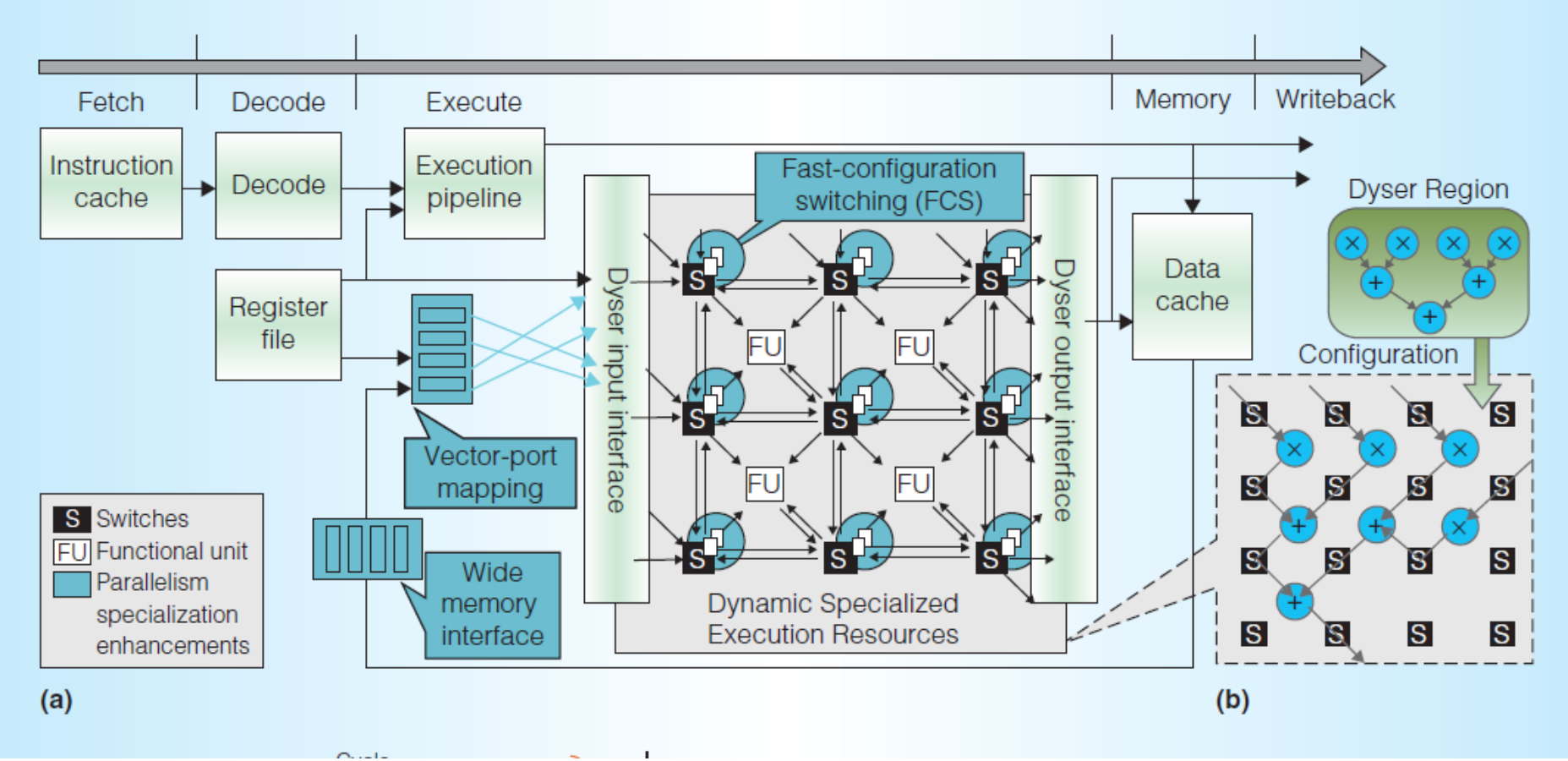
Lecture 17
Coarse-Grained Reconfigurable Computing

Joel Emer and Daniel Sanchez

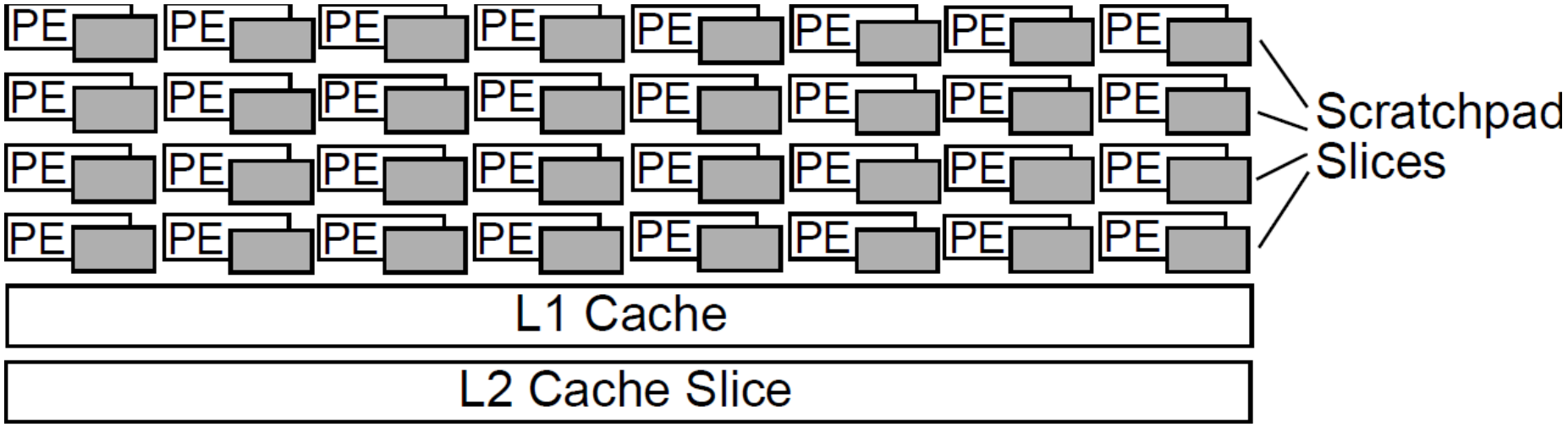
6.888 Parallel and Heterogeneous Computer Architecture
Spring 2013

Massachusetts Institute of Technology

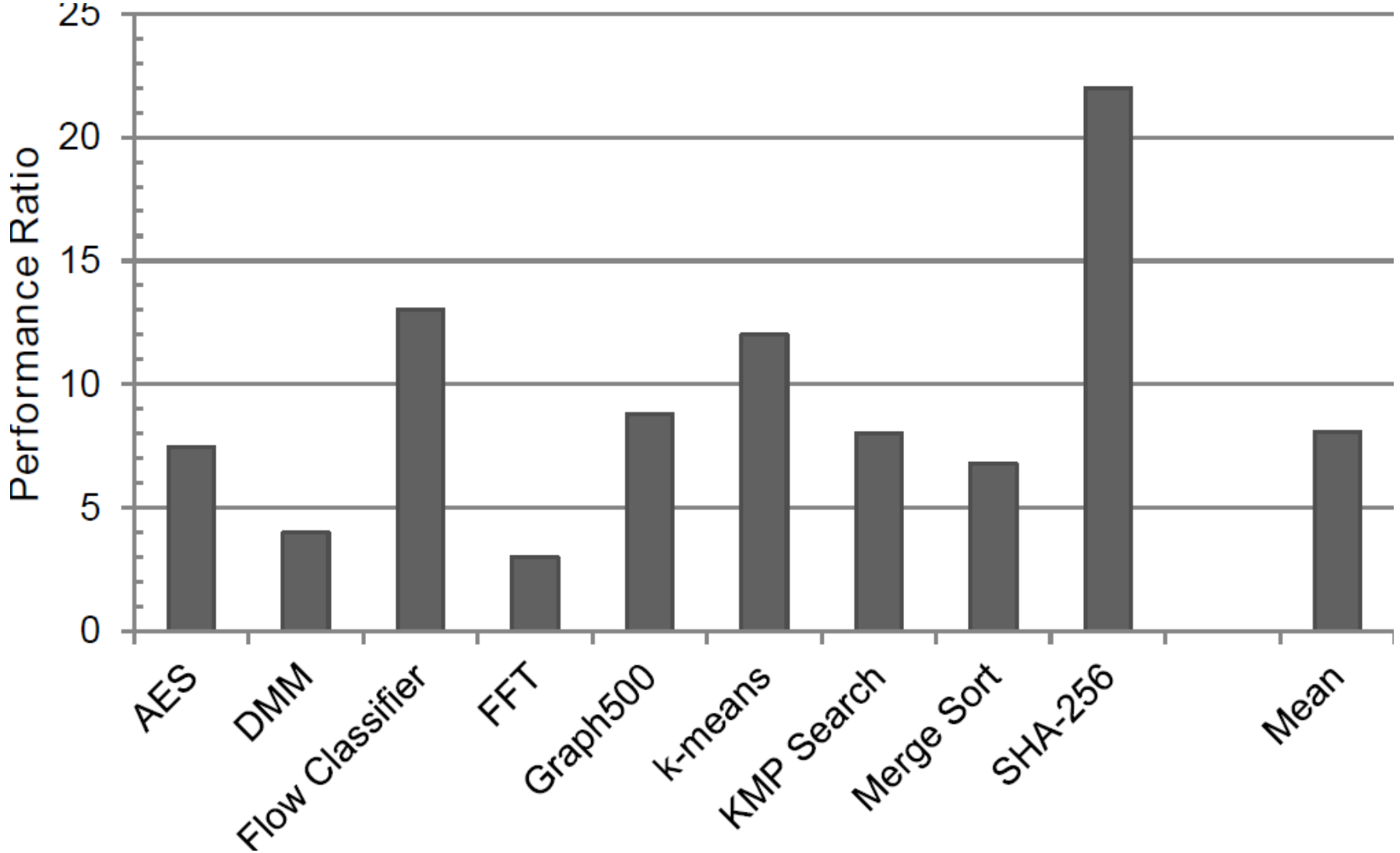
Dyser


Fetch
Decode
Execute
Memory
Writeback
Instruction cache
Decode
Execution pipeline
Register file
Vector-port mapping
Wide memory interface
Fast-configuration switching (FCS)
Dyser input interface
Dyser output interface
FU
Dynamic Specialized Execution Resources
Data cache
Dyser Region
Configuration
S Switches
FU Functional unit
Parallelism specialization enhancements
(a)
(b)

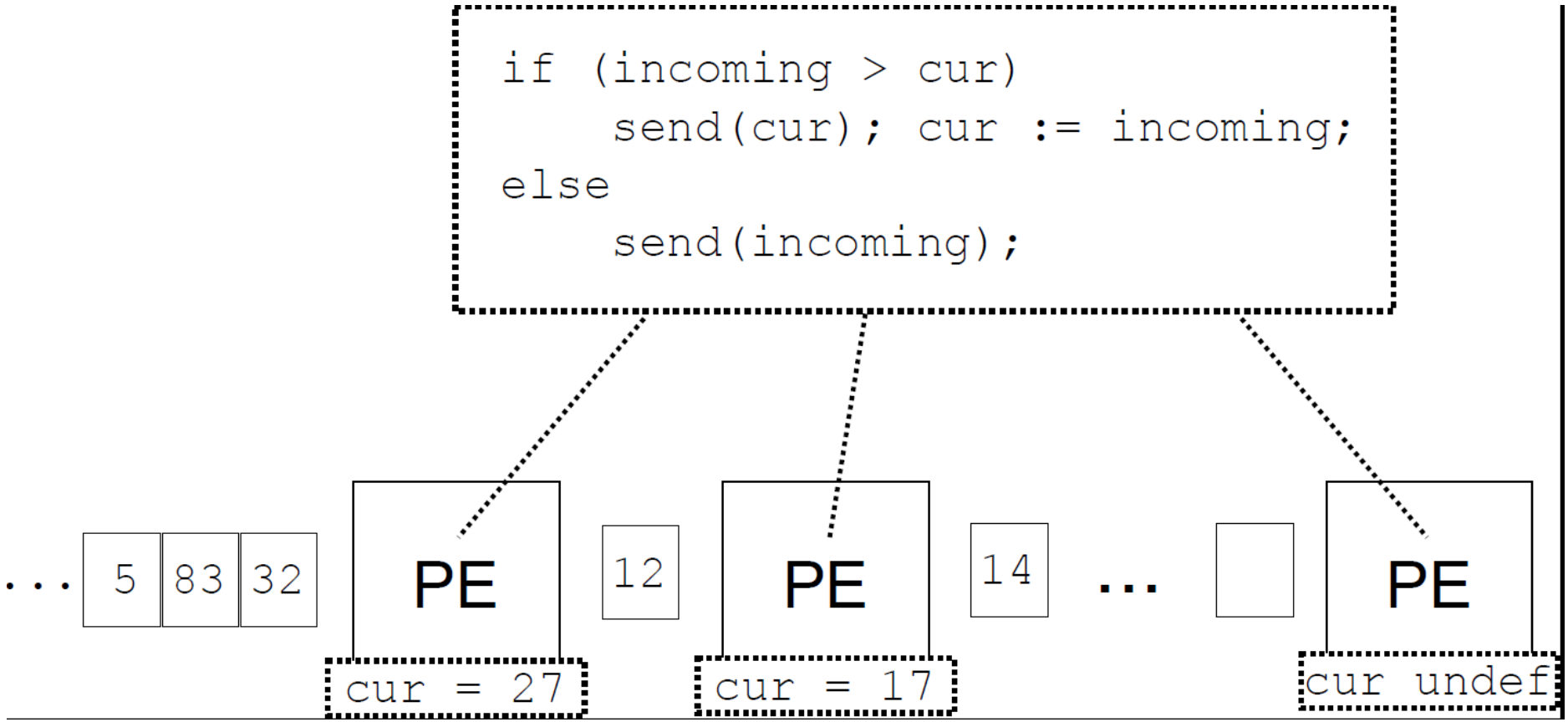
Coarse-grained Block Diagram



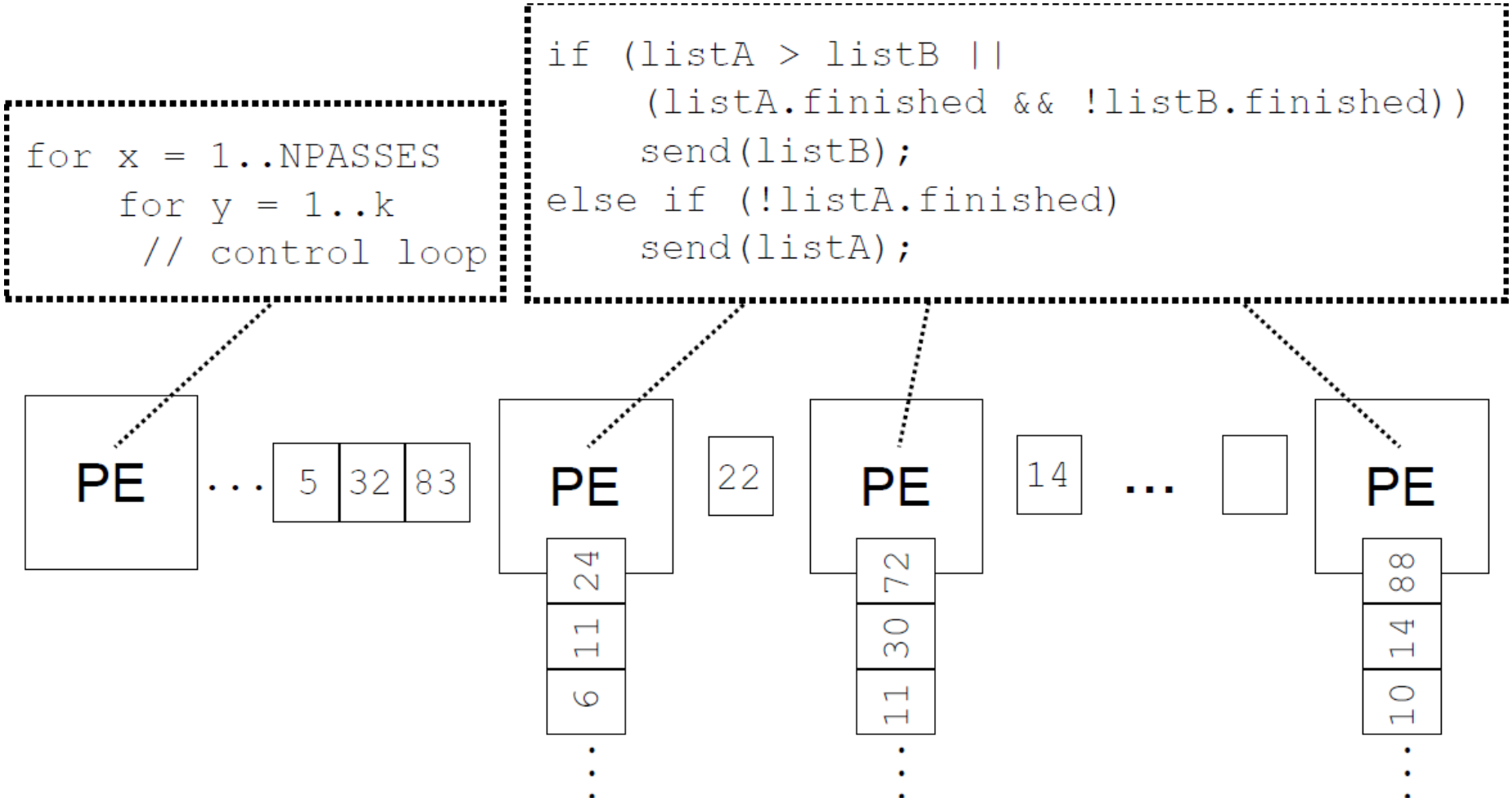
Performance/Area



Sorting Network



Merge Sort


for x = 1..NPASSES
for y = 1..k
// control loop
if (listA > listB ||
(listA.finished && !listB.finished))
send(listB);
else if (!listA.finished)
send(listA);
PE
5
32
83
PE
24
11
6
22
PE
72
30
11
14
PE
88
14
10

Merge sort worker – PC+RegQueue


```
check_a: beqz   %in0.notEmpty, check_a // listA
check_b: beqz   %in1.notEmpty, check_b // listB
check_o: beqz   %out0.notFull, check_o // outList
         beq    %in0.tag, EOL, a_done
         beq    %in1.tag, EOL send_a
         cmp.lt %r0, %in0.first, %in1.first
         bnez   %r0, send_a
send_b:  enq    %out0, %in1.first
         deq    %in1
         jump   check_a
send_a:  enq    %out0, %in0.first
         deq    %in0
         jump   check_a
a_done:  beq    %in1.first, EOL, done
         jump   send_b
done:    deq    %in0
         deq    %in1
         return;
```

Static Insts	18
Avg Insts/Iteration	10
Avg Branches/Iteration	7

PC-based design alternatives

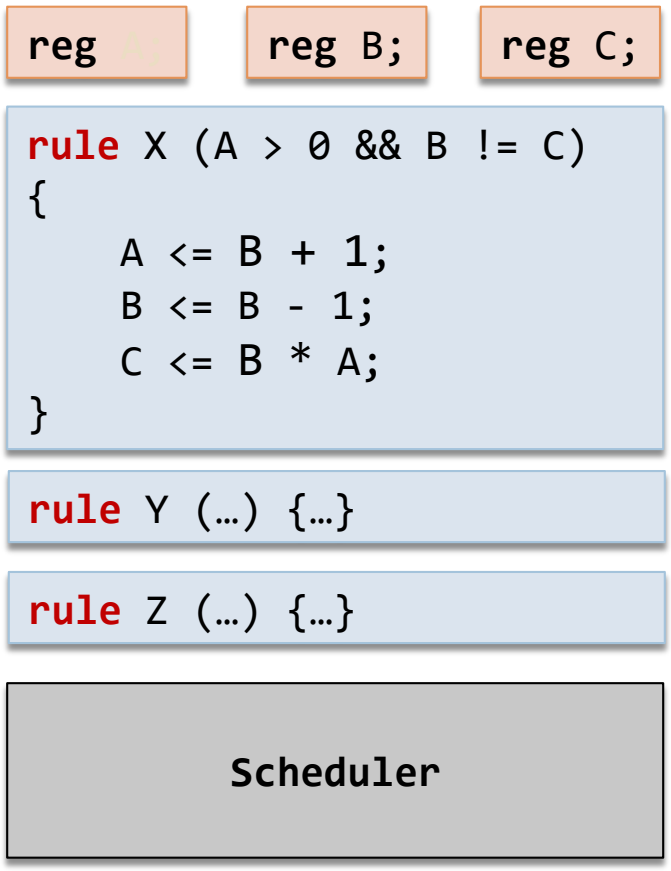
Feature	Description
PC (Baseline)	PEs use program counters, communicate using shared-memory queues.
+RegQueue	Expose register-mapped queues to ISA, test via active polling.
+FusedDeq	Destructive read of queue registers without separate instructions.
+RegQSelect	Allow indirect jump based on register queue status bits.
+RegQStall	Issue stalls on queue input/output registers without special instructions.
+QMultiThread	Stalling on empty/full queue yields thread.
+Predication	Predicate registers that can be set using queue status bits.
+Augmented	ISA augmented with all of the above features except +QMultiThread.

Merge sort worker – PC+Augmented


```
start:          beq     %in0.tag, EOL, a_done
                beq     %in1.tag, EOL, send_a
                cmp.ge  p2, in0.first, in1.first
send_b:   (p2)  enq     %out0, %in1.first (deq %in1)
send_a:  (!p2)  enq     %out0, in0.first (deq %in0)
                jump    start
a_done:         cmp.ne  p2, %in1.first, EOL
          (p2)  jump    send_b
                nop     (deq %in0, deq %in1)
                return;
```

Static Insts	9
Avg Insts/Iteration (Issued)	6
Avg Insts/Iteration (Committed)	5
Avg Branches/Iteration	3
Speedup vs PC+RegQueue (Fig 3)	1.4×

Guarded Actions



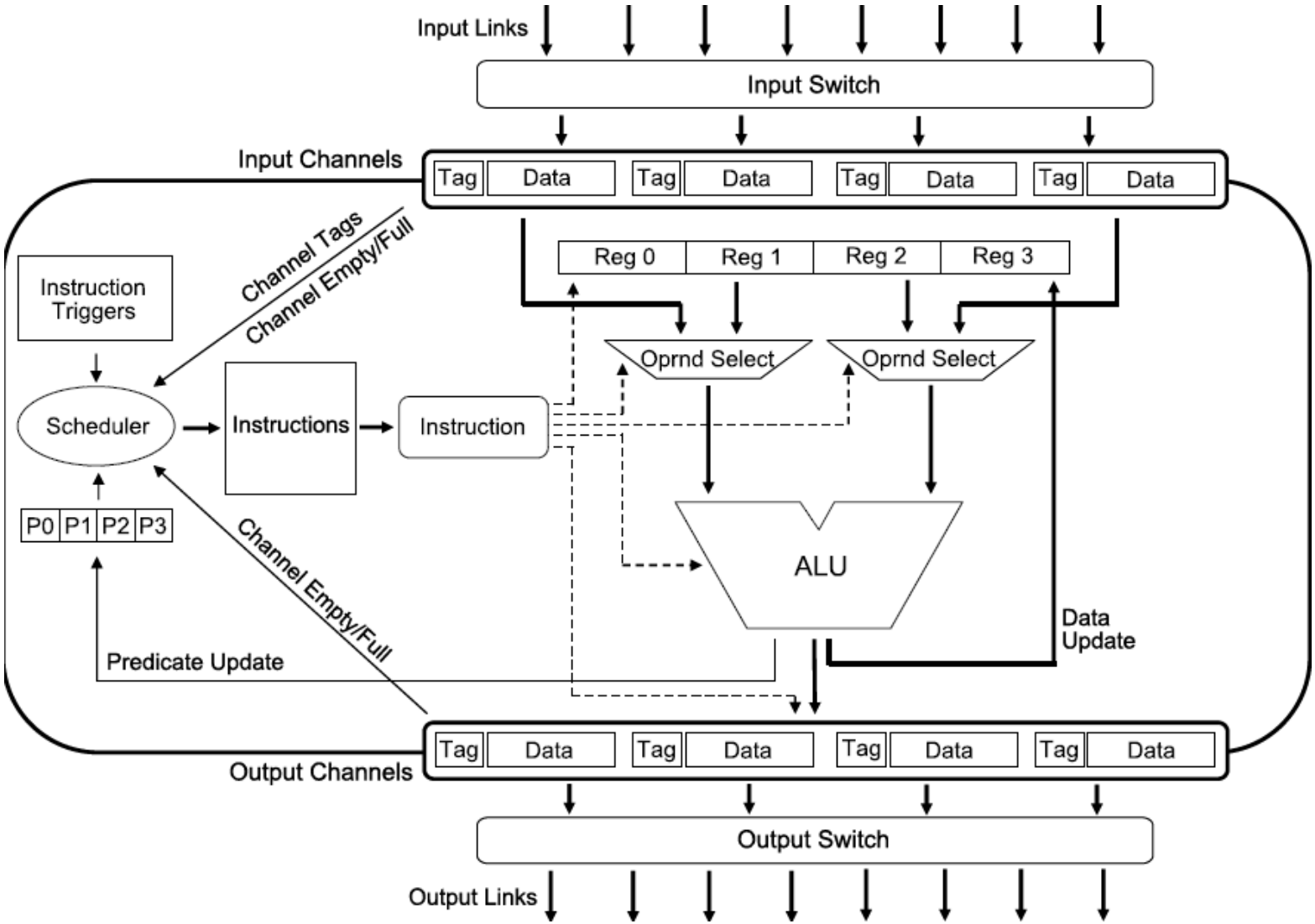
- Program consists of **rules** that may perform computations and read/write state
- Each rule specifies conditions (**guard**) under which it is allowed to fire
- Separates description and execution of data (rule body) from control (guards)
- A **scheduler** is generated (or provided by hardware) that evaluates the guards and schedules rule execution
- Sources of Parallelism
  - Intra-Rule parallelism
  - Inter-Rule parallelism
  - Scheduler overlap with Rule execution
  - Parallel access to state

Merge sort worker – Trigger Inst

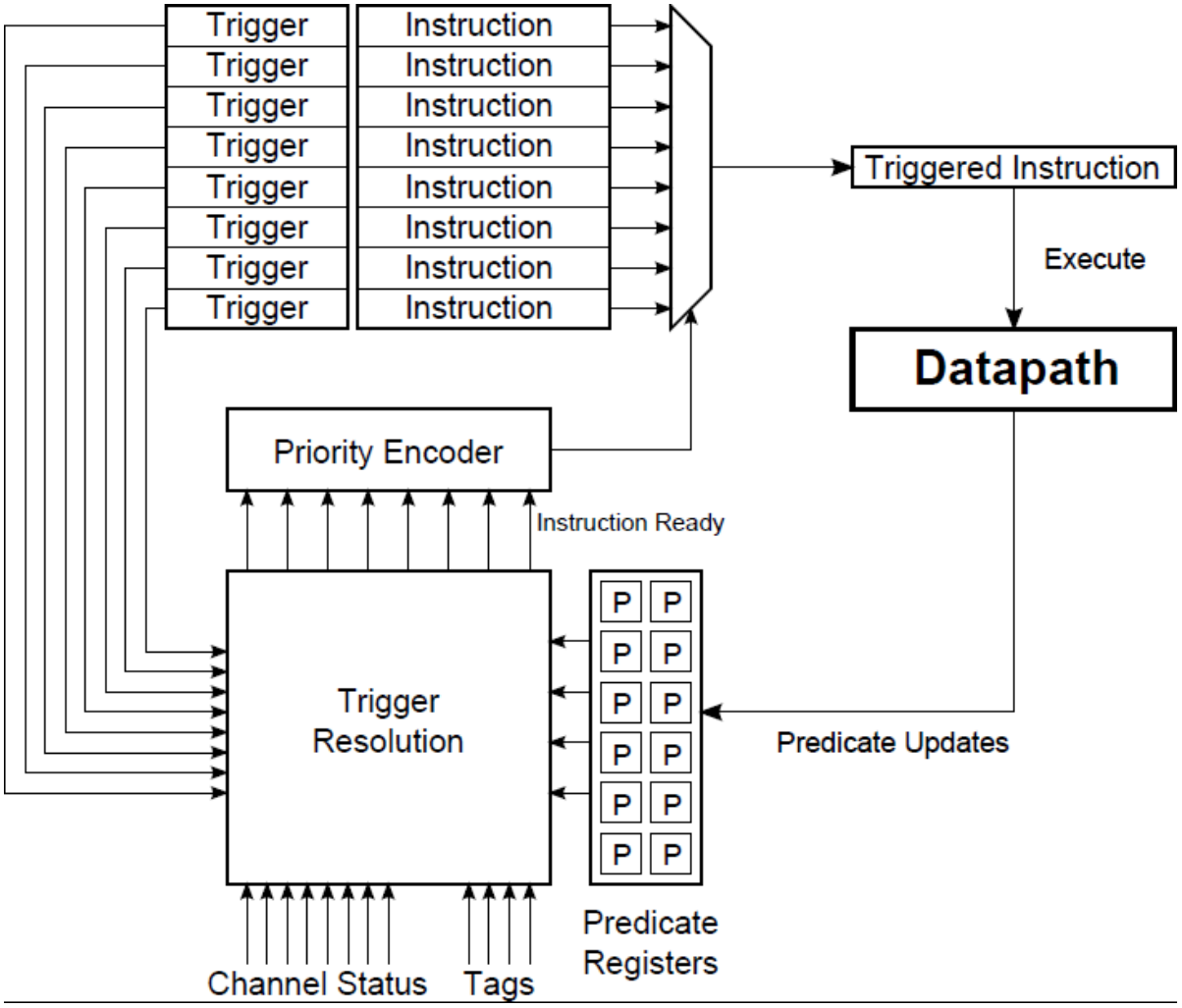

```
doCheck:
    when (!p0 && %in0.tag != EOL
              && %in1.tag != EOL) do
        cmp.ge p1, %in0.data, %in1.data (p0 := 1)
sendA:
    when (p0 && p1) do
        enq %out0, %in0.data (deq %in0, p0 := 0)
sendB:
    when (p0 && !p1) do
        enq %out0, %in1.data (deq %in1, p0 := 0)
drainA:
    when (%in0.tag != EOL && %in1.tag == EOL) do
        enq %out0, %in0.data (deq %in0)
drainB:
    when (%in0.tag == EOL && %in1.tag != EOL) do
        enq %out0, %in1.data (deq %in1)
bothDone:
    when (%in0.tag == EOL && %in1.tag == EOL) do
        nop (deq %in0, deq %in1)
```

Static Insts	6
Avg Insts/Iteration	2
Speedup vs PC+RegQueue (Fig 3)	5×
Speedup vs PC+Augmented (Fig 4)	3×

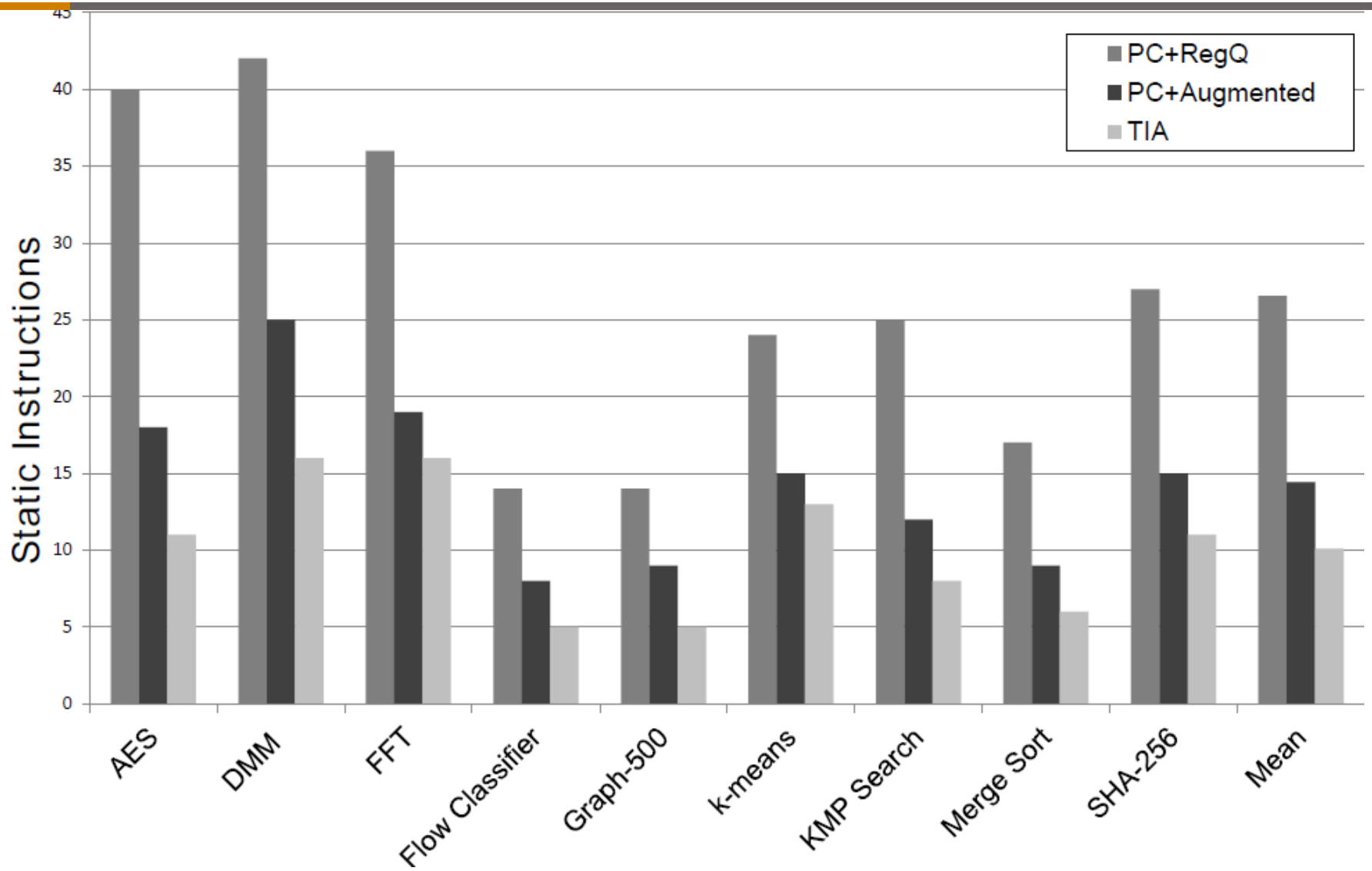
PE Block Diagram



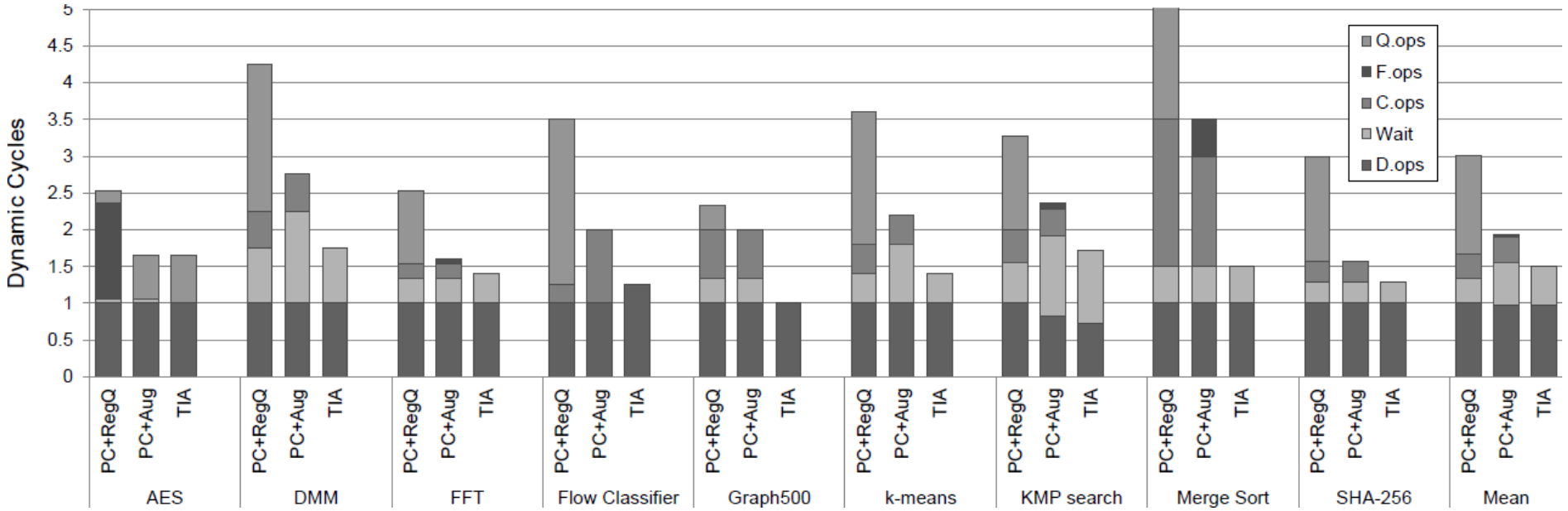
Scheduler


Trigger
Instruction
Triggered Instruction
Execute
Datapath
Priority Encoder
Instruction Ready
Trigger Resolution
P
Predicate Updates
Predicate Registers
Channel Status
Tags

Static Instructions



Dynamic Instructions



Control Instructions Evaluation