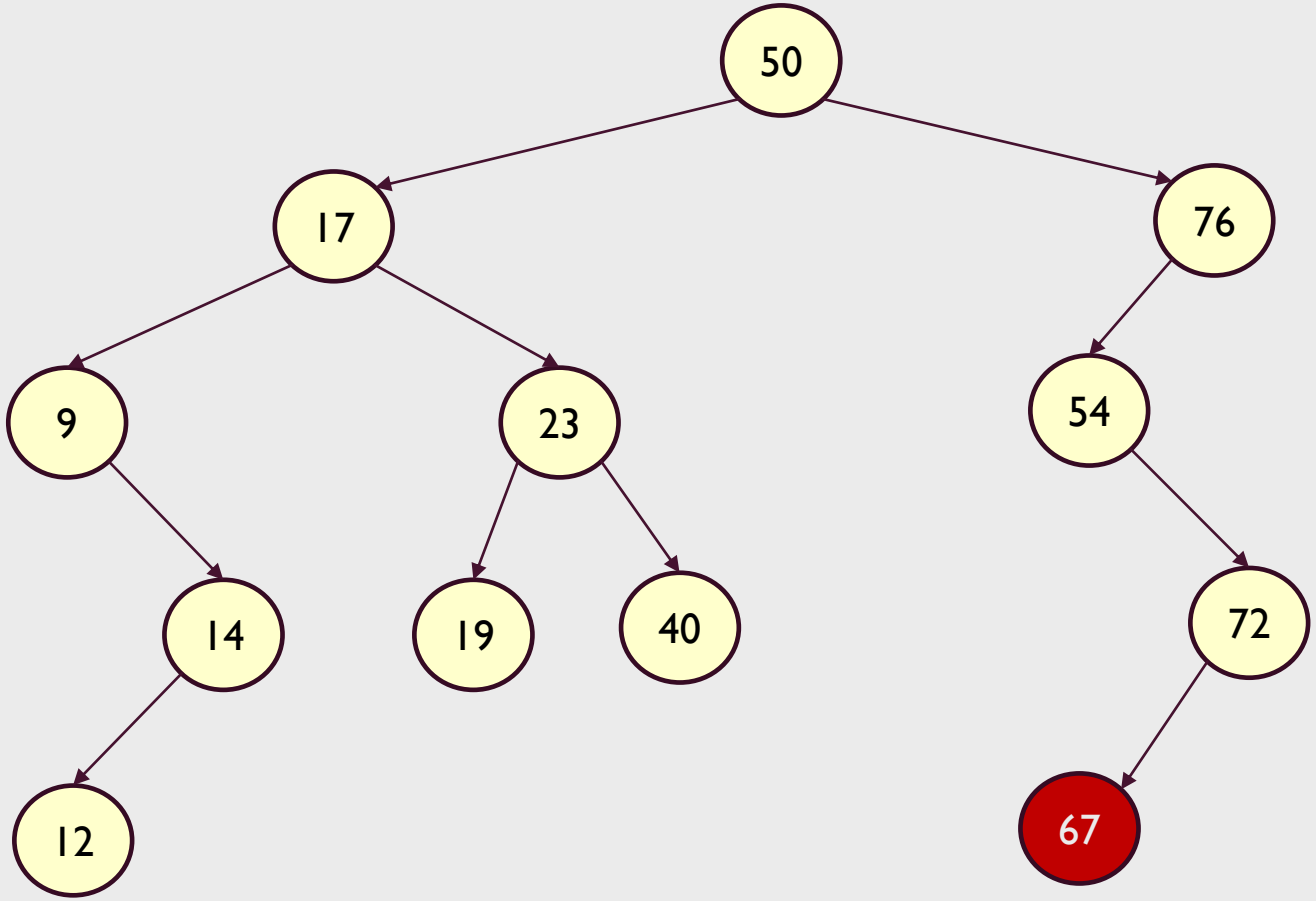




**Founding min: go down left.**
**no more left → min**
**found → found nx!**

# Innovations – BST Sorter

## Traversal Example

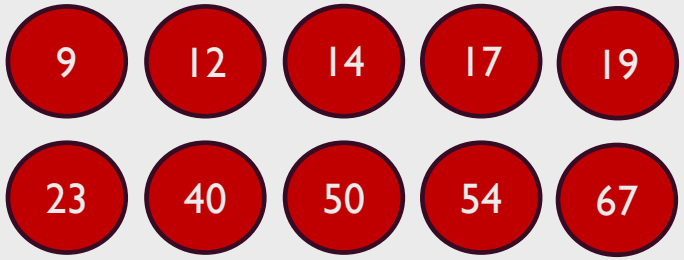


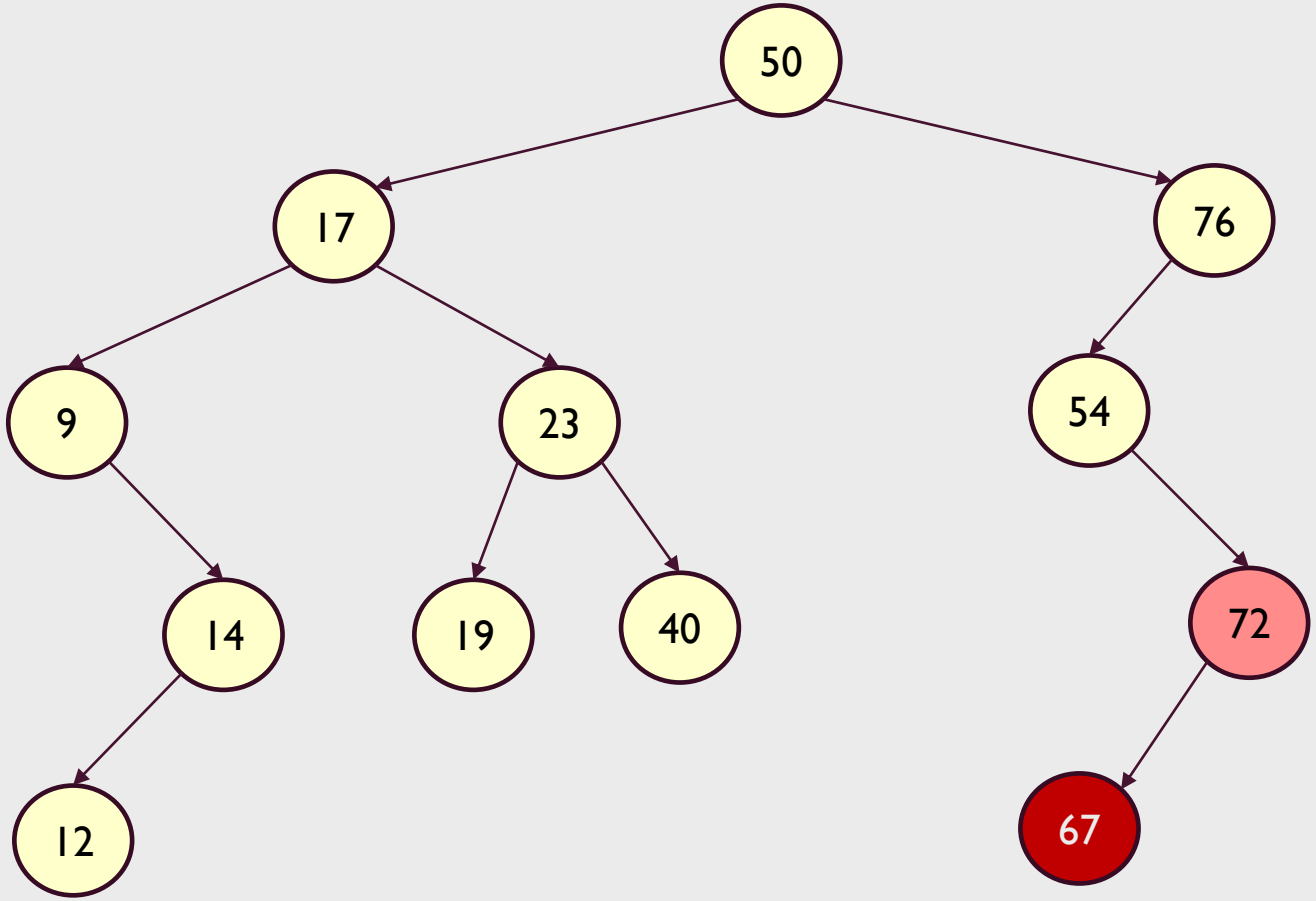




**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

## Traversal Example

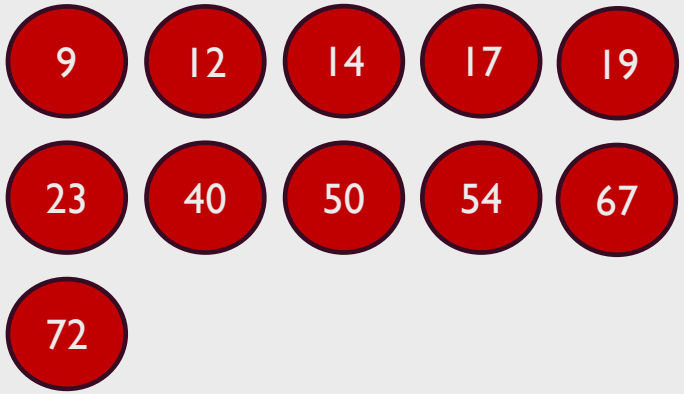


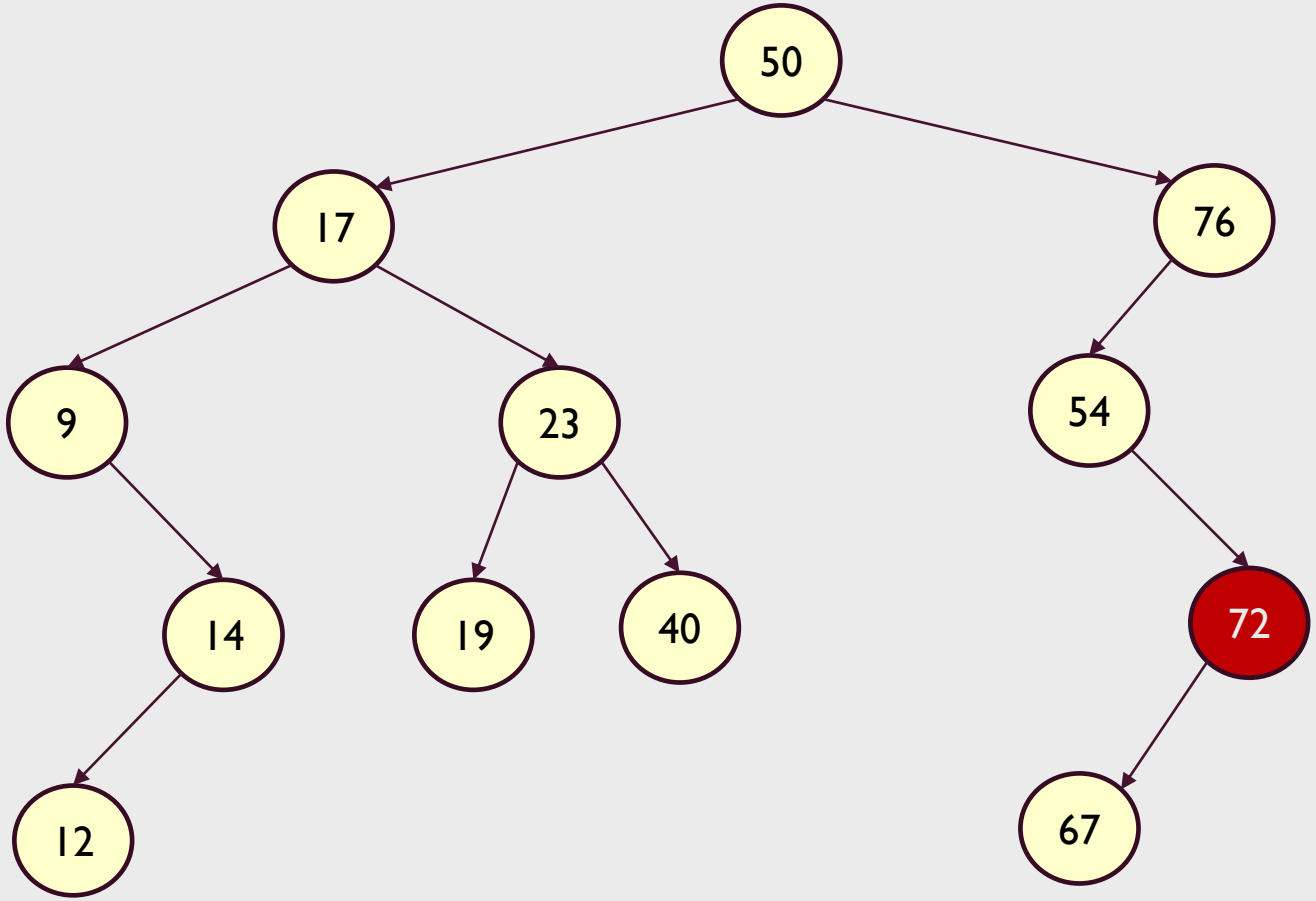




**We are not right child → found nx!**

# Innovations – BST Sorter

## Traversal Example

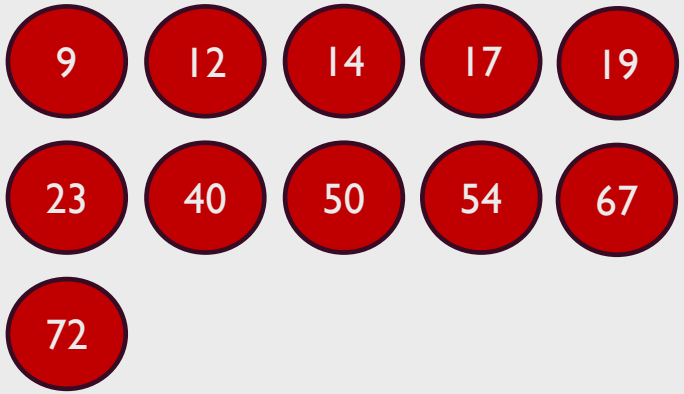


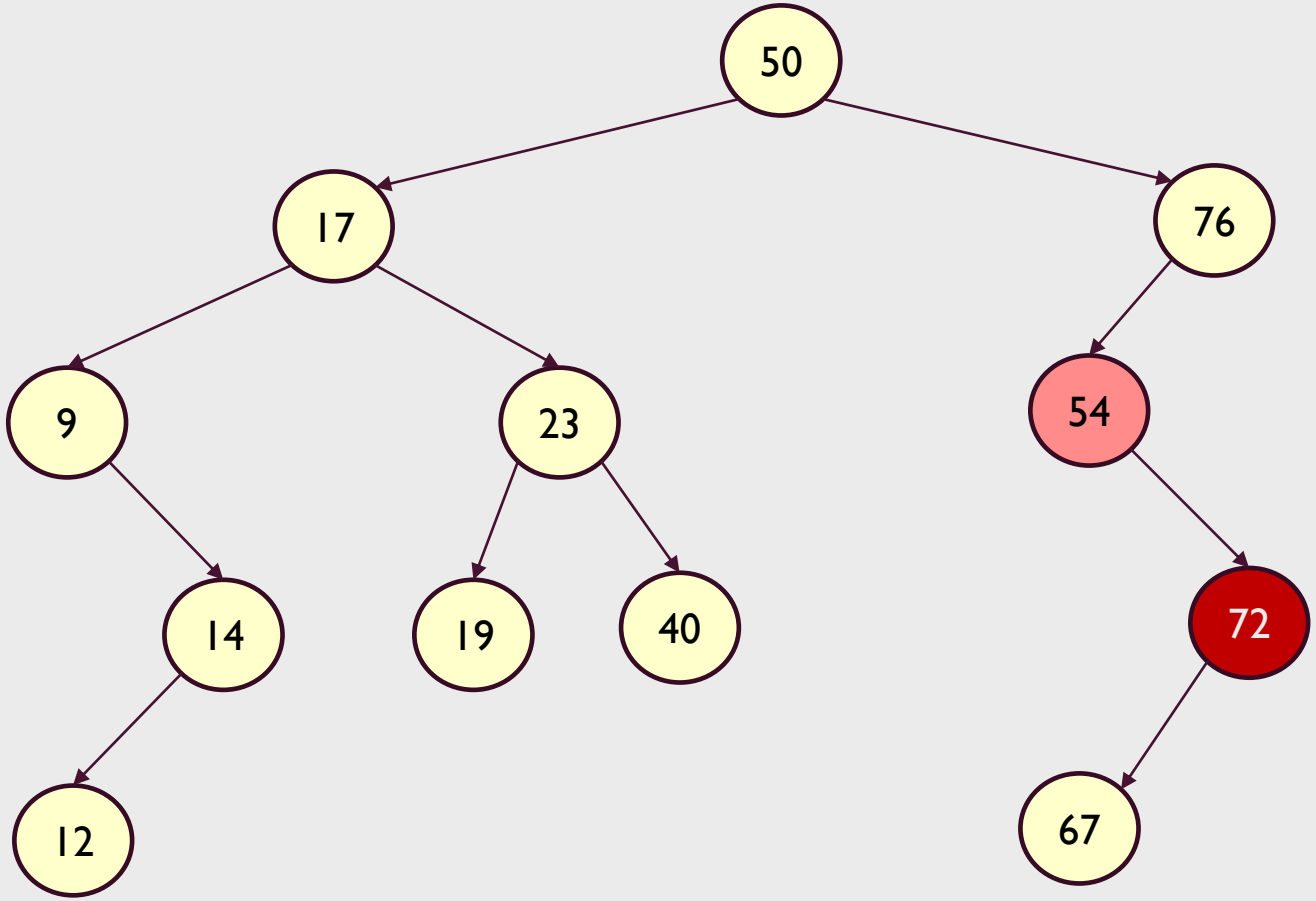

**Don't have right subtree. Go up until you're not right child.**

# Innovations – BST Sorter

## Traversal Example

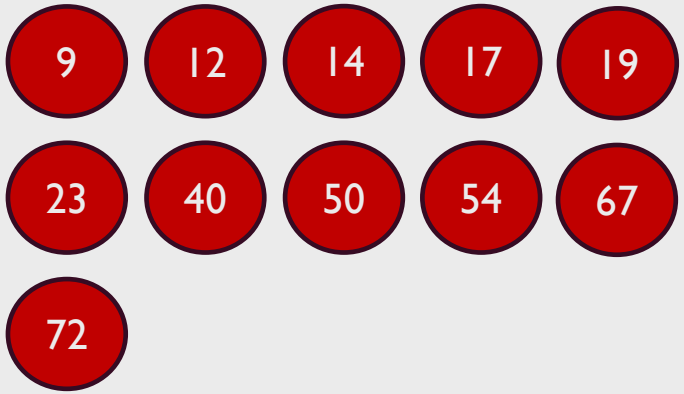



**Keep going up**

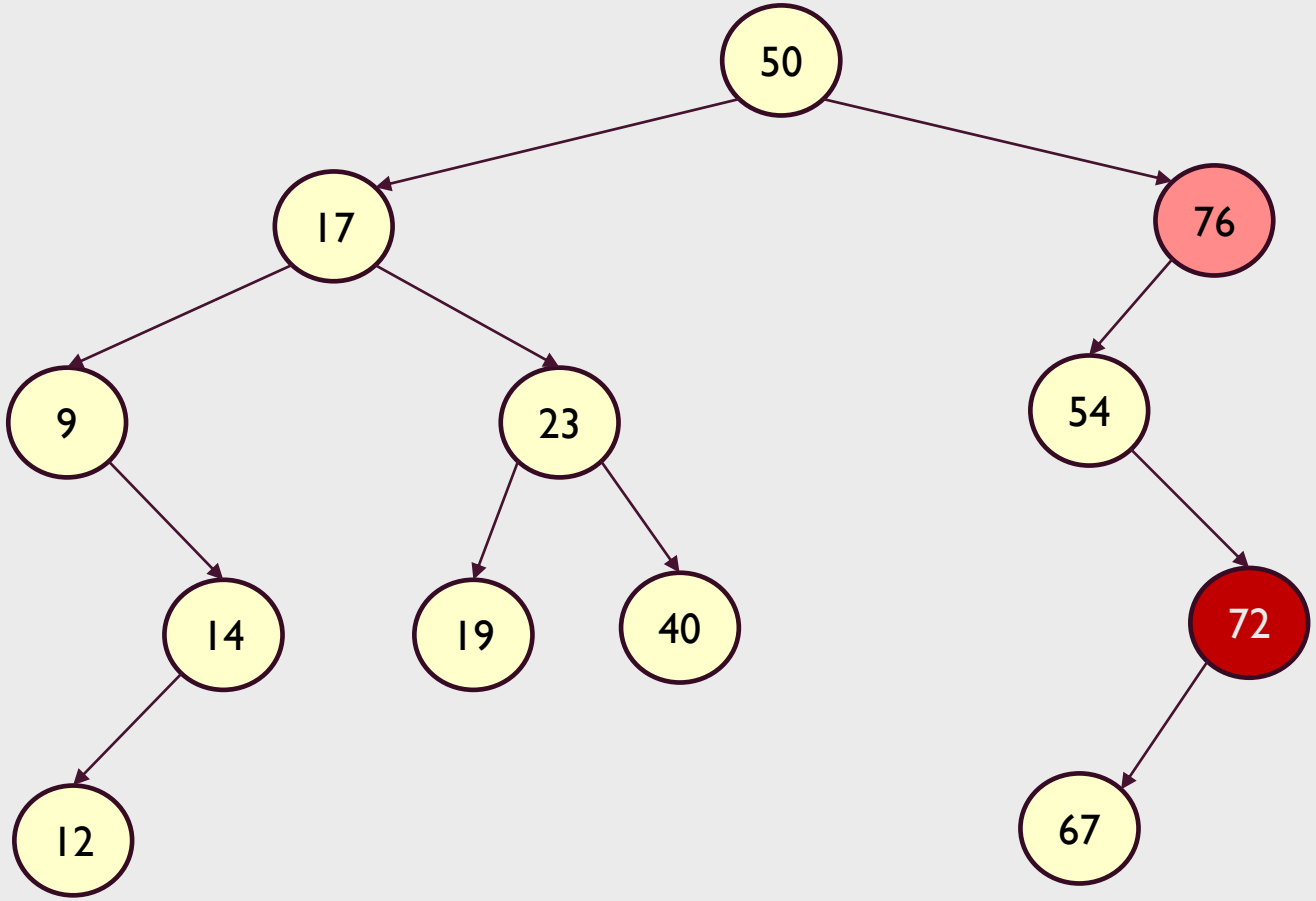


# Innovations – BST Sorter

## Traversal Example



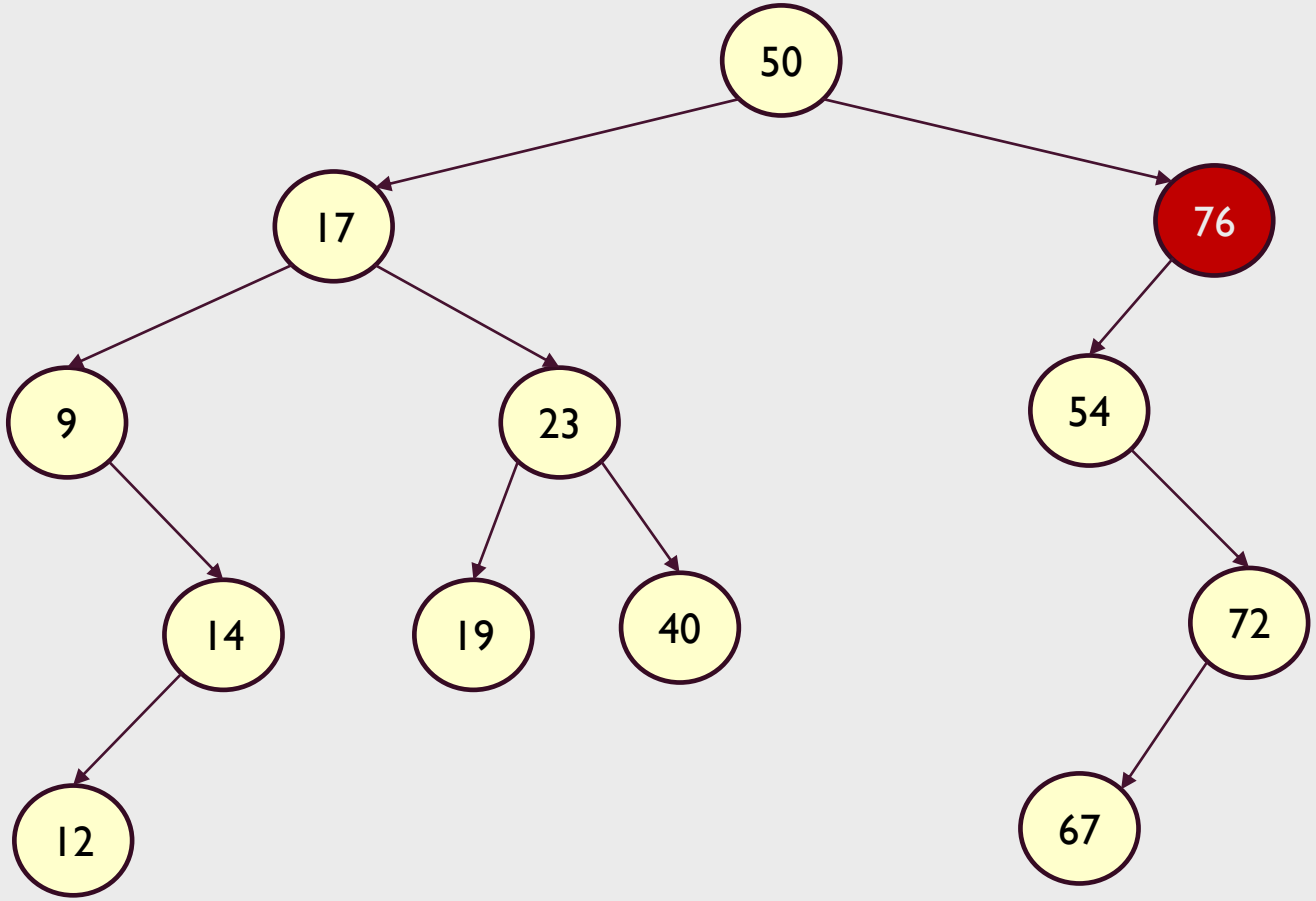

**We are not right child → found nx!**



# Innovations – BST Sorter

## Traversal Example

9 12 14 17 19
23 40 50 54 67
72 76




**We are max (and no list)**
**→ ALL DONE!**

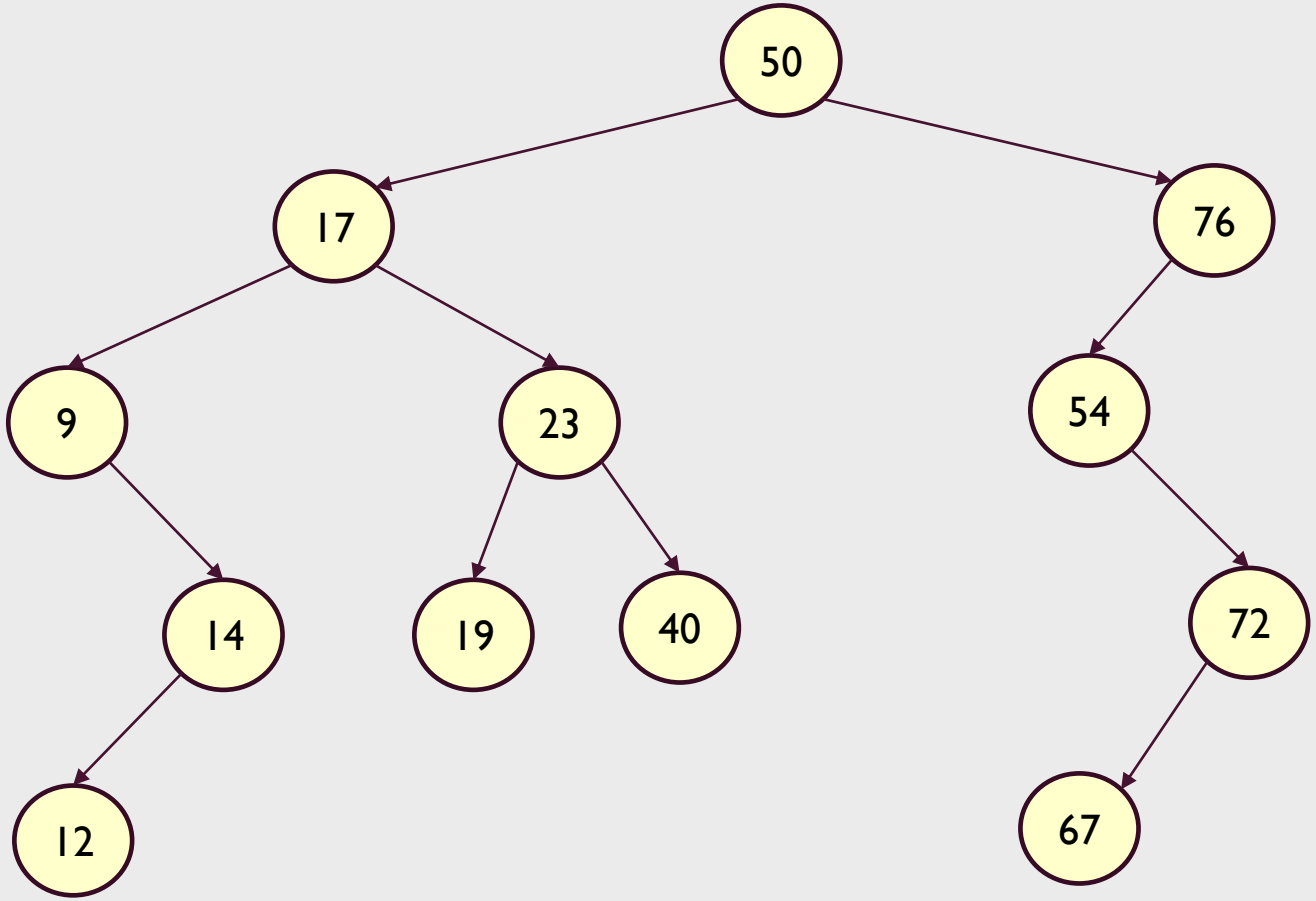
# Innovations – BST Sorter

## Traversal Example

9 12 14 17 19

23 40 50 54 67

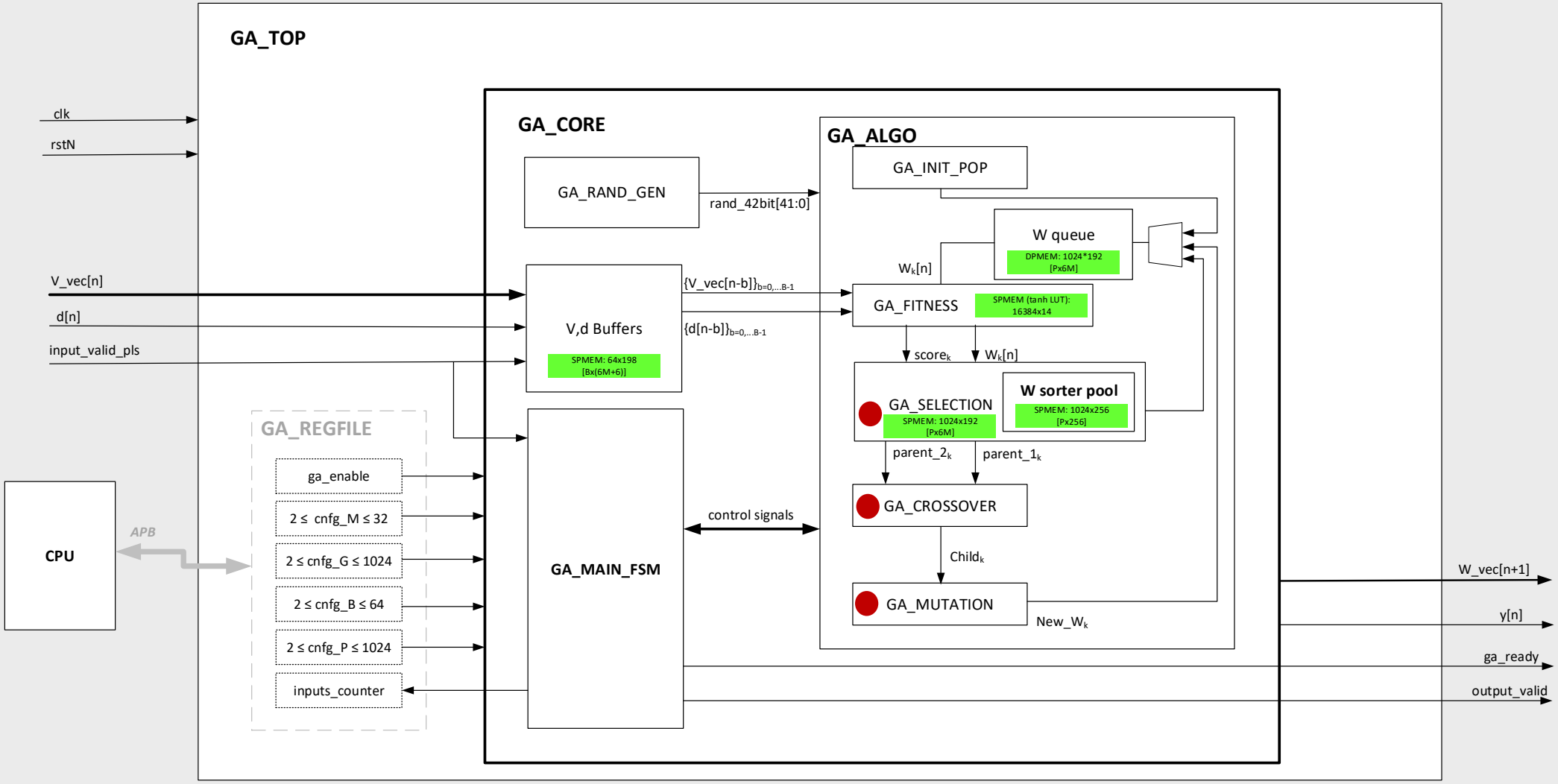
72 76



# Innovations – BST Sorter

- **BST sorter:**
  - Average number of clocks per insert operation: log(N)
  - Average number of clocks for in-order traversal: 2N
  - Area efficient
    - SRAM
    - Less comparators

# Innovations – Pseudo-modulus



# Innovations – Pseudo-modulus

- **Problem**:
  - Random parent address selection
  - Random weight index selection
  - Overall: **random number between 0 and z-1 is required**
    - z – register
    - N random bits are provided

# Innovations – Pseudo-modulus

- **Can't use the N bits as they are!**
  - Example: N=4 , z=12: If 12,13,14,15 will be the random number – the result will be out of range
- **Conclusion:** manipulation should be made on the N random bits
- **Area efficient and quick solution is needed!**

# Innovations – Pseudo-modulus

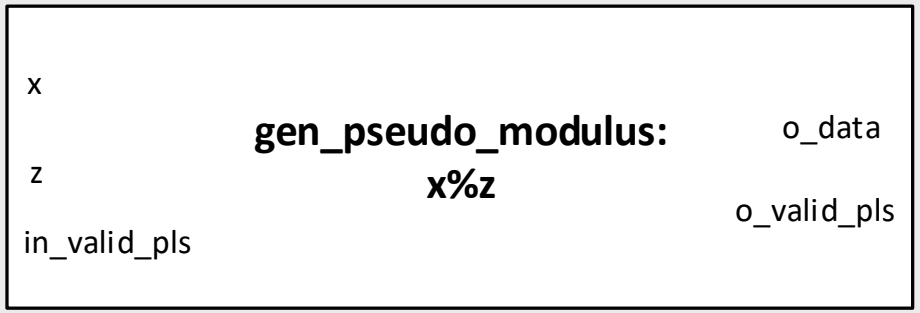
- **Possible solutions**:
  - Modulus calculator
    - Single accumulative subtractor: area efficient, slow
    - Combinatorial: fast, not-area efficient
  - Modulus LUT
    - 1 clock cycle, requires large memory

# Innovations – Pseudo-modulus

- **Observation:** The real original random number is not important, only its distribution

***Chosen solution: pseudo modulus operation***

# Innovations – Pseudo-modulus

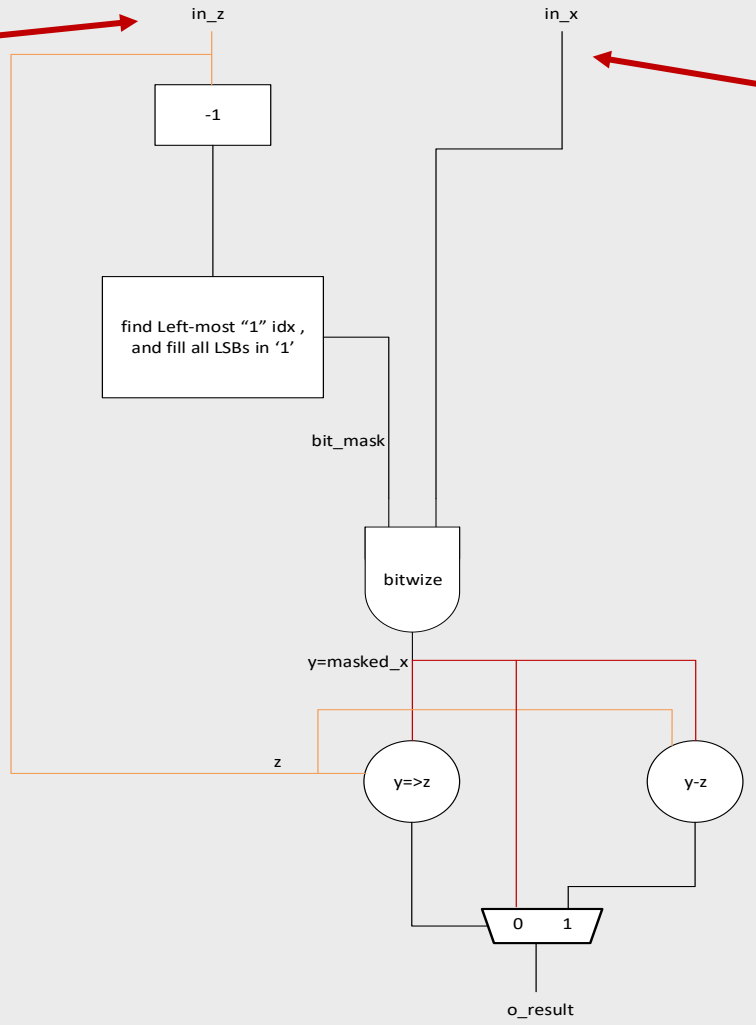


# Innovations – Pseudo-modulus – Example

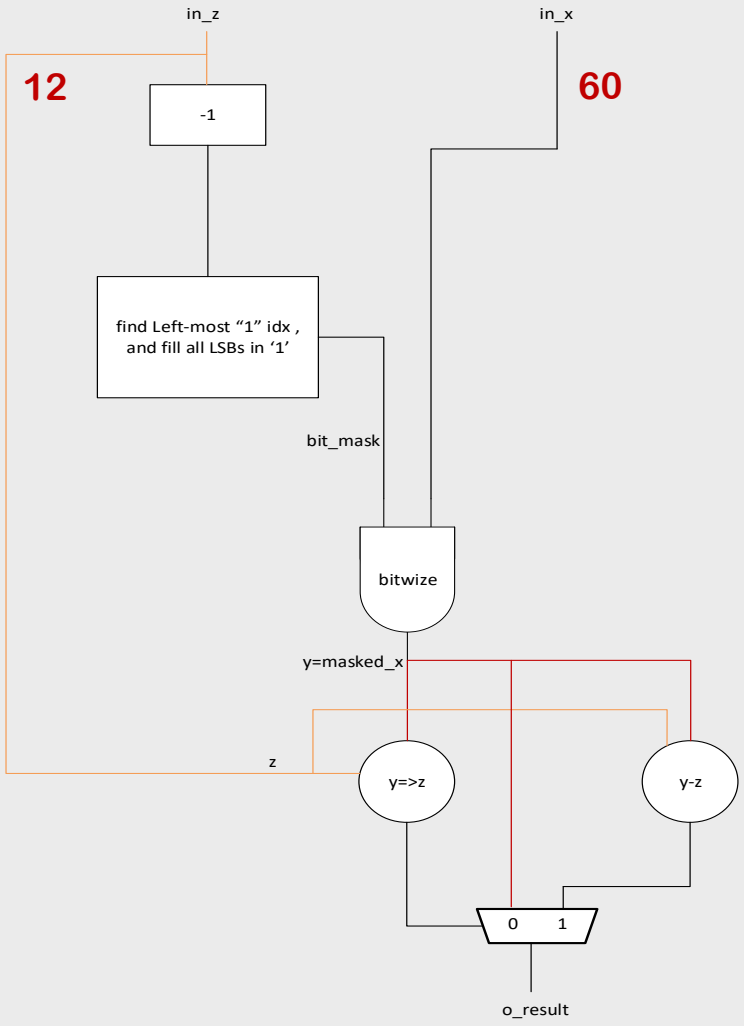
reg: 12 (z=12)

6bit random: 60 (x=60)

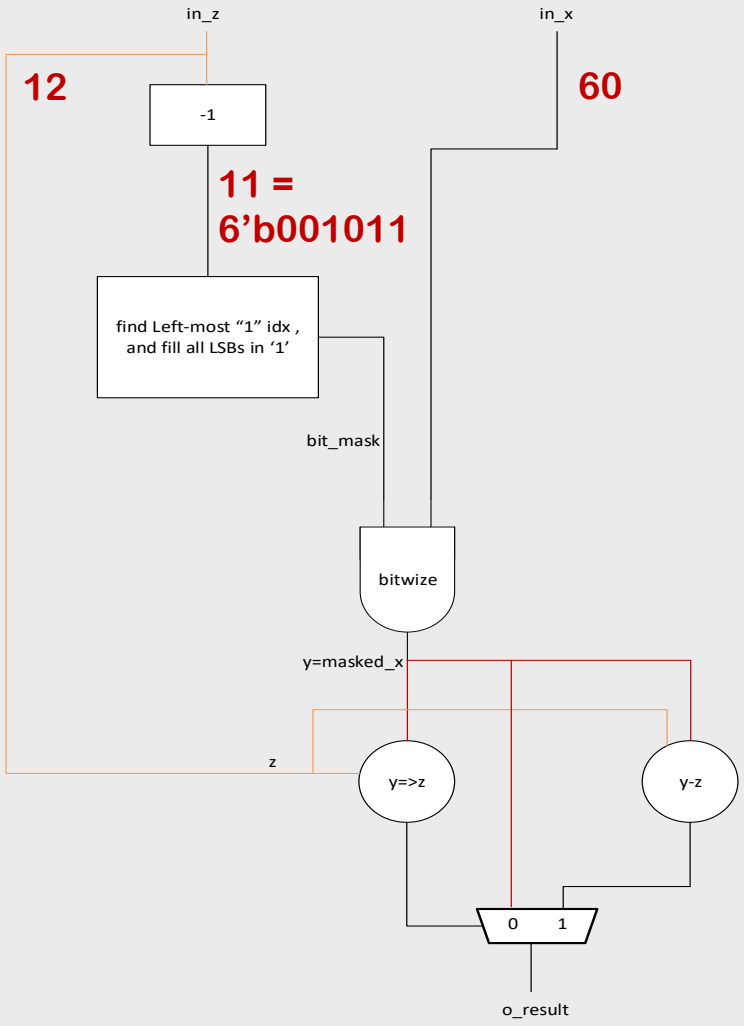
**Allowed result range: [0,11]**



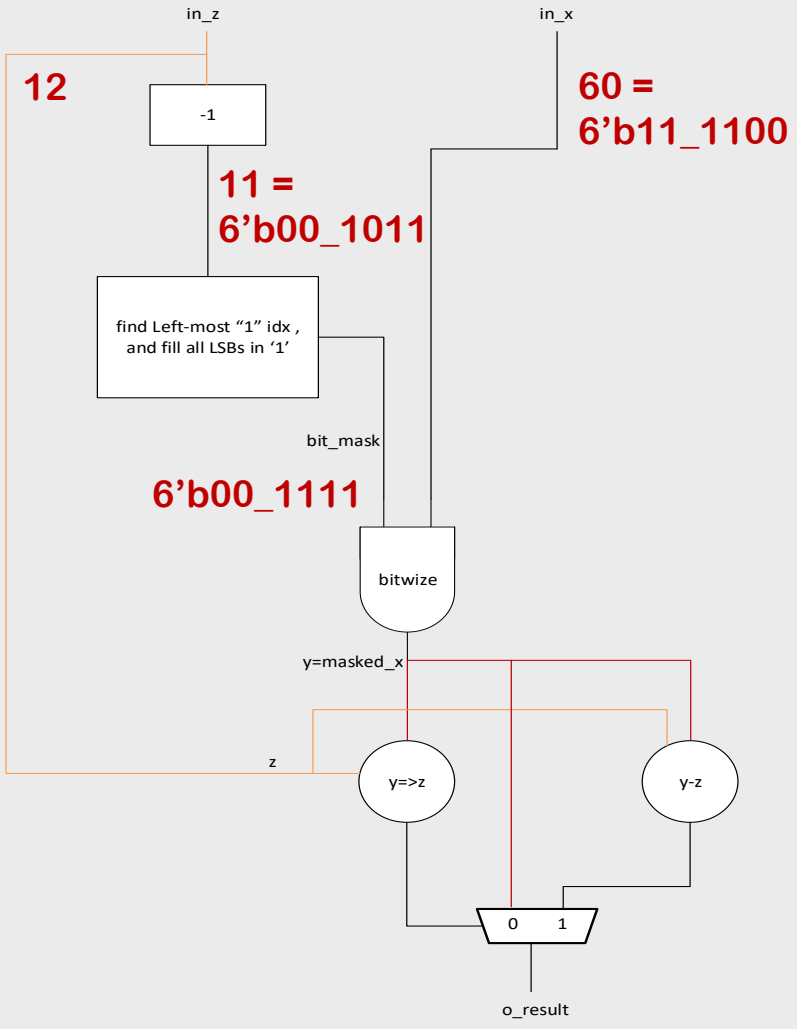
# Innovations – Pseudo-modulus – Example



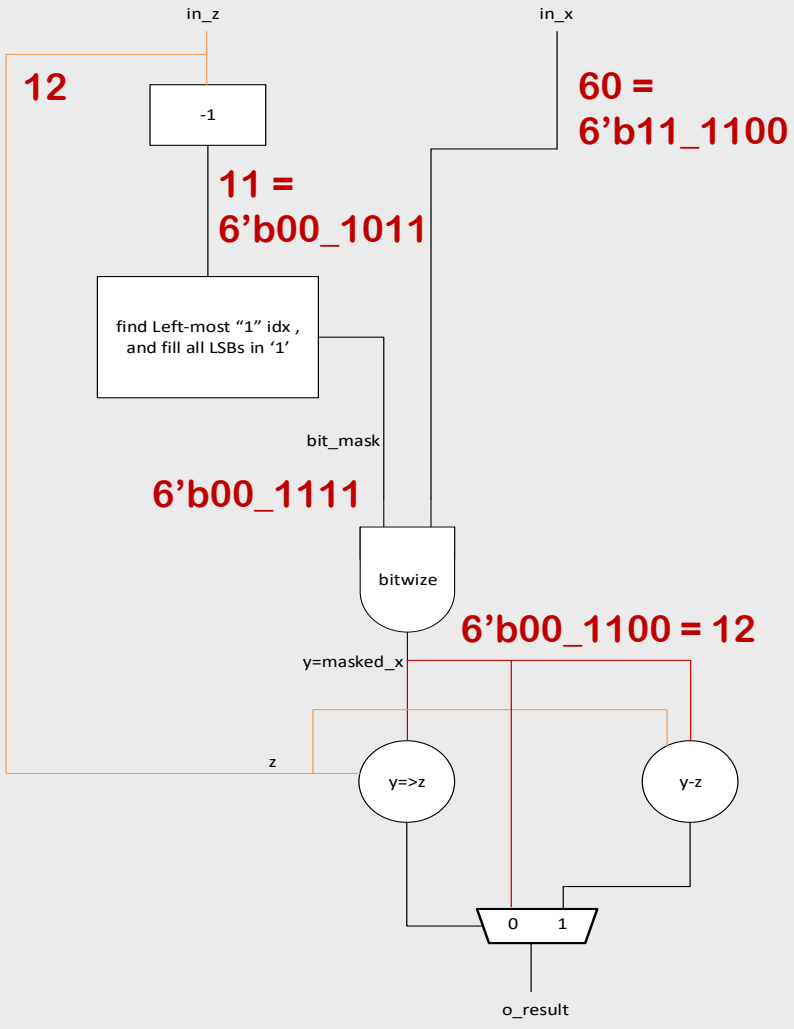
# Innovations – Pseudo-modulus – Example



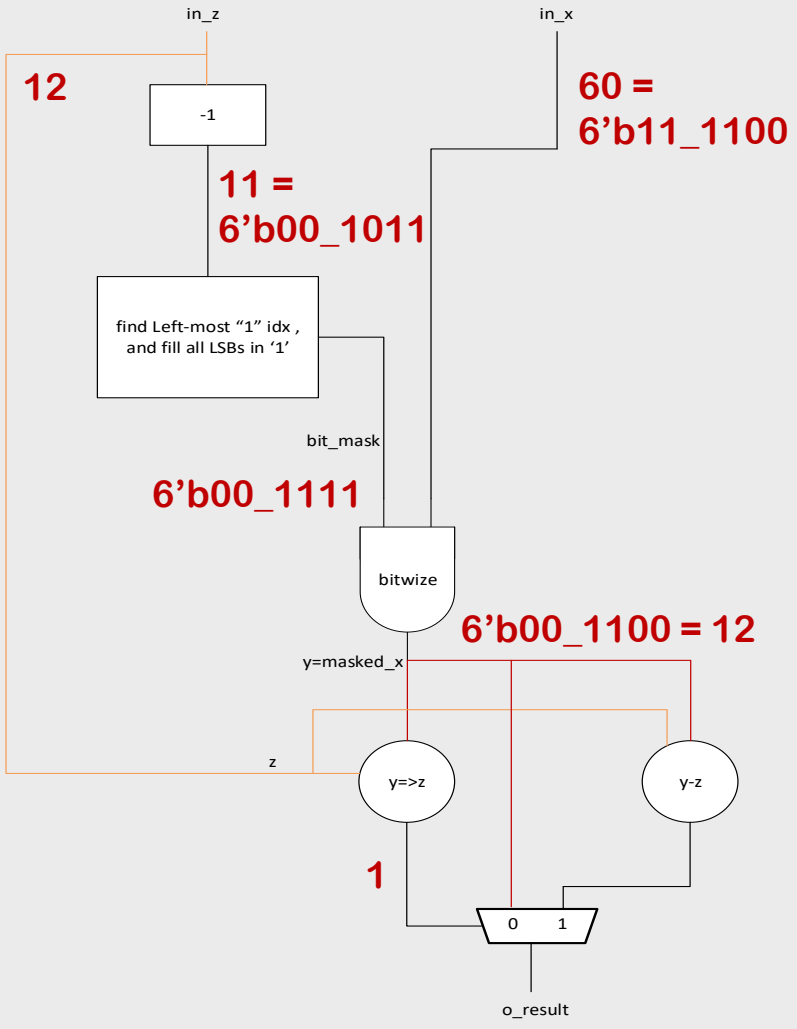
# Innovations – Pseudo-modulus – Example



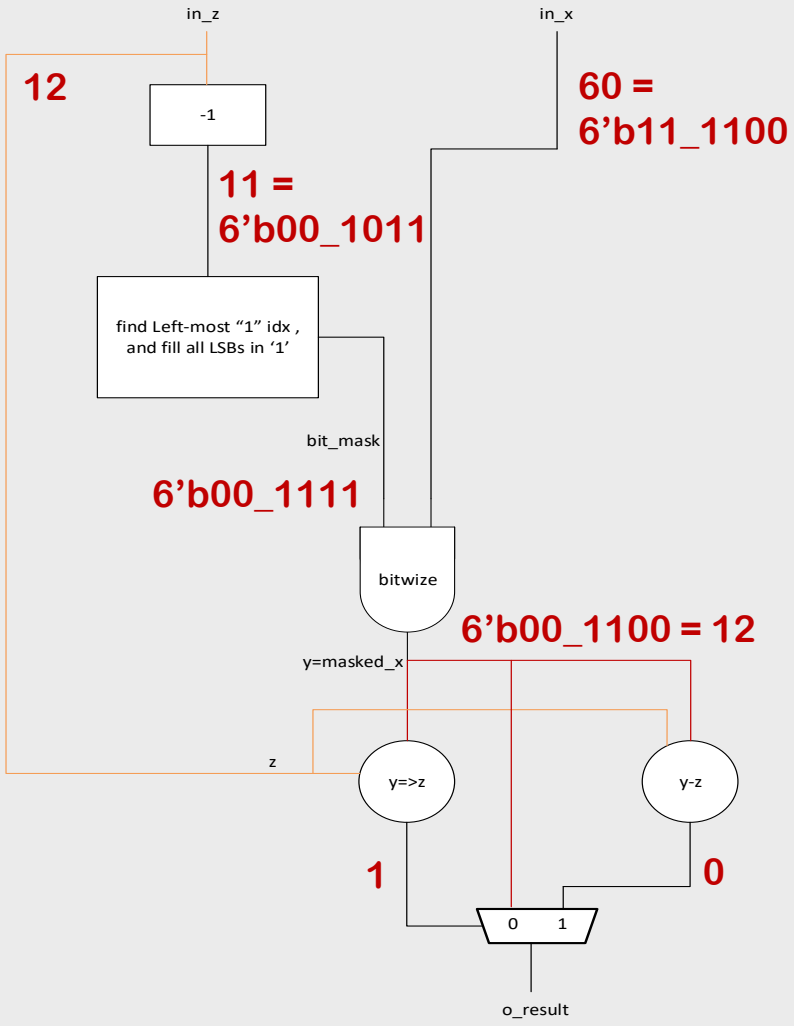
# Innovations – Pseudo-modulus – Example



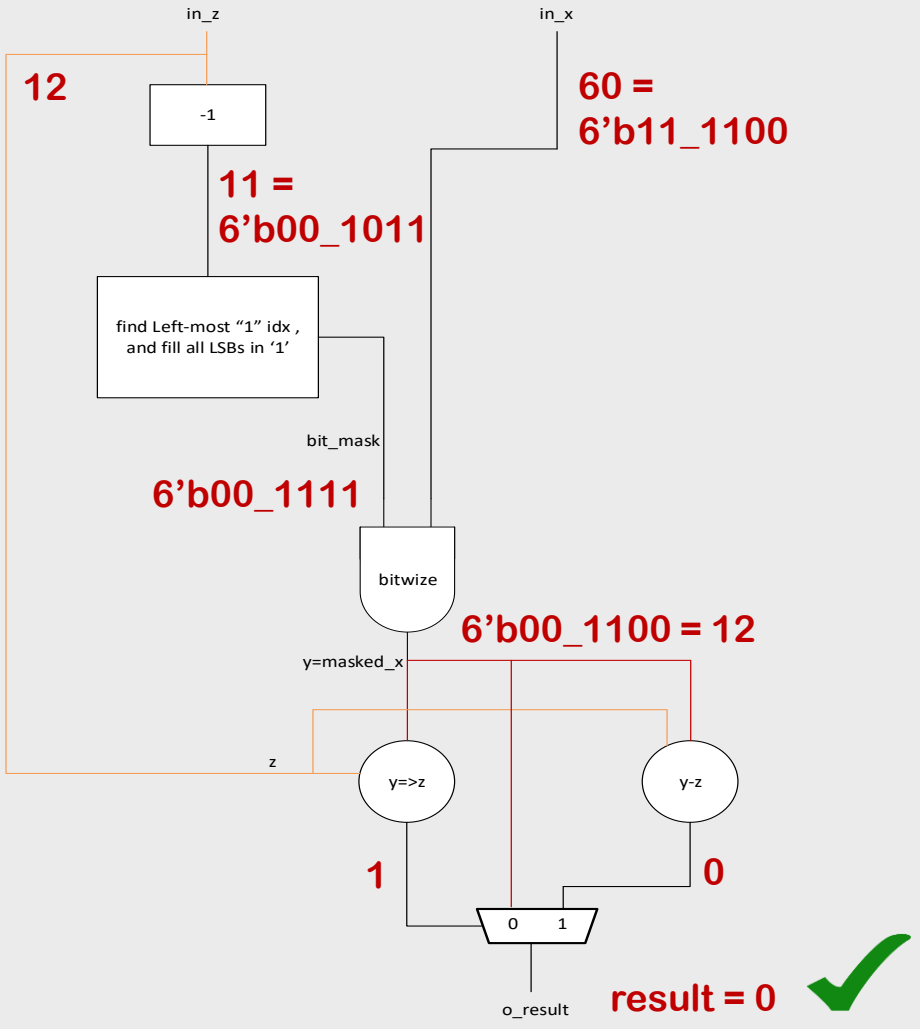
# Innovations – Pseudo-modulus – Example



# Innovations – Pseudo-modulus – Example



# Innovations – Pseudo-modulus – Example



# Innovations – Pseudo-modulus

- **Pseudo modulus:**
  - Preserves original distribution
  - Combinatorial (fast)
  - Area efficient

# Steps

- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

# Steps

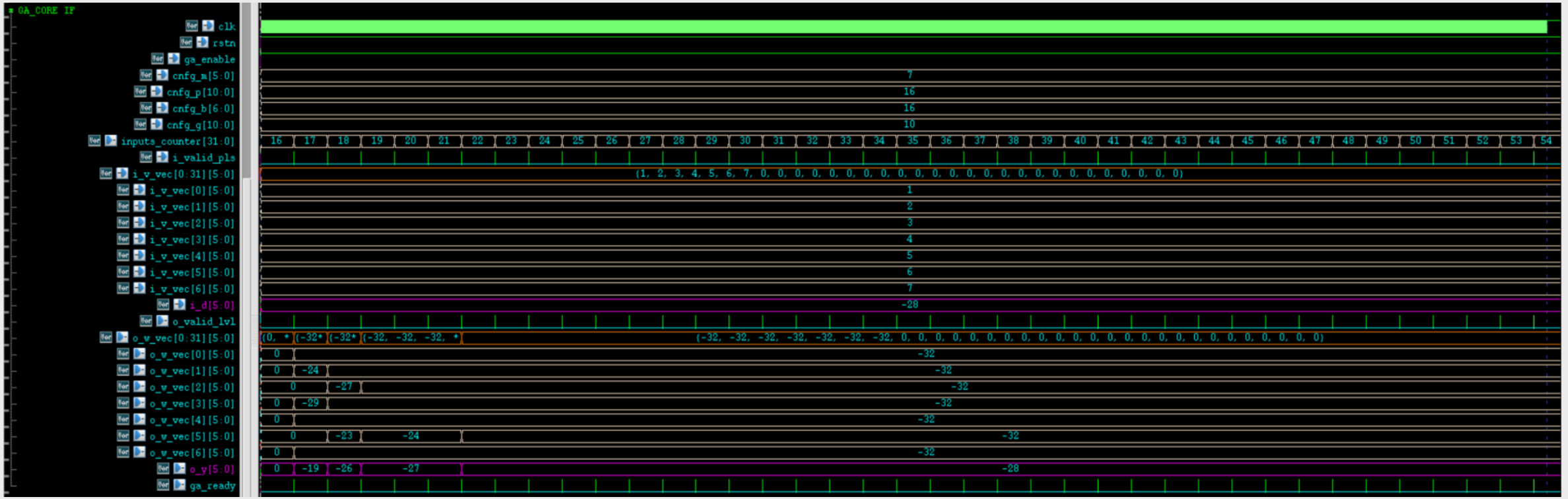
- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

# Functional Simulations

- Generic component simulations
- Unit level simulations
- Top level simulations

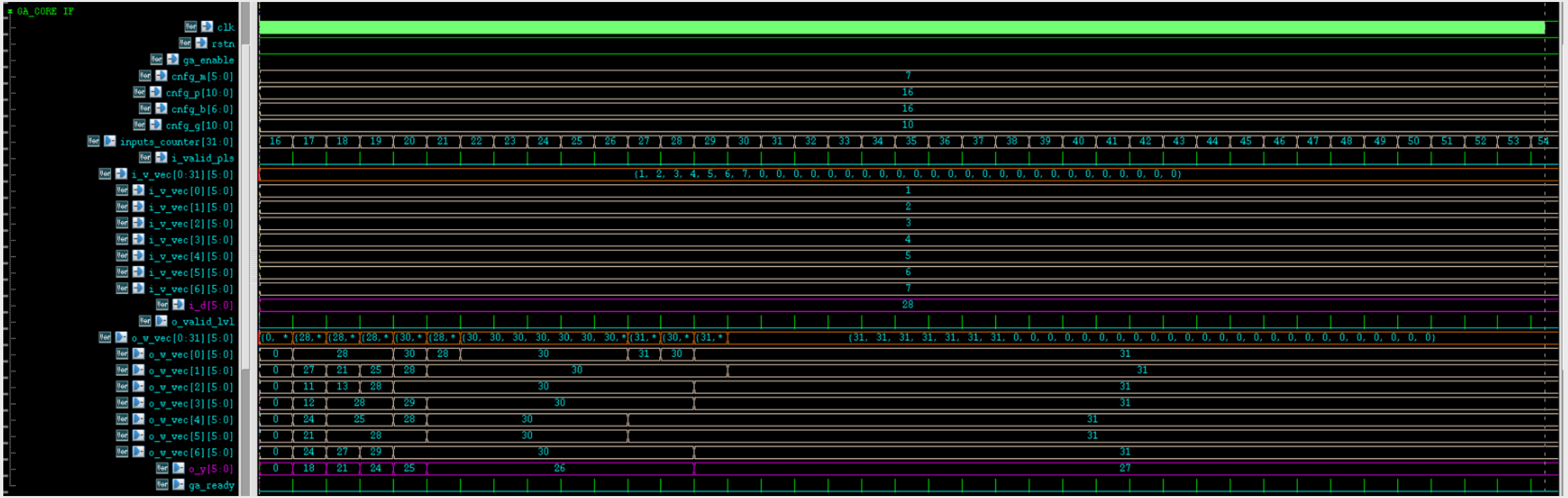
# Functional Simulations – Top Level Simulation (1)

**Constant V_vec, d = - $\sum_{k=0}^{6}$ V_vec[k] ( real_W = {-1, -1, -1, -1, -1, -1, -1} · 32 )**



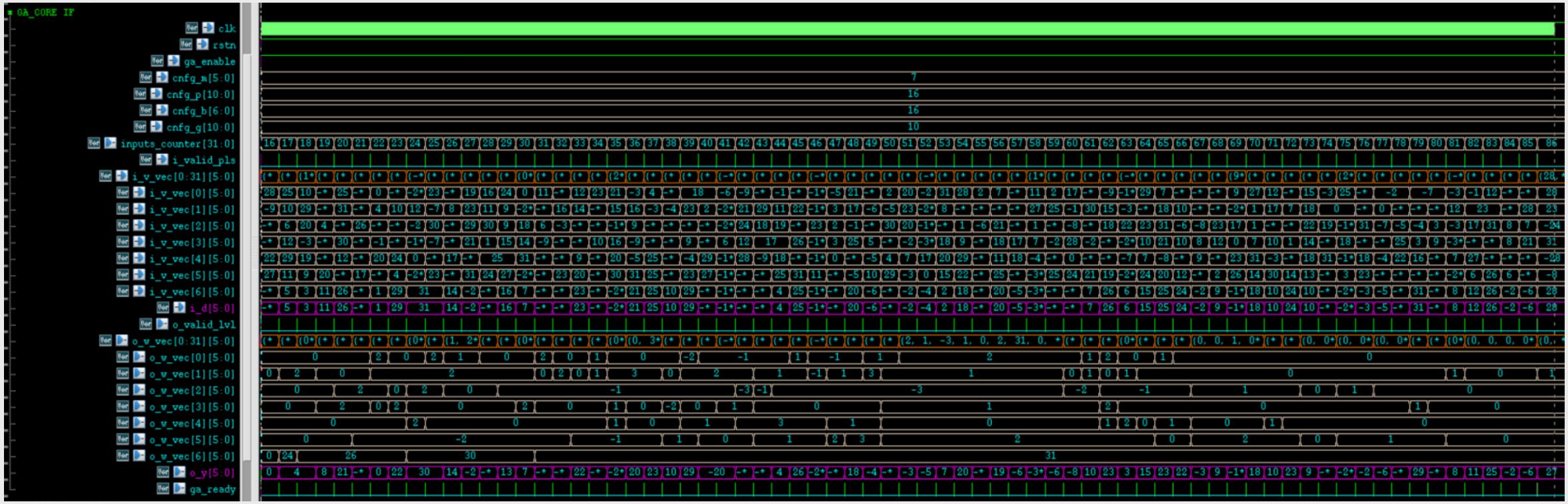
# Functional Simulations – Top Level Simulation (2)

**Constant V_vec, d = $\sum_{k=0}^{6}$ V_vec[k] ( real_W = {1, 1, 1, 1, 1, 1, 1} · 32 )**



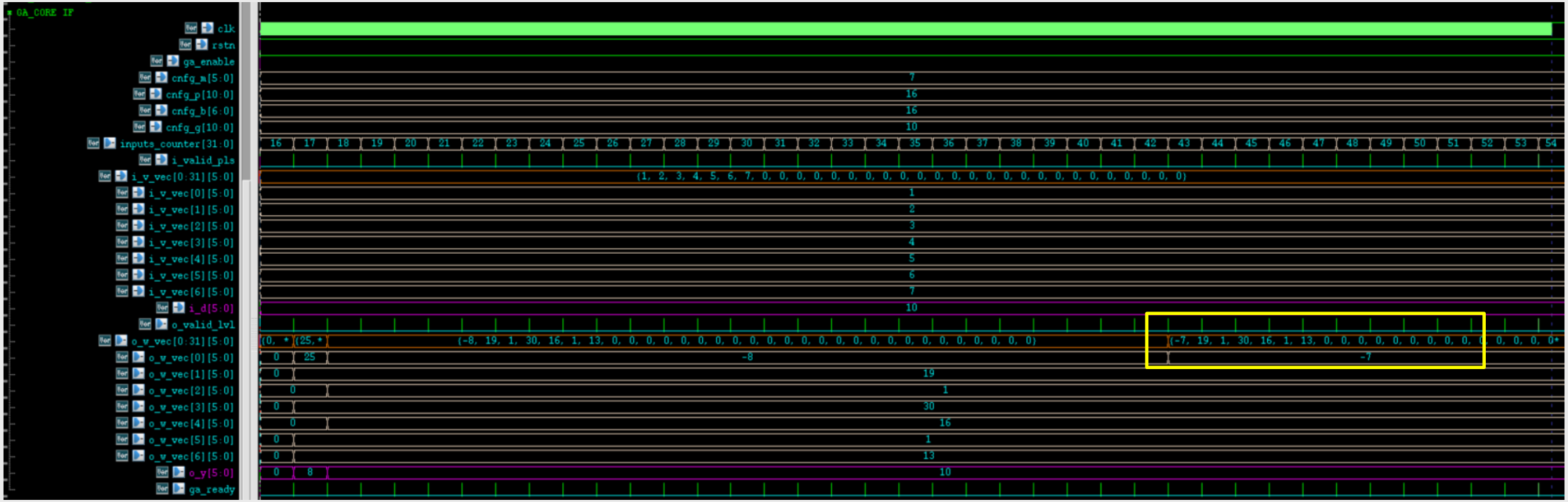
# Functional Simulations – Top Level Simulation (3)

**Random V_vec, d=V_vec[6] ( real_W = {0, 0, 0, 0, 0, 0, 1} · 32 )**



# Functional Simulations – Top Level Simulation (4)

**Constant V_vec, d = $\sum_{k=0}^{3}$ V_vec[k] ( real_W = {1, 1, 1, 1, 0, 0, 0} · 32 )**



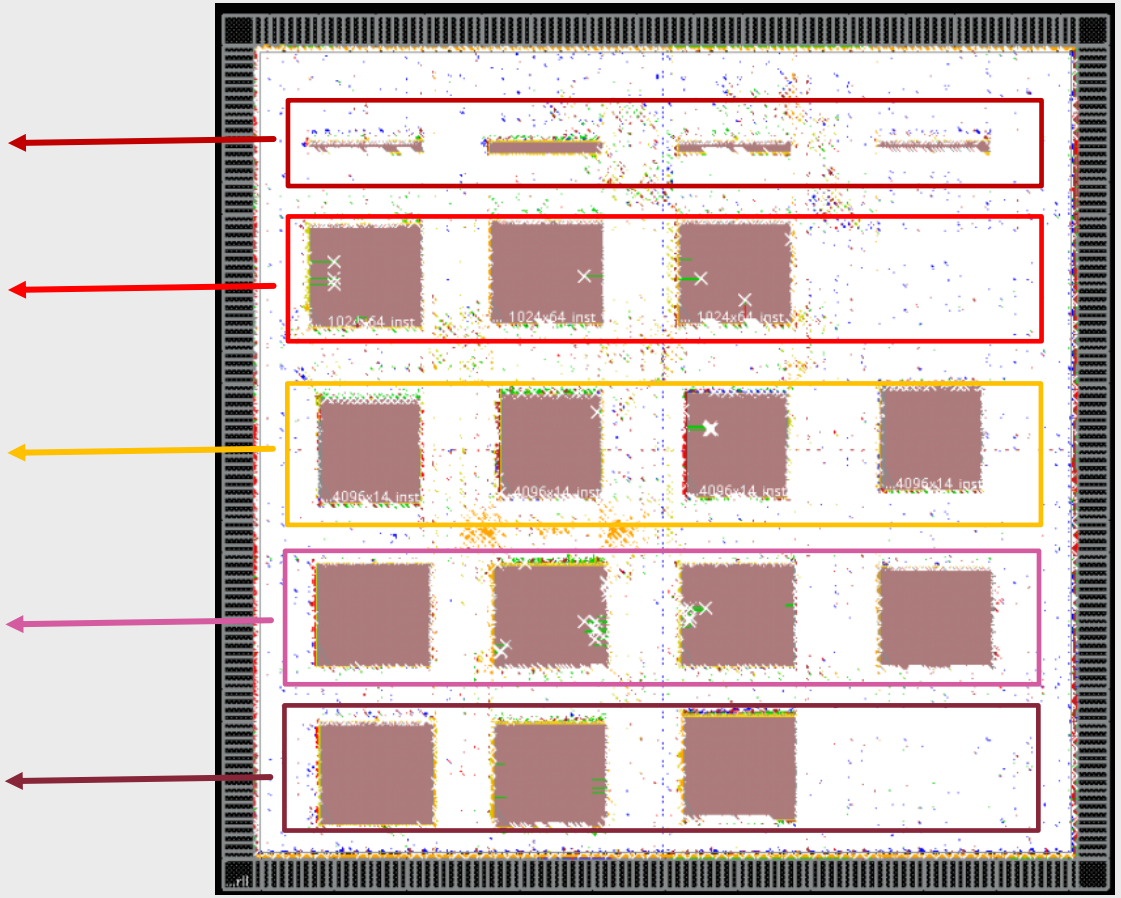
# Steps

- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

# Synthesis and Floorplan

- Tower 180um technology
- Clock Frequency: 125MHz
- Synthesis tool: Synopsys Design Vision
- Floorplan tool: Cadence Innovus

# Floorplan



# Synthesis Timing Report

- **Worst slack = 0**
  - **Timing requirements met**
- **Critical path:**

```
 Startpoint: cnfg_p_reg[4]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint:
u_ga_core_inst/u_ga_algo_top_inst/u_selection_inst/u_pool_inst/MEM_IF_DEPTH_GT_64_LTE_1
024.INST_4WIDTH_LOOP[0].u_mem_1024x64_inst
            (falling edge-triggered flip-flop clocked by clk)
```

# Steps

- Architectural and logic design
- SystemVerilog implementation
- SystemVerilog functional simulations
- Synthesis and floorplan
- Performance analysis

# Performance

- GA_SLECTION sorter extraction phase is the bottleneck
- GA_FITNESS and GA_SELECTION sorter are the longest units
  - Depending on parameters one will be longer then the other
- Deterministic latency/throughput can not be set
  - Average case was calculated

# Summary

- Genetic algorithm approach is becoming popular.
- GA accelerator hardware was designed to provide full solution for the prediction of filters fixed-point weights with simple data interface and high flexibility to software.
- Area and timing trade-offs were taken into account.

# Next steps

- Full verification
- Performance improvement
  - GA_FITNESS pipeline
  - Separate clocks for CPU IF and CORE
- Algorithm improvements
  - Long filter penalty

# Personal point of view

- We have really enjoyed the project
- The project contained a lot of topics from different and diverse fields - hardware, software, optimization, adaptive algorithms, etc.…
- VLSI lab

# Acknowledgements


- **Mr. Shahar Gino**

  Project Supervisor

- **Mr. Geol Samuel**

  Faculty Supervisor


# Thank You!

Questions?