Delay-Bounded Routing for Shadow Registers

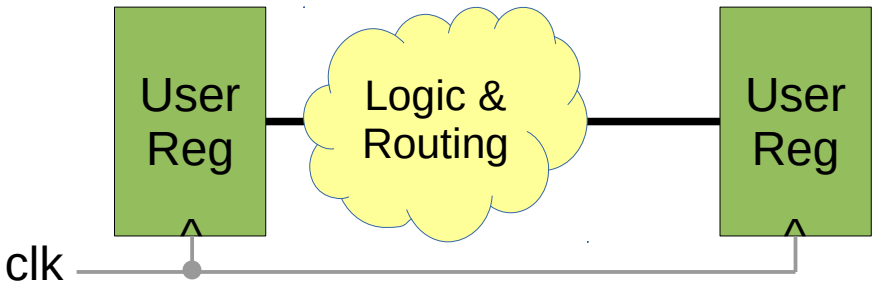
Eddie Hung, Josh Levine, Ed Stott,
George Constantinides, Wayne Luk
{e.hung,josh.levine,ed.stott,
g.constantinides,w.luk}@ic.ac.uk

Department of Computing
Department of Electrical and Electronic Engineering
Imperial College London, UK

FPGA15 :: 23 Feb '15

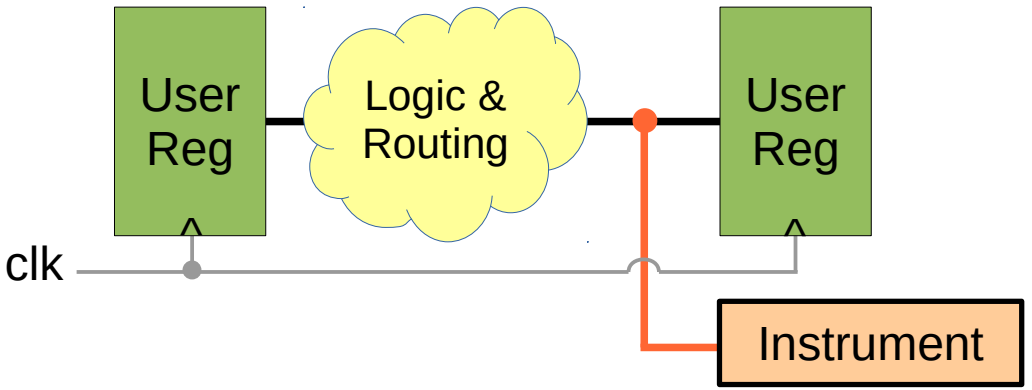
In a nutshell, this is about...

- More accurate on-chip timing measurements
  - By inserting instruments precisely



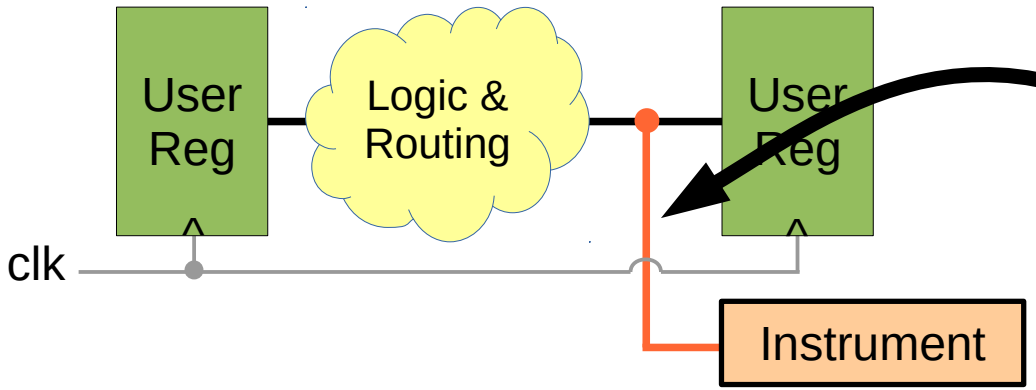
In a nutshell, this is about...

- More accurate on-chip timing measurements
  - By inserting instruments precisely



In a nutshell, this is about...

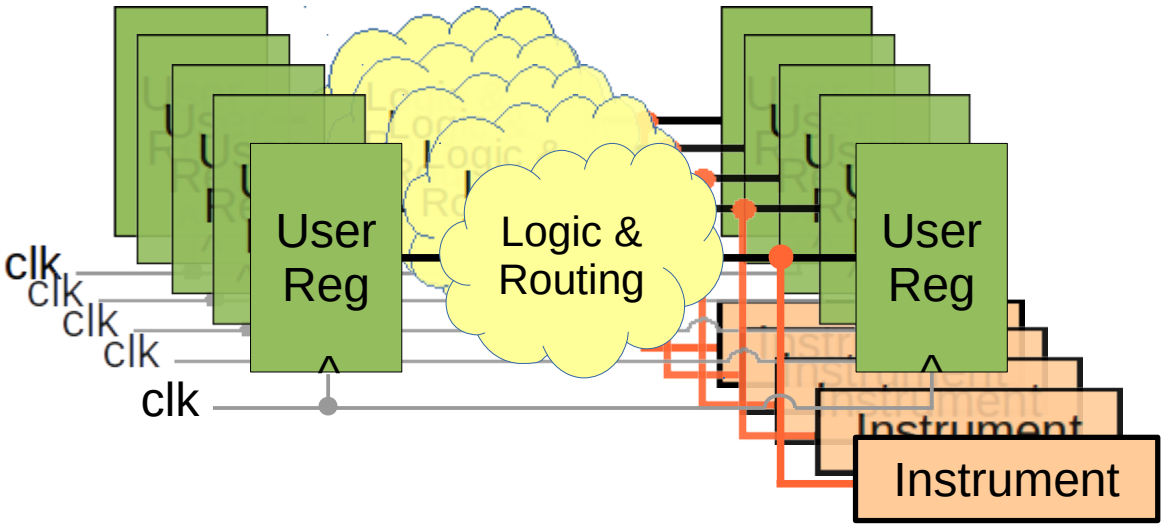
- More accurate on-chip timing measurements
  - By inserting instruments precisely



In particular, we want to bound this routing delay

In a nutshell, this is about...

- More accurate on-chip timing measurements
  - By inserting instruments precisely

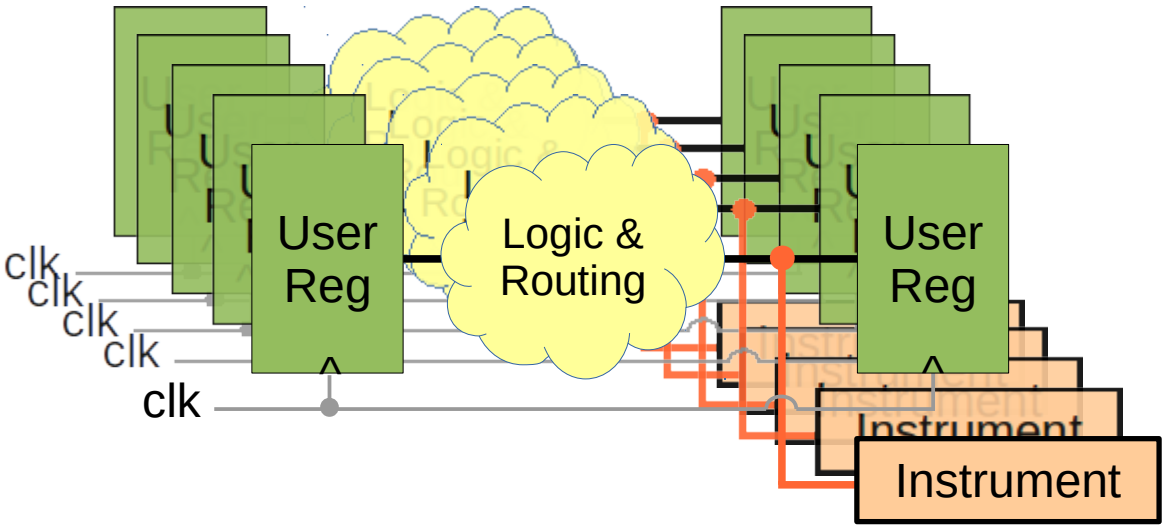


In particular, we want to bound this routing delay

<u>across 1000s of instruments</u>

In a nutshell, this is about...

- More accurate on-chip timing measurements
  - By inserting instruments precisely



In particular, we want to bound this routing delay

<u>across 1000s of instruments</u>

  - Most CAD focusses on minimising (or maximising)
    - We place and route instruments with bounded delay
- **Key result:** Insertion within 200ps avg. error

Introduction

- Why do we care about on-chip timing?
  - Because of variation, vendors forced to provision for all possible scenarios
  - Fmax is worst-case path, at worst-case timing corner
  - Can we safely operate better than worst-case?
    - “Why operate at the worst corner if silicon is rarely that slow, and if critical-path is rarely (never) exercised?”

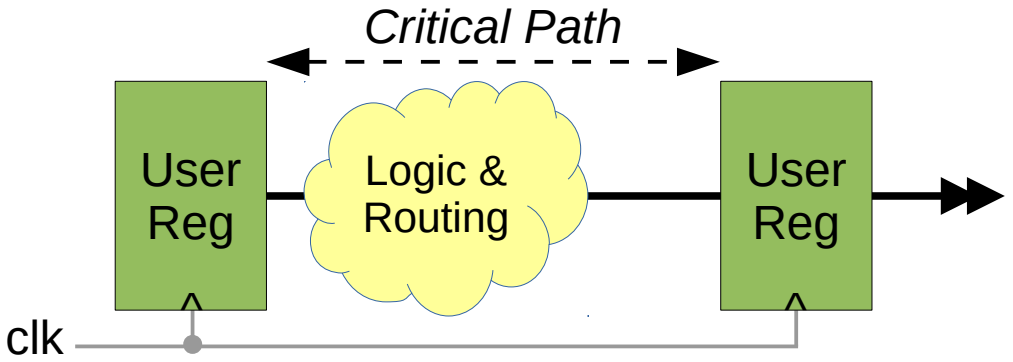
Main Contributions

- Proposal for inserting shadow registers post place-and-route, using spare resources
  - Does not disrupt original circuit timing
- Modification to Dijkstra's algorithm for minimum cost (delay) bounds
- Experimental evaluation on commercial architecture
  - Achieve average bound error of 200ps

Introduction

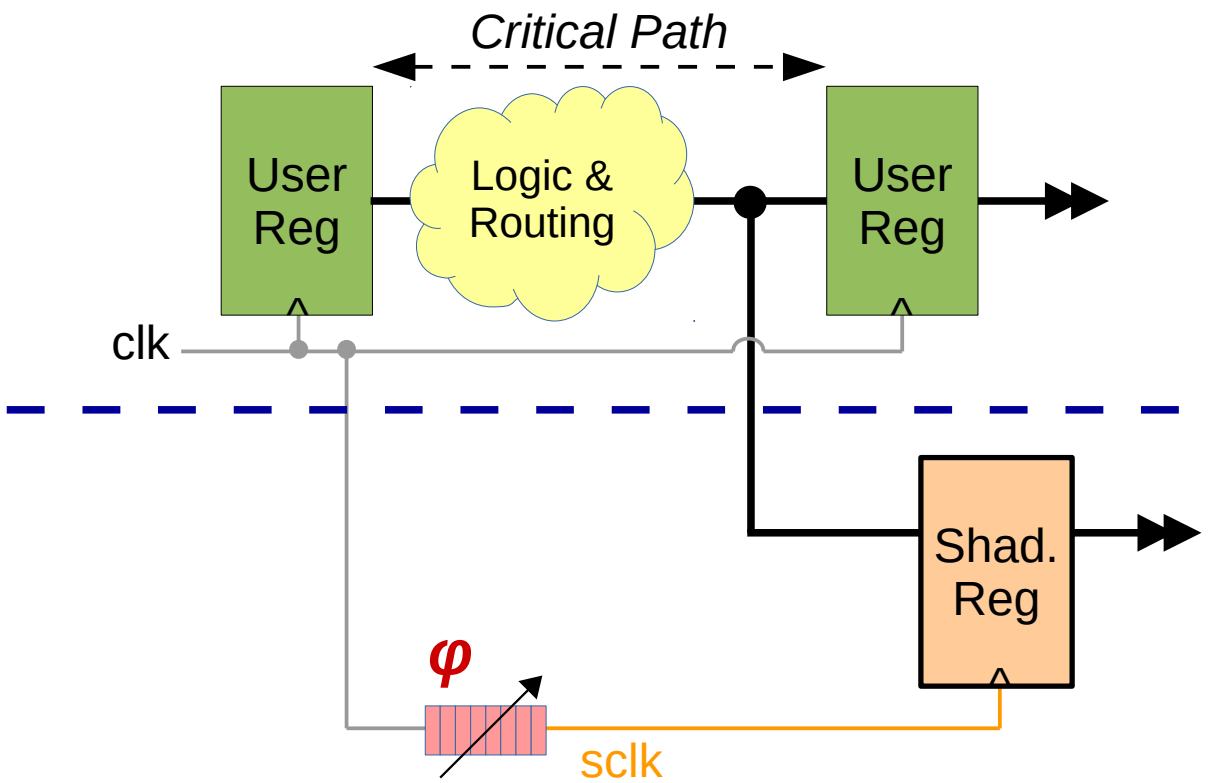
- How can on-chip timing instruments help?
  - Failure prediction
    - “Is my FPGA showing signs of being close to failing?”
  - Error detection
    - “Safe” overclocking (e.g. Razor)
  - Slack measurement
    - “How much leeway do I have before failing?”
  - All possible using shadow registers

Background: Shadow Registers



User circuit (compiled using regular tools, and locked down)

Background: Shadow Registers

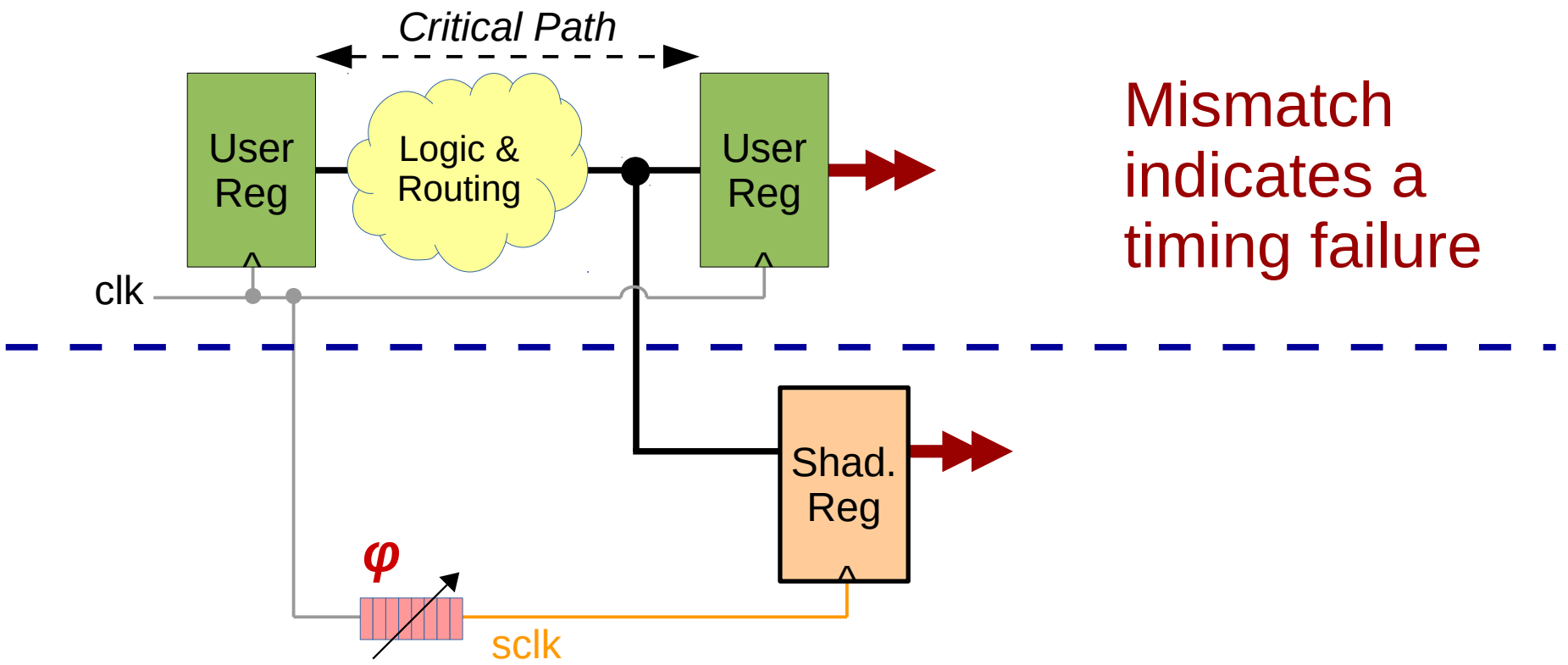


User circuit (compiled using regular tools, and locked down)

*Timing instruments (built out of **spare** resources only)*

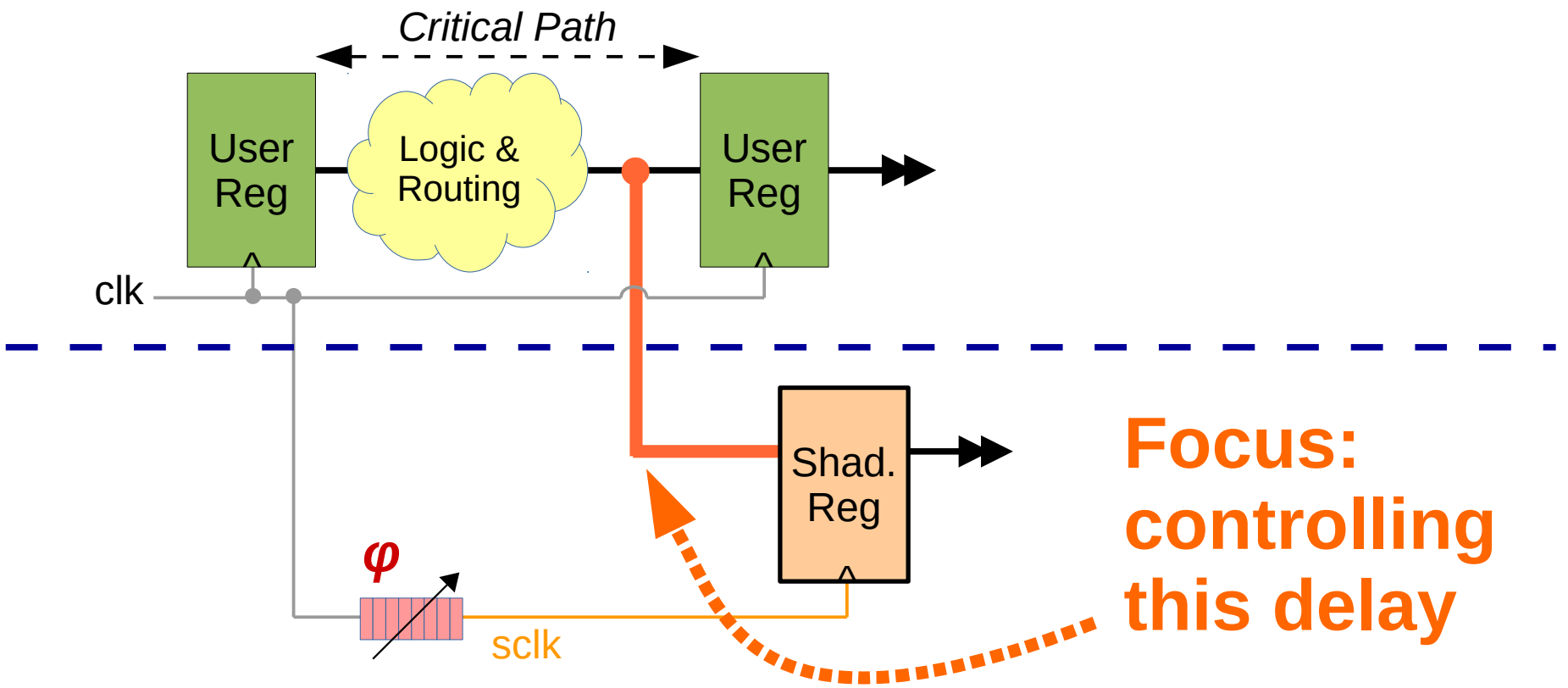
Duplicates a critical path endpoint, but with a phase-shifted clock

Background: Shadow Registers



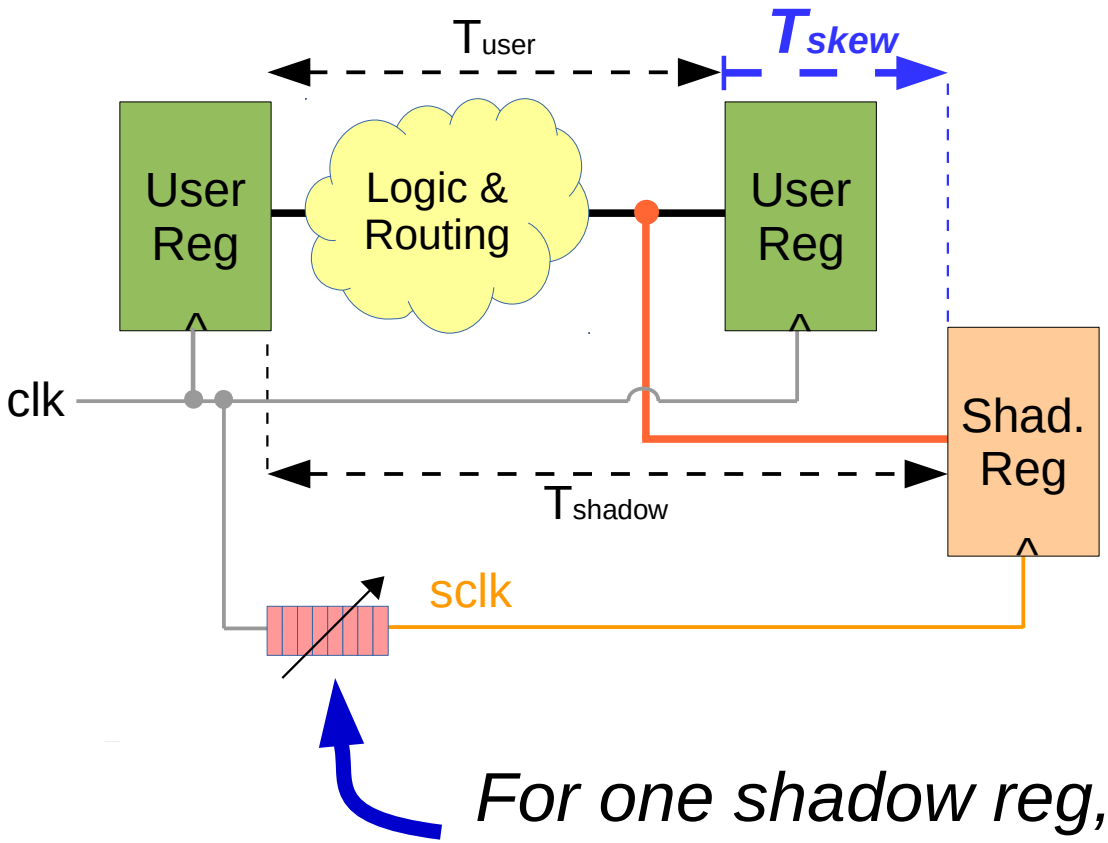
Duplicates a critical path endpoint, but with a phase-shifted clock

Background: Shadow Registers



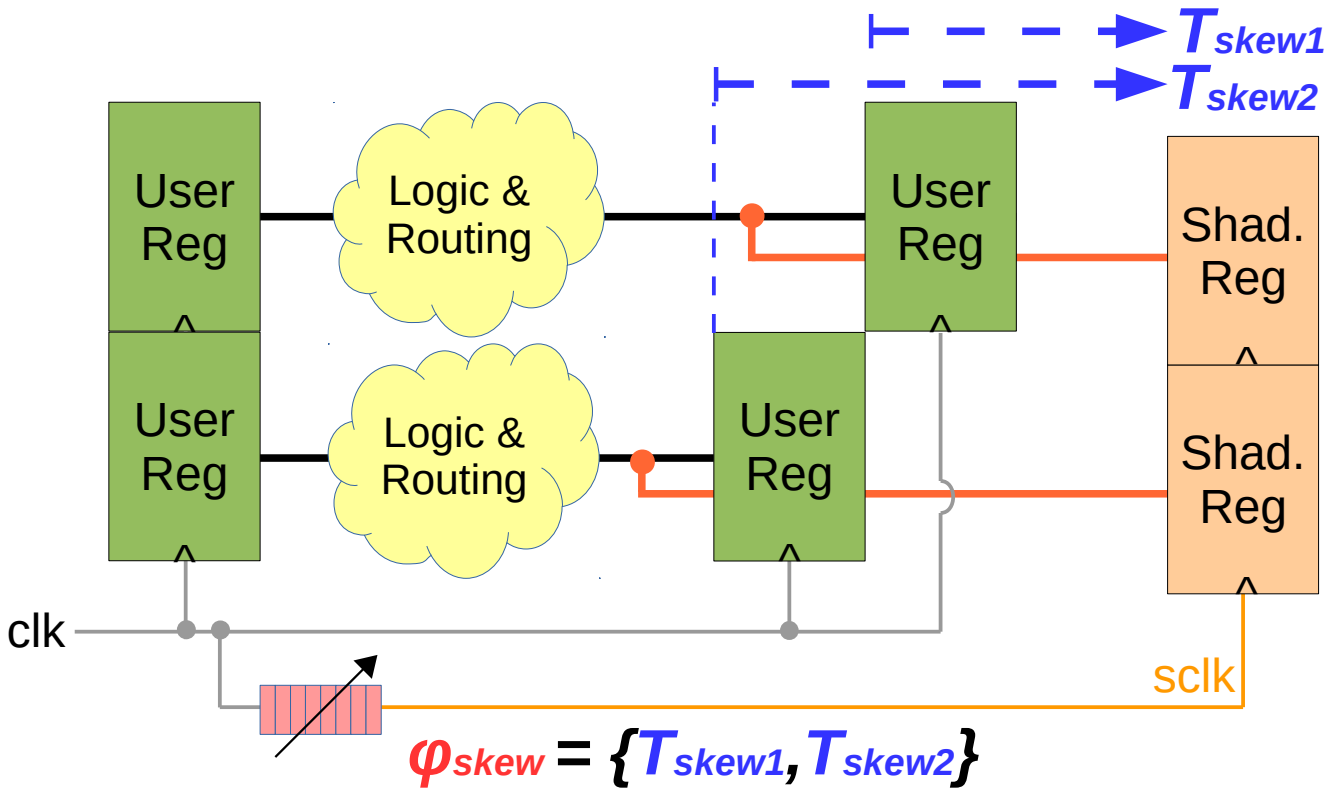
Duplicates a critical path endpoint, but with a phase-shifted clock

Challenge: Consistent Skew

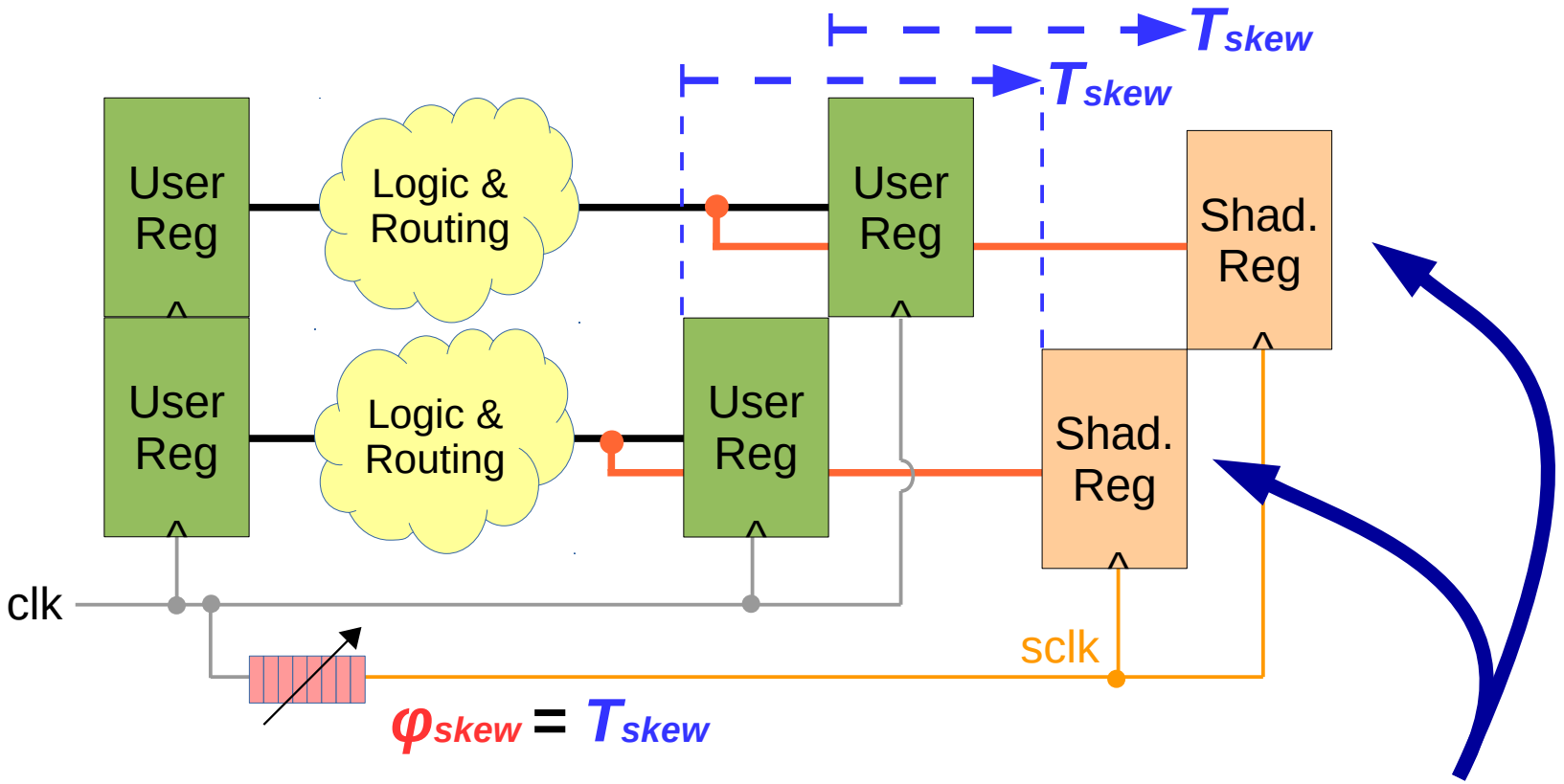


For one shadow reg, we can cancel out any path skew using a phase offset

Challenge: Consistent Skew



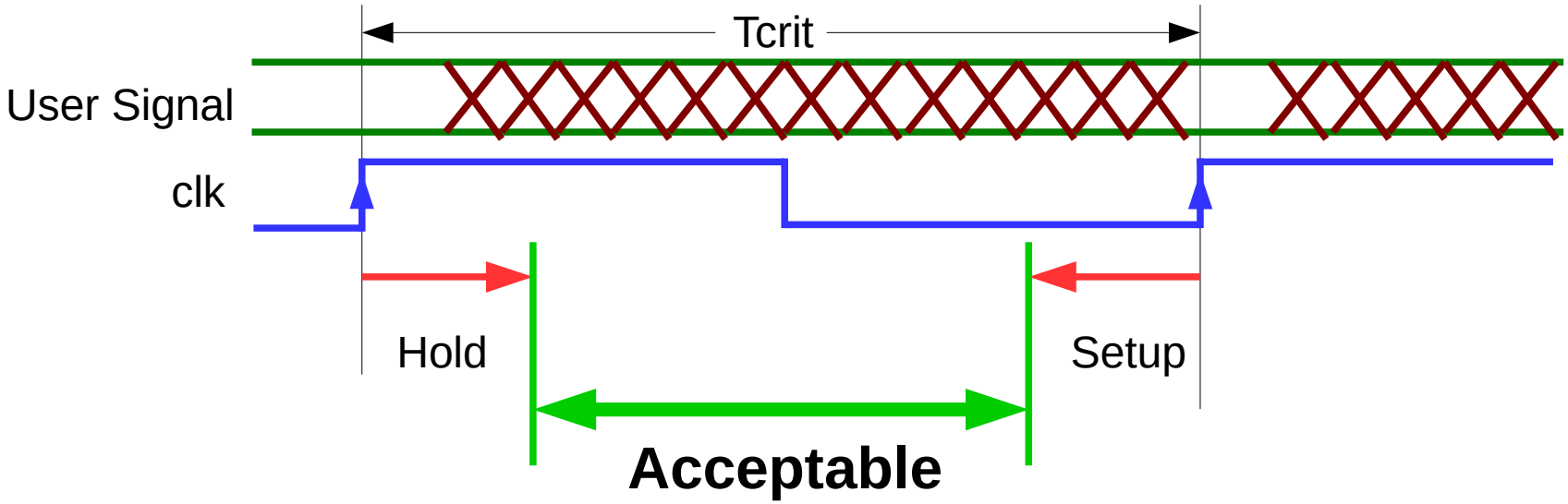
Challenge: Consistent Skew



By placing and routing these shadow registers carefully, target *consistent* skew

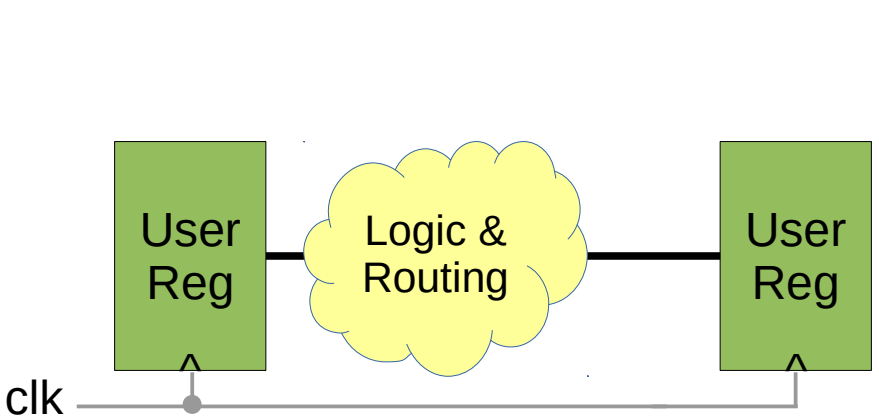
Existing Work

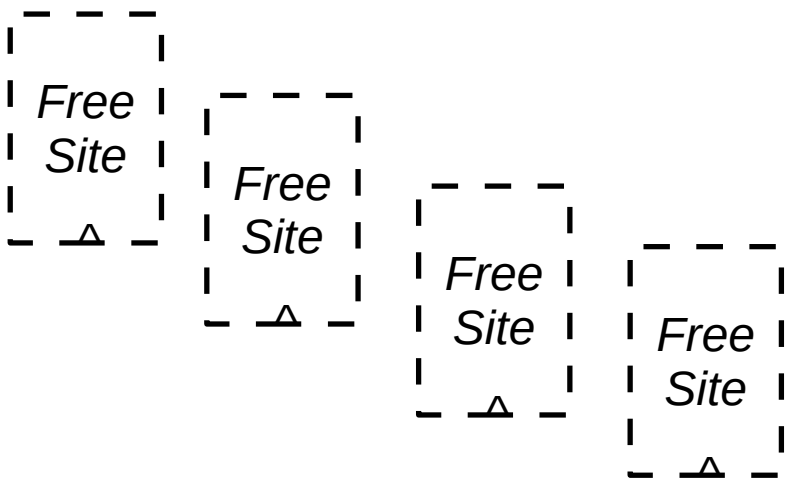
- Existing CAD tools already do bounded routing
  - Minimum delay: Hold time minus Clock-to-Q
  - Maximum delay: Clock period minus Setup time
  - Algorithms target a relatively big landing window:



Challenge: Consistent Skew

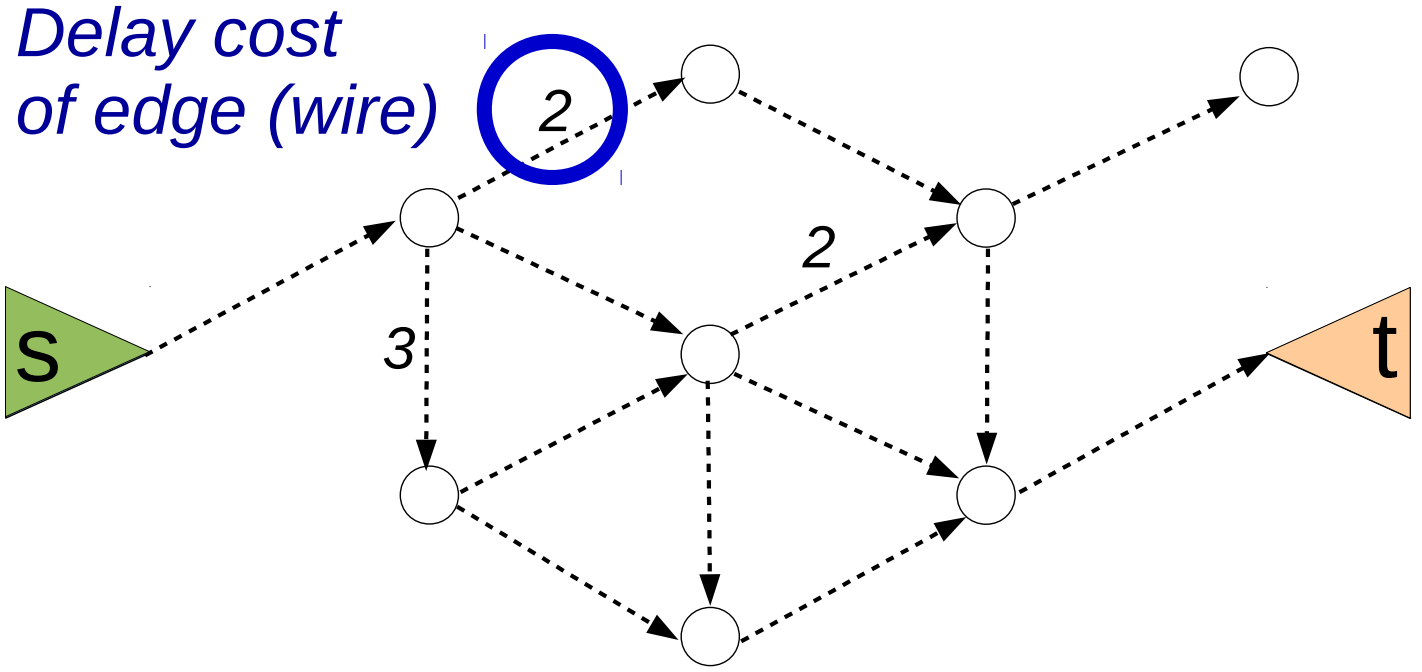
- We desire a very narrow window (ideally, 1ps)
  - Hard problem, but aided by two characteristics:
    - Freedom to place shadow register on any spare site
    - Freedom to route to any site using any spare resources
  - Solved simultaneously by routing to all sites





Problem

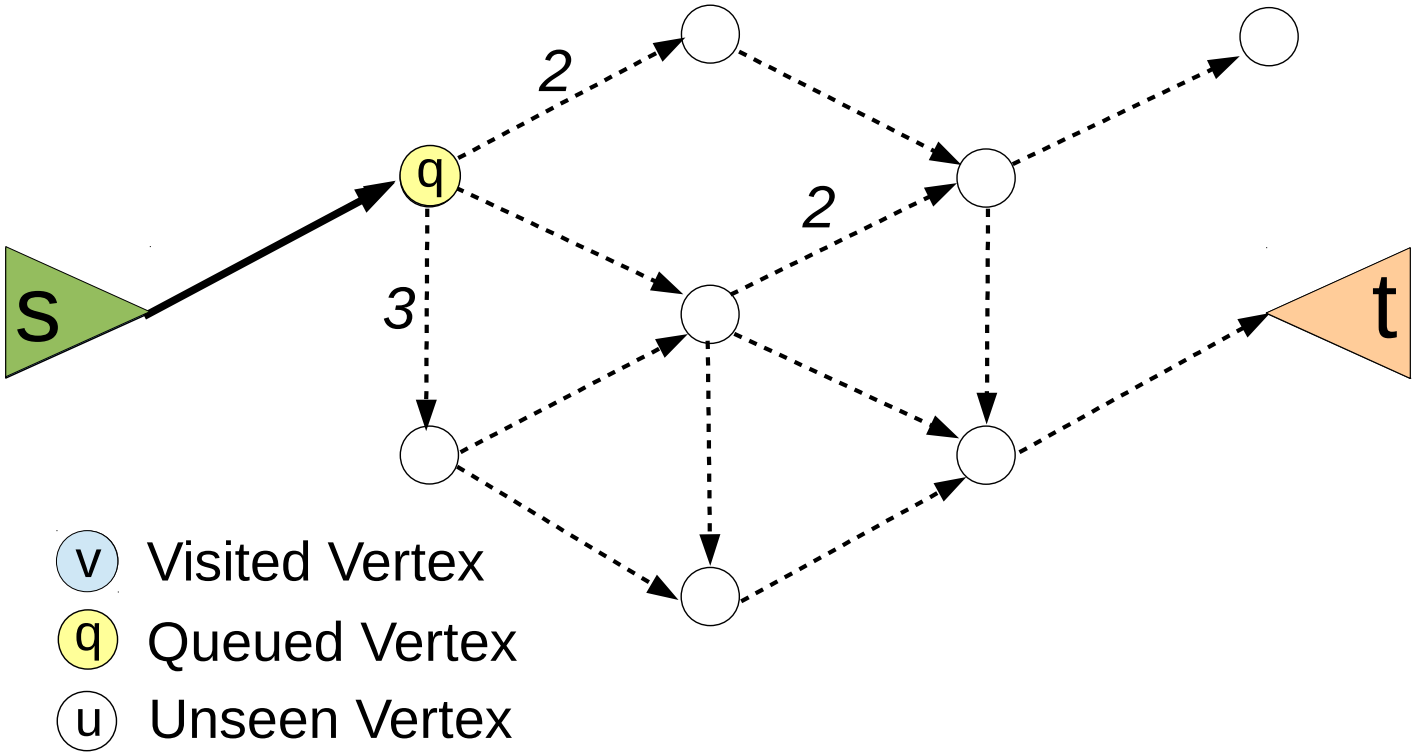
- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra



Edge cost = 1
unless labelled

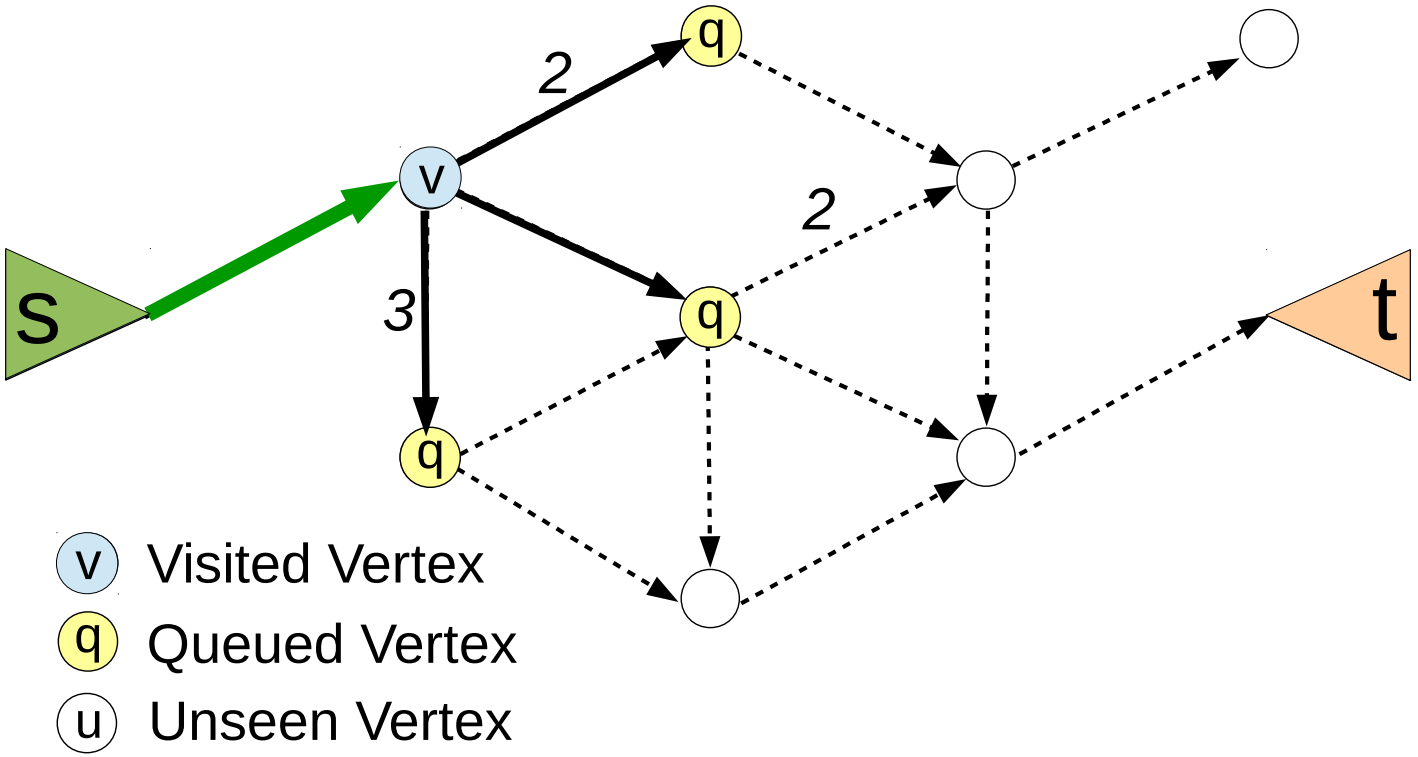
Problem

- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra



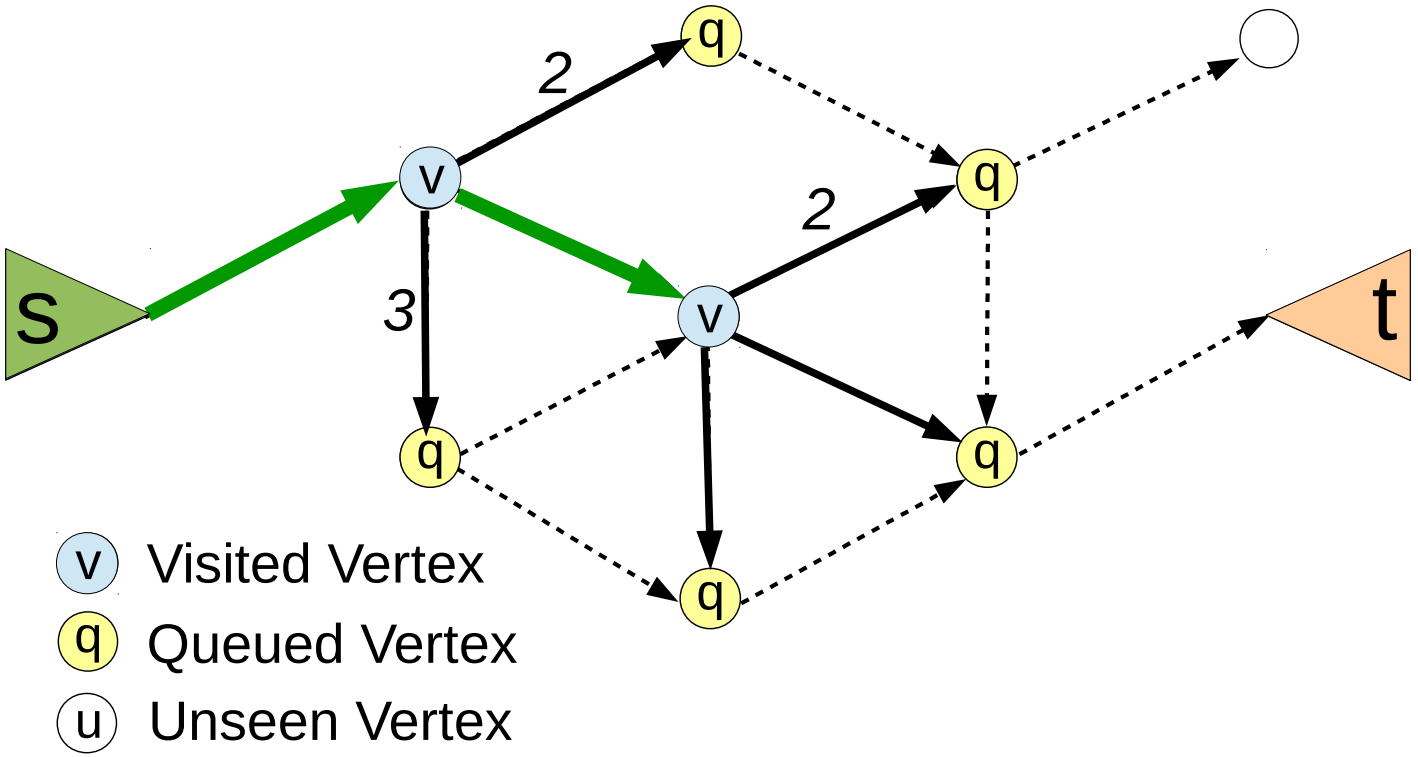
Problem

- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra



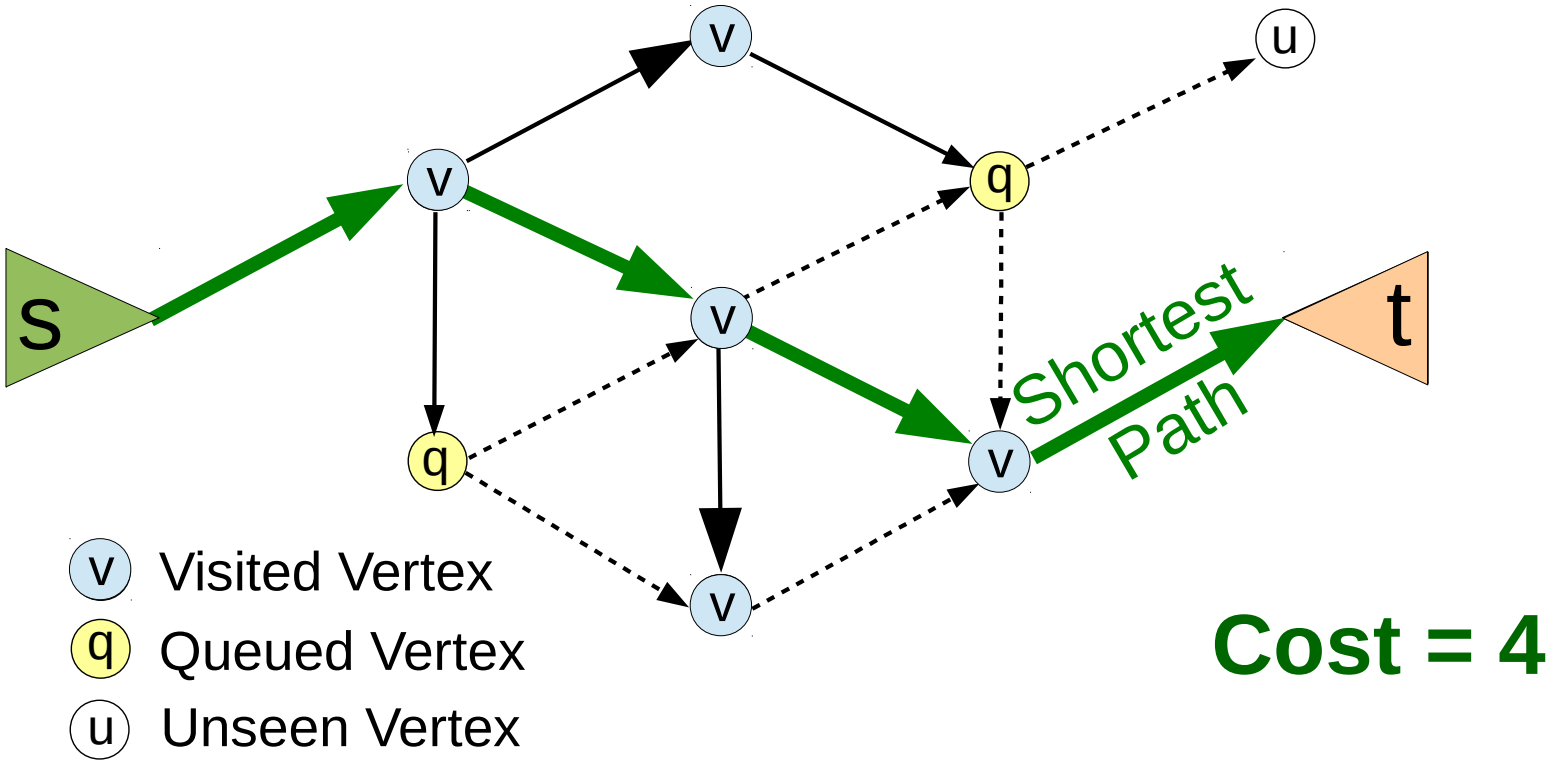
Problem

- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra



Problem

- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra

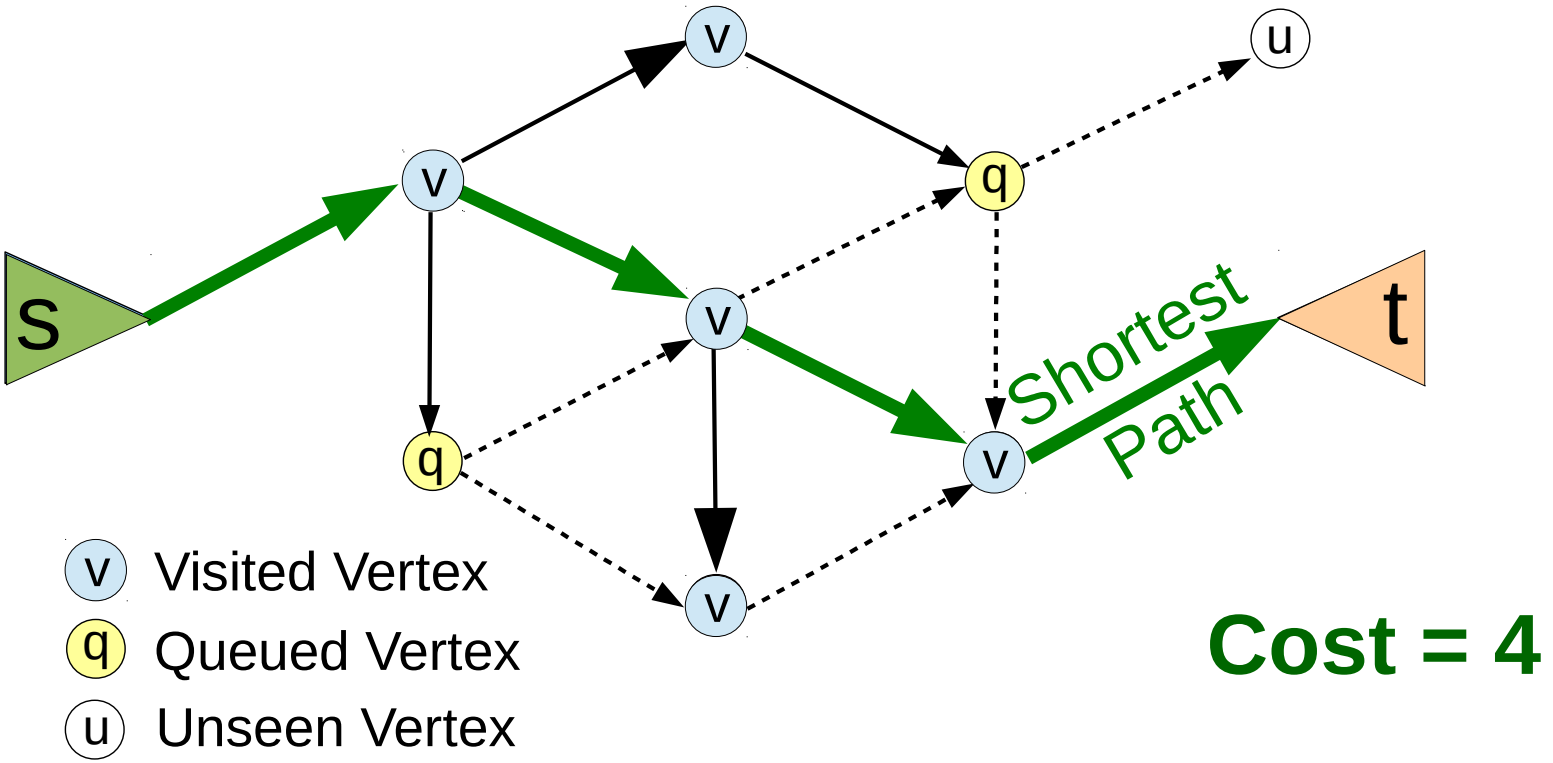


Problem

- FPGA routing is a case of graph search
  - Finding shortest path has long been solved: Dijkstra
  - But how can we find not-so-short paths?

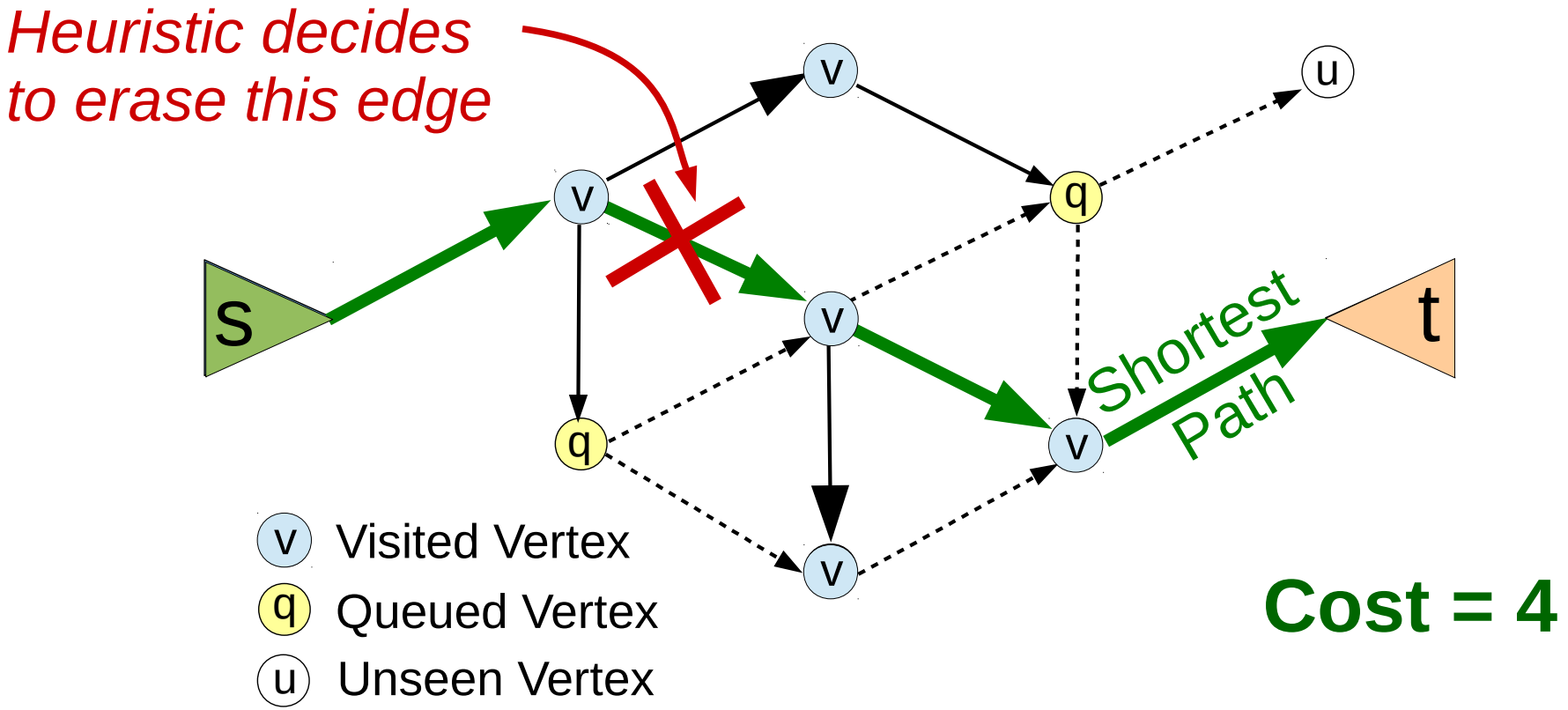
Proposal: Dijkstra with Rollback

- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist



Proposal: Dijkstra with Rollback

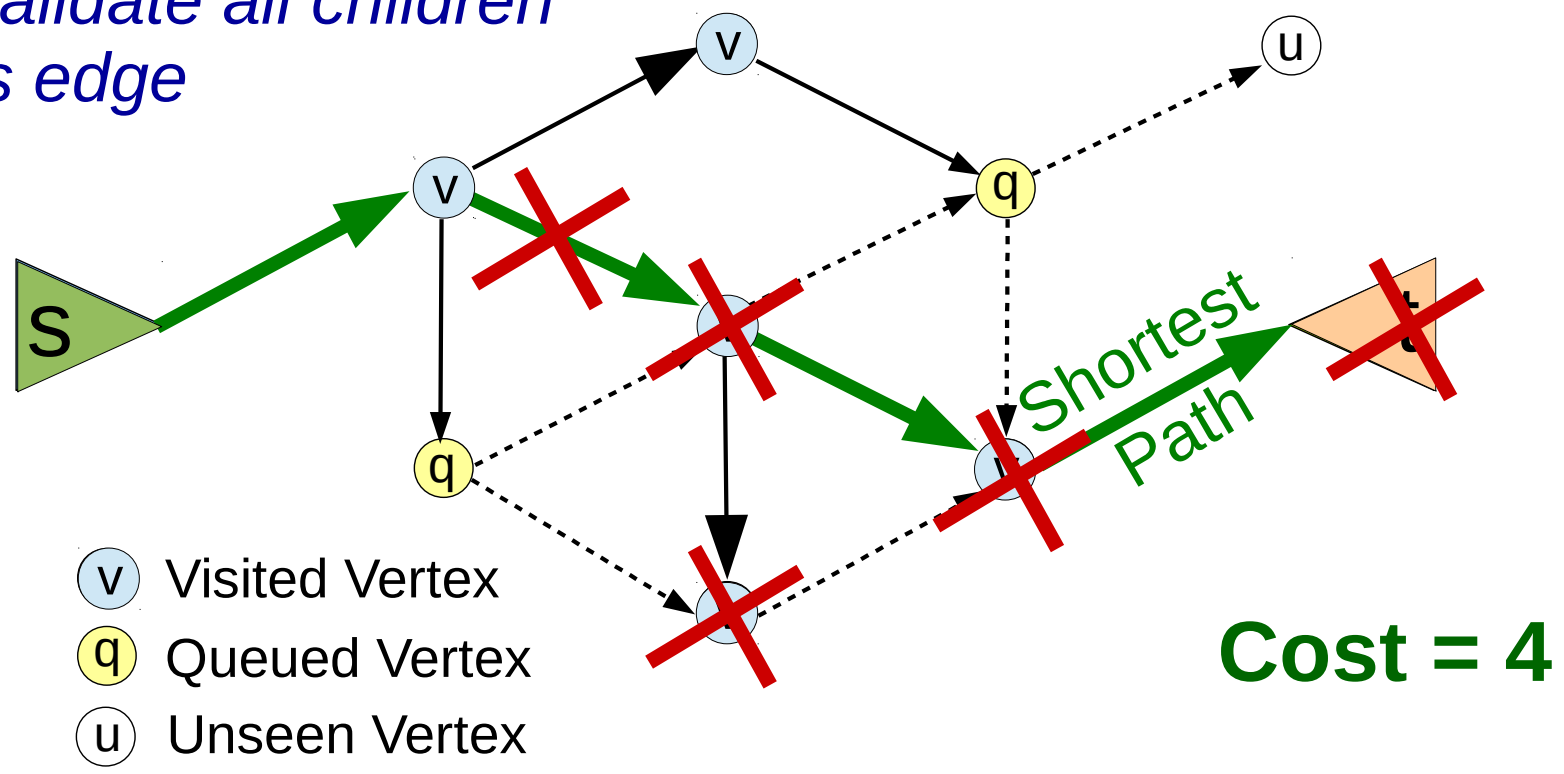
- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist



Proposal: Dijkstra with Rollback

- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist

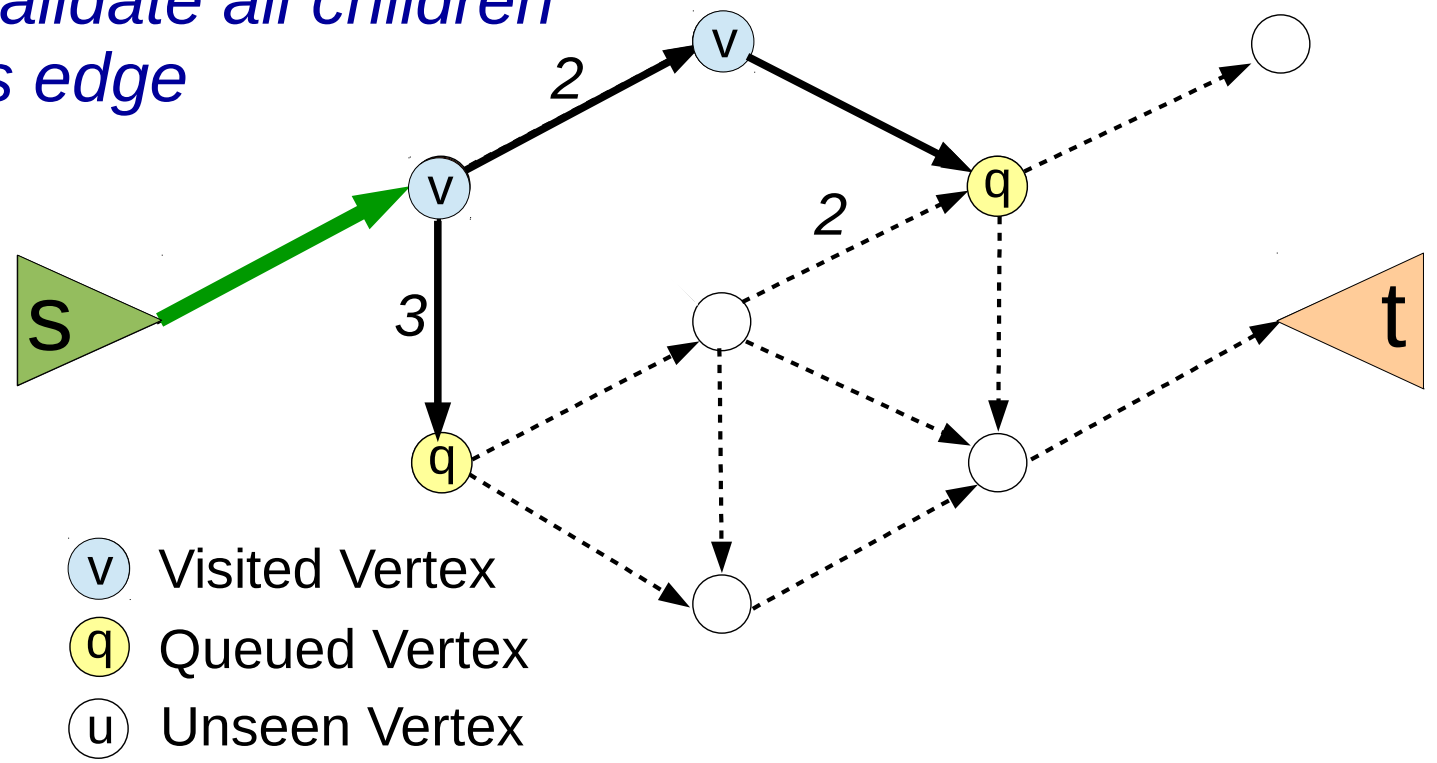
1. *Invalidate all children of this edge*



Proposal: Dijkstra with Rollback

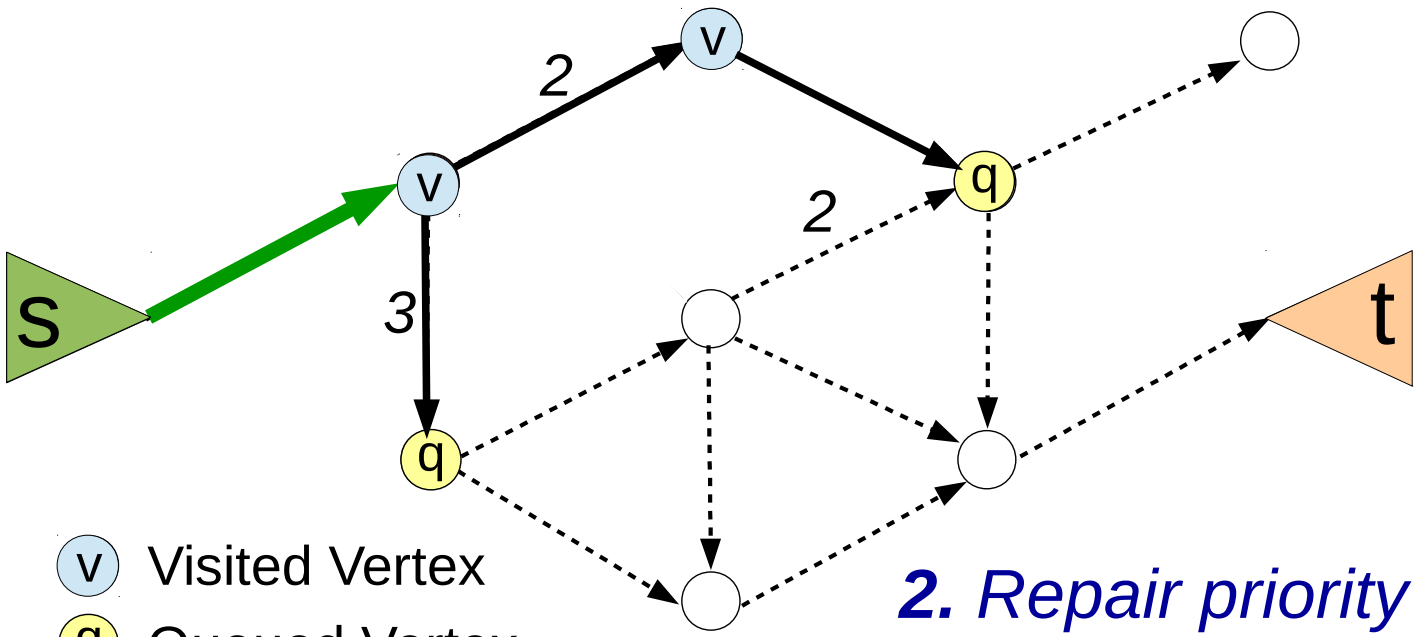
- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist

1. *Invalidate all children of this edge*



Proposal: Dijkstra with Rollback

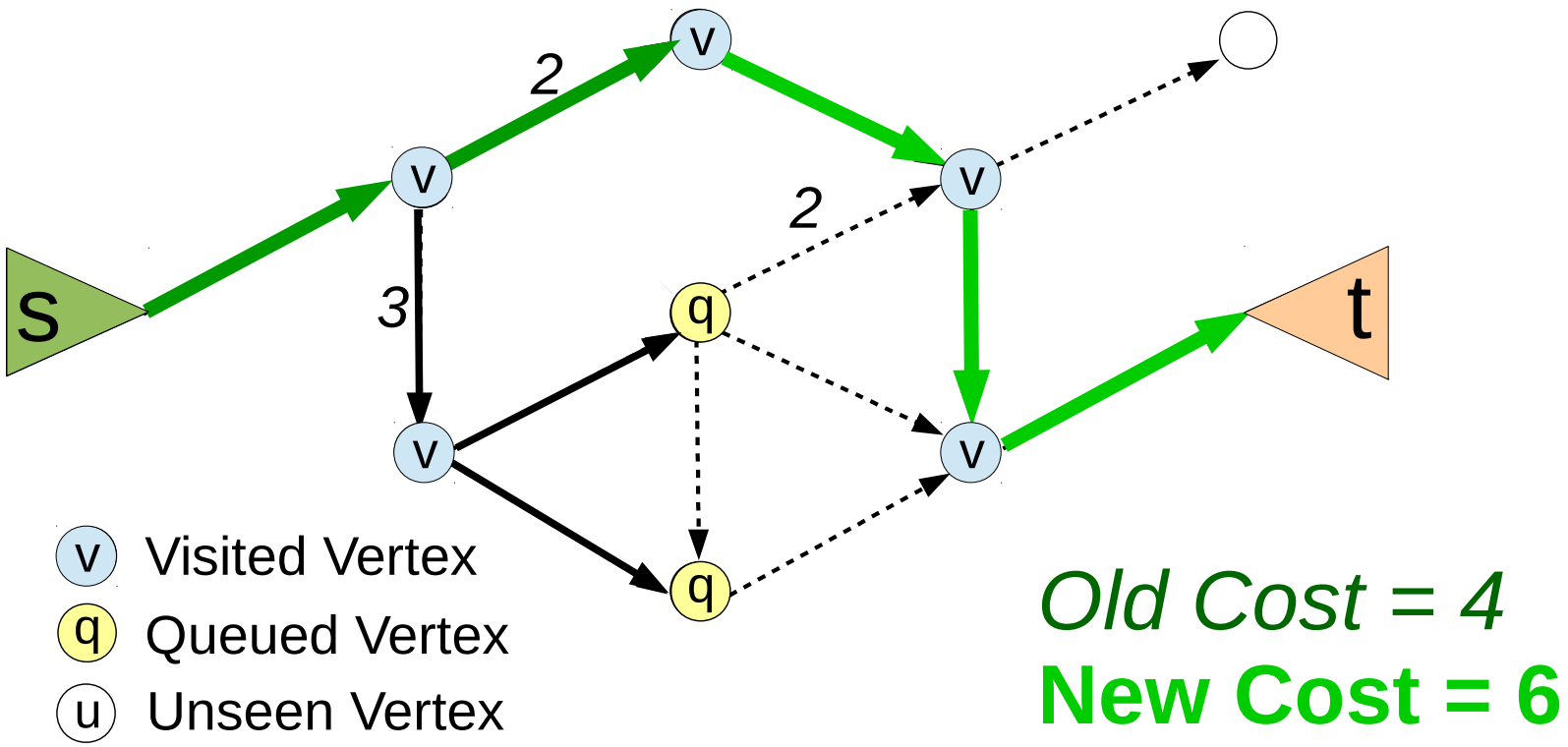
- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist



2. *Repair priority queue (fix costs of those nodes that would have been visited by another edge)*

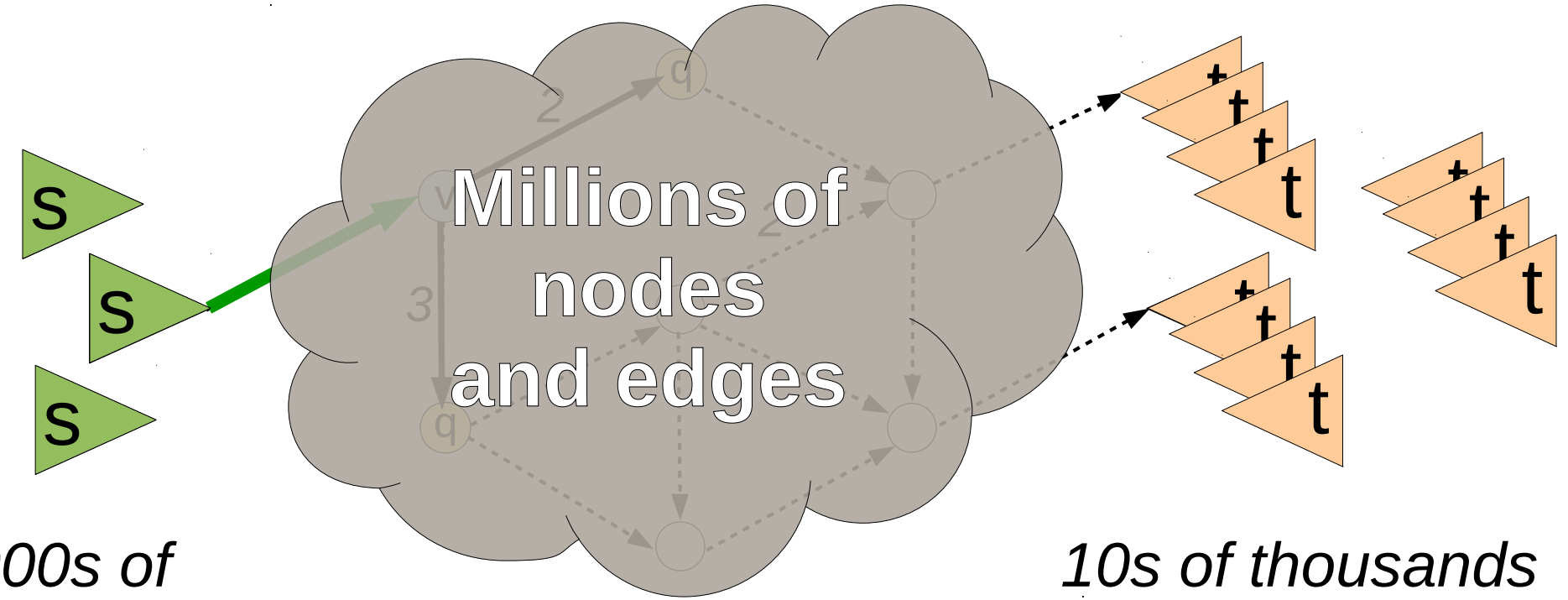
Proposal: Dijkstra with Rollback

- Summary: if path found is too short, undo search and rerun as if a prior edge did not exist



Proposal: Dijkstra with Rollback

- Big picture: disjoint paths from each **s** to any **t**

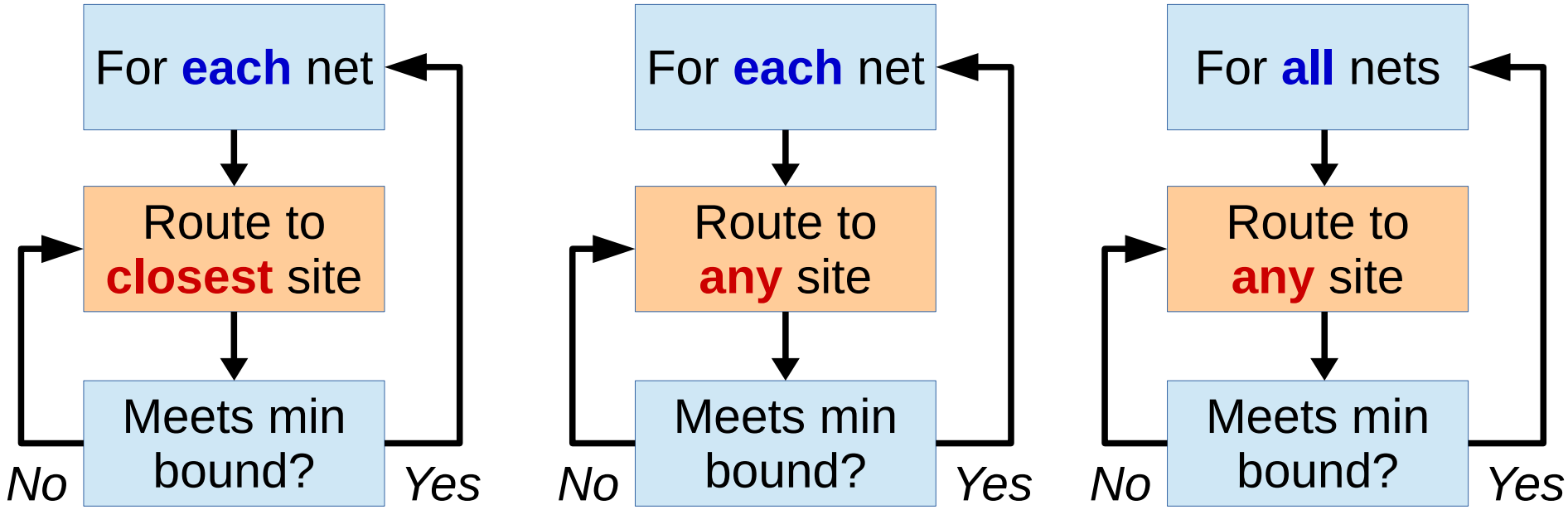


Proposal: Dijkstra with Rollback

- Not dissimilar to Yen's K-shortest path algorithm
  - Guaranteed to find the shortest, and K-1 next shortest, paths
  - Essence: remove edge(s) of prior shortest paths, restart Dijkstra, then replace edge(s) and retry
  - Our heuristic:
    - Removes each edge permanently
    - Repair and continue, instead of restarting Dijkstra
    - No guarantee of finding next shortest, but much faster

Proposal: Dijkstra with Rollback

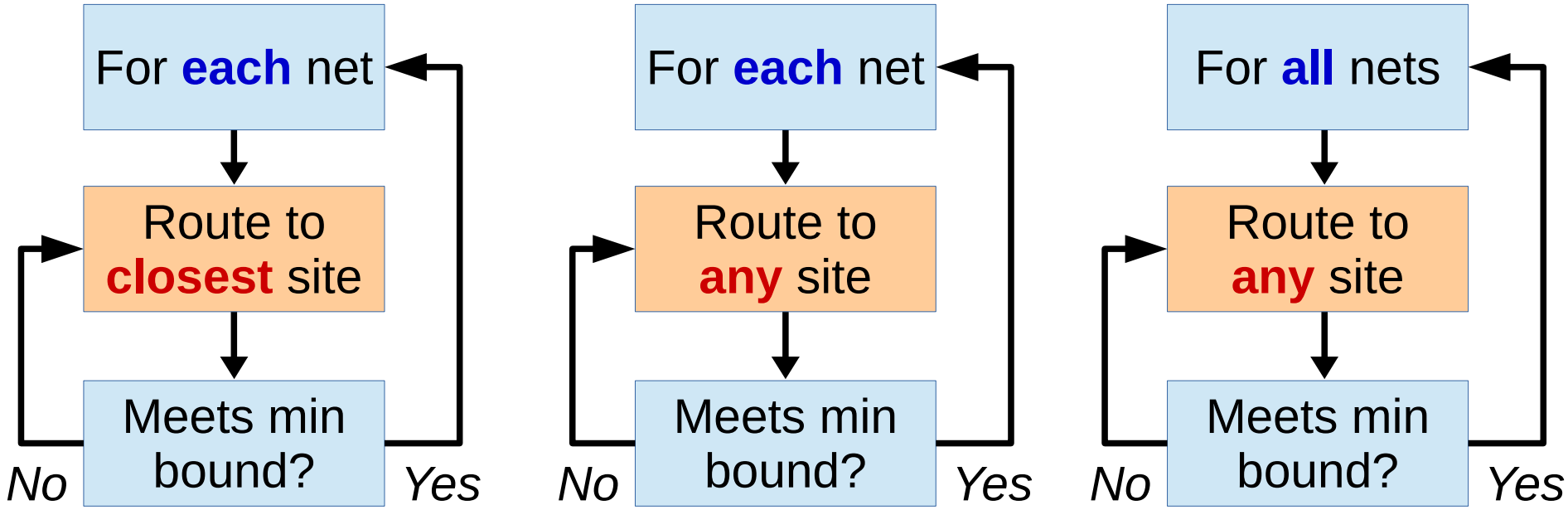
- We propose three different algorithms:



One-Closest **One-All** **All-All**

Proposal: Dijkstra with Rollback

- We propose three different algorithms:



One-Closest **One-All** **All-All**

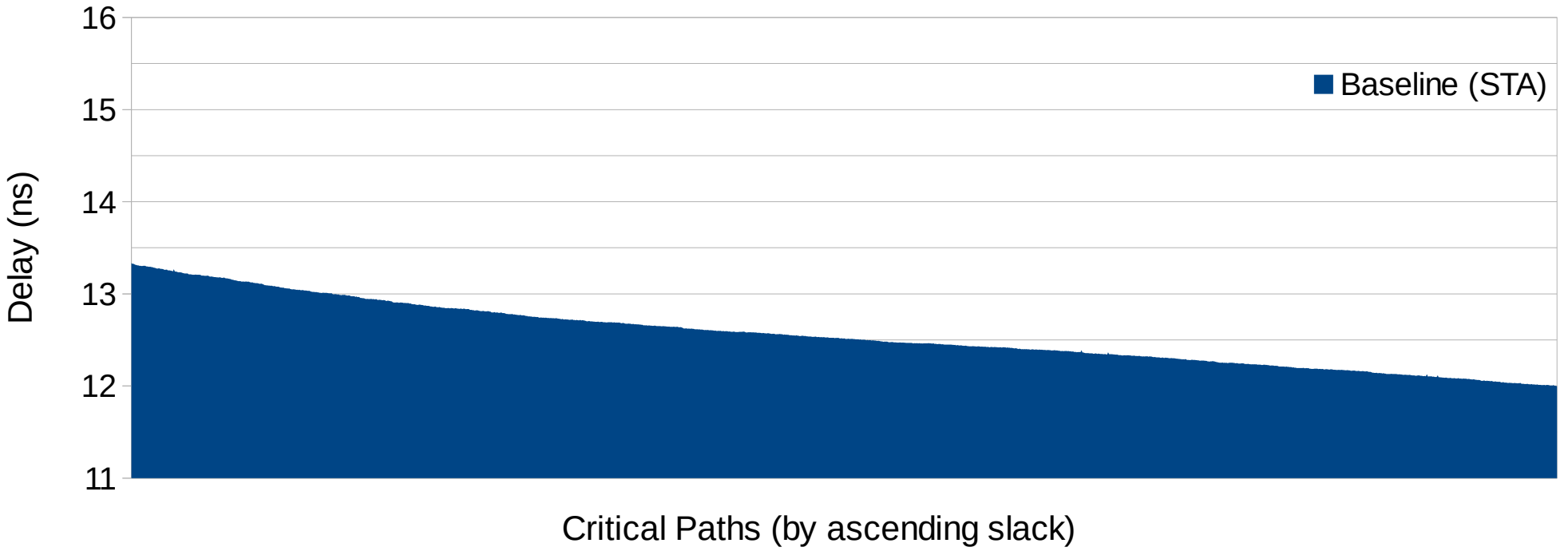
Full details are in our paper!

Experimental Flow

- Evaluated on Xilinx, but applicable to others
  - Place and route using ISE, apply our shadow register tool, then re-analyse using vendor STA
- Experimented on three large benchmarks:
  - LEON3: an 8-core system-on-chip
  - AES-x3: a chain of 3 encoders then 3 decoders
  - JPEG-x2: two parallel instances of CHStone JPEG synthesised using Vivado HLS
  - All occupying ~90% of mid-range Virtex6

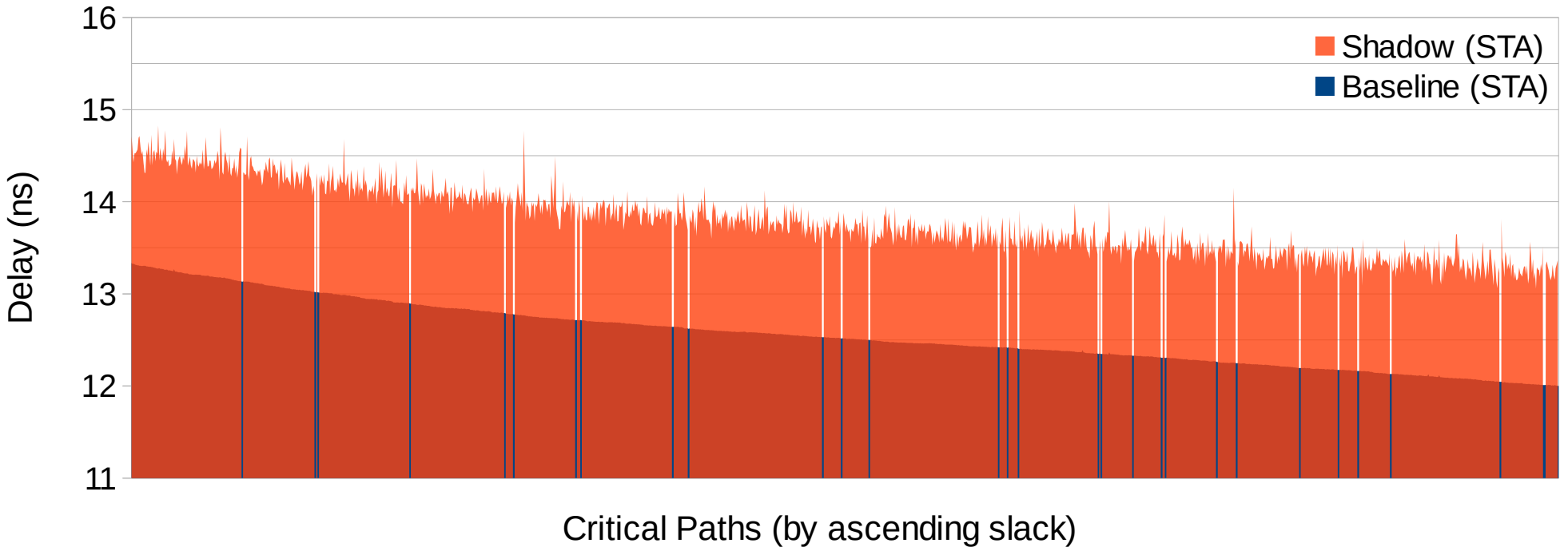
Results – LEON3

- Critical endpoints with <10% slack (1424)



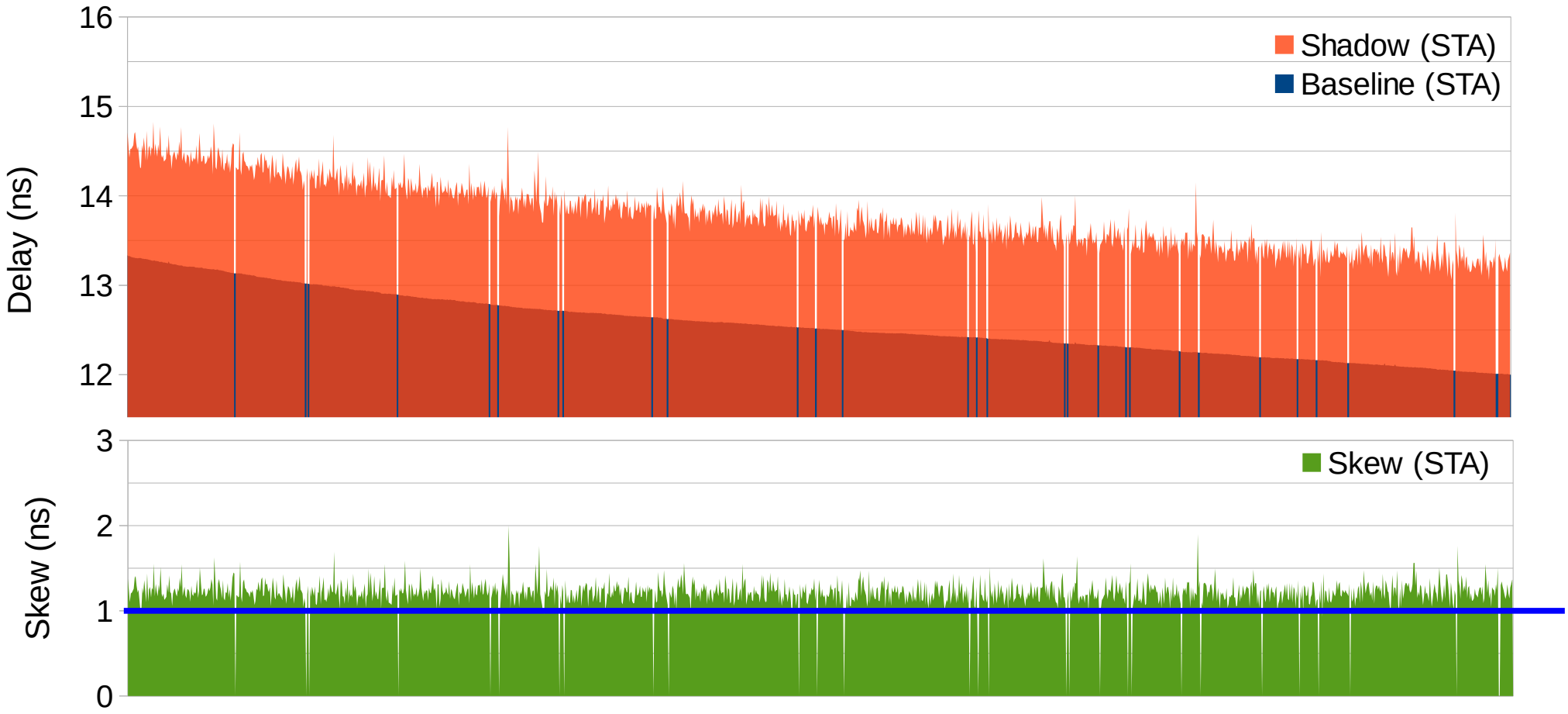
Results – LEON3

- Critical endpoints with <10% slack (1424)
  - With 1417 shadow registers at >1ns skew target



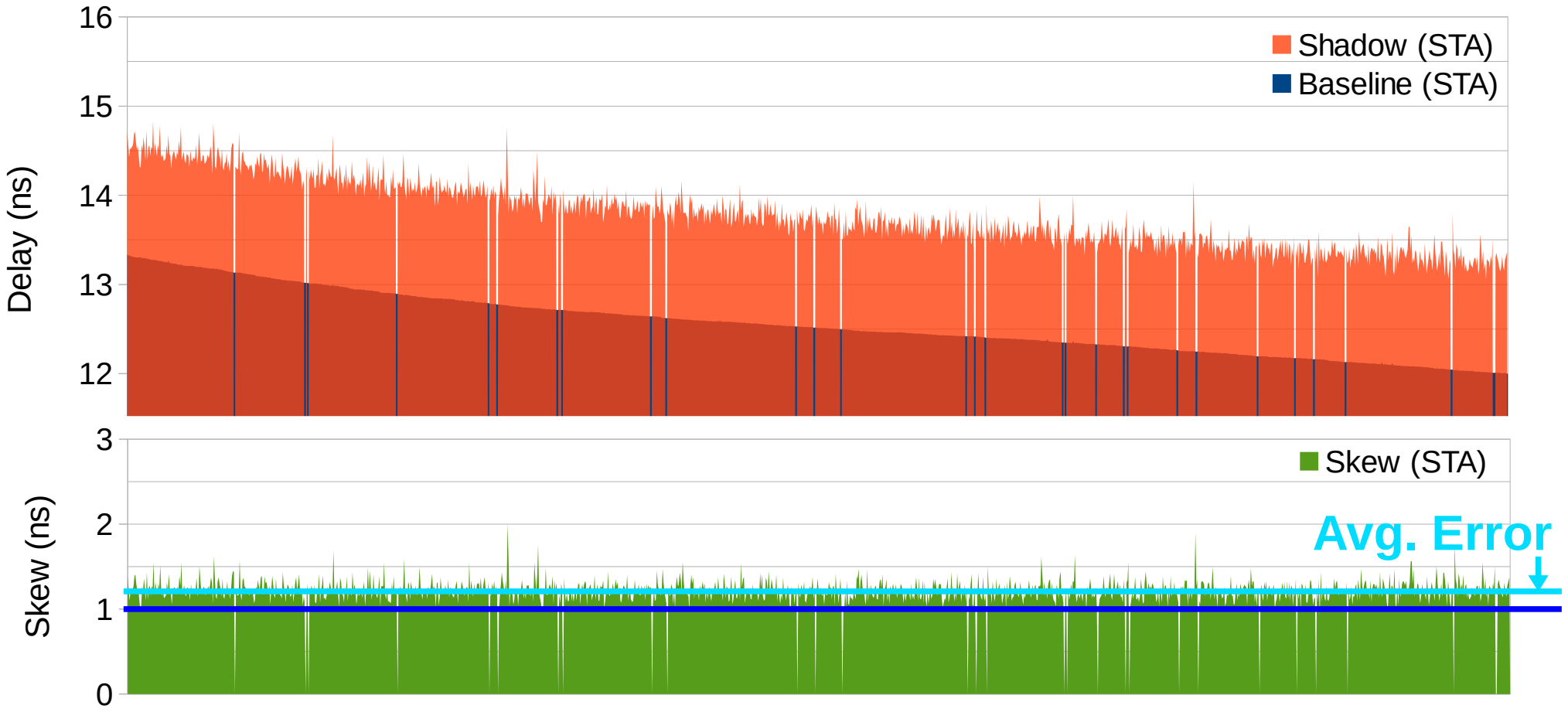
Results – LEON3

- Critical endpoints with <10% slack (1424)
  - With 1417 shadow registers at **>1ns skew target**



Results – LEON3

- Critical endpoints with <10% slack (1424)
  - With 1417 shadow registers at >1ns skew target

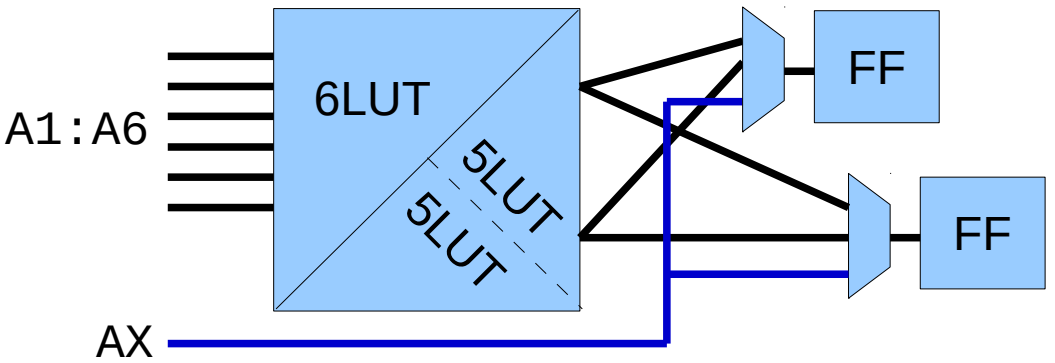


Results – LEON3

- Average skew error of 200ps (from Xilinx STA)
  - Due to incomplete timing information
  - Based on our delay model, we achieve <1ps error!

Results – LEON3

- Average skew error of 200ps (from Xilinx STA)
  - Due to incomplete timing information
  - Based on our delay model, we achieve <1ps error!
  - Why so good?
    - Many spare registers (total: 300K, used: 62K, spare: 78K)
    - Millions of wires (nodes), 10s of millions of switches (edges)
    - … compounding with up to 7 ways (with differing delays) to get to each:



More in the paper...

- Experimental results for:
  - Three different algorithms on three benchmarks
  - Different number of steps to roll back
  - Different values for skew target
- Analysis on why not all nets could be shadowed

More in the paper...

- Experimental results for:
  - Three different algorithms on three benchmarks
  - Different number of steps to roll back
  - Different values for skew target
- Analysis on why not all nets could be shadowed

- Future work: insert downstream infrastructure
  - Comparators, counters, recovery logic, etc.
  - Opportunities: latency-insensitive, reduction operation

Conclusion

- Shadow registers can help detect, measure and react to physical imperfections
  - But need to be inserted precisely: consistent skew

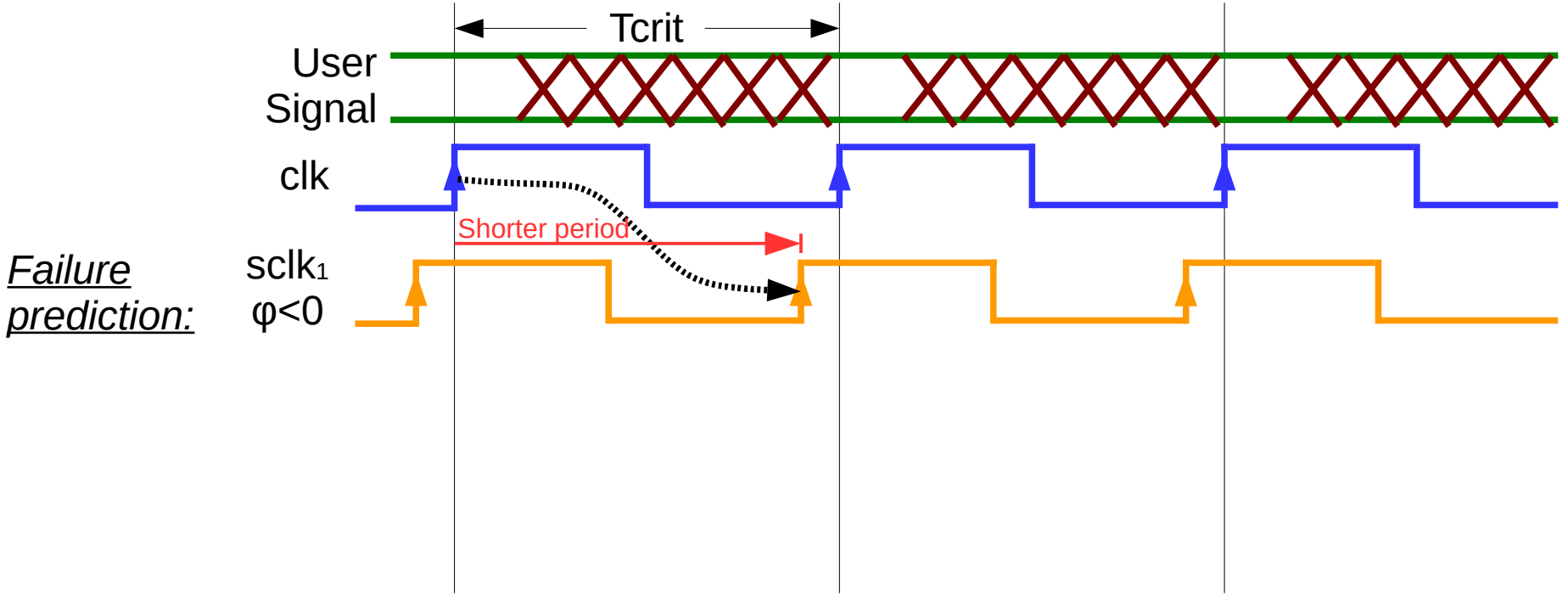
Conclusion

- Shadow registers can help detect, measure and react to physical imperfections
  - But need to be inserted precisely: consistent skew
- Our proposal: insert post place and route, with a minimum delay bound
  - Using just leftover resources ⇒ no timing impact
  - Bounding achieved by Dijkstra with Rollback
- Key result: achieve 200ps average skew error
  - With more timing information, may do even better!

Backup

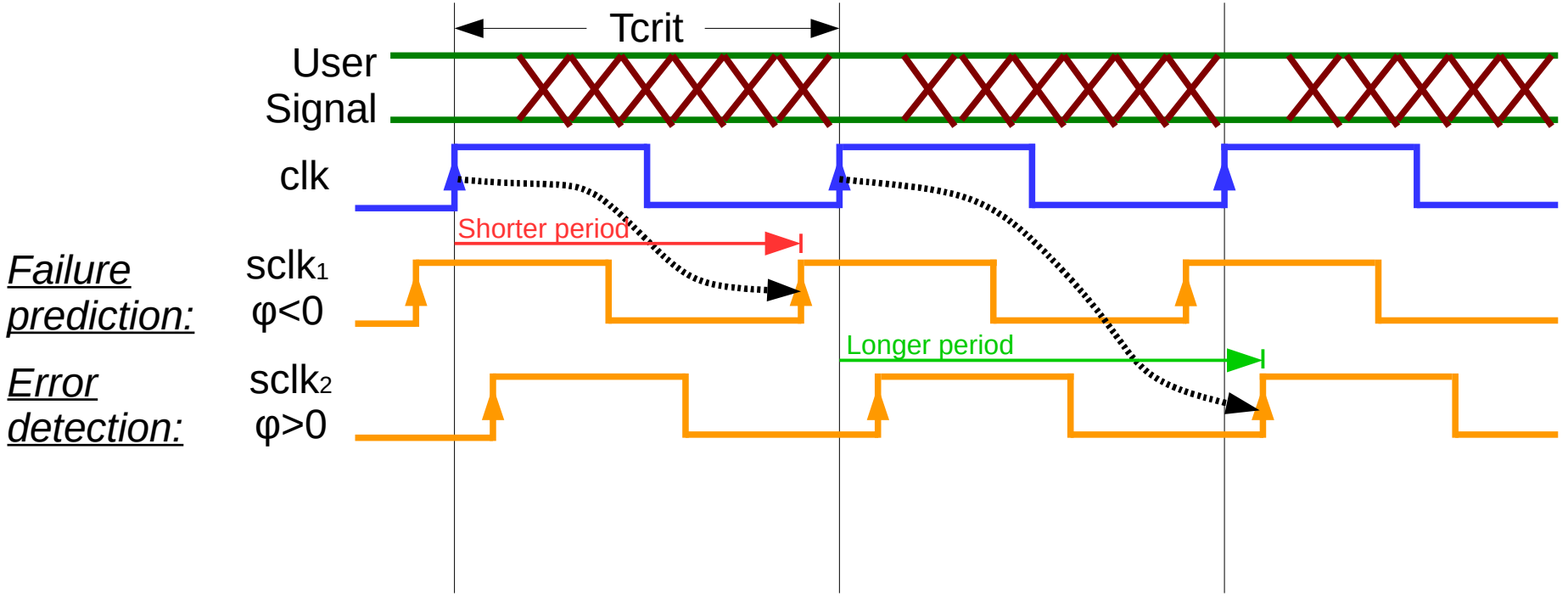
Introduction

- Shadow register clock phase φ



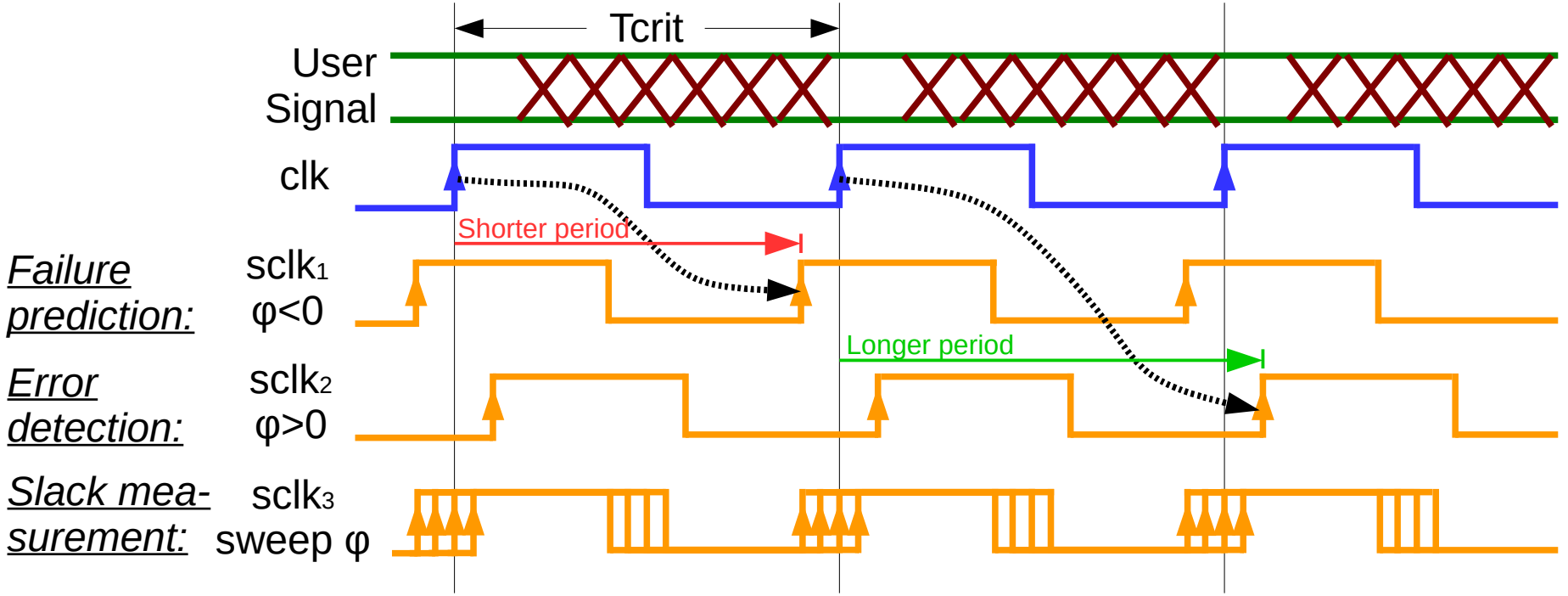
Introduction

- Shadow register clock phase φ



Introduction

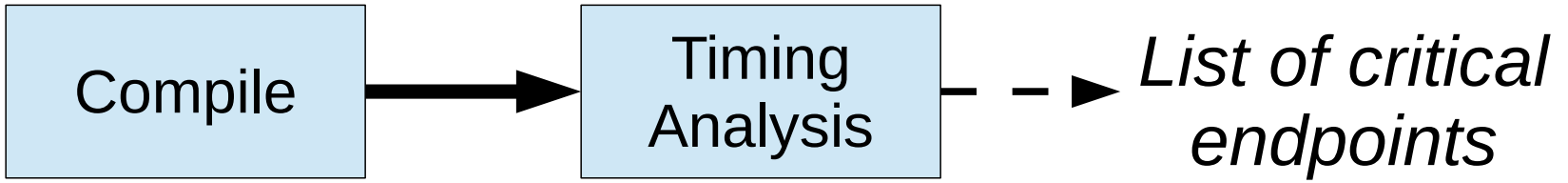
- Shadow register clock phase φ



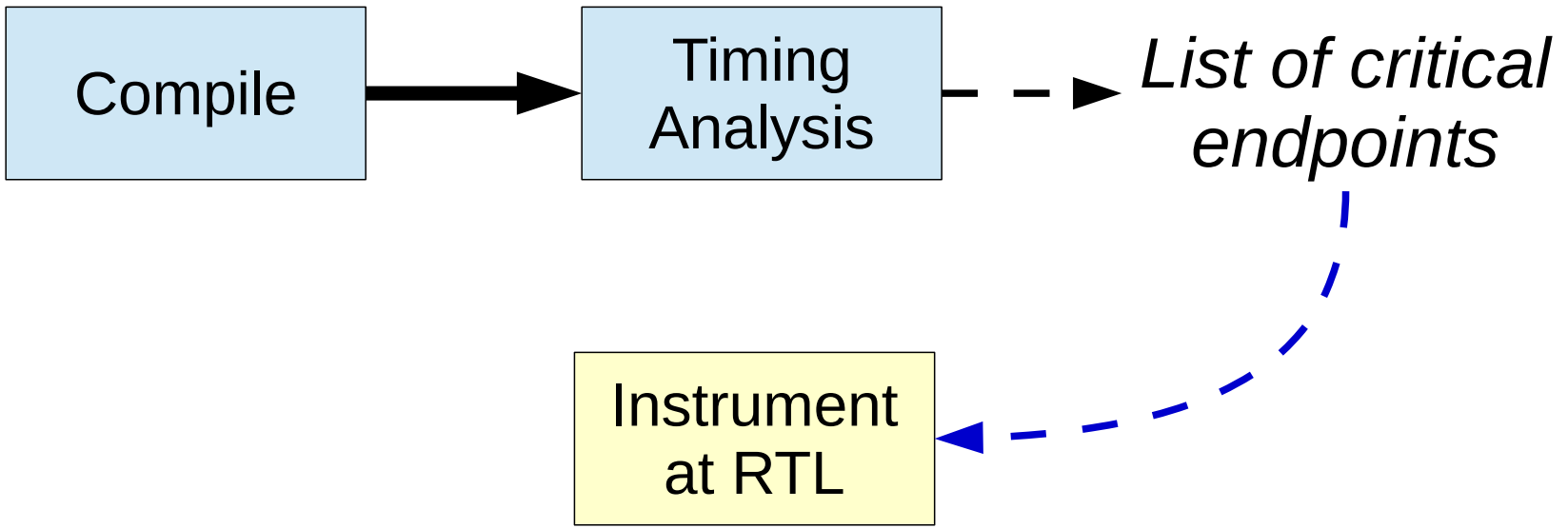
Case for Post Place and Route

- We insert shadow instruments entirely post P&R
  - By locking down all parts of existing user circuit ...
  - … and using spare, leftover, resources only
  - This preserves timing of original user circuit
    - Barring one extra switch load: ~3ps

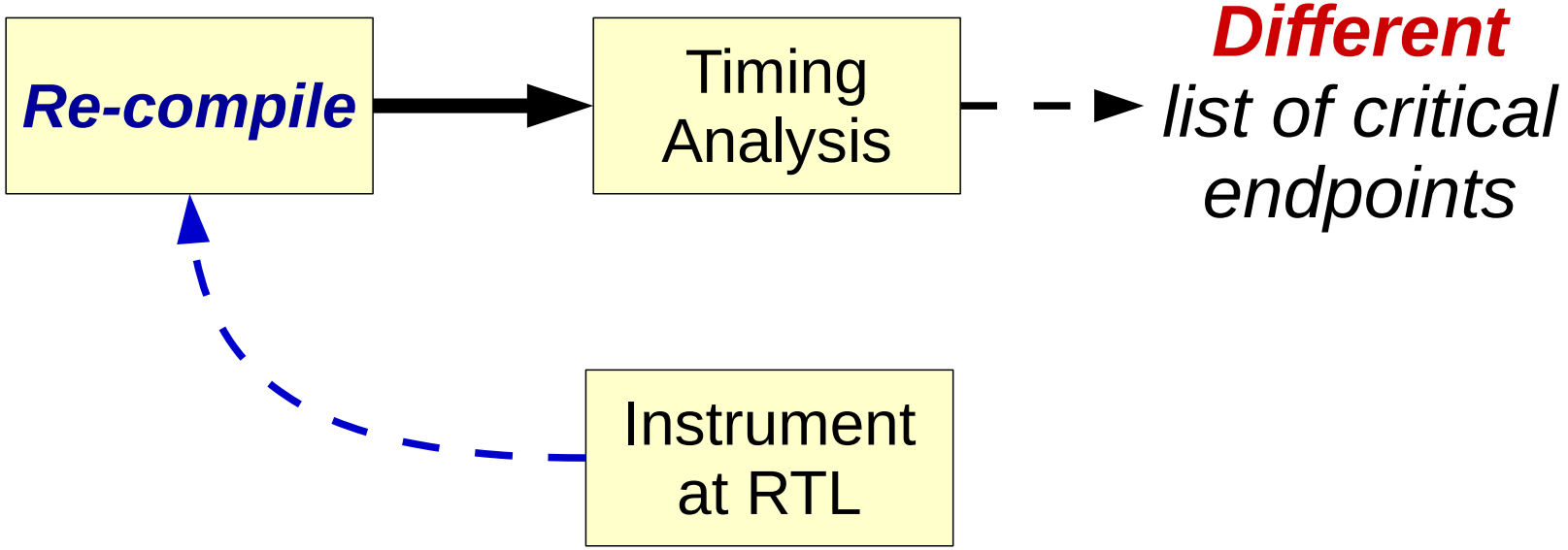
Case for Post Place and Route



Case for Post Place and Route

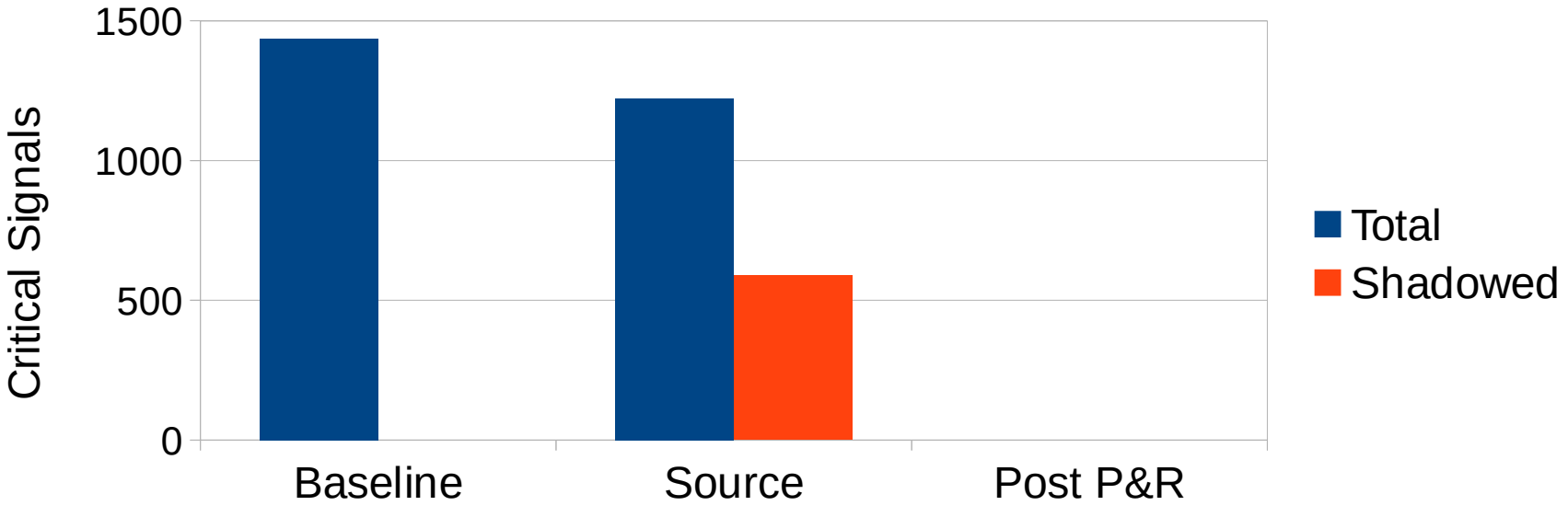


Case for Post Place and Route



Case for Post Place and Route

- For LEON3, shadowing top 10% critical nets:

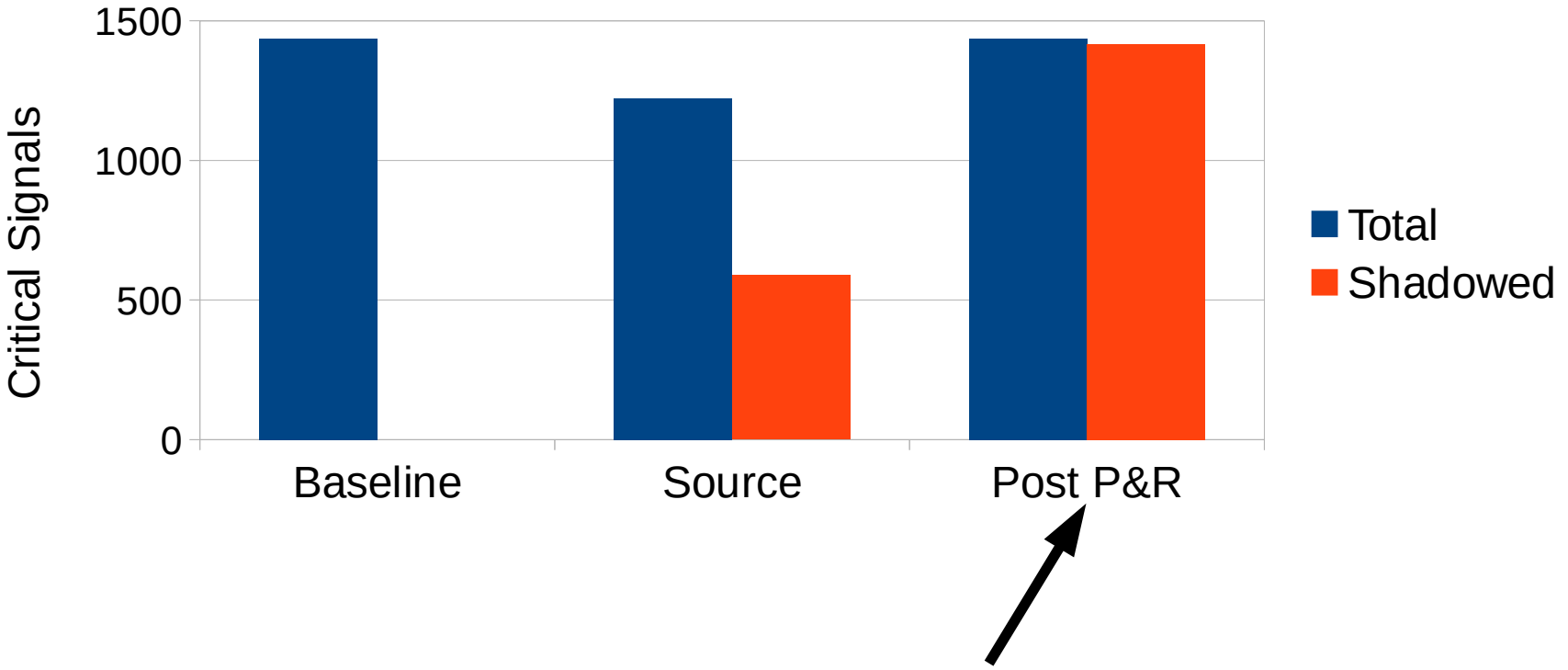


Instrument critical nets at source-level, then recompile

⇒ **Of what was critical initially, only ~50% remains critical after recompiling!**

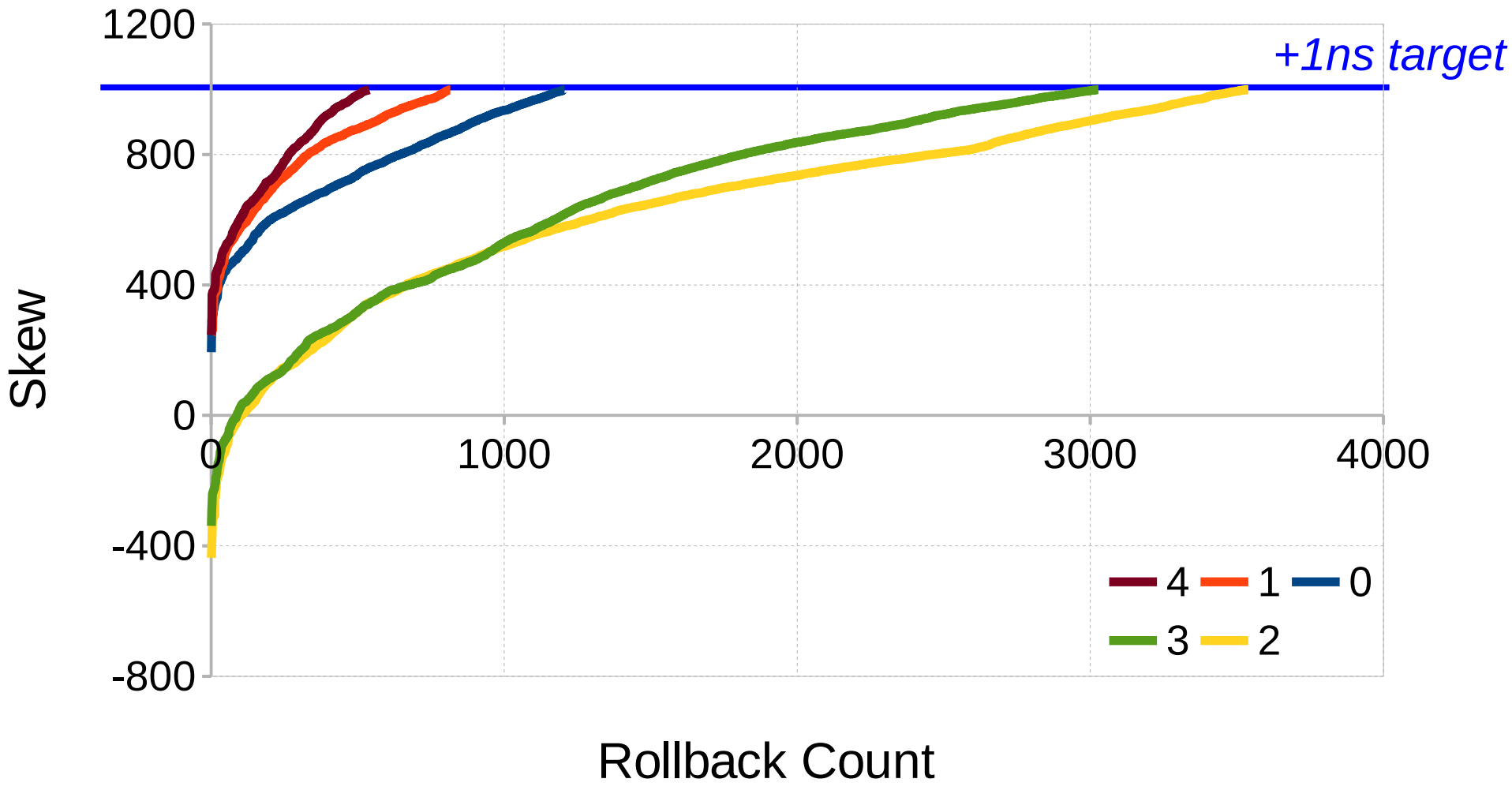
Case for Post Place and Route

- For LEON3, shadowing top 10% critical nets:

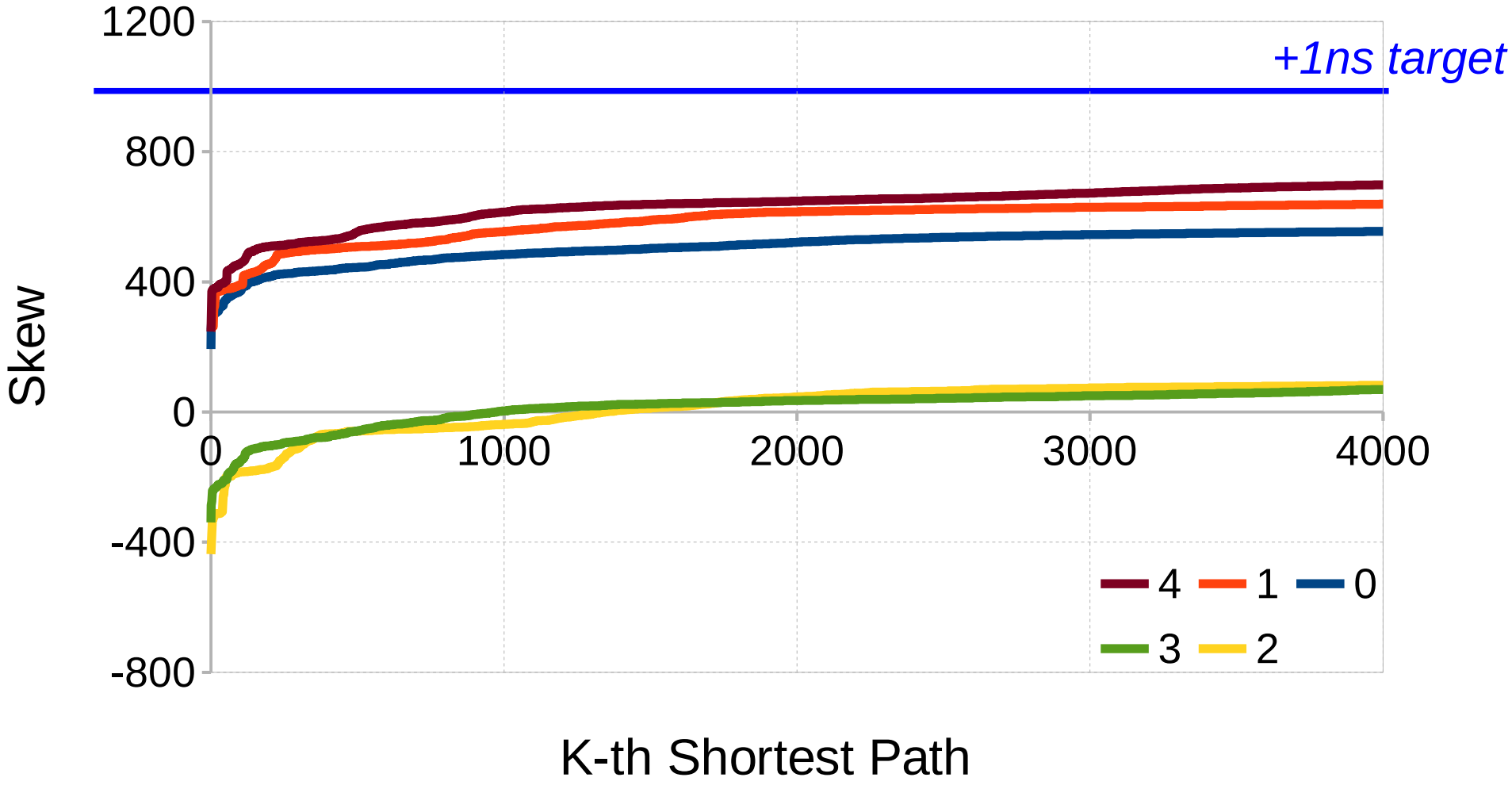


Instrument critical nets
post place and route,
using spare resources only

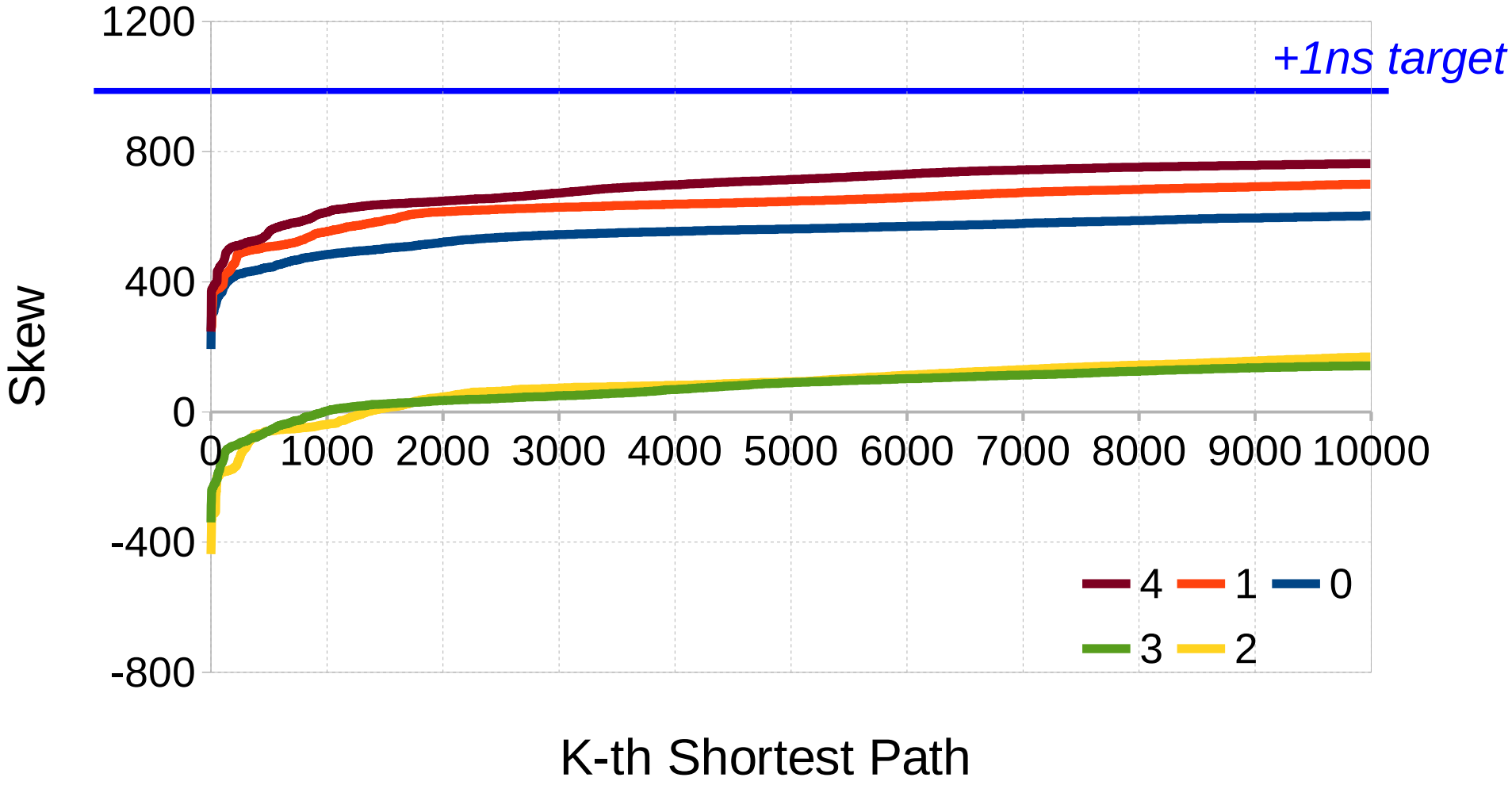
Dijkstra with Rollback Profile



Yen's K-shortest Profile



Yen's K-shortest Profile

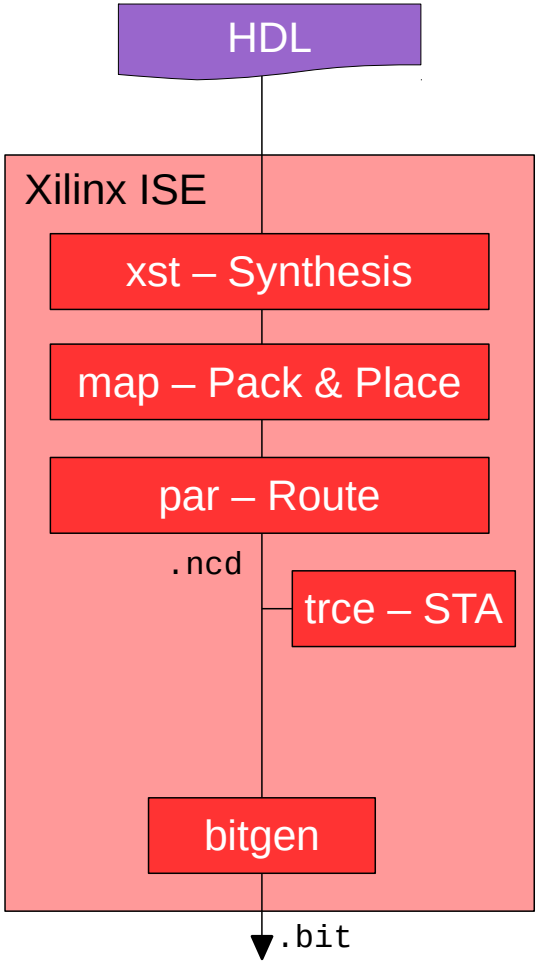


Benchmark Summary

	LEON3	AES-x3	JPEG-x2	xc6vlx240t capacity
Slices	35217	32809	32733	37680
LUTs	95766	107685	91741	150720
Registers	60562	32025	125642	301440
RAMs	688	-	124	832
DSPs	32	-	172	768
Tcrit (ns)	13.329	6.055	13.936	-
Wirelength	2.1M	1.1M	1.6M	6.3M

Experimental Flow

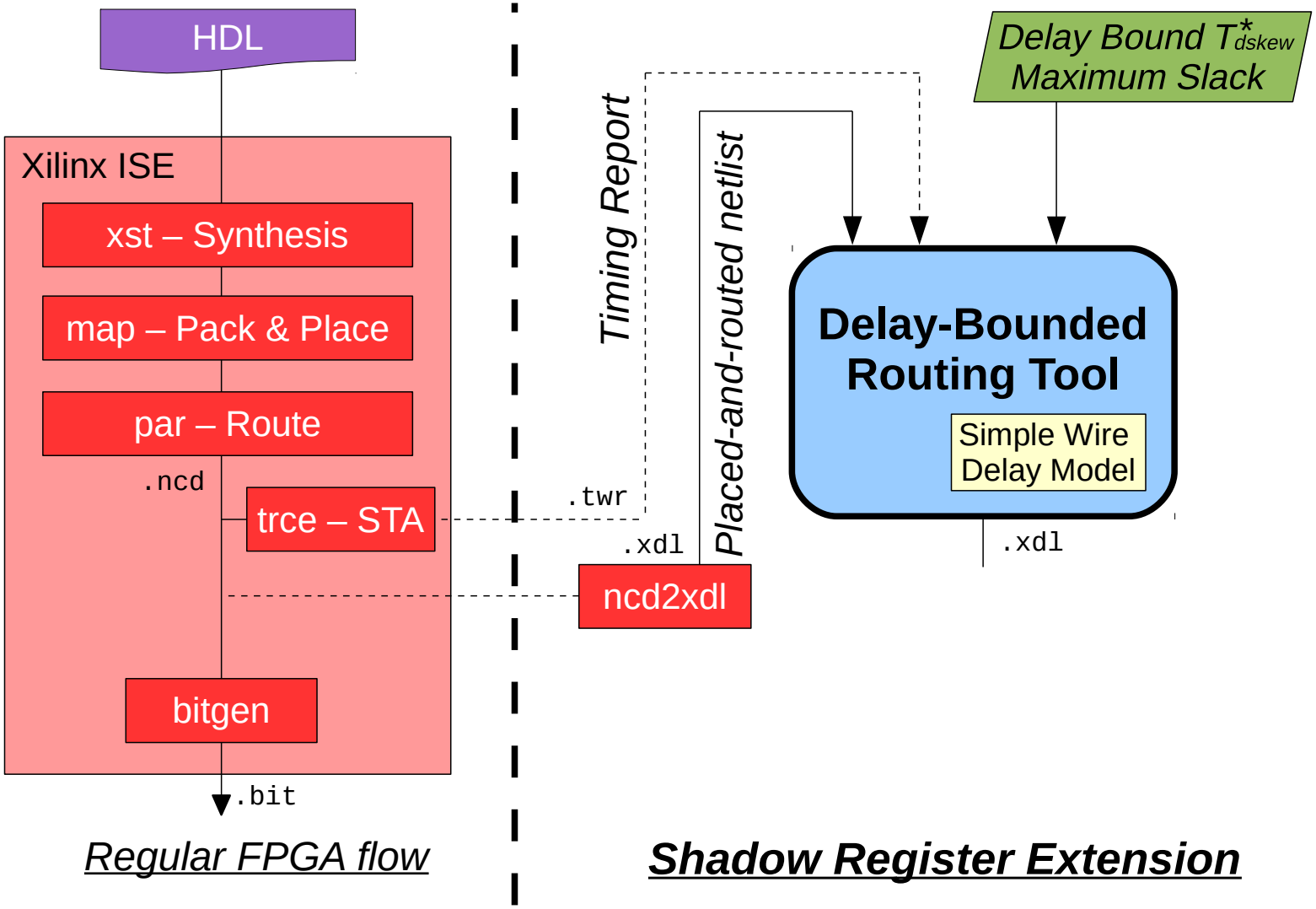
- Evaluated on Xilinx, but applicable to any vendor



Regular FPGA flow

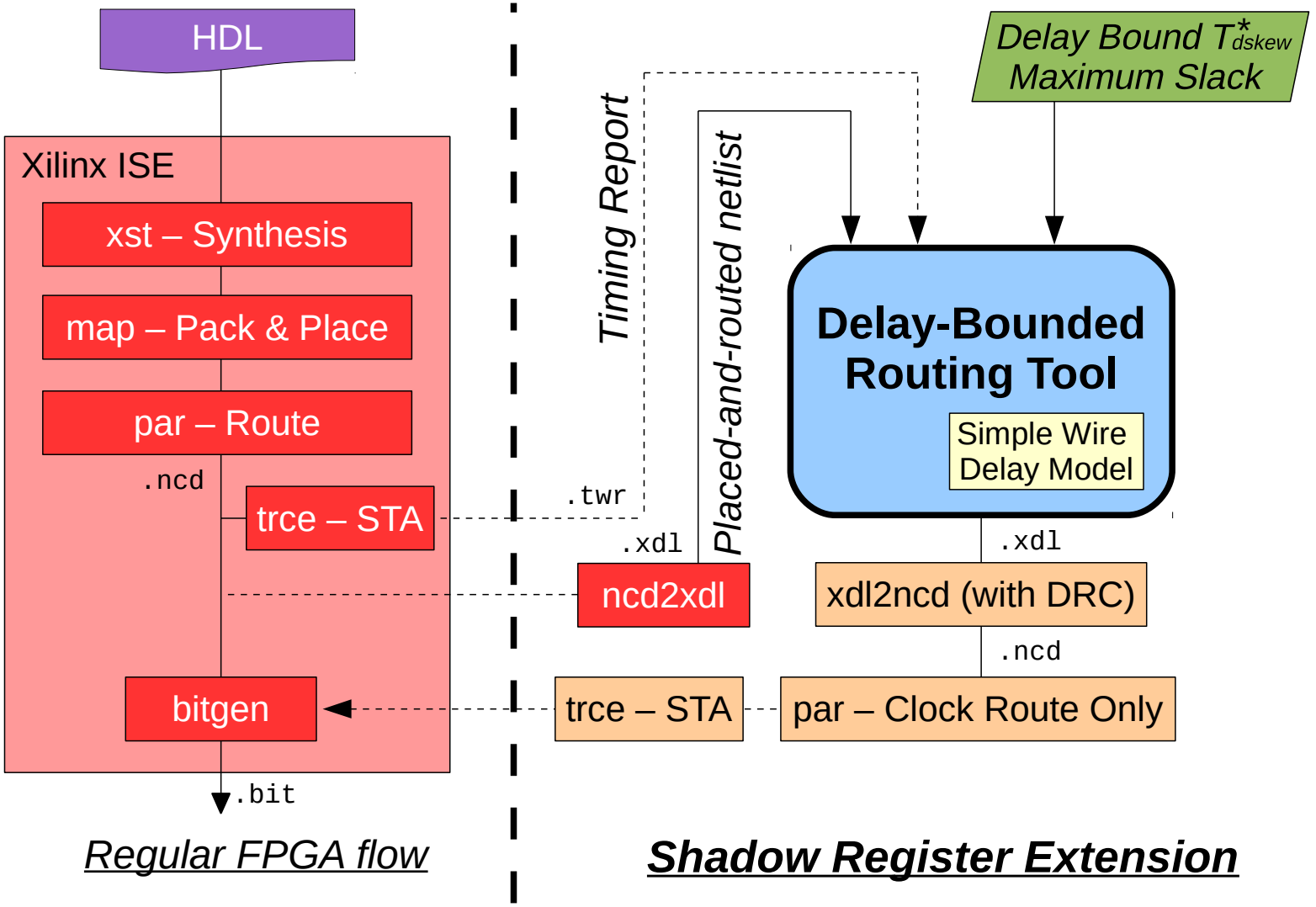
Experimental Flow

- Evaluated on Xilinx, but applicable to any vendor



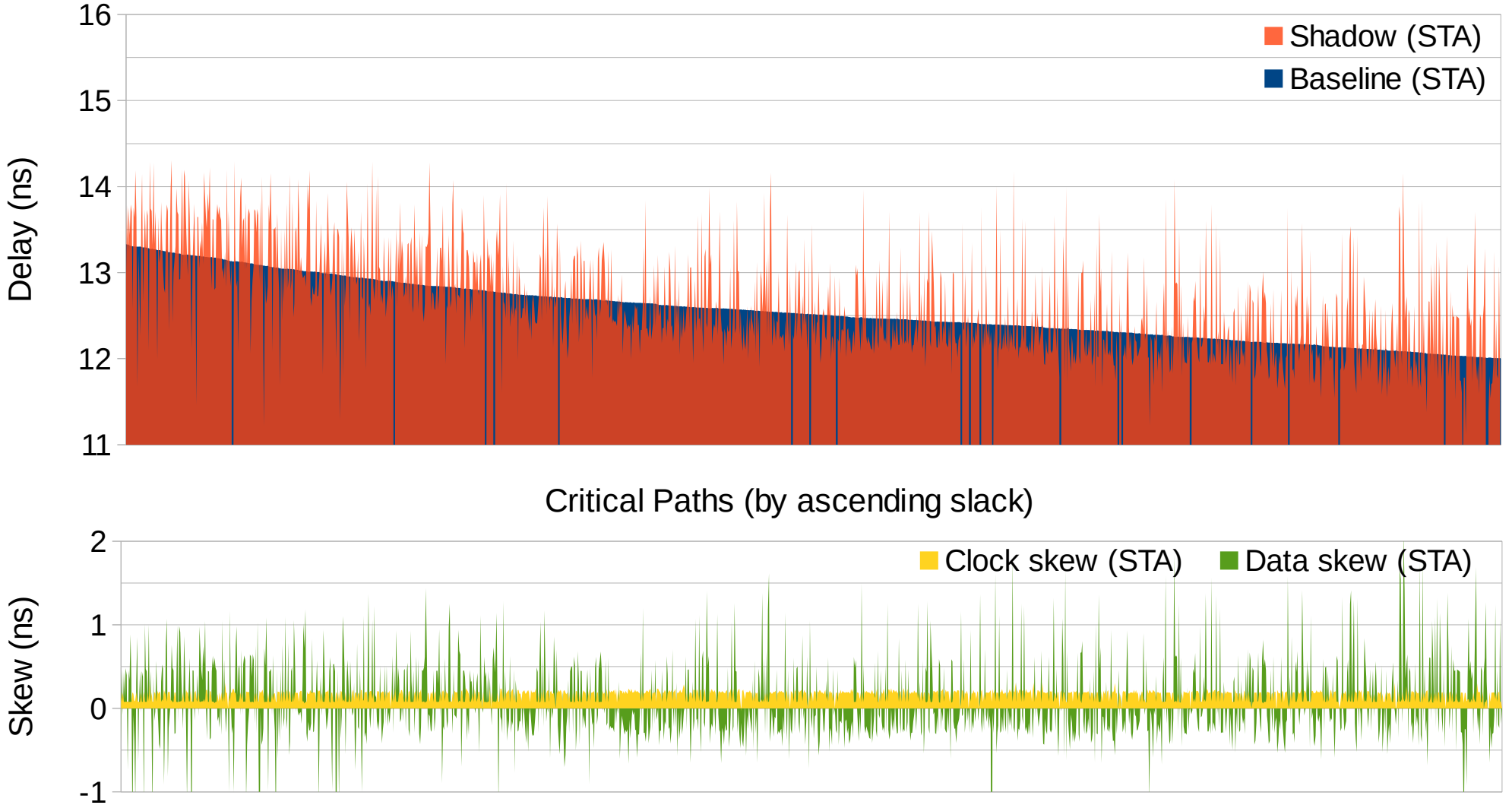
Experimental Flow

- Evaluated on Xilinx, but applicable to any vendor



Experimental Results

- Unbounded (place closest reg, Xilinx PAR):



Experimental Results

- >1ns on LEON3:

	Base	Shadow Method	
		Source	**Post P&R (this work)**
Slice util.	93.5%	88.9%	**94.5%**
LUT util.	63.5%	56.6%	**64.0%**
Register util.	20.1%	20.6%	**20.6%**
Tcrit (ns)	13.329	13.331	**13.333**
Num. critical nets	1436	1222	**1436**
Common to Base	100%	41.0%	**100%**
Shad. coverage	-	41.0%	**98.7%**
Pack & Place time (s)	1875	1974	**-**
Routing runtime (s)	1281	1241	**183**

(a) LEON3 (all critical nets within 10% of Tcrit)

Experimental Results

- >1ns on LEON3:

(ns)	Unbounded	**Delay-bounded**		
	One-Closest	One-Closest	One-All	All-All
Nets shad.	1424	1051	1417	1418
Max. |Err|	1.942	1.779	1.002	0.928
Mean |Err|	0.867	0.329	0.223	0.249
StdDev Err	0.367	0.187	0.111	0.107
Range Err	2.771	1.744	1.075	0.956

(a) From Xilinx `trce` STA.

(ns)	Unbounded	**Delay-bounded**		
	One-Closest	One-Closest	One-All	All-All
Max. |Err|	-	1.636	0.114	0.114
Mean |Err|	-	0.089	<0.001	0.001
StdDev Err	-	0.165	0.004	0.004
Range Err	-	1.636	0.114	0.114

(c) From our delay model, omitting clock skew.

Experimental Results

- >1ns on AES-x3:

	Base	Shadow Method Source	**Post P&R (this work)**
Slice util.	87.0%	86.9%	**89.8%**
LUT util.	71.4%	71.7%	**72.7%**
Register util.	10.6%	12.4%	**12.3%**
Tcrit (ns)	6.055	5.865	**6.055**
Num. critical nets	5362	7400	**5362**
Common to Base	100%	53.1%	**100%**
Shad. coverage	-	53.1%	**94.2%**
Pack & Place time (s)	1014	1408	**-**
Routing runtime (s)	479	486	**224**

(b) AES-x3 (within 40% of Tcrit)

Experimental Results

- >1ns on JPEG-x2:

	Base	Shadow Method Source	**Post P&R (this work)**
Slice util.	86.8%	88.8%	**90.1%**
LUT util.	60.9%	61.3%	**61.8%**
Register util.	41.7%	42.7%	**42.6%**
Tcrit (ns)	13.936	15.611	**13.939**
Num. critical nets	3290	3335	**3295**
Common to Base	100%	46.7%	**99.8%**
Shad. coverage	-	46.7%	**84.0%**
Pack & Place time (s)	871	911	**-**
Routing runtime (s)	748	3658	**276**

(c) JPEG-x2 (within 40% of Tcrit)

Experimental Results

- Reasons behind failure

	LEON3	AES-x3	JPEG-x2
Num. critical nets	1436	5362	3290
Nets shadowed	1417	5049	2769
i) Blocked in LE	10	313	468
ii) Blocked out LE	4	-	23
iii) Path search failed	5	-	30

  - i) and ii) require user circuit to be ripped up
  - Only iii) could be solved by better algorithms