Minimum Dominating Set
(possibly with minimum time?)

Vivatsathorn Thitasirivit

2023/2 2110452 High Performance Architecture
Faculty of Engineering, Chulalongkorn University
22 Apr 2024

Outlines

- Minimum Dominating Set (MDS) Problem (skipped)
- Approaches
- Optimizing Optimization Problem
- What about Docker?

Initial Approach: Better brute-force?

- Checking all 2^v combinations linearly has $O(v^2 2^v)$ time complexity
  - For each $k \in v$: Test k-chosen vertices
    - Fill bitmask of length v
    - Do combinatorial check of $C(v, k)$
      - Check v vertices
- Checking combinations using binary search uses about $O(v^2 1.865^v)$ for v between 0 and 100. (No closed solution for time complexity)
  - Search within space of $k \in v$ using binary search:
    - Fill bitmask of length v
    - Do combinatorial check of $C(v, k)$
      - Check v vertices
- Ternary search?, *N*-ary search?, Offset binary search?

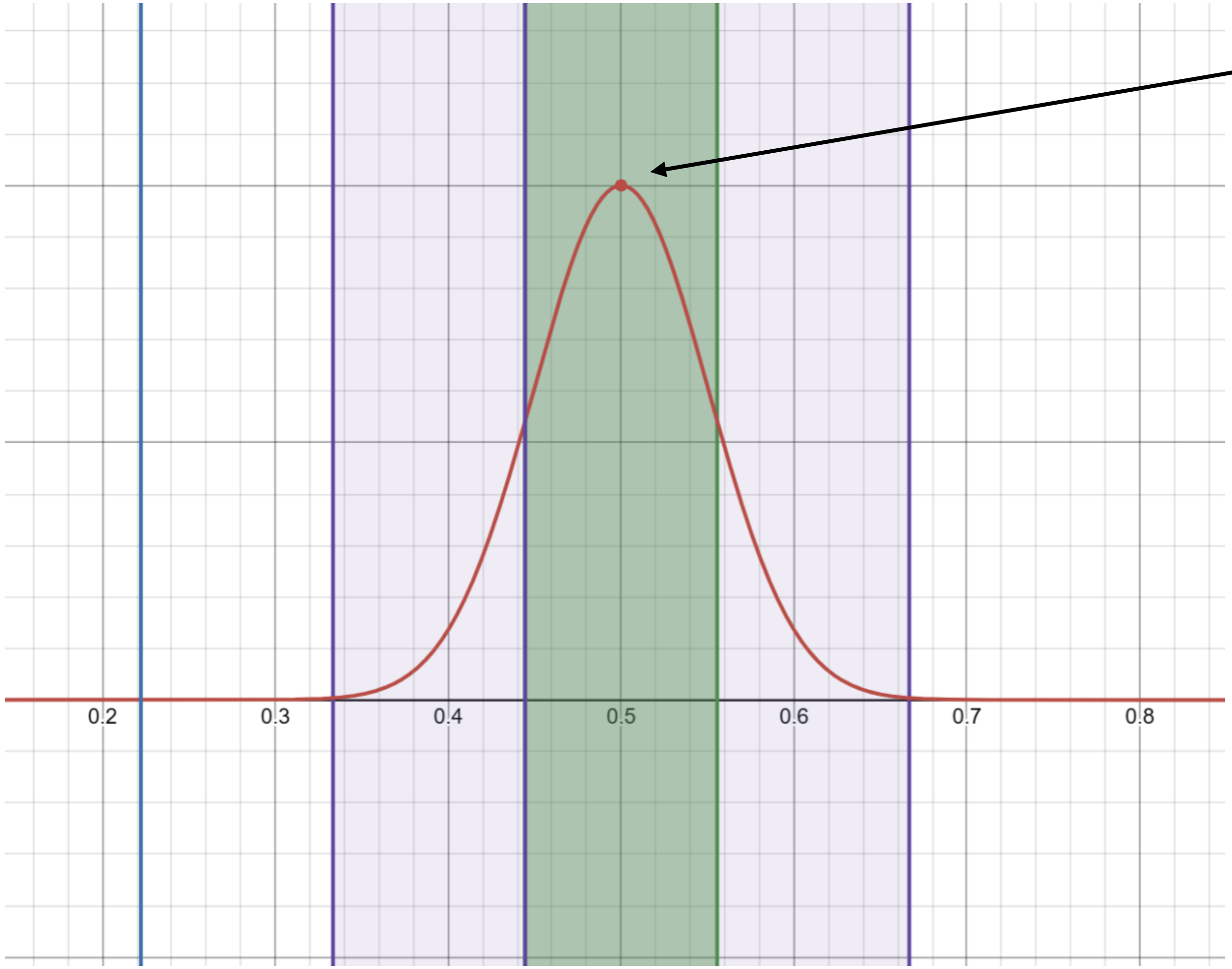
Initial Approach: Better brute-force?

- Checking all 2^v combinations linearly has $O(v^2 2^v)$ time complexity
  - For each $k \in v$: Test k-chosen vertices
    - Fill bitmask of length v
    - Do combinatorial check of $C(v,k)$
      - Check v vertices
- Checking combinations using binary search uses about $O(v^2 1.865^v)$ for v between 0 and 100. (No closed solution for time complexity)
  - Search within space of $k \in v$ using binary search:
    - Fill bitmask of length v
    - Do combinatorial check of $C(v,k)$
      - Check v vertices

$$t(v) = \sum_{j=1}^{\mathrm{ceil}(\log_2 v)} \left(2v + \left(\mathrm{nCr}\left(v, \frac{v}{2^j}\right)\right)v\right)$$

- Ternary search?, *N*-ary search?, Offset binary search?

Initial Approach: Better brute-force?



10^{29} combinations
when $v = 100$

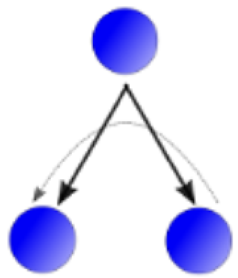
New Approach: Integer Programming

- Note 1: It's an optimization problem, why not use it?
- Note 2: A little bird told me, "Google OR-Tools."

https://developers.google.com/optimization

New Approach: Integer Programming

- Note 1: It's an optimization problem, why not use it?
- Note 2: A little bird told me, "Google OR-Tools."



OR-Tools won gold in the international constraint programming competition every year since 2013.

https://developers.google.com/optimization

Real question: which solver to use?

- OR-Tools provides several solvers to use.

- All ILP solvers I can use without acquiring licenses:
  - SCIP: Solving Constraint Integer Programs
  - CBC: COIN-OR Branch and Cut
  - BOP: Boolean Optimization Problem
  - SAT: Satisfiability Problem

- Also, Google's Constraint Programming solver worth testing
  - Google's CP-SAT Solver

What language to use OR-Tools?

- OR-Tools is written in C++, but also provides Python wrapper.

- Dry running on Python and C++ to compare speed
  - Using CBC Solver and ring-100-100 test input.
  - Python: 7000 microseconds average
  - C++: 6700 microseconds average

- No surprise, that 300 us is just an overhead calling shared objects.

- I'll use C++ anyway, why not?

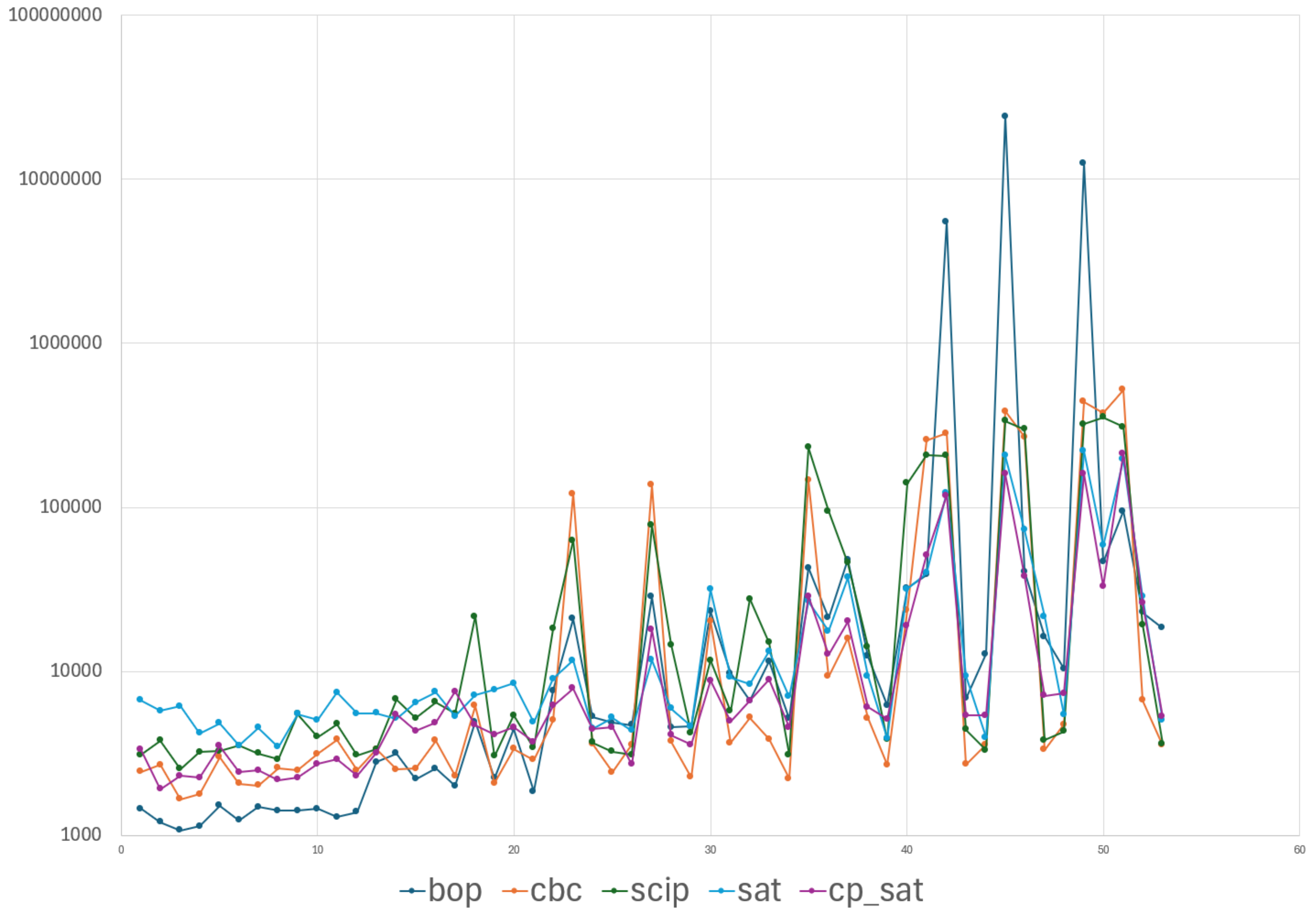
Time for some benchmarking!

- Script:
  - for file in $(ls ./data/input); do ./benchmark ./data/input/$file | tee -a out.txt; done

- Example output:
  - Time taken for procedure "I/O & Graph Initialization": 167 microseconds.
  - Time taken for procedure "BOP Backend": 24223 microseconds.
  - Time taken for procedure "CBC Backend": 5259 microseconds.
  - Time taken for procedure "SCIP Backend": 18454 microseconds.
  - Time taken for procedure "SAT Backend": 17674 microseconds.
  - Time taken for procedure "CP-SAT Model": 8918 microseconds.

- Parse and analyze...

Benchmark Results



Real question: which solver to use?

- From the benchmark these are solvers that takes the least time:

BOP	**19**
CBC	**20**
SCIP	2
SAT	2
CP-SAT	**10**

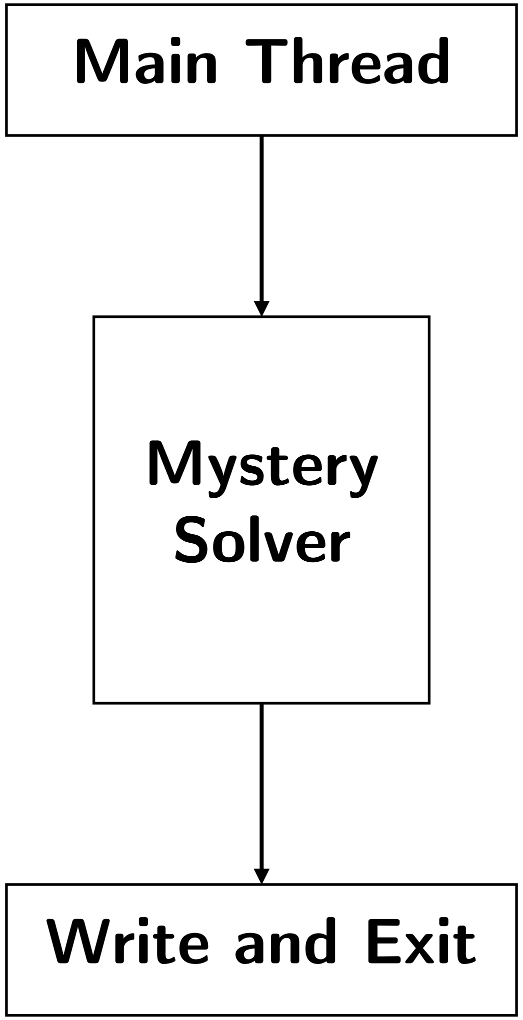
Candidate Solvers: Good for what graphs?

BOP	CBC	CP-SAT
Small	Large	Large
Simple/Complex	Simple	Complex

From my benchmark

Small: 1 to 20 Simple: not random graph
Large: 21 to 100 Complex: Random graph

Simple implementation



Why not use all?

Main Thread

Solver Threads

Spin up threads of solvers in this order

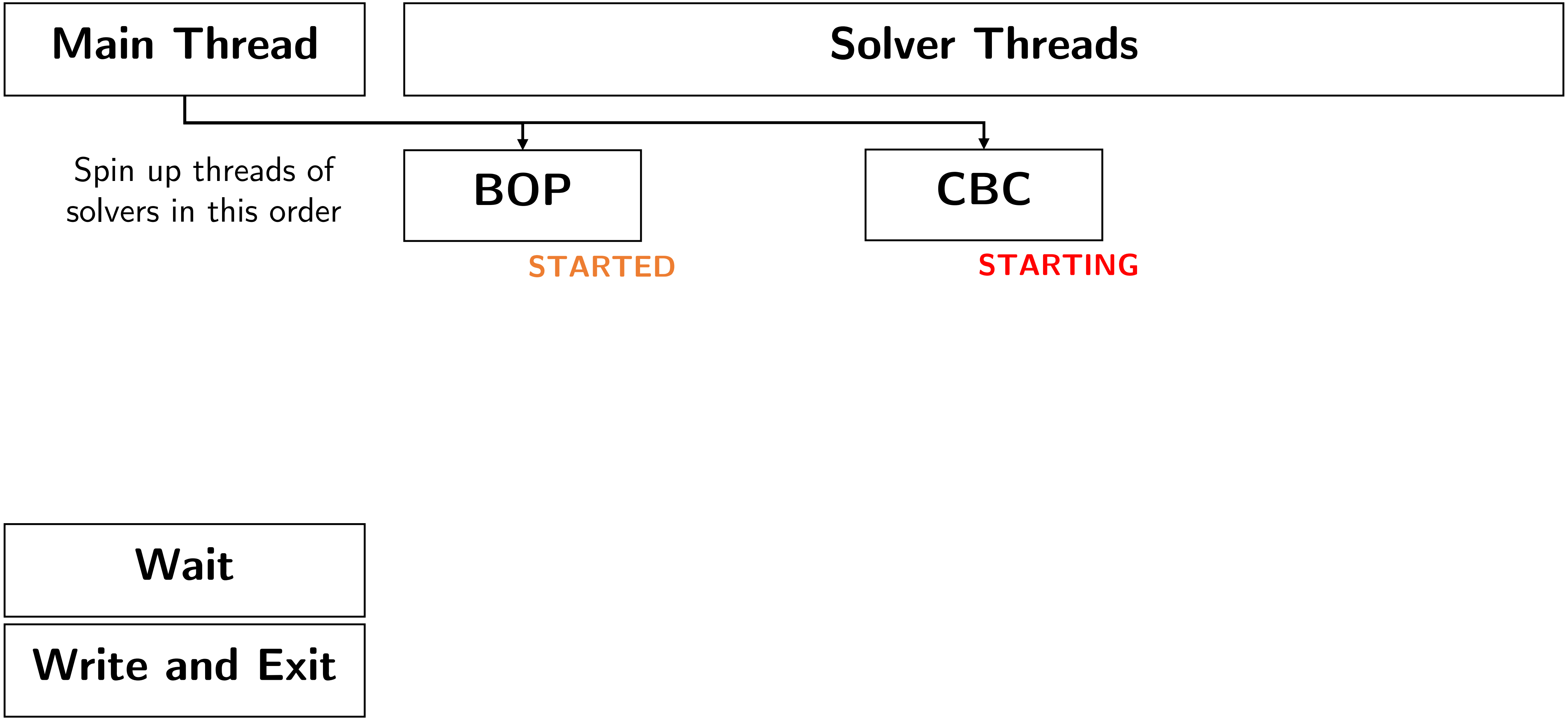
BOP

STARTING

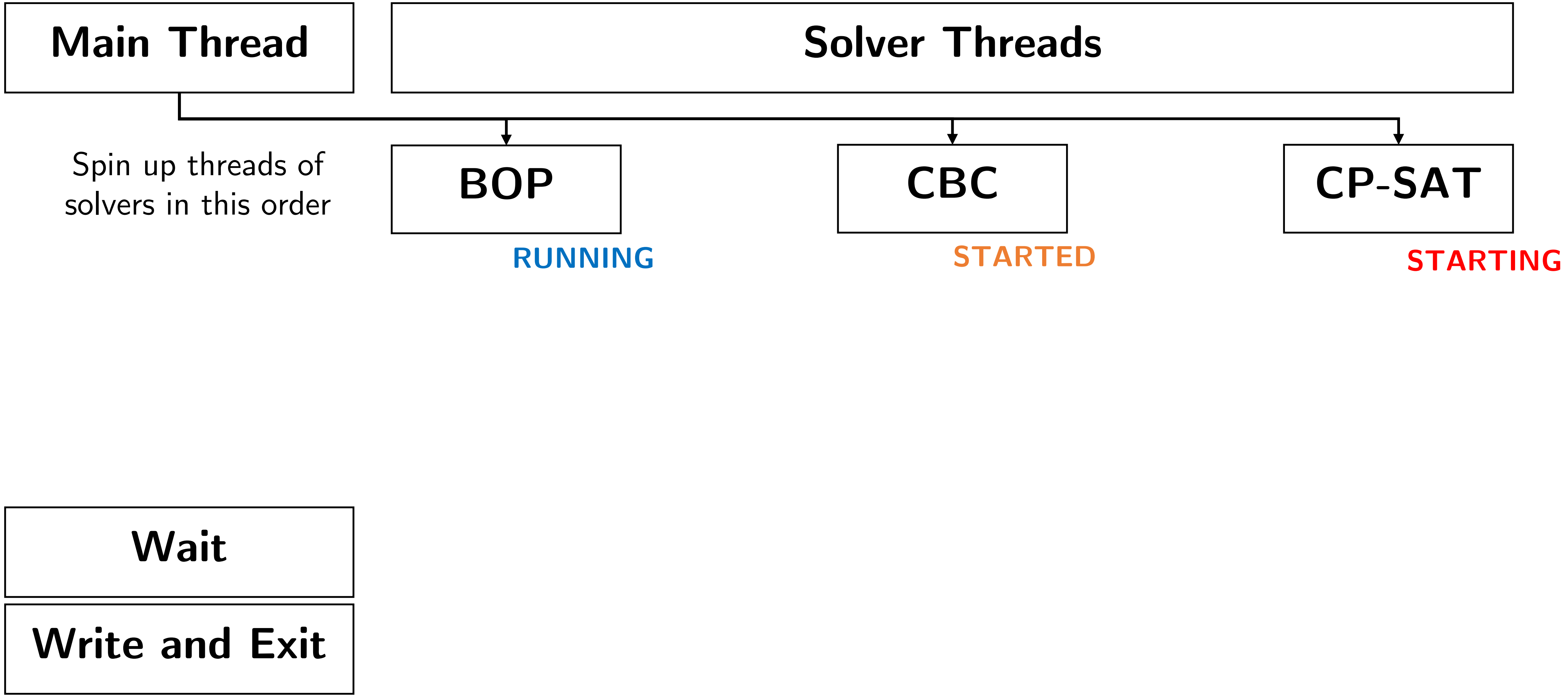
Wait

Write and Exit

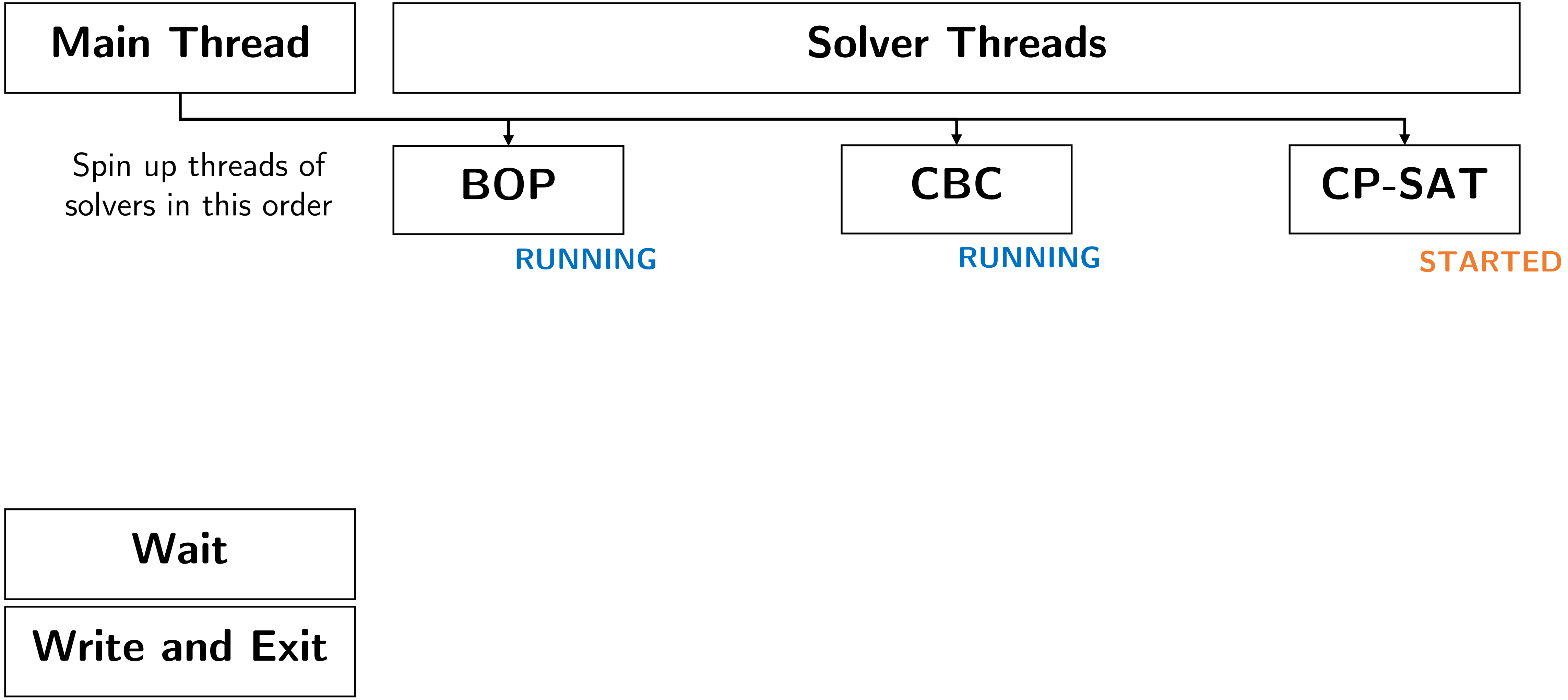
Why not use all?



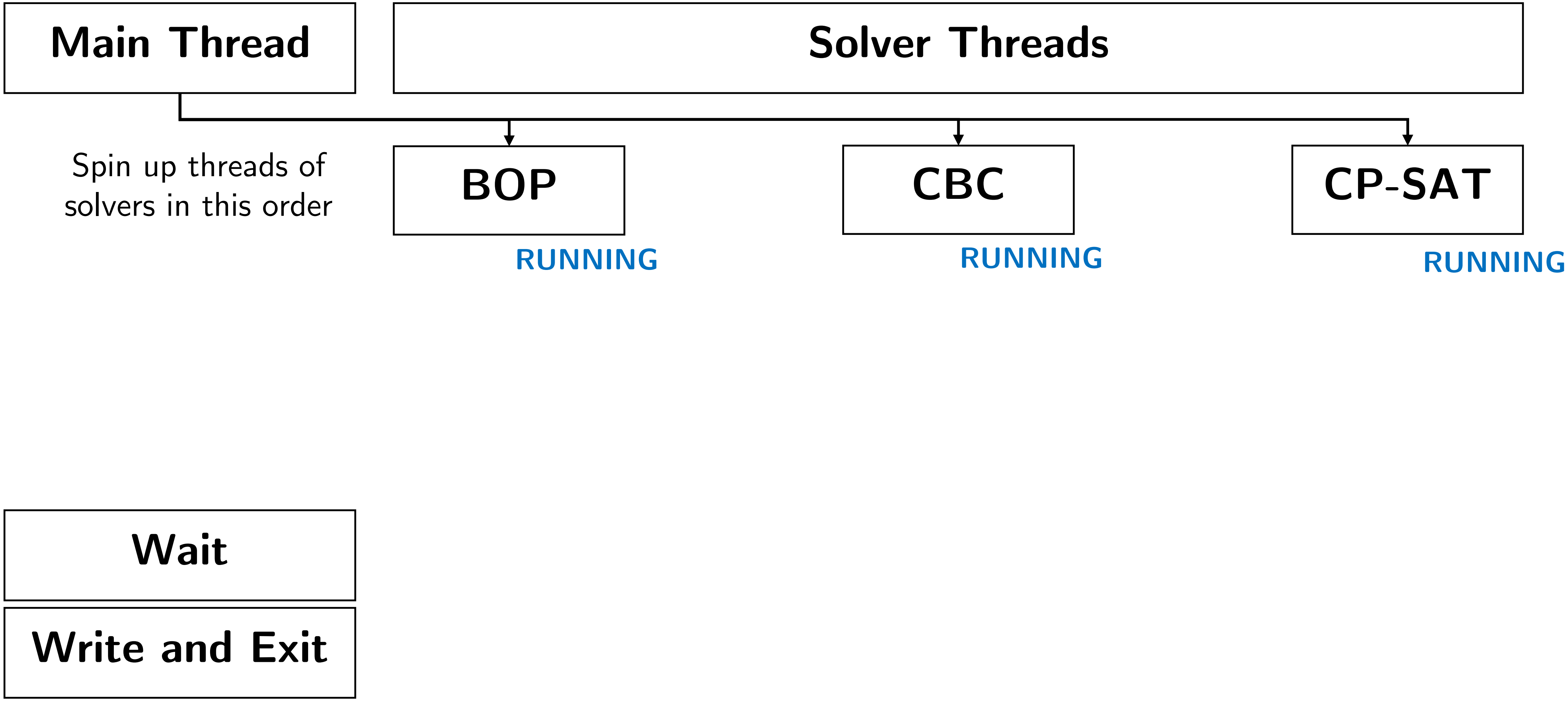
Why not use all?



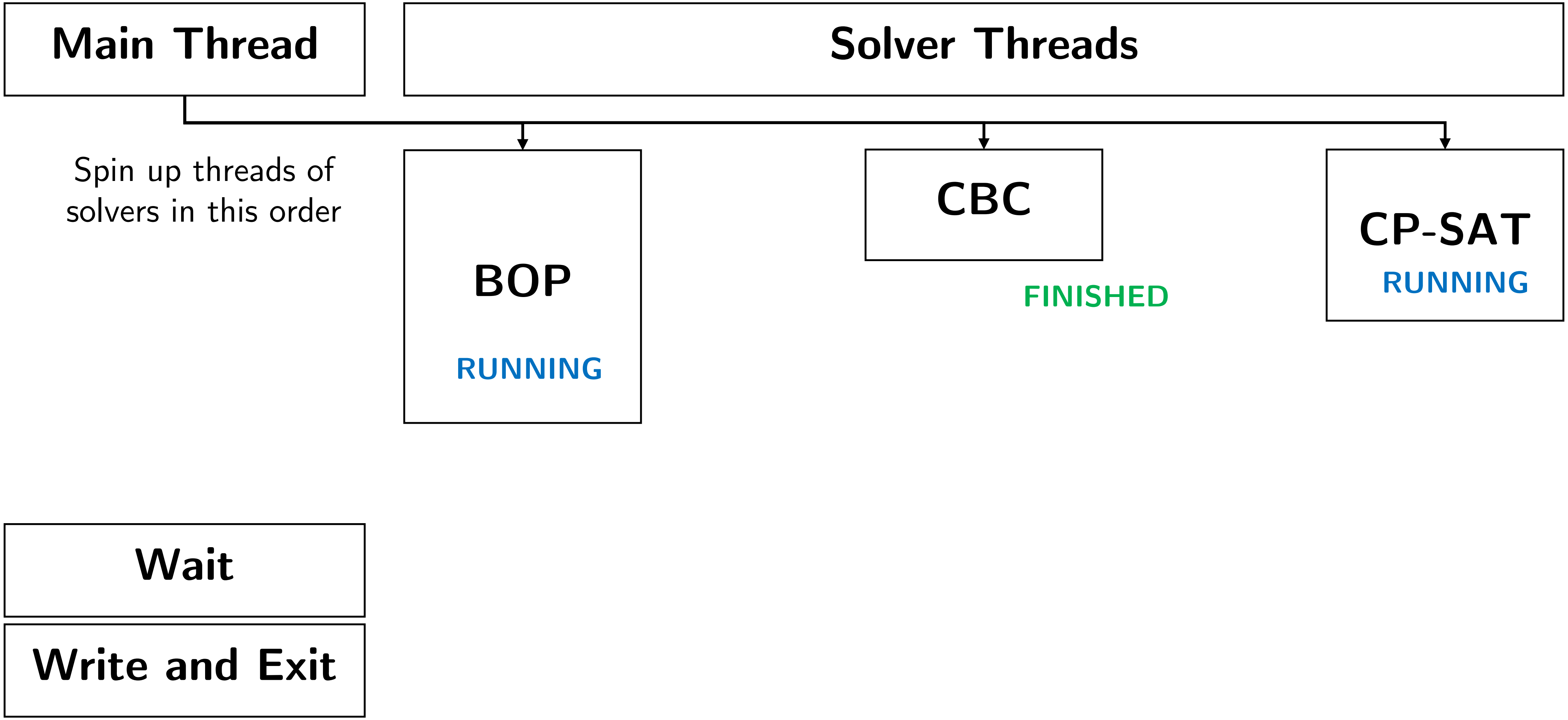
Why not use all?



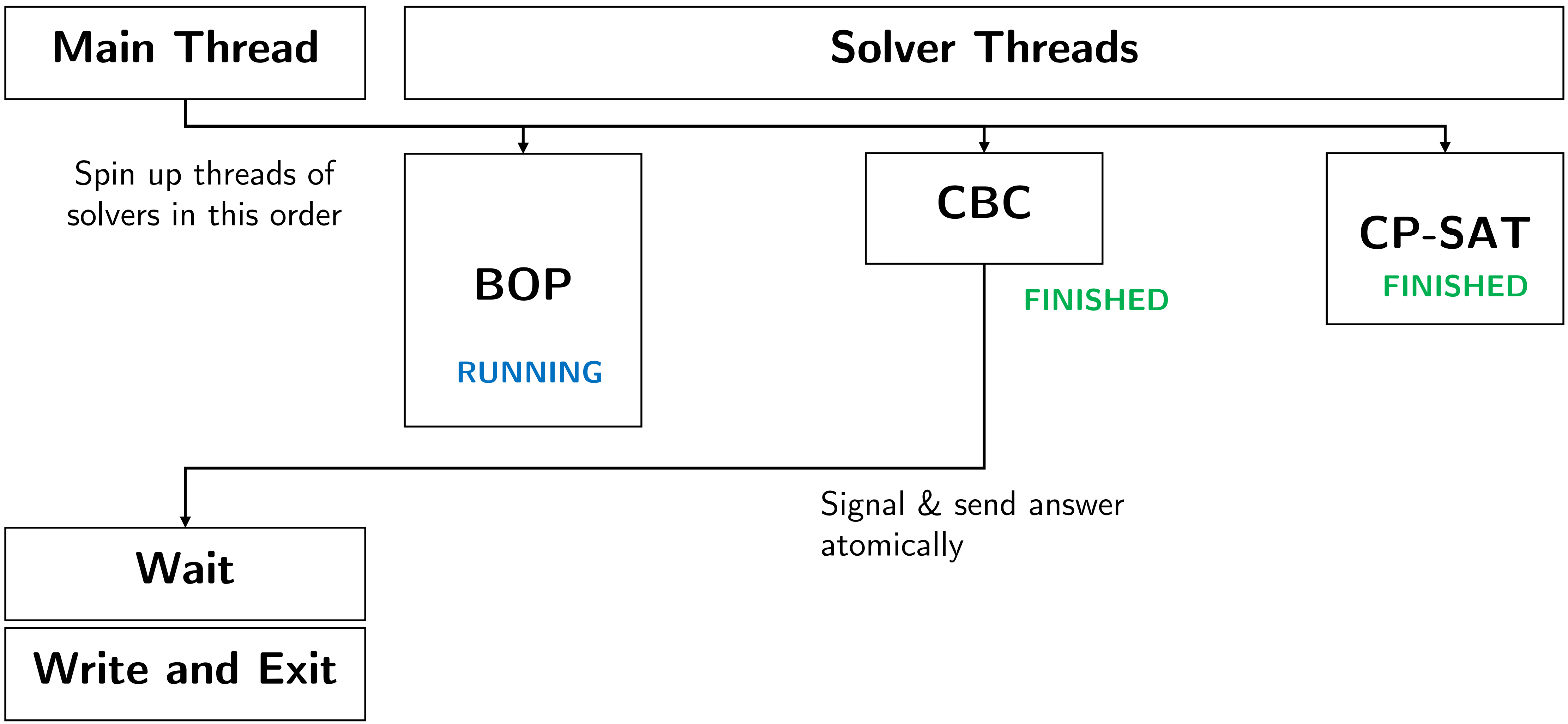
Why not use all?



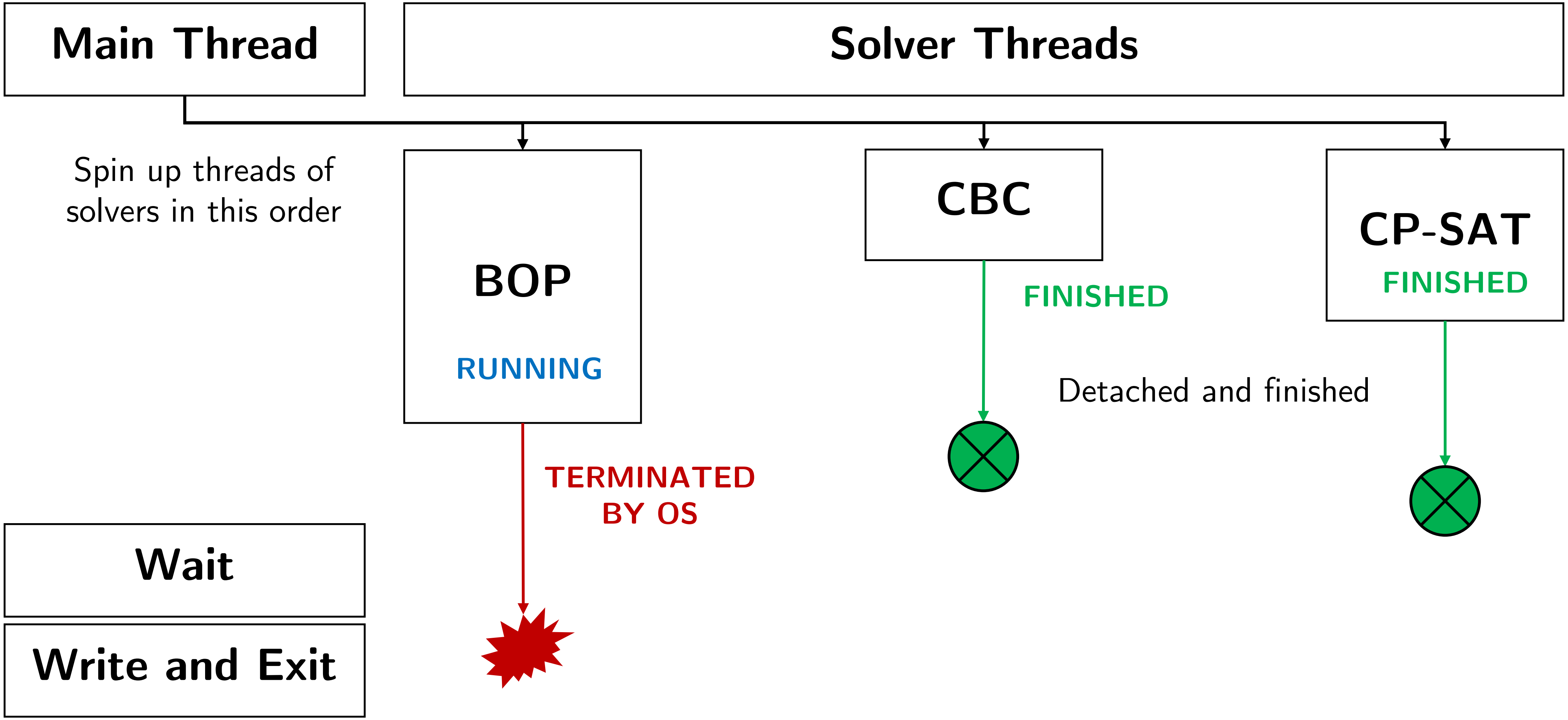
Why not use all?



Why not use all?



Why not use all?



C++ Compiler Choices

- GNU GCC
  - Standard
- LLVM Clang
  - A bit fancy (Output program is faster than GCC in some cases.)
- Intel oneAPI DPC++
  - A beast for specific hardware target

What about Docker?

Submission

You have to create a container image and publish it on a public repository (eg. gitlab or docker hub). Only submit the image repository to me. I (my bot) will pull it down and grade it for you.

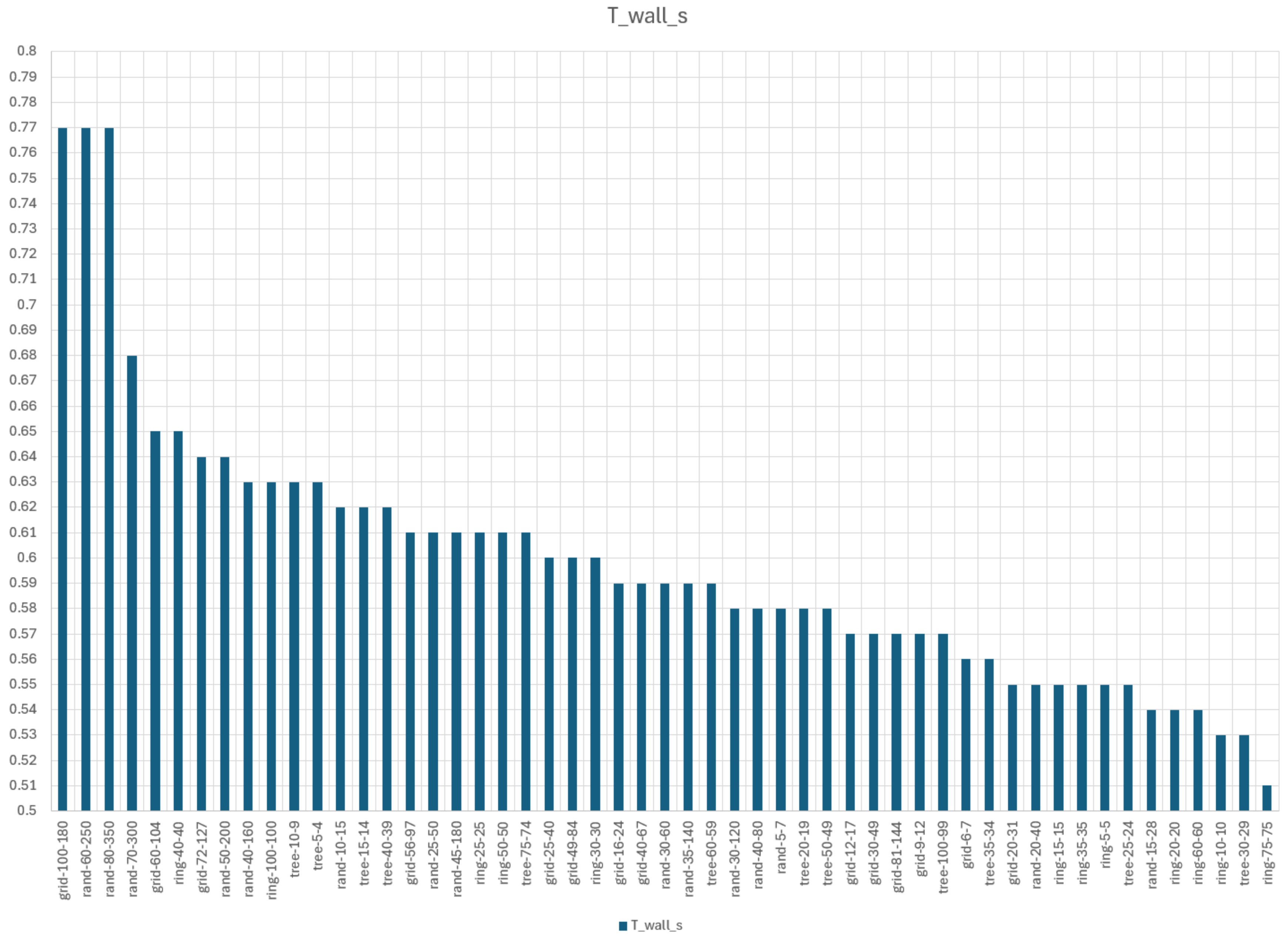
The command will be executed like this.

```
$ time docker run -v data/input:/input  -v data/output:/output [your docker image] /input/grid-6-7.txt /output/grid-6-7.out
```

What about Docker?

- `docker run` will take some time to spin up and tear down the image.
- More accurate measurement of the actual program should be made between the spinning up and tearing down process.
- What can reduce this overhead?
  - Use small base image
  - Want small output image too, how?
  - Multi-stage build
  - Linux shared objects linking

```
❯ docker image ls | grep neil4884
neil4884/mds                    latest              c9034ee462bf    41 hours ago    172MB
```


T_wall_s
0.8
0.79
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0.71
0.7
0.69
0.68
0.67
0.66
0.65
0.64
0.63
0.62
0.61
0.6
0.59
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
0.5
grid-100-180
rand-60-250
rand-80-350
rand-70-300
grid-60-104
ring-40-40
grid-72-127
rand-50-200
rand-40-160
ring-100-100
tree-10-9
tree-5-4
rand-10-15
tree-15-14
tree-40-39
grid-56-97
rand-25-50
rand-45-180
ring-25-25
ring-50-50
tree-75-74
grid-25-40
grid-49-84
ring-30-30
grid-16-24
grid-40-67
rand-30-60
rand-35-140
tree-60-59
rand-30-120
rand-40-80
rand-5-7
tree-20-19
tree-50-49
grid-12-17
grid-30-49
grid-81-144
grid-9-12
tree-100-99
grid-6-7
tree-35-34
grid-20-31
rand-20-40
ring-15-15
ring-35-35
ring-5-5
tree-25-24
rand-15-28
ring-20-20
ring-60-60
ring-10-10
tree-30-29
ring-75-75
T_wall_s

DISCLAIMER: UNOFFICIAL AND UNVERIFIED

Docker image	rand-30-60	ring-25-25	ring-15-15	ring-30-30	grid-25-40	grid-30-49	tree-30-29	tree-25-24	rand-30-12	tree-20-19
kanin49144/hpa_project	129.33	188.33	133.33	129.00	146.00	138.67	149.00	144.33	356.00	132.33
pnnnnnnnn/hpa-project:v1	2919.33	2543.33	86.00	45055.67	548.33	8272.00	91583.00	1651.00	213.00	176.67
natthaphons/factory-ortools	103.33	131.00	94.33	114.67	114.67	99.67	102.00	99.67	107.00	101.00
looknat/hpa	68.33	87.33	67.67	71.67	127.00	75.00	73.33	71.00	70.67	73.00
tnptw/hpa-powerplant	94.67	152.33	94.33	104.33	122.00	110.00	115.67	92.00	160.00	101.33
noppakorn/hpa-project:2024-04-19	410.00	530.33	412.67	438.33	414.67	432.00	434.67	404.67	419.33	410.33
nopparujp/hpa-proj:0.0.0	96.00	124.33	97.00	110.67	132.67	131.33	102.00	118.33	183.67	95.67
nanthicha/hpa-project	251.33	307.33	248.33	242.33	247.33	245.33	239.67	237.00	449.33	249.33
rynparin/powerplant	100.00	128.67	100.00	106.00	125.33	130.00	102.33	105.00	178.33	96.00
taecv/dominating-set	249.33	311.00	240.00	242.00	248.33	244.00	243.67	240.33	456.00	248.67
ghcr.io/nowarm/hpa-final:latest	132.67	137.67	120.00	144.67	133.33	134.67	139.67	133.33	344.33	143.00
nichapanit/finalhighperf	89.00	106.33	91.67	92.33	107.67	107.33	95.33	92.00	161.33	86.00
pacharaponarp/high_perf_powerplant	98.00	121.00	100.67	95.00	112.00	101.33	105.33	99.67	106.00	92.67
pattanan/high_perf_project	103.33	125.00	100.33	107.67	107.33	105.33	105.67	101.67	108.00	96.33
namolert/high-perf	1060.33	940.33	598.00	1052.00	905.67	1049.67	1051.33	900.00	1238.33	750.33
karnjj/power_plant	114.67	119.00	98.33	240.33	101.33	127.33	325.67	114.33	103.00	100.00
khunanondock/vertex-cover	122.00	207.00	77.00	1496.33	96.33	251.67	3314.33	117.67	73.33	78.67
asiaseek/power_plant	102.67	104.33	133.00	113.67	134.33	133.00	117.33	110.33	199.33	145.33
ghcr.io/miello/high-perf-project:v1.0.2-python	136.33	137.00	121.00	157.67	147.67	155.33	167.00	139.67	339.00	153.00
cdgstudent/power-plant-problem	327.33	319.00	331.67	341.00	336.00	319.00	340.33	310.33	373.33	299.00
gri11/hpc-project	85.33	103.67	85.67	93.67	117.67	110.00	87.00	90.33	168.33	88.00
meensan/highperf:fast	76.33	75.00	77.67	77.00	84.33	84.00	74.00	82.67	261.67	86.00
byte101/powerplant	130.33	170.00	151.67	138.00	156.67	186.00	170.67	145.67	215.00	133.67
peammy1146/highpref:latest	86.33	86.33	77.67	78.67	85.33	82.33	73.33	84.33	271.67	84.00
jayjacka/hpa-project	284.67	292.33	274.67	288.33	264.00	283.00	274.67	292.33	262.67	292.33
takriz/hpa-project	109.00	119.33	96.00	104.33	111.00	108.00	103.00	110.00	293.33	106.33
lectroz/hpa-final-project	83.33	129.67	88.00	135.67	89.00	101.33	116.33	94.00	83.67	90.33
jirayuwat12/hpa_my_optimal_solution	107.33	122.33	103.00	102.33	109.67	99.67	108.00	100.67	287.00	96.33
vv1n/high-perf-project	141.33	150.00	144.33	141.67	136.00	134.33	148.67	145.67	341.00	141.67
spzbmp/powerplant_solver	378.00	405.00	404.67	402.00	388.00	413.67	405.00	387.00	383.00	412.00
thamph/mds-solver	80.33	88.67	84.33	83.00	82.00	85.33	82.67	87.00	83.33	81.33
masternonnolnw/hpa-parallel	104.00	106.00	100.67	101.00	122.33	120.33	96.33	96.33	168.67	93.00
differentail/hpa_final	337.00	345.67	325.67	317.33	323.00	351.00	320.00	320.00	450.00	320.33
neil4884/mds	79.00	93.33	81.00	83.67	89.33	87.67	80.33	82.33	84.33	98.00
preamza02/prame_high_perp_2	98.00	105.00	92.33	90.67	127.67	153.67	104.33	100.33	178.00	103.00
MEDIAN	**107.33**	**129.67**	**100.33**	**114.67**	**127.00**	**131.33**	**116.33**	**110.33**	**213.00**	**101.33**
MINIMUM	**68.33**	**75.00**	**67.67**	**71.67**	**82.00**	**75.00**	**73.33**	**71.00**	**70.67**	**73.00**

DISCLAIMER: UNOFFICIAL AND UNVERIFIED

Source code

- Docker image: neil4884/mds
- GitHub repository: vtneil/mds