SIMD Instruction Set Extensions for Keccak with Applications to SHA-3, Keyak and Ketje

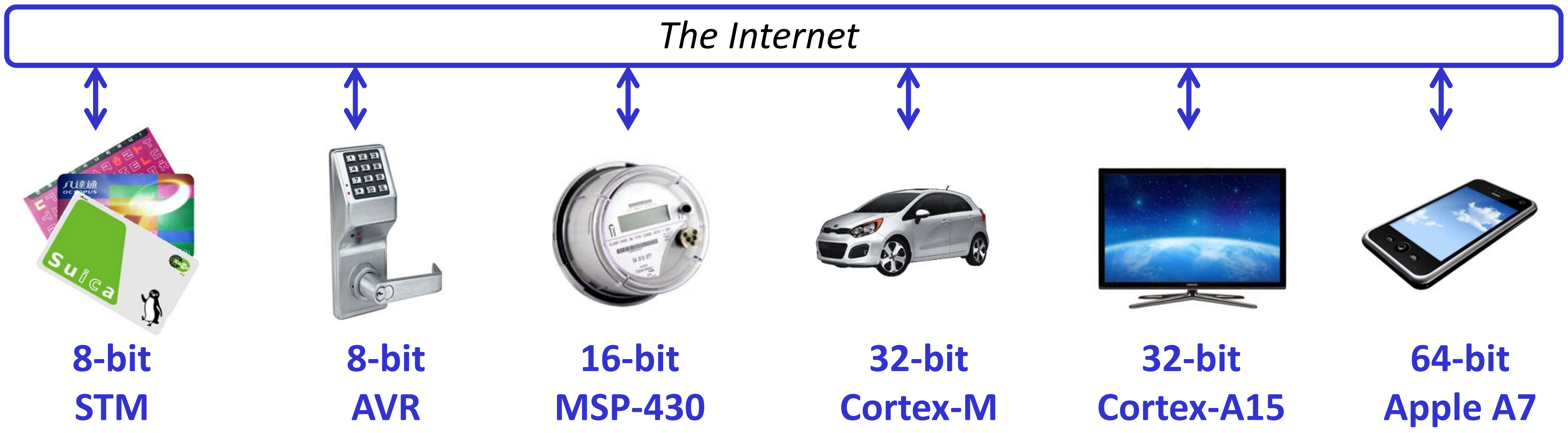
Patrick Schaumont
Hemendra Rawat

Bradley Department of ECE
Virginia Tech

Outline

- About the Things in the Internet (of Things)
  - Making a case for custom instructions
- Keccak in custom instructions
  - Application stack and layering
  - Design principles
  - Instruction set
- Performance analysis and portability
- Conclusions

About the Things



Things (in the Internet of Things) are **extremely heterogeneous**

- Computing capabilities
- Communication capabilities
- Storage capabilities
- Energy/Power constraints

Efficient Cryptography

8-bit
STM

8-bit
AVR

16-bit
MSP-430

32-bit
Cortex-M

32-bit
Cortex-A15

64-bit
Apple A7

IoT architectures must support **flexible crypto**

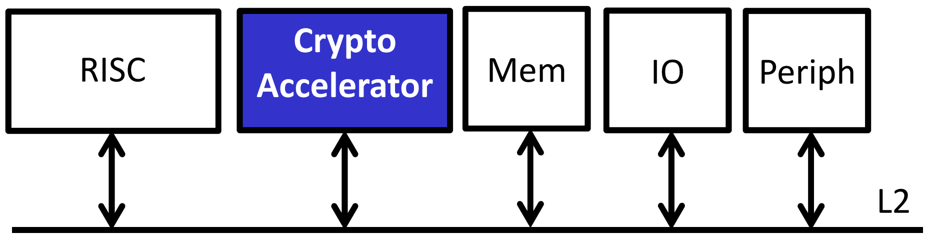
- Multiple kernels (RNG, MAC, AEAD, ..)
- Multiple modes of operation per kernel
- Multiple security levels

with **high efficiency**

- Max performance (min cycles/byte)
- Min energy/power (J/byte, W/(byte/s))

Codesign of Crypto and Computing

8-bit STM	8-bit AVR	16-bit MSP-430	32-bit Cortex-M	32-bit Cortex-A15	64-bit Apple A7



Low-complexity systems:

- Single core
- 1-2 bus levels
- Single master
- Bare metal SW

A ***Memory-mapped Coprocessor*** accelerates crypto near the processor (near the information processing of bits)

Codesign of Crypto and Computing

8-bit
STM

8-bit
AVR

16-bit
MSP-430

32-bit
Cortex-M

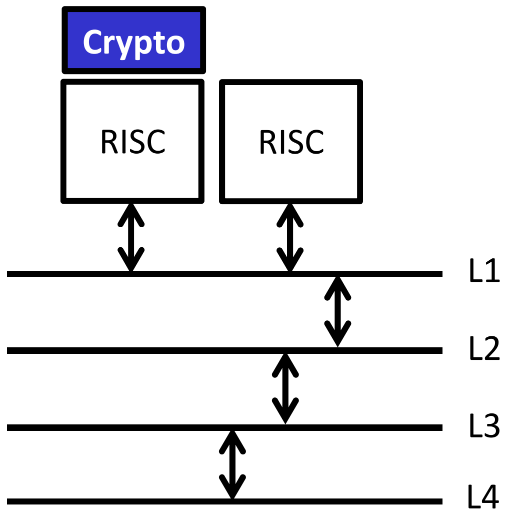
32-bit
Cortex-A15

64-bit
Apple A7

High-complexity systems:

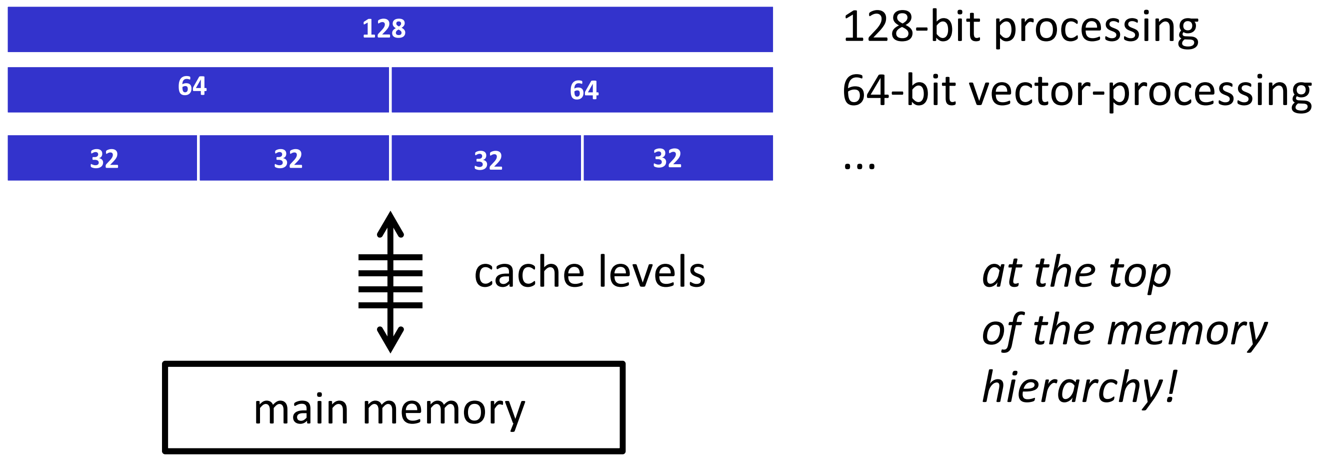
- Multi-core
- Multiple bus levels (>=4)
- Multi-master
- OS, multi-task SW

A ***Custom Instruction*** accelerates crypto near the processor (near the information processing of bits)



Crypto Instruction Sets

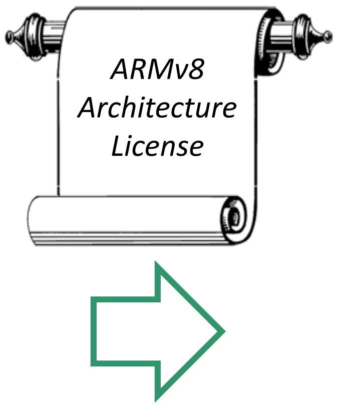
- **Current generation (Westmere+, ARM v8)**
  - **AES**
  - **SHA-1, SHA-256**
  - **Carrier-less multiplication**
- **SIMD Units**
  - **Intel AVX (128-bit), AVX2 (512-bit)**
  - **ARM NEON (128-bit)**



at the top
of the memory
hierarchy!

New Crypto Instruction Sets?

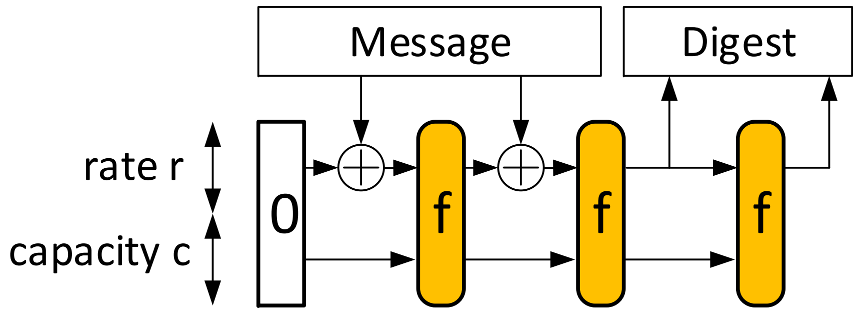
- **SoC Cores are licensed, customizable**



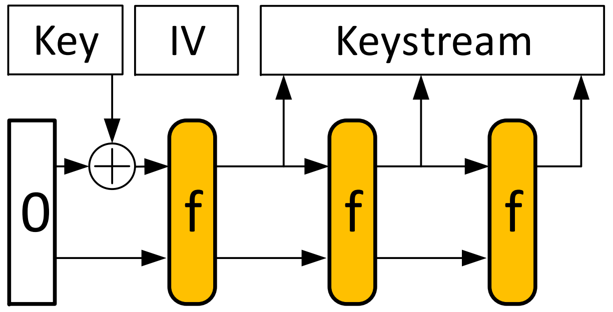
Company	System-on-Chip
Apple	A-series
Qualcomm	Snapdragon-series
Samsung	Exynos-series

- **New, specialized instructions need no compiler**
  - **Have limited software infrastructure needs**
  - **Can be supported through specialized libraries**

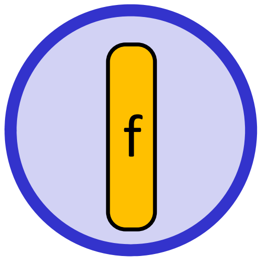
Crypto with the Keccak Sponge & Duplex



Hashing

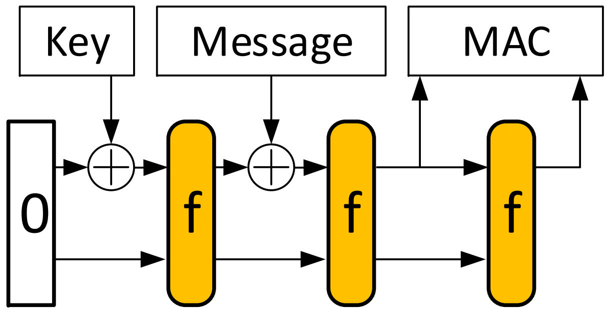


Keystream Generation

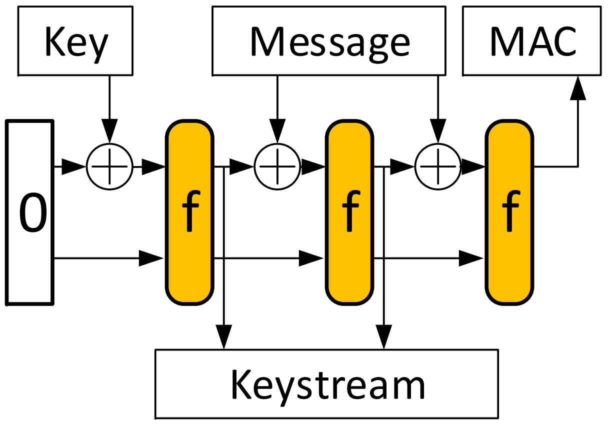


Keccak-f/p permutation

MACing

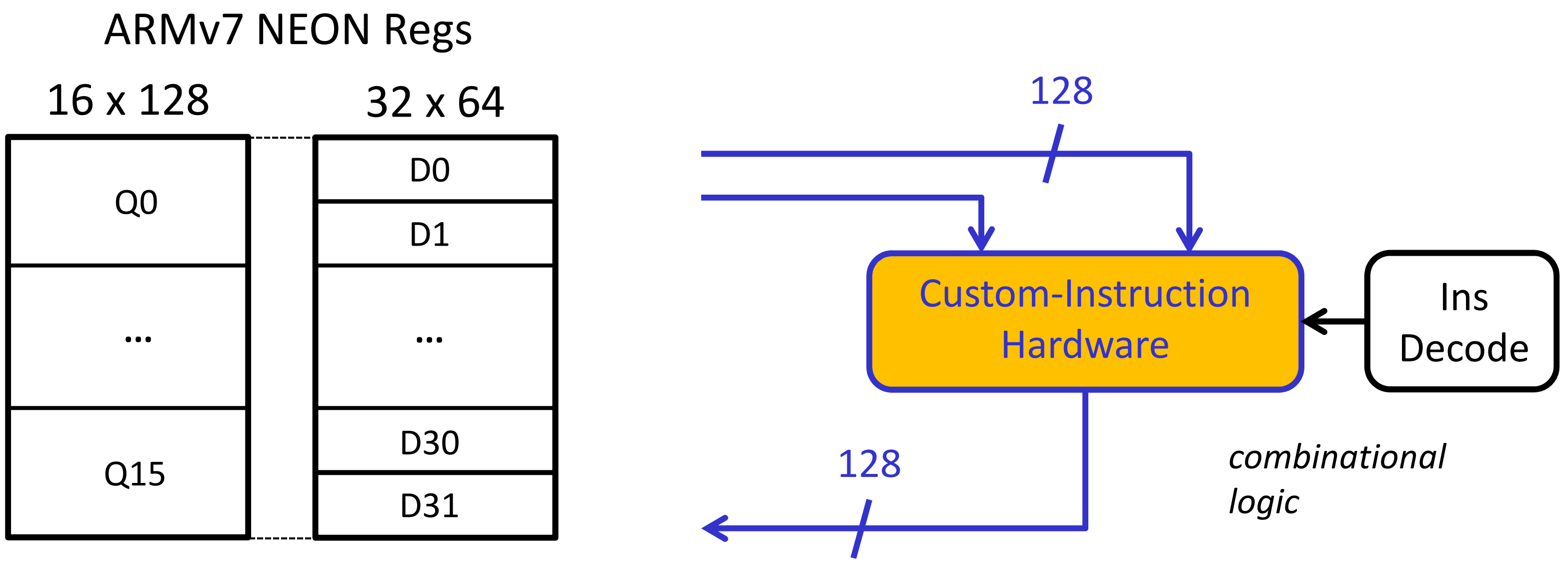


Authenticated Encryption

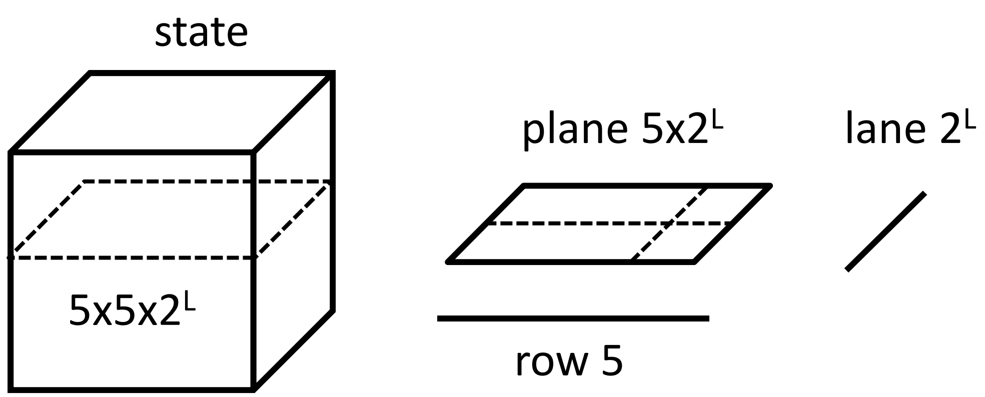


Design Objective

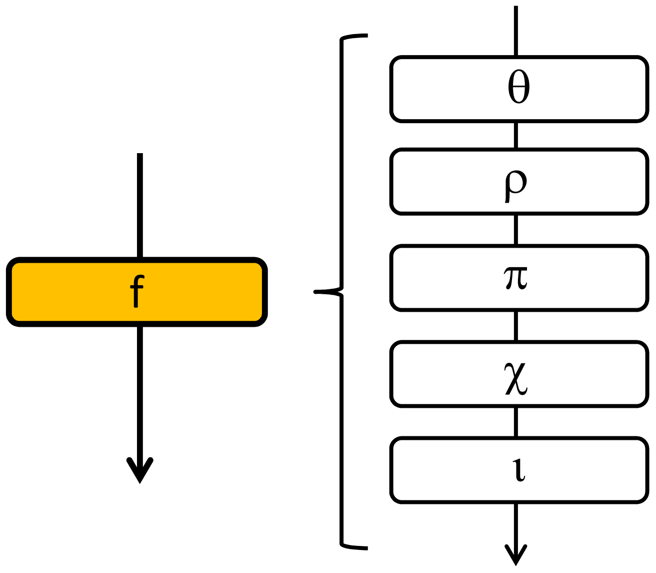
- **Custom-instruction Design for**
  - **Keccak-p[1600,800,400,200]**
  - **ARM v7 128-bit ARM NEON SIMD interface**
- **Performance against C, ASM, NEON-ASM**



The Keccak State



	2^L
Keccak-f[1600]	64
Keccak-f[800]	32
Keccak-f[400]	16
Keccak-f[200]	8



Each round has 5 steps
Each step is a n-lane to lane transform

- n = {5,3,2,1}
- rotation, bitwise logic

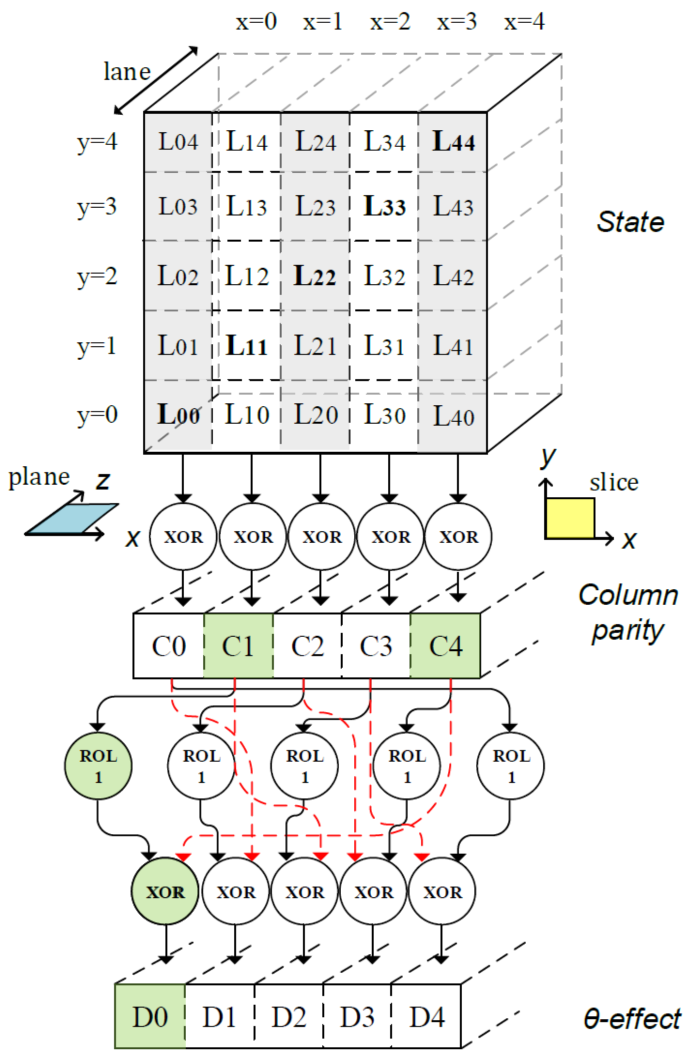
Custom-Instruction Design Principles

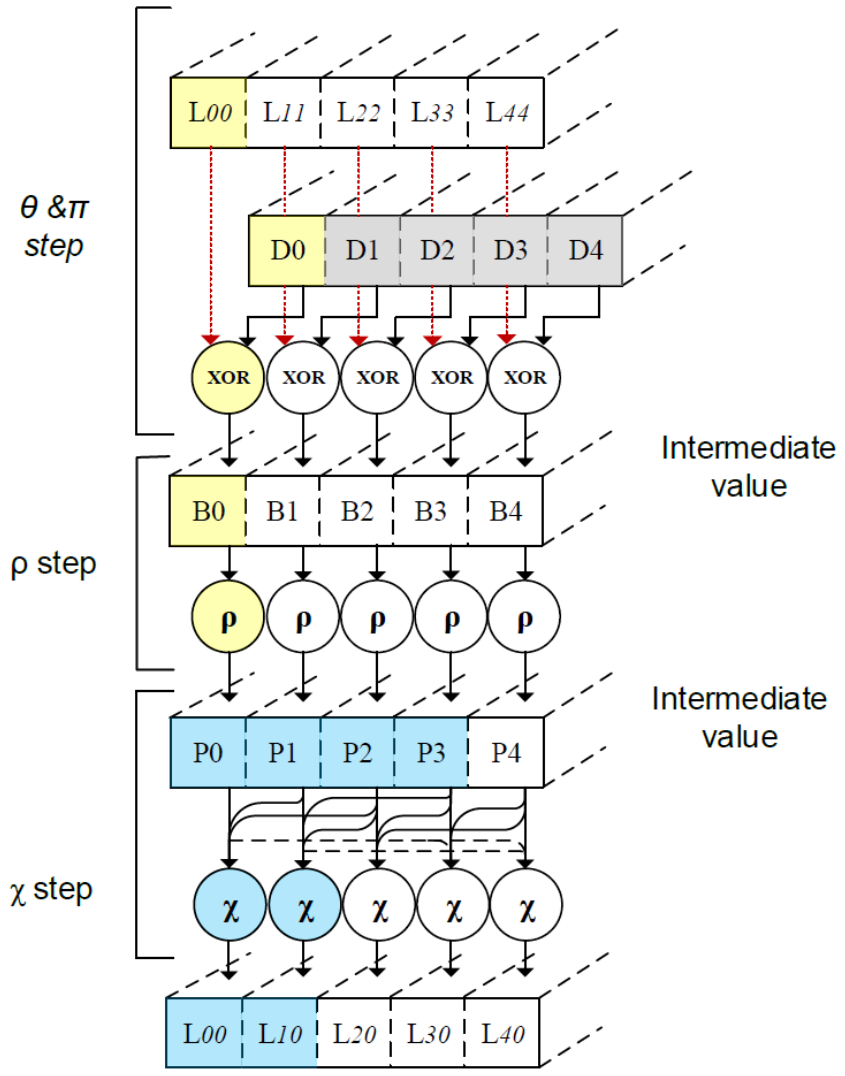
- **Partition Keccak round DFG into custom operations**

  (128-bit, 128-bit) -> 128-bit
  or (64-bit, 64-bit) -> 64-bit

- **Optimize custom operations**
  - **Keep custom instructions simple**
    = Purely combinational, low logic depth
  - **Minimal schedule length for Keccak round**
    = Minimal number of instructions per round
  - **Minimize register pressure**
    = Use in-place operations
  - **Minimize register reordering (MOV/VEXT)**
    = Manual NEON register allocation

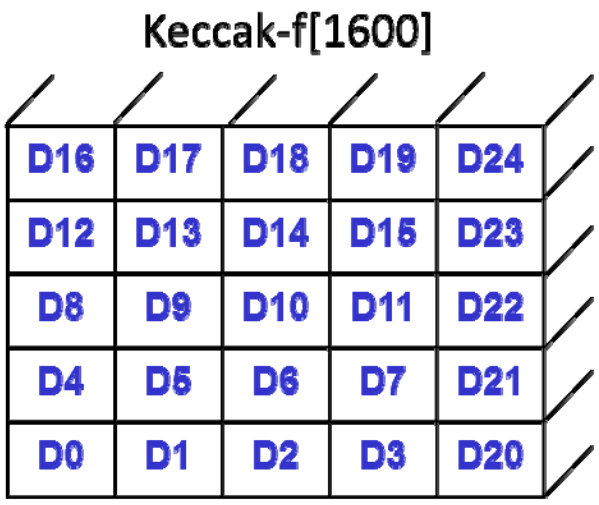
The Keccak Round DFG

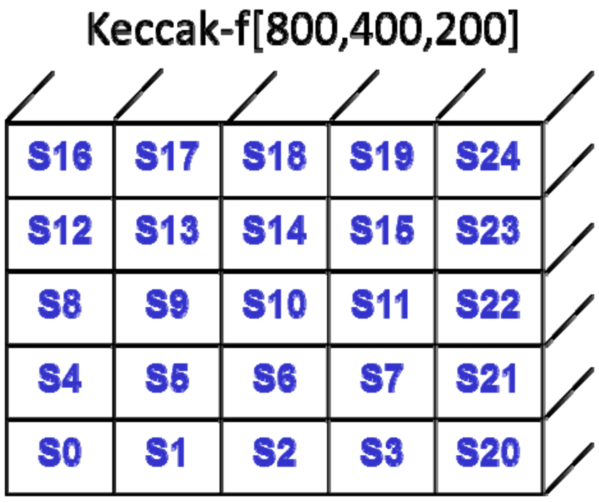




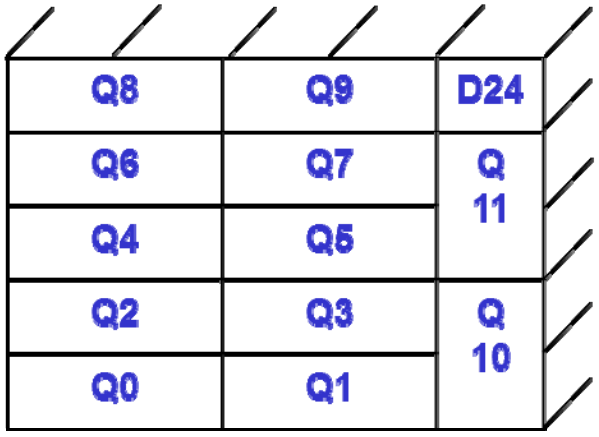
(ι step not shown)

Keccak NEON State Mapping



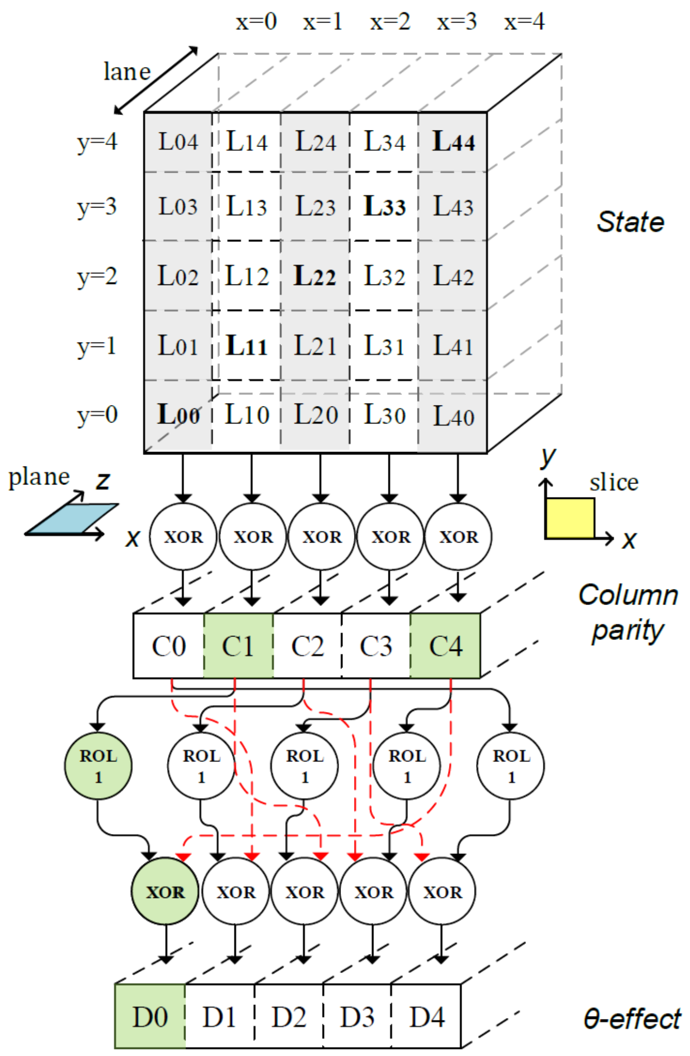


Leaves 7 working registers (D25-D31) to implement Keccak round



Vector operations can work on two lanes at the same time

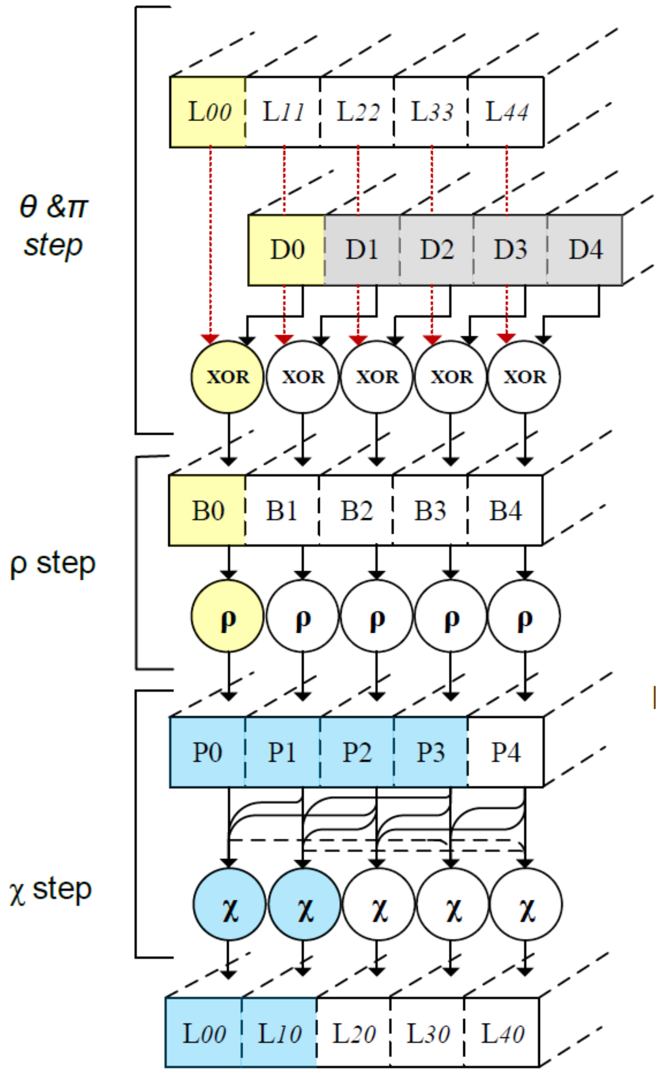
Example 1: r1lx (rotate-left-1 and xor)



```
veor.64     q13, q0,  q2
veor.64     q14, q1,  q3
veor.64     q15, q10, q11
veor.64     q13, q4,  q13
veor.64     q14, q5,  q14
veor.64     d30, d30, d31
veor.64     q13, q6,  q13
veor.64     q14, q7,  q14
veor.64     q13, q8,  q13
veor.64     q14, q9,  q14
veor.64     d30, d24, d30

rl1x.u64    d25, d26, d28
rl1x.u64    d31, d29, d26
rl1x.u64    d26, d28, d30
rl1x.u64    d28, d27, d29
rl1x.u64    d29, d30, d27
```

Example 2: kxorr64 (Keccak Xor Rot)



```
kxorr64   d0,  d0,  d29, #0
kxorr64   d2,  d10, d28, #12
kxorr64   d3,  d15, d26, #18
kxorr64   d10, d11, d26, #13
kxorr64   d15, d14, d28, #17
kxorr64   d11, d23, d31, #19
kxorr64   d14, d9,  d25, #11
kxorr64   d23, d19, d26, #23
kxorr64   d9,  d6,  d28, #7
kxorr64   d19, d12, d29, #15
kxorr64   d6,  d8,  d29, #10
kxorr64   d12, d20, d31, #4
kxorr64   d8,  d1,  d25, #1
...
```

target π output	source θ input	θ effect	ρ rotation table index

Keccak in 6 Custom Instructions

Instruction	Step	Description	Target Primitive	Syntax

`rl1x`	θ	rotate left 1 & xor	1600
			800
			400
			200
				`rl1x.u64  d2, d0, d1`
				`rl1x.u32  s2, s0, s1`
				`rl1x.u16  s2, s0, s1`
				`rl1x.u8   s2, s0, s1`

`kxorr64`	θ, ρ, π	xor & rotate imm	1600	`kxorr64   d2, d0, d1, #i`

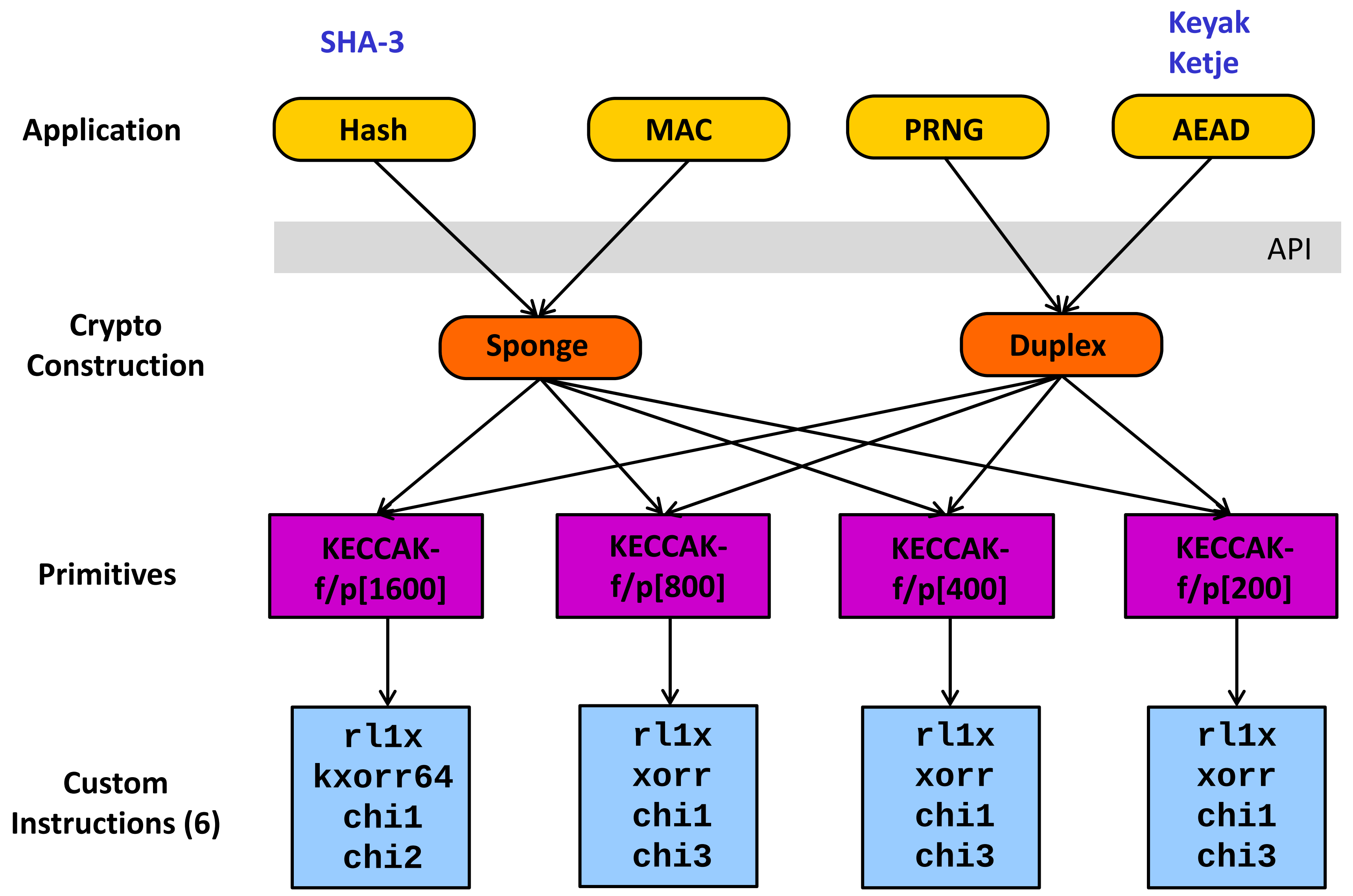
`xorr`	θ, ρ, π	xor & rotate imm	800
			400
			200
				`kxorr.u32 s2, s0, s1, #i`
				`kxorr.u16 s2, s0, s1, #i`
				`kxorr.u8  s2, s0, s1, #i`

`chi1`	χ	chi	1600
			800,400,200
				`chi1.u32  q2, q0, q1`
				`chi1.u64  q2, q0, q1`

`chi2`	χ	chi (lane 5)	1600	`chi2.u64  d4, q0, q1`
`chi3`	χ	chi (lane 5)	800,400,200	`chi3.u32  s4, d0, d1`

Hardware cost: 18,624 GE in 90nm

Keccak Application Stack



Performance Evaluation Methodology

- Reference KeccakCodePackage
  - C, 32-bit ASM, NEON
- Target platform GEM5
  - Native execution of ARMv7 binaries
  - Added Keccak instructions in ISA model
- Modified cross-GCC compiler
  - Support for additional instructions (assembly)
- Simulation
  - 'TimingSimpleCPU' (single-core non-OOO)
  - 32K L1, 2M L2 Cache memory model
  - Performance measured in 'Instructions/Byte'

Performance Results

Instructions per Round

Primitive	C	32b ASM	NEON	CI
Keccak-f[1600]	713	414	145	66
Keccak-f[800]	271	194	NA	56
Keccak-f[400]	370	217	NA	55
Keccak-f[200]	361	168	NA	57

Instructions per Byte

Application	C	32b ASM	NEON	CI	Speedup
SHA-3	243.5	143.9	**48.1**	21.9	**2.2**
Lake Keyak (E)	61.0	NA	**13.4**	7.7	**1.7**
River Keyak (E)	55.2	**39.3**	NA	14.8	**2.6**
Ketje SR (E)	166.1	**87.9**	NA	55	**1.6**
Ketje JR (E)	309.1	**146.6**	NA	106.5	**1.4**

Conclusions

- **Keccak NEON custom instructions deliver 2.2x performance improvement over hand-optimized designs**
- **Compatible with existing SoC IP customization flow**
- **Complete suite of symmetric crypto using Keccak-f/p**

- **Open Issues**
  - **Impact of parallel modes of operation**
  - **Impact of wider processing, eg. AVX2, AVX-512**