Encrypted Computing

Team 1

施承志、吳靖崴、范勝禹

github link

Outline

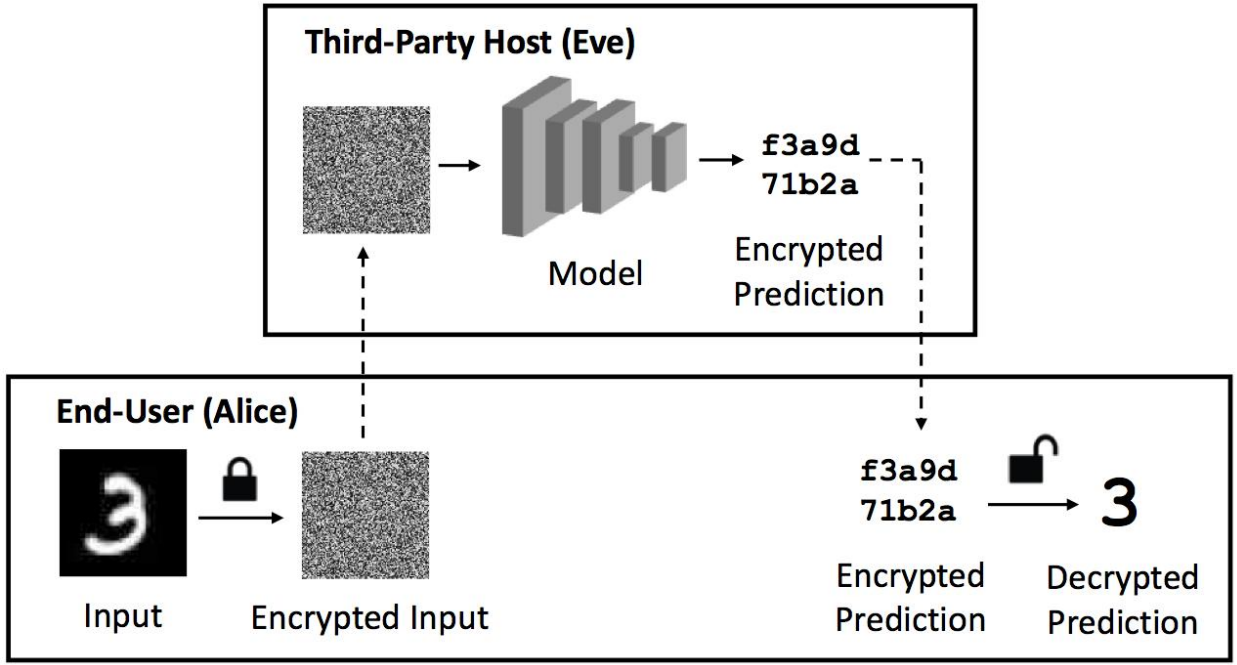
- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

Outline

- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

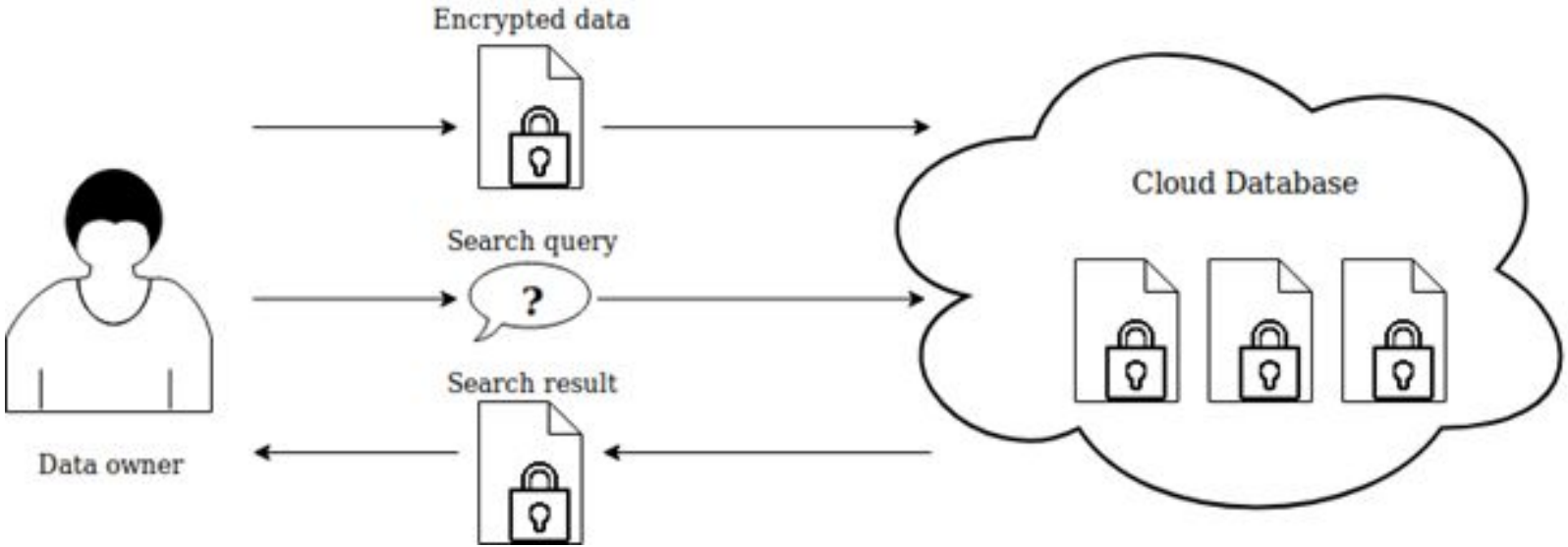
Scenario 1

- Encrypted inference



Scenario 2

- Encrypted database query



Outline

- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

Modular Reduction Is Costly

- In NTT Core, Modular Reduction is Costliest
- Modulo Operator “%” is Costly
- Naive “%” Operator : (x*y)%p →
- Don’t Use Any “%” Operator

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	-	4	8866	6608	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	329	-
Register	-	-	137	-	-
Total	0	4	9003	6937	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	8	13	0

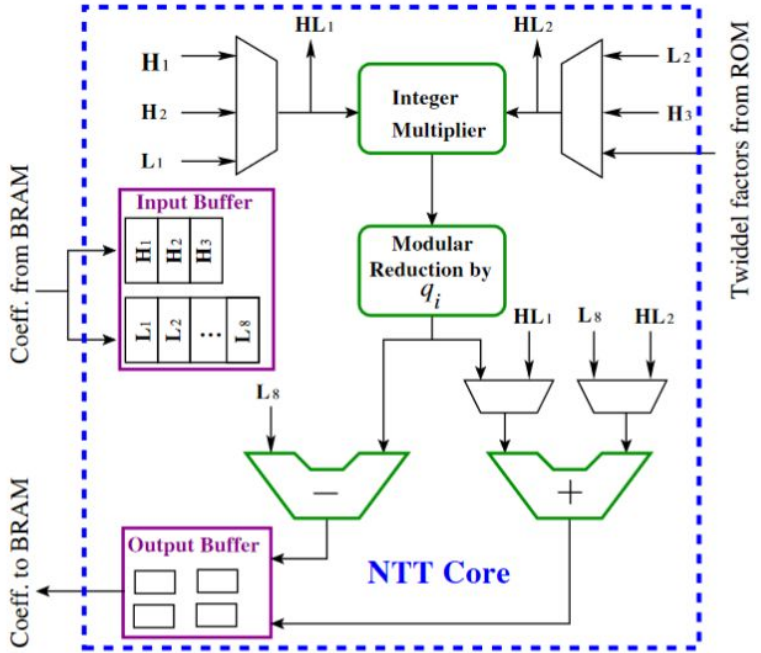
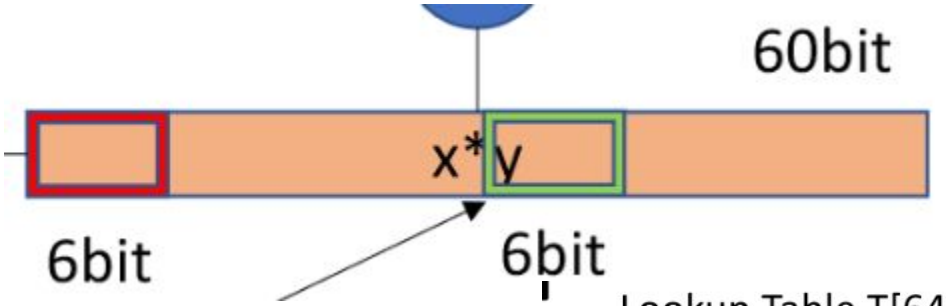


Fig. 4. Architecture of NTT Core.

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
72	72	0.380 us	0.380 us	72	72	none

Algorithmic Optimization of Modulo : Sliding Window

- Find (x * y) % p
  - (x, y, p : 30 bits)
- Modular Lookup Table
  - 2^6 = 64 = Table Size
- 6 Bit Width Sliding Window
  - from MSB to lower



Lookup Table T[64]

w	$w \cdot 2^{30} \bmod p$
0	
1	
2	
...	
63	

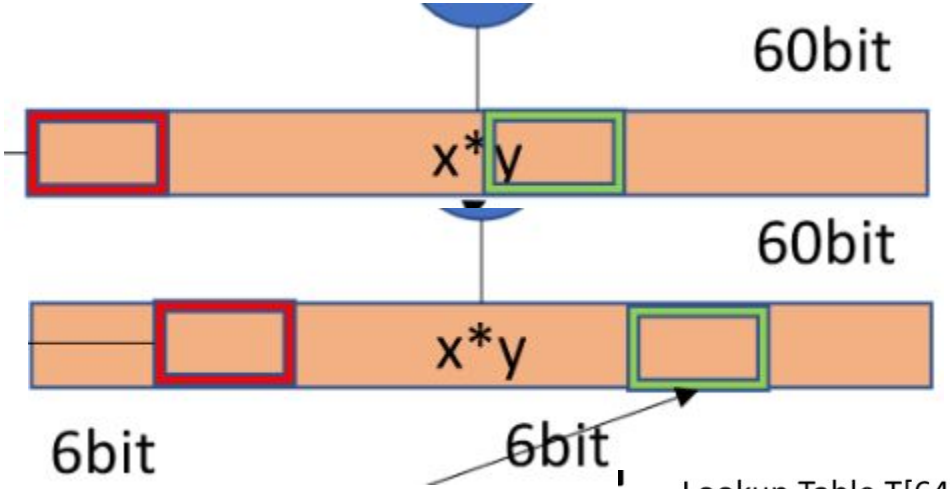
$$x \cdot y \equiv w[59,54] \cdot 2^{54} + \ldots + w[29,24] \cdot 2^{24} \ldots (mod\, p)$$

$$x \cdot y \equiv \left(w[59,54] \cdot 2^{30}\right) \cdot 2^{54-30} + \ldots (mod\, p)$$

$$x \cdot y \equiv 0 \cdot 2^{54} + \ldots + (T[w[59,54]] + w[29,24]) \cdot 2^{24} \ldots (mod\, p)$$

Overflow to Last Window : 1 Bit Overlap Sliding Windows

- p & Its Residue : 30 bits
- Table Element T[w] : 30 bits



- T[w[59,54]] Add 30 Bits To [53,24]
- T[w[59,54]] + Old [53,24] =
  30 Bits + 30 Bits = 1 Bit Overflow !!!

- So 1 Bit Overlap (as window slides to lower) →
- w[59,54] → next w[54,49]

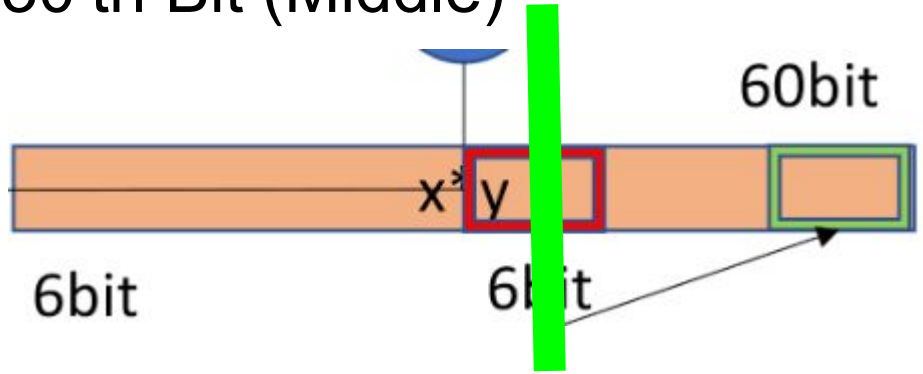
Lookup Table T[64]

w	$w \cdot 2^{30} \bmod p$
0	
1	
2	
...	
63	

$$x \cdot y \equiv 0 \cdot 2^{54} + \ldots + (T[w[59,54]] + w[29,24]) \cdot 2^{24} \ldots (mod\, p)$$

Sliding Window Until Crossing 30’th Bit (Middle)

- Crossing Window w[34,29]
- Just Take Bits Above 30’th Bit Line :
  - Only w[34,30]
  - Add T[w[34,30]] to [29,0]



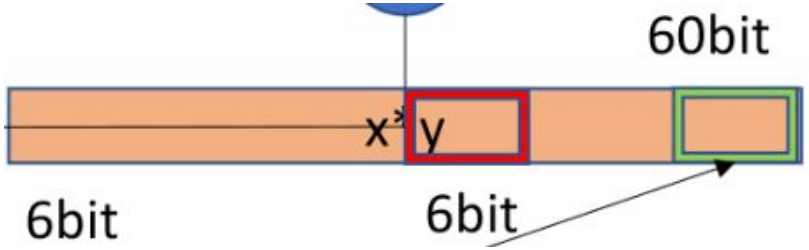
Lookup Table T[64]

w	$w \cdot 2^{30} \bmod p$
0	
1	
2	
...	
63	

Sliding Window Crossing 30 Bit Line

Algorithmic Optimization Finale: Might Minus p or 2p

- Now the Residue After The 30’th Bit Line →
  - From Max 60 Bits
  - To At Most 30 Bits



- p == Roughly 30 Bits Prime →
- 1 Bit Difference == 2 Times →
- Final Residue = Residue minus 0, p, or 2p

Algorithmic Optimization For Signed : Negative Shift

- For N Bits Signed →

$$-2^{N-1} \sim 2^{N-1} - 1$$

$$For\ N = 64\ Bits,$$

$$\begin{aligned} -2^{63} &\sim 2^{63} - 1 \\ +) \qquad & \quad 2^{63}\ (\equiv Negative\ Shift) \\ 0 \quad &\sim 2^{64} - 1 \end{aligned}$$

- Add Negative Shift → Become Nonnegative
- After Unsigned Modulo, Minus Negative Shift's Residue Back

Further Hardware Optimization : Factors

- Pipeline → Final II = 1
- Unroll

- Sliding Window Width (6 Bits) → →
  - How Many Loops of Sliding Window → Latency
  - Modular Table Size = (#p) * 2^6 = (#p) * 64 → (#p) * 2^(window width) → BRAM / FF / LUT

- Modular Table Memory Layout → →
  - Array Partition : complete dim=0
  - Resource : core=ROM_nP_LUTRAM

The Power of Pipeline

- Final II = 1
- left:only algorithmic optim right:algorithmic optim + pipeline

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
93	93	0.465 us	0.465 us	93	93	none

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	2	4	2169	3820	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	41	-
Register	-	-	72	-	-
Total	2	4	2241	3861	0
Available	280	220	106400	53200	0
Utilization (%)	~0	1	2	7	0

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
30	30	0.150 us	0.150 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	20	4	2052	1141	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	288	64	-
Total	20	4	2340	1209	0
Available	280	220	106400	53200	0
Utilization (%)	7	1	2	2	0

Array Partition on Table with Few / Many Primes (actual)

- Array_Partition variable=Modular_Table complete dim=0
- left:with 2 primes (idealized) right:with 15 primes (actual case) :

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
19	19	95.000 ns	95.000 ns	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	4	1859	4748	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	276	64	-
Total	0	4	2135	4816	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	2	9	0

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
576	576	2.880 us	2.880 us	1	1	function

Small Window Width, Making the Table Small

- many primes (actual, 15)
- left:window width=2,array_partition complete dim=0 right:window_width=15

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
42	42	0.210 us	0.210 us	1	1	function

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
576	576	2.880 us	2.880 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	4	4323	10858	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	300	64	-
Total	0	4	4623	10926	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	4	20	0

Array Parition on Table & Negative Shift Mod Table

left: without right: with partition on negative shift mod table

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
42	42	0.210 us	0.210 us	1	1	function

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
44	44	0.220 us	0.220 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	4	4323	10858	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	300	64	-
Total	0	4	4623	10926	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	4	20	0

1D / 2D Table Initialization

- left:window width=2, 2D, array_partition right:window_width=2, 1D, array_partition

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
42	42	0.210 us	0.210 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	4	4323	10858	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	300	64	-
Total	0	4	4623	10926	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	4	20	0

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
42	42	0.210 us	0.210 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	0	4	4323	10858	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	300	64	-
Total	0	4	4623	10926	0
Available	280	220	106400	53200	0
Utilization (%)	0	1	4	20	0

Table Resource

- left:Resource : core=ROM_nP_LUTRAM right:only pipeline

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
30	30	0.150 us	0.150 us	1	1	function

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
30	30	0.150 us	0.150 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	12	4	2262	1621	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	288	64	-
Total	12	4	2550	1689	0
Available	280	220	106400	53200	0
Utilization (%)	4	1	2	3	0

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	20	4	2052	1141	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	288	64	-
Total	20	4	2340	1209	0
Available	280	220	106400	53200	0
Utilization (%)	7	1	2	2	0

Component Resource Trade-off on Table Configuration

Resource Trade-off on Table Configuration

- Many Modulo Cores of Few Number of Primes → Small Table Size → Latency

If One of x, y Is Fixed : Further Optimzed

Modular reduction

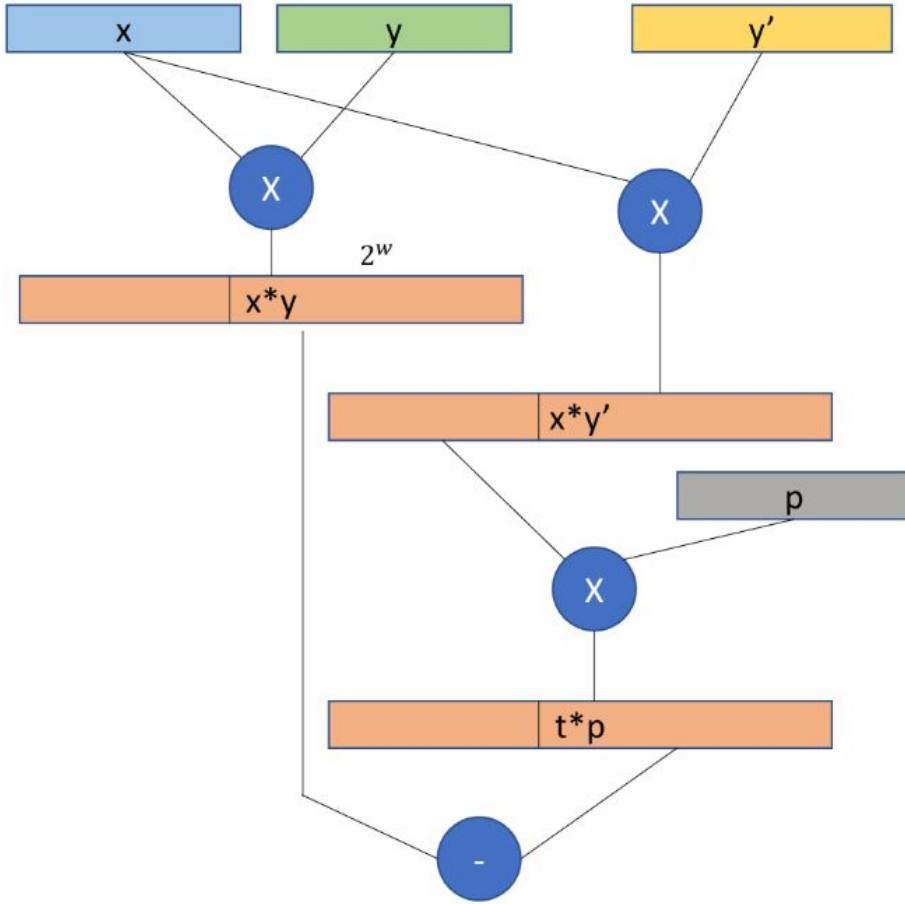
- Implement (a*b)%p
  - x,y,p 53 bits

Algorithm 1 Optimized Modular Mult. | $\texttt{MulRed}(x, y, y', p)$

Input: $x, y \in \mathbb{Z}_p, p < 2^{w-2}$, and $y' = \lfloor y \cdot 2^w / p \rfloor$
Output: $z \leftarrow x \cdot y \pmod{p}$

1: $z \leftarrow x \cdot y \pmod{2^w}$ ▷ the lower word of the product
2: $t \leftarrow \lfloor x \cdot y' / 2^w \rfloor$ ▷ the upper word of the product
3: $z_\epsilon \leftarrow t \cdot p \pmod{2^w}$ ▷ the lower word of the product
4: $z \leftarrow z - z_\epsilon$ ▷ single-word subtraction
5: **if** $z \geq p$ **then**
6: $z \leftarrow z - p$
7: **end if**

y,y' is know



Activate W

If One of x, y Is Fixed : Further Optimzed : Comparison

- left: ASPLOS2020, 1 of x,y fixed. right: 3, ROM_nP_LUTRAM, 1st algorithm

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
17	17	85.000 ns	85.000 ns	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	292	-
FIFO	-	-	-	-	-
Instance	-	12	645	3	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	2	-	597	98	-
Total	2	12	1242	393	0
Available	280	220	106400	53200	0
Utilization (%)	~0	5	1	~0	0

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
30	30	0.150 us	0.150 us	1	1	function

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	12	4	2262	1621	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	0	-	288	64	-
Total	12	4	2550	1689	0
Available	280	220	106400	53200	0
Utilization (%)	4	1	2	3	0

Outline

- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

NTT module

- Since the whole computation is carried out on polynomials, we need to compute polynomial multiplication faster

$$Z_q / x^N + 1$$

計算規則

(1) 乘法加法, 結果要mod q
(2) 看到x^N, 帶成-1

$$x^4 = -1$$

$$2x^3 + 7x^2 + 1x^1 + 8x^0$$
$$\text{X)}\ 2x^3 + 0x^2 + 4x^1 + 8x^0$$

$$16x^3 + 56x^2 + 8x^1 + 64x^0$$
$$8x^4 + 28x^3 + 4x^2 + 32x^1$$
$$0x^5 + 0x^4 + 0x^3 + 0x^2$$
$$4x^6 + 14x^5 + 2x^4 + 16x^3$$

$$4x^6 + 14x^5 + 10x^4 + 60x^3 + 60x^2 + 40x^1 + 64x^0$$
$$60x^3 + 56x^2 + 26x^1 + 54x^0$$
$$9x^3 + 5x^2 + 9x^1 + 3x^0$$

$$Z_{17} / x^4 + 1$$

[2, 7, 1, 8] [2, 0, 4, 8]

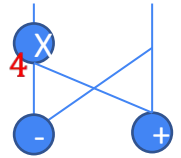
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$

NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$

[2, 7, 1, 8]



[2, -20, 1, 36]

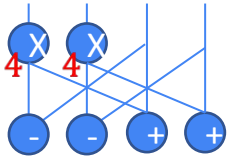
$4 = (-1)^{\frac{1}{2}} \pmod{17}$

[2, 0, 4, 8]

NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$

[2, 7, 1, 8]



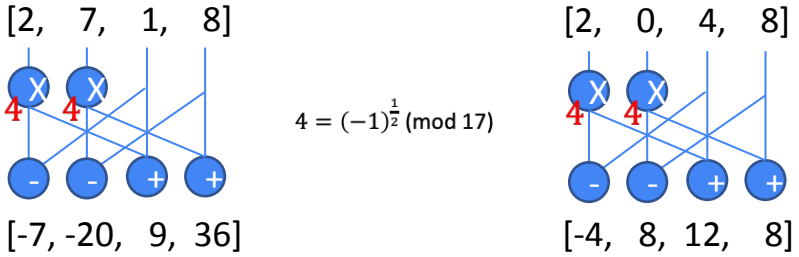
[-7, -20, 9, 36]

$4 = (-1)^{\frac{1}{2}} \pmod{17}$

[2, 0, 4, 8]

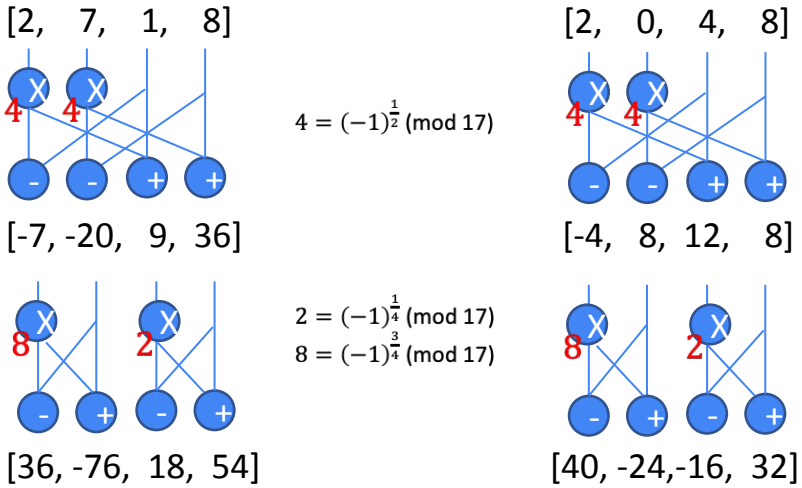
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



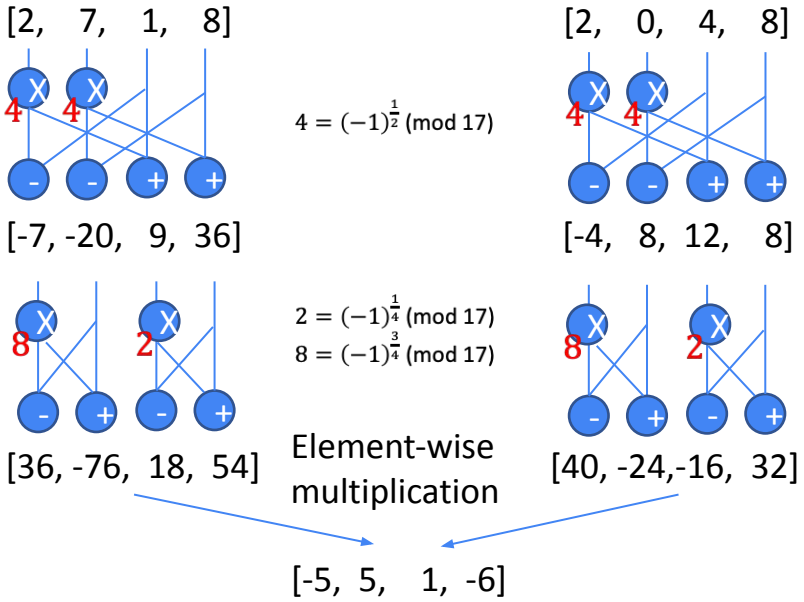
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



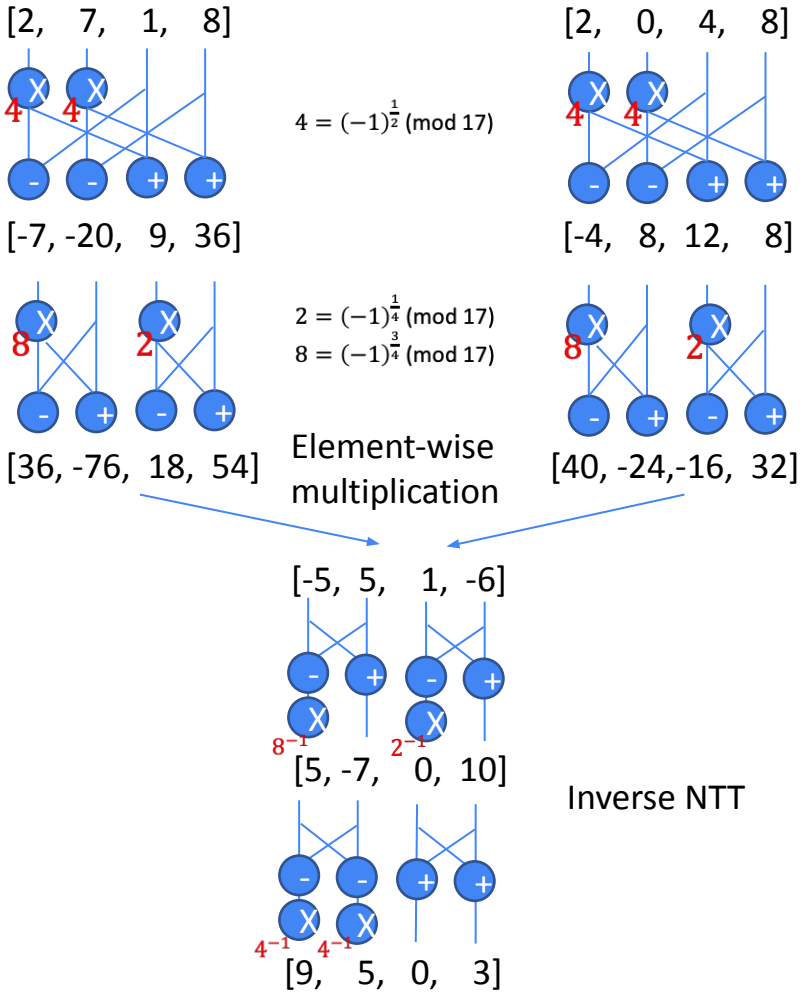
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



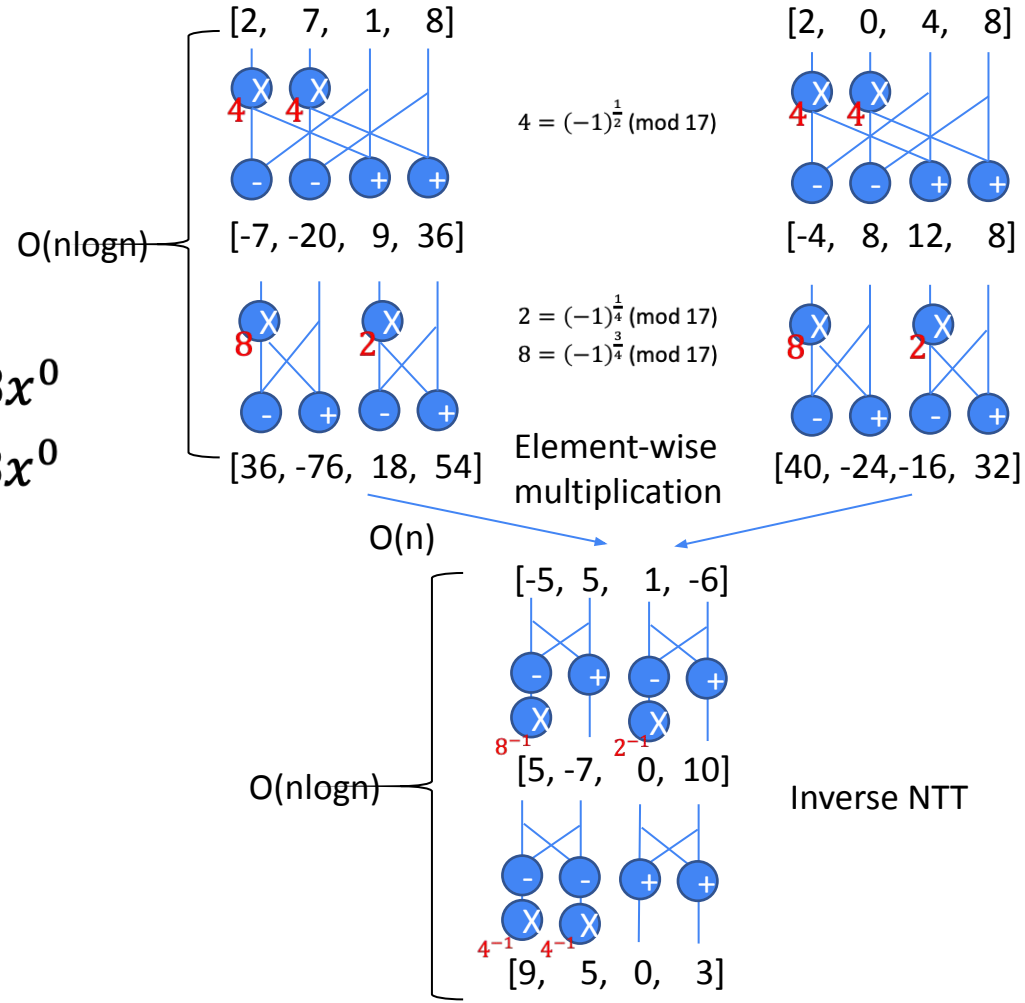
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



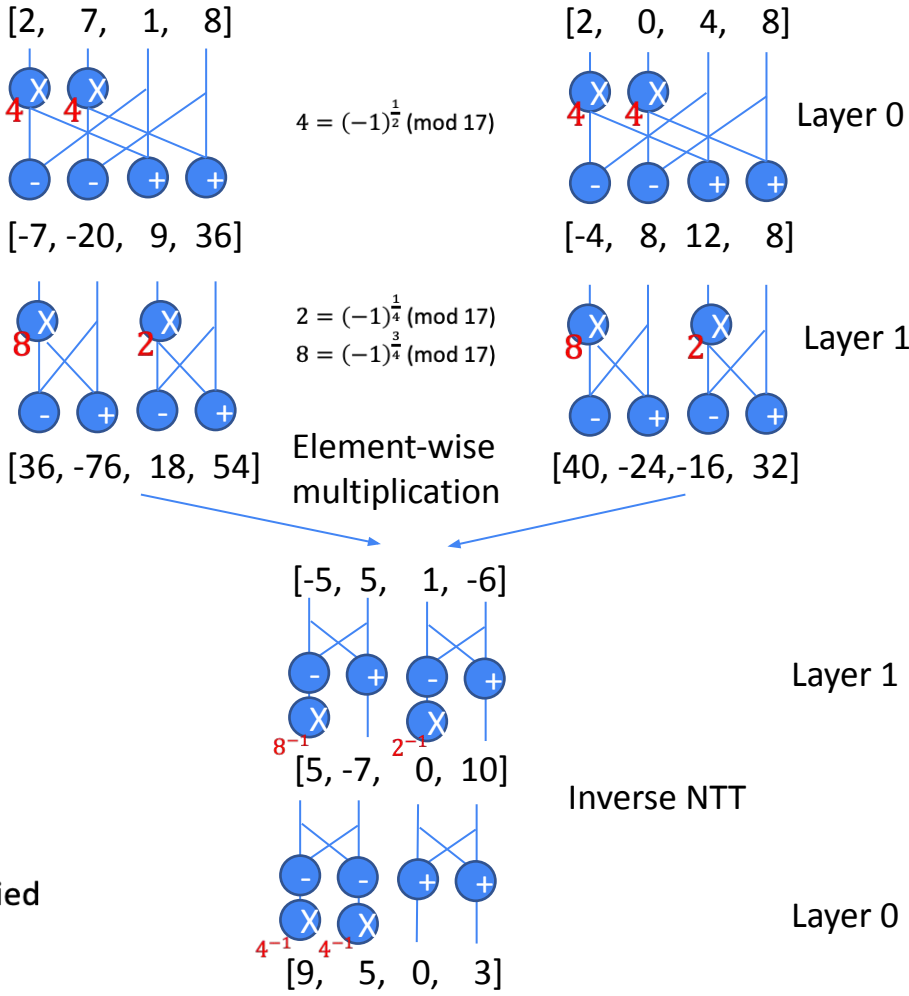
NTT

- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



NTT

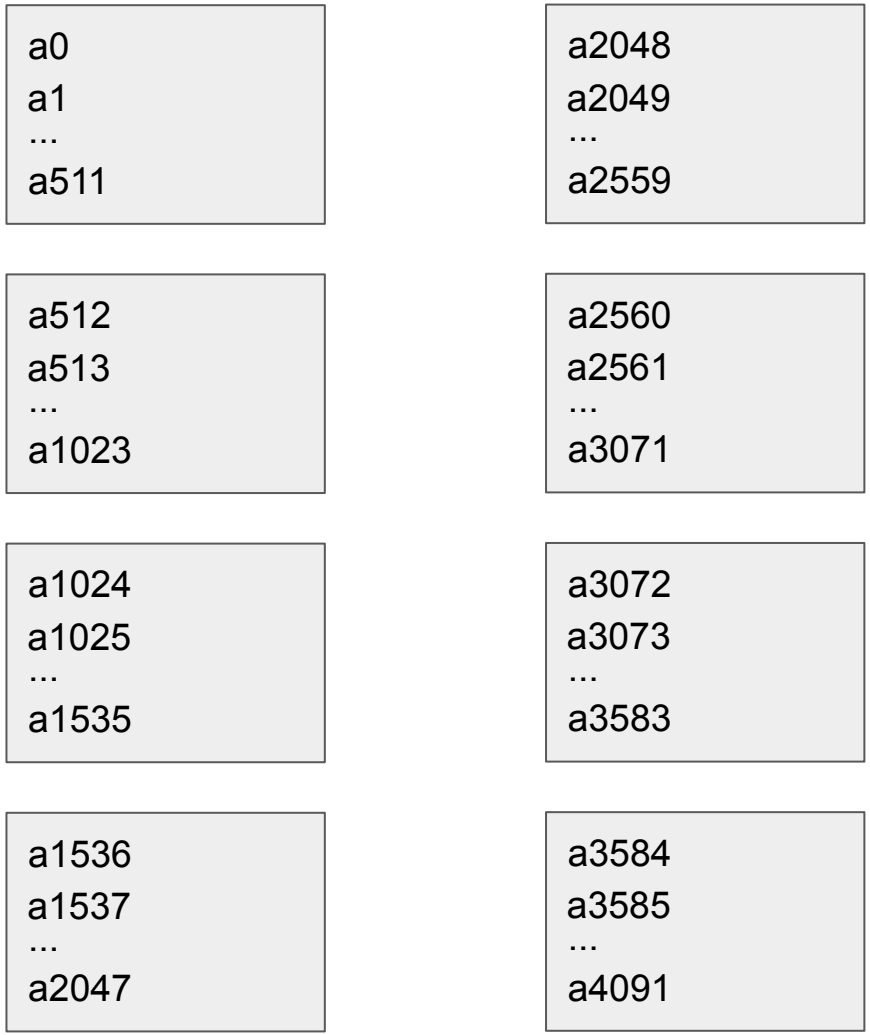
- Example $\frac{Z_{17}}{x^4+1}$
- $a \rightarrow 2x^3 + 7x^2 + 1x^1 + 8x^0$
- $b \rightarrow 2x^3 + 0x^2 + 4x^1 + 8x^0$



Layer n has 2^n different constants to be multiplied

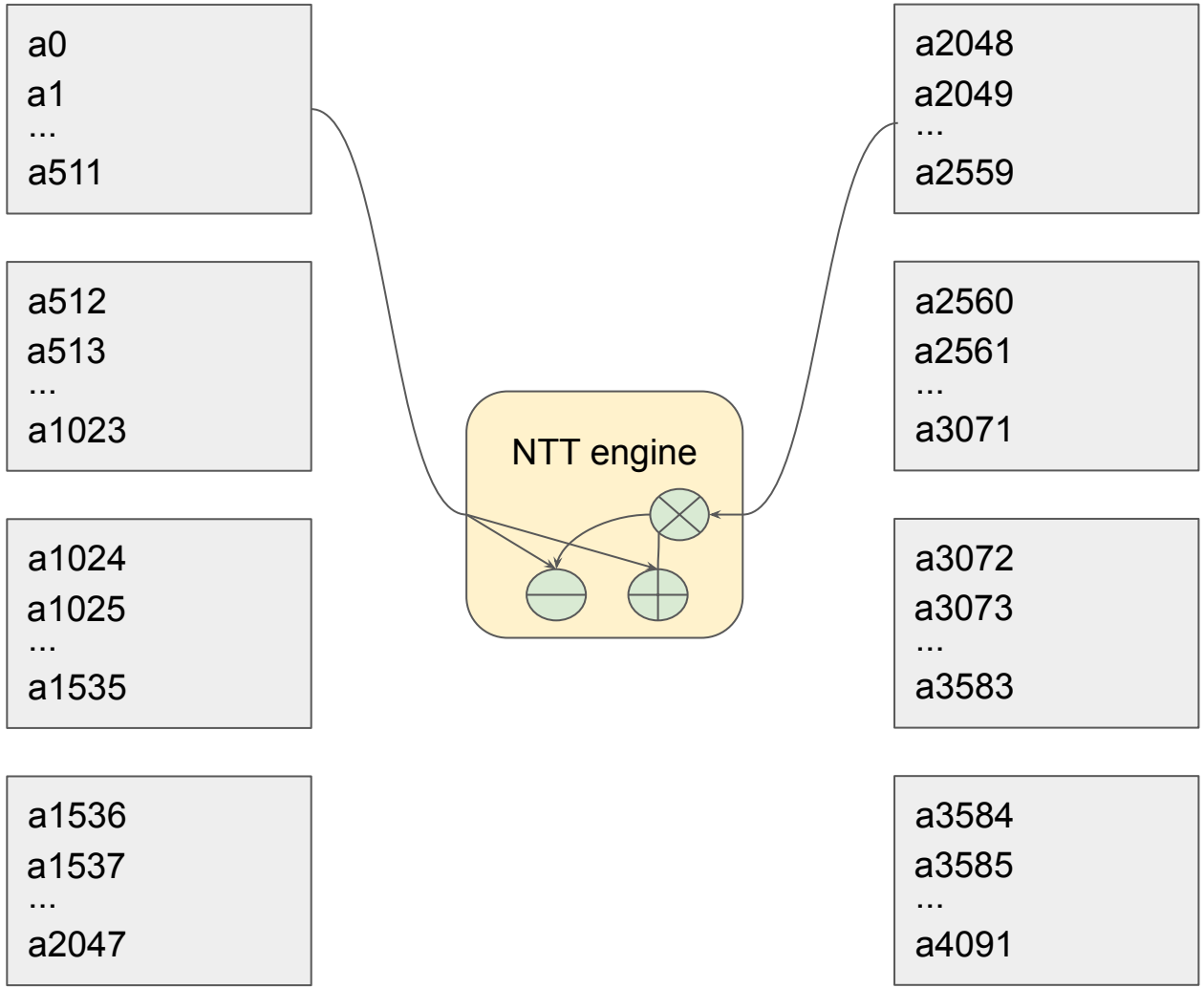
Coefficient arrangement

Suppose N=4096, each coefficient 32bits
512*32=16384 → 1 BRAM_18K



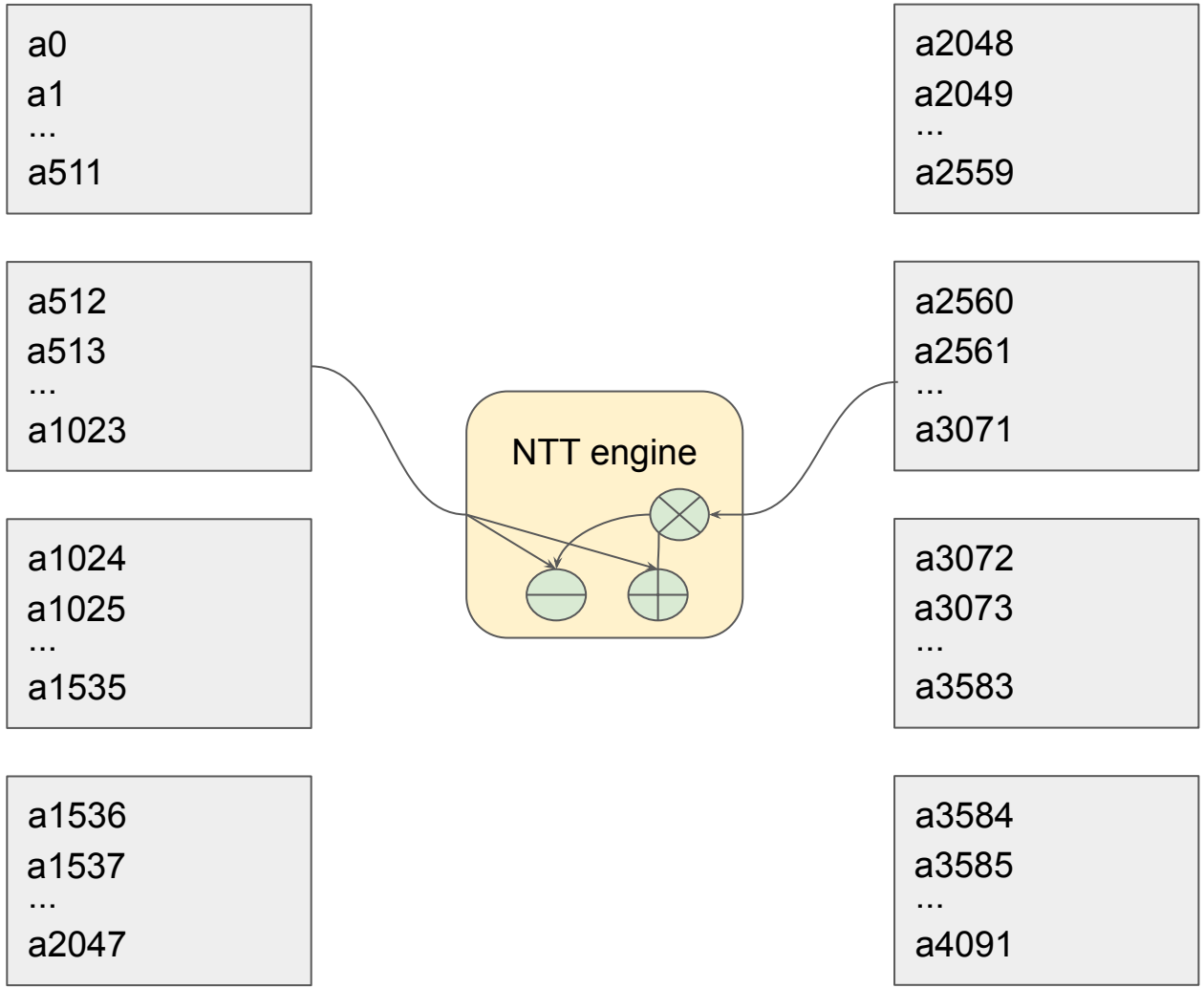
NTT engine

stage=0



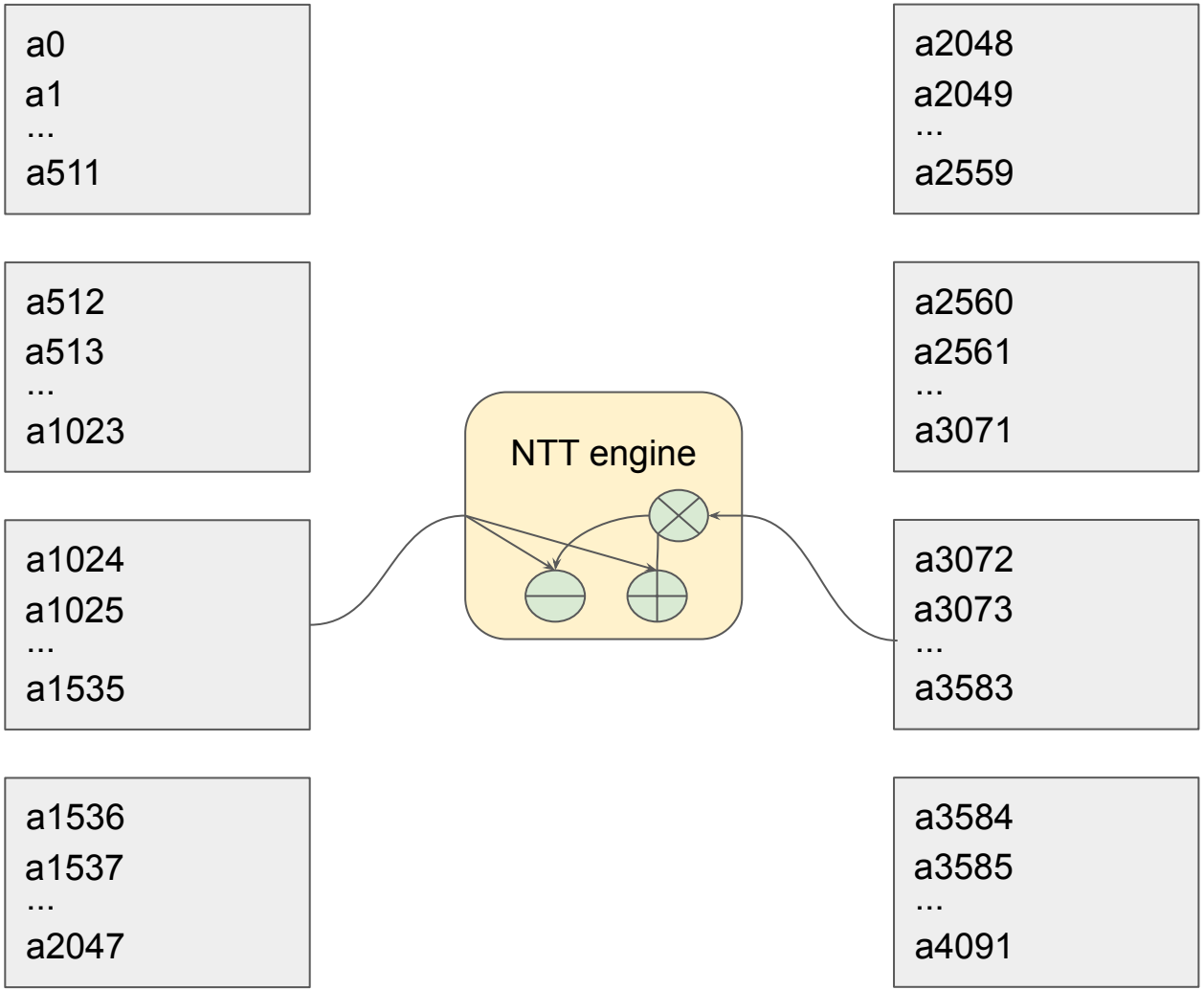
NTT engine

stage=0



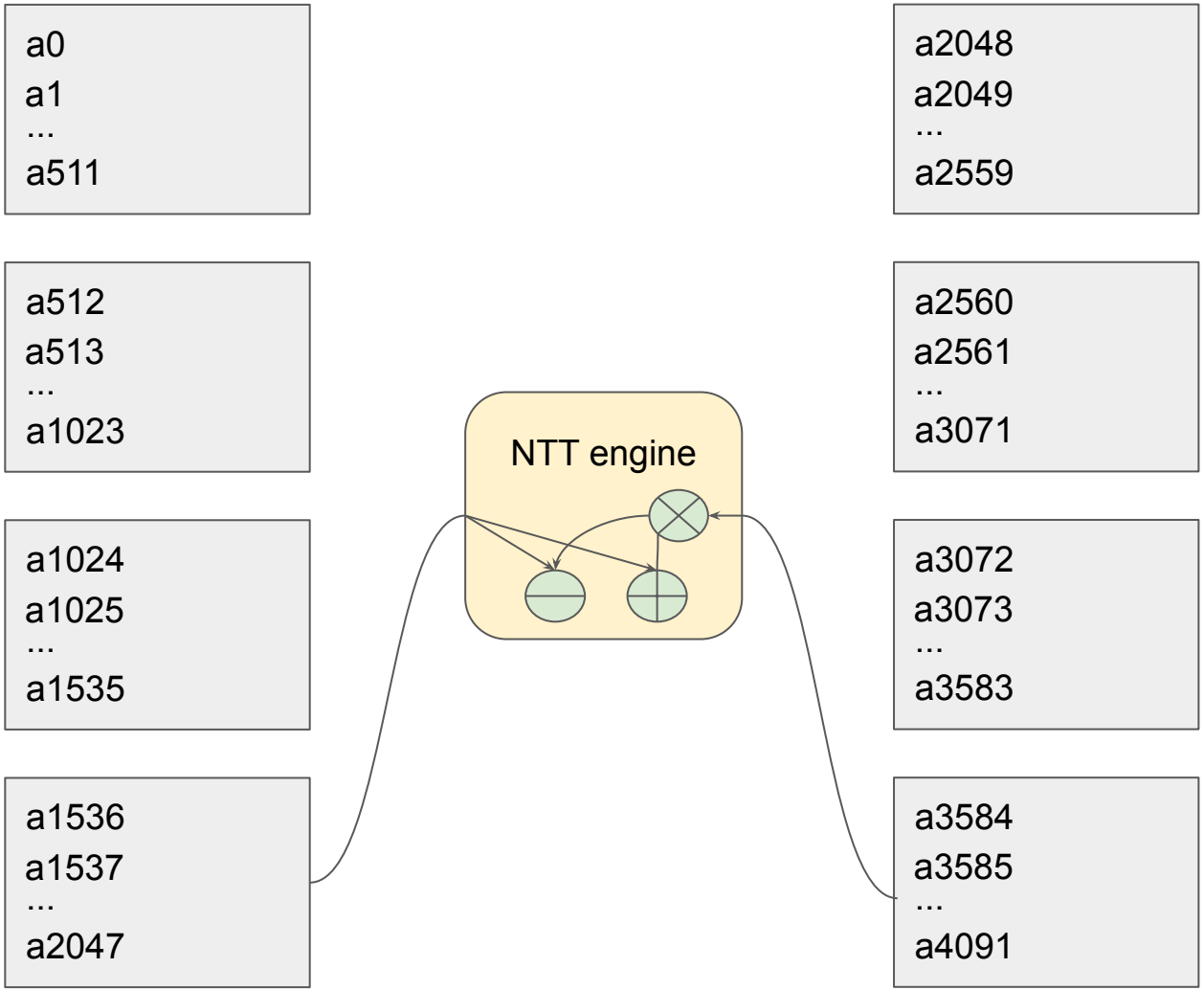
NTT engine

stage=0



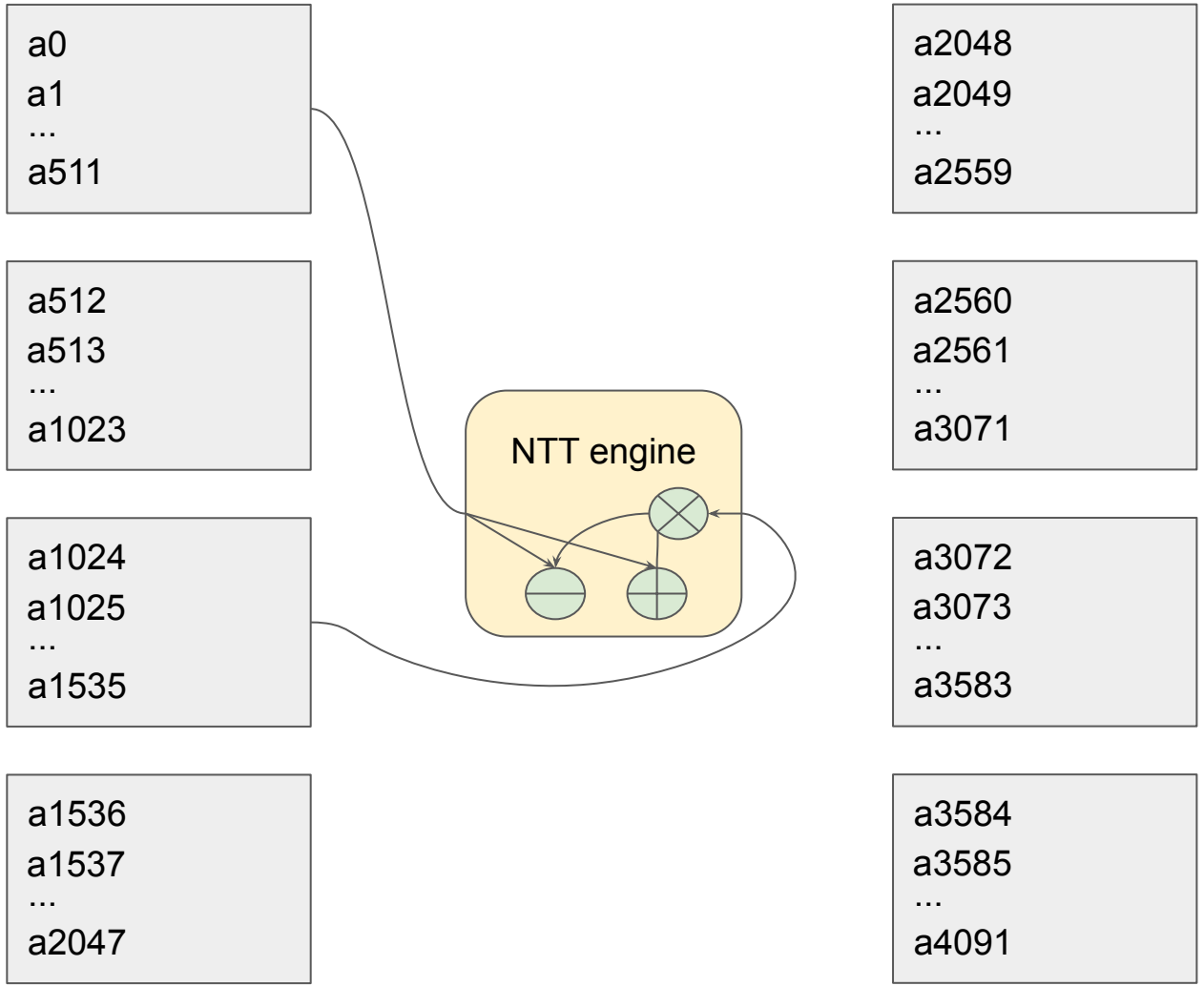
NTT engine

stage=0



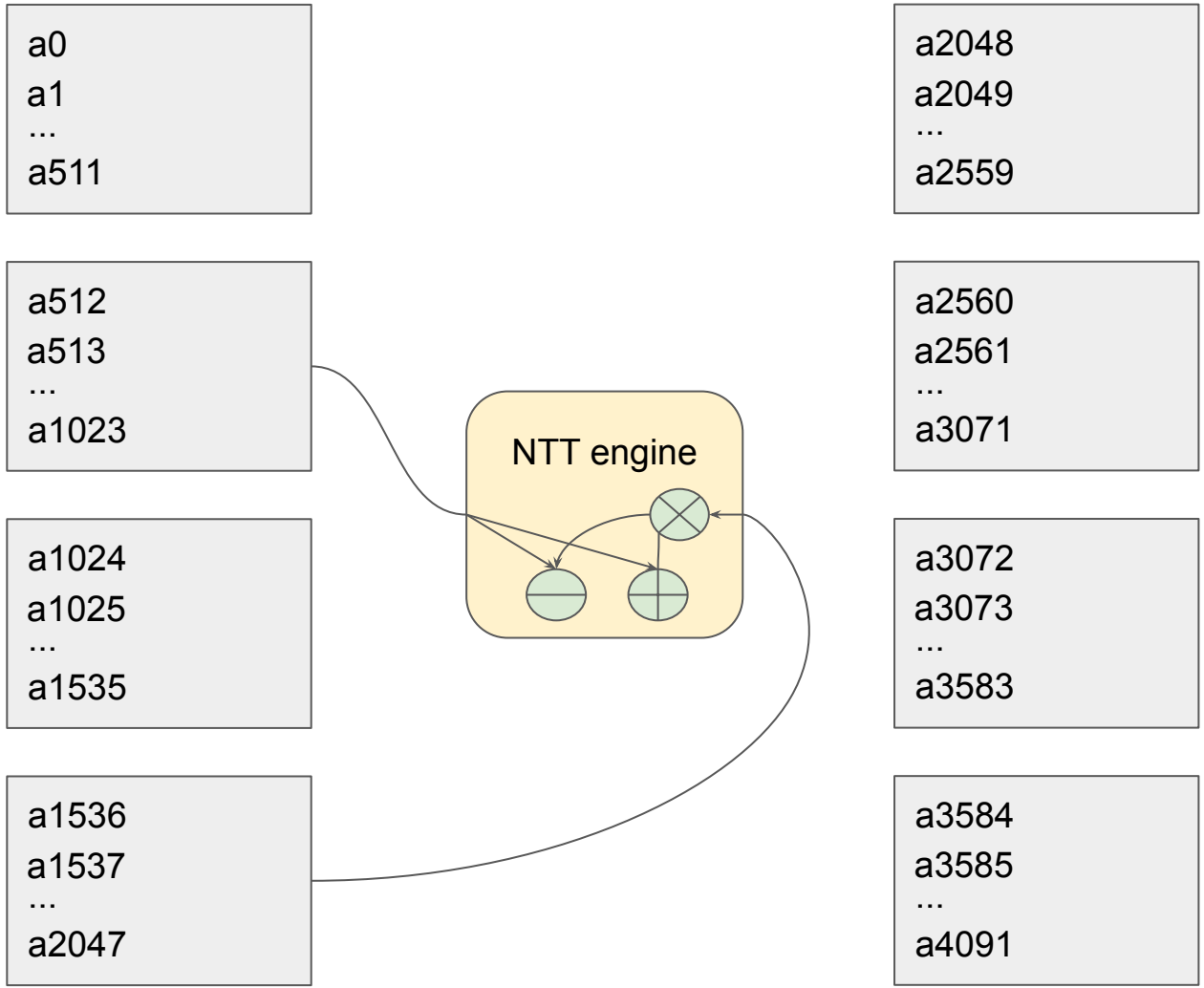
NTT engine

stage=1



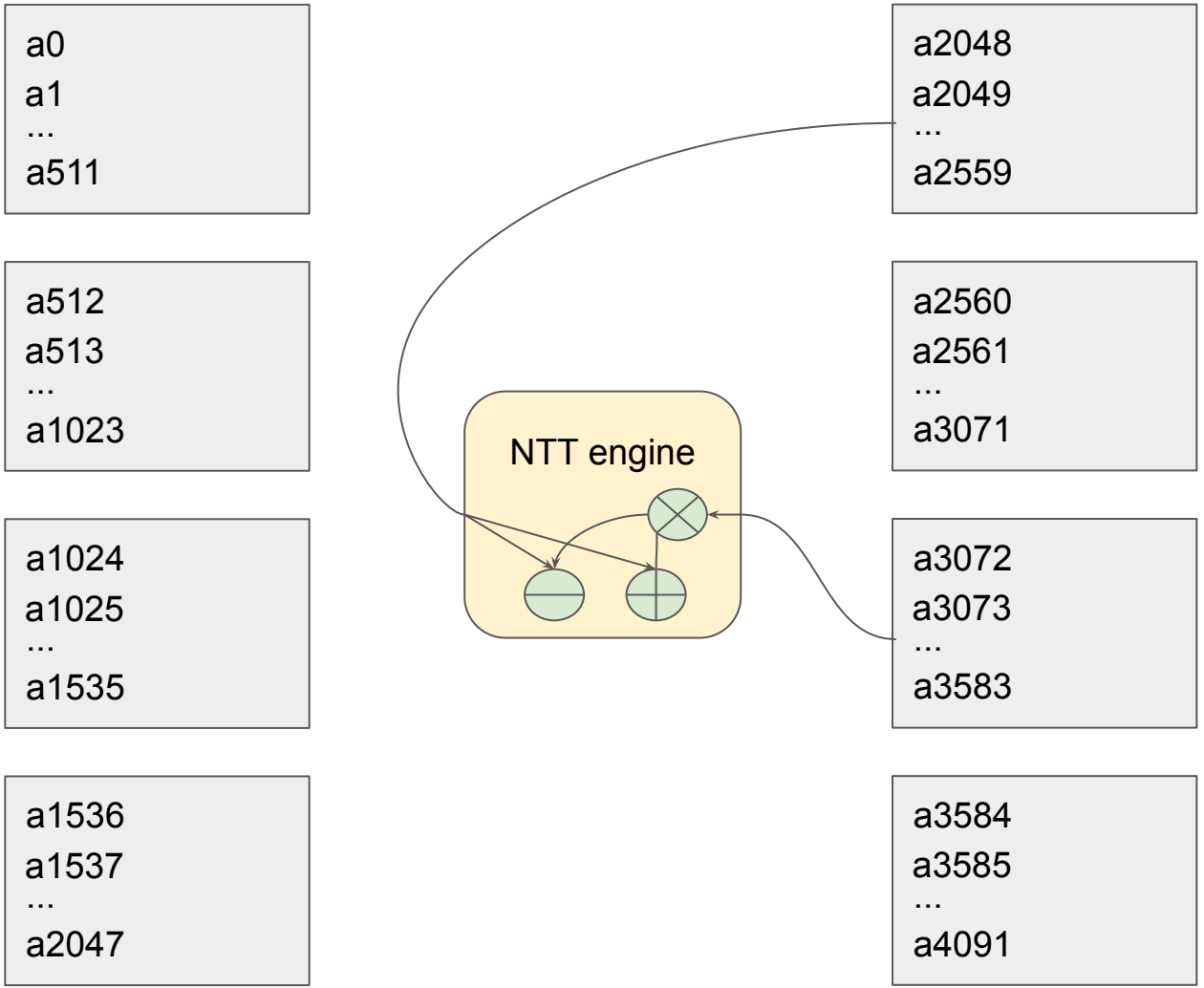
NTT engine

stage=1



NTT engine

stage=1



NTT engine

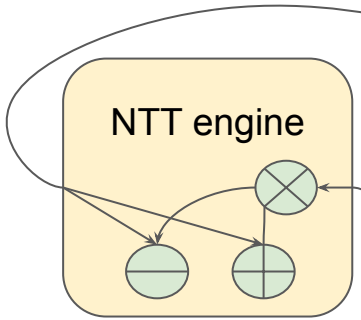
stage=1

a0
a1
...
a511

a512
a513
...
a1023

a1024
a1025
...
a1535

a1536
a1537
...
a2047



a2048
a2049
...
a2559

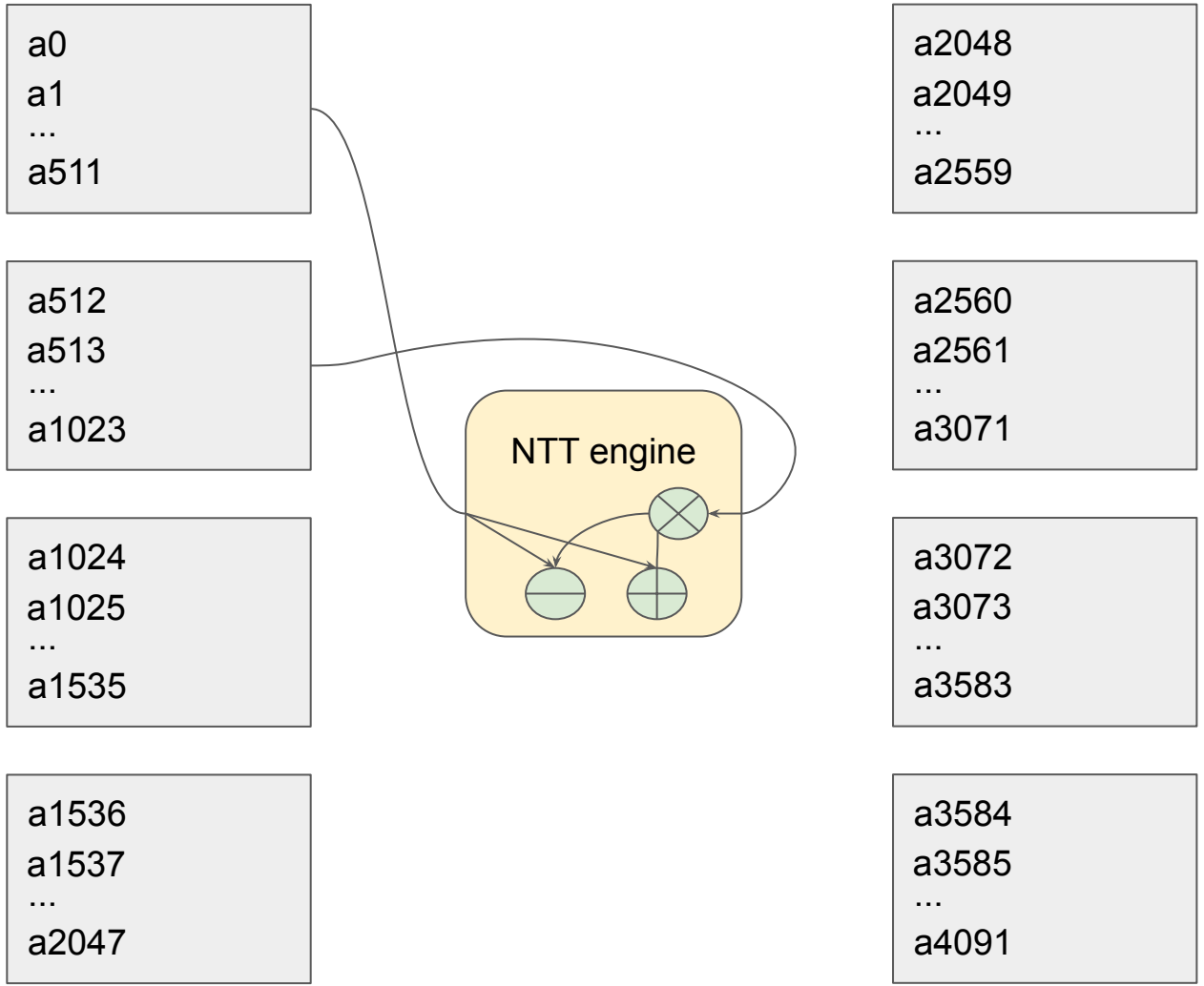
a2560
a2561
...
a3071

a3072
a3073
...
a3583

a3584
a3585
...
a4091

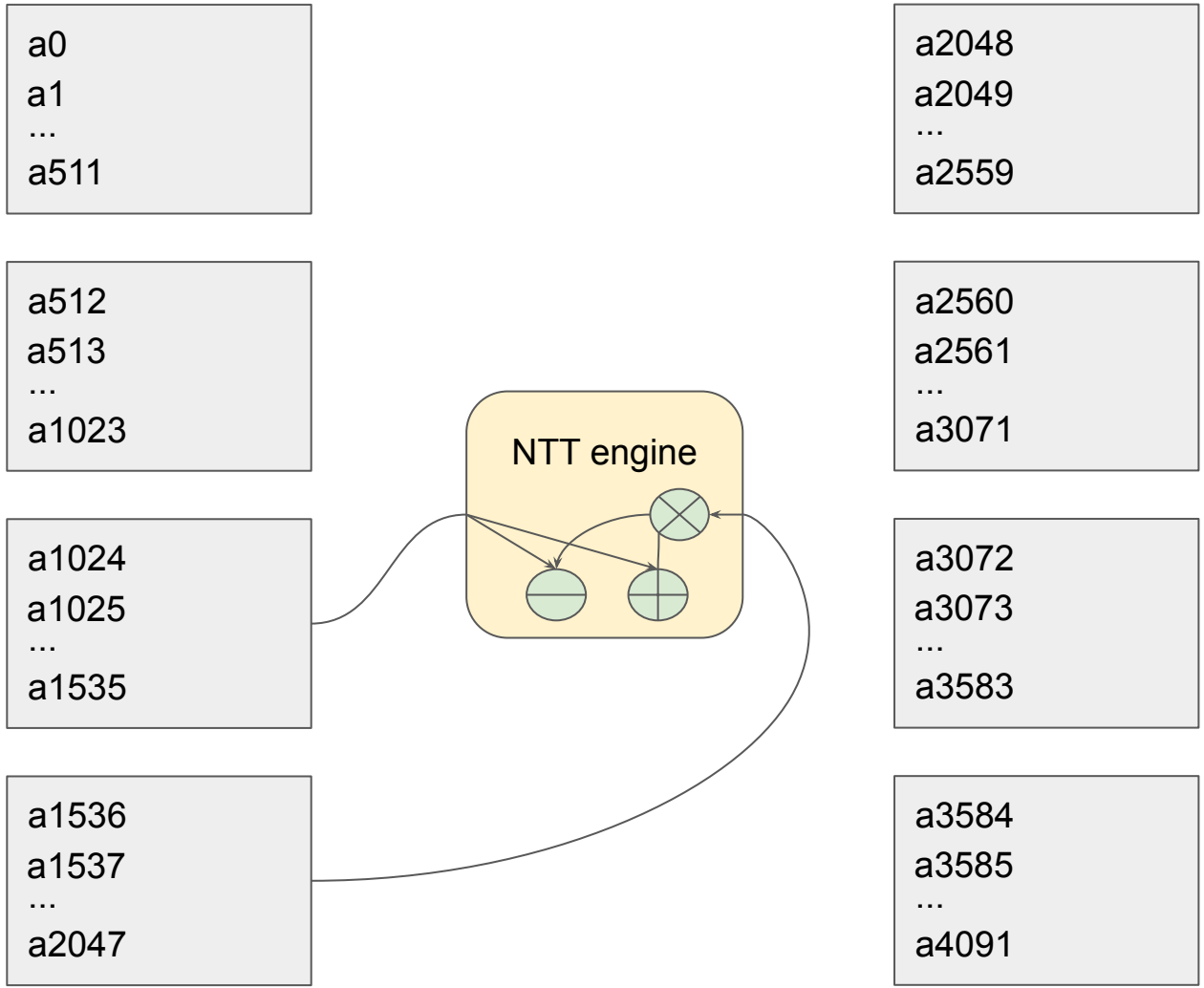
NTT engine

stage=2



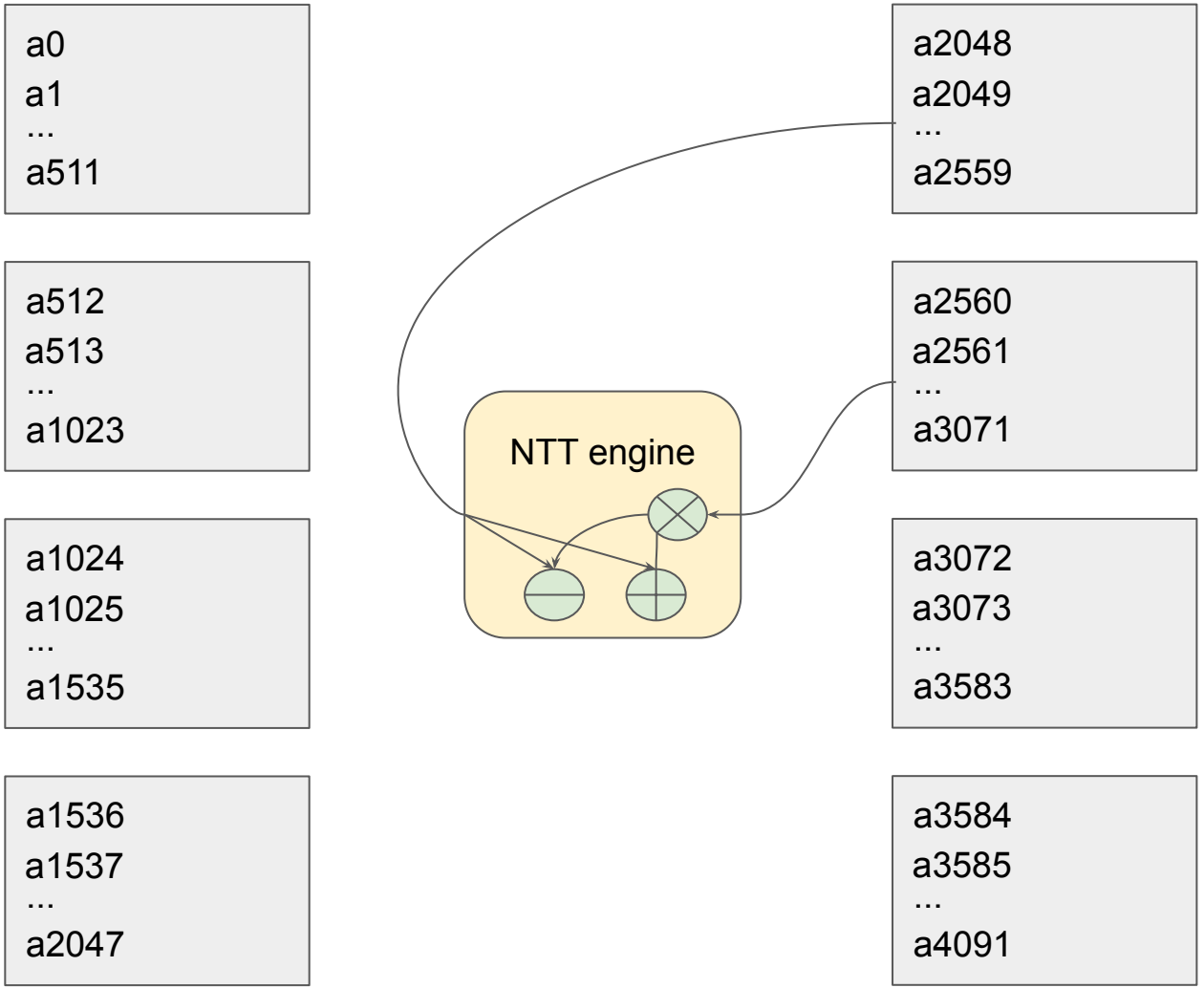
NTT engine

stage=2



NTT engine

stage=2



NTT engine

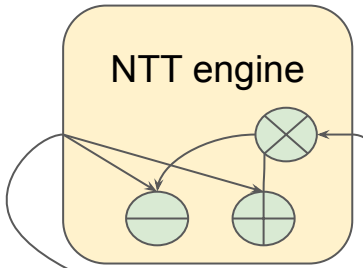
stage=2

a0
a1
...
a511

a512
a513
...
a1023

a1024
a1025
...
a1535

a1536
a1537
...
a2047



a2048
a2049
...
a2559

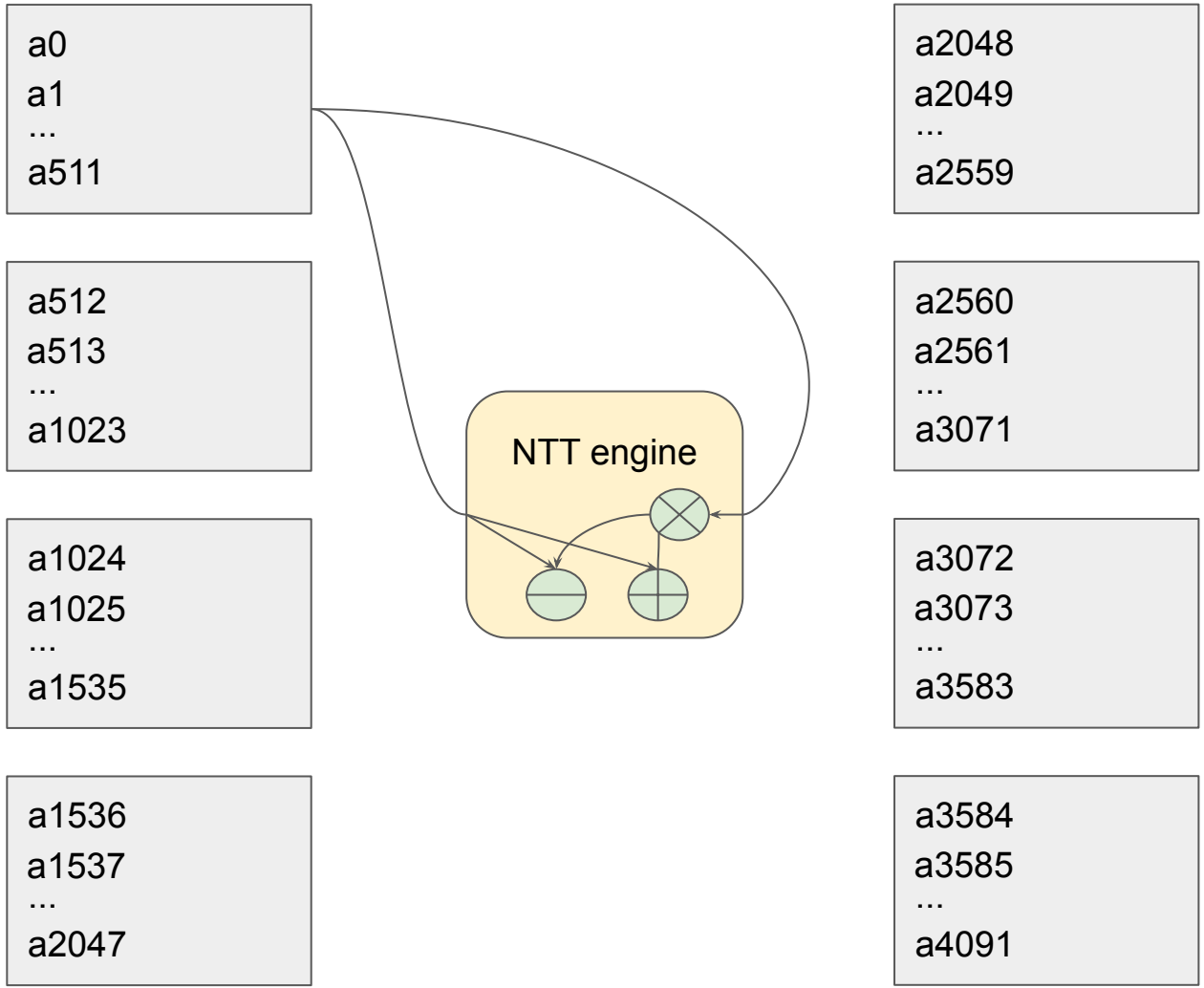
a2560
a2561
...
a3071

a3072
a3073
...
a3583

a3584
a3585
...
a4091

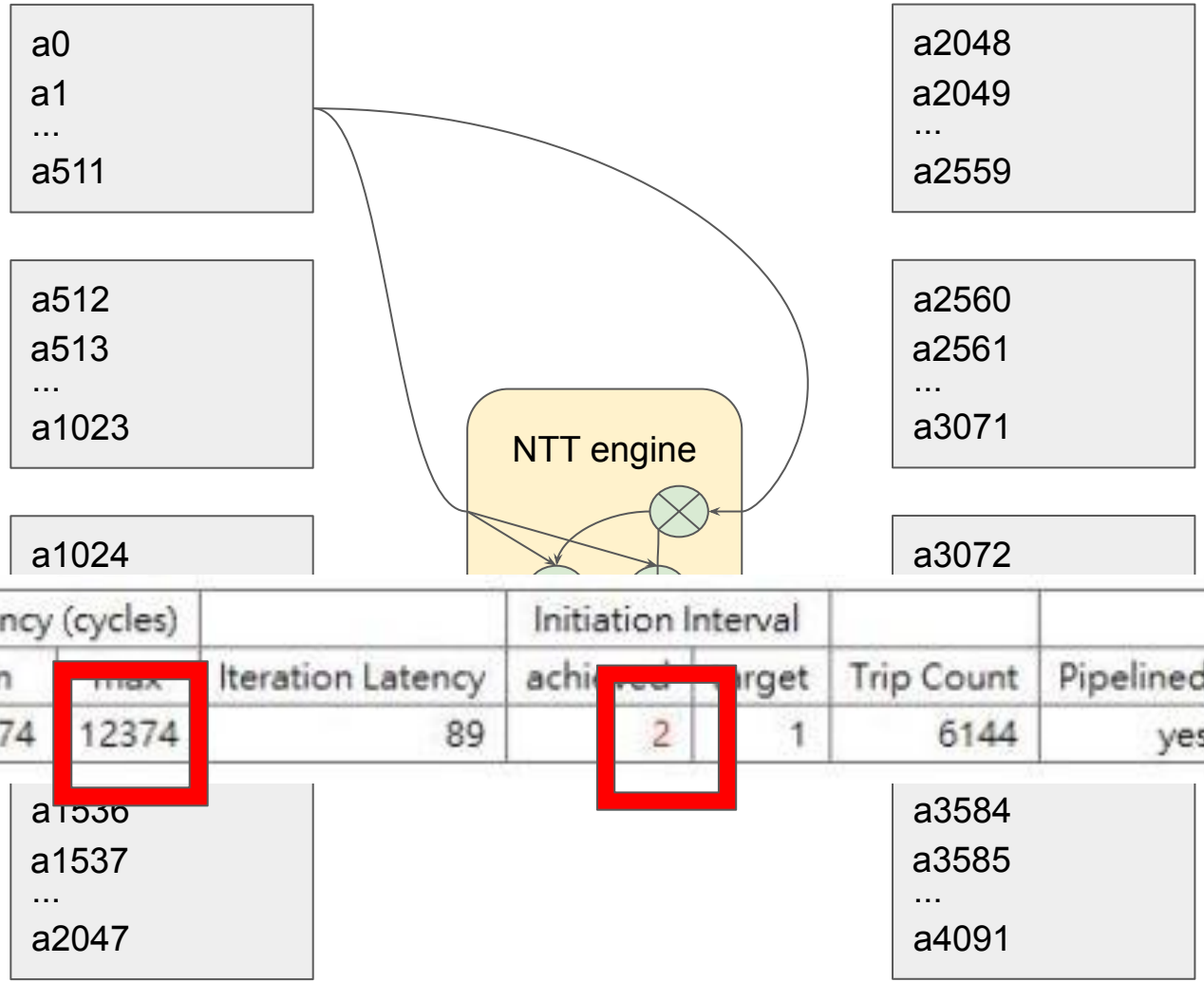
NTT engine

stage=3



NTT engine

stage=3



	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- stage_butterfly	12374	12374	89	2	1	6144	yes

Reorder and pack data

[a0, a2048]
[a1, a2049]
…
[a255, a2559]

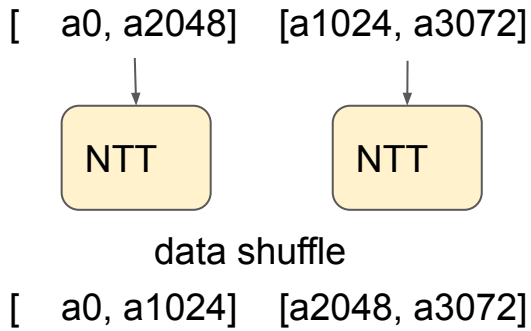
[a1024, a3072]
[a1025, a3073]
…
[a1279, a3327]

Reorder and pack data

[a0, a2048]
[a1, a2049]
…
[a255, a2559]

[a1024, a3072]
[a1025, a3073]
…
[a1279, a3327]

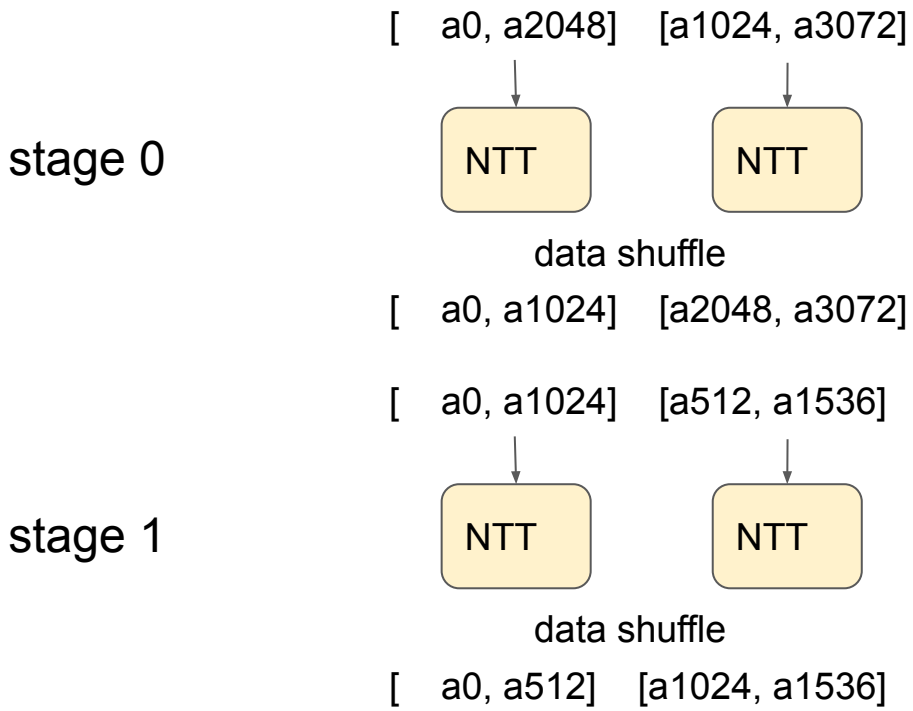
stage 0



Reorder and pack data

[a0, a2048]
[a1, a2049]
…
[a255, a2559]

[a1024, a3072]
[a1025, a3073]
…
[a1279, a3327]

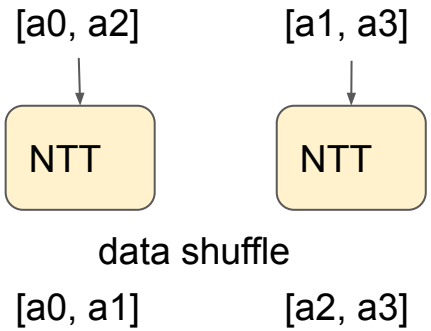


Reorder and pack data

[a0, a2048]
[a1, a2049]
…
[a255, a2559]

[a1024, a3072]
[a1025, a3073]
…
[a1279, a3327]

stage 11



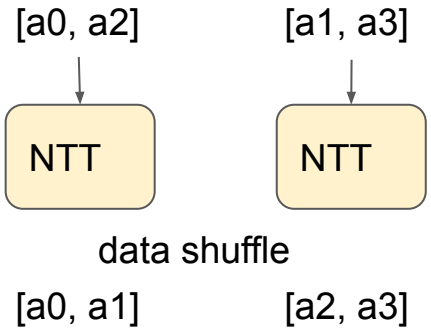
Same NTT can be shared in
two different iterations.
Only need to buffer the result

Reorder and pack data

[a0, a2048]
[a1, a2049]
…
[a255, a2559]

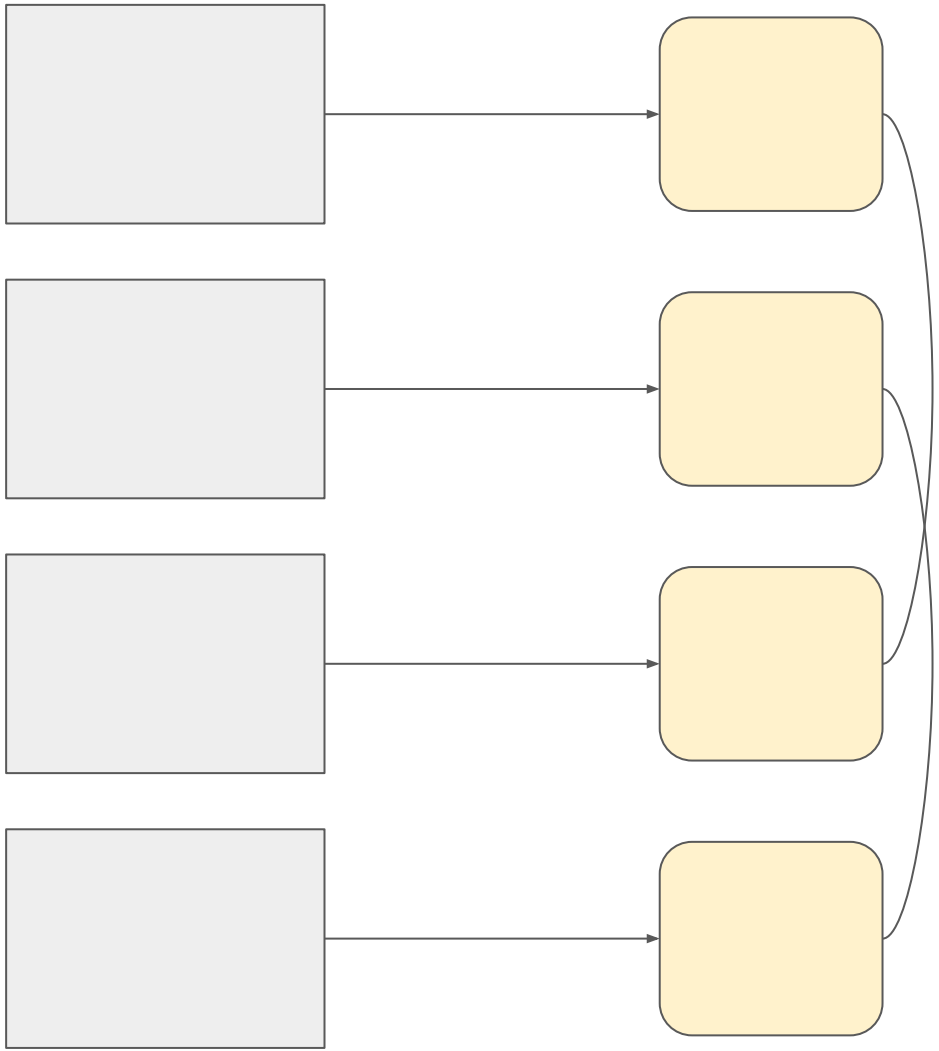
[a1024, a3072]
[a1025, a3073]
…
[a1279, a3327]

stage 11



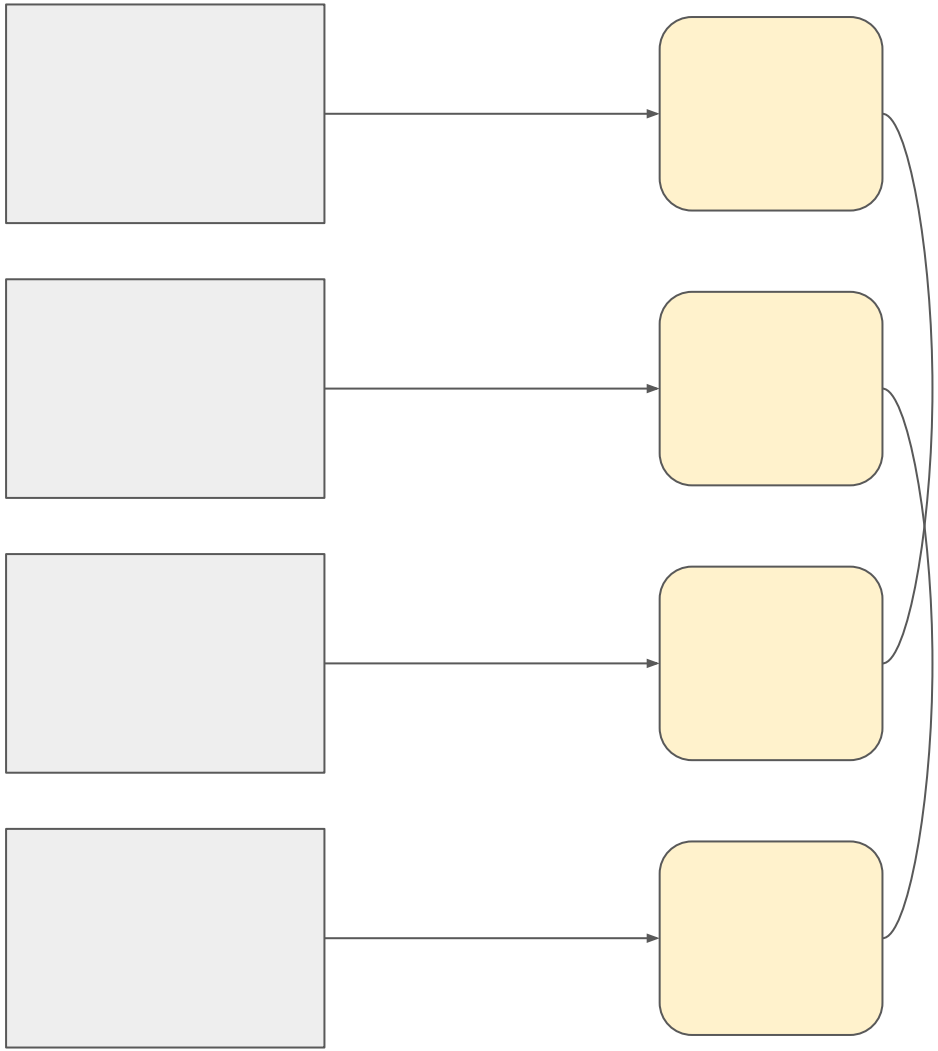
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- stage	6420	6564	535 ~ 547	-	-	12	no
+ group	528	528	18	1	1	512	yes

Variable number
NTT engine



Variable number NTT engine

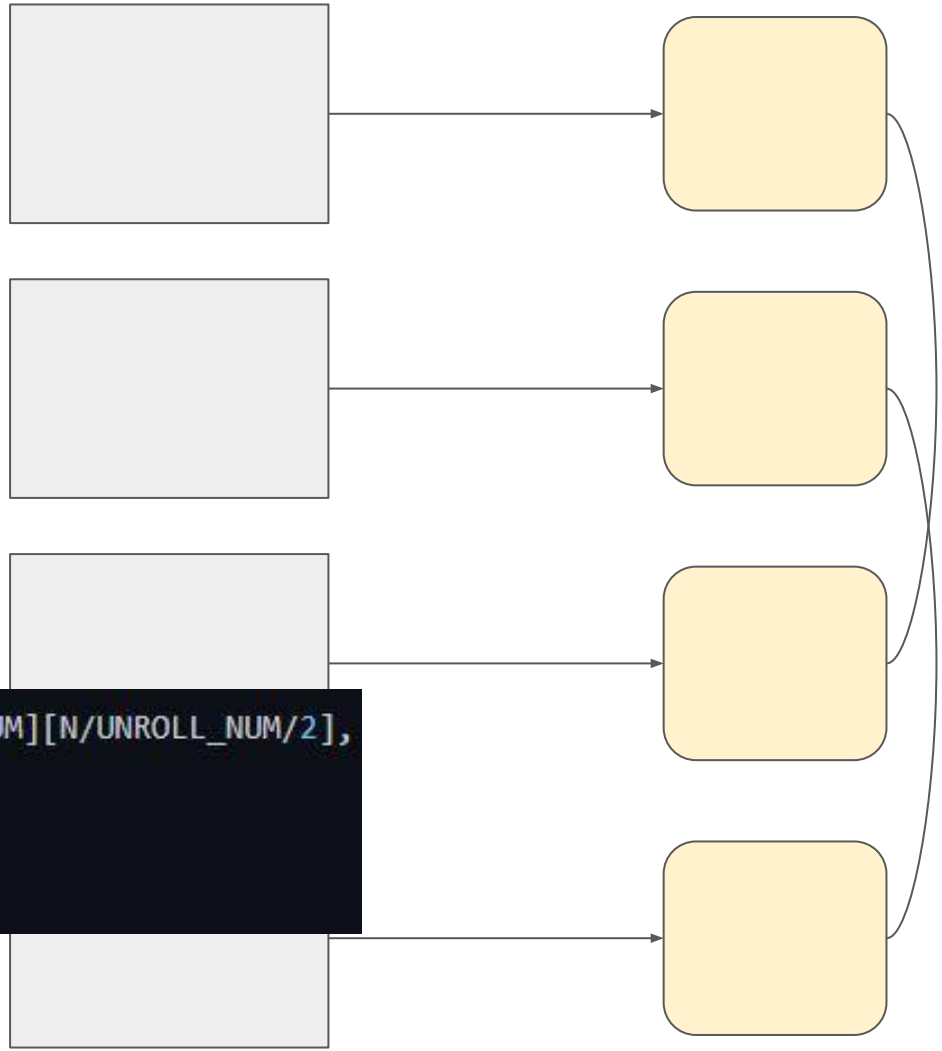
Better write 2D array of poly coefficient
to explicitly tell that
there’s no memory conflict here



Variable number NTT engine

Better write 2D array of poly coefficient
to explicitly tell that
there's no memory conflict here

```
void NTT(ap_uint<2*PRIME_BIT> poly_coeff[UNROLL_NUM][N/UNROLL_NUM/2],
         bool inv,
         ap_uint<PRIME_BIT> mod,
         ap_int<PRIME_BIT>  mprime)
```



Different number of UNROLL

N=4096	BRAM_18K	DSP48E	FF	LUT	latency (cycle)
UNROLL=1	18	13	2770	3186	24877
UNROLL=2	20	25	4027	5617	12553
UNROLL=4	40	49	7100	10335	6421
UNROLL=8	80	97	13528	20091	3385

Different number of UNROLL

N=4096	BRAM_18K	DSP48E	FF	LUT	latency (cycle)
UNROLL=1	18	13	2770	3186	24877
UNROLL=2	20	25	4027	5617	12553
UNROLL=4	40	49	7100	10335	6421
UNROLL=8	80	97	13528	20091	3385

HPCA'19

Instruction	# of Calls	Speed per Call (cycles)	(μ sec)
NTT	14	87,582	73.0
Inverse-NTT	8	102,043	85.0
Coeff. wise Multiplication	20	15,662	13.1
Coeff. wise Addition	26	16,292	13.6
Memory Rearrange	22	25,006	20.8
$\text{Lift}_{q\to Q}$ (2 cores)	4	99,137	82.6
$\text{Scale}_{Q\to q}$ (2 cores)	3	99,274	82.7

TABLE 3
Cost of a Single Coprocessor (Having Two Instances of Each Computation Core), Resource Overhead of Each Core, and the Estimated Execution Time of Homomorphic Multiplication Different Number of Cores

	Resources					Mult. Time in % w/ Core Count		
	LUT	REG	BRAM	URAM	DSP	1	2	4
Copro.	57,877	25,648	249	56	208			
NTT	483	80	69	0	0	81	100	137
Lift	13,807	6,157	0	0	76	94	100	112
RPAU	12,274	6,528	0	0	28	87	100	125

Lift, Scale, keyswitching module

- We reference HEAWS's lift, scale and implement them using HLS

N=8192	BRAM_18K	DSP48E	FF	LUT	latency (cycle)
lift	32	264	96716	102165	32836
scale	48	208	62982	58673	32804

Lift, Scale, keyswitching module

- We reference HEAWS's lift, scale and implement them using HLS

N=8192	BRAM_18K	DSP48E	FF	LUT	latency (cycle)
lift	32	264	96716	102165	32836
scale	48	208	62982	58673	32804

HPCA'19

Instruction	# of Calls	Speed per Call (cycles)	(μ sec)
NTT	14	87,582	73.0
Inverse-NTT	8	102,043	85.0
Coeff. wise Multiplication	20	15,662	13.1
Coeff. wise Addition	26	16,292	13.6
Memory Rearrange	22	[illegible]	20.8
$\text{Lift}_{q\to Q}$ (2 cores)	4	99,137	82.6
$\text{Scale}_{Q\to q}$ (2 cores)	3	99,274	82.7

TABLE 3
Cost of a Single Coprocessor (Having Two Instances of Each Computation Core), Resource Overhead of Each Core, and the Estimated Execution Time of Homomorphic Multiplication Different Number of Cores

	Resources					Mult. Time in % w/ Core Count		
	LUT	REG	BRAM	URAM	DSP	1	2	4
Copro.	57,877	25,648	249	56	208			
[illegible]	483	80	69	0	0	81	100	137
Lift	13,807	6,157	0	0	76	94	100	112
RPAU	12,274	6,528	0	0	28	87	100	125

Lift, Scale, keyswitching module

N=8192	BRAM_18K	DSP48E	FF	LUT	latency (cycle)
ntt	560 (~20%)	25	16880 (~1%)	14230 (~2%)	313411
lift	32 (~1%)	264 (~4%)	96716 (~6%)	102165 (~11%)	32836
scale	48 (~1%)	208 (~3%)	62982 (~4%)	58673 (~6%)	32804
apply_galois	2	2	1022	1290	65552
mulConst	2	4	2268	2694	65539
mulWise	2	4	2397	2741	131095
add	2	0	1286	1467	131088
mulInv	2	2	2218	2635	65559
mov	2	0	752	931	65550
Available	2688	5952	1743360	871680	

HE_primitive

- With all kernels available, we can expose

  HE primitives through series of kernel executions

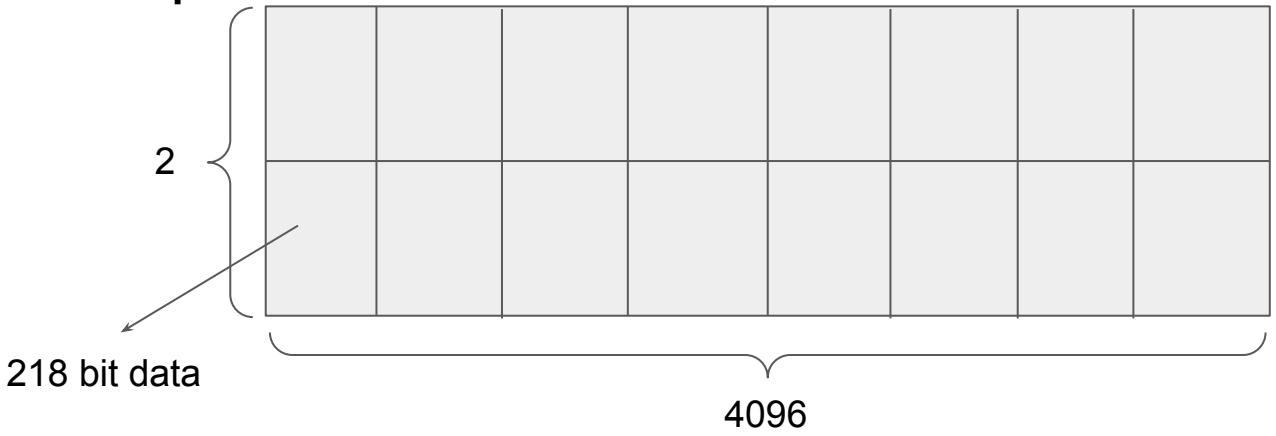
```
void rotate_column(cl_mem in1[4][2],
                   int r, // +- 1, 2, 4, ...
                   cl_mem out[4][2]) {
    vector<int> rot;
    vector<int> which_key;
    r = -r;
    while (r != 0) {
        double temp = log2(abs(r));
        int t = round(temp);
        int num = 1<<t;
        if (r < 0) {
            num = -num;
            t   = t + ((int)(log2(N))-1);
        }
        r -= num;
        if (num % (N/2) == 0)
            continue;
        num = (num + N/2) % (N/2);
```

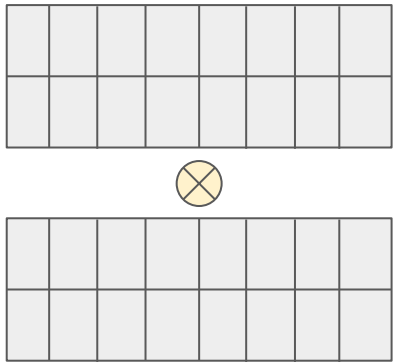
```
void HMul(cl_mem in1[4][2],
          cl_mem in2[4][2],
          cl_mem out[4][2])
{
    for (int i = 0; i < 4; i++) {
        do_k_NTT(in1[i][0], in1[i][0], INVERSE, FIRST, 0);
        do_k_NTT(in2[i][0], in2[i][0], INVERSE, FIRST, 0);
        do_k_NTT(in1[i][1], in1[i][1], INVERSE, FIRST, 0);
        do_k_NTT(in2[i][1], in2[i][1], INVERSE, FIRST, 0);

        do_k_lift(in1[i][0], temp1[0], LIFT_TO_P);
        do_k_lift(in2[i][0], temp2[0], LIFT_TO_P);
        do_k_lift(in1[i][1], temp1[1], LIFT_TO_P);
        do_k_lift(in2[i][1], temp2[1], LIFT_TO_P);

        do_k_NTT(in1[i][0], in1[i][0], FORWARD, FIRST, 0);
        do_k_NTT(in2[i][0], in2[i][0], FORWARD, FIRST, 0);
        do_k_NTT(in1[i][1], in1[i][1], FORWARD, FIRST, 0);
        do_k_NTT(in2[i][1], in2[i][1], FORWARD, FIRST, 0);
        do_k_NTT(temp1[0], temp1[0], FORWARD, SECOND, 0);
        do_k_NTT(temp2[0], temp2[0], FORWARD, SECOND, 0);
        do_k_NTT(temp1[1], temp1[1], FORWARD, SECOND, 0);
        do_k_NTT(temp2[1], temp2[1], FORWARD, SECOND, 0);

        // Tensoring
        do_k_mulWise(in1[i][0], in2[i][0], temp3[0], 0);
        do_k_mulWise(in1[i][1], in2[i][0], temp3[1], 0);
        do_k_mulWise(in1[i][0], in2[i][1], temp3[3], 0);
        do_k_mulWise(in1[i][1], in2[i][1], temp3[2], 0);
```

Supported HE operations

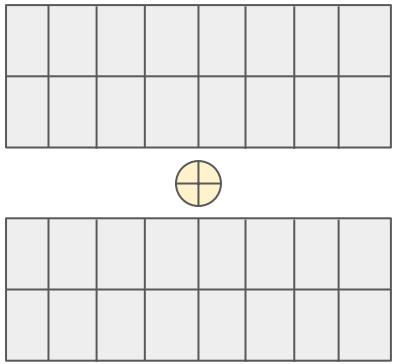
- N=8192

  q=218bit

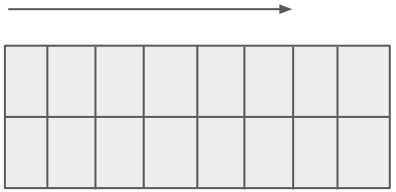




Element-wise mul



Element-wise add



rotate column

Inference using HE primitives

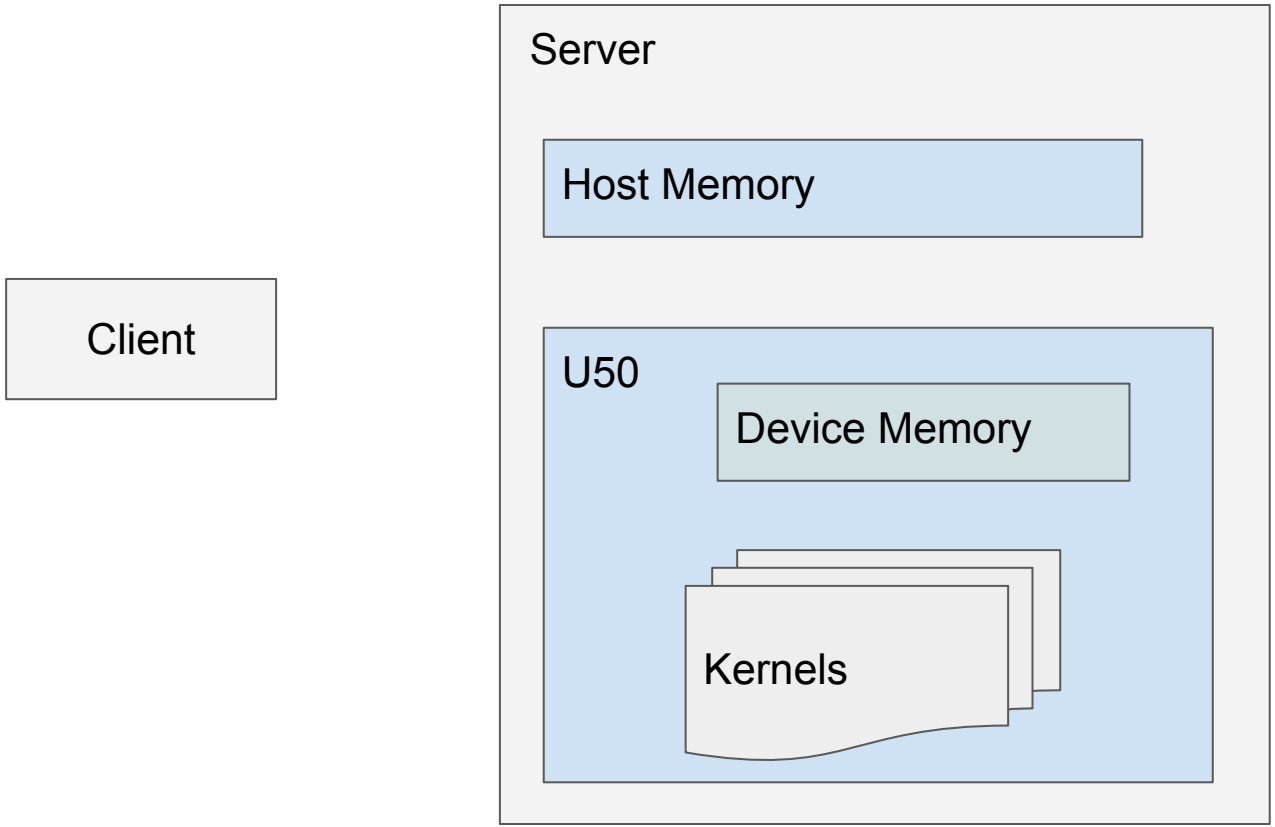
```
// inference.cpp
void inference()
{
    CNN     (GM_ct, GM_ct[0]); clEnqueueBarrier(Command_Queue);
    Square  (GM_ct);           clEnqueueBarrier(Command_Queue);
    Dense100(GM_ct);           clEnqueueBarrier(Command_Queue);
    Square  (GM_ct);           clEnqueueBarrier(Command_Queue);
    Dense10 (GM_ct);           clEnqueueBarrier(Command_Queue);
}
```

```
void CNN(cl_mem in1[25][4][2], cl_mem out[4][2]) {
    for (int c = 0; c < 5; c++) {
        for (int i = 0; i < 25; i++) {
            MulConst(in1[i], temp_result[c][i], cnn_weight[c][i]);
        }
    }
    for (int c = 0; c < 5; c++)
        for (int i = 1; i < 25; i++)
            HAdd(temp_result[c][0], temp_result[c][i], temp_result[c][0]);
    for (int i = 1; i < 5; i++)
        rotate_column(temp_result[i][0], 169*i, temp_result[i][0]);
    for (int i = 1; i < 5; i++)
        HAdd(temp_result[0][0], temp_result[i][0], temp_result[0][0]);
    for (int i = 0; i < 4; ++i) {
      for (int j = 0; j < 2; ++j){
        do_k_mov(temp_result[0][0][i][j], out[i][j]);
      }
    }
    do_k_mov(temp_result[0][0], out[0]);
    do_k_mov(temp_result[0][1], out[1]);
}
```

Outline

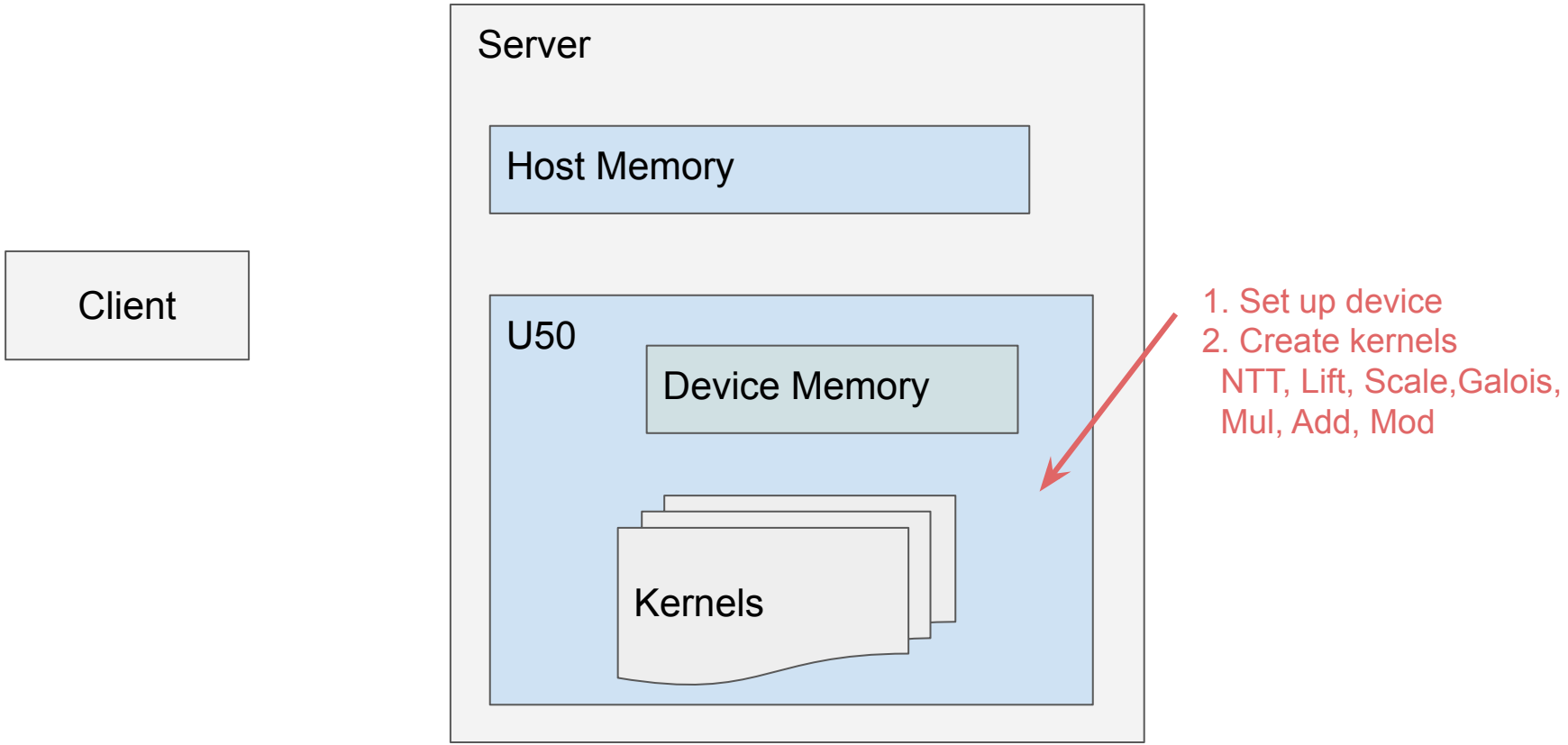
- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

System Architecture

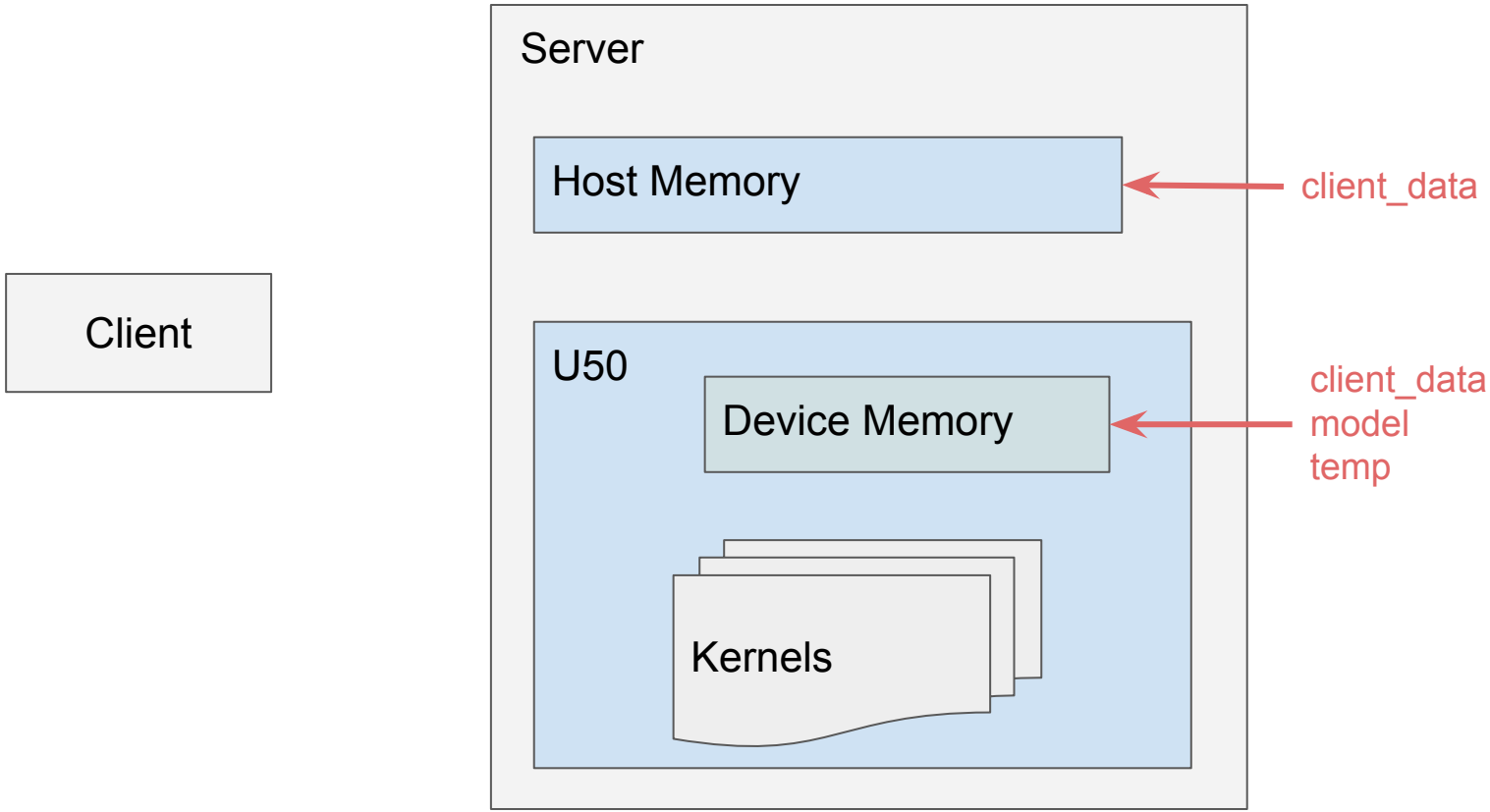


Example:
MNIST inference
Low Latency Privacy Preserving Inference

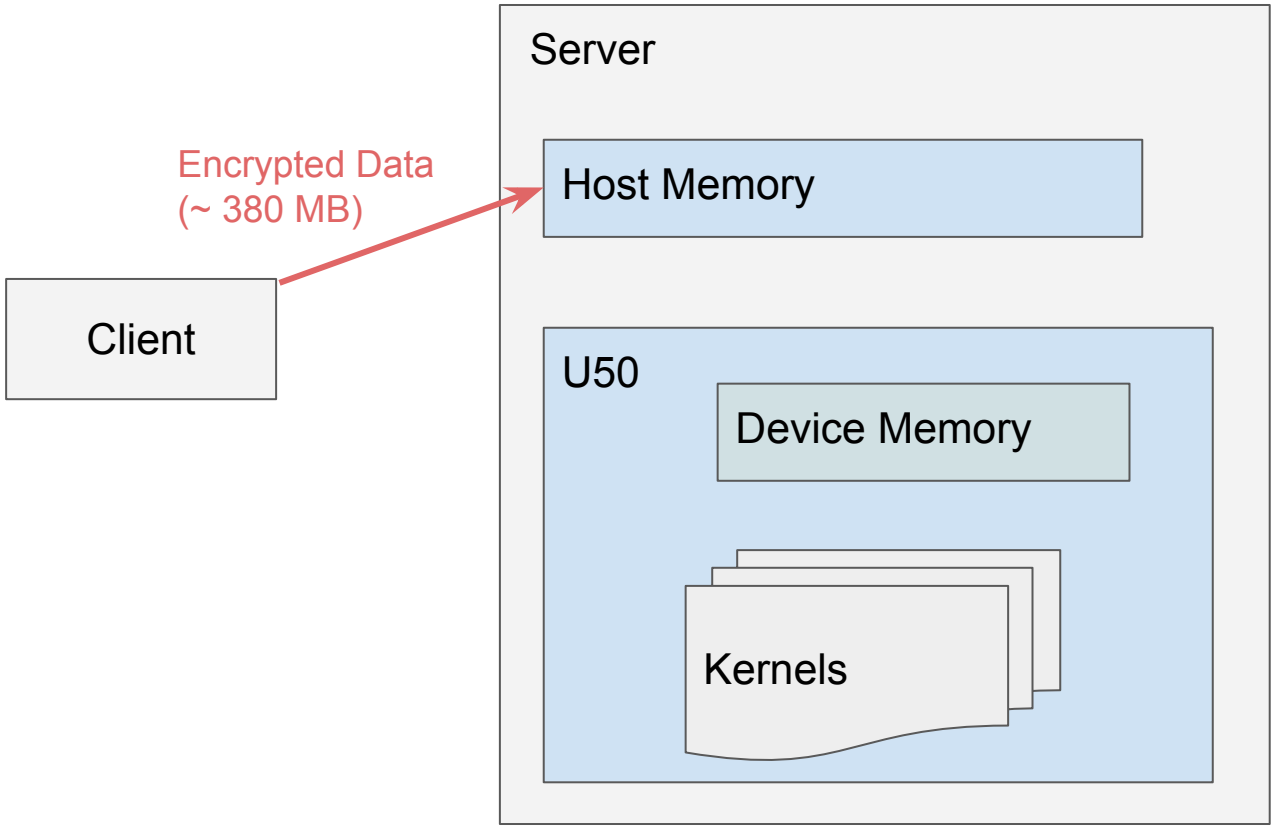
WorkFlow - Setup Device and Create Kernels



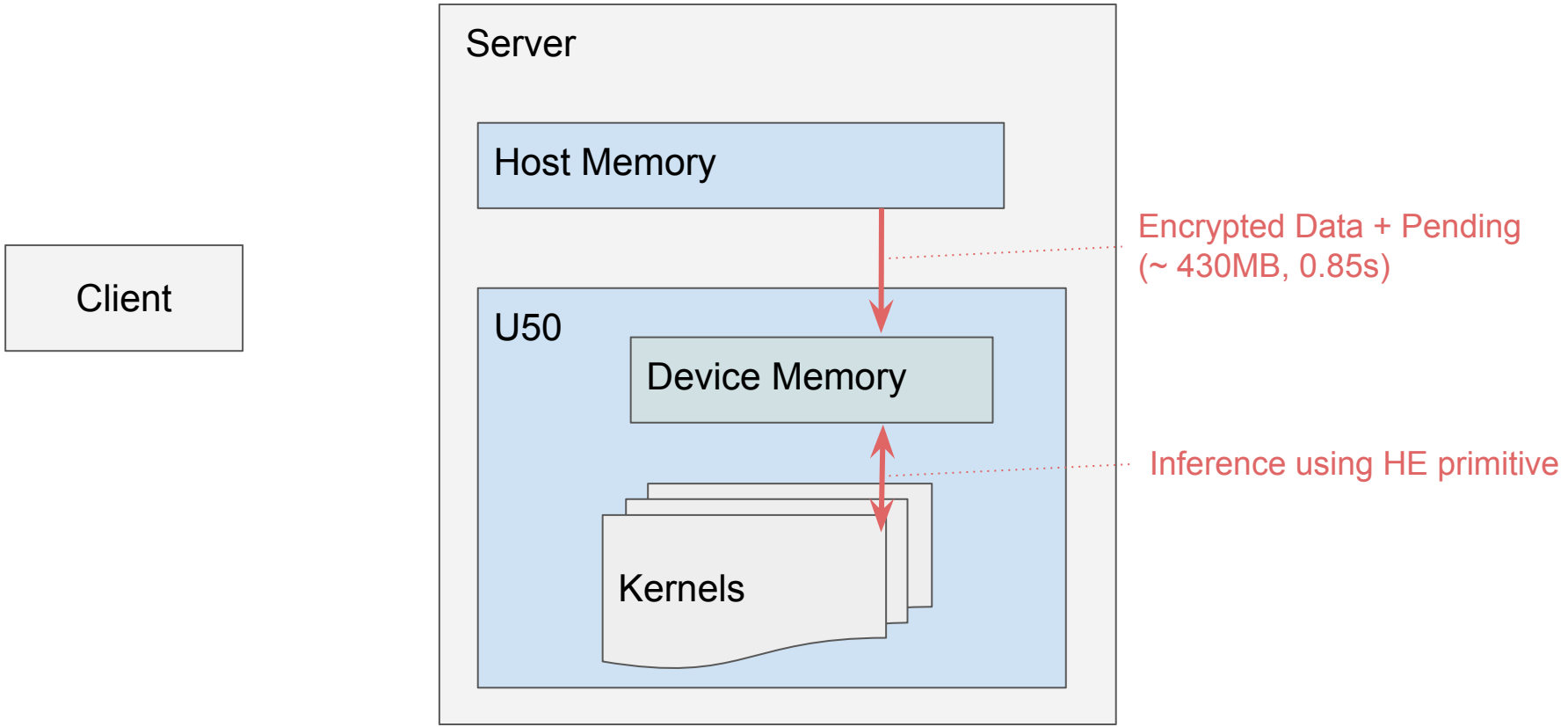
WorkFlow - Create Host Buffer and Device Buffer



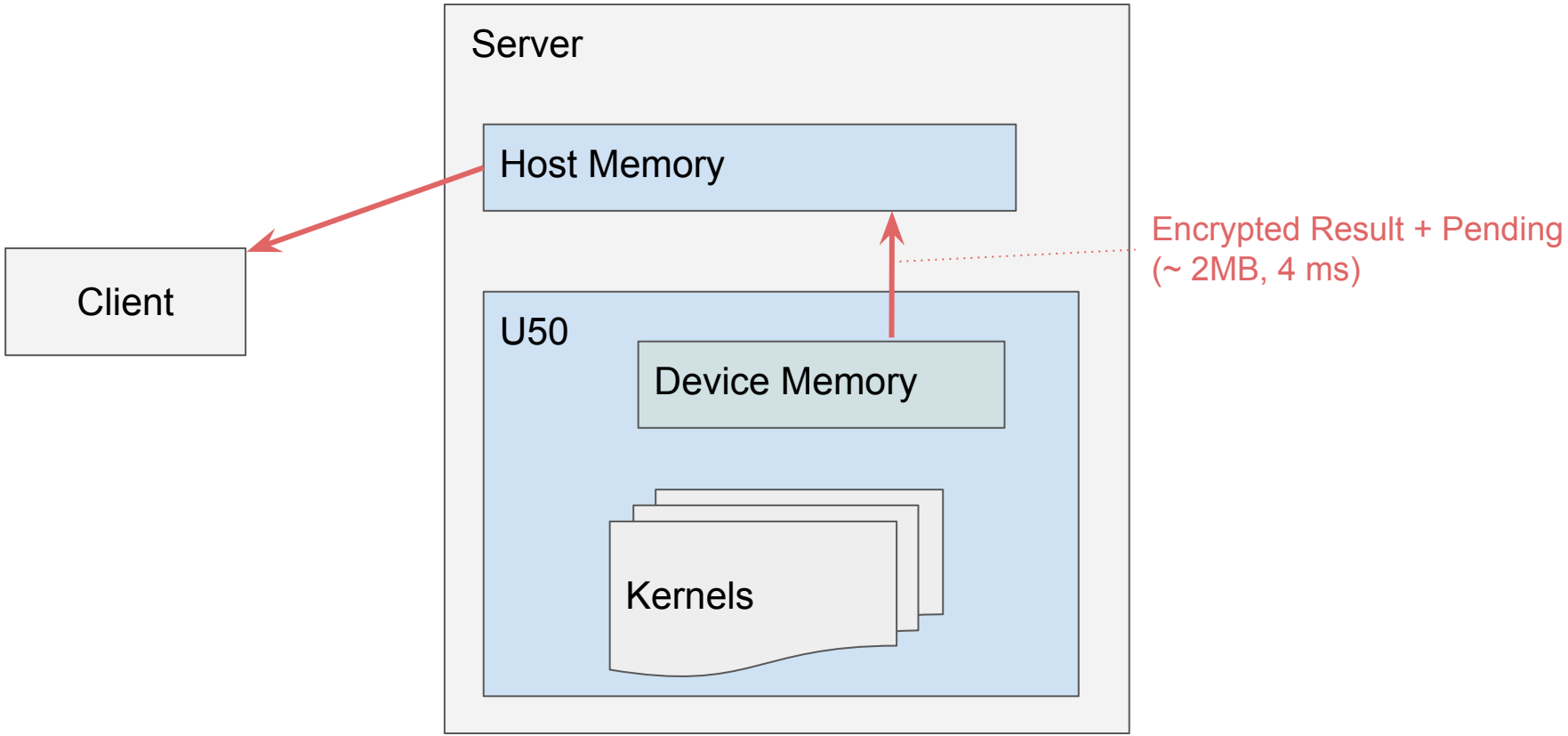
WorkFlow - Client Sends Data to Server



WorkFlow - Data to Device and Start Inference



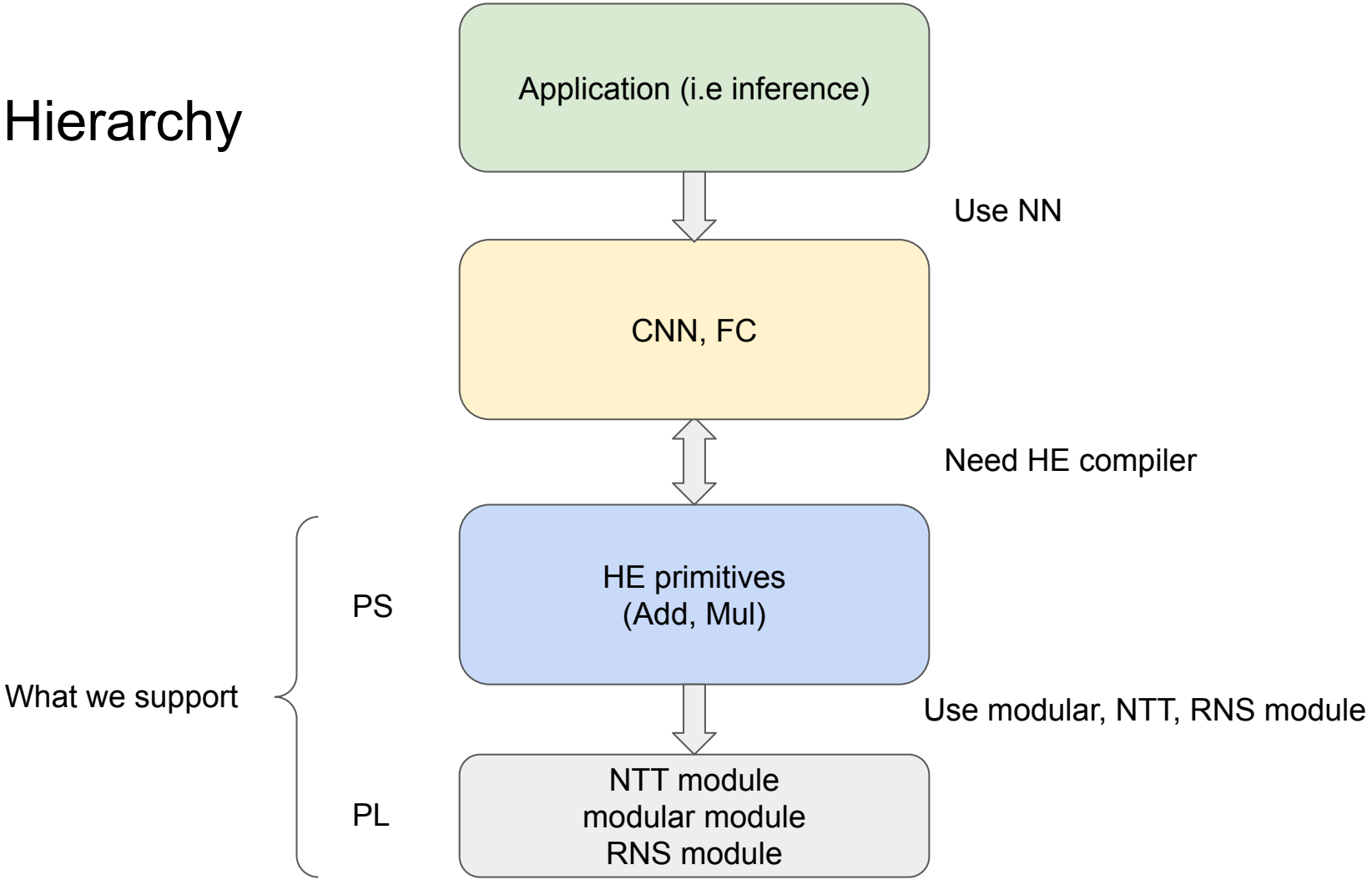
WorkFlow - Send Result Back to Client



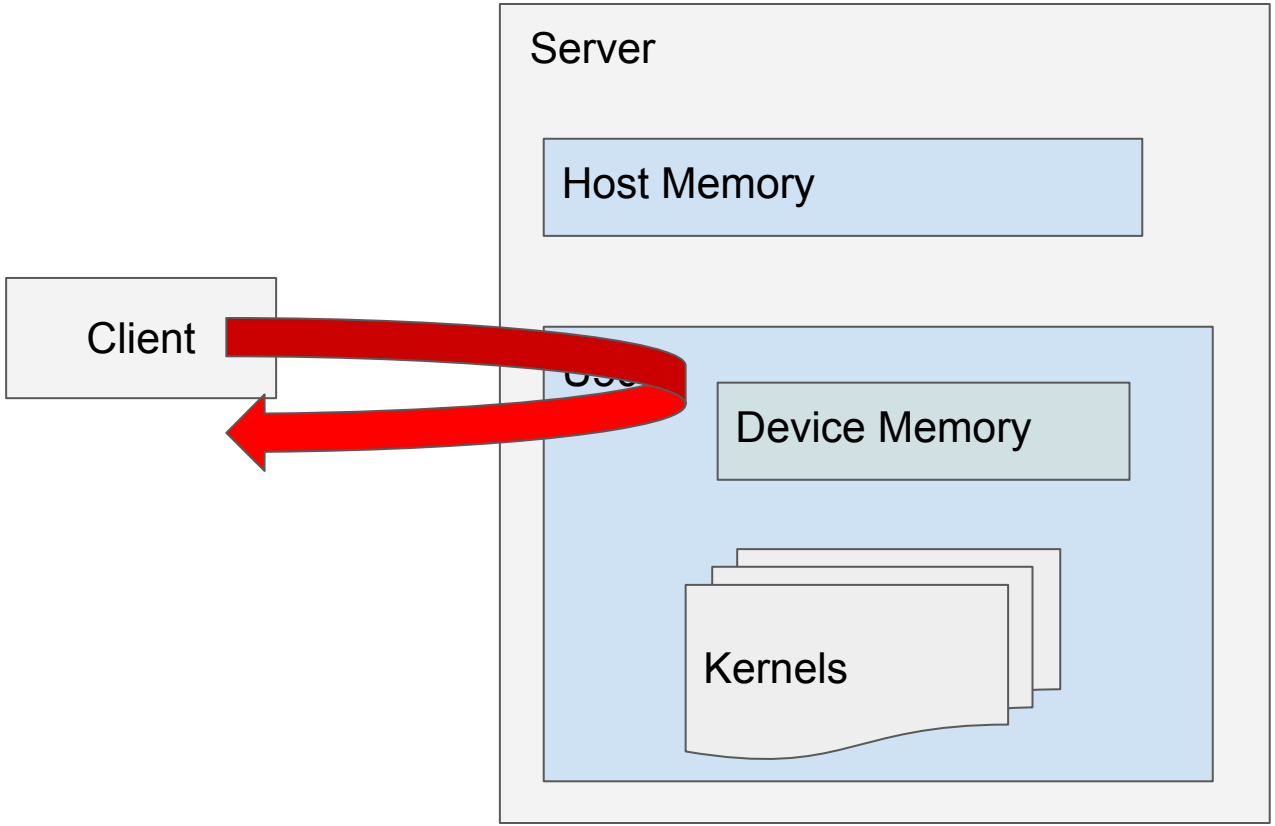
Outline

- Introduction
- Reduction
- NTT, RNS
- System architecture
- Summary

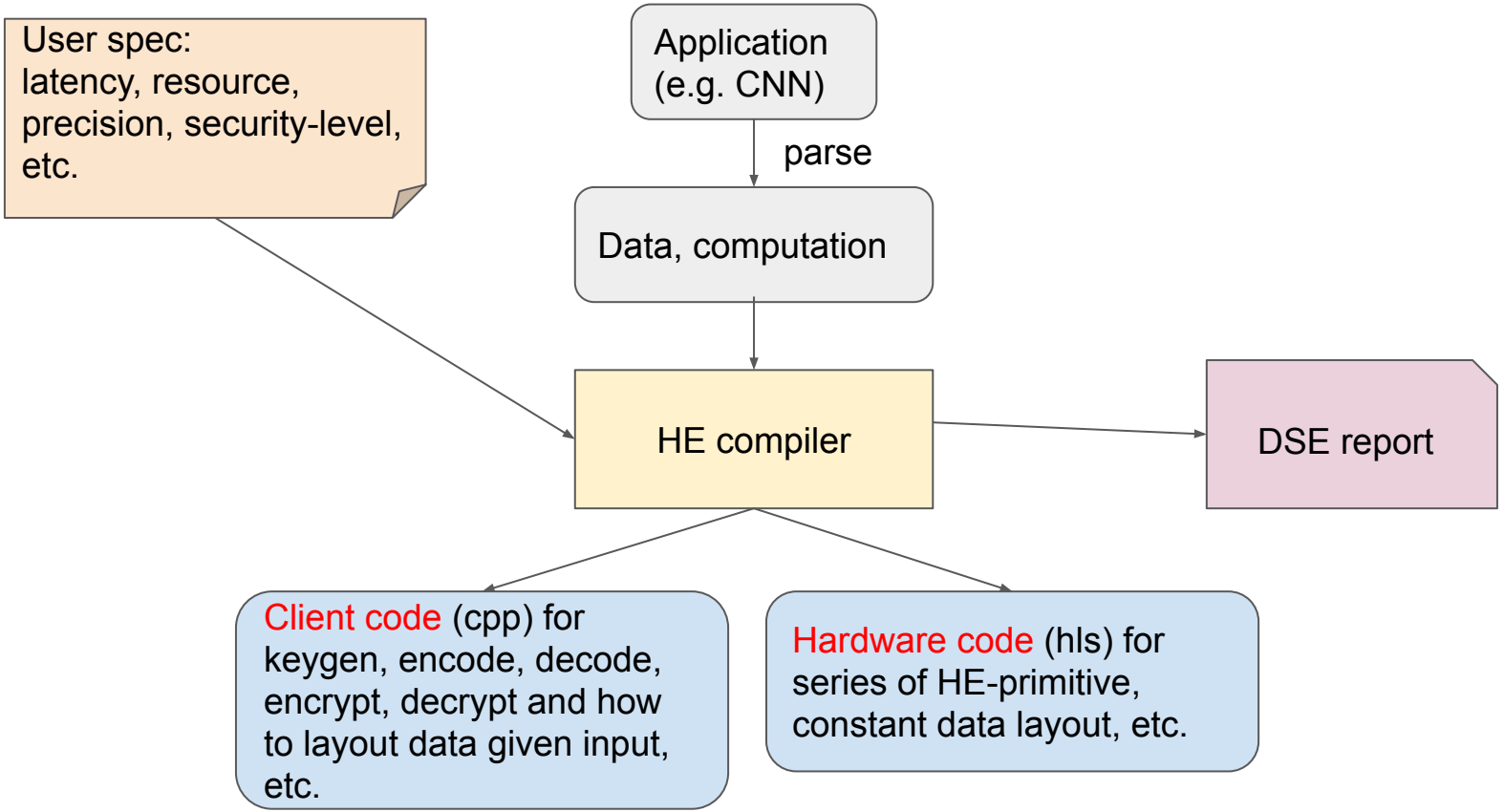
Hierarchy


Application (i.e inference)
Use NN
CNN, FC
Need HE compiler
HE primitives
(Add, Mul)
Use modular, NTT, RNS module
NTT module
modular module
RNS module
What we support
PS
PL

Client-Server interface



Future work: End-to-end application specific optimization



Reference

- **HEAX: An Architecture for Computing on Encrypted Data**
  - **ASPLOS'20, M. Sadegh Riazi, Kim Laine, Blake Pelton, Wei Dai**
- **HEAWS: An Accelerator for Homomorphic Encryption on the Amazon AWS FPGA**
  - **IEEE Transactions on Computers 2020, Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede**
- **FPGA-based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data**
  - **HPCA'19, Sujoy Sinha Roy, Furkan Turan, Kimmo Jarvinen, Frederik Vercauteren and Ingrid Verbauwhede**
- **Cheetah: Optimizations and Methods for Privacy Preserving Inference via Homomorphic Encryption**
  - **HPCA'21, Brandon Reagen, Wooseok Choi, Yeongil Ko, Vincent Lee, Gu-Yeon Wei, Hsien-Hsin S. Lee, and David Brooks**
- **Simple Encrypted Arithmetic Library**
  - **Microsoft open source library**
- **Somewhat Practical Fully Homomorphic Encryption**
  - **Junfeng Fan and Frederik Vercauteren**
- **An Improved RNS Variant of the BFV Homomorphic Encryption Scheme**
  - **Shai Halevi, Yuriy Polyakov, and Victor Shoup**

Q & A