ESSENT:
A High-Performance RTL Simulator

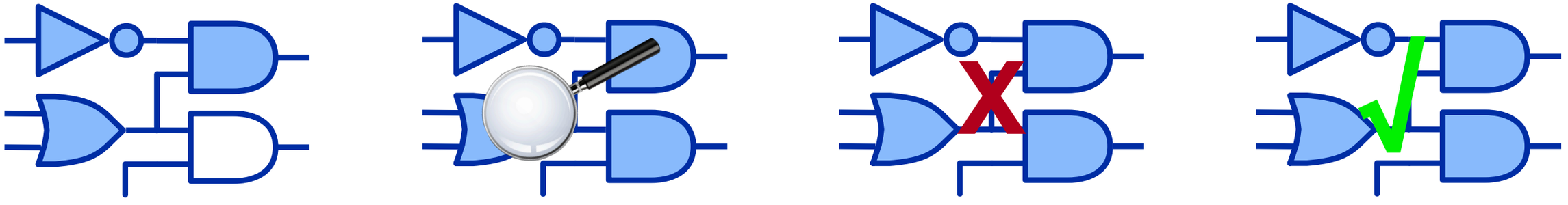
Scott Beamer Thomas Nijssen
Krishna Pandian Kyle Zhang

Computer Science & Engineering Dept.
University of California, Santa Cruz

https://github.com/ucsc-vama/essent

Simulation Speed Limits HW Design

- Simulation is a crucial tool for HW design



Accelerate Simulation Rate

+ Faster interactions with human users
+ Reduce costs by saving on simulation infrastructure
+ Enables larger design space for better QoR

Alternatives to Software Simulation

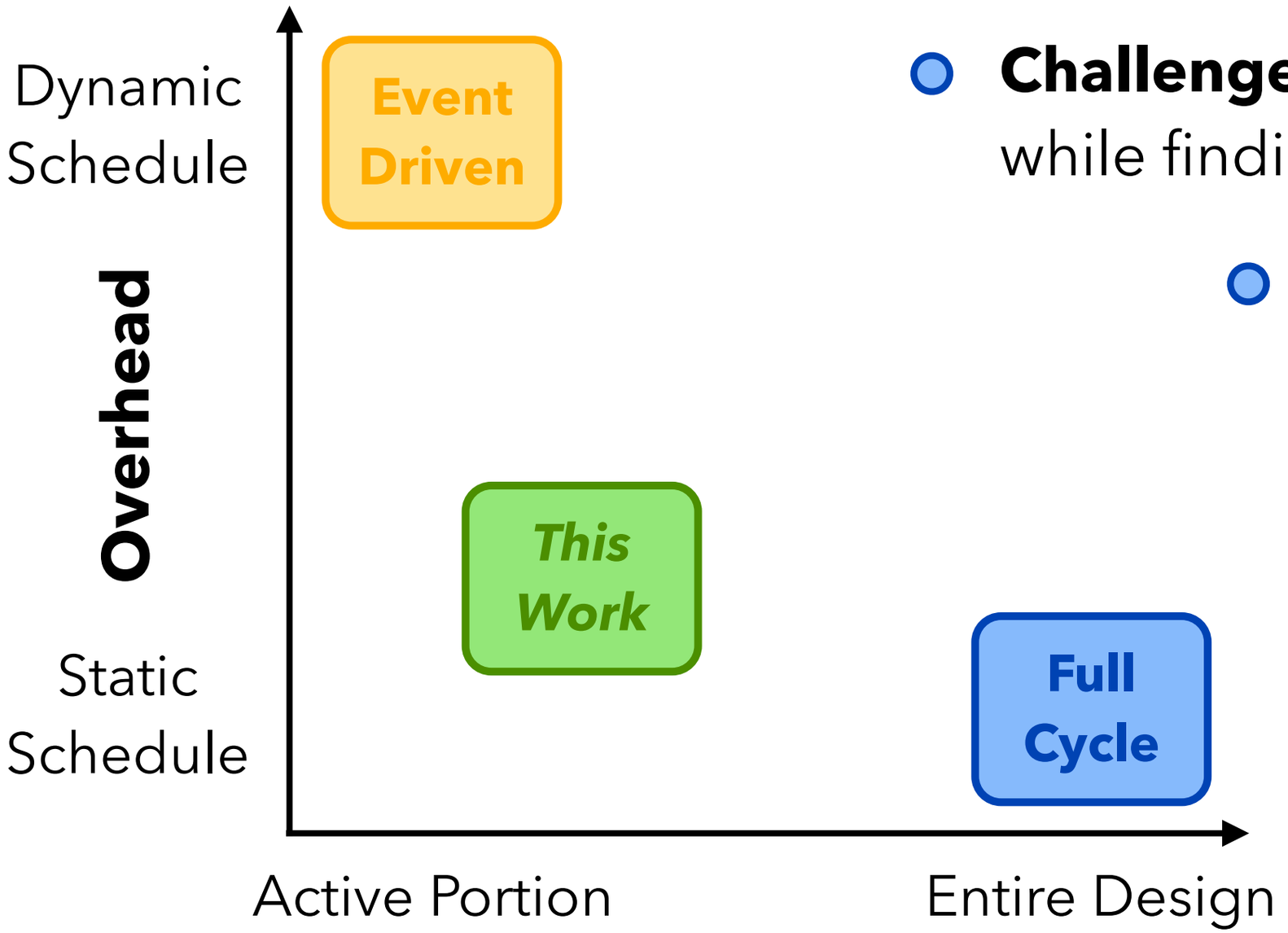
- Reduce fidelity w/ *transaction-level* models
  - SystemC
- Use *hardware acceleration*
  - FPGA simulation (e.g. FireSim/MIDAS)
  - Custom ASICS (e.g. Palladium)

- Above approaches are *complimentary*, but cycle-accurate SW simulation is still commonly used because of faster start-up and lower up-front costs
  - SW also great for agile / open-source design

Introducing ESSENT

- *Essential Signal Simulation Enabled by Netlist Transformations (ESSENT)*
- Software RTL simulator focussed on speed
- Takes FIRRTL as input, emits C++ for simulator
  - FIRRTL produced by Chisel, LiveHD, Yosys
- Uses various optimization to save work while still producing cycle-accurate result
- Enabled by efficient & user-friendly partitioner
- https://github.com/ucsc-vama/essent

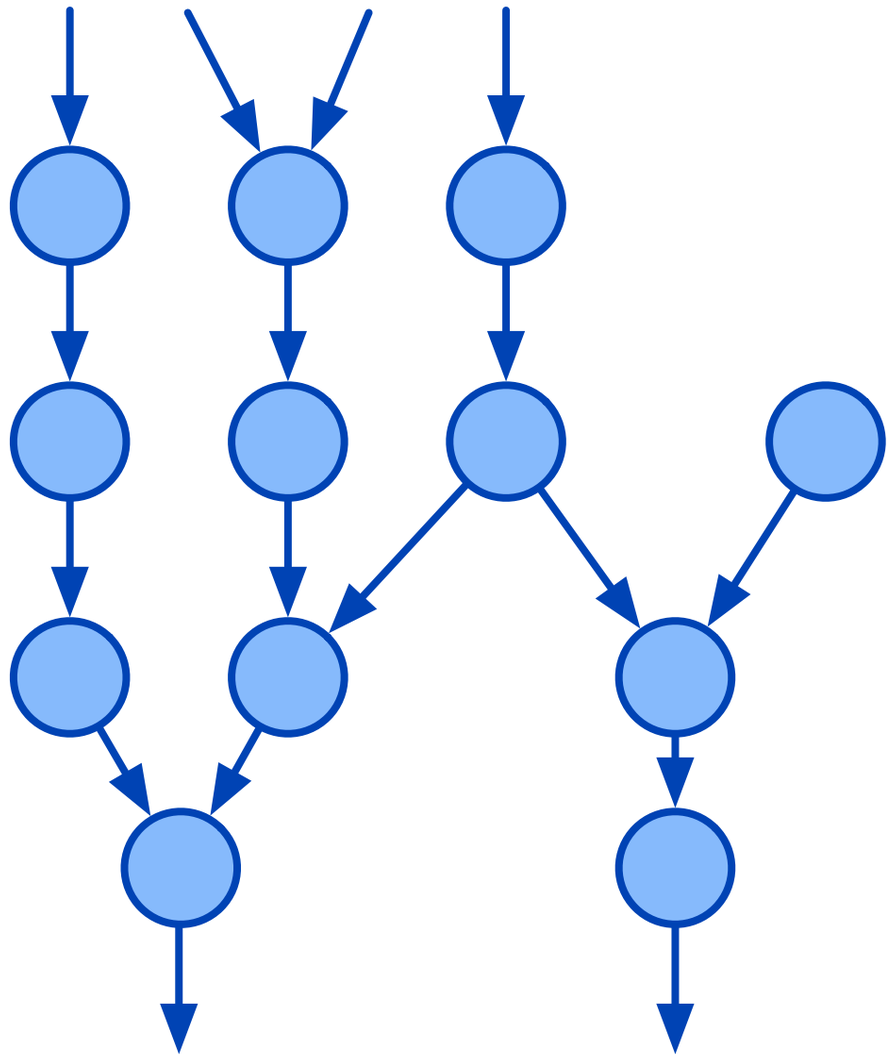
RTL Simulation Approach Tradeoffs



- **Insight:** Most signals rarely change, so should *reuse* unchanged values
- **Challenge:** How to reduce overhead while finding opportunities for reuse
  - **Summary of this Work:** *Low-overhead* techniques to reduce the fraction of the design simulation to speed up simulation

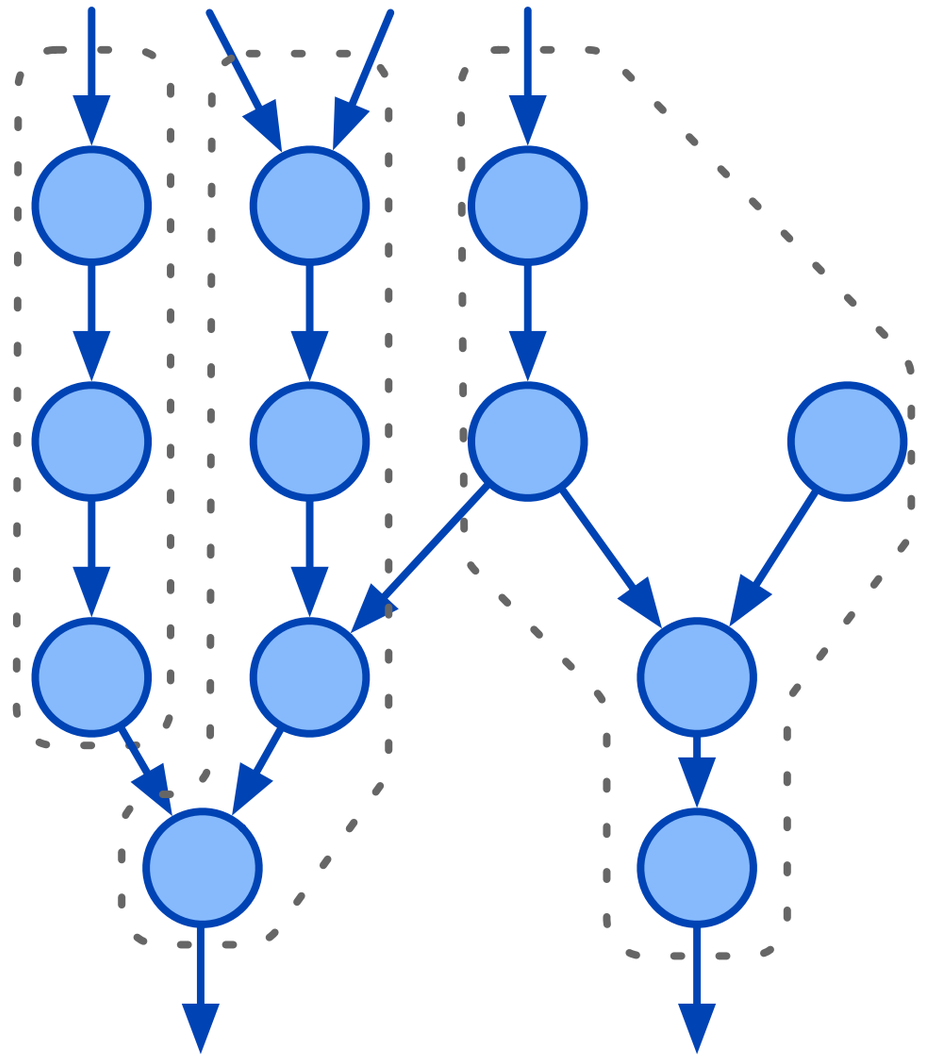
Reuse Example

- Leverage low-activity by reusing outputs if inputs unchanged



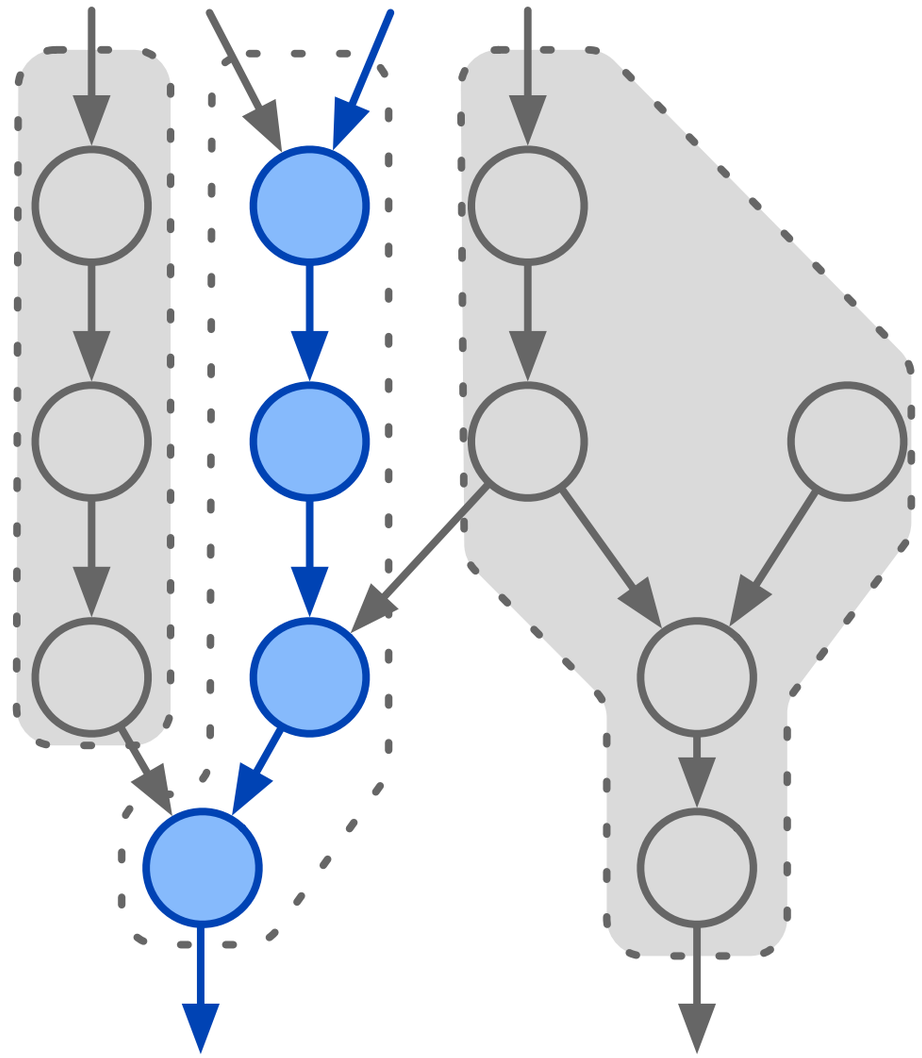
Reuse Example

- Leverage low-activity by reusing outputs if inputs unchanged
- Reduce overheads by coarsening reuse granularity



Reuse Example

- Leverage low-activity by reusing outputs if inputs unchanged
- Reduce overheads by coarsening reuse granularity



Key Features of Efficient Simulation

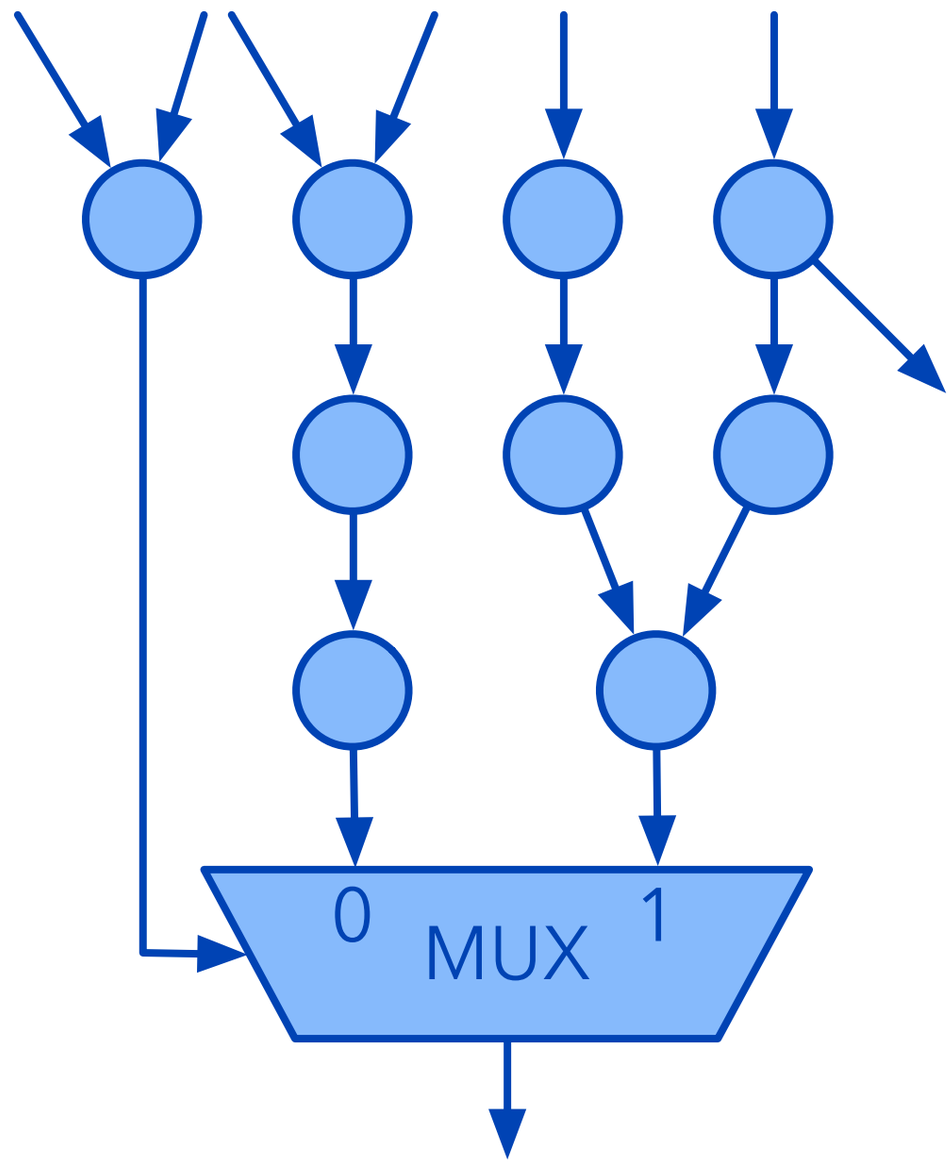
- *Conditional*
  - Only evaluate signals if their inputs change
  - Otherwise, reuse old outputs
- *Coarsened (needs partitions)*
  - Amortize overheads over multiple signals
- *Singular (needs acyclic)*
  - Evaluate each signal at most once per cycle
- *Static*
  - Perform scheduling in advance at compile time
- All combined, is our *CCSS Approach*

Automated Code Generation

Acyclic Partitioning

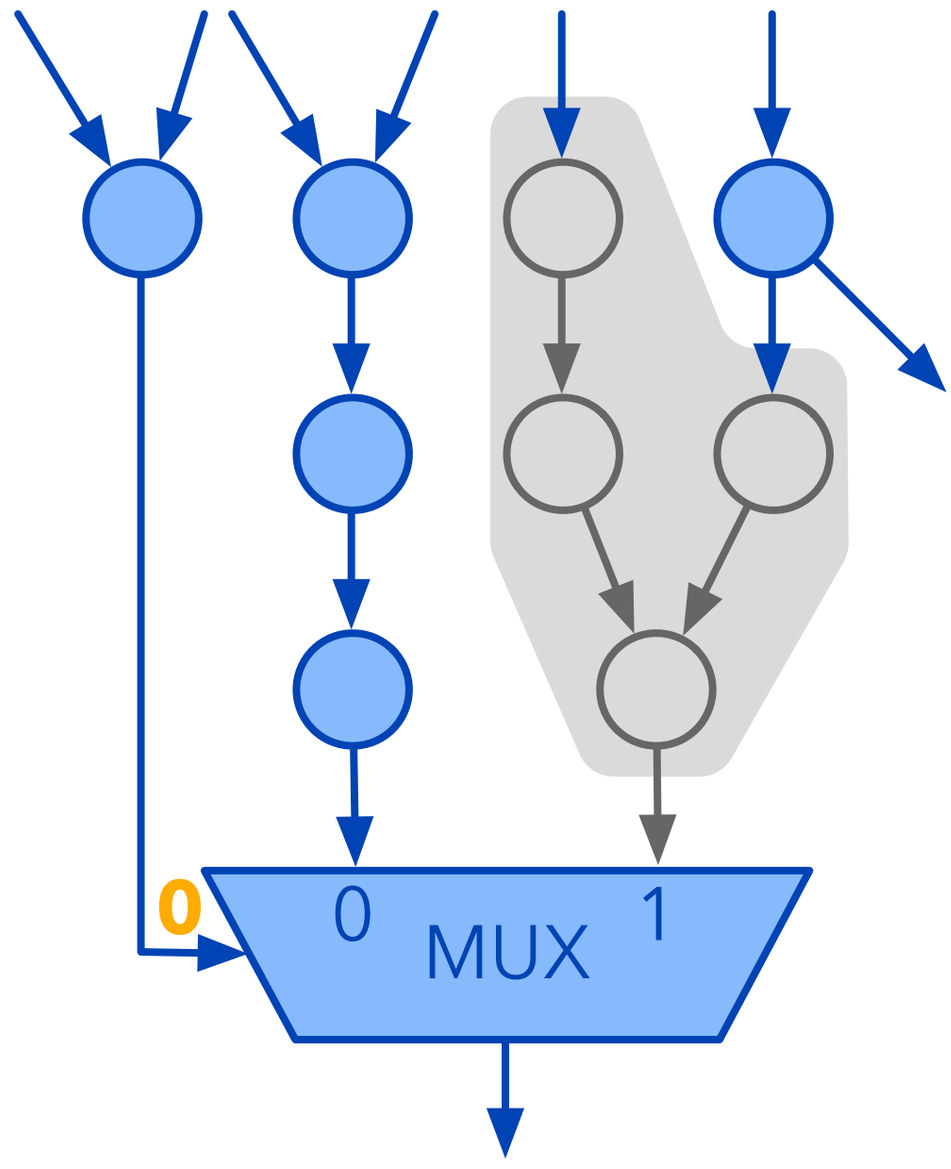
Other ESSENT Optimizations

- Don’t simulate signals who will not persist (unselected mux ways)



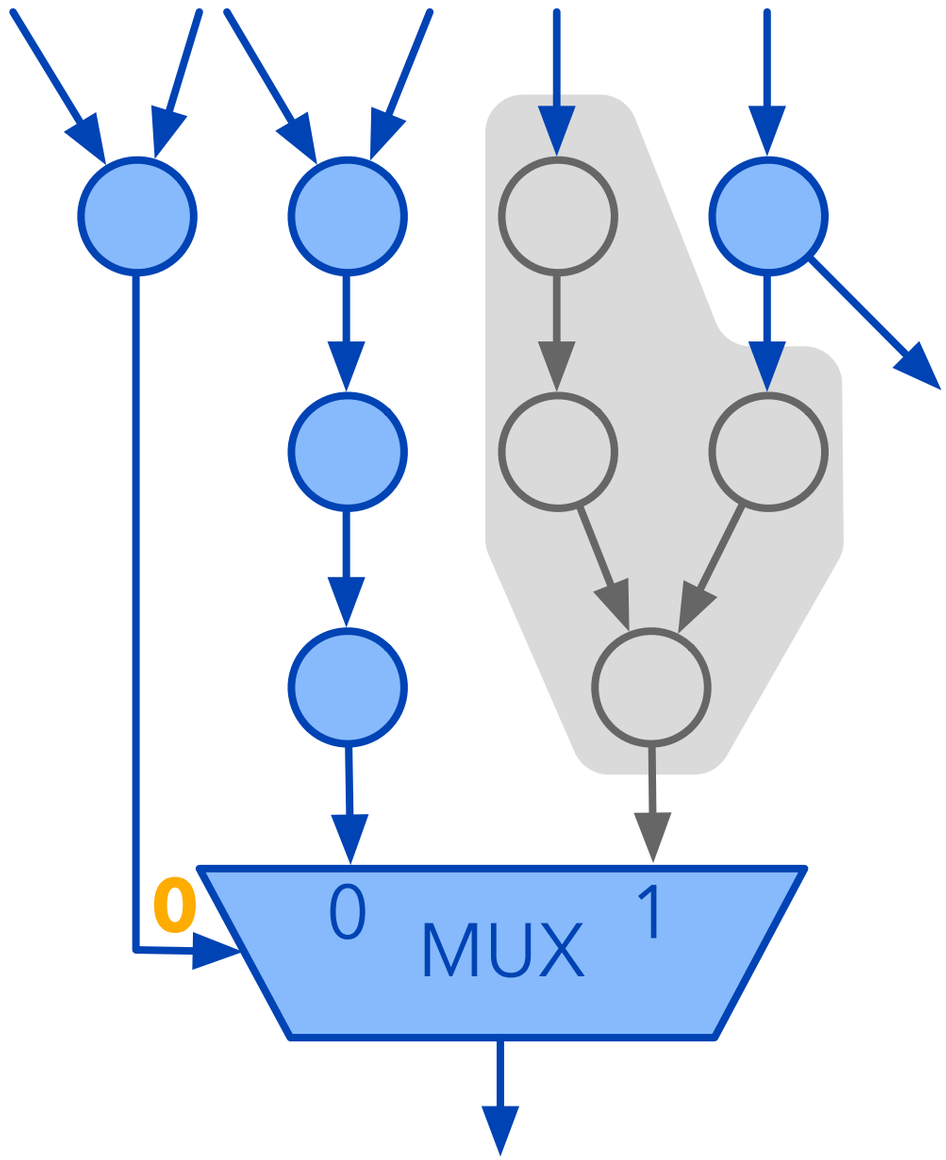
Other ESSENT Optimizations

- Don’t simulate signals who will not persist (unselected mux ways)



Other ESSENT Optimizations

- Don’t simulate signals who will not persist (unselected mux ways)
- Elide intermediate register updates, even in partitions
- Give compiler branch hints for unlikely paths (asserts, prints, reset)



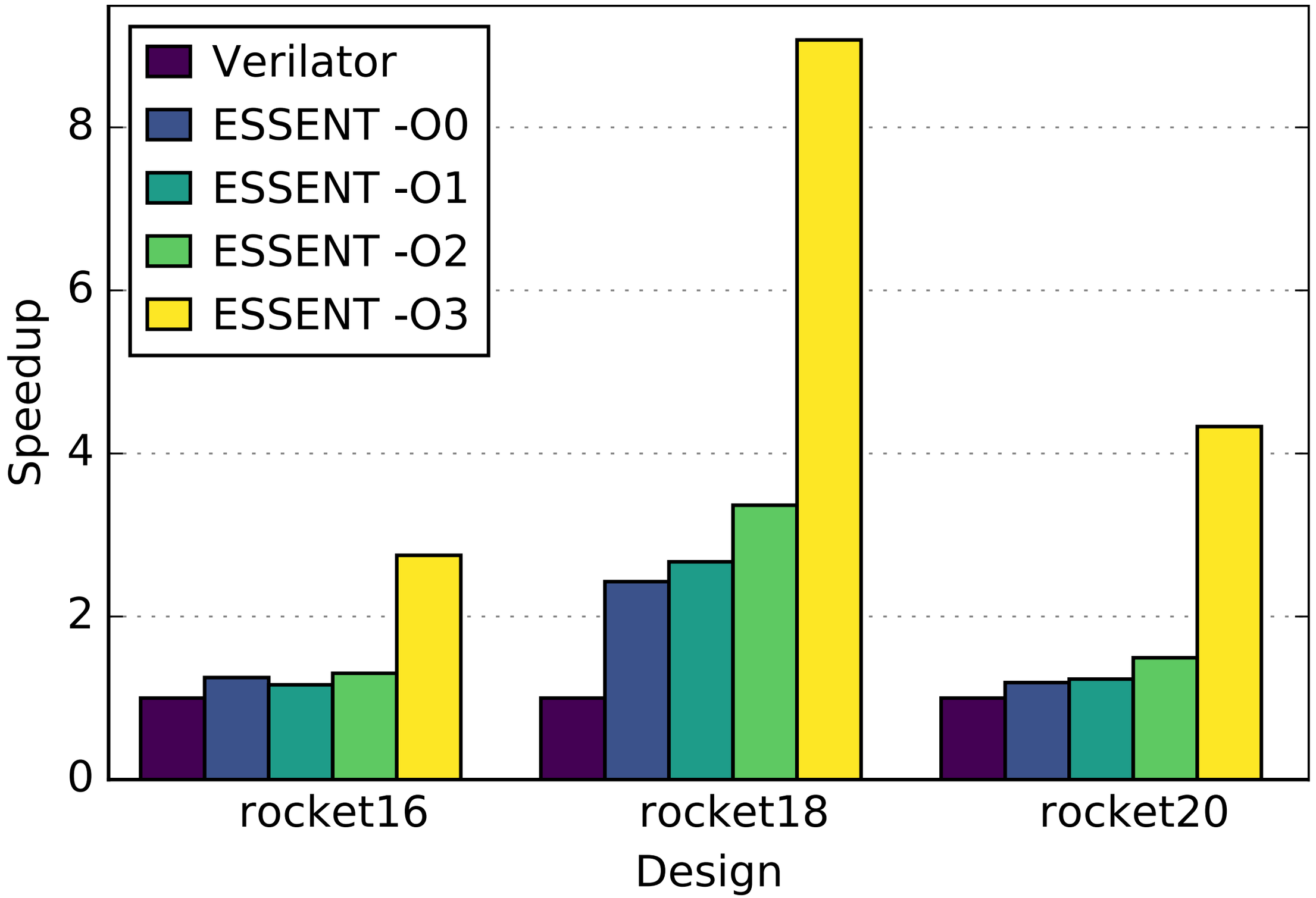
Evaluation Methodology

- ***Workload:*** Rocket Chip (RISC-V SoC) from different years executing `dhrystone`

Name (year)	Verilog (lines)	FIRRTL (nodes)	FIRRTL (edges)
rocket16	112,167	26,554	47,290
rocket18	328,367	71,545	123,226
rocket20	246,589	70,349	120,236

- ***Simulators:*** ESSENT (with varying optimization level) vs. Verilator
- ***Host Platform:*** 8-core 3.6 GHz Intel Skylake

Performance Comparison



ESSENT is Open Source!

- https://github.com/ucsc-vama/essent
- Includes example integrations with Rocket Chip & riscv-mini
- Useful for simulation and as a starting point for future simulation research
- Written in Scala to reuse FIRRTL library and passes
- Released C++ arbitrary width signal library: https://github.com/ucsc-vama/firrtl-sig

Ongoing & Future Work

- Declare version 1.0 (very close)
- Improve tool scalability with faster generation time (better algorithms) and multicore-parallel simulation
- Create more example integrations
- Improve interoperability with other tools
  - SST (for mixed simulation)

Conclusion

- ESSENT is a fast cycle-accurate simulator
  - Executes fewer instructions by leveraging low activity that is common in designs
  - Enabled by CCSS approach & partitioner
- ESSENT is open source and eager to work with your project https://github.com/ucsc-vama/essent
- Contact Scott Beamer (sbeamer@ucsc.edu)

Acknowledgements

- Contributors & Collaborators
  - Lawrence Berkeley National Laboratory (LBL)
    - David Donofrio
    - John Bachan
  - University of California, Santa Cruz (UCSC)
    - Thomas Nijssen
    - Krisha Pandian
    - Kyle Zhang
- Funding
  - Army Research Office

Publications

- A Case for Accelerating Software RTL Simulation
  Scott Beamer
  IEEE Micro, 2020

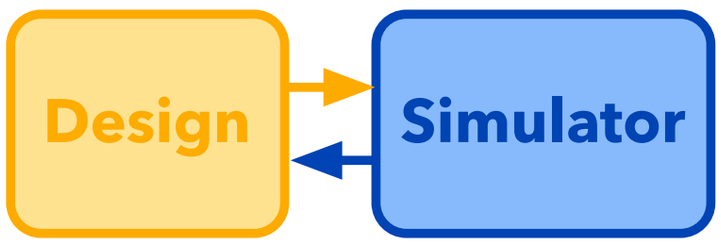
- Efficiently Exploiting Low Activity Factors to Accelerate RTL Simulation
  Scott Beamer and David Donofrio
  Design Automation Conference (DAC), 2020

Backup Slides

Classic Simulation Approaches

Event-Driven

- Substantial scheduling overhead for tracking priorities
- Activity-proportional

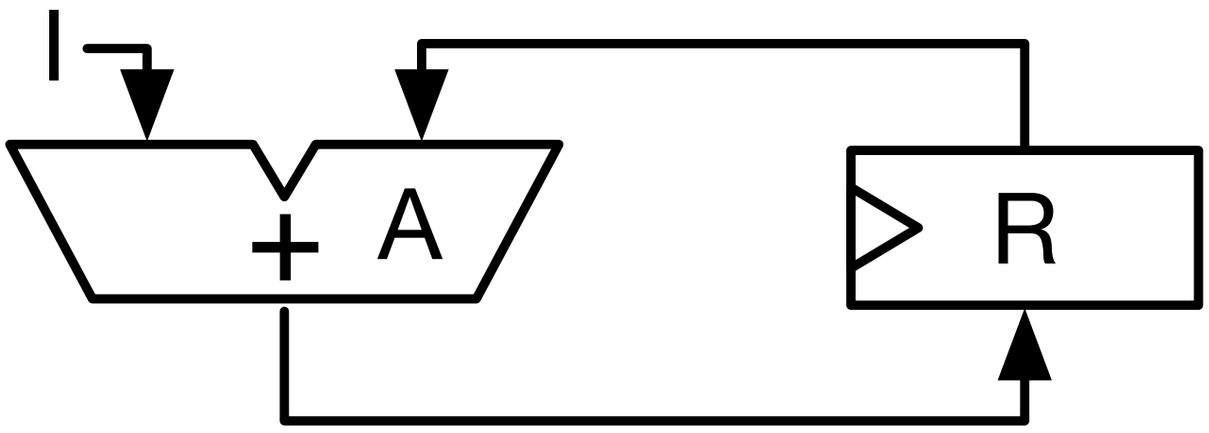


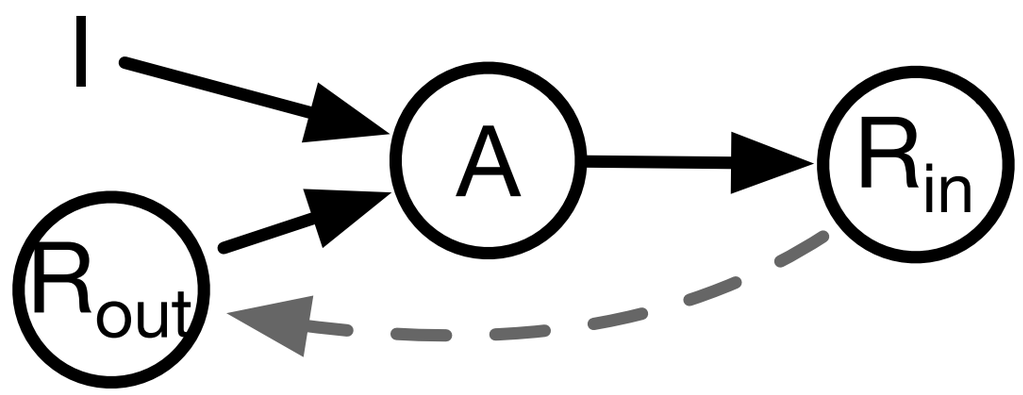
Full-Cycle

- Eliminates overhead with static schedule (requires acyclic)
- Activity-agnostic

- Despite commonly low activity factors, in practice, full-cycle is typically faster because of reduced overheads

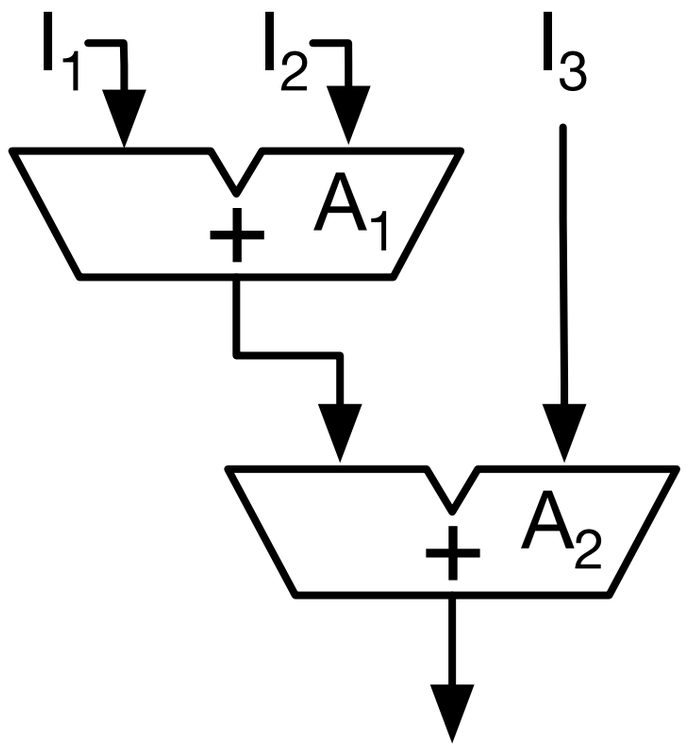
Graph Abstraction Simplifies Problem

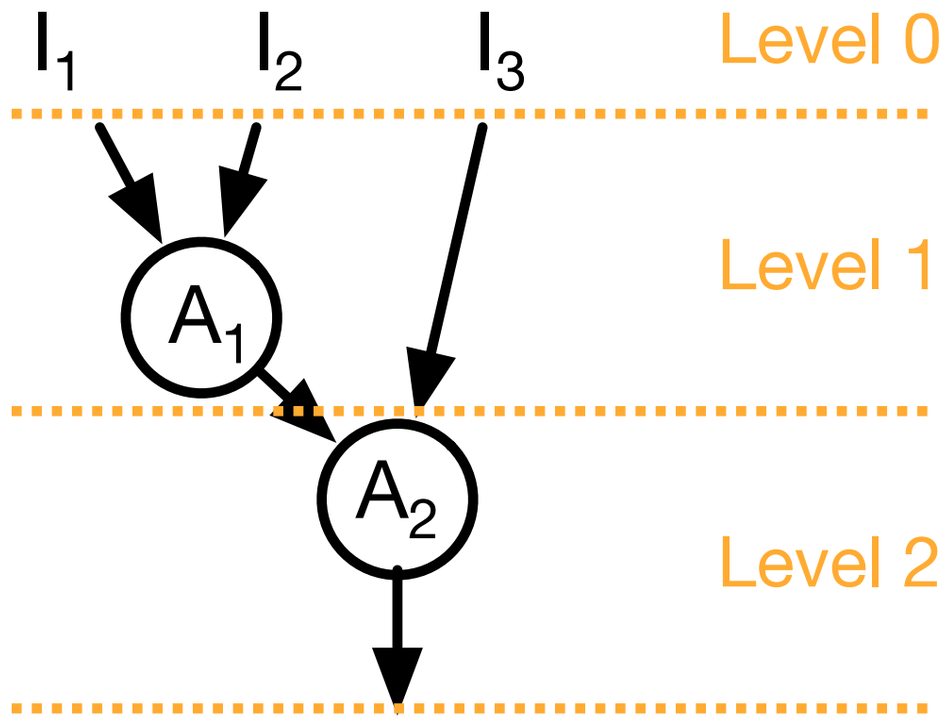




- View hardware design as a *directed graph*
- For today, design graphs are *acyclic*
  - No combinational loops allowed
  - Break up feedback path through registers by splitting them into inputs & outputs

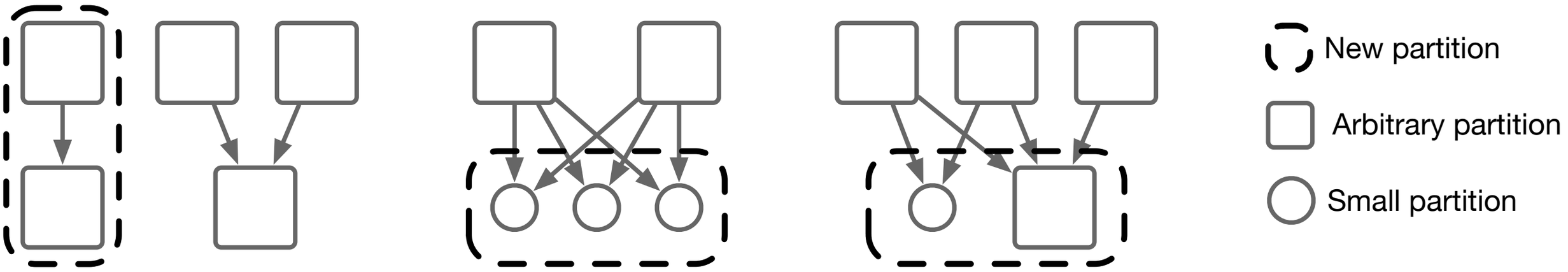
Scheduling is Big Part of Simulation





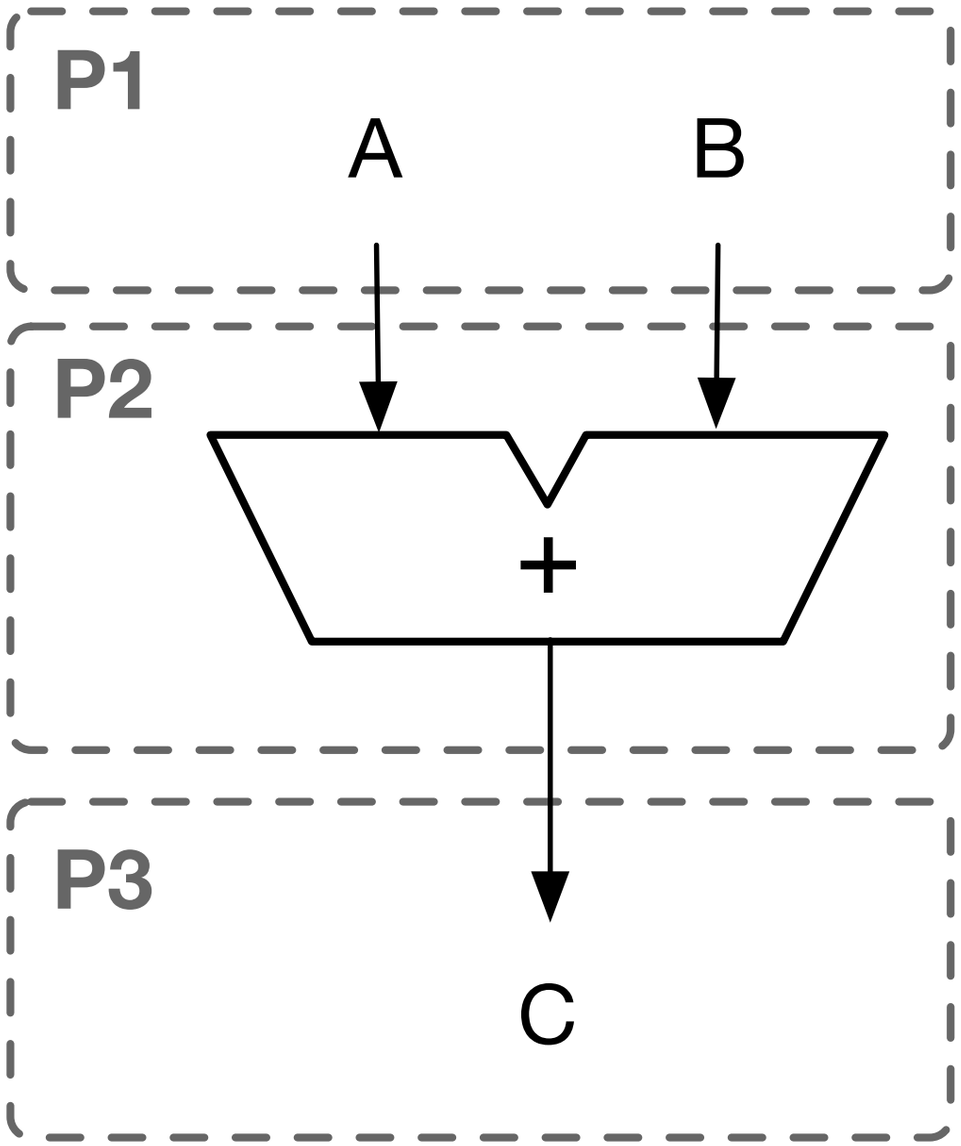
- Want to schedule work to avoid evaluating same HW more than once per cycle
- *Levelization* - evaluate an entire level before advancing (like BFS)

Novel Acyclic Partitioner



- *Key Contribution:* **Novel Acyclic Partitioner**
  - Focus on speed and specifics of this problem
  - User doesn’t need to modify design or provide parameters
- *Algorithm:* Greedily merges partitions until sufficiently coarse
  - We propose safety criteria & heuristics to select good merges
  - Bootstrap with maximum fanout free cones (MFFC)

Example High-Level Generated Code




```
…
evalP2()
   active[2] = false
   C$old = C
   C = A + B
   active[3] |= C != C$old
…
eval()
   …
   if (active[2])
     evalP2()
   …

main()
   for (max_cycles)
     eval()
```