Verification Accelerates

Speeding up AWS's ML-KEM and ML-DSA via Optimized and Formally Verified SHA-3

Mila Anastasova
Applied Scientist
AWS

John Harrison
Senior Principal Applied Scientist
AWS

Yan Peng
Applied Scientist
AWS

Post-Quantum Crypto & Formal Verification (Why?)

AWS Cloud Services

PQ in AWS-LC
- ML-KEM
- ML-DSA
    - SHA-3

Fast & Formal in AWS-LC (How?)

Native x86-64:
- Scalar and Avx2 x86-64 code

HOL Light in AWS-LC:
- x86-64 Formal Proofs via s2n-bignum

AWS-LC Performance Results (so What?)

Real-World Performance Graphs

PQ Crypto & Formal Verification in AWS-LC (Why?)

What is AWS-LC?

AWS's general purpose crypto library

- Portable C/C++ cryptographic primitives
- Optimal target-specific implementations

Secure Connections

Confidentiality

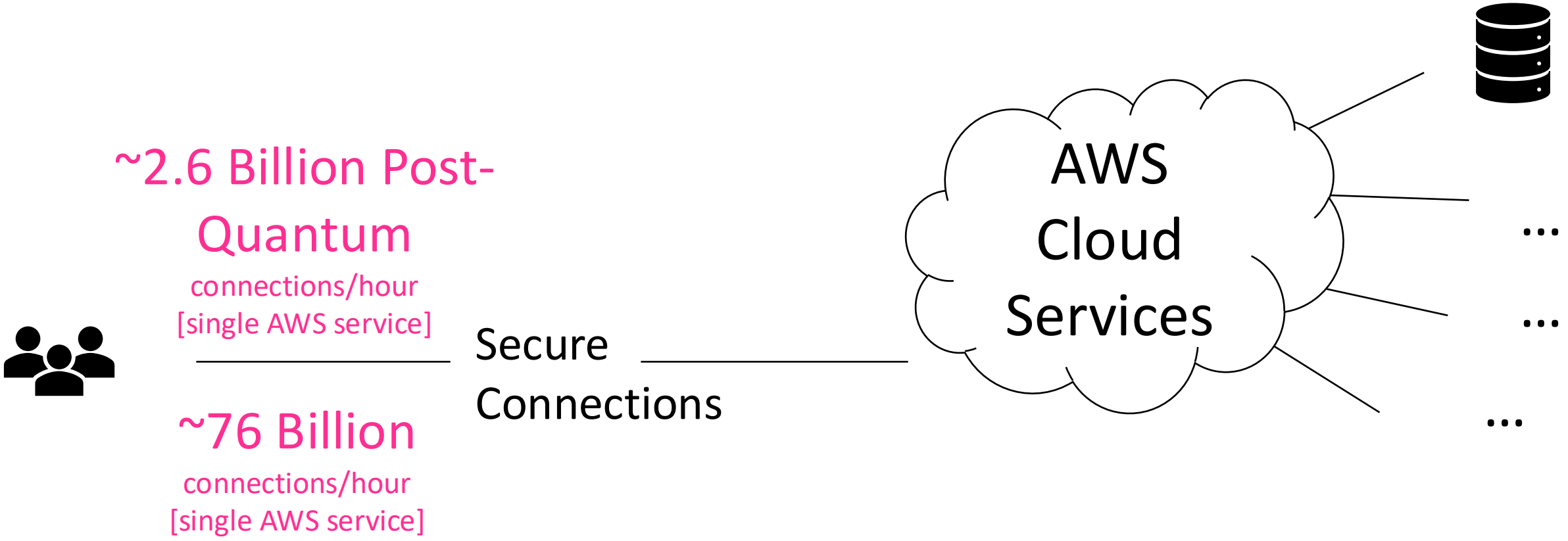
Integrity

Authentication

Non-Repudiation

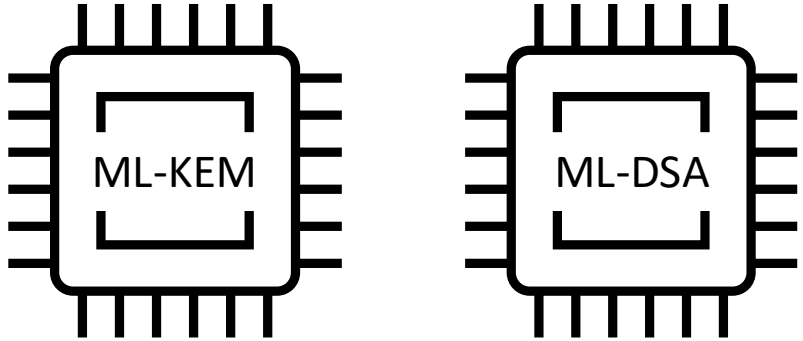
Where is AWS-LC?

- Every optimization saves a lot
  - Assembly Code
- Even a single bug is too expensive
  - Formal Verification



SHA-3 & Post-Quantum Crypto?

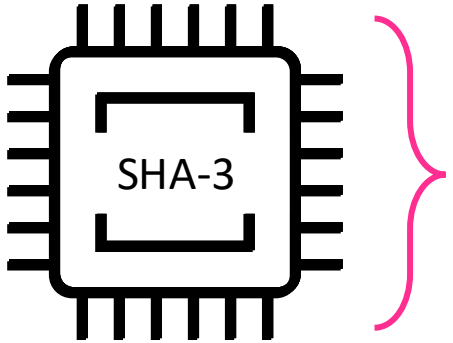
Post-Quantum



Arithmetic In
Polynomial Rings

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$$

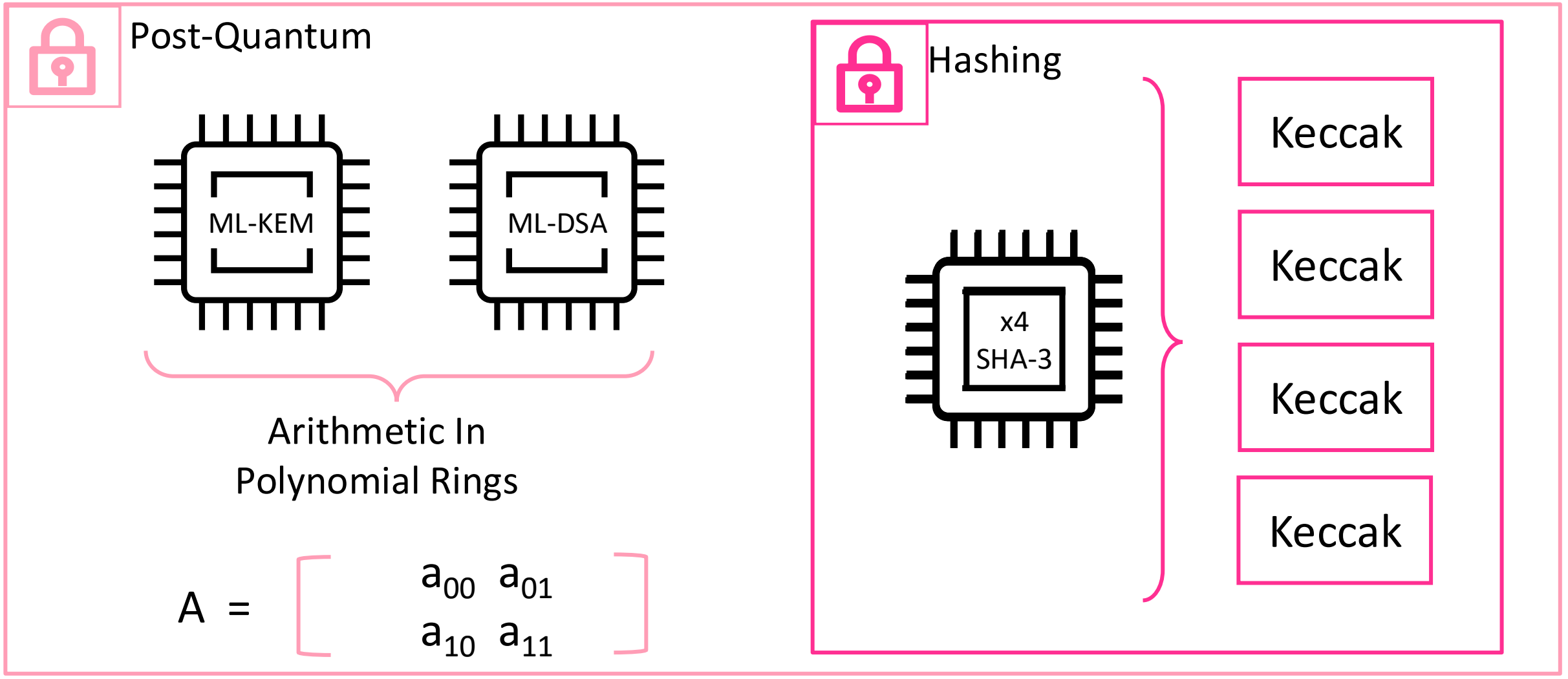
Hashing



Keccak

Fast & Formal in AWS-LC (How?)

Parallel SHA-3 via AVX2



HOL Light & s2n-bignum

What is?

- *HOL Light* is an interactive theorem prover
- *s2n-bignum* is an open-source library offering performant and formally verified cryptographic operations

s2n-bignum?

- Models AArch64 and x86_64 ISAs and µarch to allow for cryptographic operations to be proven:

  Functionally Correct & Memory Safe & Constant-Time

How?

- Checks for consistency between machine code and a specification through Symbolic Simulation

HOL Light Example

```
let SHA3_KECCAK_F1600_CORRECT = prove
  (`!rc_pointer:int64 bitstate_in:int64 A pc:num stackpointer:int64.
   nonoverlapping_modulo (2 EXP 64) (pc, 0x66a) (val  stackpointer, 208) /\
   ==> ensures x86
         (\s. bytes_loaded s (word pc) (BUTLAST sha3_keccak_f1600_tmc) /\
              read RIP s = word (pc + 0x11) /\
              read RSP s = stackpointer /\
              C_ARGUMENTS [bitstate_in; rc_pointer] s /\
              wordlist_from_memory(rc_pointer,24) s = round_constants /\
              wordlist_from_memory(bitstate_in,25) s = A)
             (\s. read RIP s = word(pc + 0x658) /\
                  wordlist_from_memory(bitstate_in,25) s = keccak 24 A)
          (MAYCHANGE [RIP; RAX; RBX; RCX; RDX; RBP;
                      R8; R9; R10; R11; R12; R13; R14; R15; RDI; RSI] ,,
           MAYCHANGE SOME_FLAGS ,,
           MAYCHANGE [memory :> bytes (stackpointer, 208)],,
           MAYCHANGE [memory :> bytes (bitstate_in, 200)])`,
   REWRITE_TAC[SOME_FLAGS] THEN
   MAP_EVERY X_GEN_TAC [`rc_pointer:int64`; `bitstate_in:int64`;`A:int64 list`]
```

Assumptions

- Ensure the separation of memory regions

Preconditions

- Assign the content of a memory region to a symbol A

Postconditions

- Check for memory region content at the end of the program against the specification

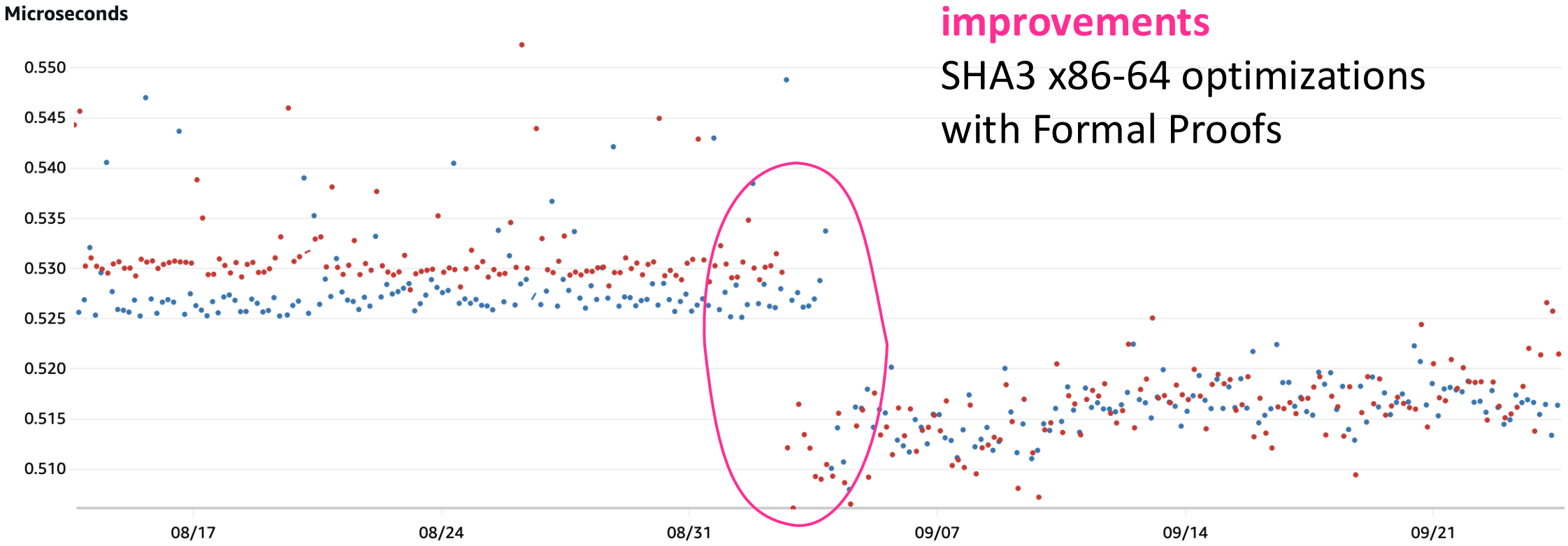
Memory Footprint

Tactics

- Set of functions that transform the goal

AWS-LC Performance Results
(so What?)

SHA3-256



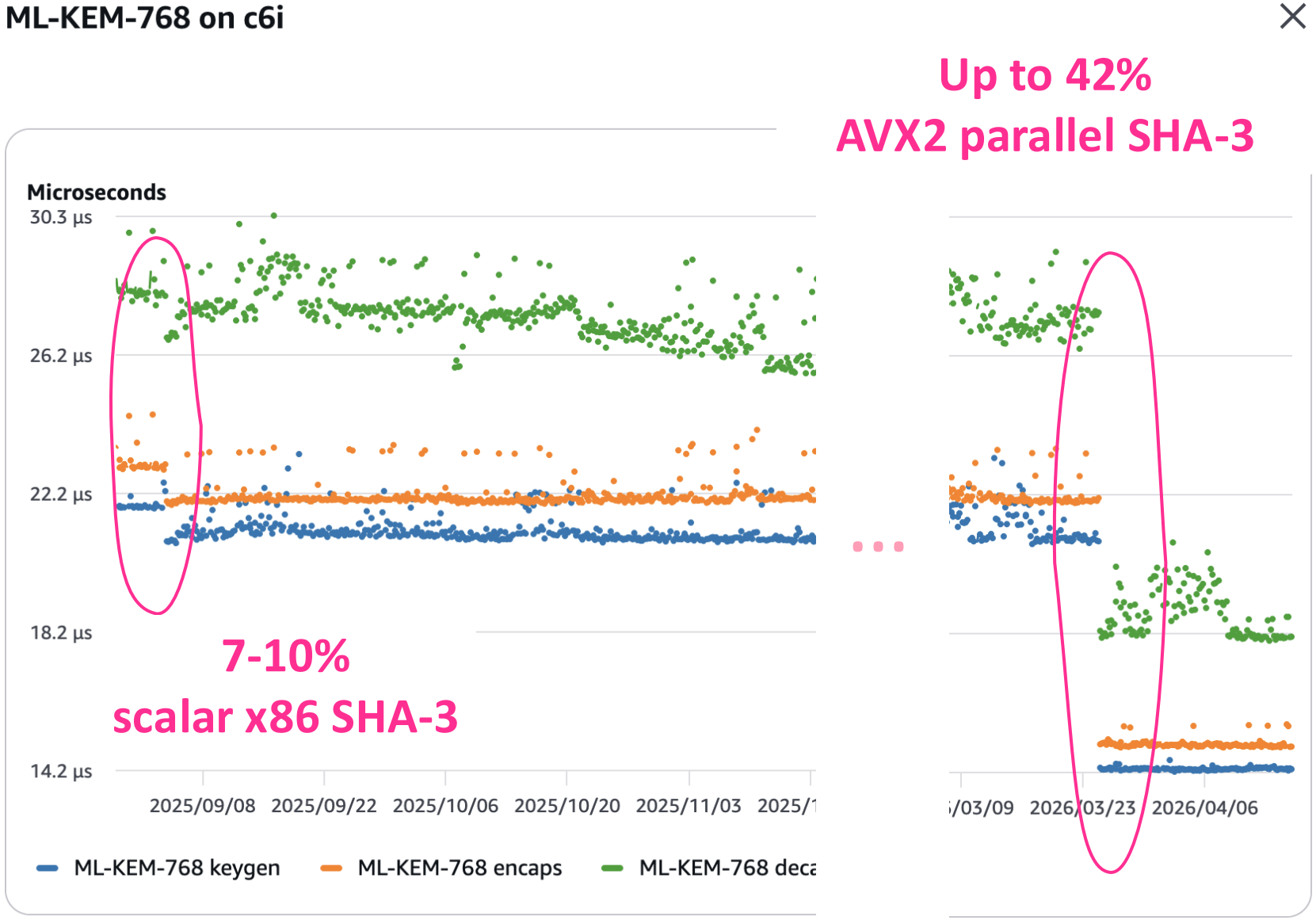
10-15% performance improvements

SHA3 x86-64 optimizations with Formal Proofs

ML-KEM-768 Performance

7-10% for ML-KEM-768
SHA3 x86_64 optimizations with Formal Proofs

Up to 42% for ML-KEM-768
Up to 59% for ML-KEM-1024
SHA3 x86_64 avx2 parallel with Formal Proofs



ML-KEM and ML-DSA Speedup

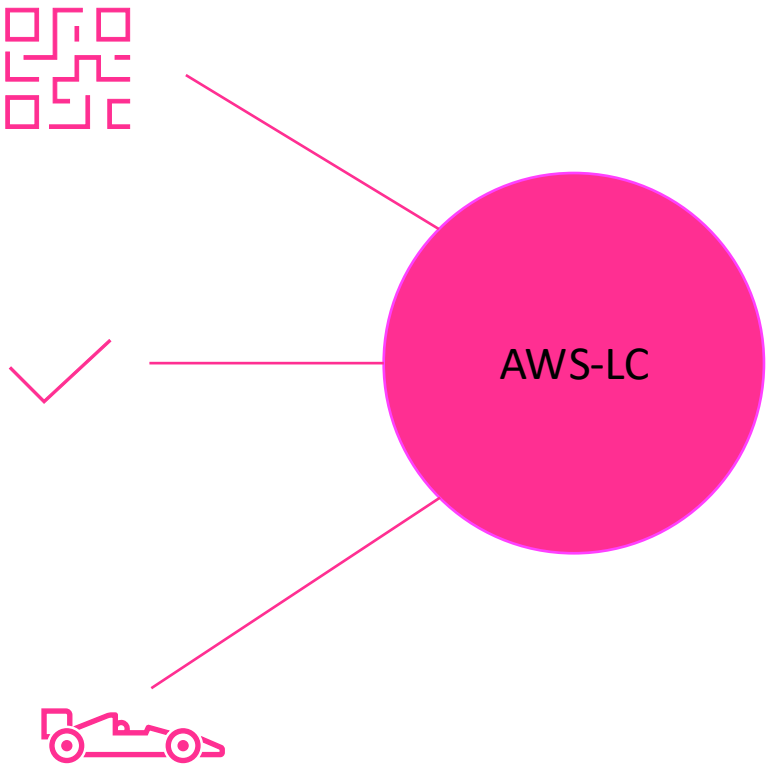
ML-KEM	Op	c7i [%]	c7a [%]	c6i [%]	c6a [%]
512	KG	+35	+45	+49	+46
	Enc	+29	+39	+42	+38
	Dec	+24	+31	+33	+29
768	KG	+32	+42	+42	+43
	Enc	+31	+39	+38	+39
	Dec	+25	+31	+33	+31
1024	KG	+42	+52	+59	+56
	Enc	+35	+48	+52	+50
	Dec	+27	+39	+42	+39

ML-DSA	Op	c7i [%]	c7a [%]	c6i [%]	c6a [%]
44	KG	+21	+28	+33	+31
	Sig	+7	+3	+6	+7
	Ver	+21	+24	+30	+26
65	KG	+24	+26	+30	+28
	Sig	+8	+2	+7	+3
	Ver	+27	+28	+35	+31
87	KG	+38	+41	+49	+44
	Sig	+10	+6	+9	+13
	Ver	+38	+37	+47	+40

Do Formal Proofs allow for Fast AWS-LC?

AWS-LC's large-scale deployment requires aggressive assembly optimizations

Assembly is challenging to develop and review, thus, requires code correctness assurances



Yes, Formal Proofs are the path to Fast & Reliable Code

Thank you!

Mila Anastasova

milaaws@amazon.com