Source Code — http://yedeheng.github.io/bitgpu/

GPU-Accelerated High-Level Synthesis for Bitwidth Optimization of FPGA Datapaths

Nachiket Kapre, Ye Deheng
nachiket@ieee.org

Tool support — XILINX ALL PROGRAMMABLE. NVIDIA LIR

Claim

Claim

- GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi “brute-force” evaluation

Claim

- GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi “brute-force” evaluation
- DUMB + clever approach

Claim

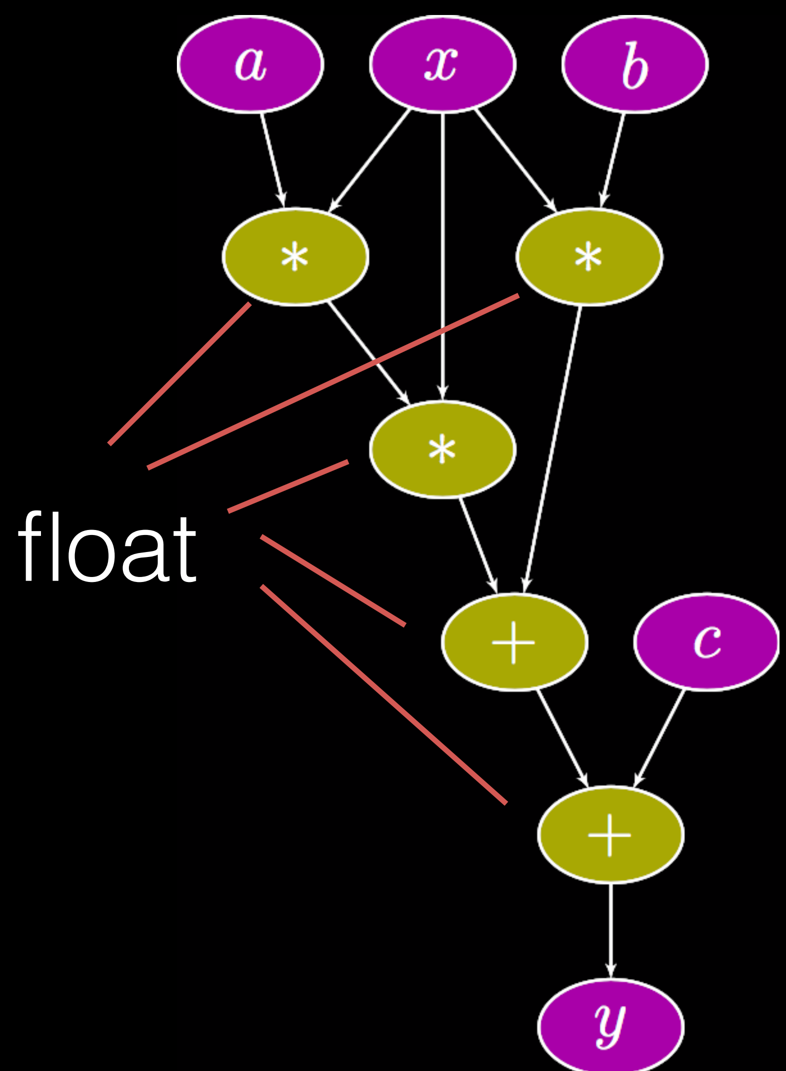
- GPUs can help us accelerate FPGA CAD
  — specifically, bitwidth optimization
  — reformulated as semi "brute-force" evaluation

- DUMB + clever approach

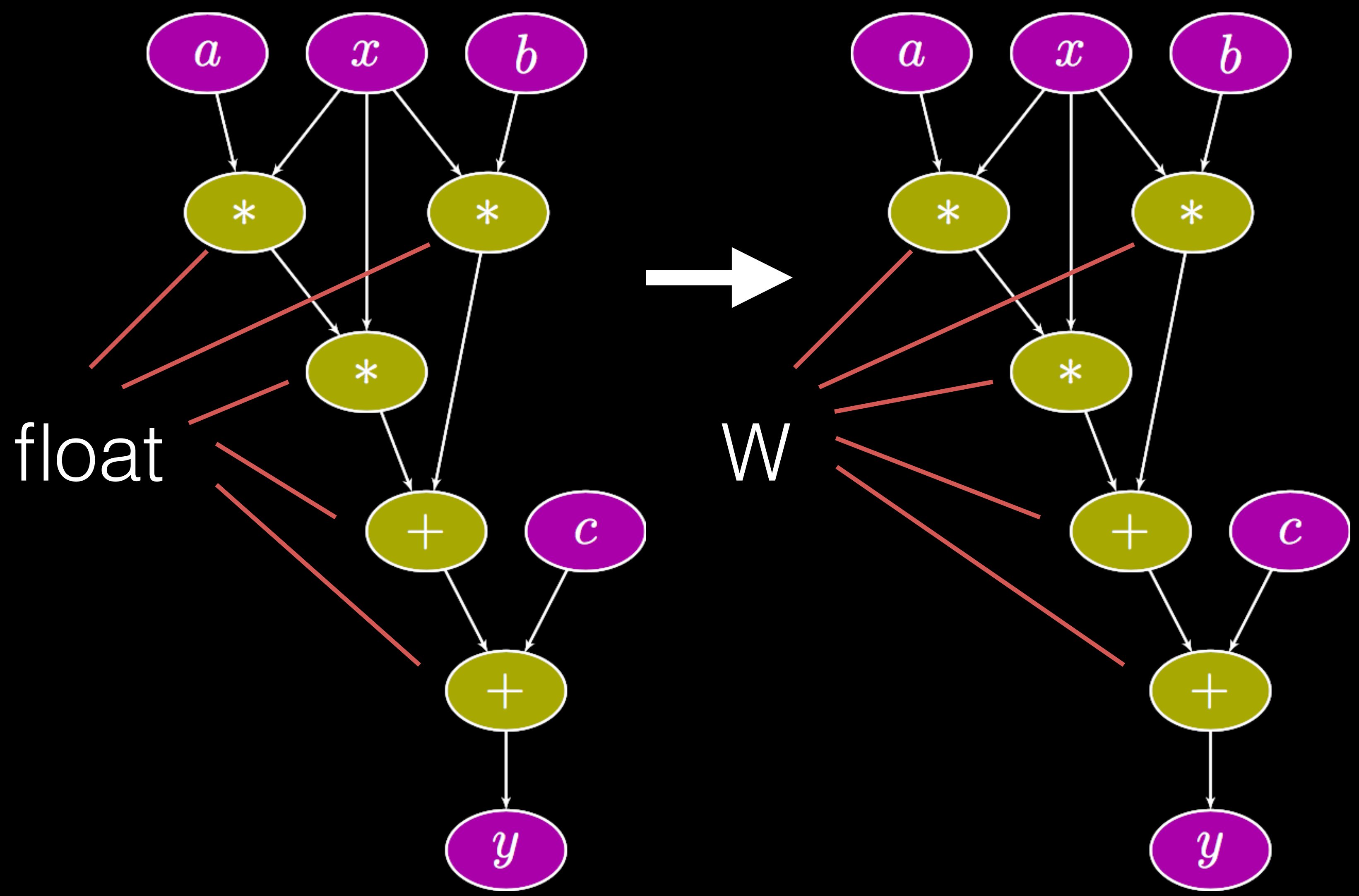
- **Idea**:
  (1) Use CPU heuristic to narrow search space,
  (2) deploy GPUs when single-shot feasible.

Why Bitwidth Optimisation?

Why Bitwidth Optimisation?

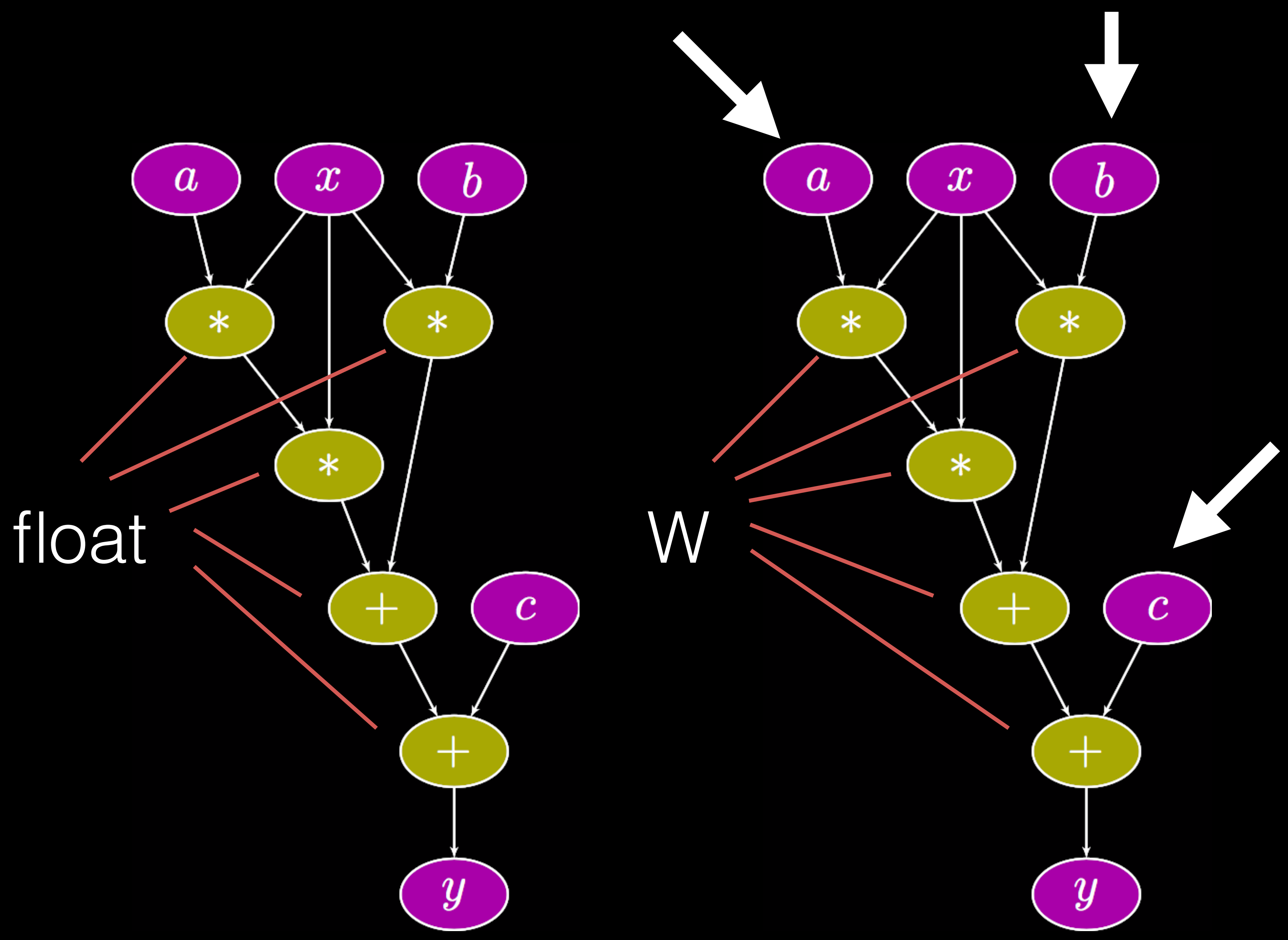


Why Bitwidth Optimisation?



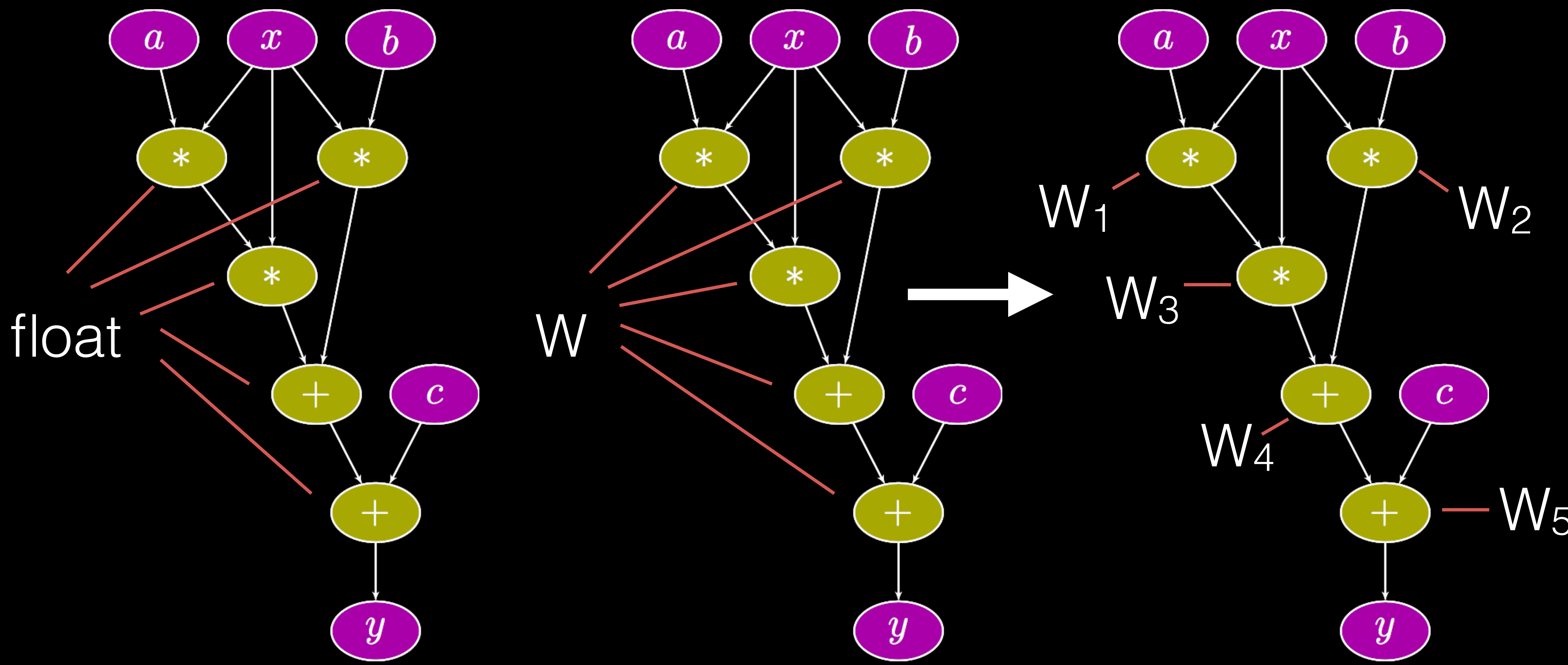
[FCCM 2012] FX-SCORE — **70% less area**

Why Bitwidth Optimisation?



[FCCM 2013] — **4x less area**

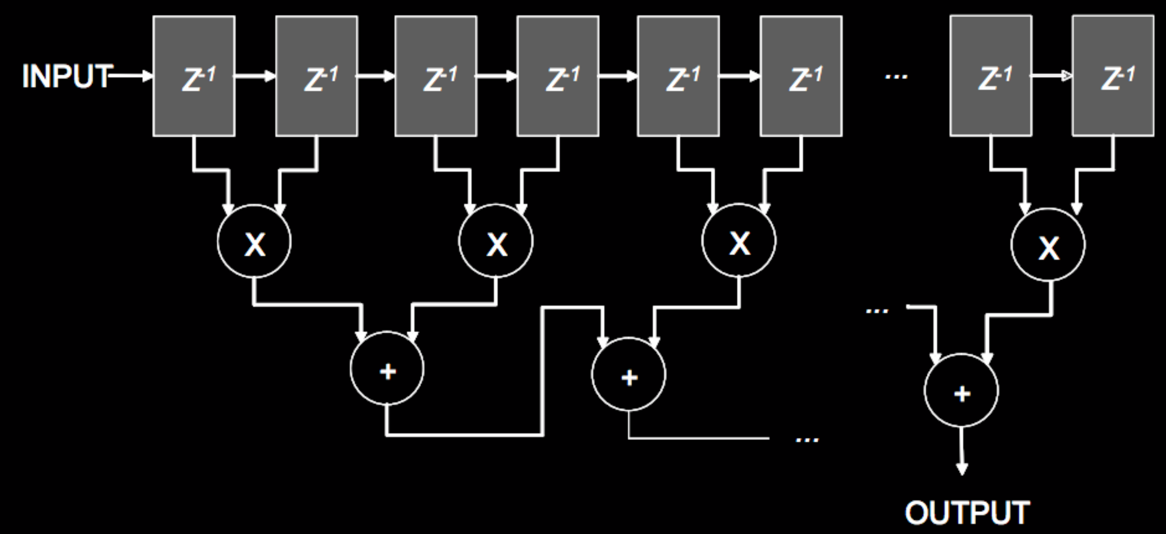
Why Bitwidth Optimisation?



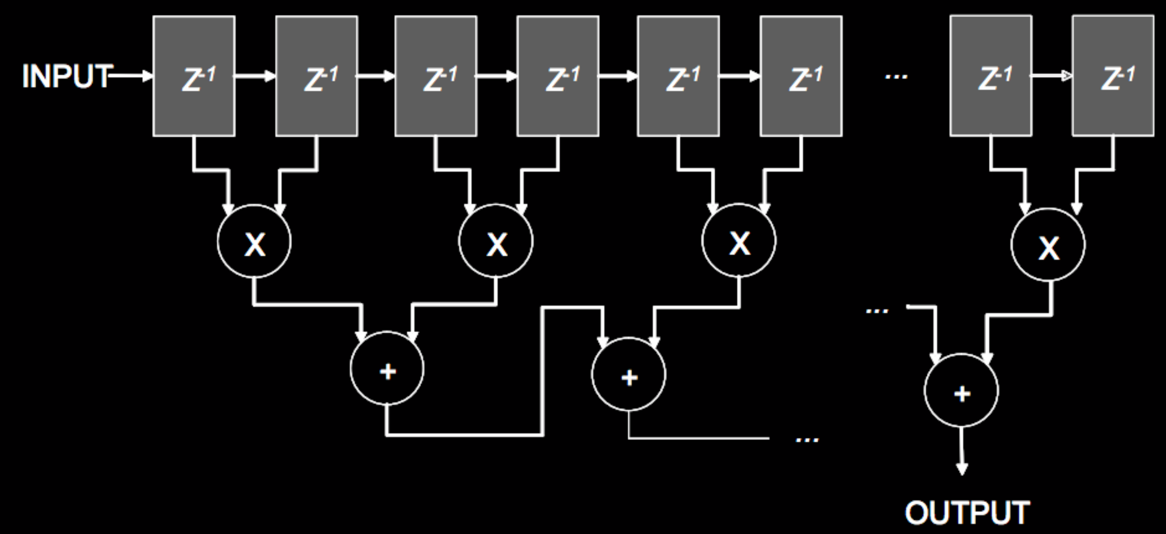
[FCCM 2014] Mix-FXSCORE— **43% less area**

But,… slow runtime!

But,… slow runtime!

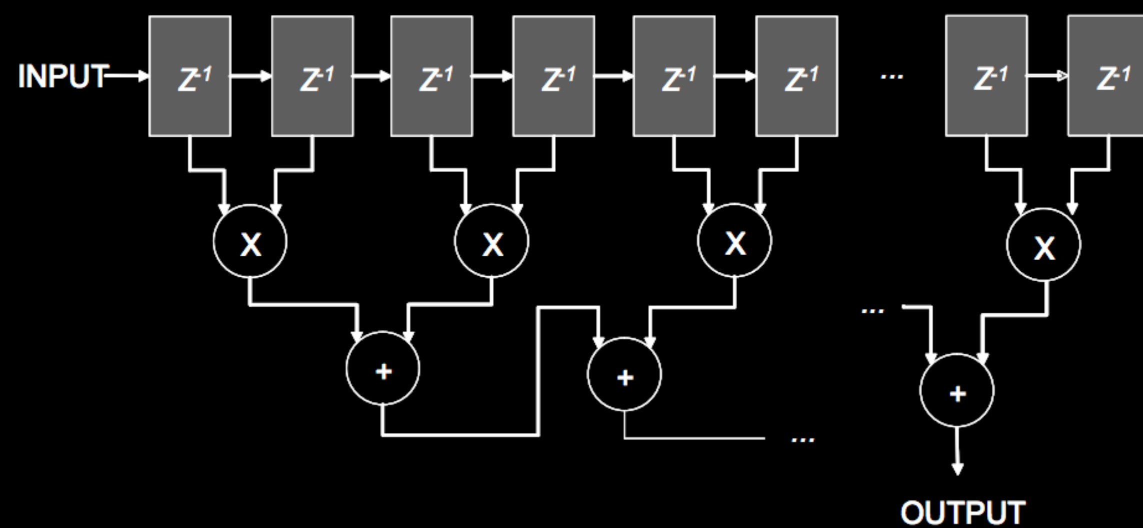


But,… slow runtime!



- Typical bitwidth optimization challenge:
  — 1K-tap FIR filter design
  — simulate various conditions ~1e^5 test vector
  — 40 days of CPU runtime

But,… slow runtime!



- Typical bitwidth optimization challenge:
  — 1K-tap FIR filter design
  — simulate various conditions ~1e^5 test vector
  — 40 days of CPU runtime

G. Caffarena and D. Menard. Many-core parallelization of fixed-point optimization of VLSI circuits through GPU devices. DASIP 2012

Outline

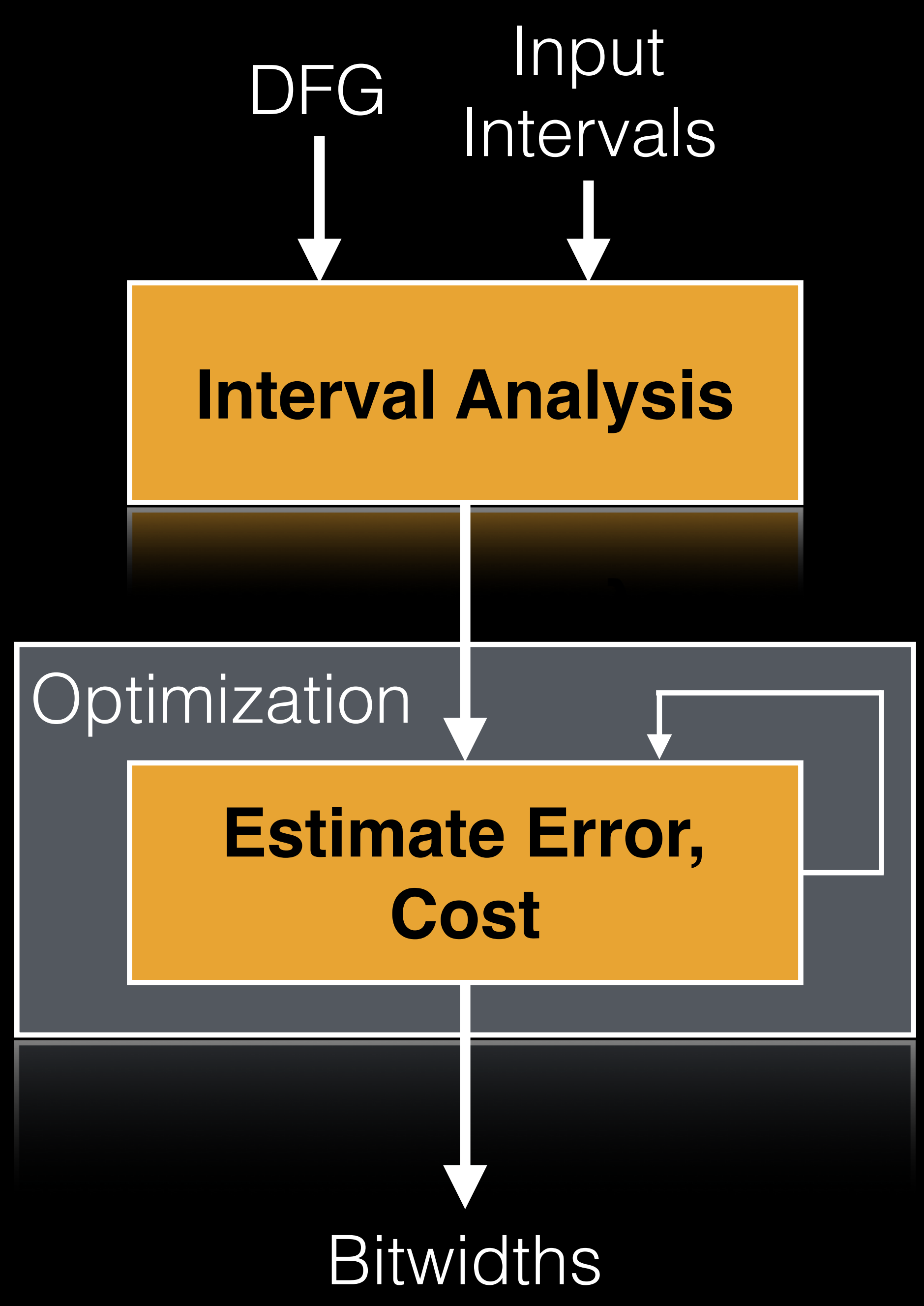
- Bitwidth Optimization Review
  — Parallelism outlook
- GPU Parallelisation
  — CPU-assisted pruning
- Quick-and-dirty HLS
- Experimental Validation
- Outlook

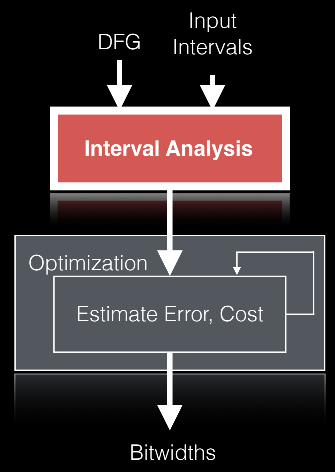
Bitwidth Optimization
Parallel Potential

Outline of algorithm

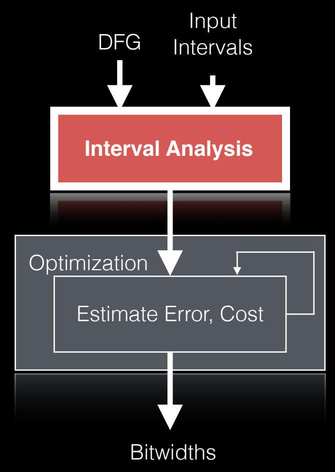
- 1. **Interval Analysis**
  — Identify range [min,max] per variable
  — One-time pre-processing

- 2. **Iterative optimization**
  — a. Assign precisions to variables
  — b. Estimate error based on interval and precision
  — c. Optimize/Loop until error meets threshold



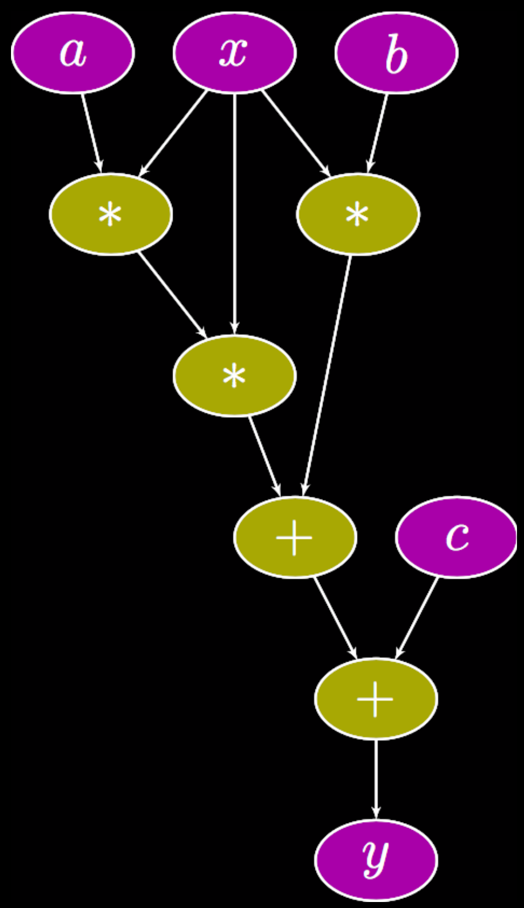


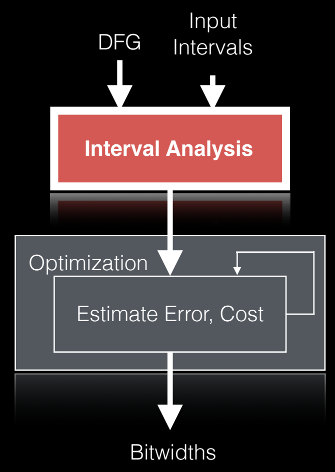
Interval Analysis



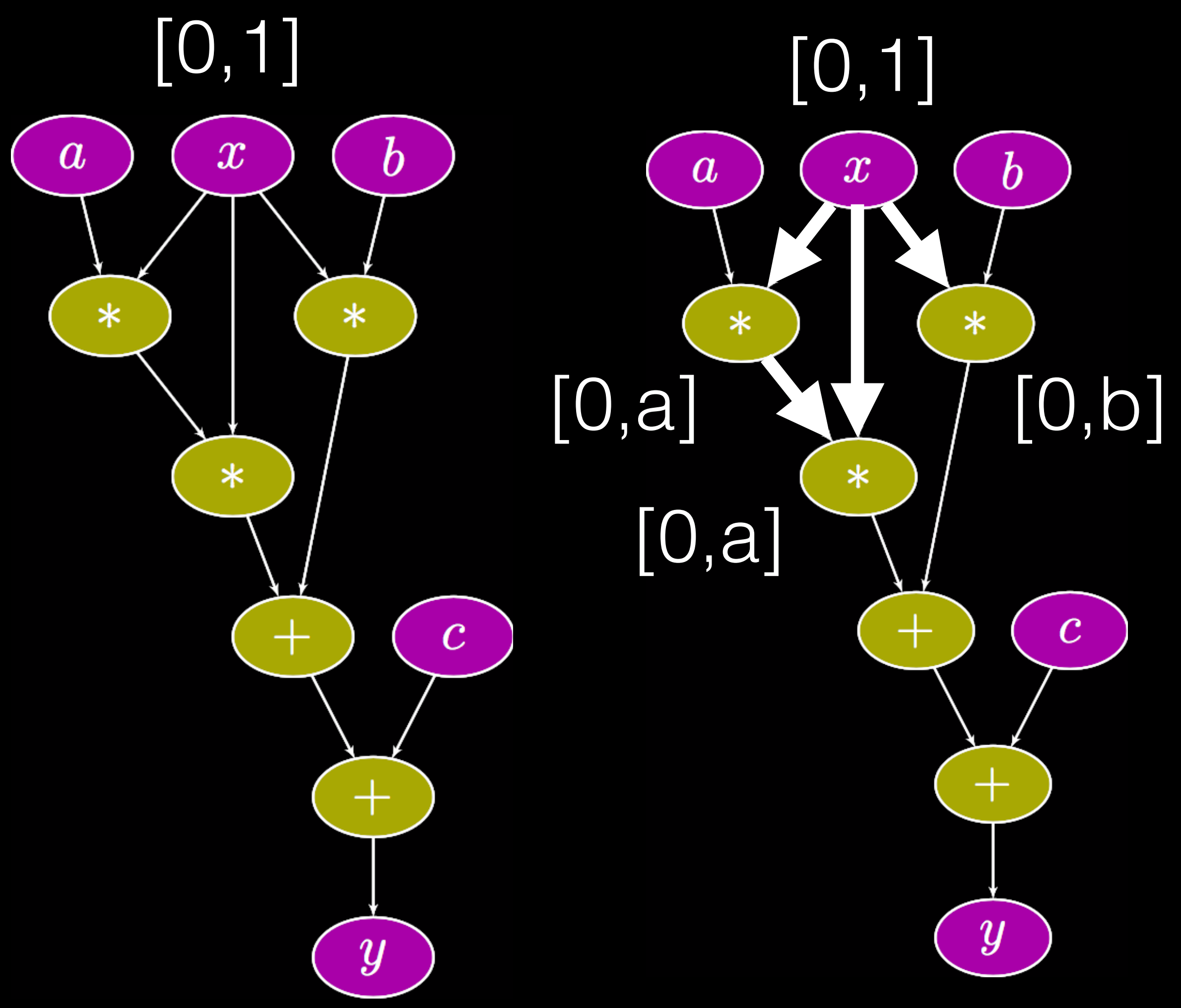
Interval Analysis

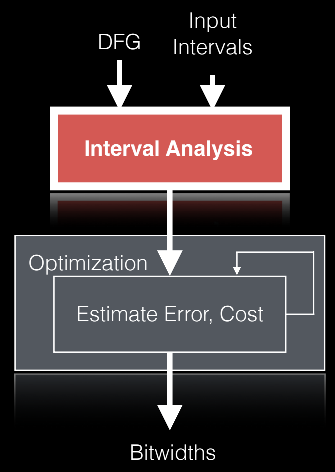
[0,1]



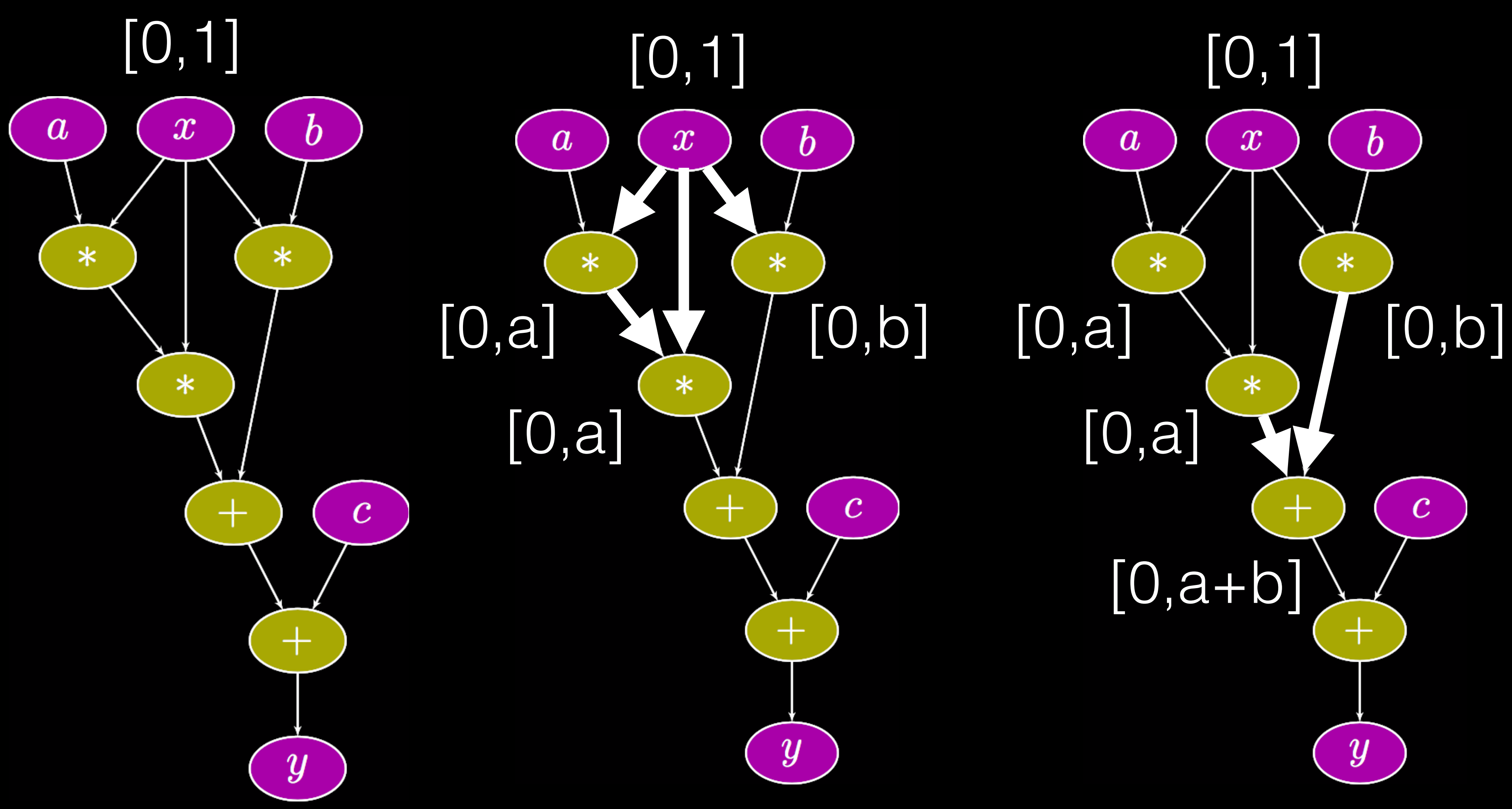


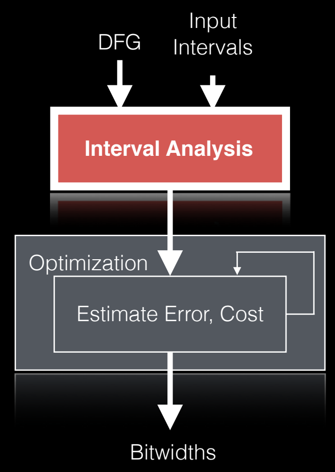
Interval Analysis



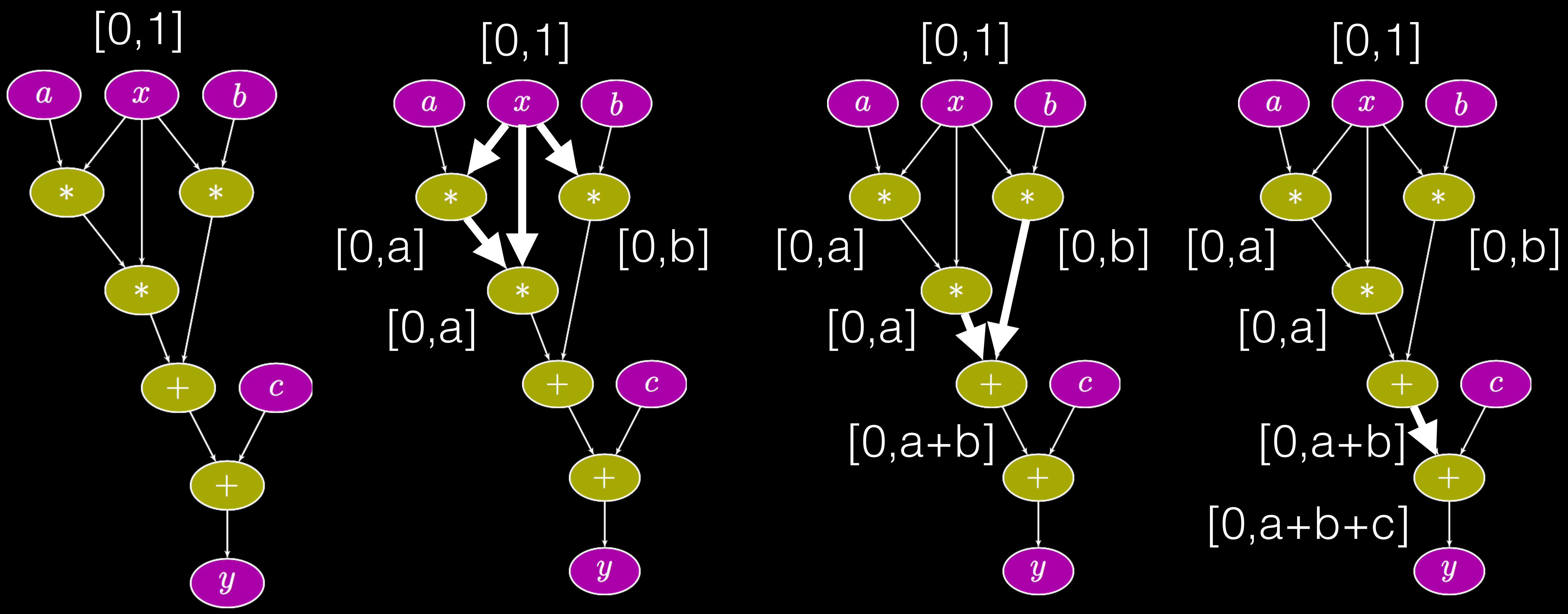


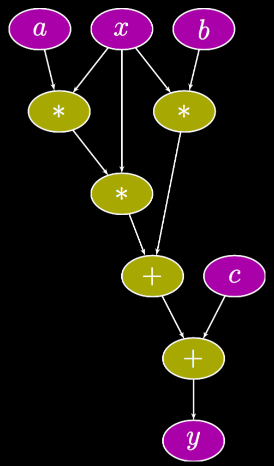
Interval Analysis



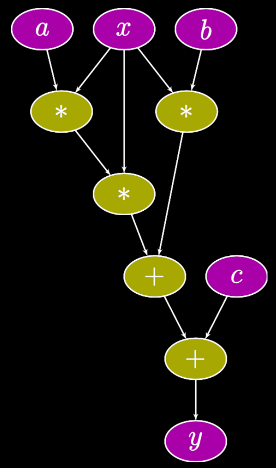


Interval Analysis



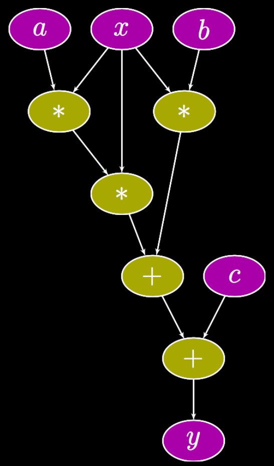


Interval Analysis (IA)



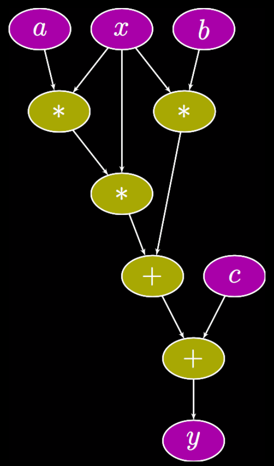
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG



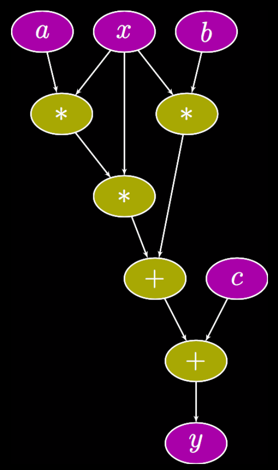
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds



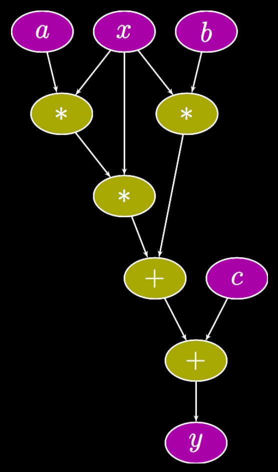
Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds
- Loose bounds — larger-than-reqd bitwidths



Interval Analysis (IA)

- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds
- Loose bounds — larger-than-reqd bitwidths
- CPUs — *affine arithmetic (AA)*, include correlations between inputs — explosion in error terms!

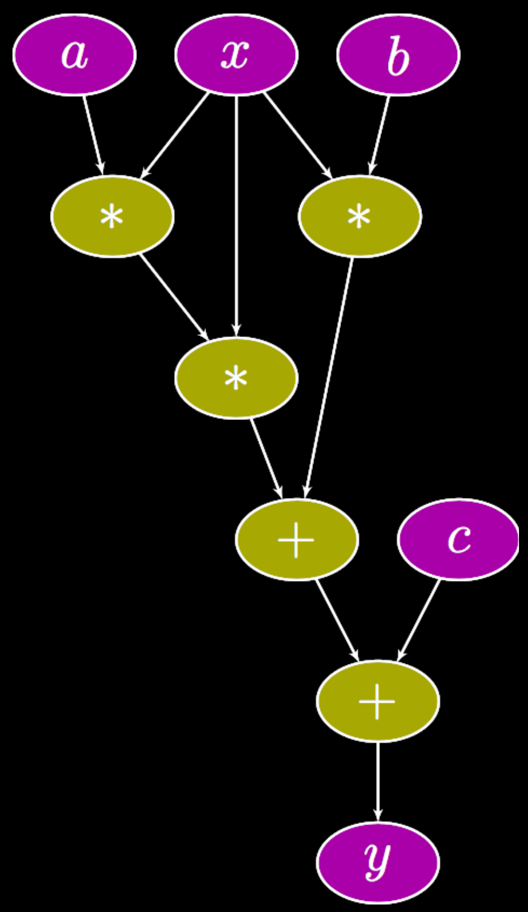


Interval Analysis (IA)

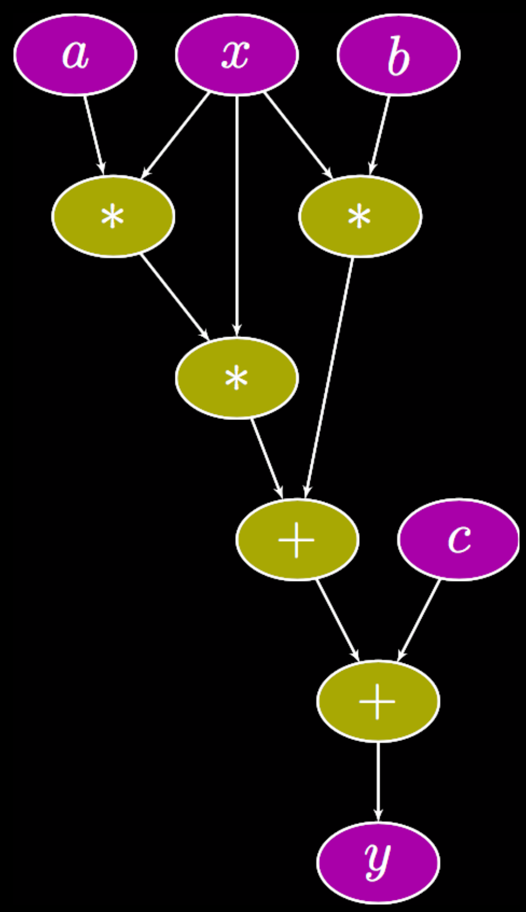
- Propagate the intervals step-by-step through DFG
- IA Pessimistic — does not generate tight bounds
- Loose bounds — larger-than-reqd bitwidths
- CPUs — *affine arithmetic (AA)*, include correlations between inputs — explosion in error terms!
- GPUs — *sub-interval analysis (Sub-IA)*

Sub-Interval Analysis

[0,1/2]

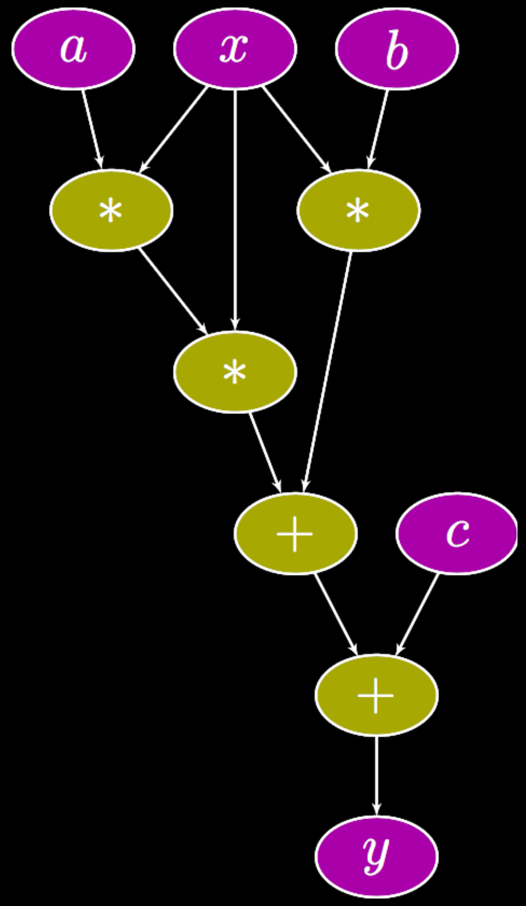


[1/2,1]

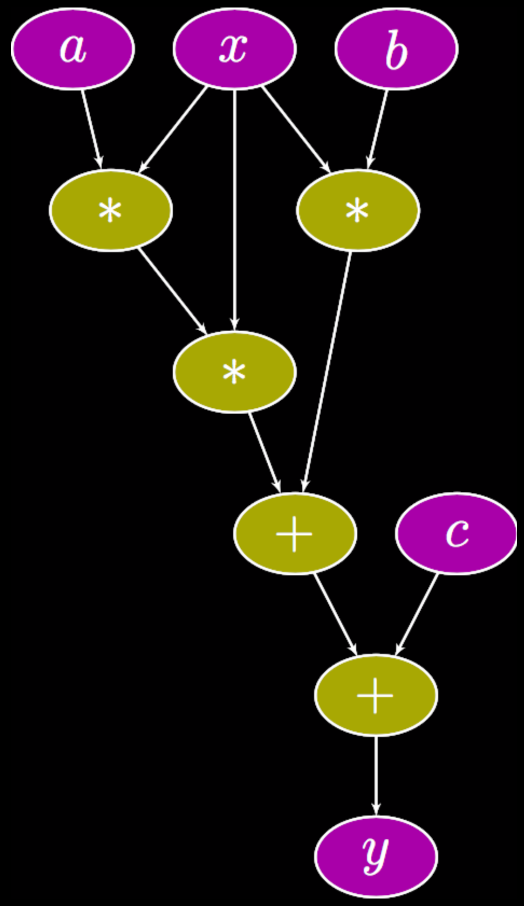


Sub-Interval Analysis

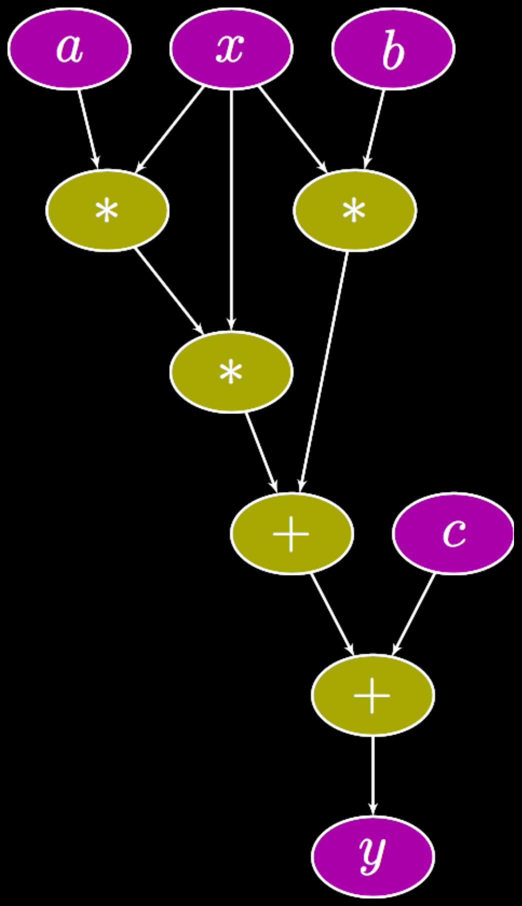
[0,1/4]



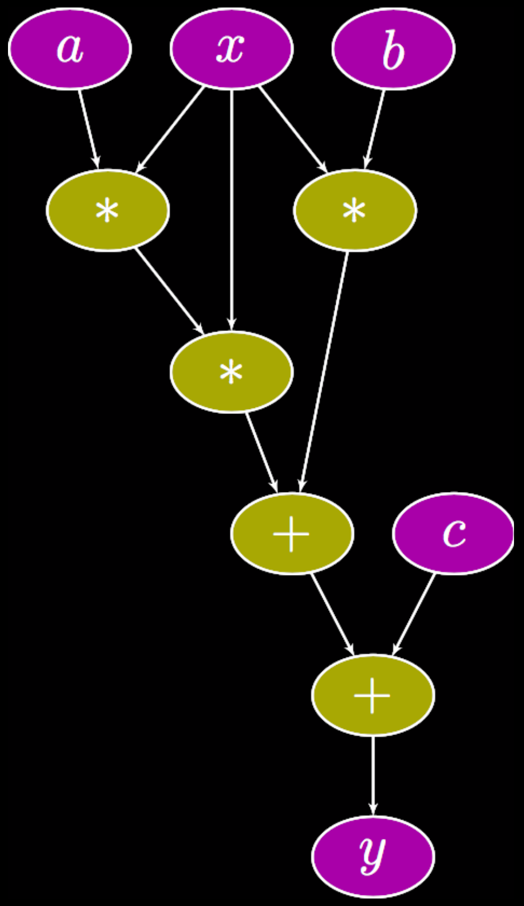
[1/4,1/2]



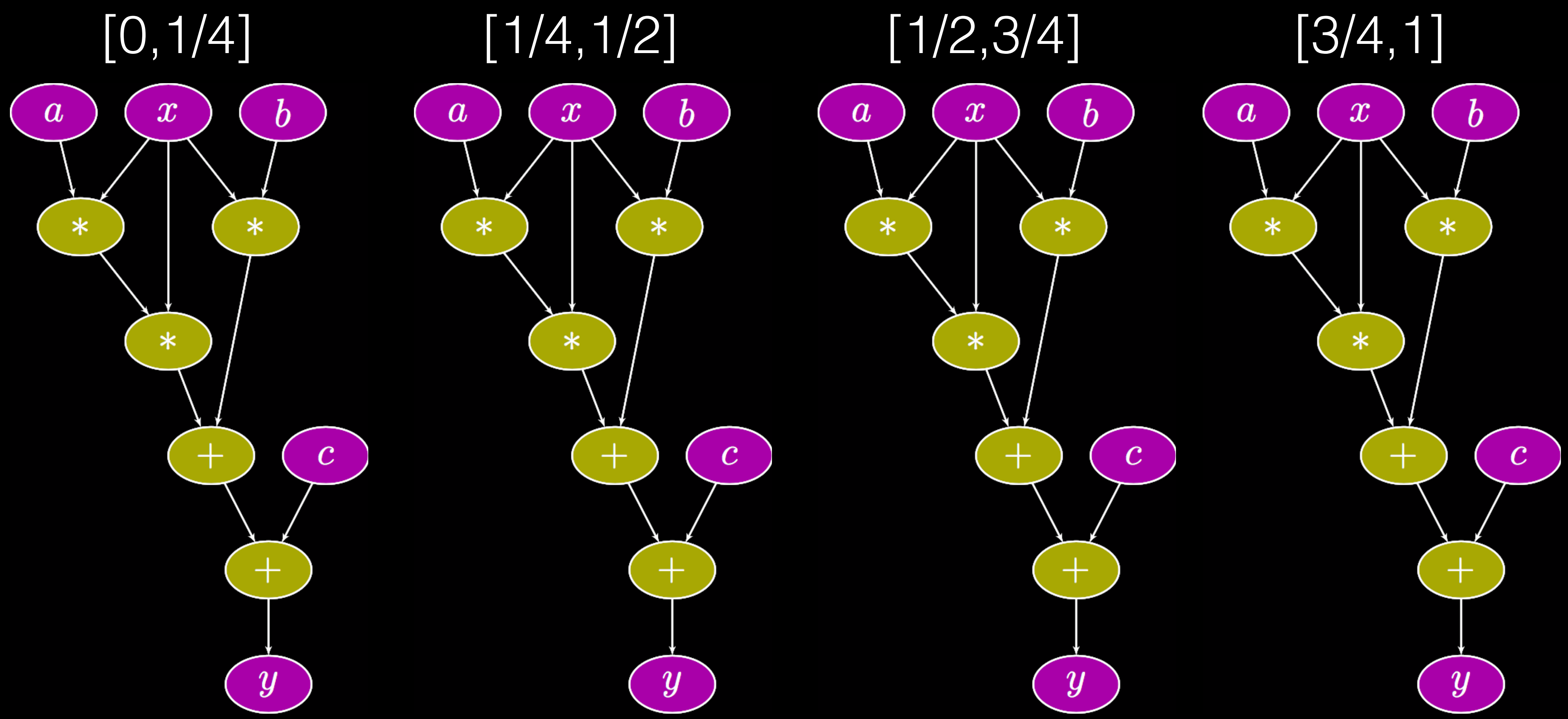
[1/2,3/4]



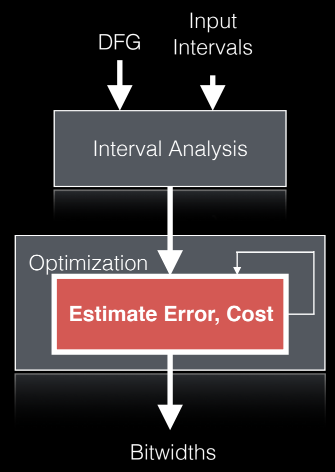
[3/4,1]



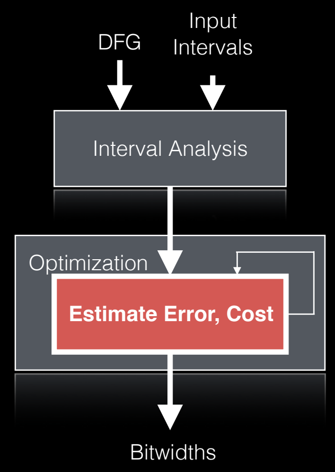
Sub-Interval Analysis



(1) Tighter bounds than plain IA
(2) Exponential threads with splits
(3) Completely parallel!

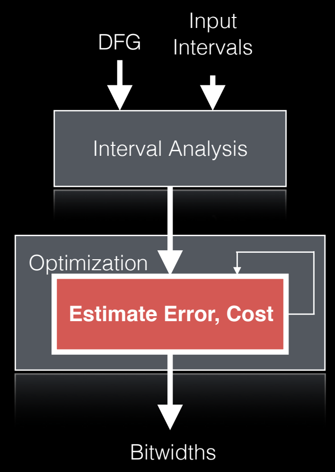


Error+Cost models



Error+Cost models

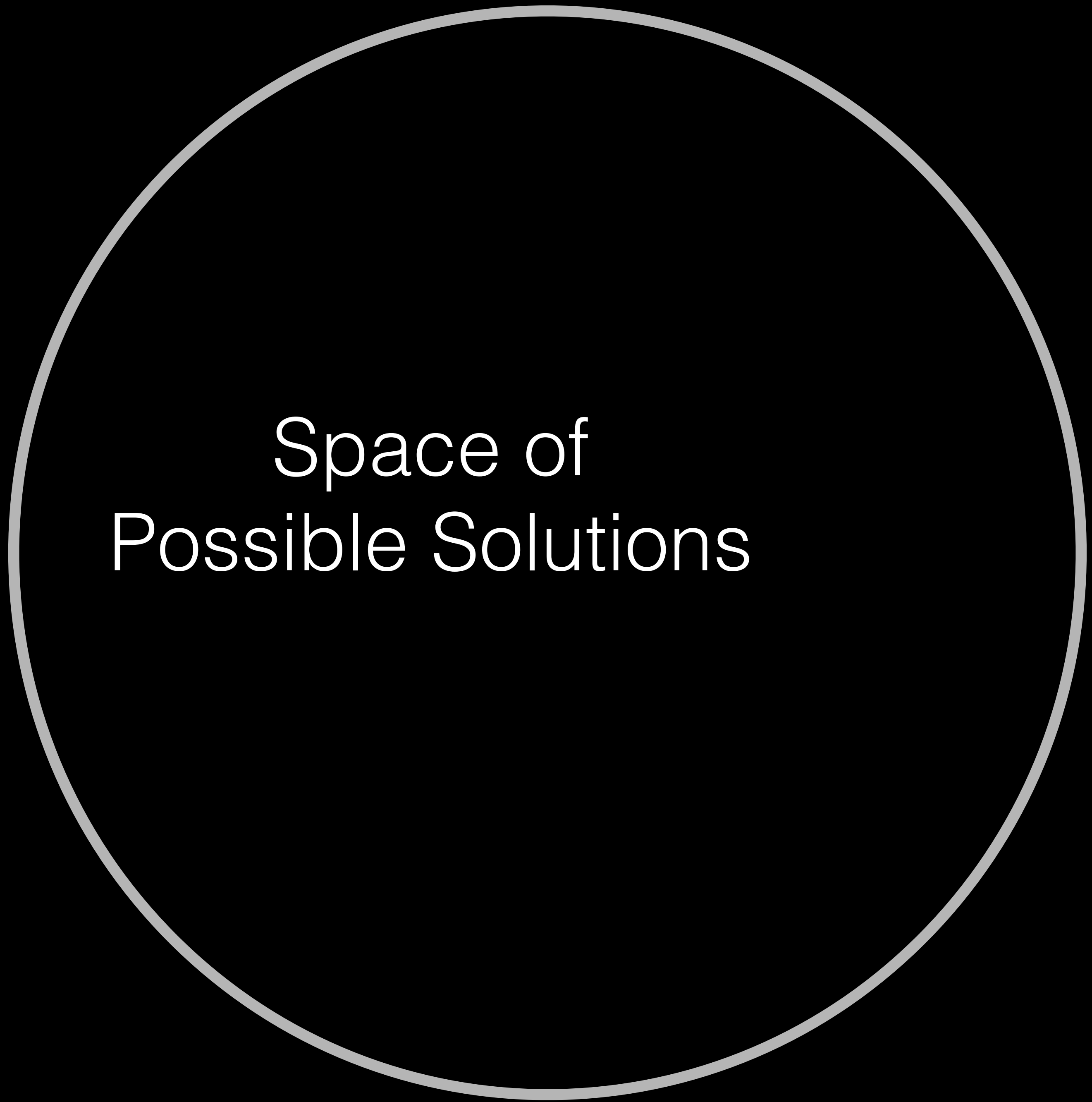
- Iterative optimization — searches through multiple candidates, refines/improves solution



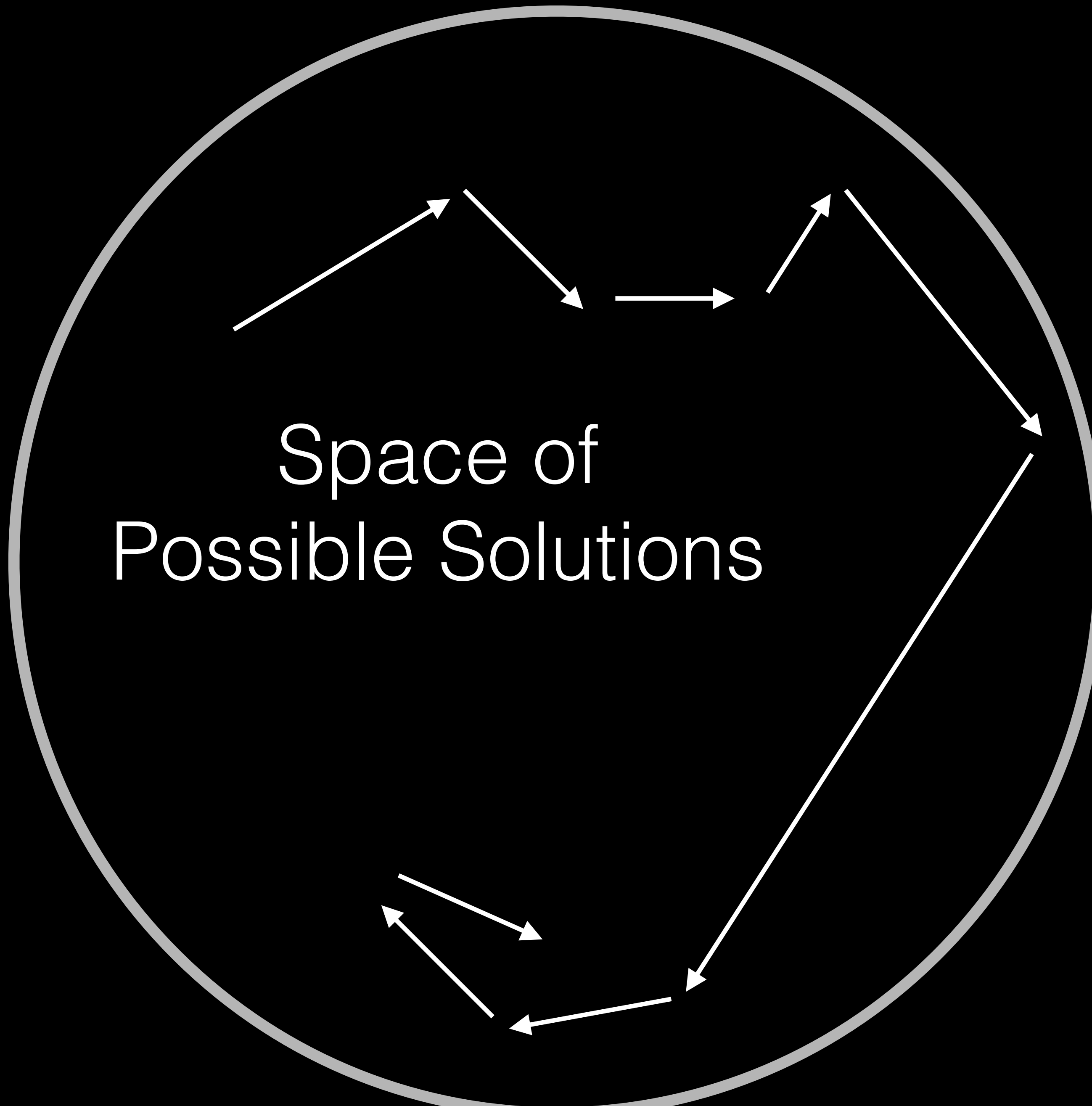
Error+Cost models

- Iterative optimization — searches through multiple candidates, refines/improves solution
- At each evaluation, must calculate
  — choose bitwidth assignments/variable
  — relative/absolute error at each variable/output
  — physical cost of mapping to FPGA

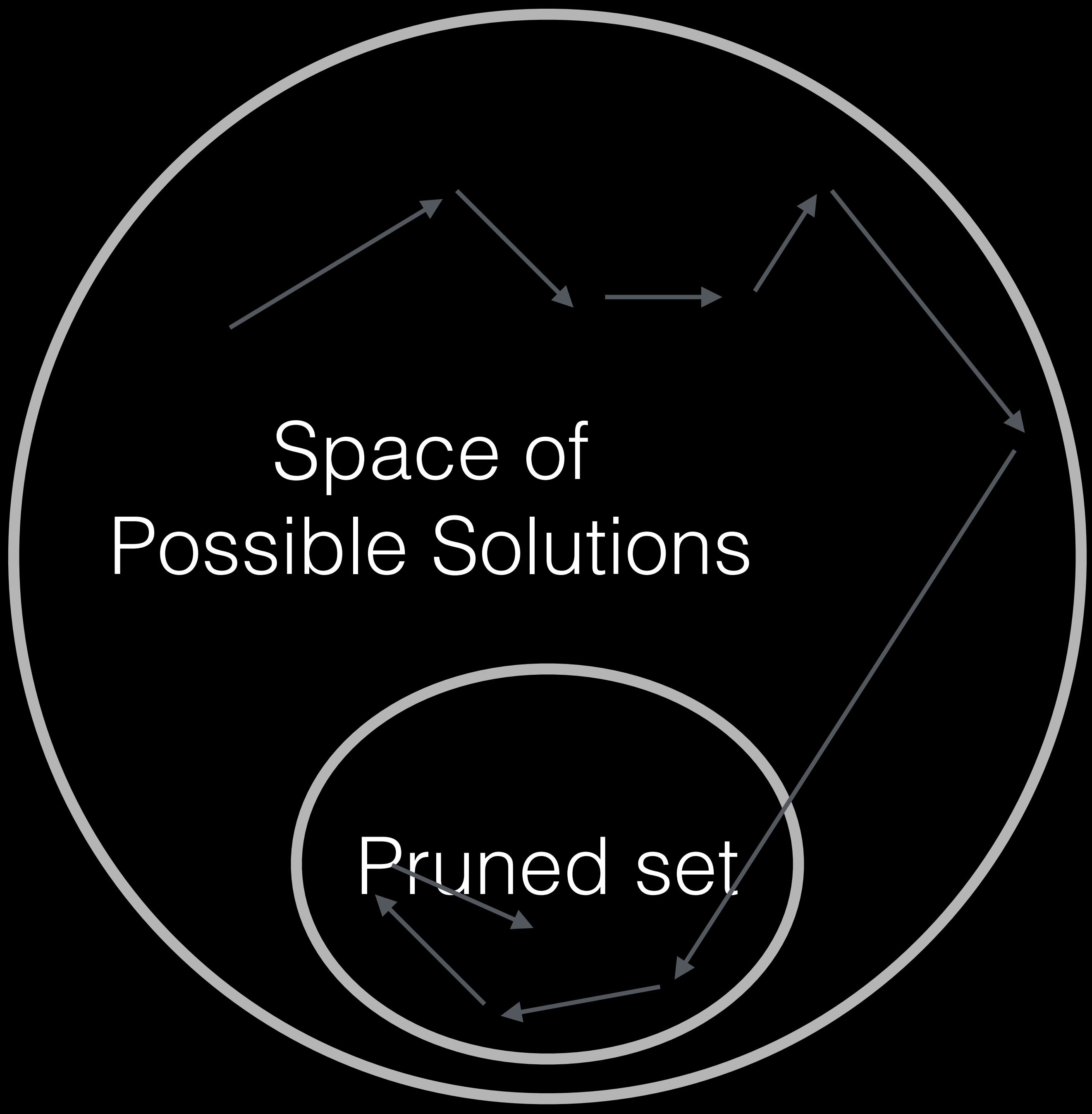
Idea



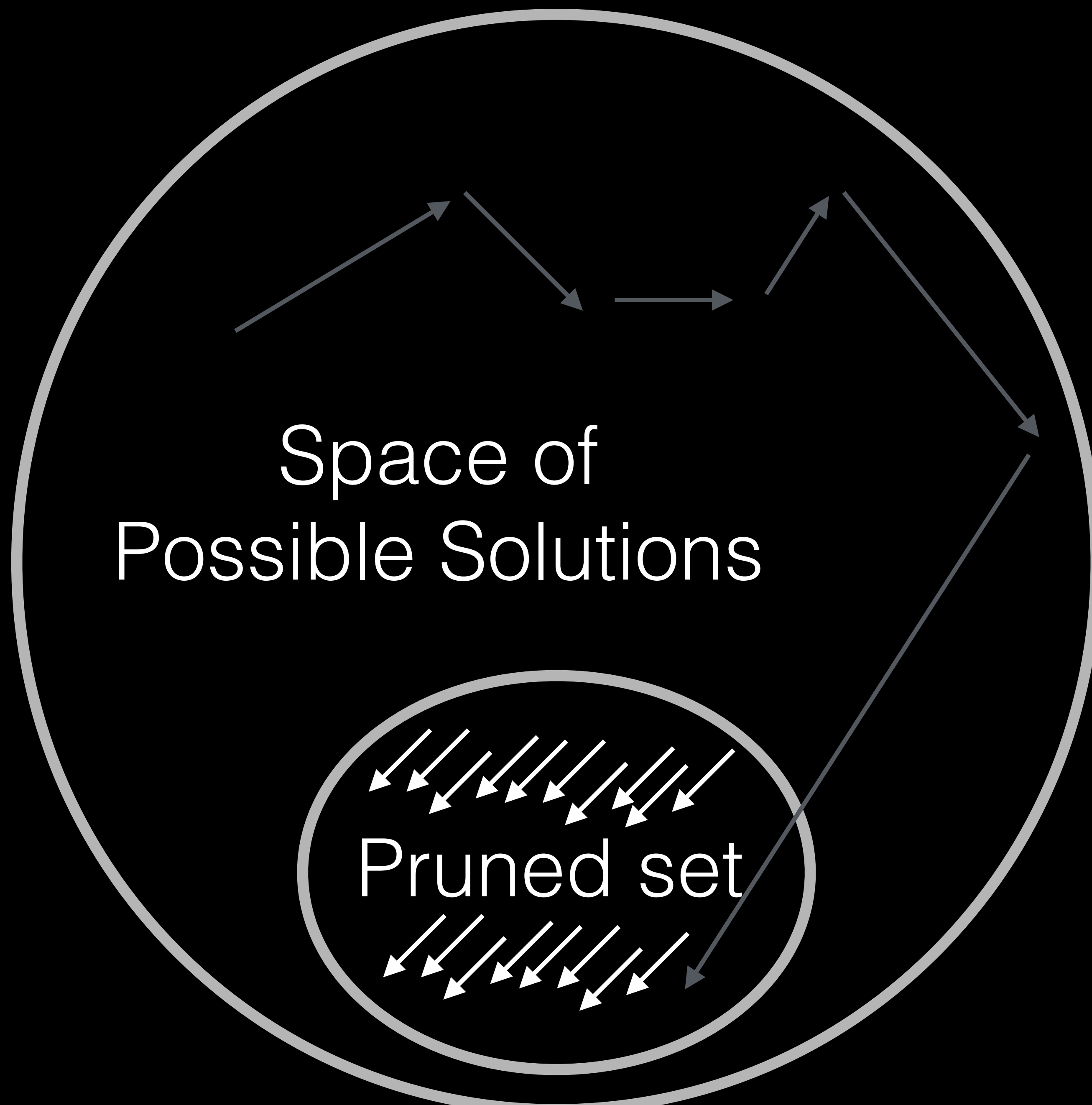
Idea



Idea

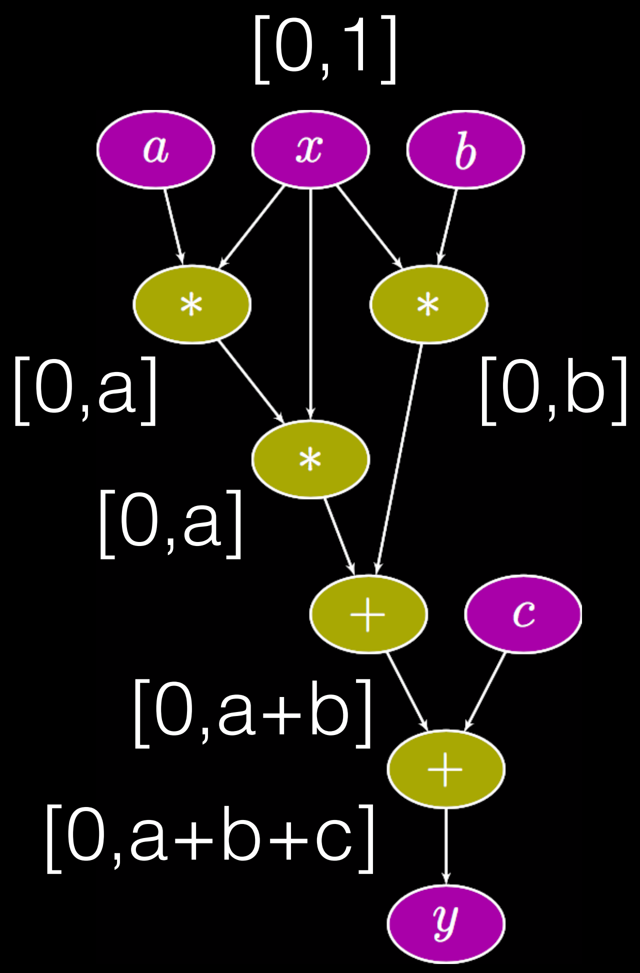


Idea



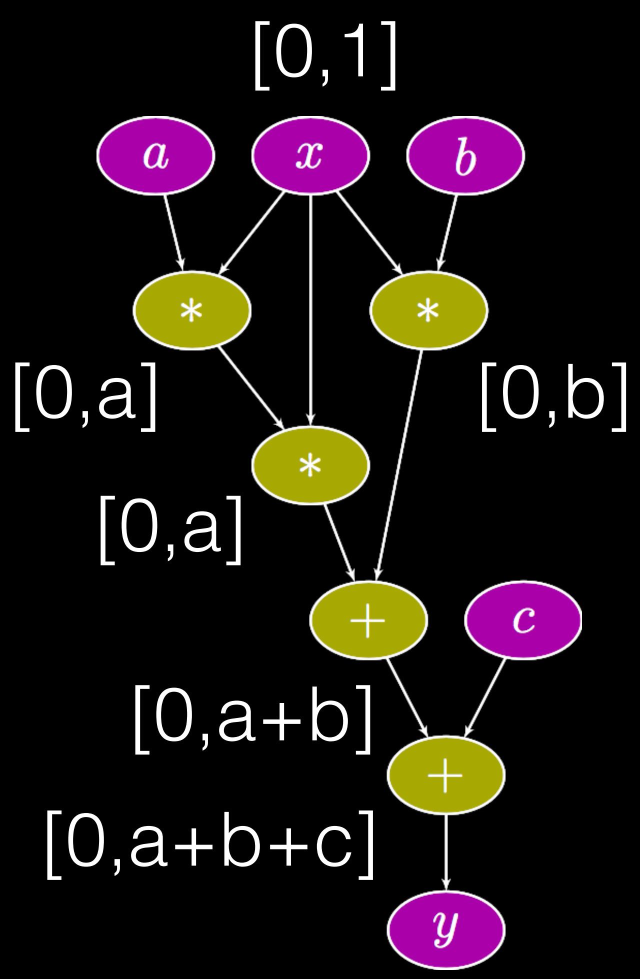
Error Propagation

Error Propagation



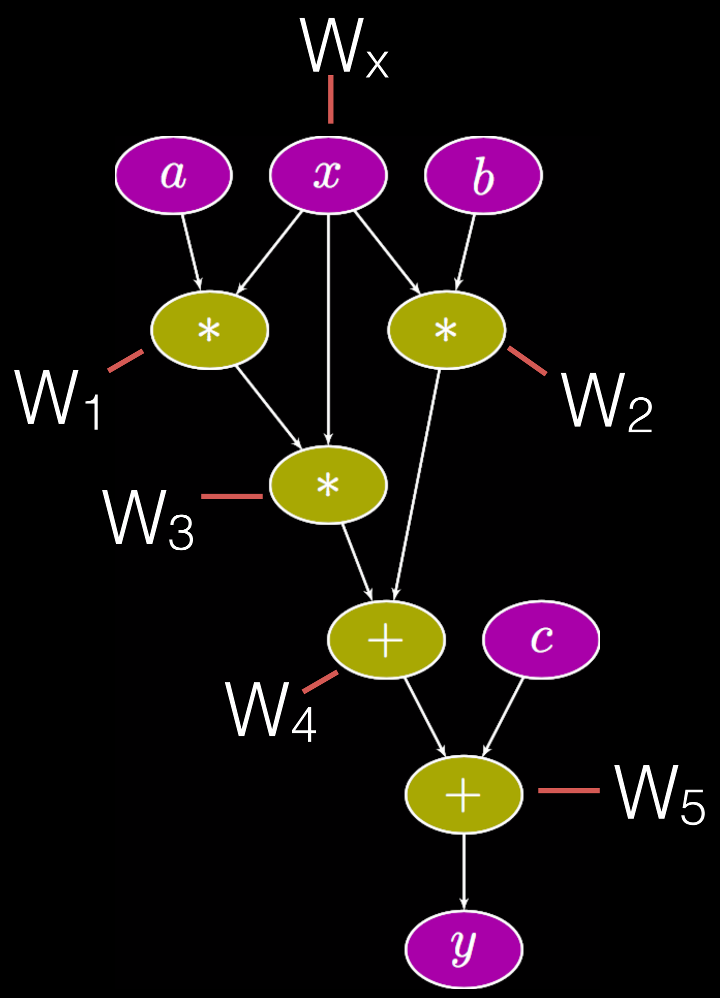
Tight
Intervals

Error Propagation



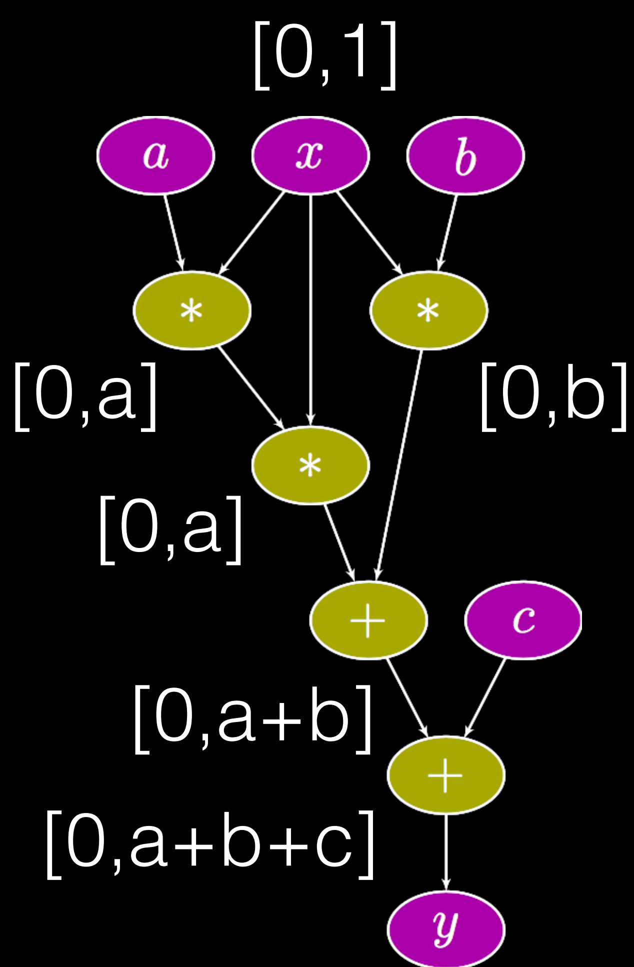
Tight Intervals

+

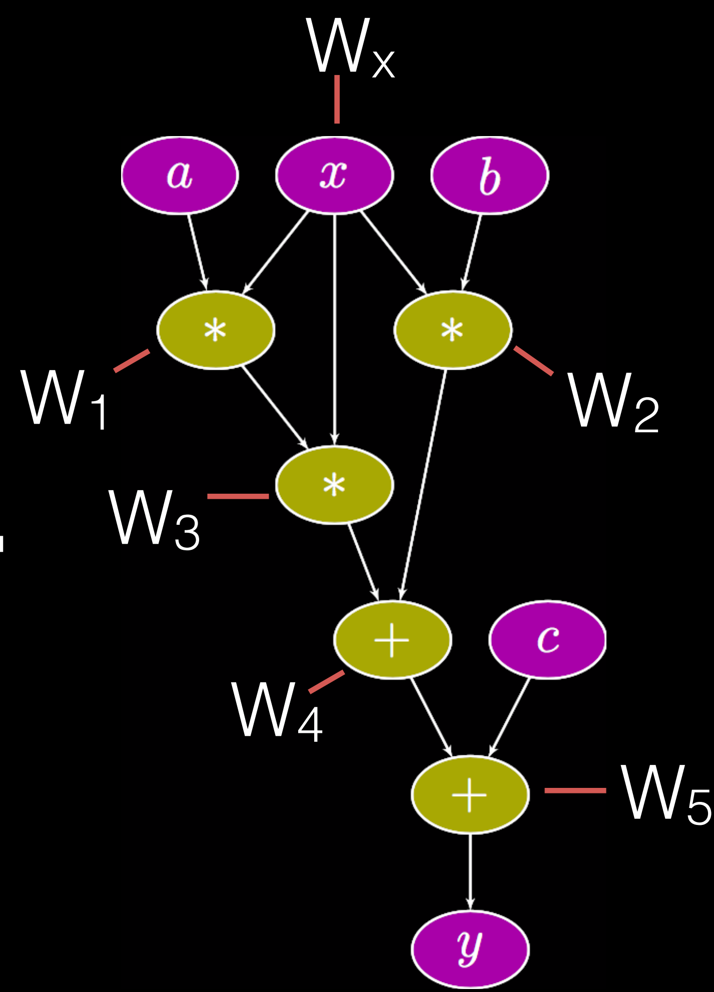


Bitwidth Assignment

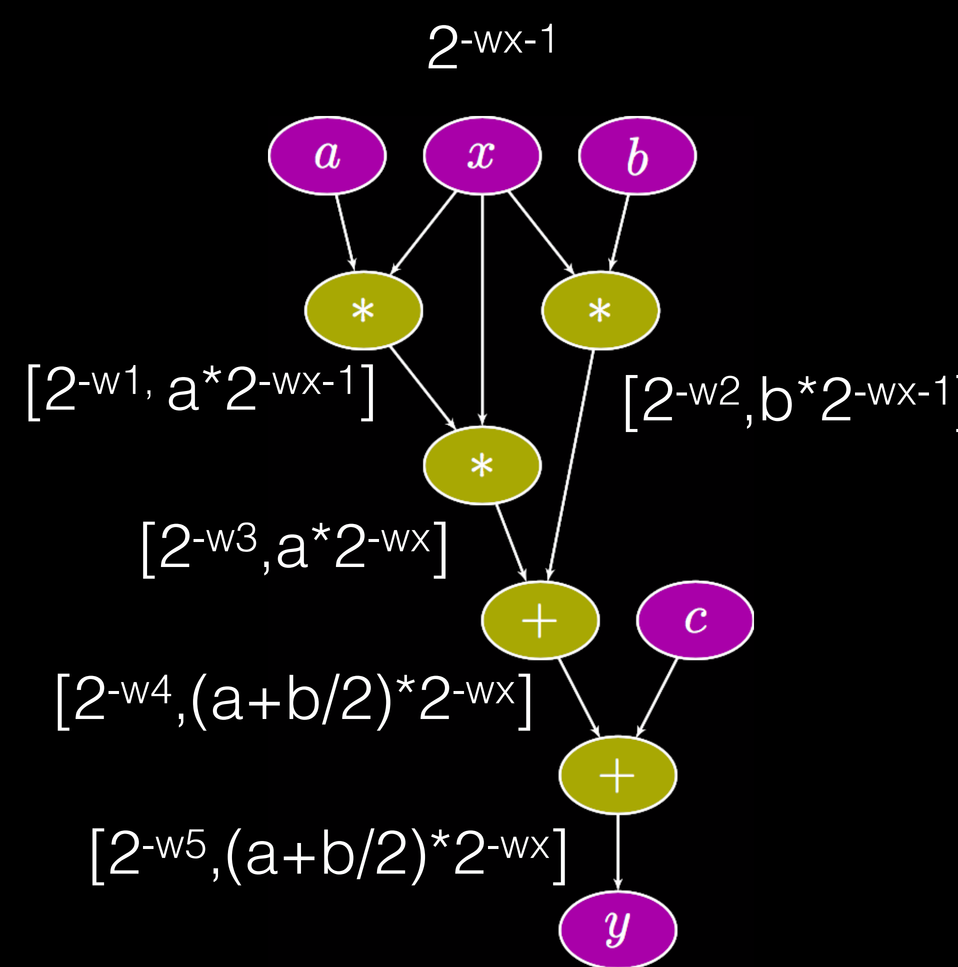
Error Propagation



+



=



Tight Intervals

Bitwidth Assignment

Error Estimates

Resource Evaluation

Resource Evaluation

RTL for operators

VIVADO™

Resource Table/Eqn.

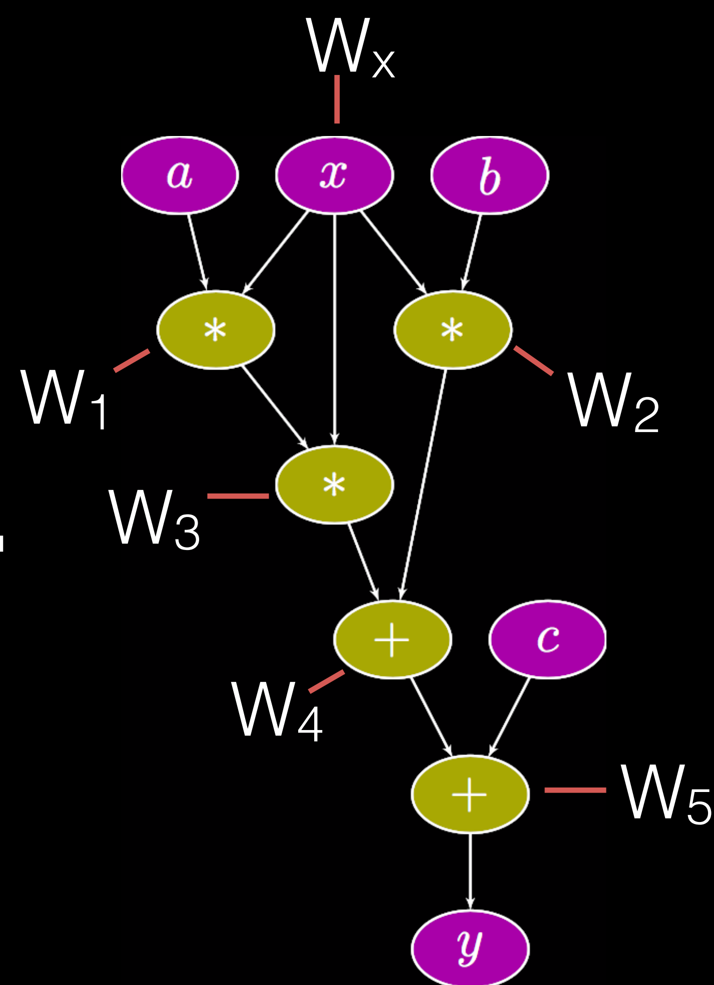
Resource Evaluation

RTL for operators

VIVADO

Resource Table/Eqn.

+

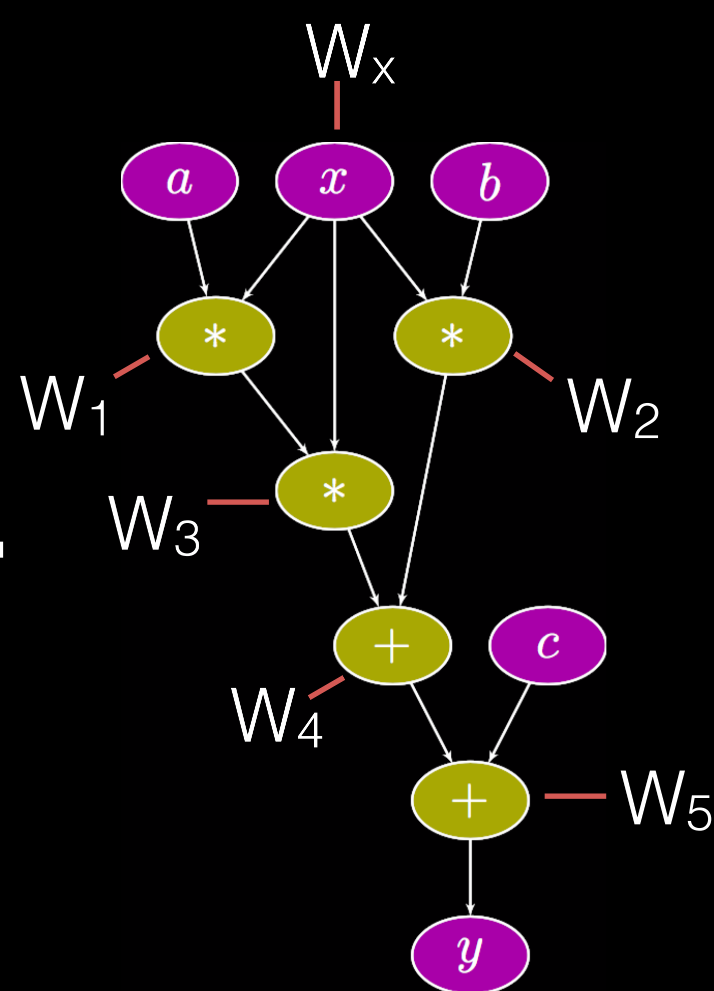


Bitwidth Assignment

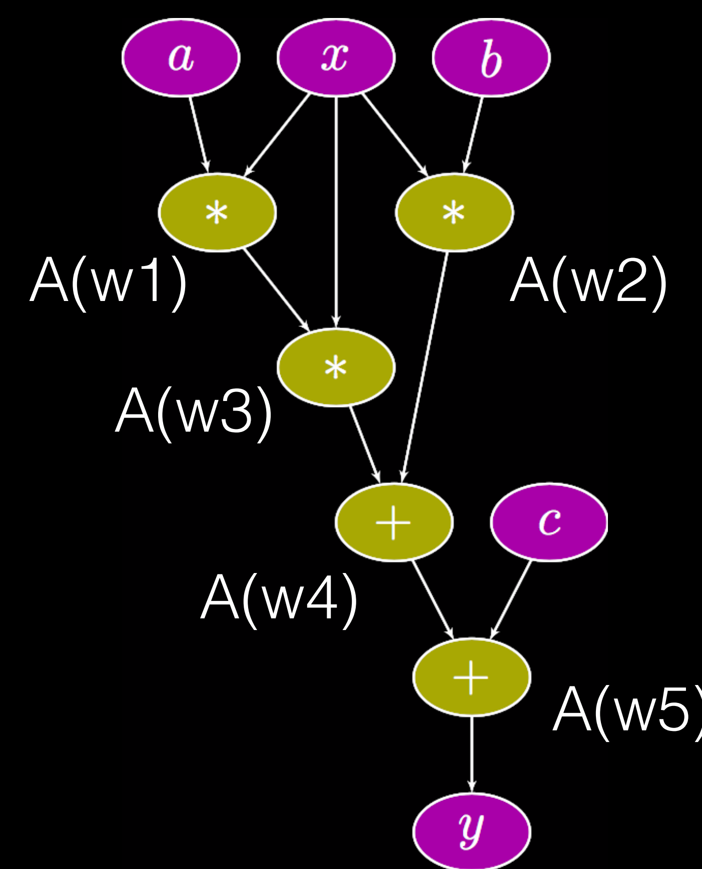
Resource Evaluation

RTL for operators

Resource Table/Eqn.

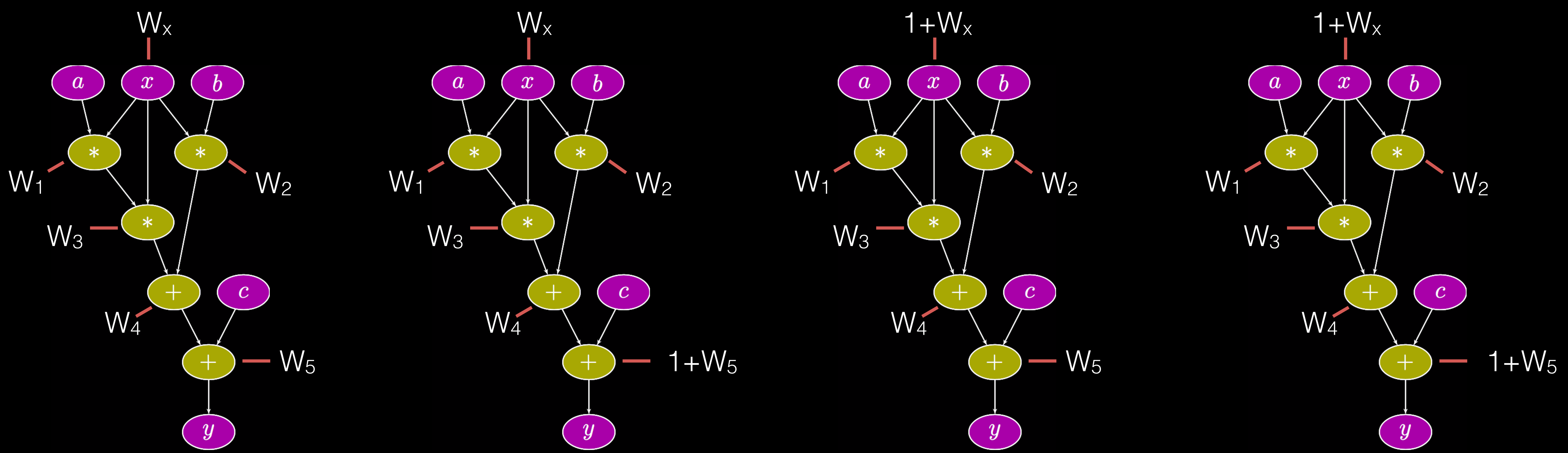


Bitwidth Assignment

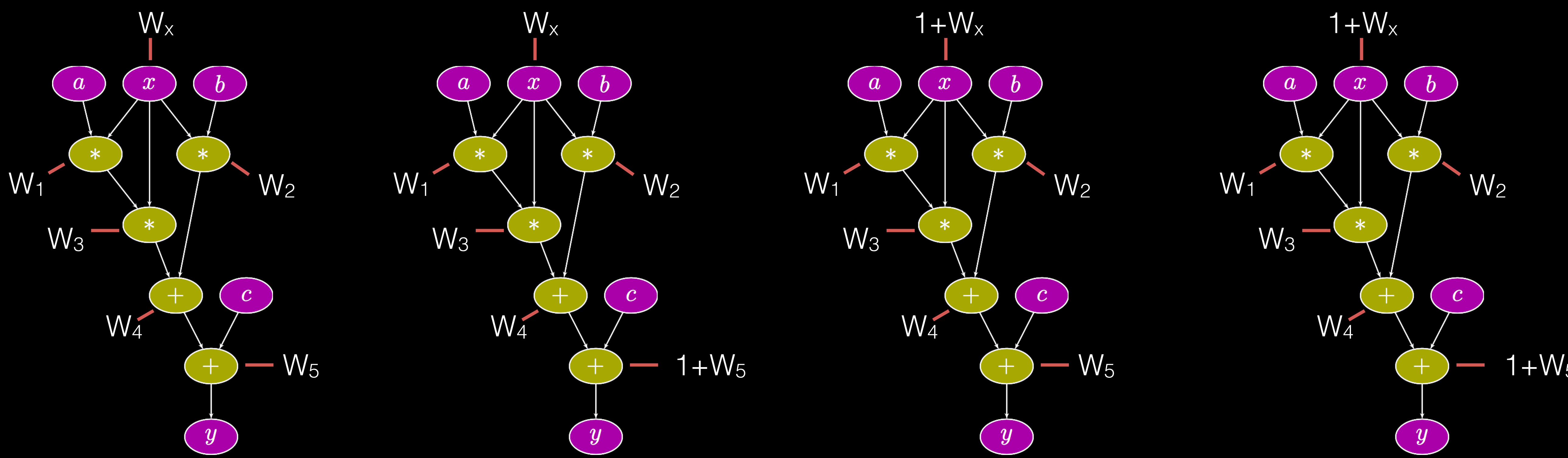


Resource Prediction

Parallel Explore



Parallel Explore



(1) Evaluate multiple combinations of bitwidths
(2) Common dataflow propagate/evaluate pattern

GPU Architecture
Parallel Potential

Modern GPU Potential

Modern GPU Potential

Modern GPU Potential

- Can run thousands of threads in parallel
  — Each thread do lightweight tasks
  — Suitable for high-throughput computing

Modern GPU Potential

- Can run thousands of threads in parallel
  — Each thread do lightweight tasks
  — Suitable for high-throughput computing

- NVIDIA Tesla K20
  — 26K threads
  — 5G RAM

Modern GPU Potential

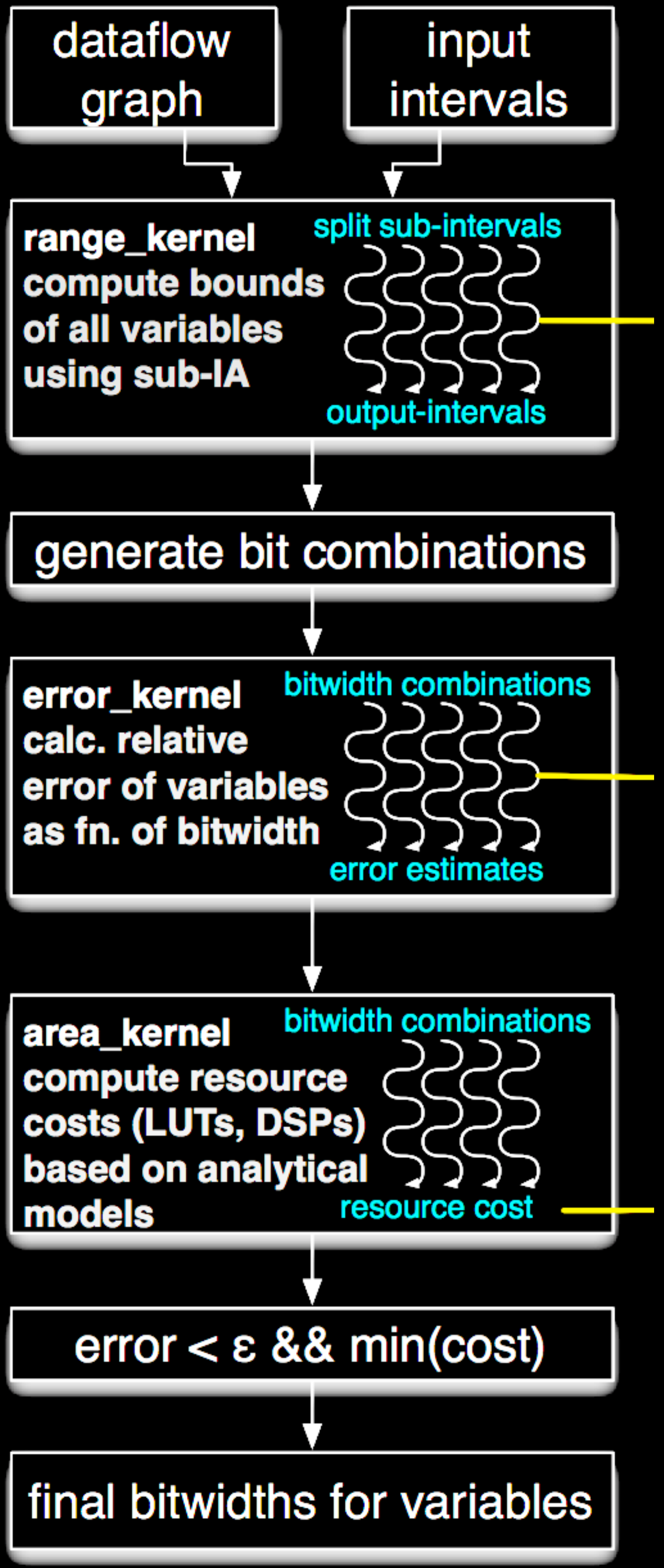
- Can run thousands of threads in parallel
  — Each thread do lightweight tasks
  — Suitable for high-throughput computing

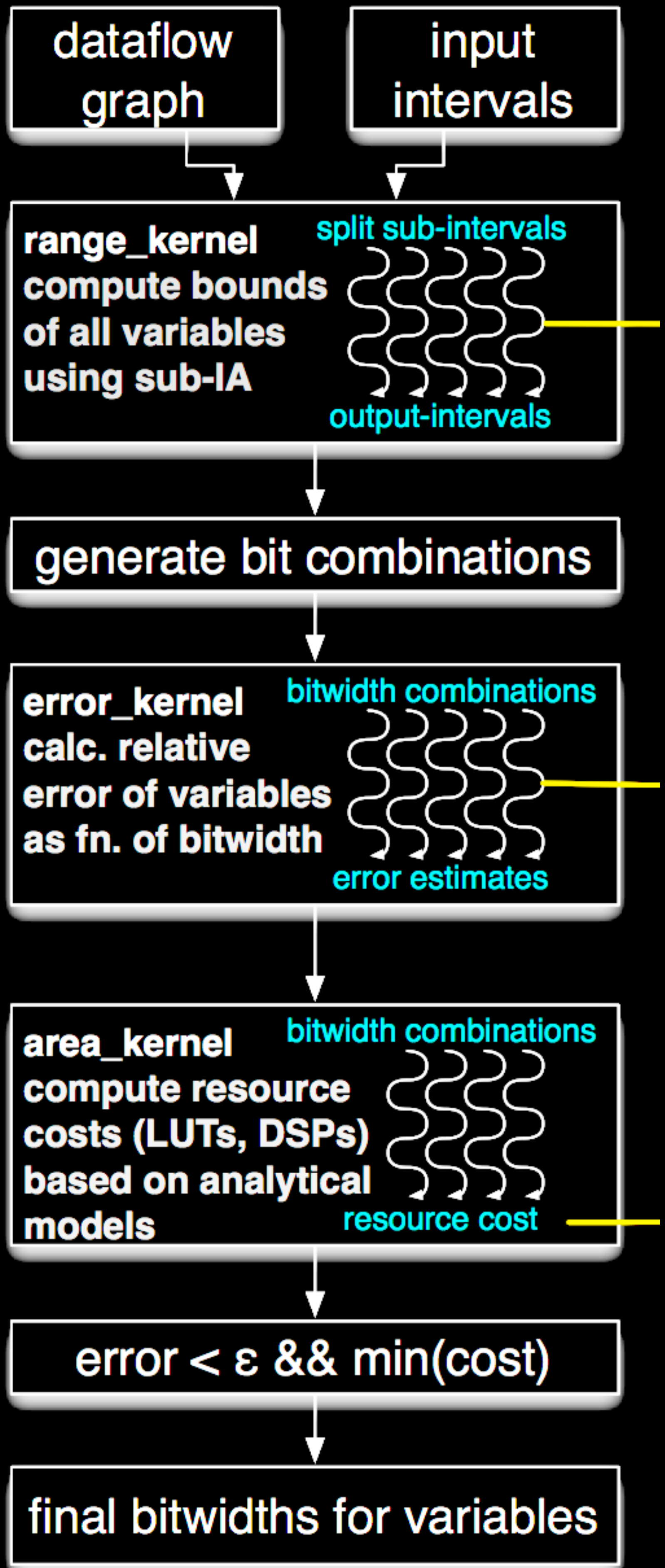
- NVIDIA Tesla K20
  — 26K threads
  — 5G RAM

- Common dataflow evaluation pattern

GPU Implementation

GPU Implementation

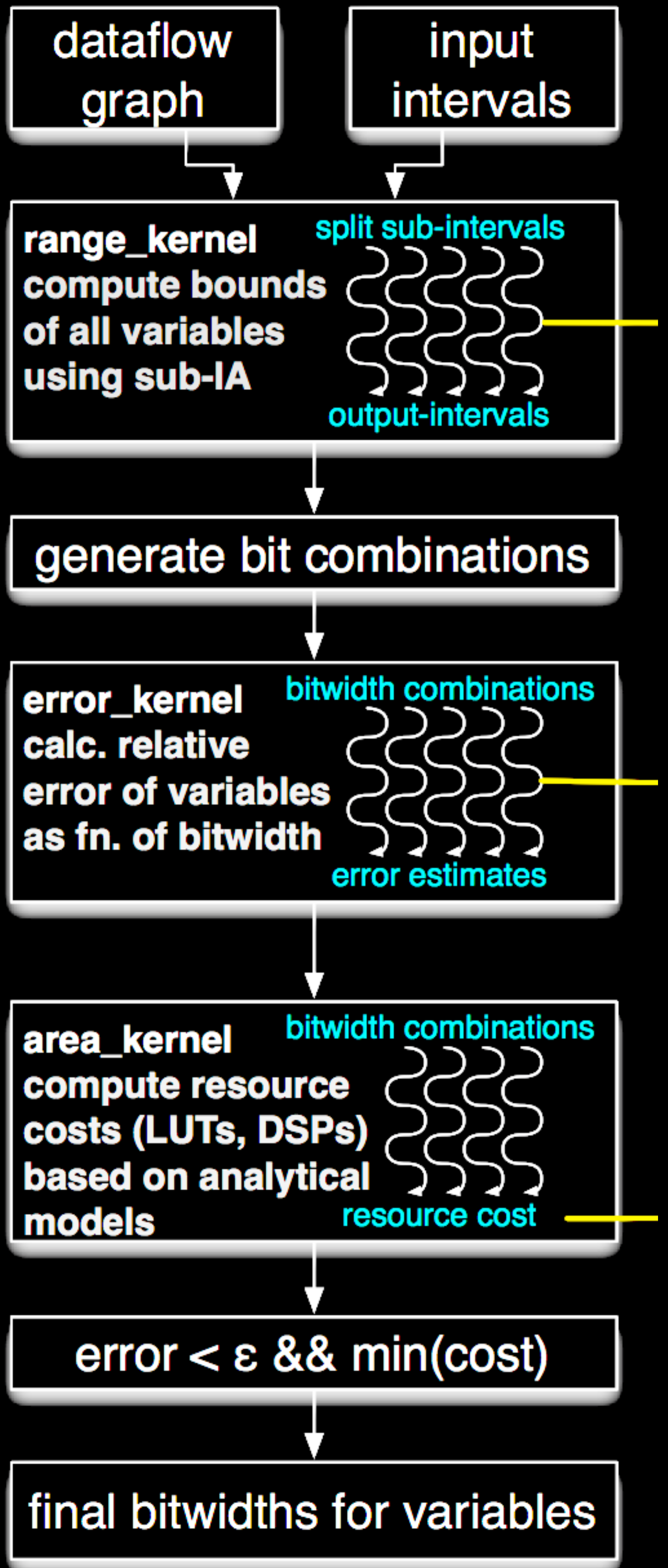




GPU Implementation

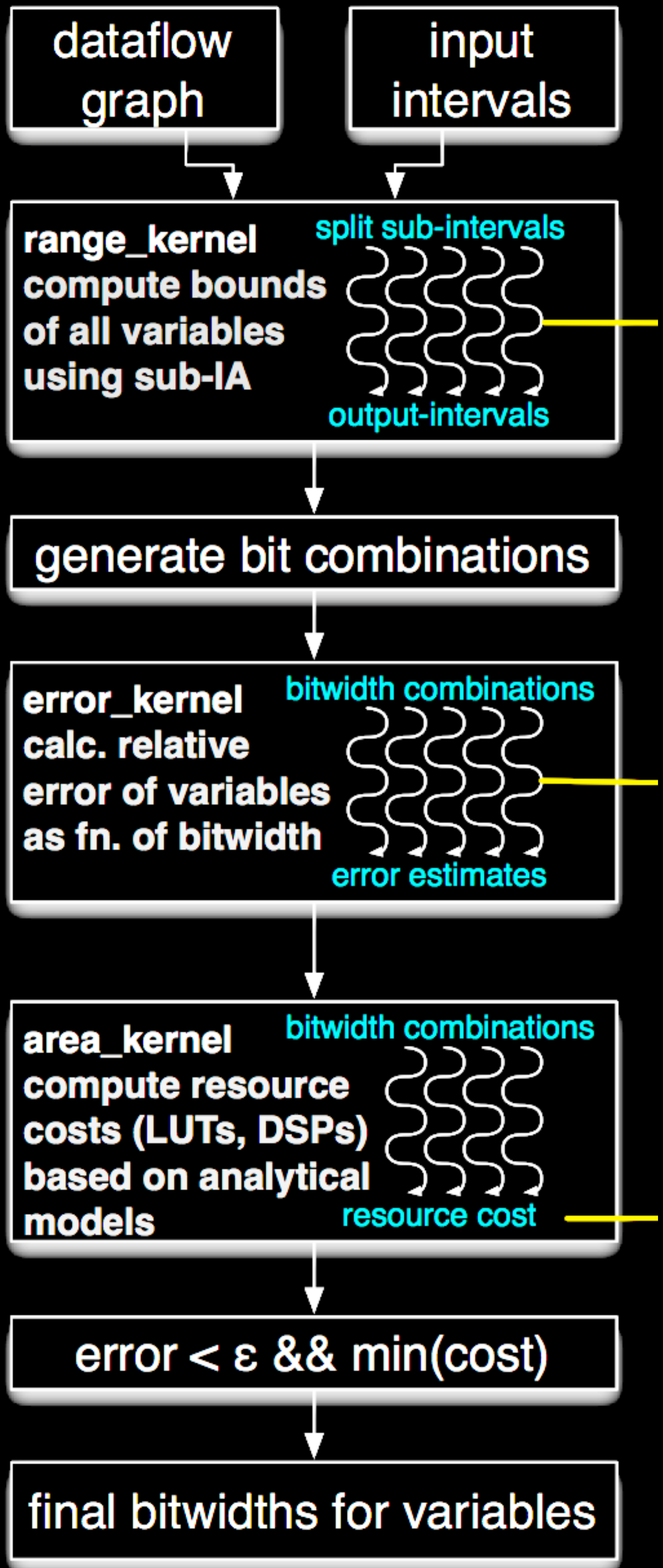
- Three core GPU-accelerated kernels
  — sub-interval analysis
  — error propagation
  — resource evaluation

GPU Implementation



- Three core GPU-accelerated kernels
  — sub-interval analysis
  — error propagation
  — resource evaluation

- Common operation: graph traversal
  — updating state per variable/node



GPU Implementation

- Three core GPU-accelerated kernels
  — sub-interval analysis
  — error propagation
  — resource evaluation

- Common operation: graph traversal
  — updating state per variable/node

- Parallelize to match GPU potential, and application requirement..

Example CUDA code

```
__device__ void add_range(...)
{
  double t1 = x1+y1;
  double t2 = x0+y0;

  *high=max(t1,t2);
  *low=min(t1,t2);
}
```

```
__device__ void add_error(...)
{
  double max = (e1>=e2)?e1:e2;
  if (e3 > max)
      *error = e3 + e1 + e2;
  else
      *error = e2 + e1;
}
```

```
__device__ void add_area(...)
{
  *area = (x1>x2)?x1:x2;
}
```

- Bulk of the work performed by leaf-level CUDA functions
- Example for fixed-point addition
- **Range:** inputs [x0,y0] and [x1,y1], compute output [high, low]
- **Error:** errors e1, and e2 (inputs) and e3 (depends on bitwidth of add)
- **Area:** approximate larger of two input precisions to lookup LUTs, DSPs, BRAMs

K20 GPU vs i5-4570 Ideal Performance

Kernel	Speedup				
	add	mult	div	exp	log
Range analysis	312×	213×	119×	298×	254×
Error propagation	246×	80×	103×	261×	297×
Resource estimation	272×	266×	251×	414×	407×

Limits of dump parallelism

Limits of dump parallelism

5—6G
RAM

Limits of dump parallelism

1. Cannot enumerate any reasonable solution space, even with 100x speedups
   — 16 variables, 8 choices/var — 8^{16} (200 peta combinations)

Limits of dump parallelism

1. Cannot enumerate any reasonable solution space, even with 100x speedups
   — 16 variables, 8 choices/var — 8^{16} (200 peta combinations)

2. Limited DRAM capacity on GPU card
   — cannot “store” all solutions!
   — 16 variables, 32b range/var/combination, 4 values/variable (2—3M combinations feasible)

Pruning of Search Space

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision
Result: Bounded search space

```
bit_width(0:N-1) ← target_fb;
while current_error > error_constraint do
    bit_width(0:N-1) ++;
end
uniform_bit = bit_width[0];
foreach n=0:N-1 do
    while current_error ≤ error_constraint do
        bit_width(n)−−;
    end
    lowest(n)← bit_width(n);
    bit_width(n)← uniform_bit;
end
bit_width(0:N-1) ← lowest(0:N-1);
while current_error ≤ error_constraint do
    bit_width(0:N-1)++;
end
highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```

Pruning of Search Space

- Start with safe common precision values for all variables

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision
Result: Bounded search space

```
bit_width(0:N-1) ← target_fb;
while current_error > error_constraint do
    bit_width(0:N-1) ++;
end
uniform_bit = bit_width[0];
foreach n=0:N-1 do
    while current_error ≤ error_constraint do
        bit_width(n)−−;
    end
    lowest(n)← bit_width(n);
    bit_width(n)← uniform_bit;
end
bit_width(0:N-1) ← lowest(0:N-1);
while current_error ≤ error_constraint do
    bit_width(0:N-1)++;
end
highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```

Pruning of Search Space

- Start with safe common precision values for all variables
- Reduce each variable precise separately until failure — error>threshold

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision
Result: Bounded search space

```
bit_width(0:N-1) ← target_fb;
while current_error > error_constraint do
    bit_width(0:N-1) ++;
end
uniform_bit = bit_width[0];
foreach n=0:N-1 do
    while current_error ≤ error_constraint do
        bit_width(n)−−;
    end
    lowest(n)← bit_width(n);
    bit_width(n)← uniform_bit;
end
bit_width(0:N-1) ← lowest(0:N-1);
while current_error ≤ error_constraint do
    bit_width(0:N-1)++;
end
highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```

Pruning of Search Space

- Start with safe common precision values for all variables
- Reduce each variable precise separately until failure — error>threshold
- Use these as lower limits per variable

Algorithm 1: Search Space Pruning Heuristic

Data: The number of variables N; Targeted Fixed-point Precision
Result: Bounded search space

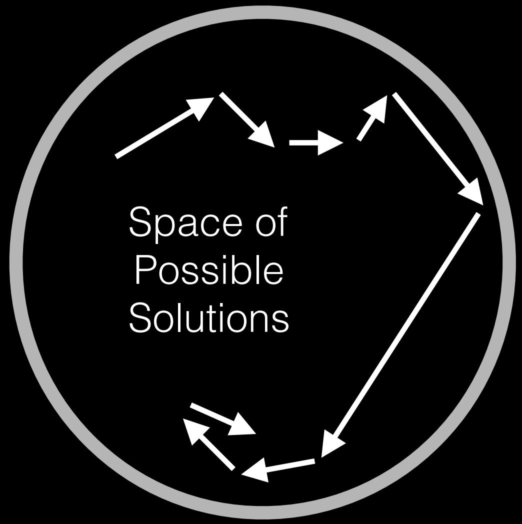
```
bit_width(0:N-1) ← target_fb;
while current_error > error_constraint do
    bit_width(0:N-1) ++;
end
uniform_bit = bit_width[0];
foreach n=0:N-1 do
    while current_error ≤ error_constraint do
        bit_width(n)−−;
    end
    lowest(n)← bit_width(n);
    bit_width(n)← uniform_bit;
end
bit_width(0:N-1) ← lowest(0:N-1);
while current_error ≤ error_constraint do
    bit_width(0:N-1)++;
end
highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```

Pruning of Search Space

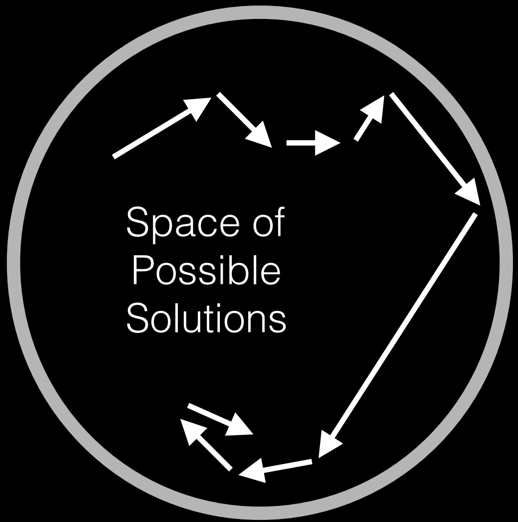
- Start with safe common precision values for all variables
- Reduce each variable precise separately until failure — error>threshold
- Use these as lower limits per variable
- Recalculate upper limits for search — recognizing GPU bounds

Algorithm 1: Search Space Pruning Heuristic

```
Data: The number of variables N; Targeted
      Fixed-point Precision
Result: Bounded search space
bit_width(0:N-1) ← target_fb;
while current_error > error_constraint do
  | bit_width(0:N-1) ++;
end
uniform_bit = bit_width[0];
foreach n=0:N-1 do
  | while current_error ≤ error_constraint do
  |   | bit_width(n)−−;
  | end
  | lowest(n)← bit_width(n);
  | bit_width(n)← uniform_bit;
end
bit_width(0:N-1) ← lowest(0:N-1);
while current_error ≤ error_constraint do
  | bit_width(0:N-1)++;
end
highest(0:N-1) ← bit_width(0:N-1) + guard_bit;
```

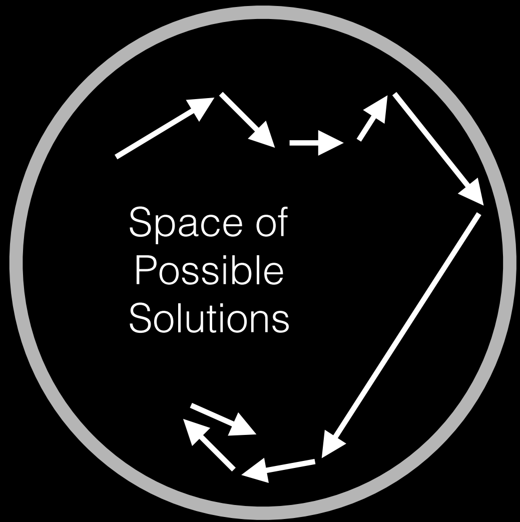


Simulated Annealing



Simulated Annealing

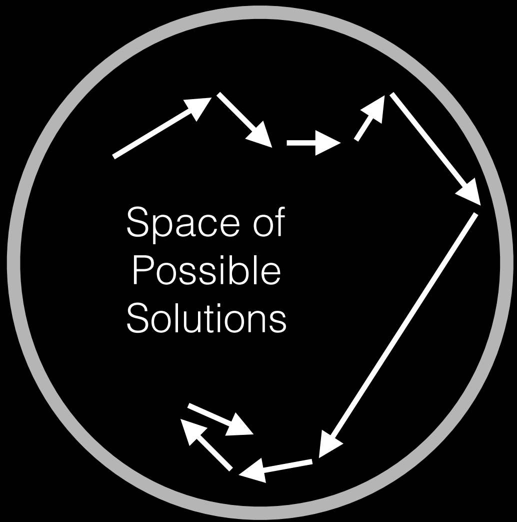
- ASA — Adaptive Simulated Annealing
  — Lester Ingber's Caltech page
  — great email support (to help us get started)



Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber’s Caltech page
  — great email support (to help us get started)

- This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:

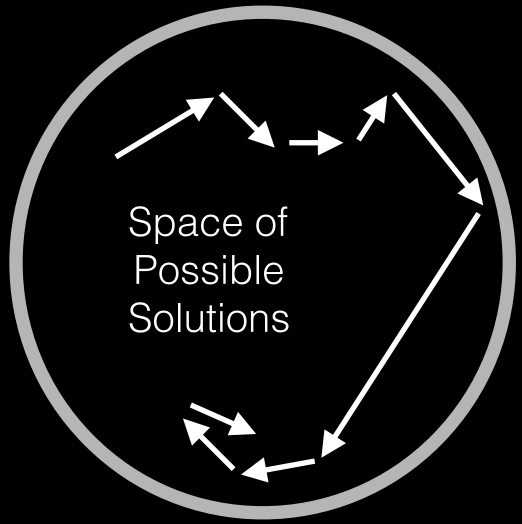


Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber's Caltech page
  — great email support (to help us get started)

- This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:

- (1) Modified asa_usr_cst.c to support FPGA resource models that we supplied ourselves



Simulated Annealing

- ASA — Adaptive Simulated Annealing
  — Lester Ingber's Caltech page
  — great email support (to help us get started)

- This is our CPU baseline (pruning required prior to running ASA, else system quits). Needs two tweaks:

- (1) Modified asa_usr_cst.c to support FPGA resource models that we supplied ourselves

- (2) Links to Gappa for analysing relative error

Quick HLS Flow

Demonstrate ideas

HLS Flow

HLS Flow

- Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development

HLS Flow

- Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development

- Develop a lightweight compilation flow
  — prove your idea/transformation
  — distribute, integrate with LLVM with community support

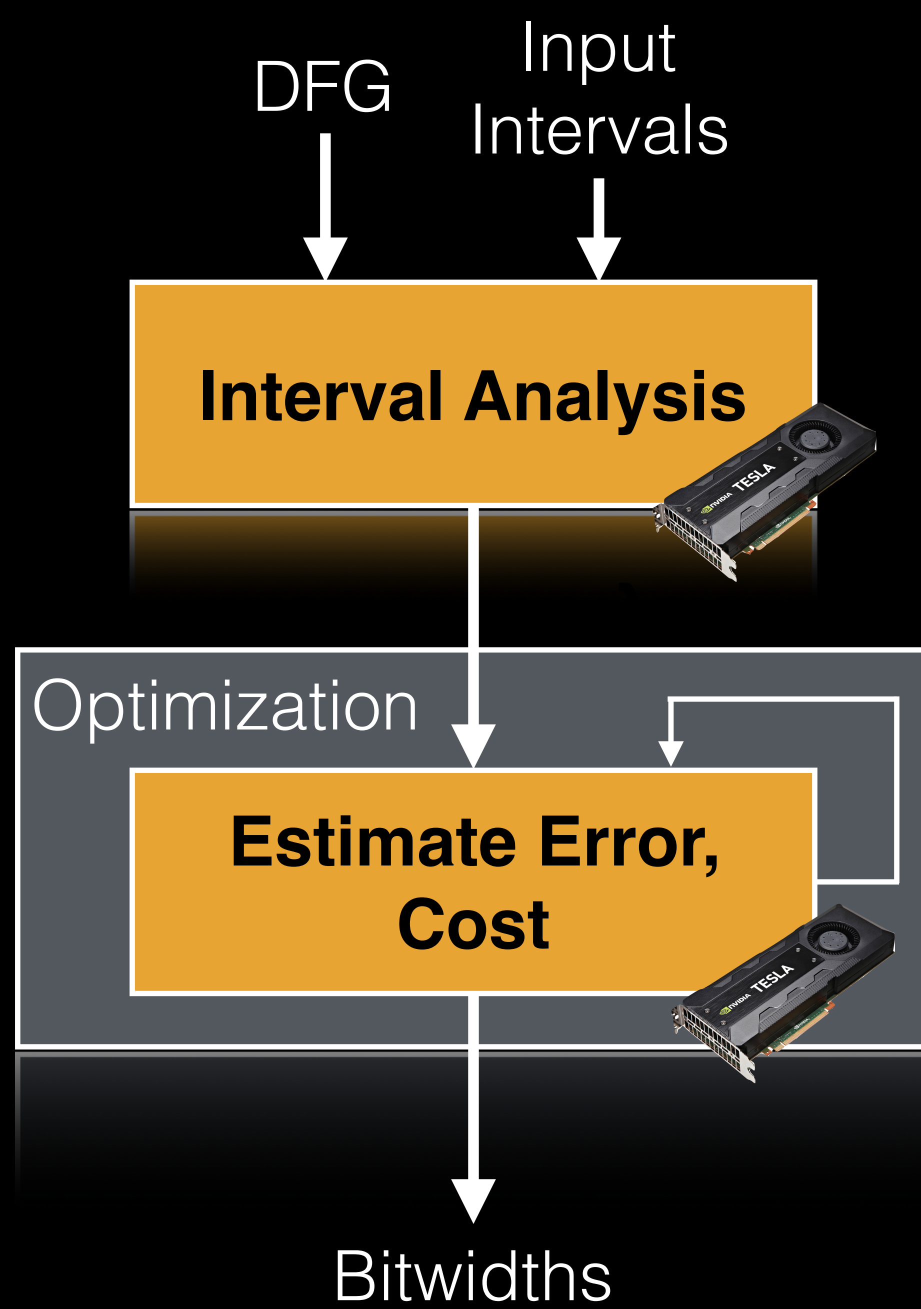
HLS Flow

- Engineering Constraints:
  — limited students/staff with LLVM experience
  — project spans hardware tools, software frameworks, CUDA development

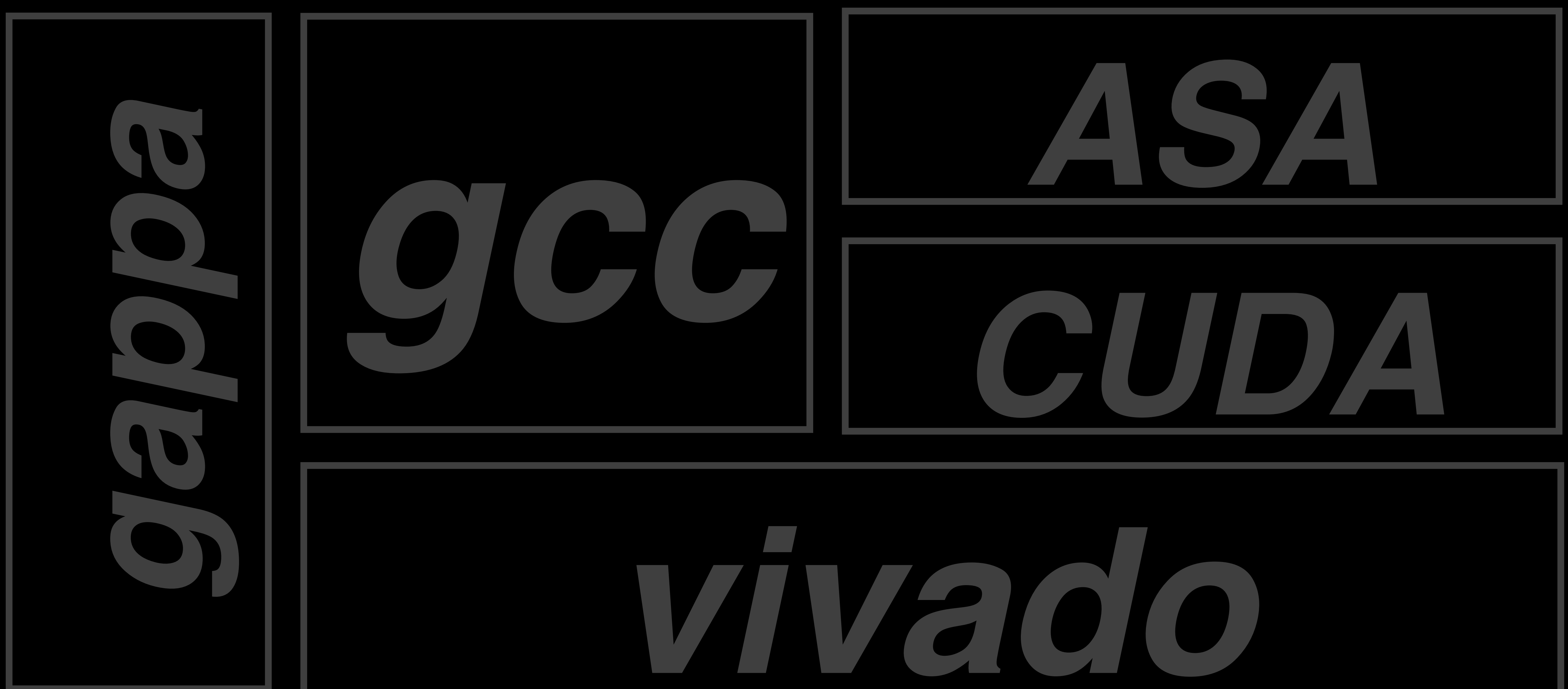
- Develop a lightweight compilation flow
  — prove your idea/transformation
  — distribute, integrate with LLVM with community support

- Not trying to build production-ready compiler
  — just interested in research/proof of concept

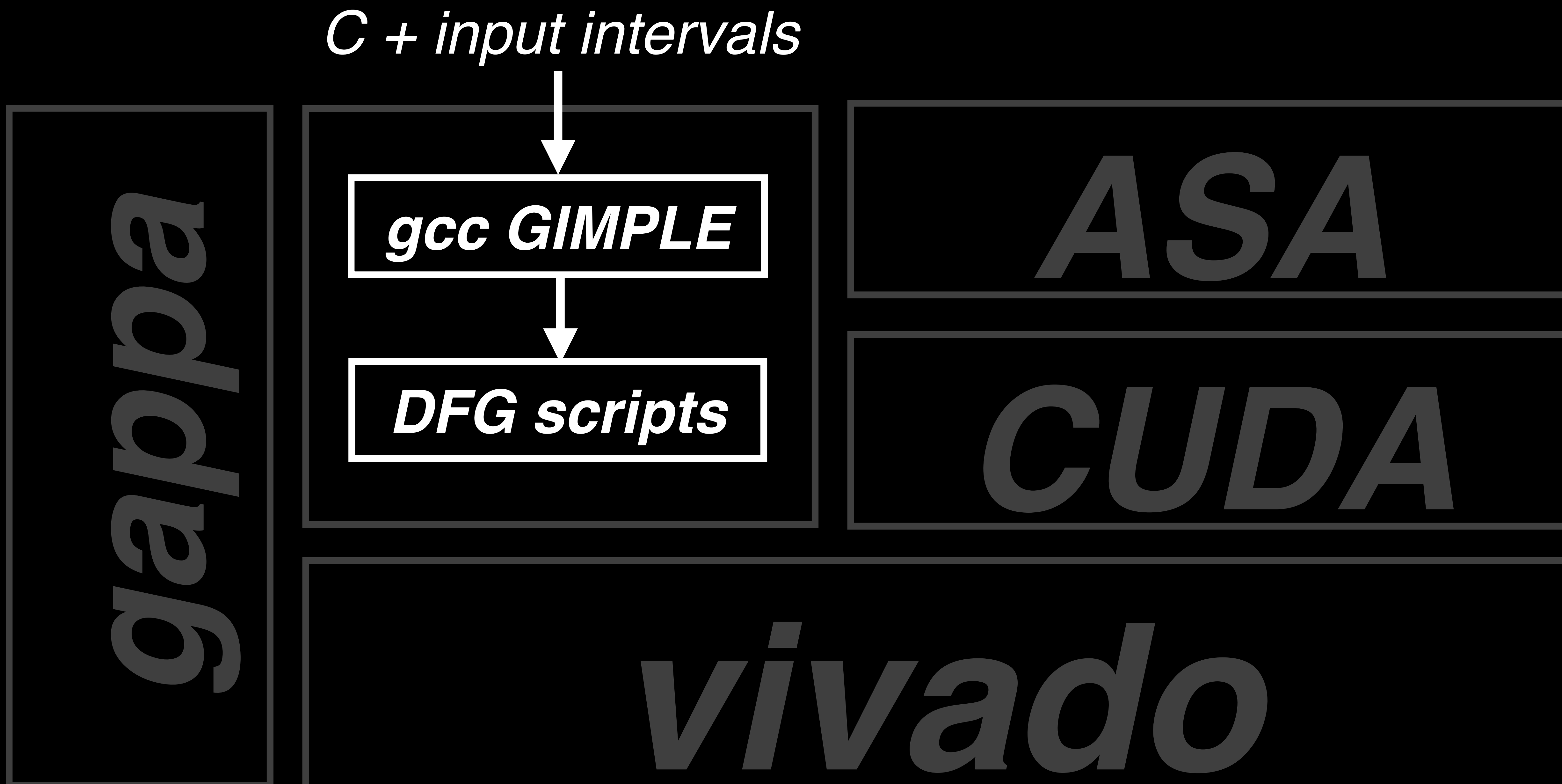
Quick HLS Toolflow



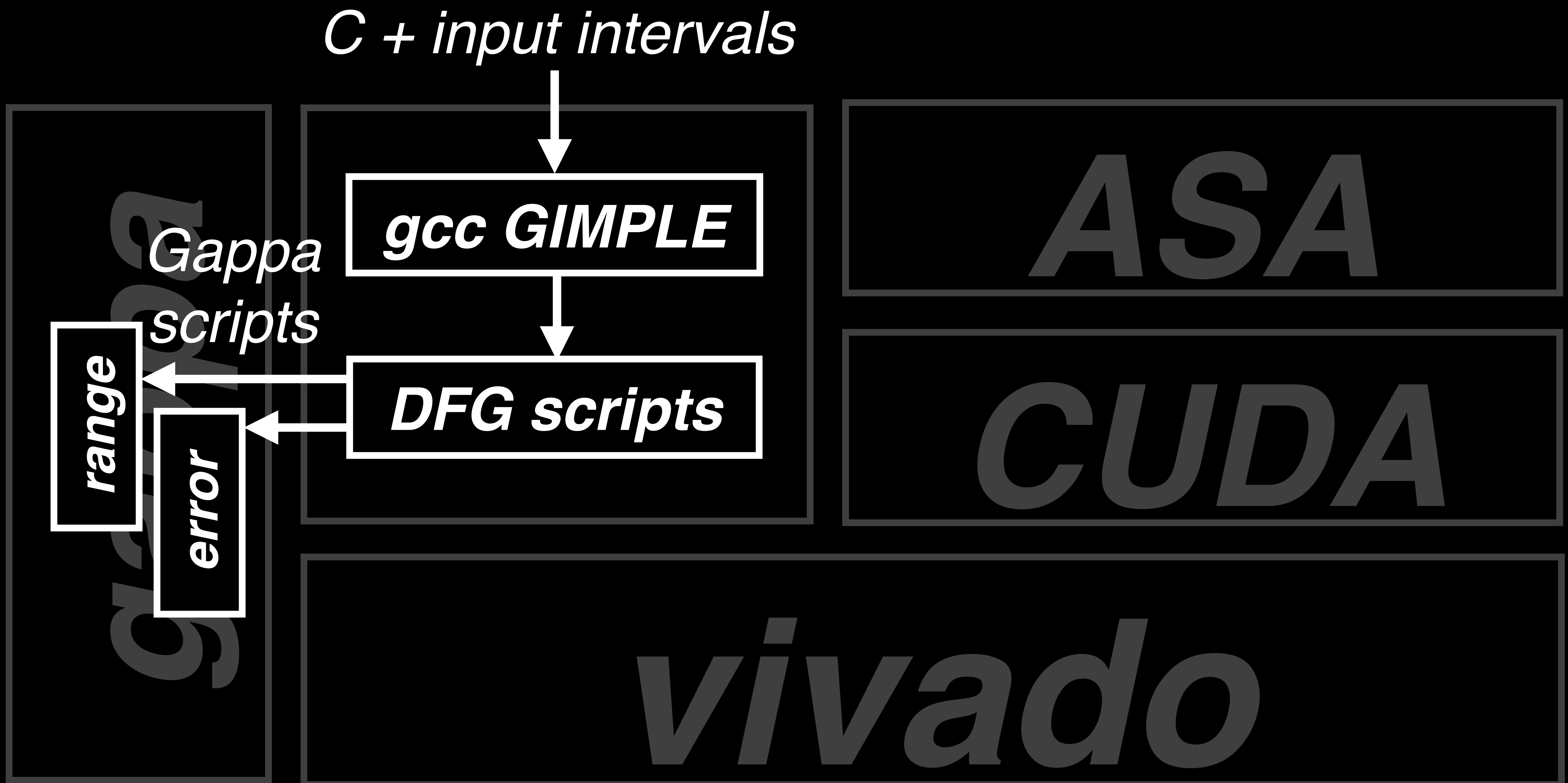
Quick HLS Toolflow



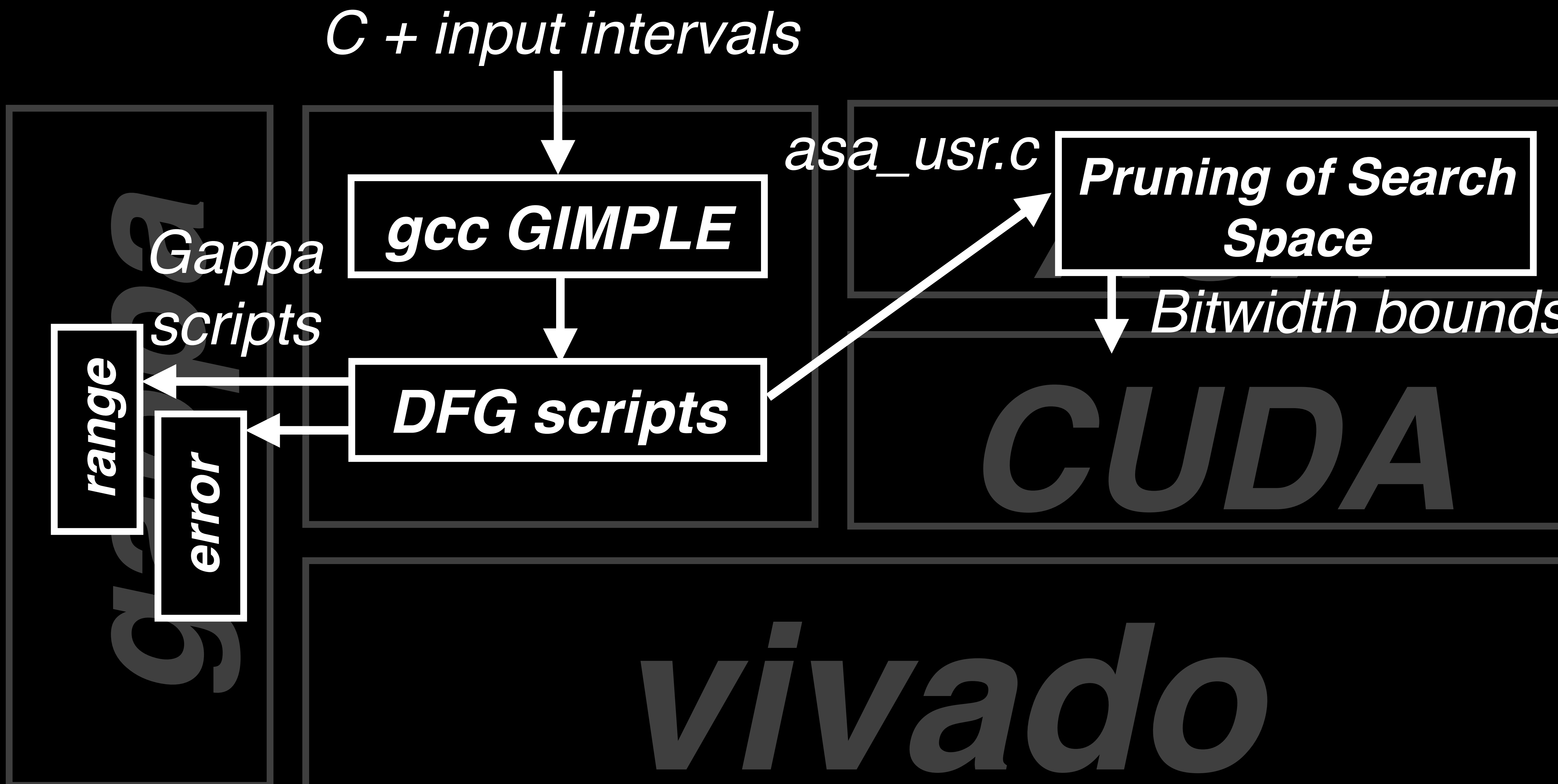
Quick HLS Toolflow



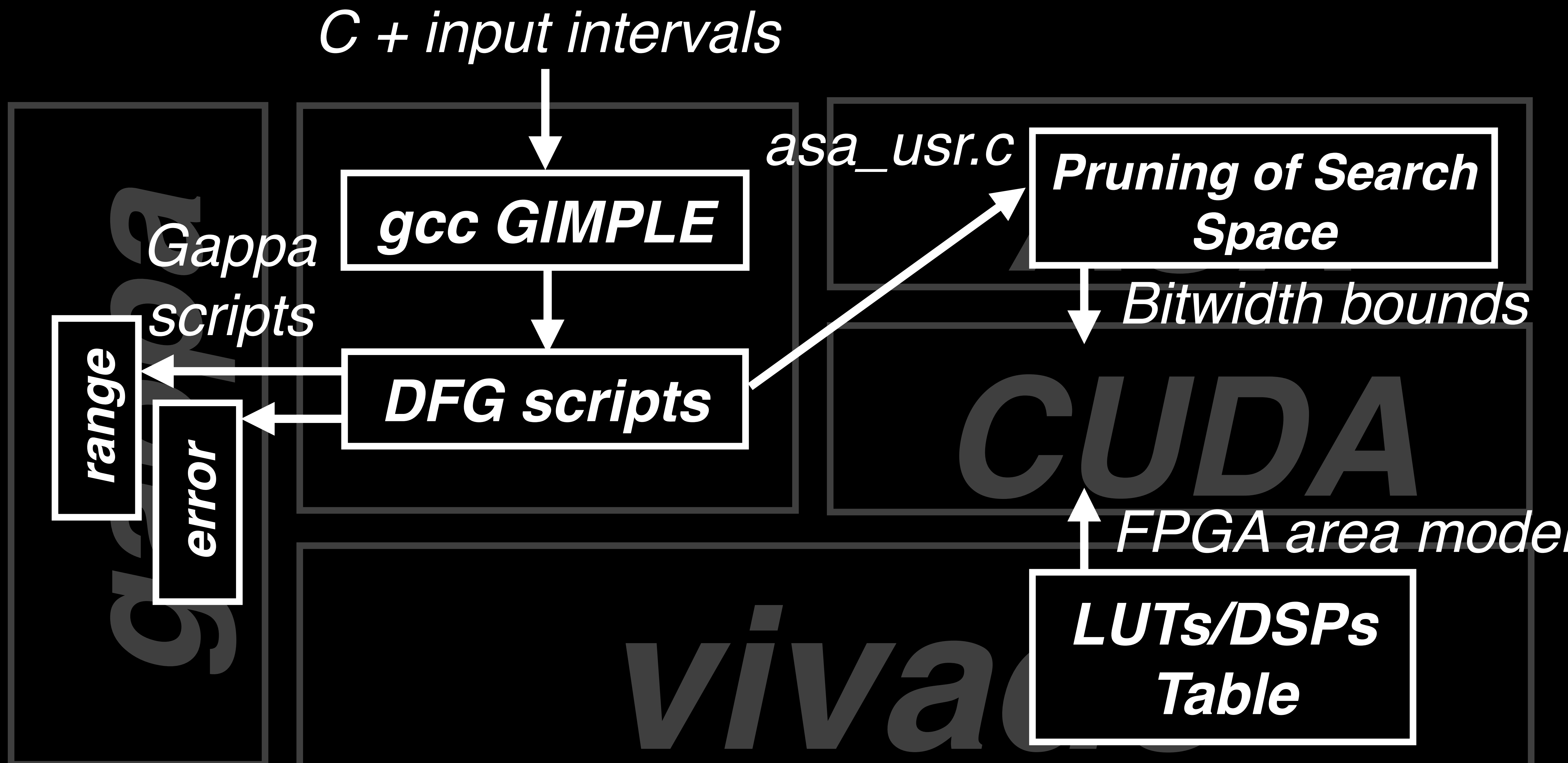
Quick HLS Toolflow



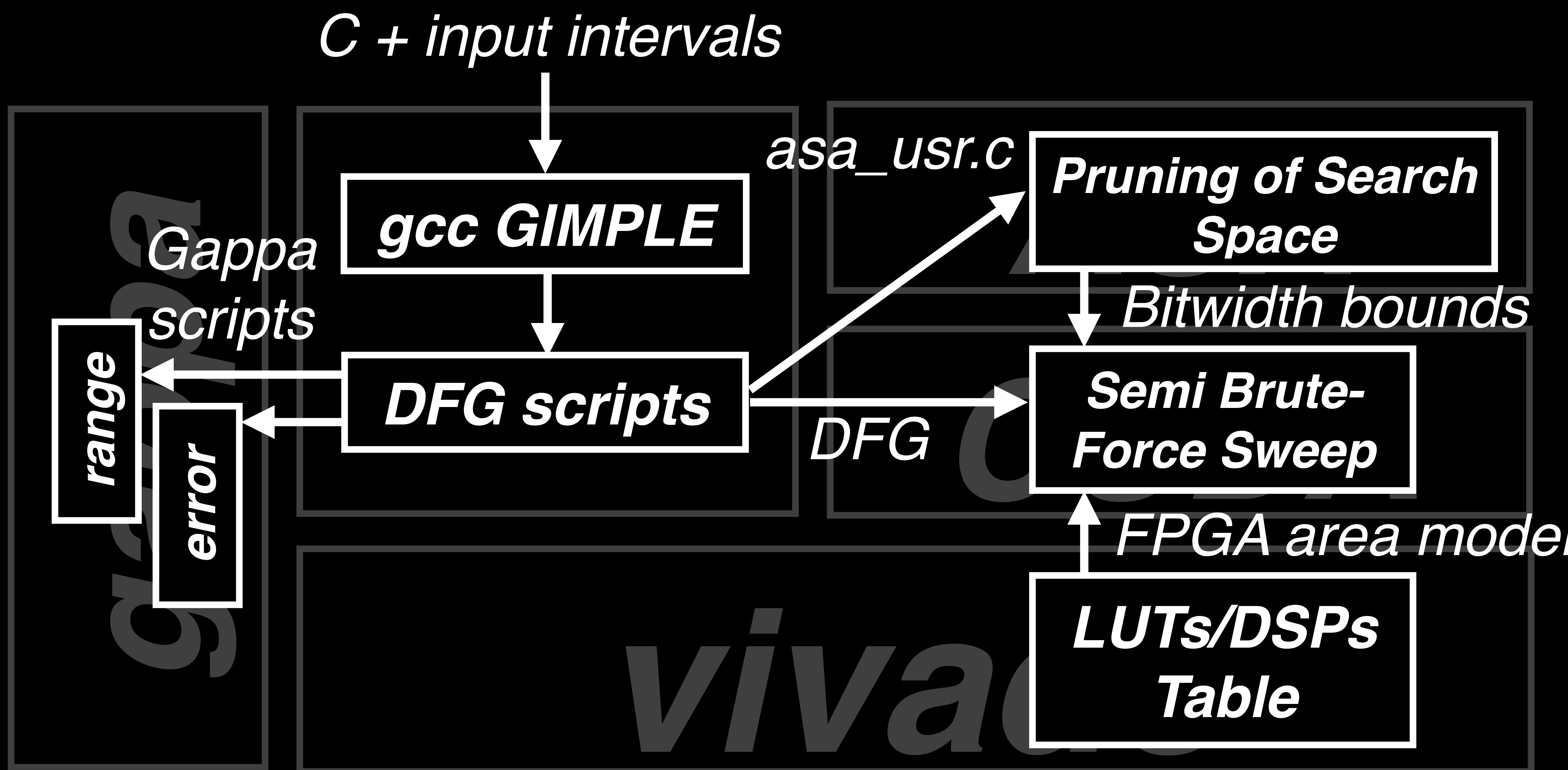
Quick HLS Toolflow



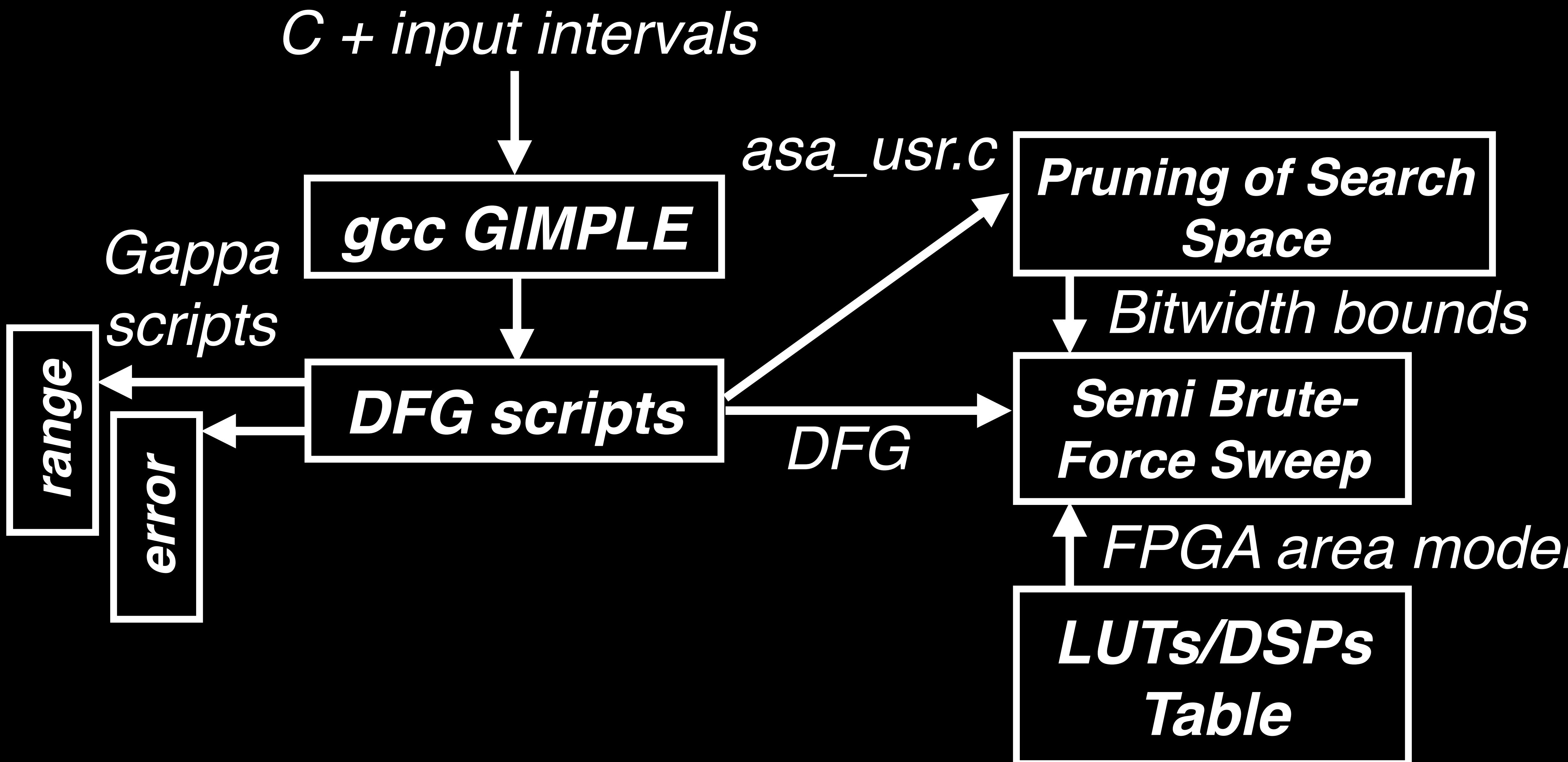
Quick HLS Toolflow



Quick HLS Toolflow



Quick HLS Toolflow

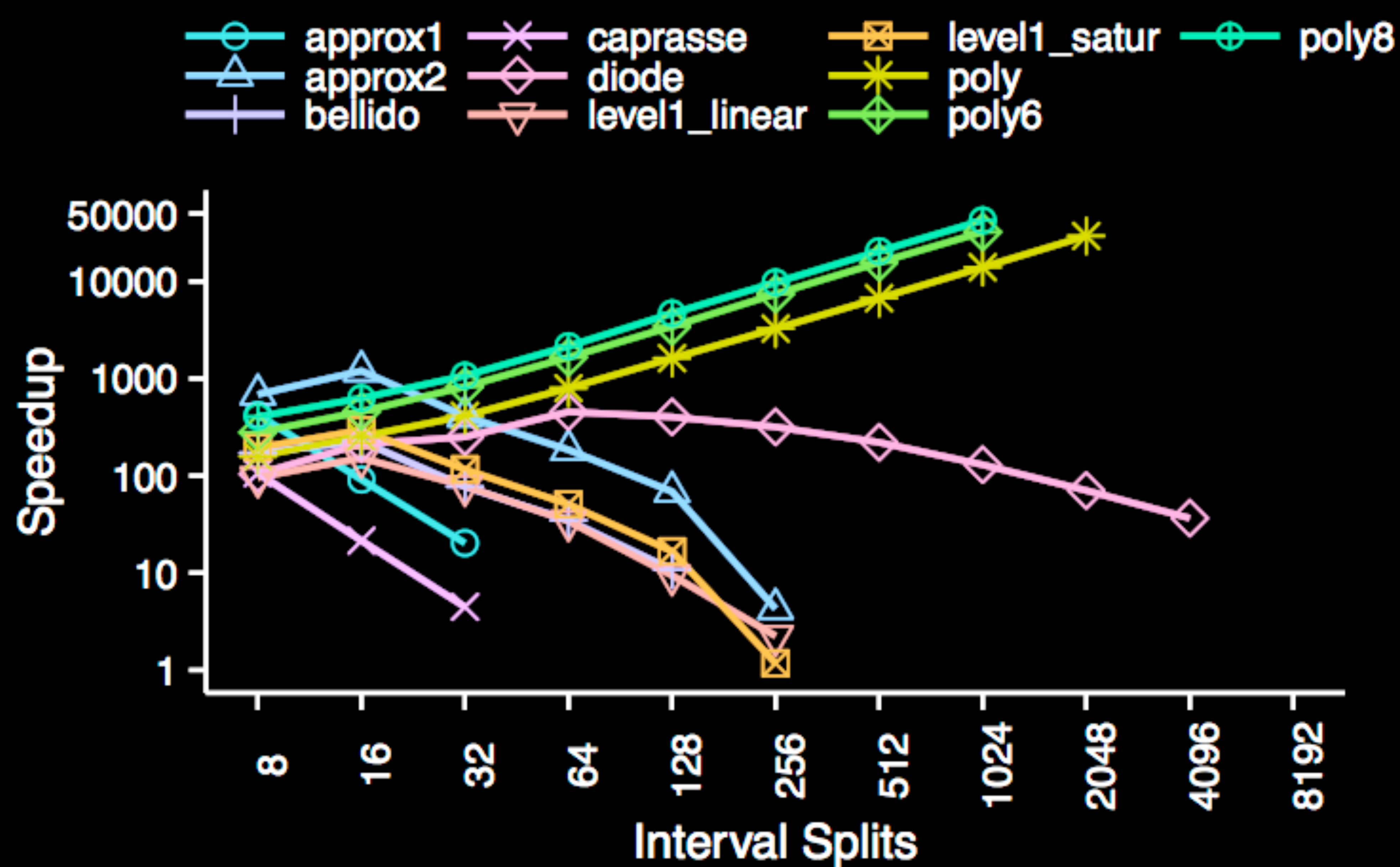


Experimental Validation

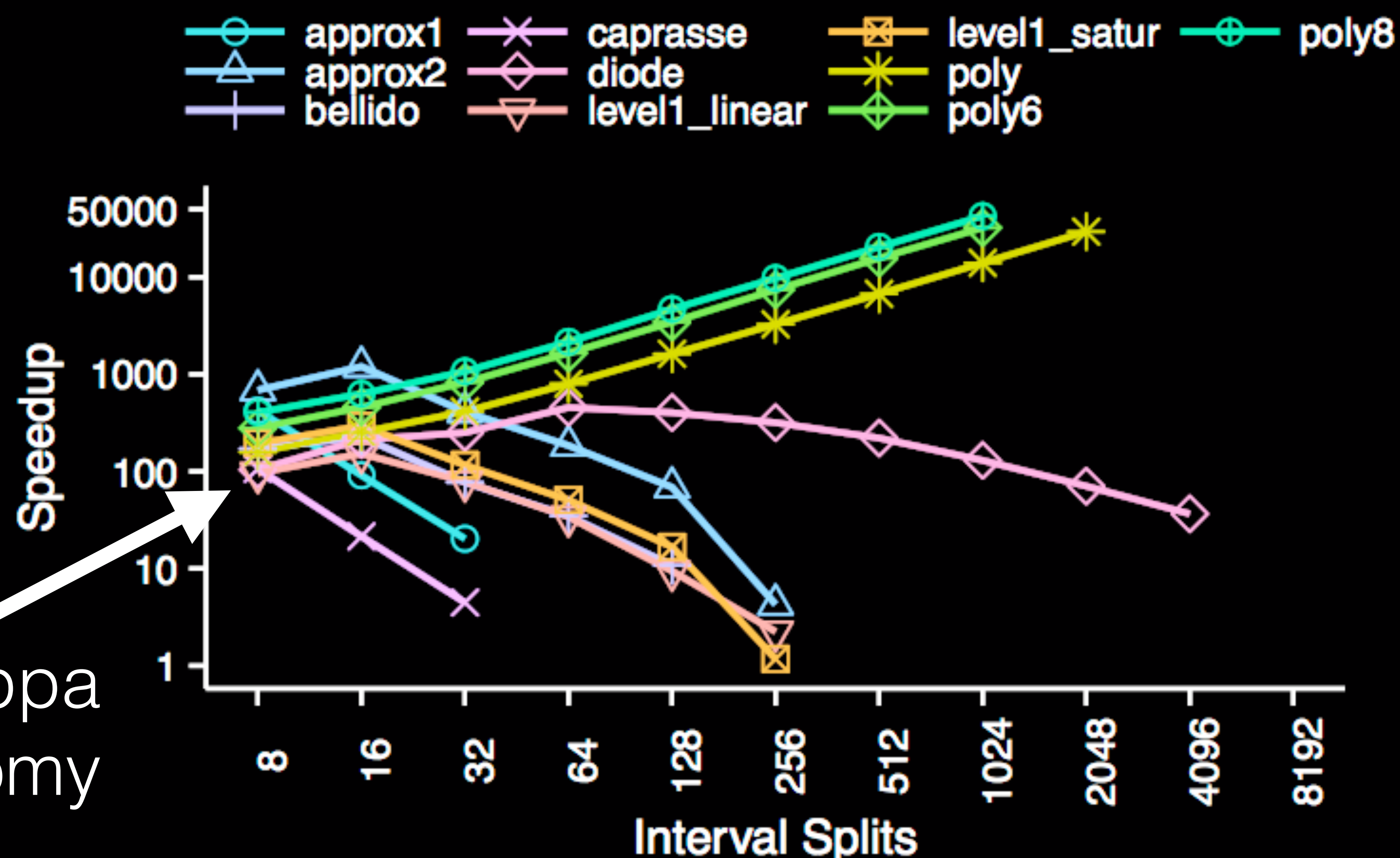
Outline of Results

- Speedup for range analysis
- Speedup for bitwidth optimization
- Quality-Time trends

Sub-interval arithmetic

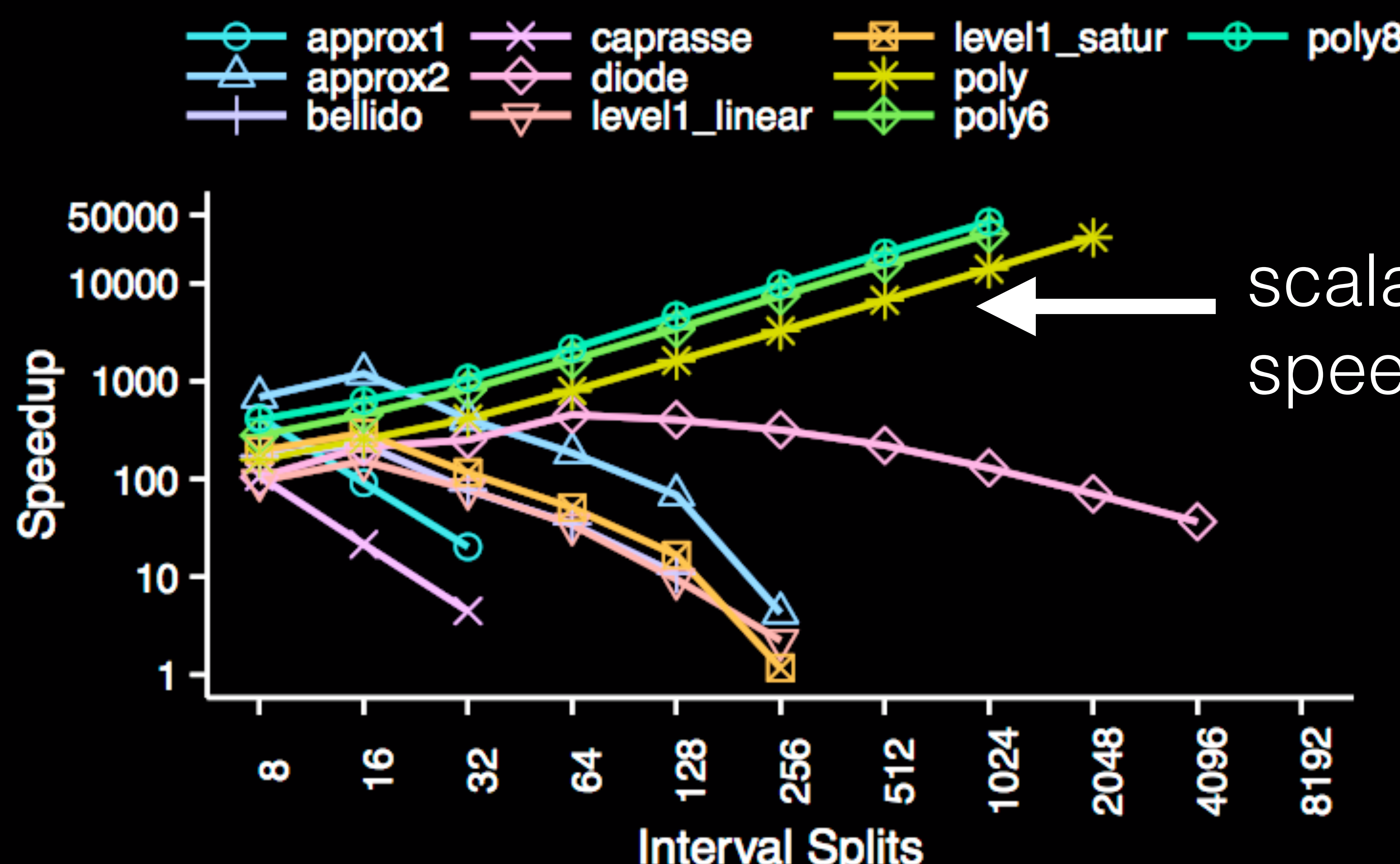


Sub-interval arithmetic



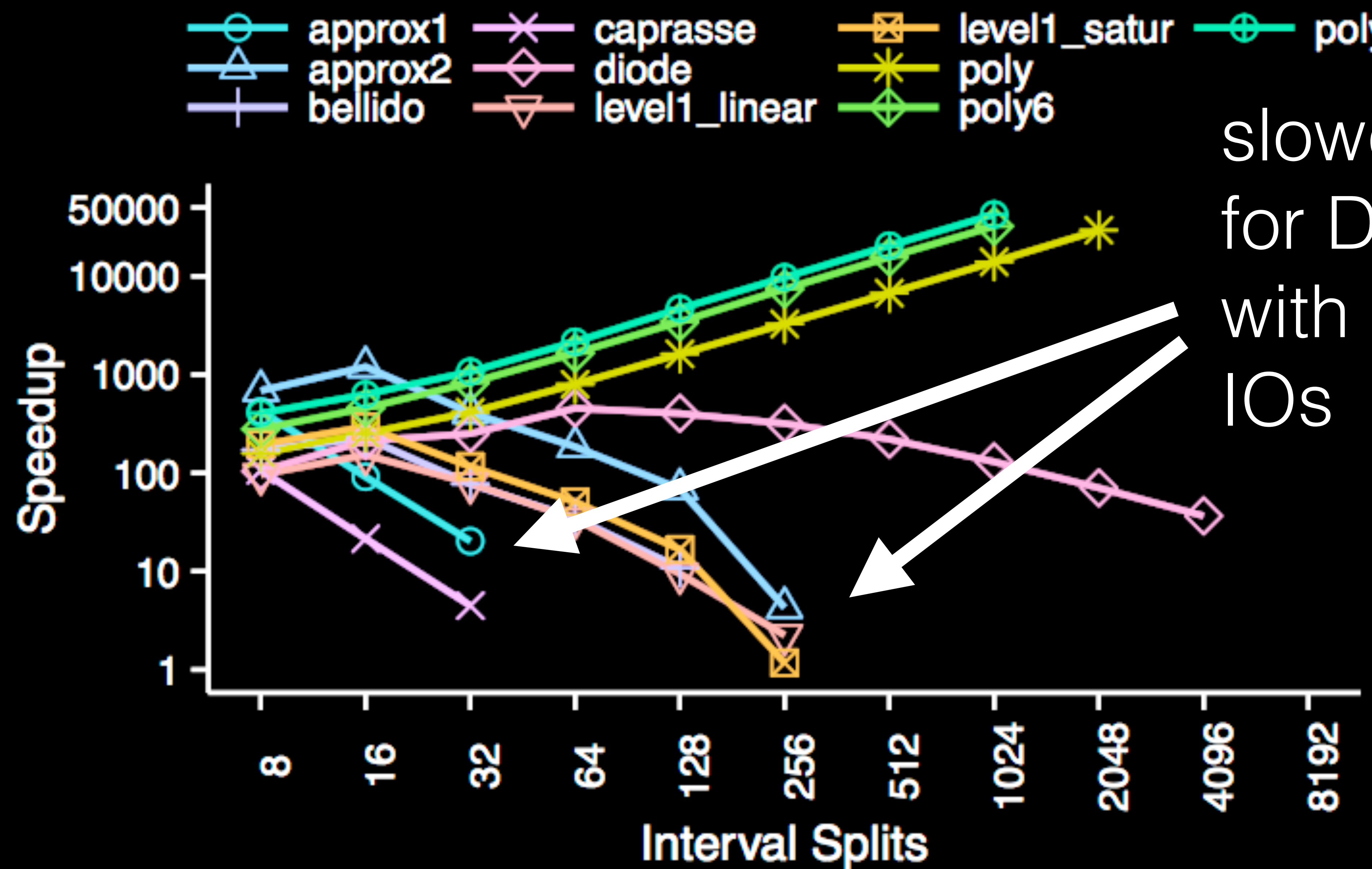
vs. Gappa dichotomy search

Sub-interval arithmetic



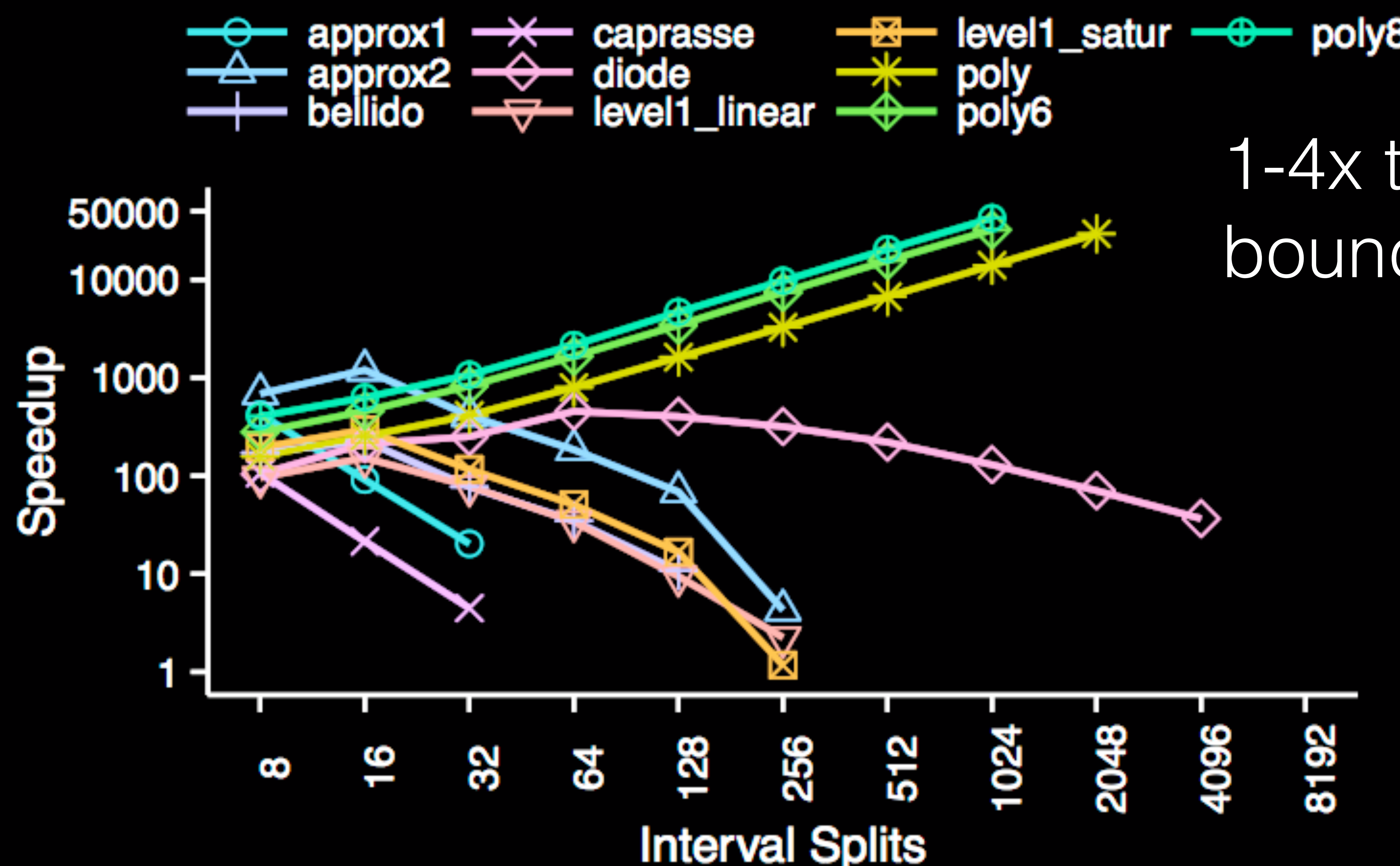
scalable speedups

Sub-interval arithmetic



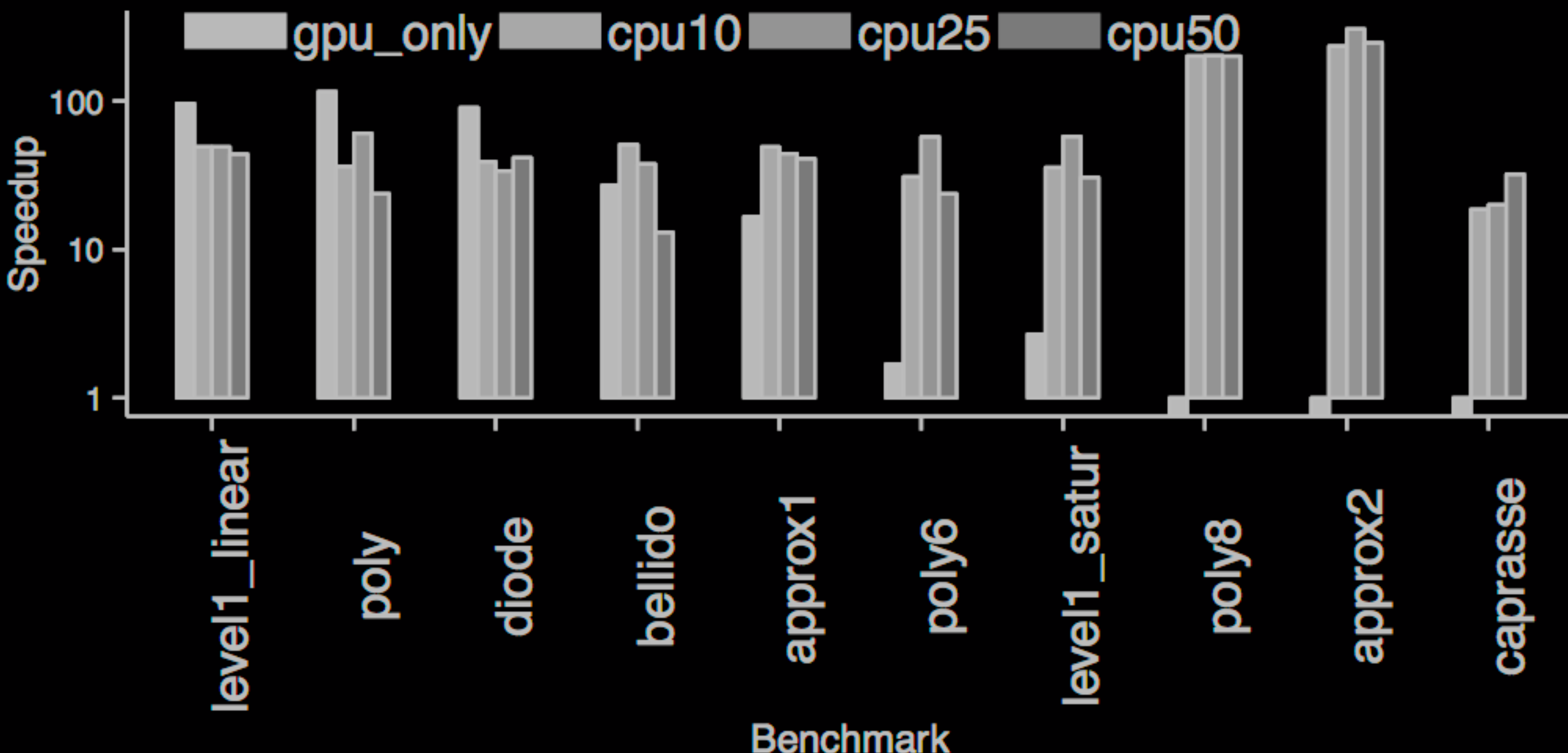
slowdown for DFGs with more IOs

Sub-interval arithmetic

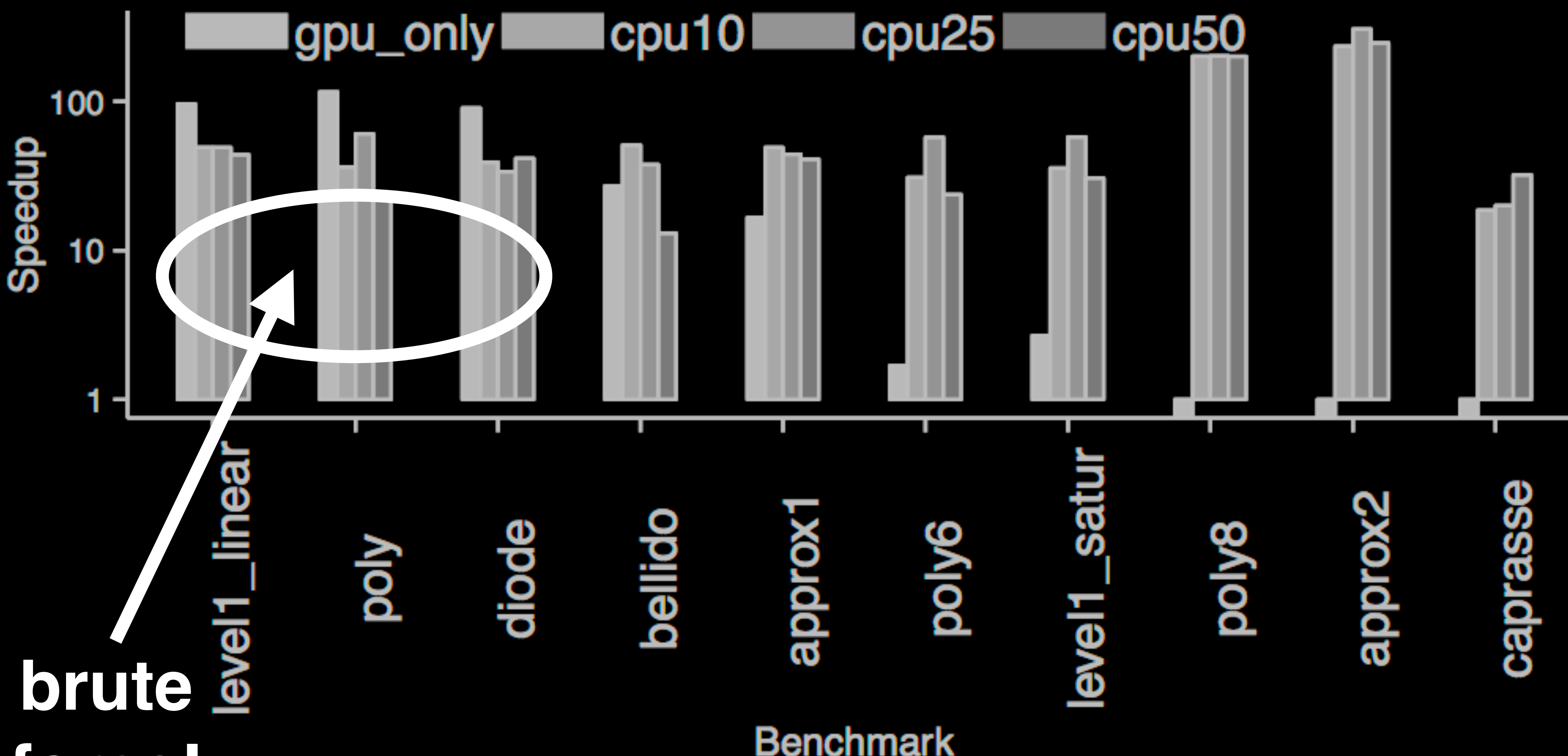


1-4x tighter bounds!!

Bitwidth Optimisation

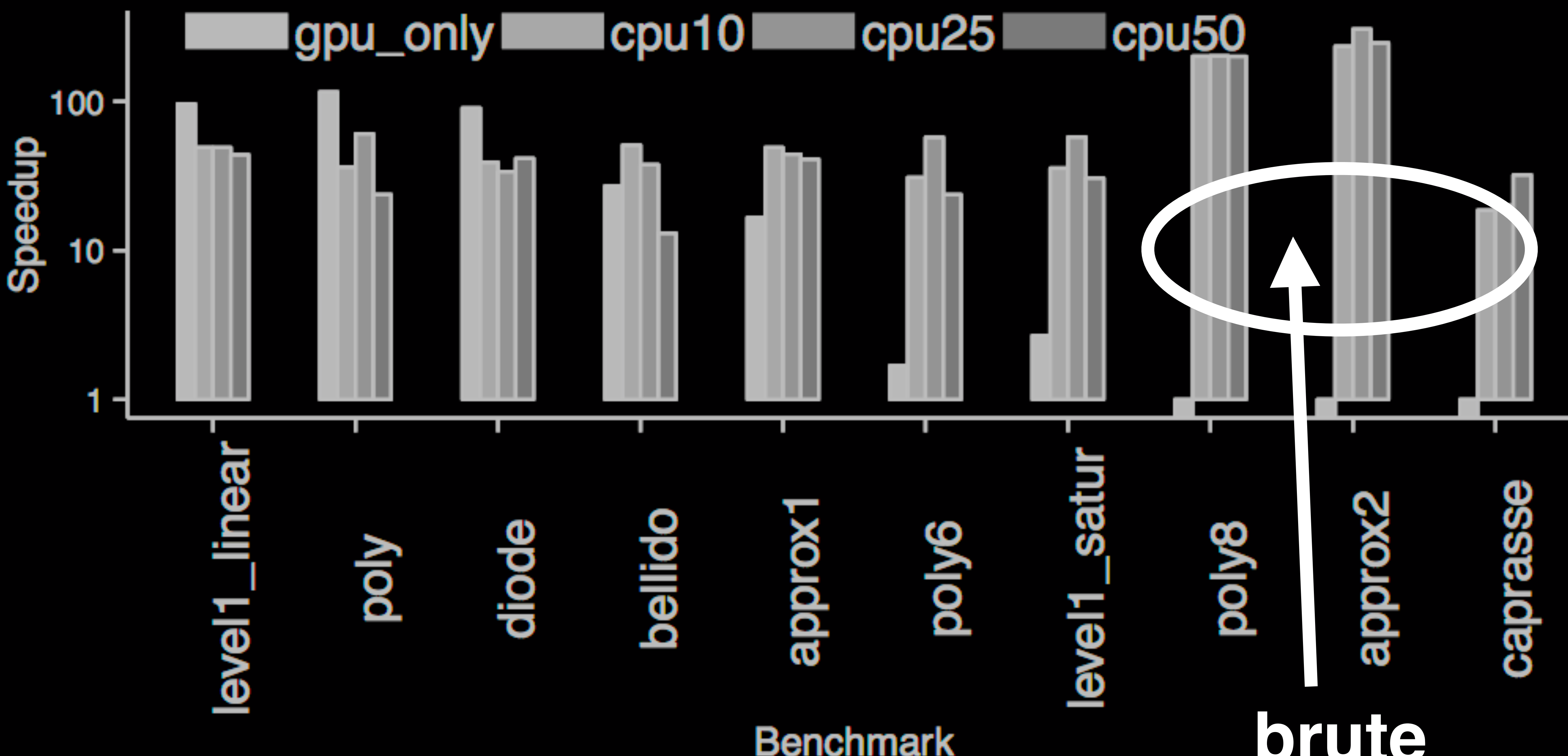


Bitwidth Optimisation



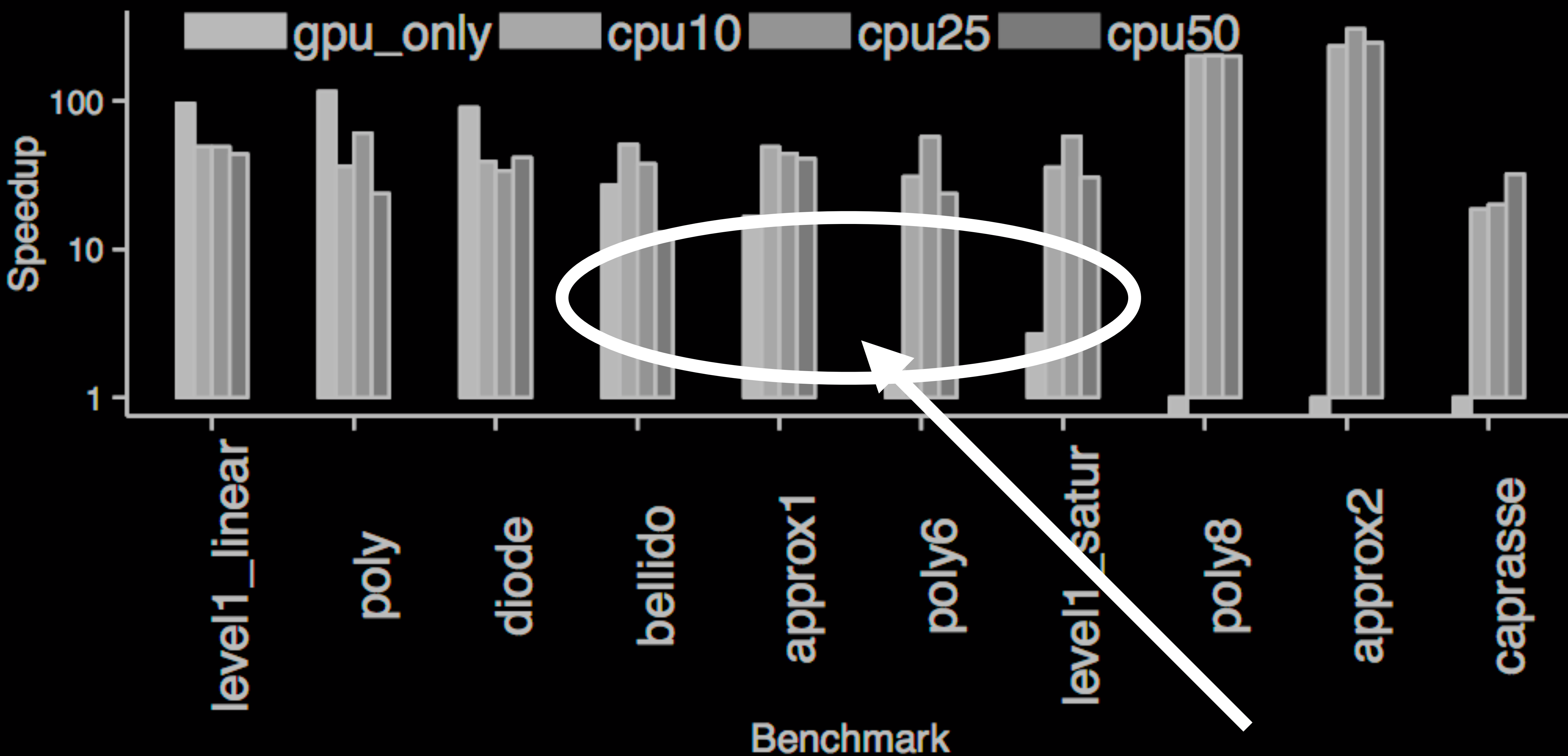
brute force!

Bitwidth Optimisation



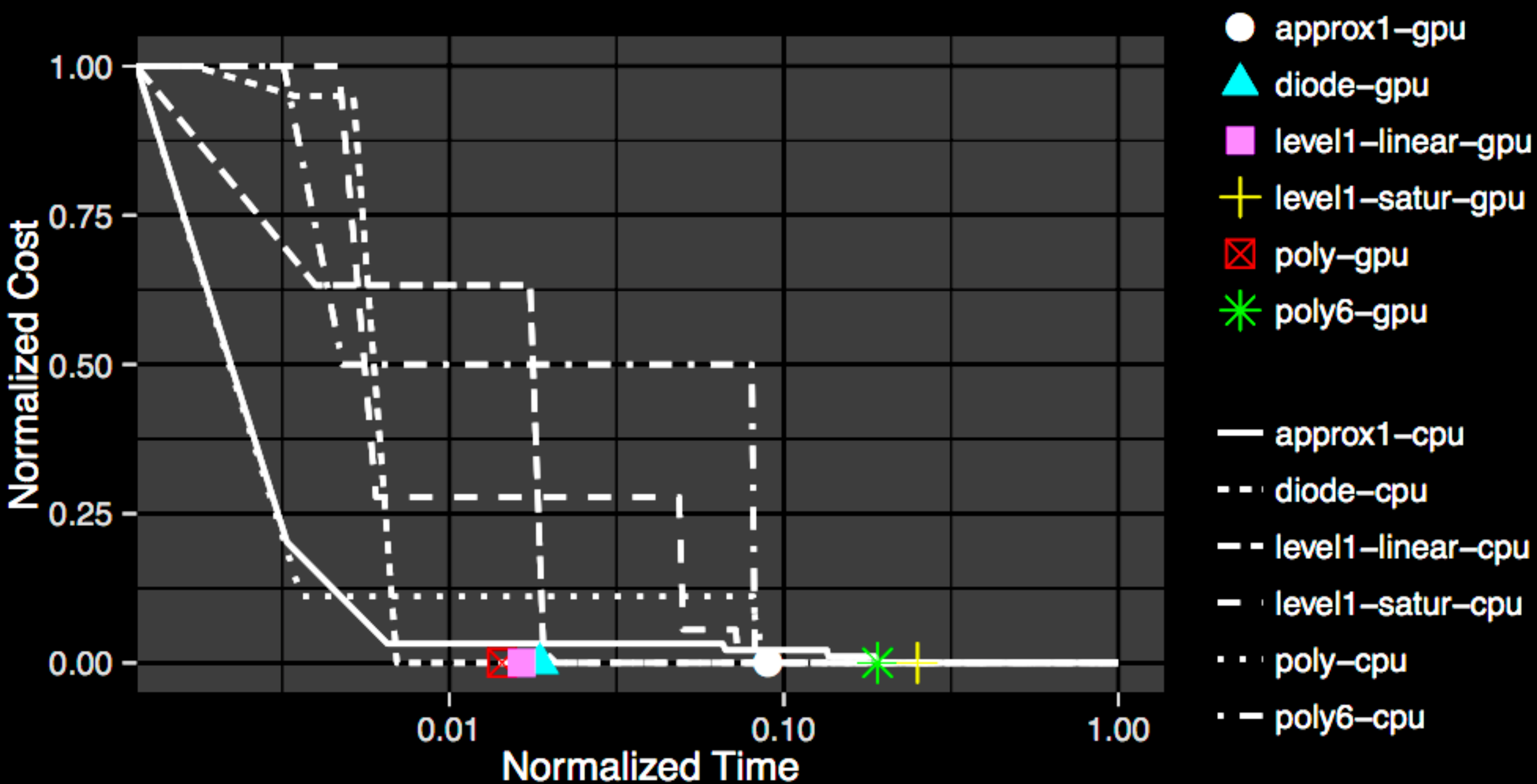
brute force infeasible

Bitwidth Optimisation

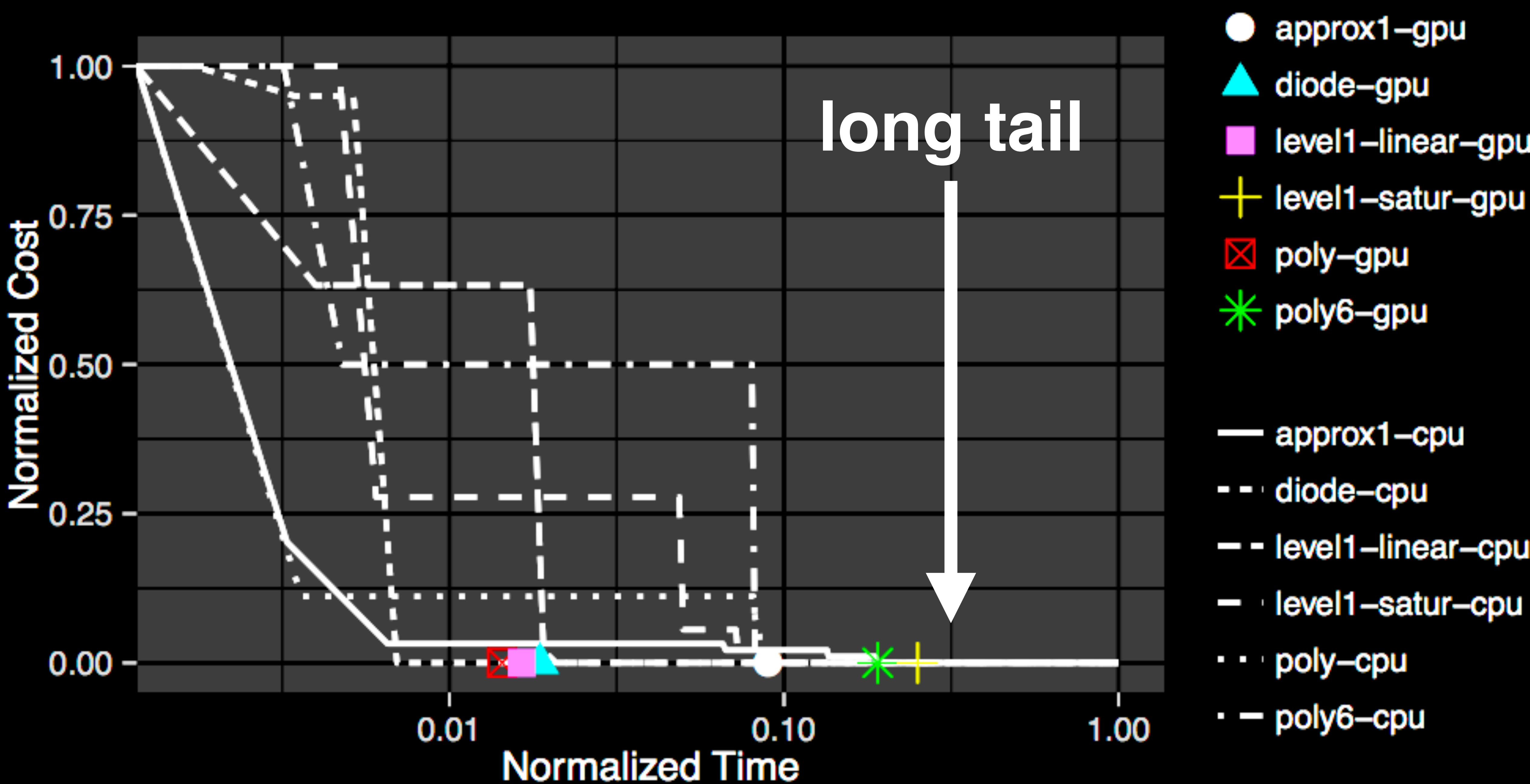


brute force slow

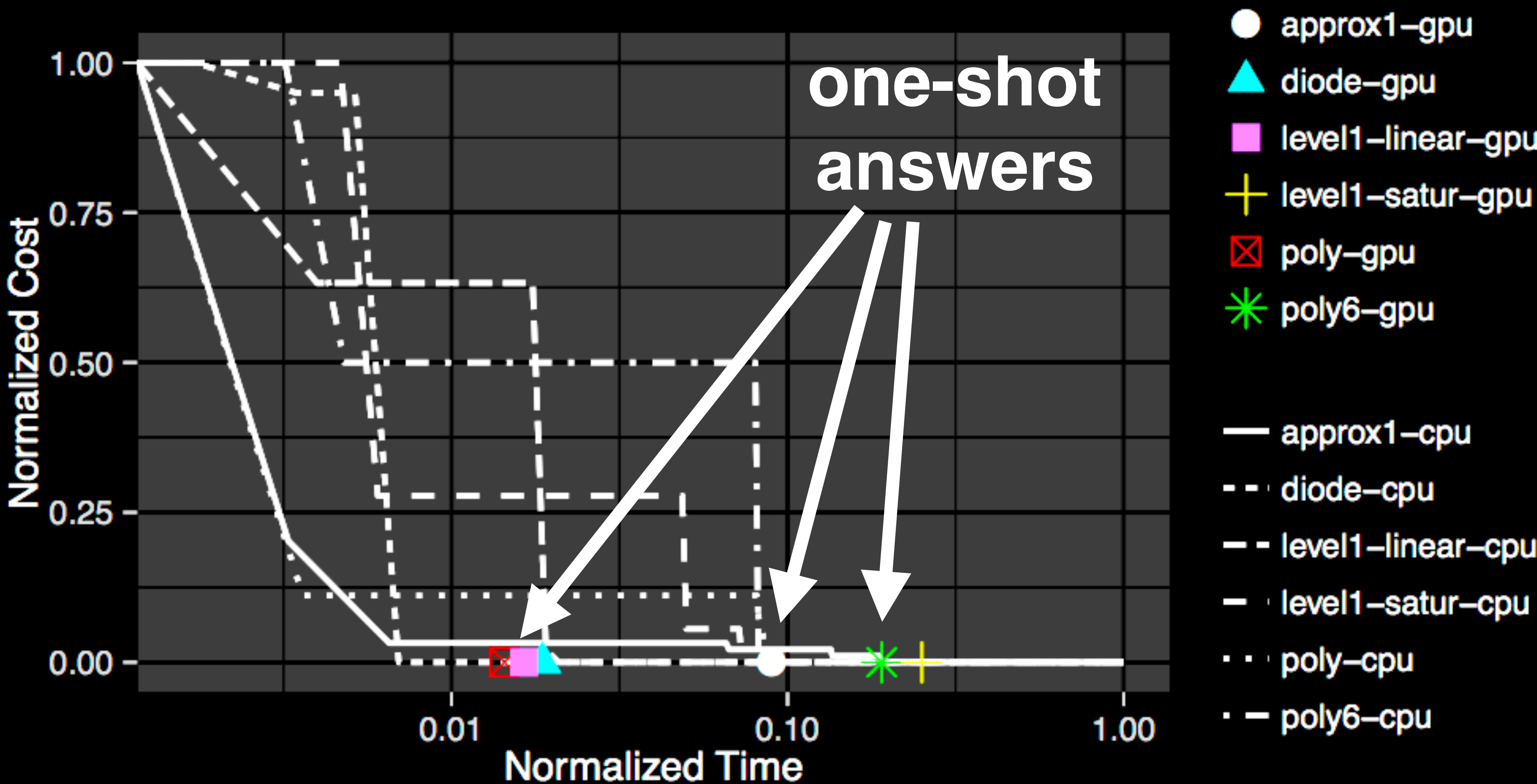
Quality-Time Tradeoffs



Quality-Time Tradeoffs



Quality-Time Tradeoffs



Conclusions and Projections

Roundup

- GPUs can help accelerate FPGA CAD (bitwidth optimization)
  — 100x+ for sub-intervals
  — 10—100x for bitwidth allocation

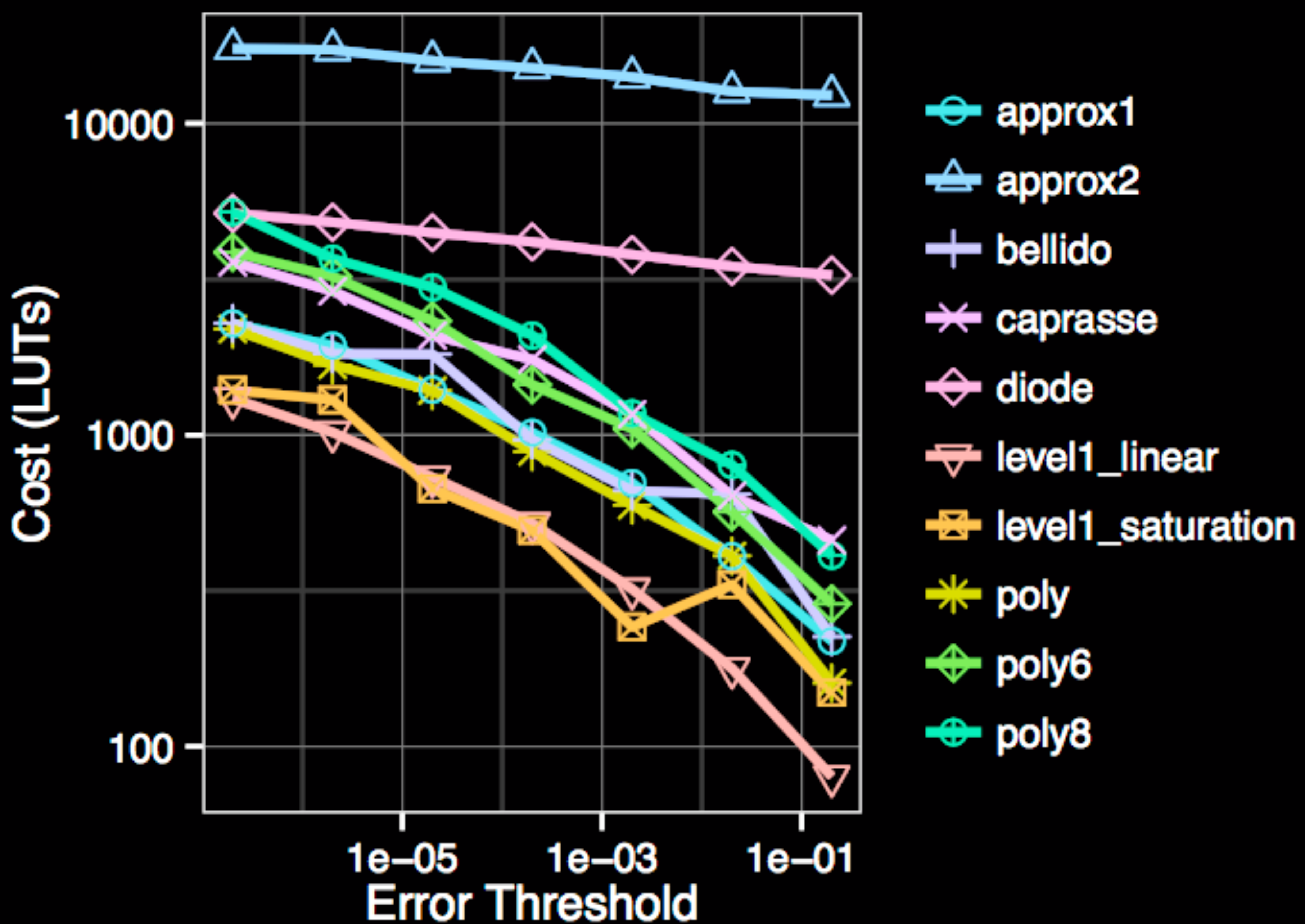
- **PRUNE+BRUTE** philosophy
  — be prepared to do more work
  — GPUs not just about speed —> optimality

Source Code — http://yedeheng.github.io/bitgpu/

Future Work

- Do more work per GPU thread, only save best, local merge operations — better use of GPU threads
- Affine analysis formulations for GPU parallelism — potentially improve accuracy, converge faster with fewer sub-interval splits
- Modified/parallel Monte-Carlo approaches for covering search space
  — no need to cover every single instance
- Think about **prune+brute** strategy for other CAD problems

Varying Error Threshold



Fidelity of FPGA models