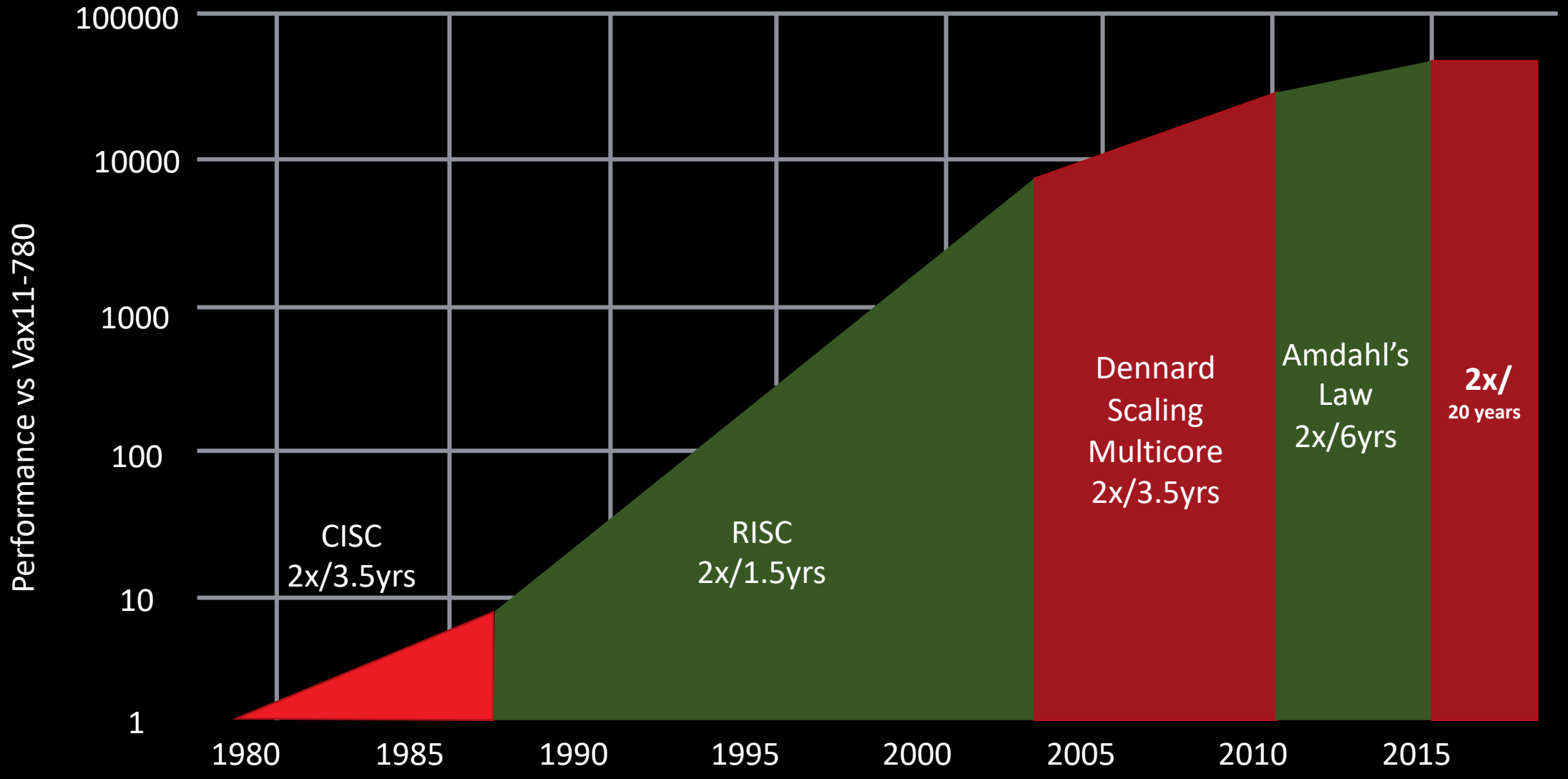
Software-like Compilation for Datacenter FPGA Accelerators

James Thomas, Chris Lavin, Alireza Kaviani
Stanford University and Xilinx Research Labs

XILINX

A New Age of Domain-specific Computing

Hennessy & Patterson



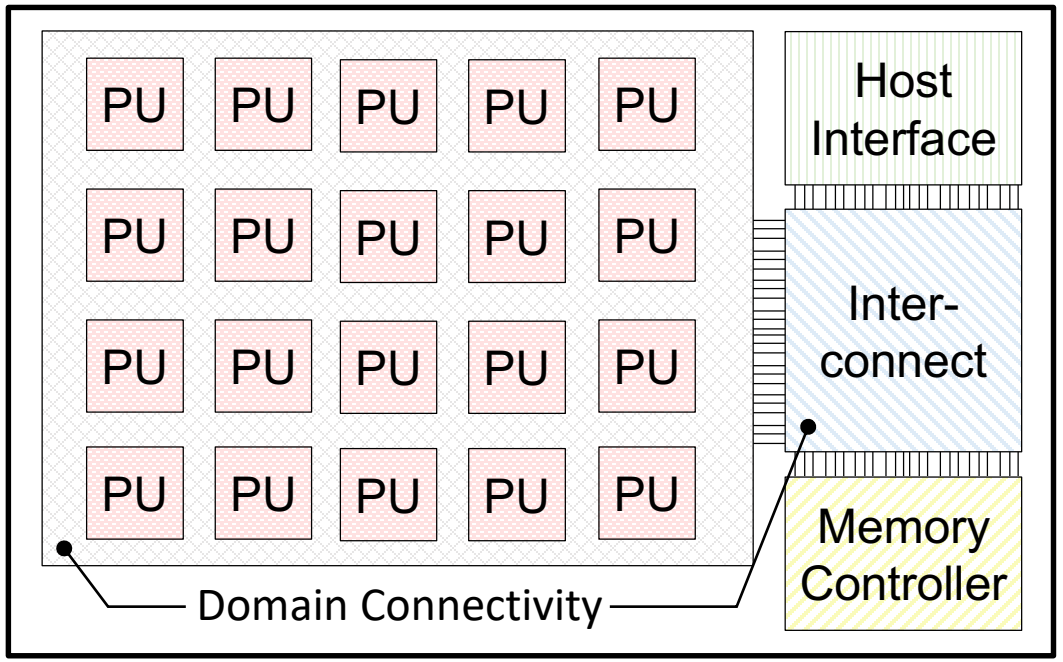
"A New Golden Age of Computer Architecture", Hennessy & Patterson, Turing Lecture

Rise of FPGAs in the Datacenter

- Large amount of interest from Amazon, Microsoft, and others
- Many massively parallel datacenter workloads that should work well on FPGAs
- But difficult for software programmers to harness them

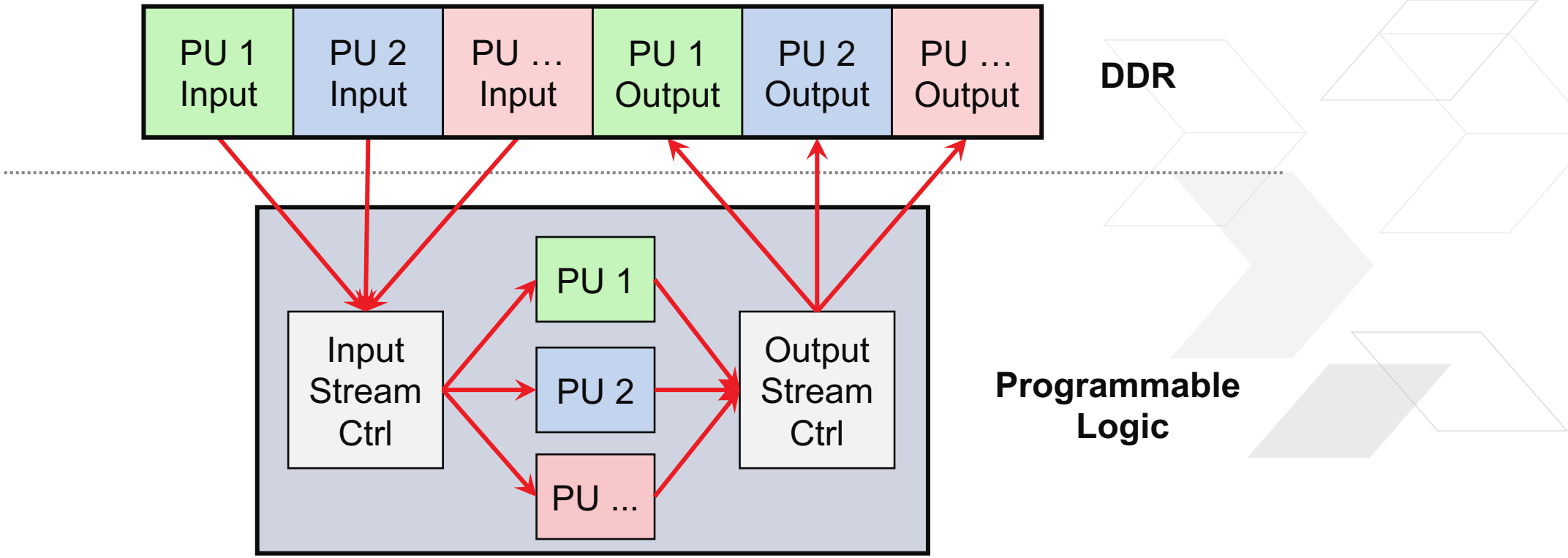
Common Pattern in Datacenter FPGA Designs

- Many identical processing units (PUs)
- Some sort of “memory controller” to facilitate communication among the processing units and to DRAM (data analytics, machine learning, etc.)



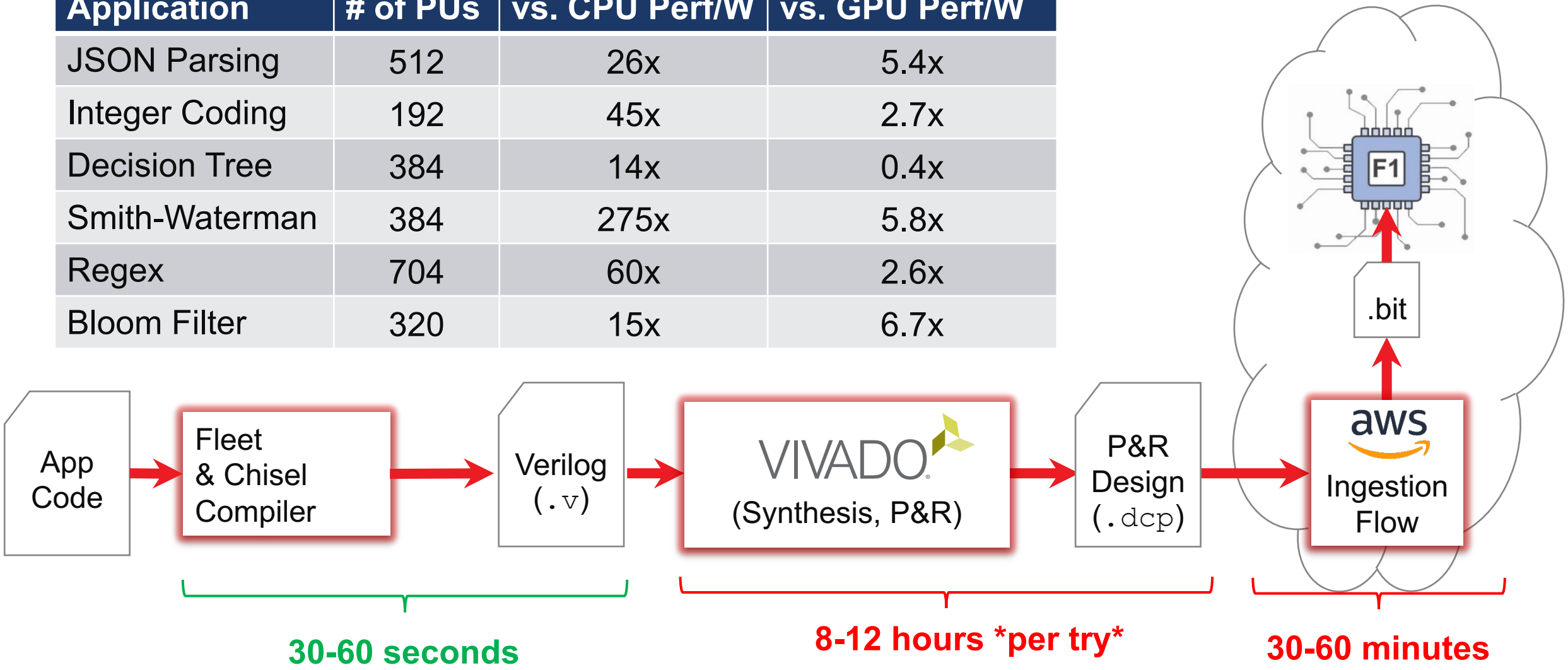
Fleet Computing Domain

- As an example, I previously worked on a system called Fleet in the data analytics / stream processing domain
- Included a DSL for PUs and targeted Amazon F1



Fleet: Good Performance, Slow Compilation

Application	# of PUs	vs. CPU Perf/W	vs. GPU Perf/W
JSON Parsing	512	26x	5.4x
Integer Coding	192	45x	2.7x
Decision Tree	384	14x	0.4x
Smith-Waterman	384	275x	5.8x
Regex	704	60x	2.6x
Bloom Filter	320	15x	6.7x

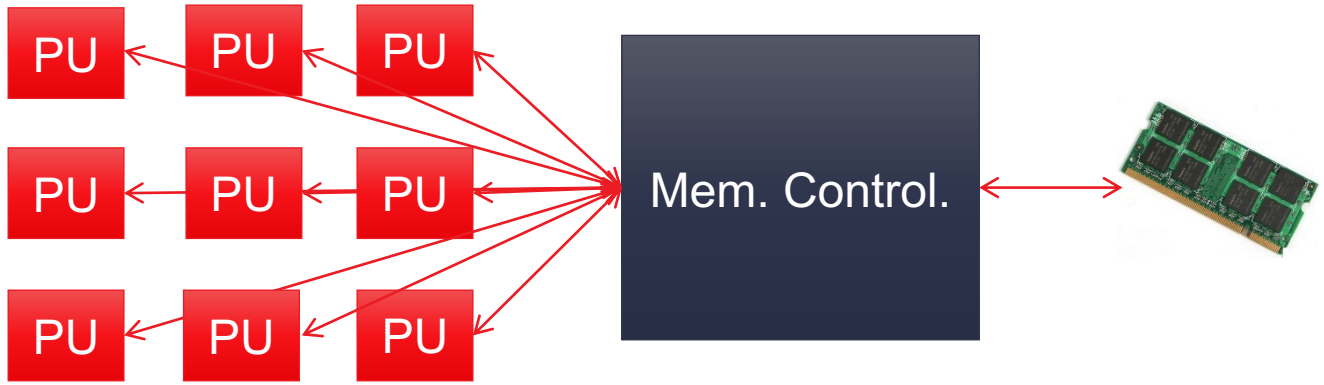


Fleet: A Framework for Massively Parallel Streaming on FPGAs, ASPLOS 2020

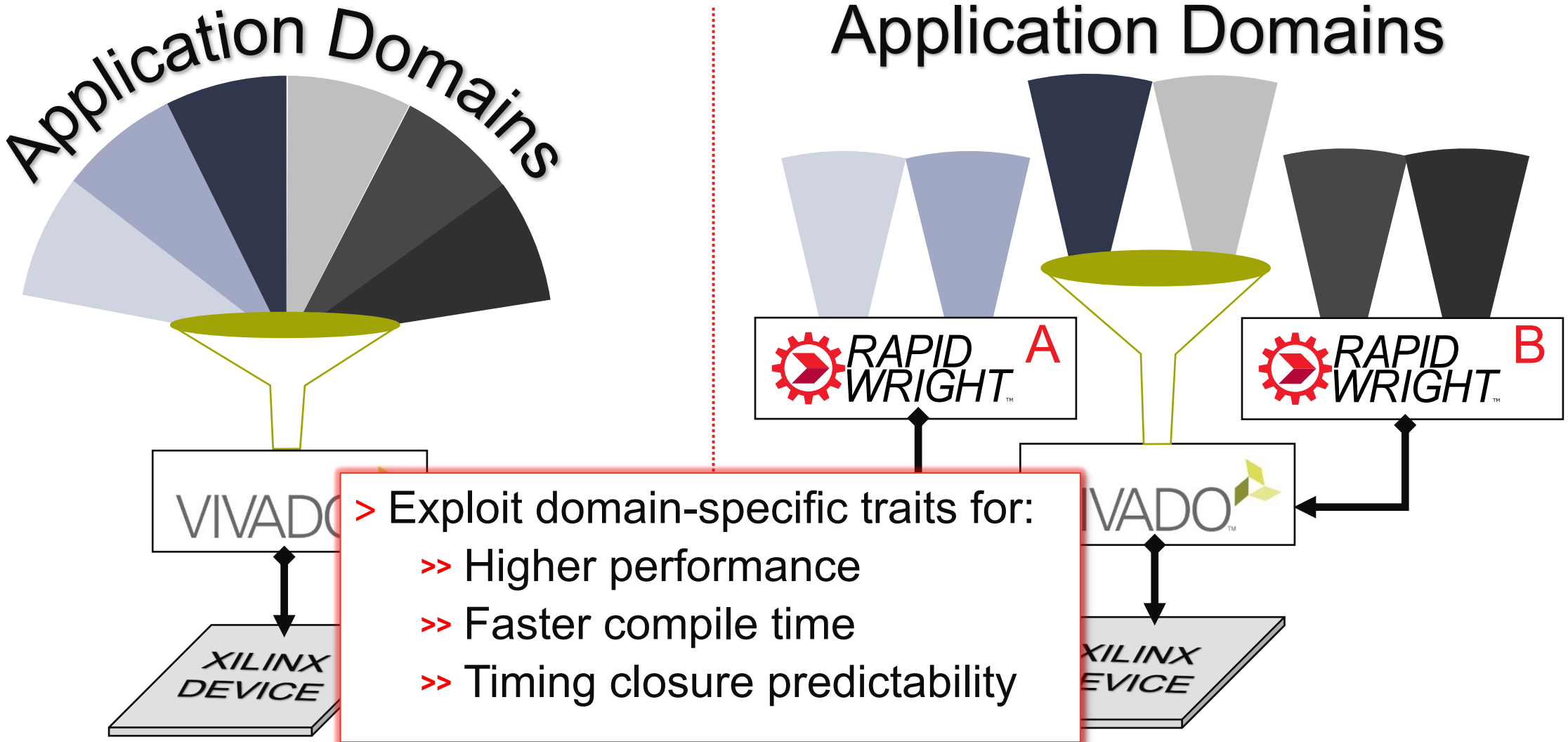
XILINX

Datacenter FPGA App Compilation

- Memory controller often doesn't change much for a class of applications – wasted work in redoing its place and route for each app
- Processing units identical – significant wasted work in redoing place and route for each one
- How can we avoid redoing work?



Domain Specific Era Needs Domain Specific Backends



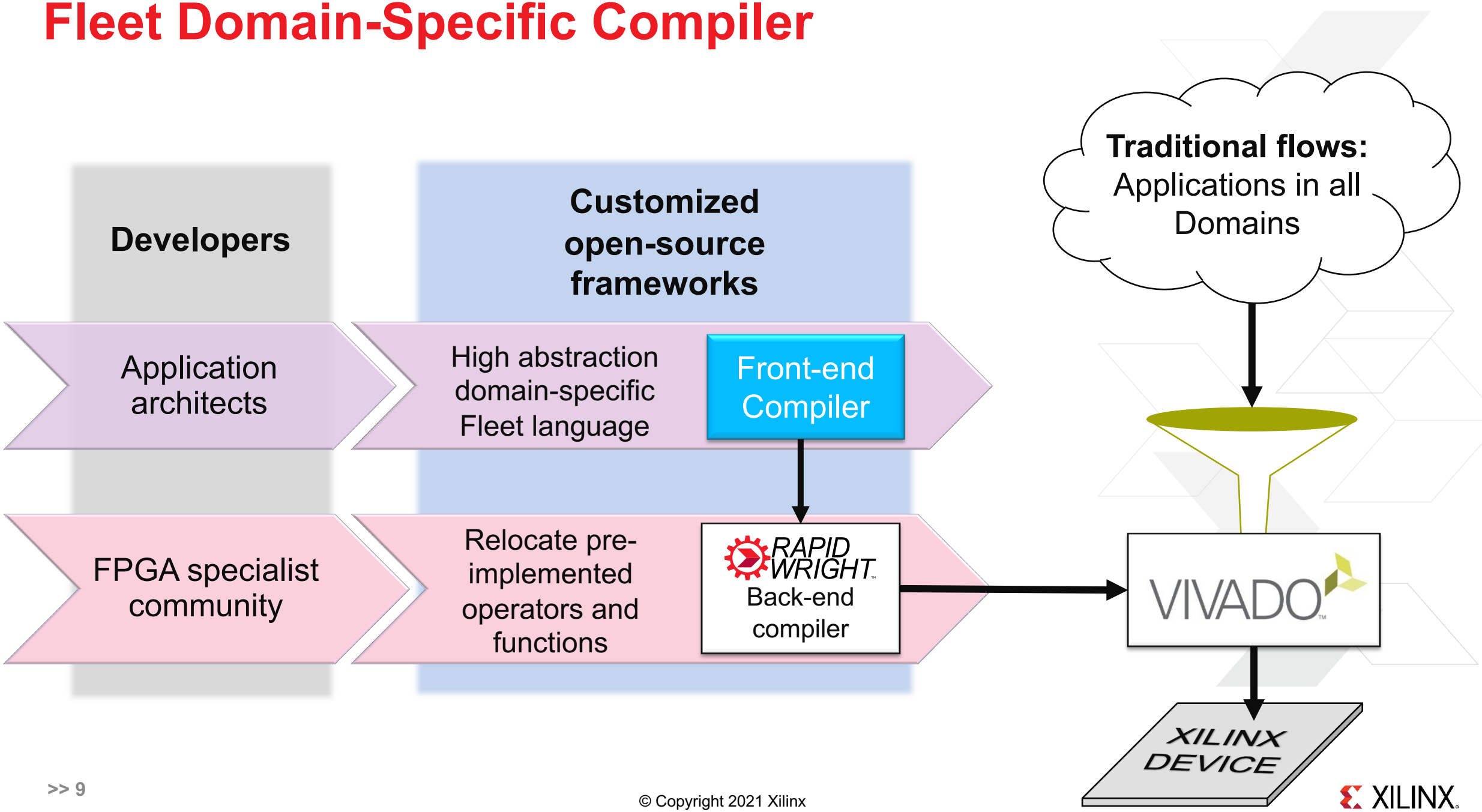
> Exploit domain-specific traits for:
>> Higher performance
>> Faster compile time
>> Timing closure predictability

Vivado must ***generalize*** solutions

RapidWright can ***specialize***

XILINX

Fleet Domain-Specific Compiler



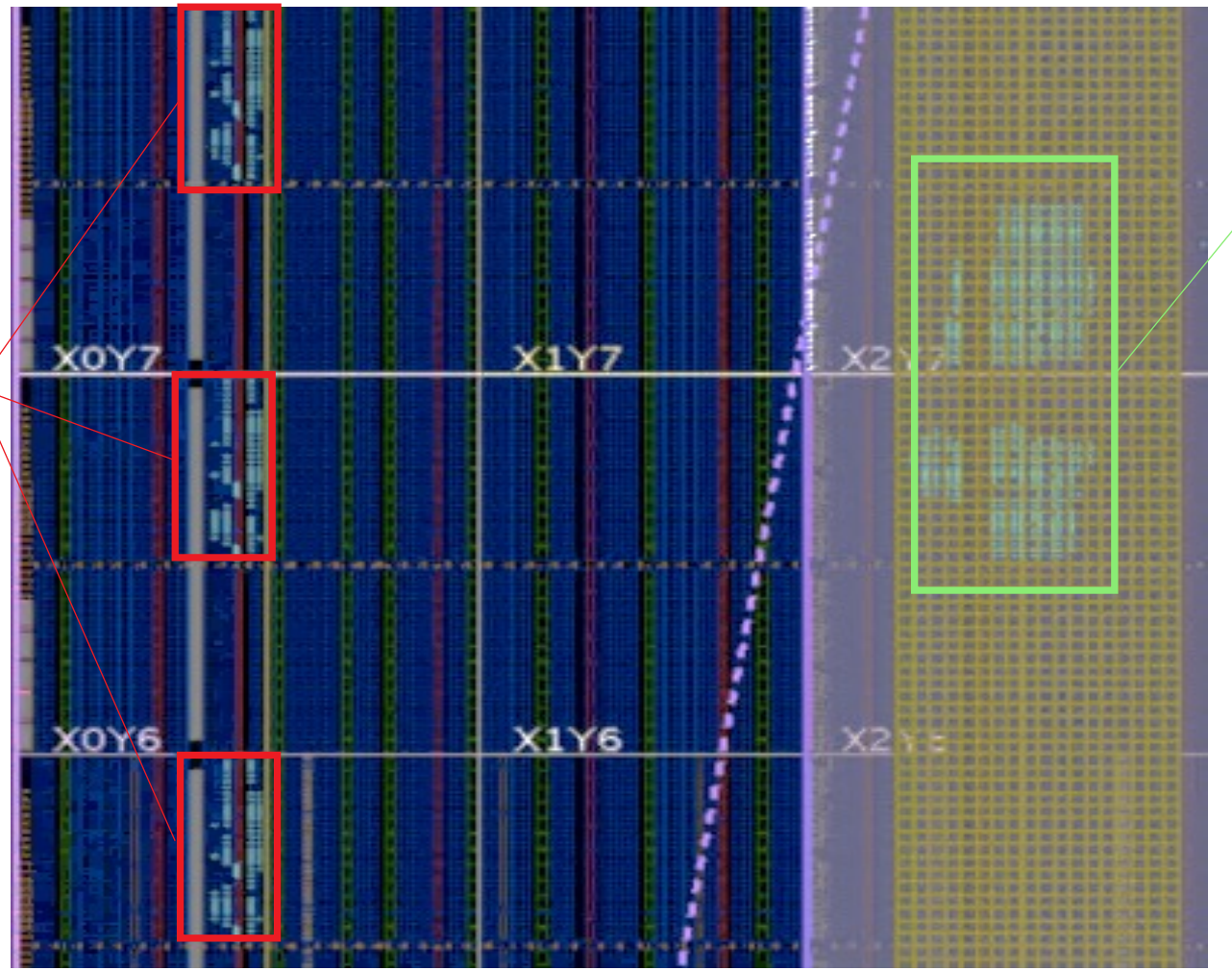
XILINX

Goal

- Fast compilation by:
  - Reusing compilation for replicated PUs
  - Take memory controller (shell) from a pre-implemented library

Simple Solution: Vivado Out-of-context Flow

Replicated PUs

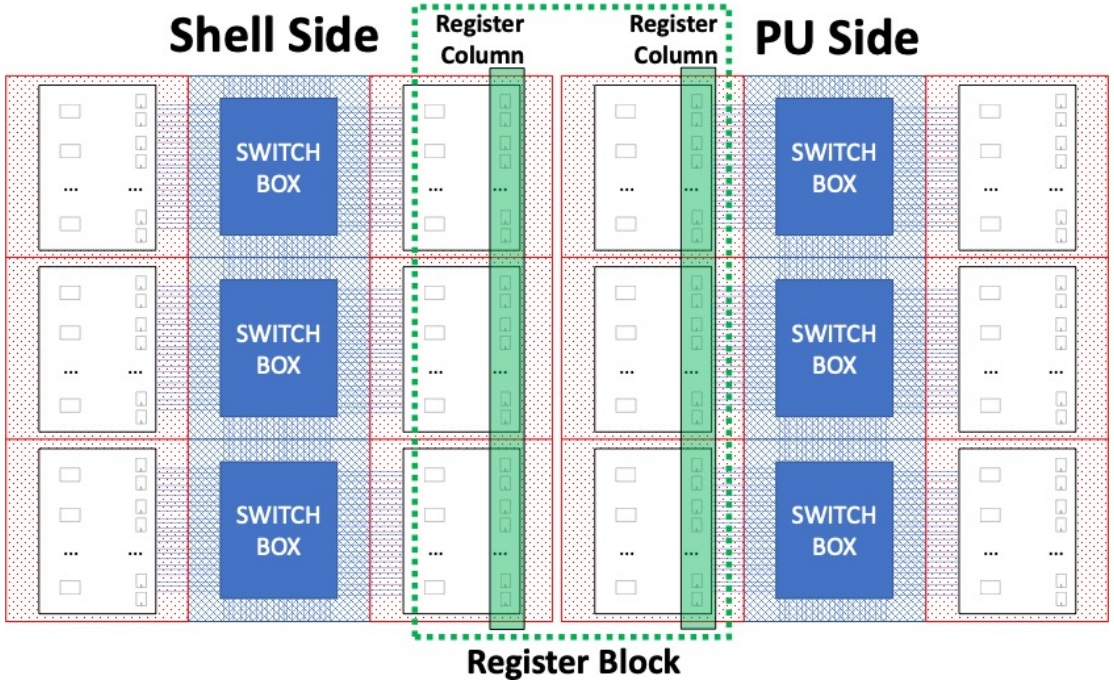


Pre-implemented memory controller (shell)

XILINX.

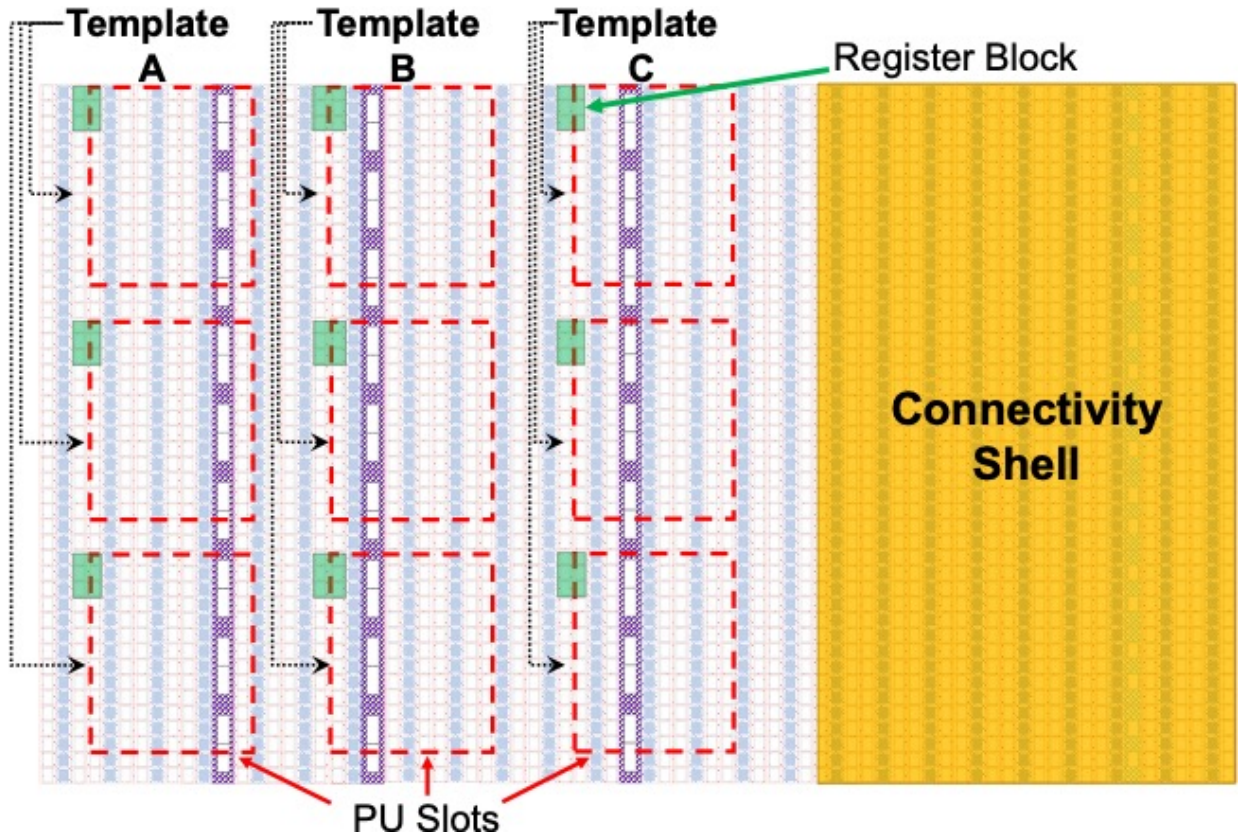
Issue: Shell-to-PU Routing

- Problem: Vivado routing from replicated PUs to shell takes time (1-2 hours or more)
- Solution: Shell is pre-implemented, so route it to a register block next to each PU location ("slot") ahead of time
  - Use two connected columns of registers & pblocks to ensure shell routes don't cross into PU slots



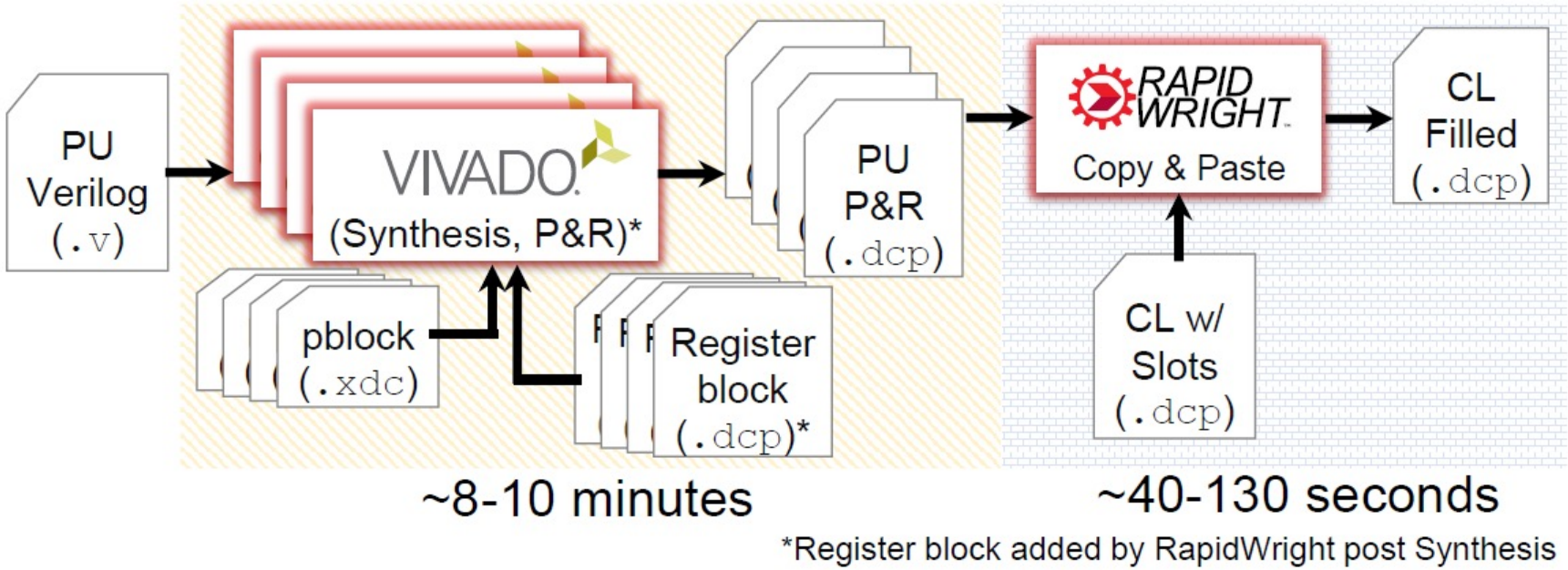
Example Shell

> Slot resource layout can be different per slot column (but resource counts are same) – requires separate implementations



Online PU Flow

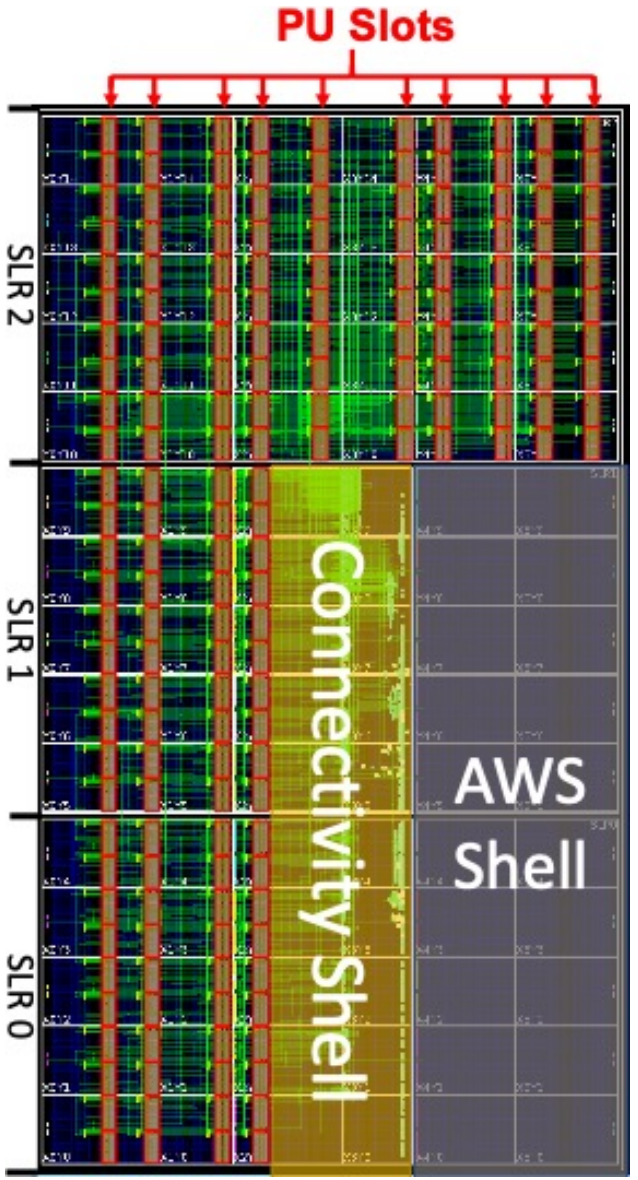
> Generate PU implementation for each slot column & replicate implementations



Results

PU	Interface Size (bits)	# Logic Cells	PU Template Implementation Runtime	RapidWright PU Replication Runtime	Our Flow Total Runtime	Standard Flow Runtime	Speedup
Dot	46	71 (incl. DSP)	9m4s	1m1s	10m5s	82m50s	8.2×
Counter	22	109 (incl. BRAM)	10m37s	0m37s	11m14s	87m25s	7.9×
Summer	46	138	8m24s	0m41s	9m5s	82m51s	9.1×
JSON	22	352 (incl. BRAM)	10m57s	0m53s	11m50s	94m30s	8.0×
Time Series Pred.	22	512	8m31s	0m51s	9m22s	95m38s	10.2×
KNN	46	800 (incl. distr. RAM & DSP)	8m25s	1m41s	10m6s	111m15s	11.0×
Integer Coder	46	1119 (incl. distr. RAM)	9m35s	2m7s	11m42s	117m19s	10.0×

180-Slot Shell



Area Tradeoff

- Previously able to get 500+ PU's with standard flow
- Still have room to add more slots
- Can still beat GPU with 180 PU's in some cases, may be enough for some users
- For others, this can be a fast flow for prototyping, can use standard flow once design is finalized

Retrospective on Vivado

- Vivado needs to start up/run more quickly for faster online flow
- Needs more low-level APIs to precisely control behavior
  - Could do something simpler/more direct than using register block for shell/PU isolation
- RapidWright achieves both goals but needs its own placer/router/etc. to be on par with Vivado

Additional Speedup with Open-Source Tools

PU	Yosys+nextpnr Runtime	# Logic Cells (Yosys synth.)	Our Flow Runtime
Dot	79s	232	544s (4.3×)
Counter	78s	231	637s (5.9×)
Summer	87s	230	504s (4.3×)
JSON	97s	560	657s (4.7×)
Time Series Pred.	100s	1248	511s (3.7×)
KNN	96s	1199	505s (3.1×)
Integer Coder	108s	2803	575s (3.0×)

Conclusion

- Fast compilation system for modular datacenter designs (~10x speedup)
- Open source at https://github.com/jjthomas/Fleet-Floorplanning