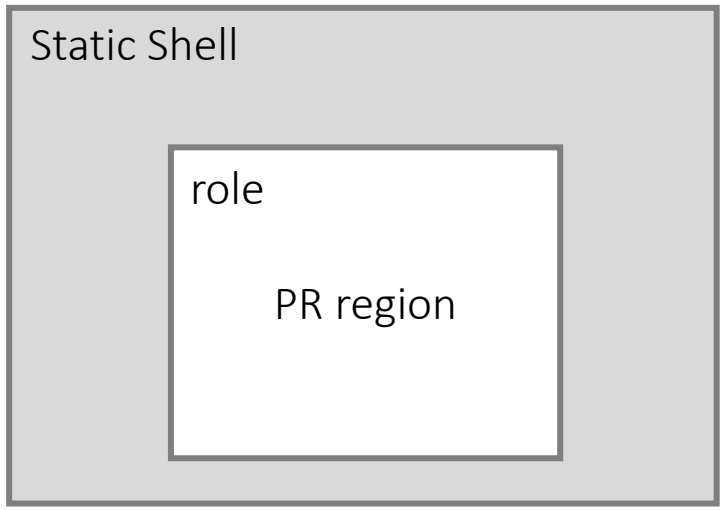
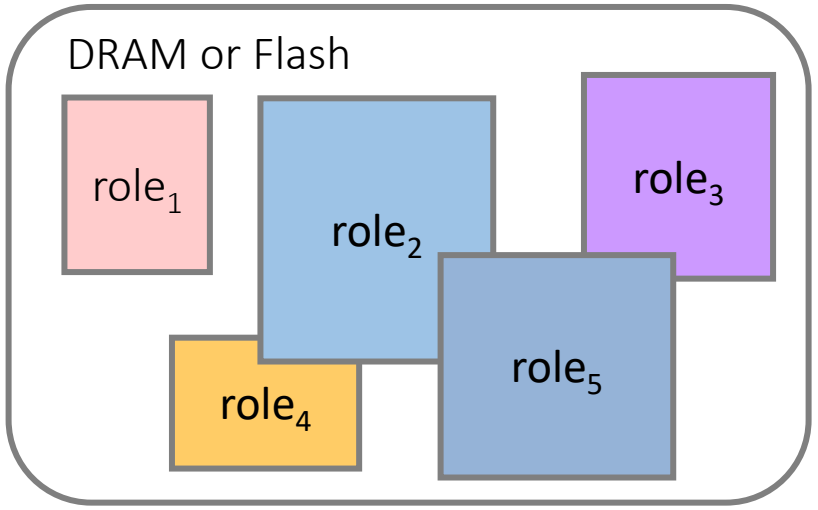
Shashank Obla (CMU)

Partial Reconfiguration: A deeper dive

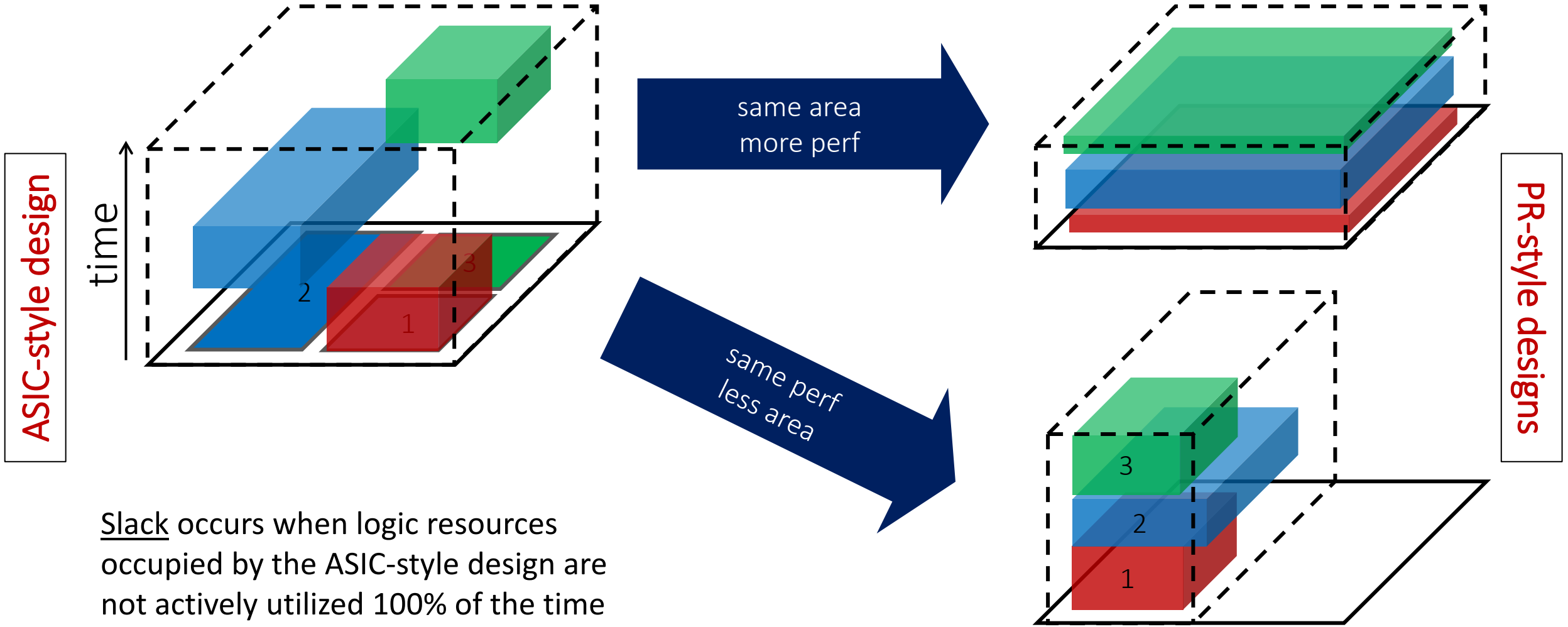
Coarse grained PR: Role-and-Shell





- One PR region enclosed by static **shell** for I/O and isolation
- At runtime, **role** designs with different functionalities loaded in PR region
- Vanilla PR Usage: Infrequent reconfigurations (hours or days between swapping)
- Shell remains active during reconfigurations

Fine grained PR: Think Time-Area Volume



Slack occurs when logic resources occupied by the ASIC-style design are not actively utilized 100% of the time

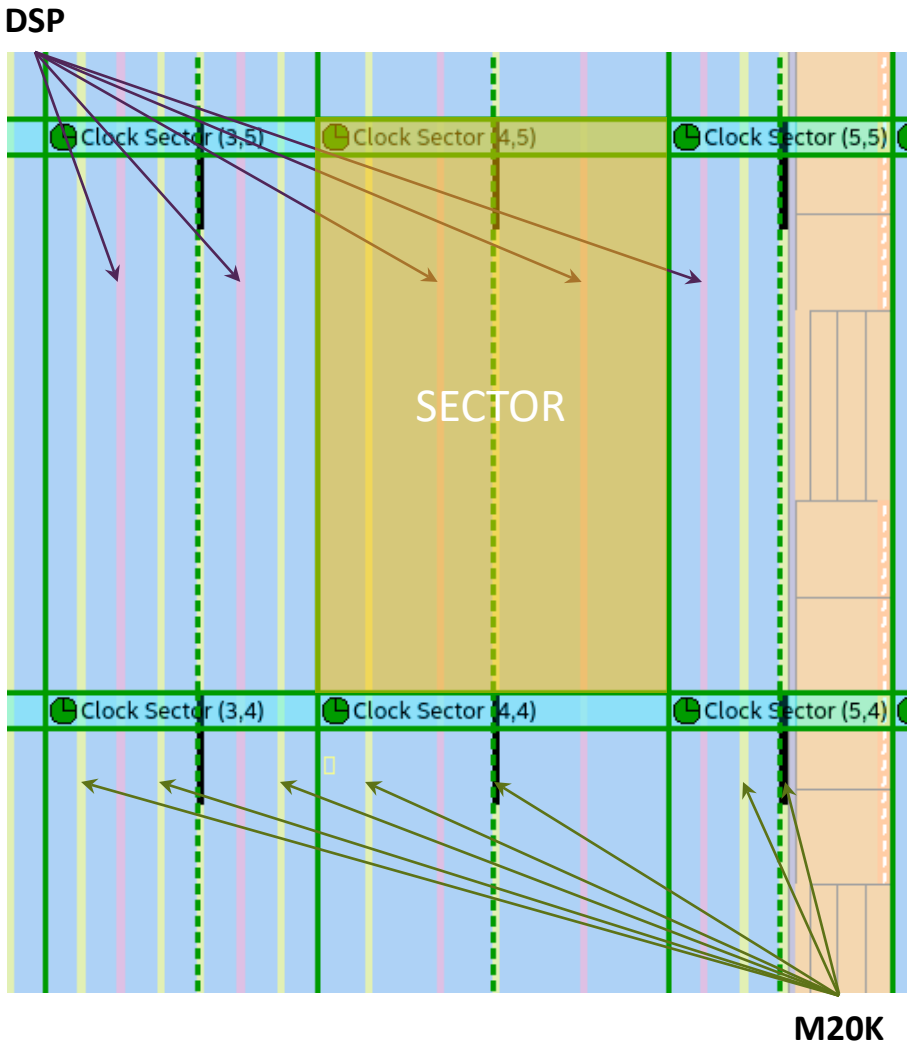
M. Nguyen, N. Serafin and J. C. Hoe, "Partial Reconfiguration for Design Optimization," *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, 2020, pp. 328-334, doi: 10.1109/FPL50879.2020.00061.

Why can’t everyone do it yet?

~~Why isn’t everyone doing it?~~

Background: Some Stratix 10 Architecture

- Fabric organized into sectors
- Programming through Secure Device Manager (SDM)
- Heterogeneity in sectors
- Routability also matters
- Single PR Controller interface



PR Time

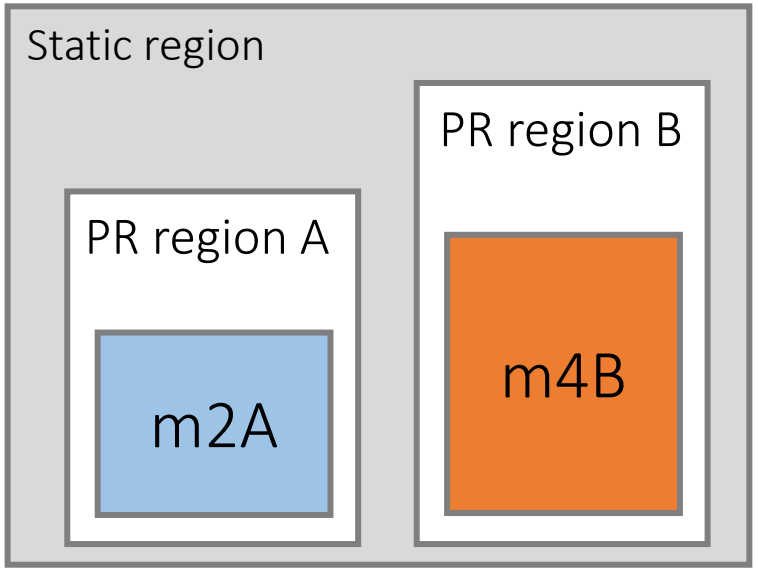
From Pigasus

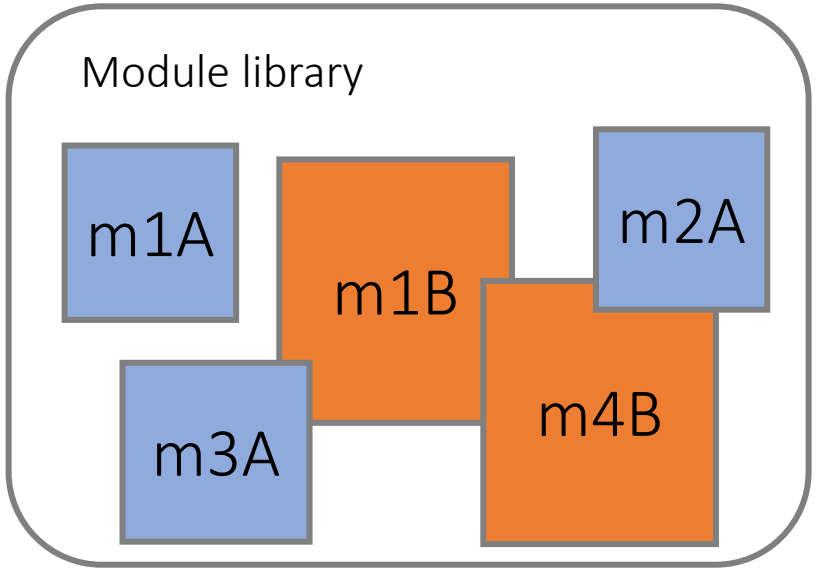
Module	Sectors	Logic (ALMs)	Registers (Dedicated)	Memory (M20K)	DSPs
Flow Table	1.5	81%	24%	100%	0%
Hash Table	12	15%	10%	6%	90%
1MB Cache	5	10%	1%	78%	0%

Values obtained on Bittware 520N-MX FPGA Board with Stratix 10 MX2100 - 1SM21CHU2F53E2VG with default settings from the BIST design and the partial reconfiguration bitstreams residing in the on-board DDR4

PR Floorplanning

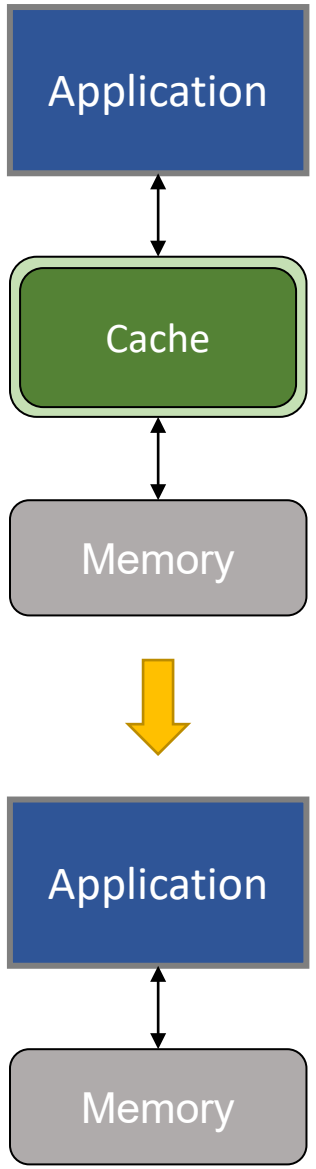
- Prior fixed floorplanning leads to fragmentation
  - Internal and External
- Lack of Relocatability
  - Each region is considered different



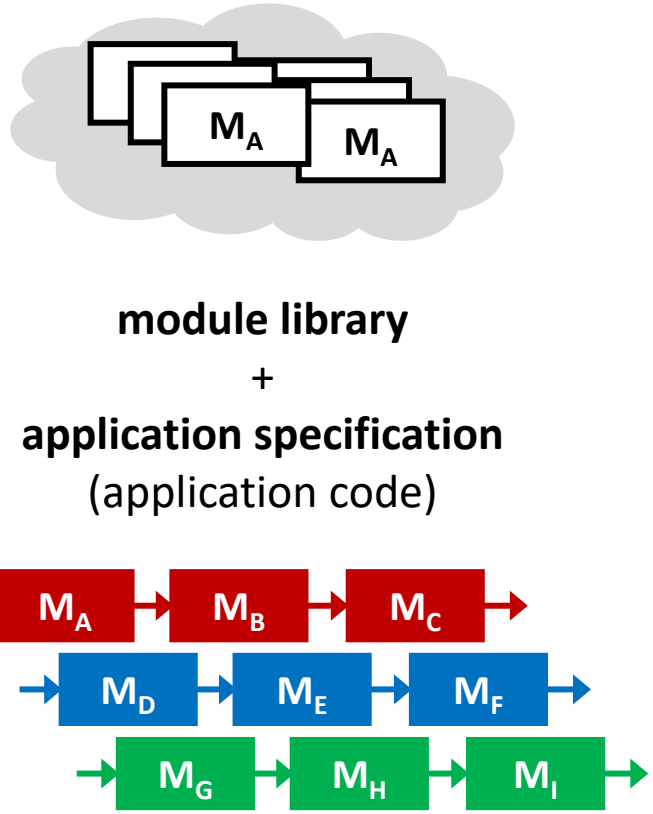


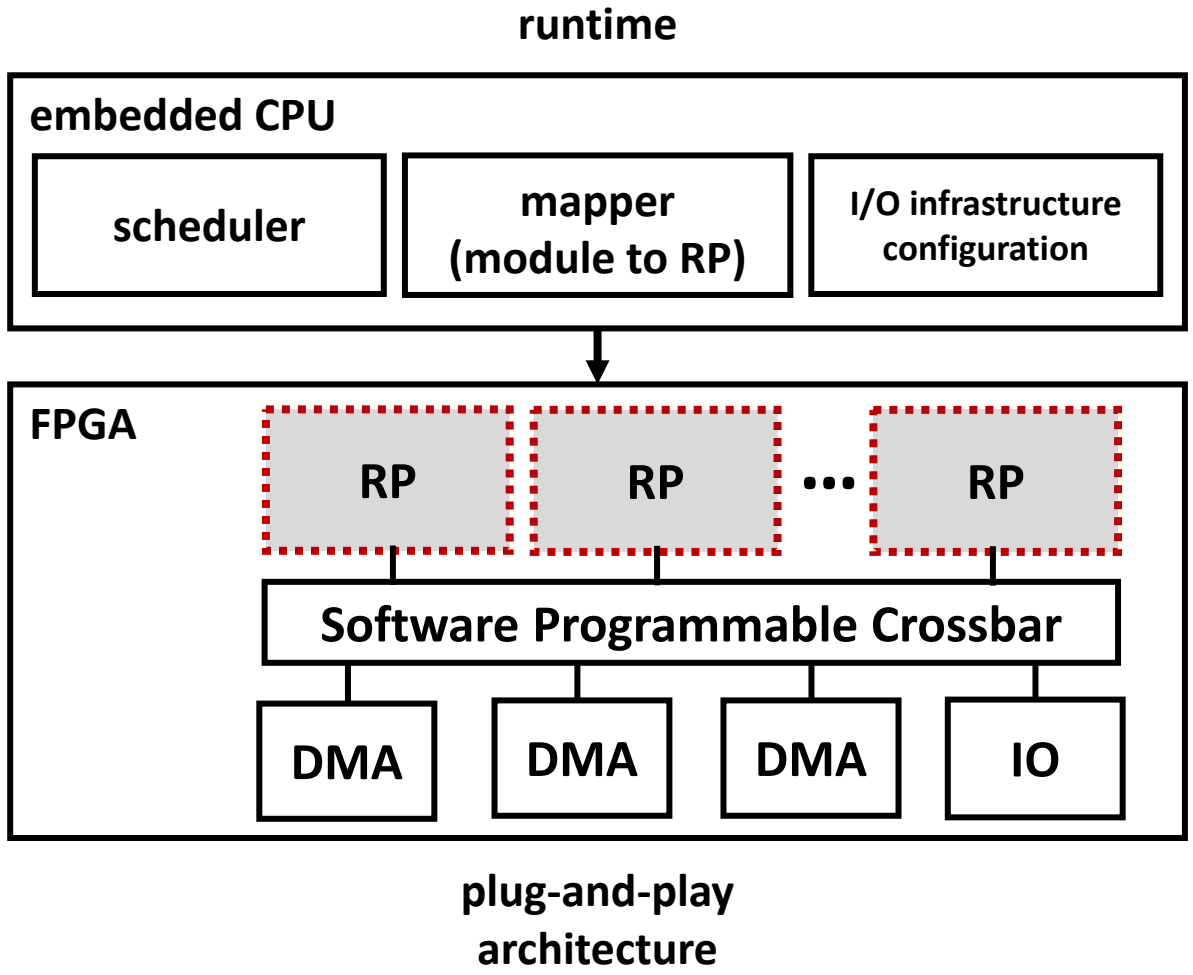
Context Saving on FPGAs

- Need ability to stop and start the module
- Not an issue for stateless systems
- Example: direct-mapped write-through cache
  - What is its internal state?
  - Where do you store it when it's gone?
  - Whose responsibility is it?
  - What if it was a write-back cache?
- Issues exacerbated by fine granularity



Framework for Dynamic Execution of Vision Apps





M. Nguyen, R. Tamburo, S. Narasimhan and J. C. Hoe, "Quantifying the Benefits of Dynamic Partial Reconfiguration for Embedded Vision Applications," 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 2019, pp. 129-135, doi: 10.1109/FPL.2019.00029.

Hardware Design for Compute using PR

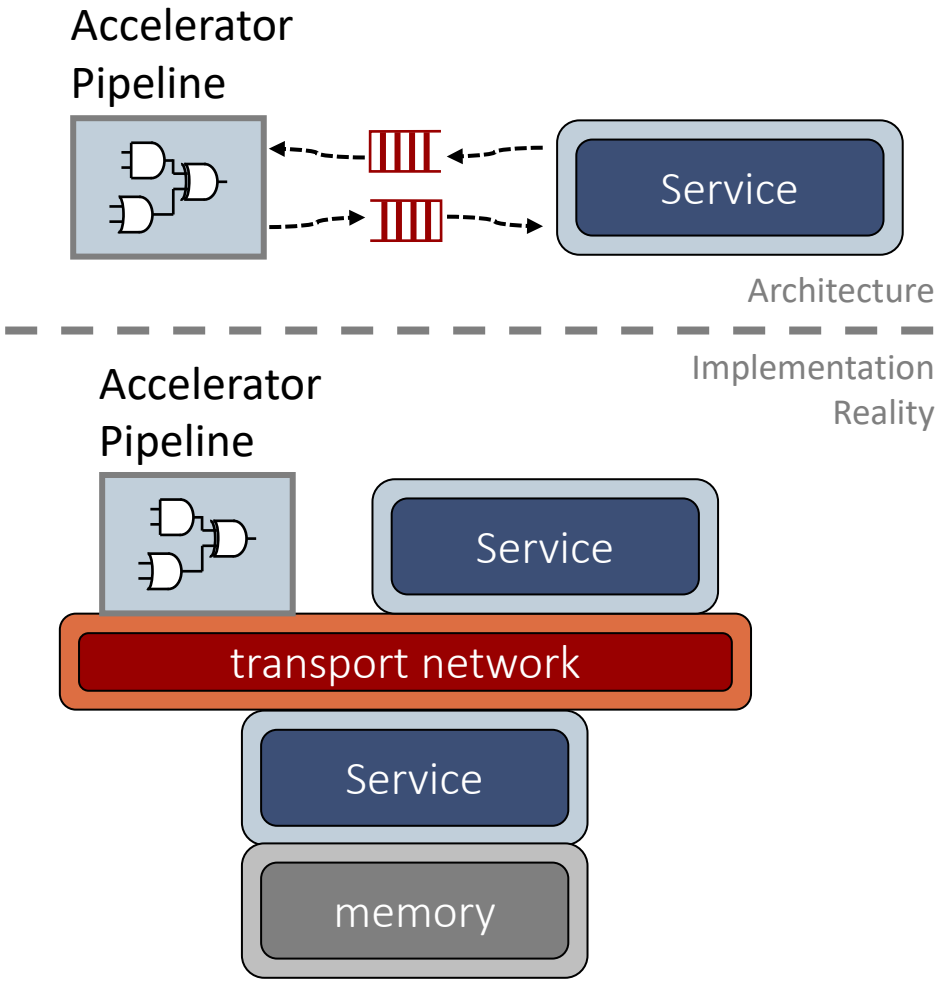
- A general framework, rich enough to map problems in our domain
- PR lets us generalize and specialize at the same time

1. PR boundaries are special – should not corrupt the system
   ⇒ Latency-Insensitive channels
2. Need a rigorous methodology to identify PR modules
   ⇒ Come up with a design paradigm

- A good framework does the above and goes beyond

Service-Oriented Architecture

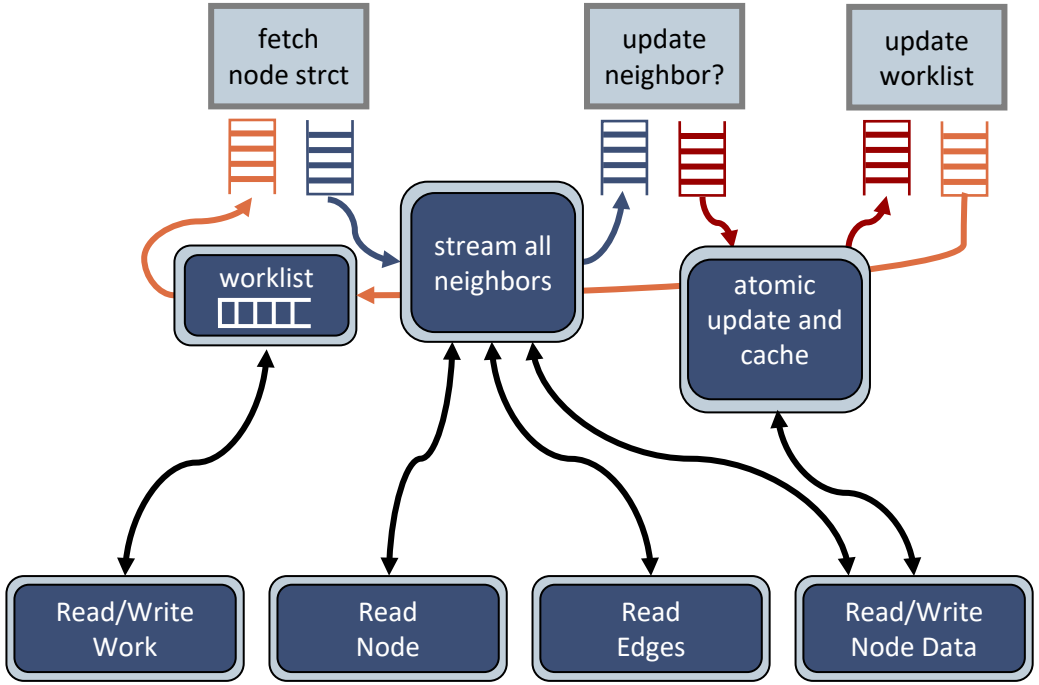
- Abstracting data operations as a service
- Services encapsulate high-level memory operations hiding complexity
- Interface standards force reusability and portability
- Composability without loss of efficiency



J. Melber and J. C. Hoe, "A Service-Oriented Memory Architecture for FPGA Computing," 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 2020, pp. 91-97, doi: 10.1109/FPL50879.2020.00025.

Service Memory System API

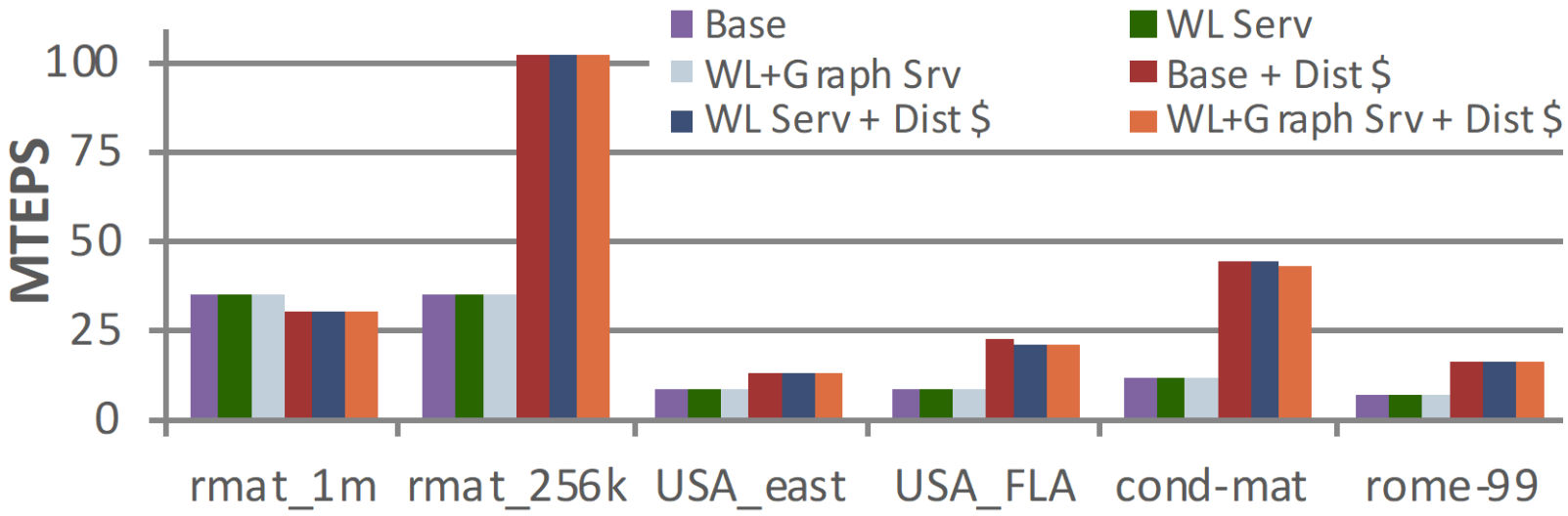
- Catalog (encapsulate expertise)
  - describes a service's functionality
  - interface channels
  - operations ascribed to each channel
  - parameters
  - message structure



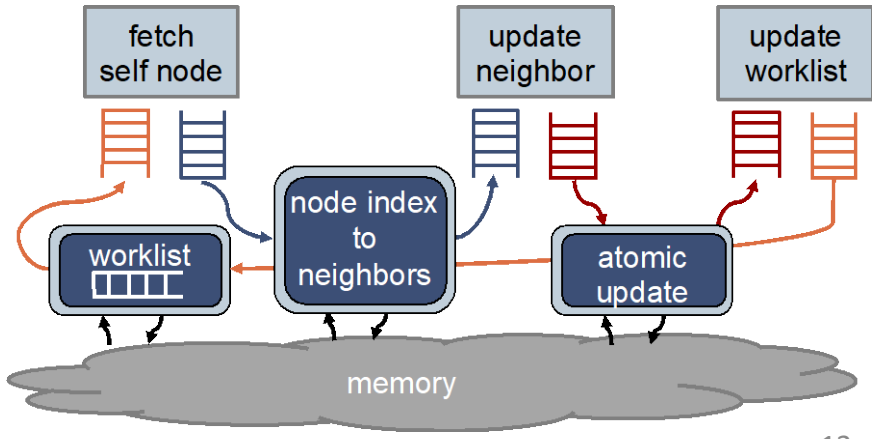
Breadth-First Search design using Services

- Registry (ease composition)
  - instantiates services and user modules
  - specifies the logical organization of connections between modules

Evaluating Service Abstraction



Resource Utilization			
Configuration	ALM	Registers	SRAM
Base	141495	163643	10590080
Worklist Service	141598	164131	10590080
Worklist + Graph Services	138817	164895	10590080



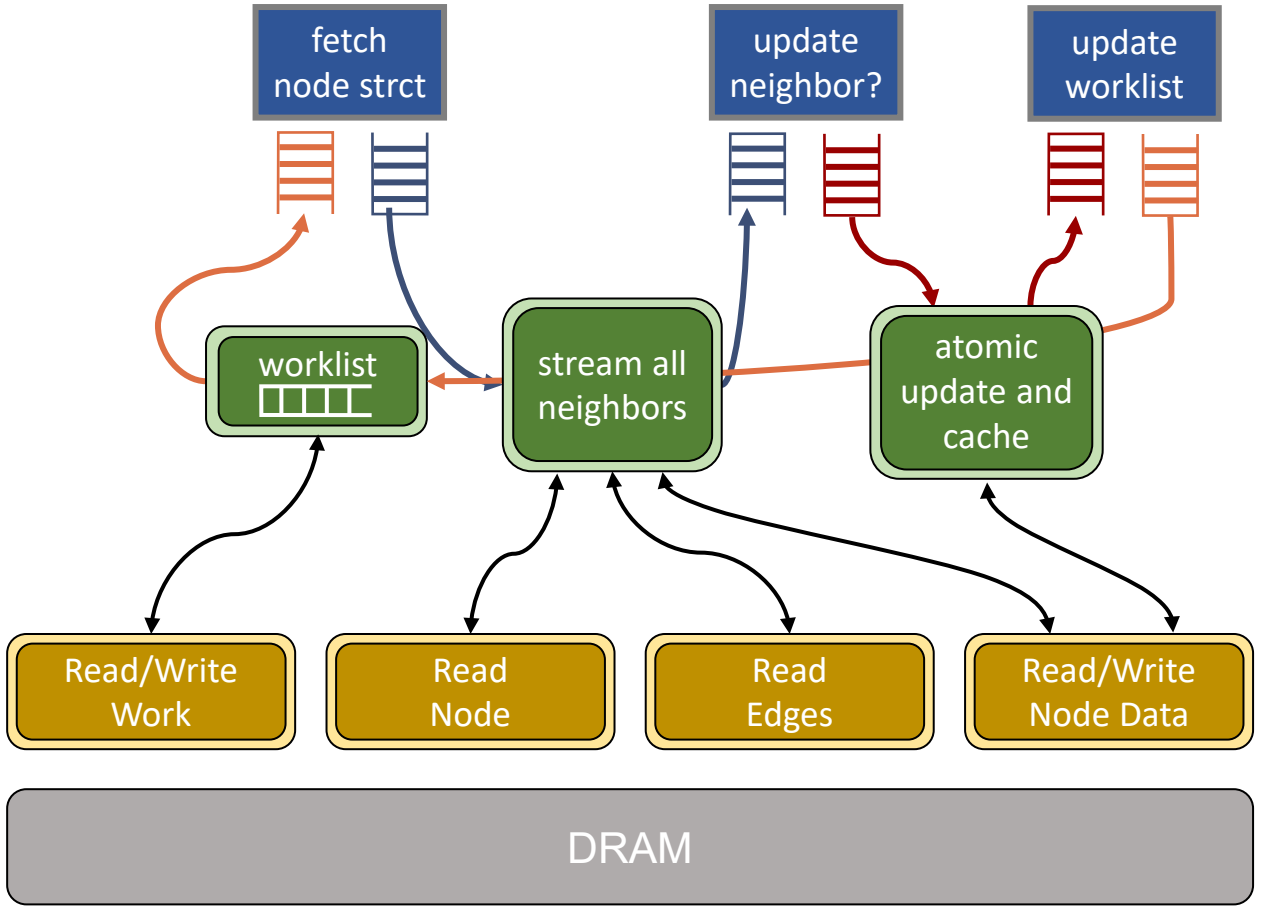
Composability across the stack

APPLICATION

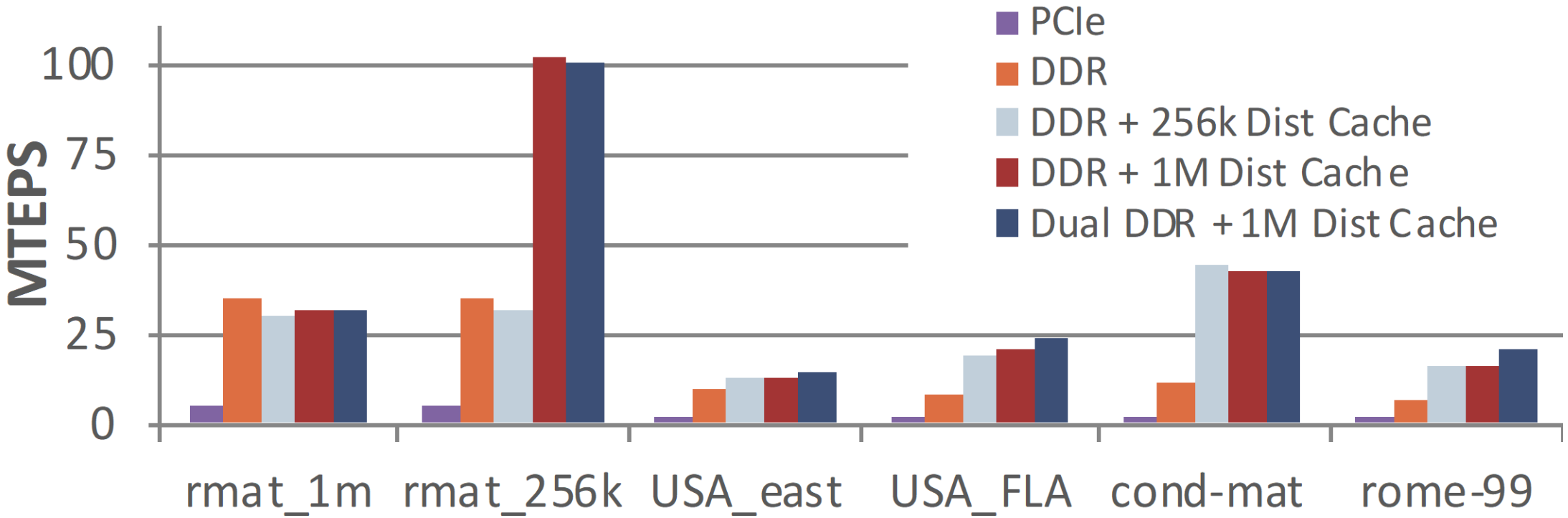
DATA-STRUCTURE SERVICES

MEMORY SERVICES

MEMORY



Tradeoff even in low-level services



Same user logic transparently served with different underlying options

Looking Ahead

- We have seen a lot of these questions before
- RV3: Improved architectural support for PR
  - Faster PR: Reduce the performance hit of switching
  - Improved relocatability by sectorized fabric
  - NoC carries messages across latency-insensitive channels
- Use PR to bring specialization within a generalized framework

Partial Reconfiguration is key to enabling dynamic scheduling, multitenancy, and over all virtualization