

Hardware-Software Co-Design for Brain-Computer Interfaces

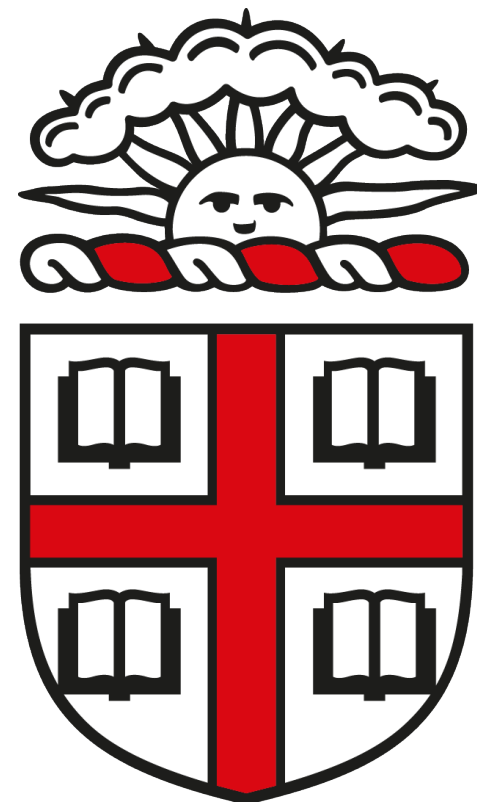
**Ioannis Karageorgos*, Karthik Sriram*, Ja'án Vesely*, Michael Wu
Marc Powell ,David Borton , Rajit Manohar , Abhishek Bhattacharjee**

ISCA 2020

Presented by Cheng Xuan, 10.12.2020



Yale University



BROWN



RUTGERS

Executive Summary

- **Problem:** Current **implantable BCIs** (as chips) are realized with **custom ASICs** (ASIC: Application-specific integrated circuit) and therefore treat only certain diseases or perform specific tasks in specific brain regions (**one architecture for one task**) => **low flexibility**
- **Goal:** Design a **general-purpose architecture** for implantable BCIs which **realizes multiple tasks** by one common architecture with **low power consumption** to satisfy safety constraint for implantable BCIs
- **Challenge:** Keep power consumption low while the circuit becomes complex
- **General Idea:** Using the principles of **hardware-software co-design** to **preprocess** the algorithms before implementing them into hardware
- **Key Mechanism:** **Refactor** the underlying algorithm of each task into **distinct pieces** that realize **different phases** of the algorithm and then implement each piece into distinct hardware block (**Processing Element**)
 - For each supported task: Configure all necessary processing elements into **pipeline** to execute it
- **Result:** Realizes an extensible and general-purpose hardware architecture for low-power implantable BCIs

Outline

- **Background**
- Problems
- Goals
- Key Mechanism
- Implementation
- Evaluation
- Strengths
- Weaknesses
- Discussion

Brain-Computer Interfaces (BCI)

- Direct **pathway** between the **brain** and an **external device**
- “**From brain to computer**” direction: Collects neuronal signal from neurons, digitize and then process
- “**From computer to brain**” direction: Change behavior of neurons by:
 1. Stimulating the neurons directly (rough way)
 2. Translating digital signals to a format which can be understood by neurons

=> Control neurons precisely : bottleneck of BCI

Brain-Computer Interfaces (BCI)



- Technology of BCI has gained more attention
- Used by over 160K patients worldwide
- Neuralink
 - Neurotechnology company
 - Founded by Elon Musk



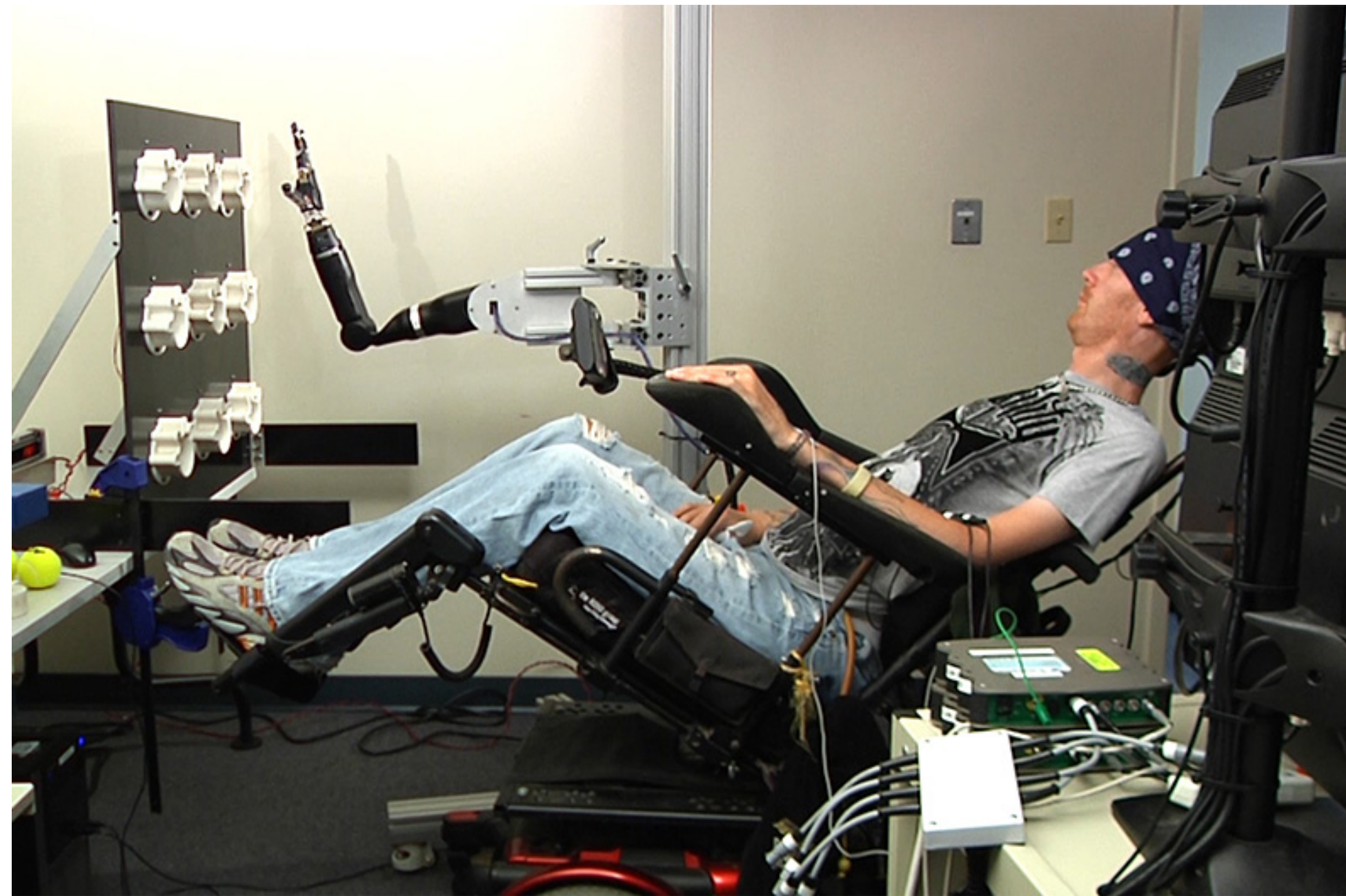
Applications of BCI

- **Treatment of neurological diseases**

E.g., epilepsy, Parkinson's disease, anxiety

- **Support research of brain functions**

- **Repair of perceptions (Cochlear implant)**



Realization of BCI

1. **Headsets or electrodes** placed on the scalp

- **Do not** require **surgical deployment** (safe and beneficial to commercialization)

2. **Implantable BCIs** (**as chips embedded in brain tissue**)

- Need surgical deployment but enable BCI to **record** from and **simulate** large number of neurons with **high signal fidelity, spatial resolution, and in real time**

Outline

- Background
- **Problems**
- Goals
- Key Mechanism
- Implementation
- Evaluation
- Strengths
- Weaknesses
- Discussion

Problem of BCI as headsets (Non-invasive)

BCI as headsets do not satisfy the **performance requirements** for forward-looking BCI applications

Because:

1. Collected signals are **noisy** and **low resolution**
2. Less ideal for **real time** processing of signals

Problem of BCI as headsets (Non-invasive)

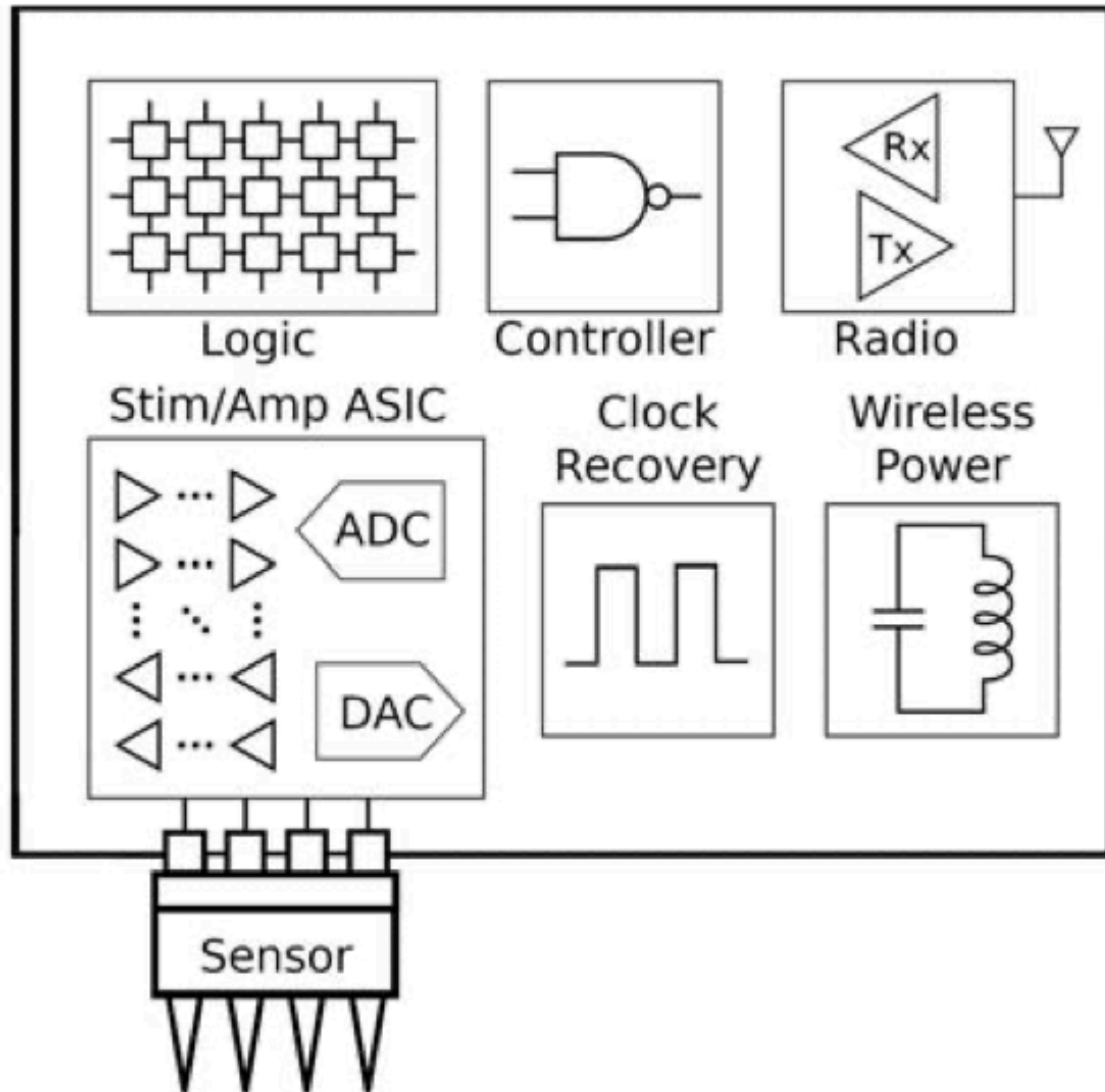
BCI as headsets do not satisfy the **performance requirements** for forward-looking BCI applications

Because:

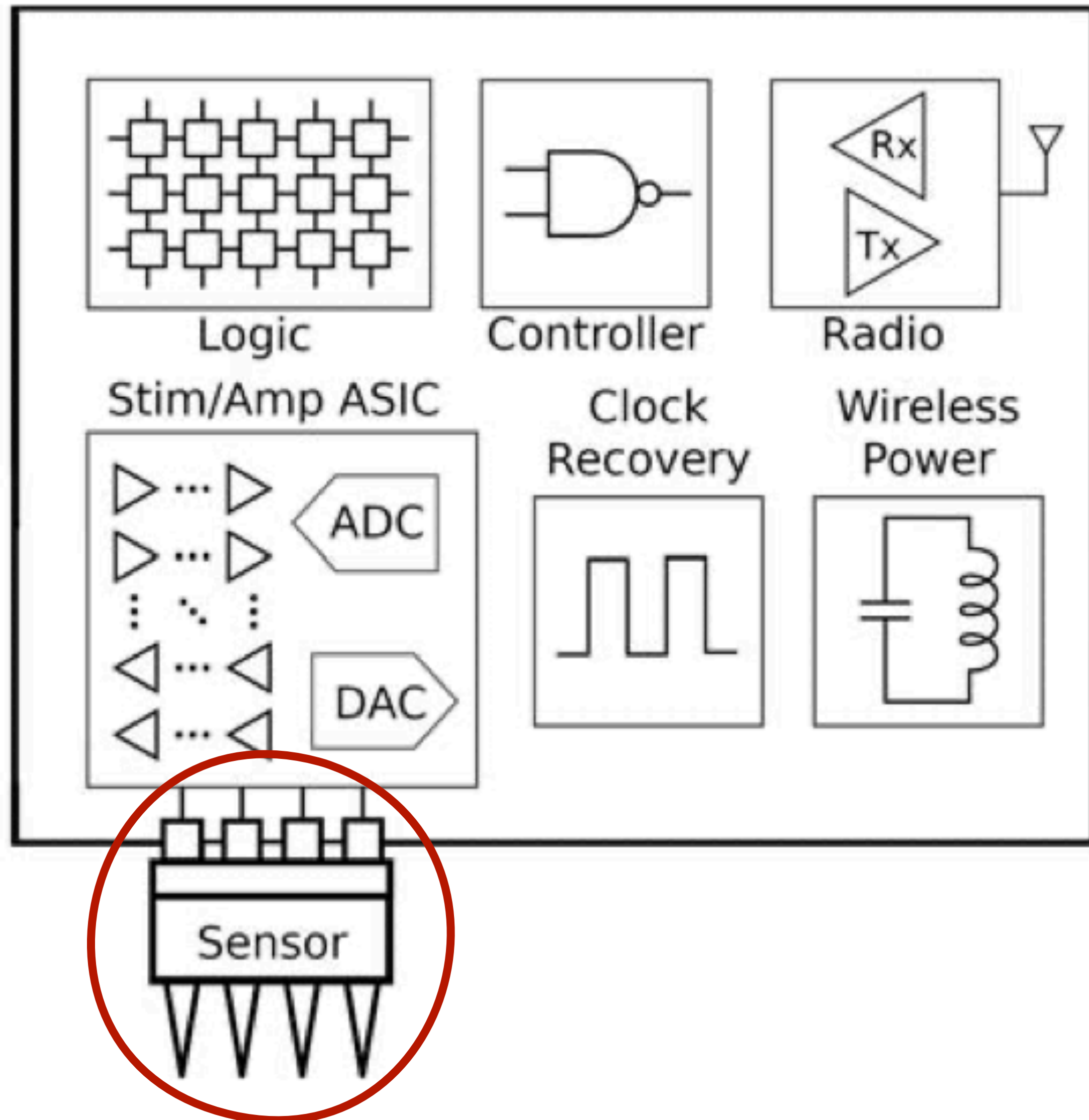
1. Collected signals are **noisy** and **low resolution**
2. Less ideal for **real time** processing of signals

For this reason, this paper focuses on implantable BCI and aims to improve it

Basic structure of implantable BCI

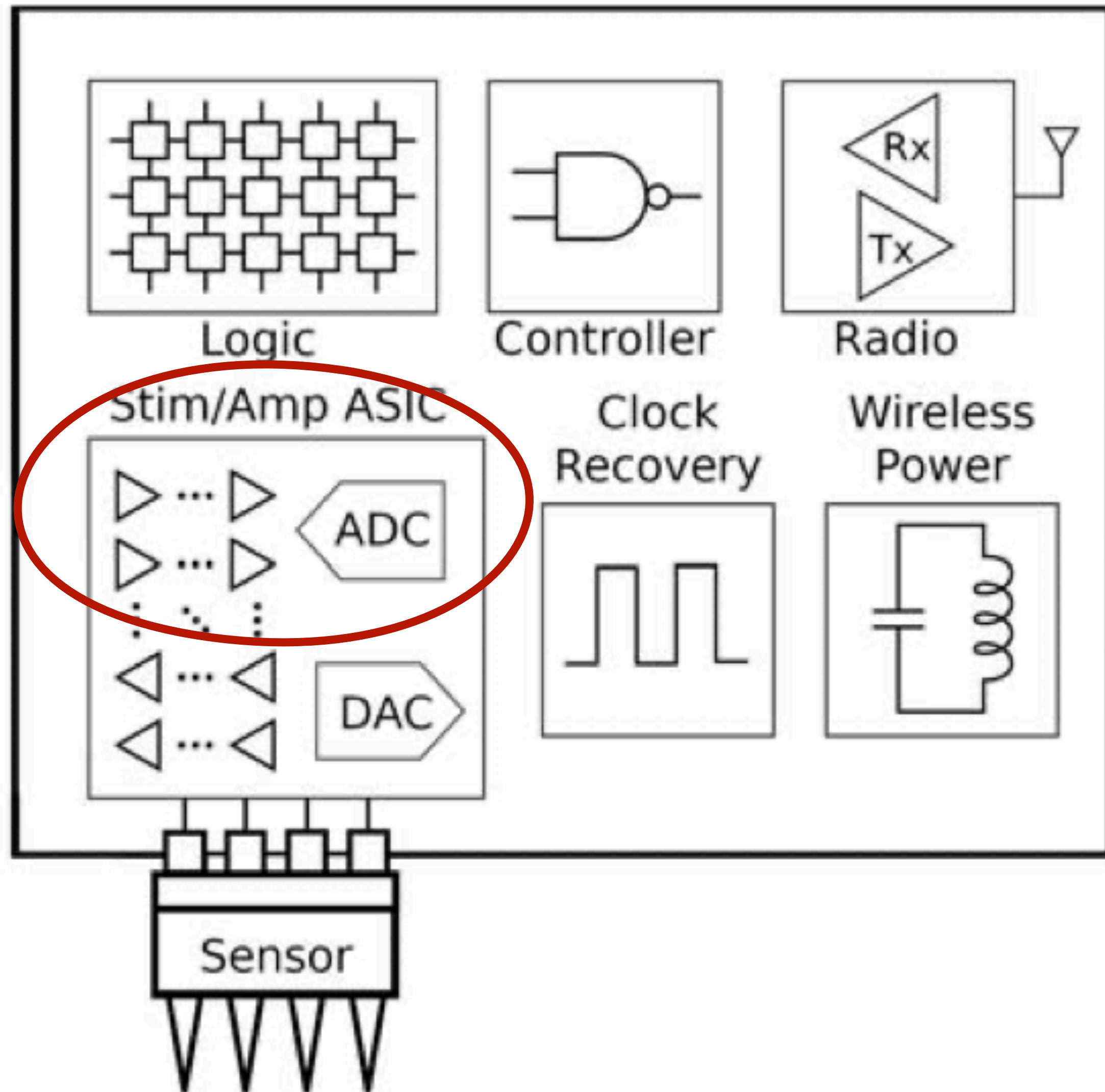


Basic structure of implantable BCI



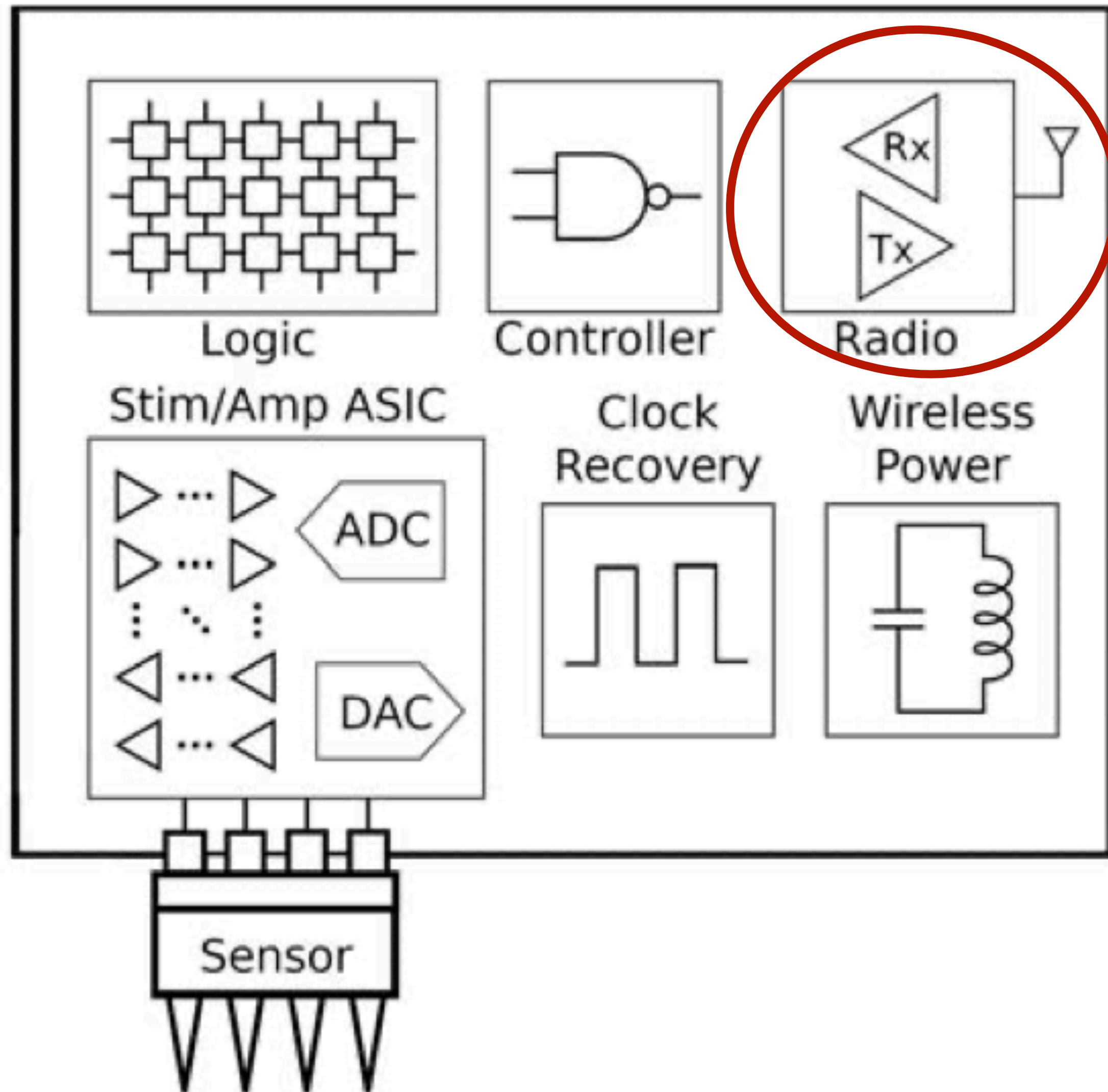
1. **Sensors:** record & stimulate neurons (read & write)
 - # Channels determine how many neurons can be processed

Basic structure of implantable BCI



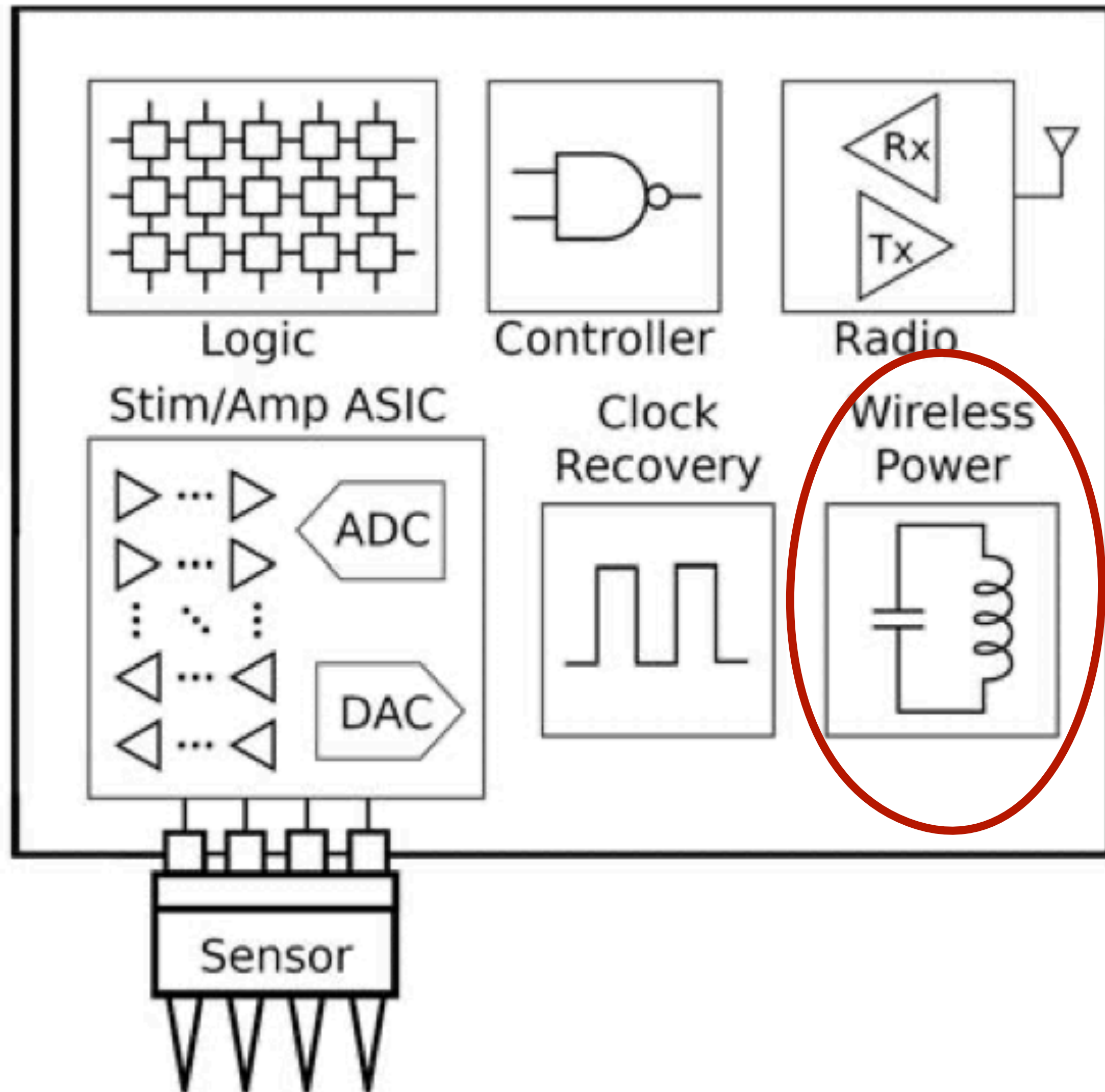
1. **Sensors:** **record** & **stimulate** neurons (read & write)
 - # Channels determine **how many** neurons can be processed
2. **Analog front-end:** **amplify** and **digitize** data from sensors
 - Via ADCs
 - Sample resolution determines quality of digitization
 - Sample frequency determines speed of digitization

Basic structure of implantable BCI



1. **Sensors:** **record** & **stimulate** neurons (read & write)
 - # Channels determine **how many** neurons can be processed
2. **Analog front-end:** **amplify** and **digitize** data from sensors
 - Via ADCs
 - Sample resolution determines quality of digitization
 - Sample frequency determines speed of digitization
3. **Communication links:** **change** data with the **outer world**
 - Via RF link

Basic structure of implantable BCI



1. **Sensors:** **record** & **stimulate** neurons (read & write)
 - # Channels determine **how many** neurons can be processed
2. **Analog front-end:** **amplify** and **digitize** data from sensors
 - Via ADCs
 - Sample resolution determines quality of digitization
 - Sample frequency determines speed of digitization
3. **Communication links:** **change** data with the **outer world**
 - Via RF link
4. **Power sources:**
 - Single-use **non-rechargeable** batteries
 - **Rechargeable** batteries by using **wireless powering**

Problems of implantable BCIS

1. **Safety constraint:** Implantable BCIs **must not** dissipate **more than 15-40mW** of power (FDA, FCC, and IEEE guidelines)
2. **Low flexibility:** (Caused by safety constraint)
 - BCIs targeting **large numbers of neurons**: To keep the circuit simple, they are realized with **custom ASICs** (treat only certain diseases or perform specific tasks in specific brain regions)

When we want to realize multiple tasks: Design ASIC for each task, and then:

 1. Combine them into one chip: called monolithic ASIC and exceed power constraint in many cases
 2. Put each ASIC into one chip : #tasks = #chips => impractical
 - The few **programmable** BCIs: Solves flexibility problem to a certain extent, but process only a **limited number of neurons** to meet the **low-power requirements** => impractical by real application

Problems of implantable BCIS

1. **Safety constraint:** Implantable BCIs **must not** dissipate **more than 15-40mW** of power (FDA, FCC, and IEEE guidelines)
2. **Low flexibility:** (Caused by safety constraint)

Flexibility and Performance: **One of them** as **victim** to satisfy power constraint

1. Combine them into one chip: called monolithic ASIC and exceed power constraint in many cases
 2. Put each ASIC into one chip : #tasks = #chips => impractical
- The few **programmable** BCIs: Solves flexibility problem to a certain extent, but process only a **limited number of neurons** to meet the **low-power requirements** => impractical by real application

Outline

- Background
- Problems
- **Goals**
- Key Mechanism
- Implementation
- Evaluation
- Strengths
- Weaknesses
- Discussion

Goals

Design a **general-purpose** architecture for implantable BCIs

- Realize multiple tasks by one common architecture
- Also target large number of neurons with high sampling frequency and resolution
- No need to design ASIC individually for each task

While:

Meeting power constraint of 15-40mW (adequately low-power)

- For safe and chronic implantation in the brain

Hardware Architecture for Low-power BCIs

(HALO)

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible
- HALO has **high** sample frequency & resolution

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible
- HALO has **high** sample frequency & resolution
- HALO meets safety constraint
 - **Safe** for chronic use
 - **While** achieve the outstanding performance

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible
- HALO has **high** sample frequency & resolution
- HALO meets safety constraint
 - **Safe** for chronic use
 - **While** achieve the outstanding performance

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible
- HALO has **high** sample frequency & resolution
- HALO meets safety constraint
 - **Safe** for chronic use
 - **While** achieve the outstanding performance

HALO & BCIS based on ASIC : Comparison

	Medtronic [10]	Neuropace [106]	Aziz [23]	Chen [37]	Kassiri [56]	Neuralink [74]	NURIP [84]	HALO
Tasks Supported								
Spike Detection	×	×	×	×	×	×	×	✓
Compression	×	×	✓	×	×	×	×	✓
Seizure Prediction	×	✓	×	✓	✓	×	✓	✓
Movement Intent	✓	×	×	×	×	×	×	✓
Encryption	×	×	×	×	×	×	×	✓
Technical Capabilities								
Programmable	✓	Limited	×	Limited	✓	×	Limited	✓
Read Channels	4	8	256	4	24	3072	32	96
Stimulation Channels	4	8	0	0	24	0	32	16
Sample Frequency (Hz)	250	250	5K	200	7.2K	18.6K	256	30K
Sample Resolution (bits)	10	10	8	10	-	10	16	16
Safety (<15mW)	✓	✓	✓	×	✓	×	✓	✓

- HALO can realize **all** listed tasks
 - Configurable and flexible
- HALO has **high** sample frequency & resolution
- HALO meets safety constraint
 - **Safe** for chronic use
 - **While** achieve the outstanding performance

HALO is comprehensive and outperforms existing BCIs

Outline

- Background
- Problems
- Goals
- **Key Mechanism**
- Implementation
- Evaluation
- Strengths
- Weaknesses
- Discussion

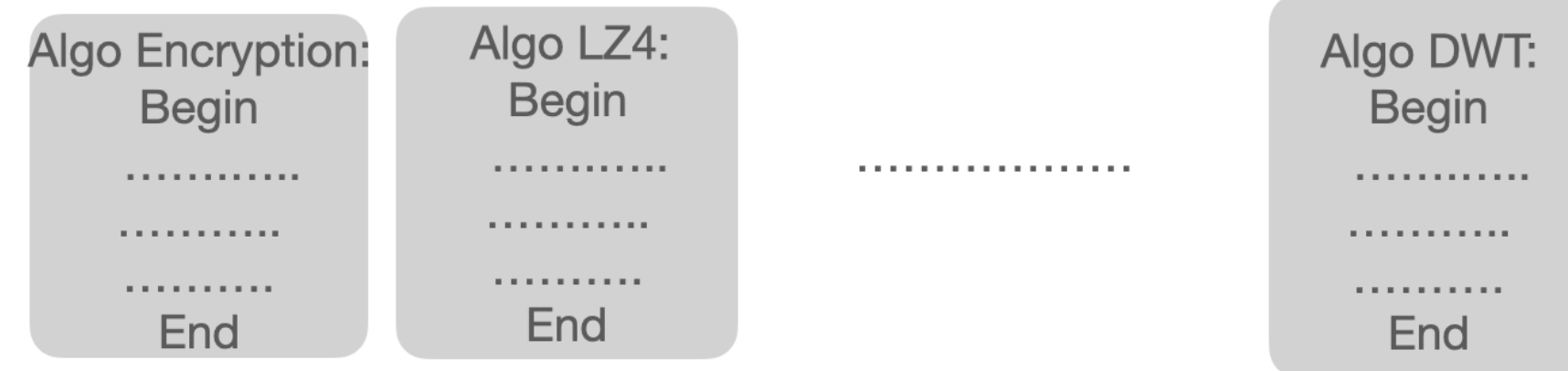
Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

- Determine a list of tasks that we want to support by HALO

Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

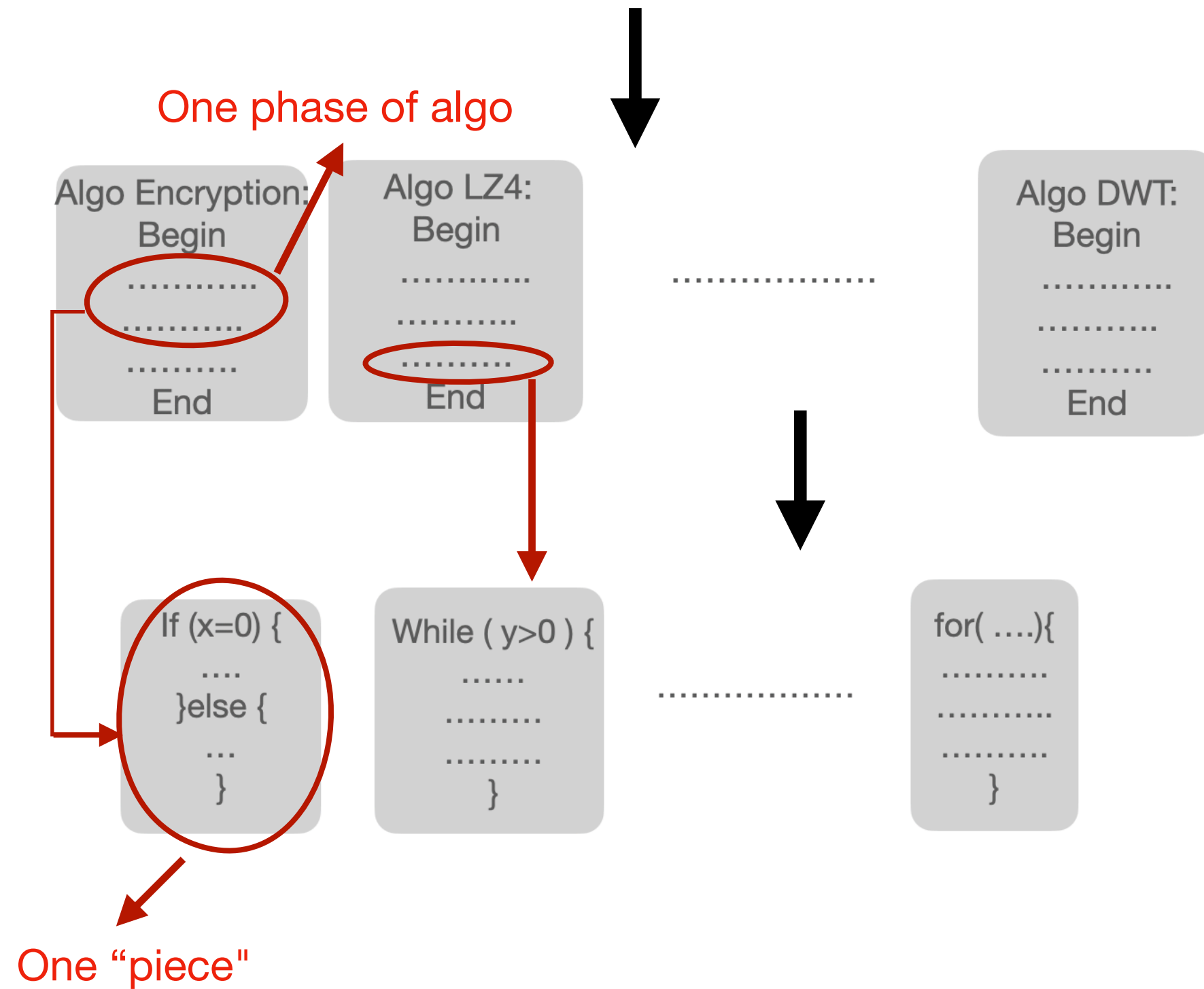


- Determine a list of tasks that we want to support by HALO
- For each task: Refactor the underlying algorithm of it into distinct pieces that realize different phases of the algorithm

Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

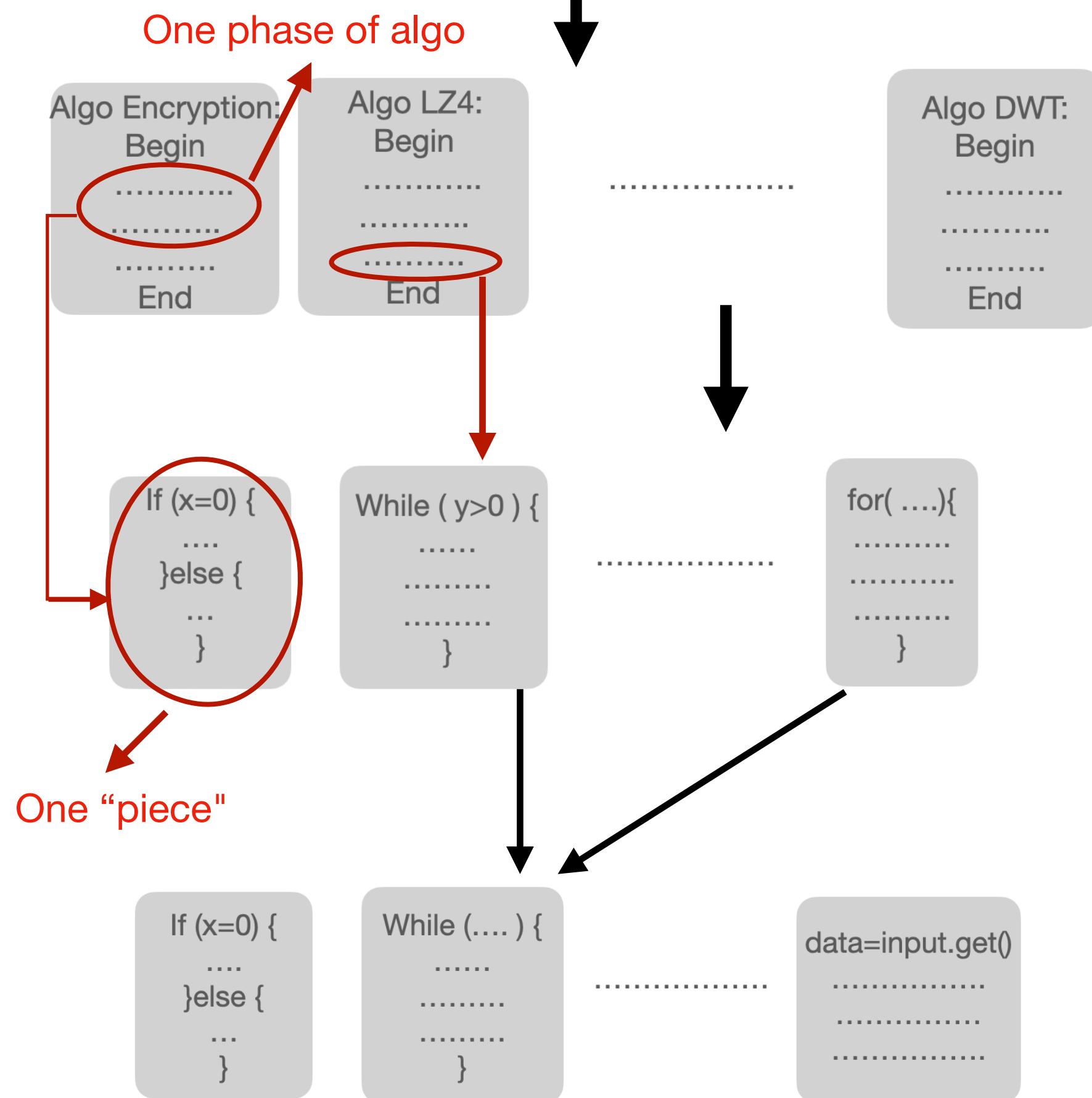
- Determine a list of tasks that we want to support by HALO
- For each task: Refactor the underlying algorithm of it into distinct pieces that realize different phases of the algorithm



Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

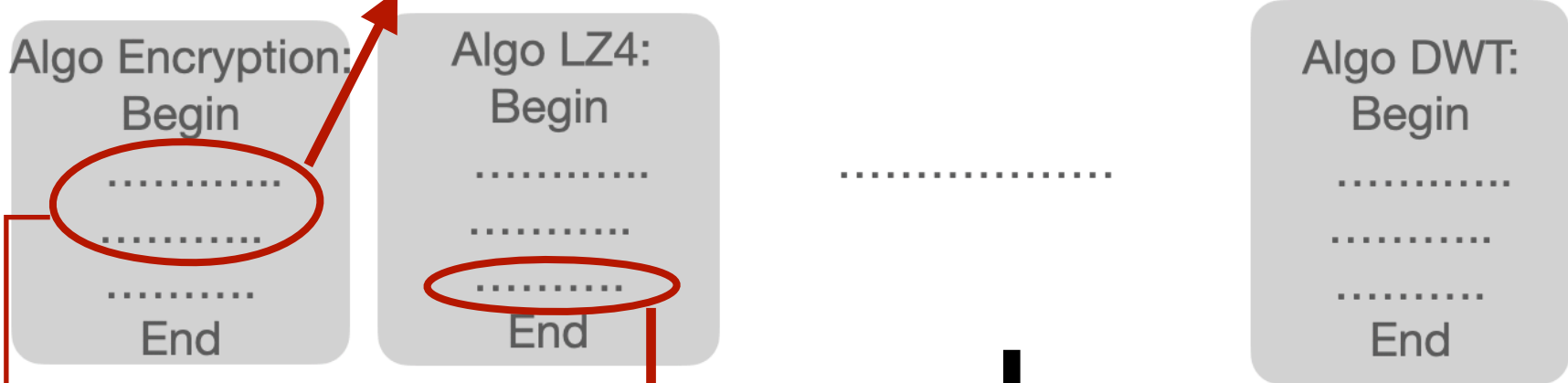
- Determine a list of tasks that we want to support by HALO
- For each task: Refactor the underlying algorithm of it into distinct pieces that realize different phases of the algorithm
- Identify shared pieces among algorithms



Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

One phase of algo



If (x=0) {
...
}else {
...
}

While (y>0) {
...
}

for(....){
...
}

One "piece"

If (x=0) {
...
}else {
...
}

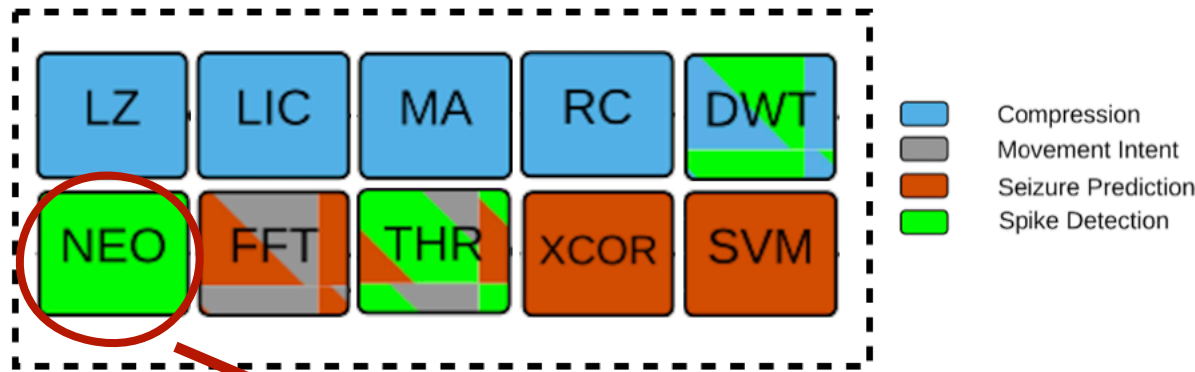
While (....) {
...
}

data=input.get()
...
...

- Determine a list of tasks that we want to support by HALO
- For each task: Refactor the underlying algorithm of it into distinct pieces that realize different phases of the algorithm
- Identify shared pieces among algorithms
- Implement these pieces into distinct hardware blocks (PEs)

Software Level

Hardware Level



One processing element

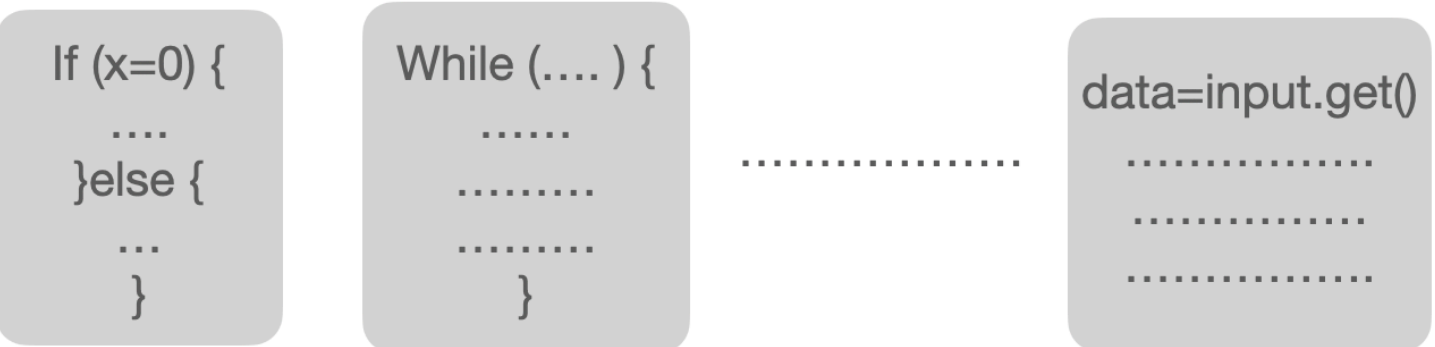
Key Mechanism

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

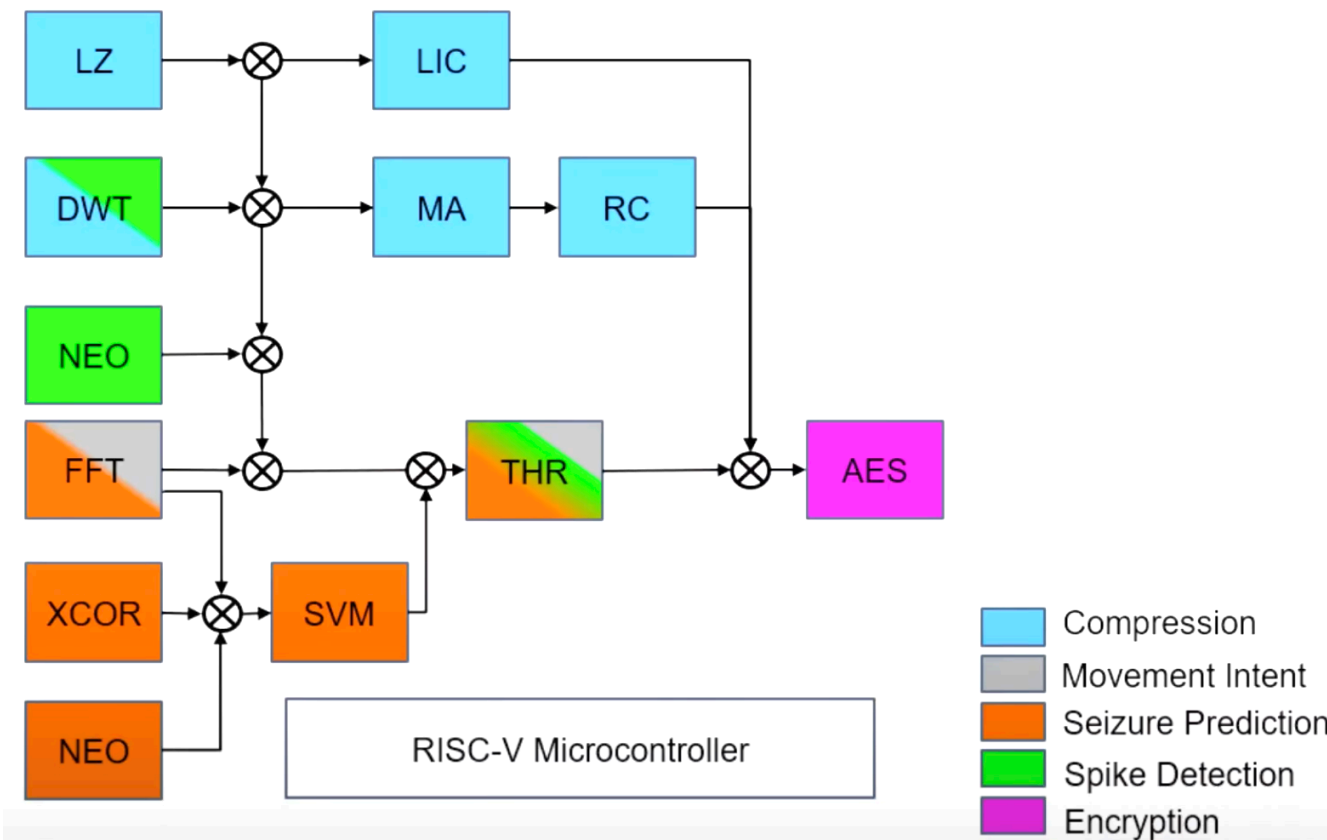
One phase of algo



One "piece"

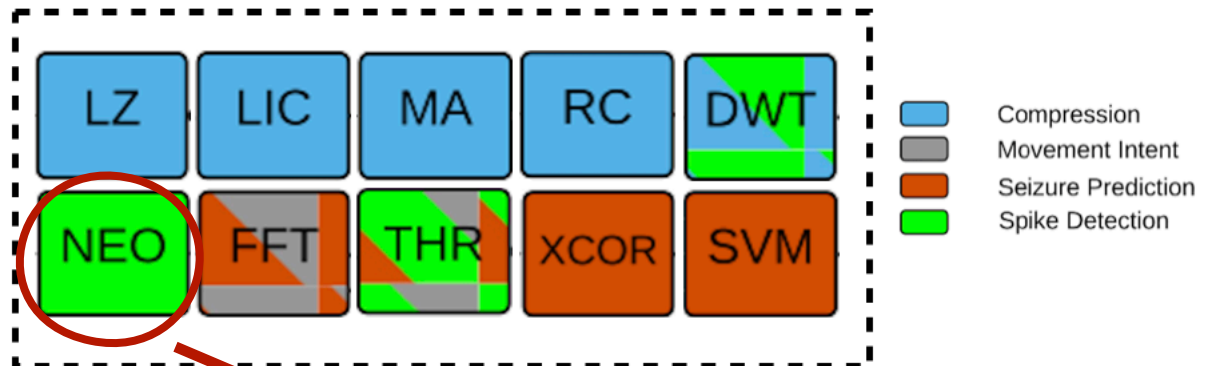


- Determine a list of tasks that we want to support by HALO
- For each task: Refactor the underlying algorithm of it into distinct pieces that realize different phases of the algorithm
- Identify shared pieces among algorithms
- Implement these pieces into distinct hardware blocks (PEs)
- Arrange all PEs together in a suitable way
- Now for each (supported) task: controller chooses all required PEs and configure them into pipeline to execute it



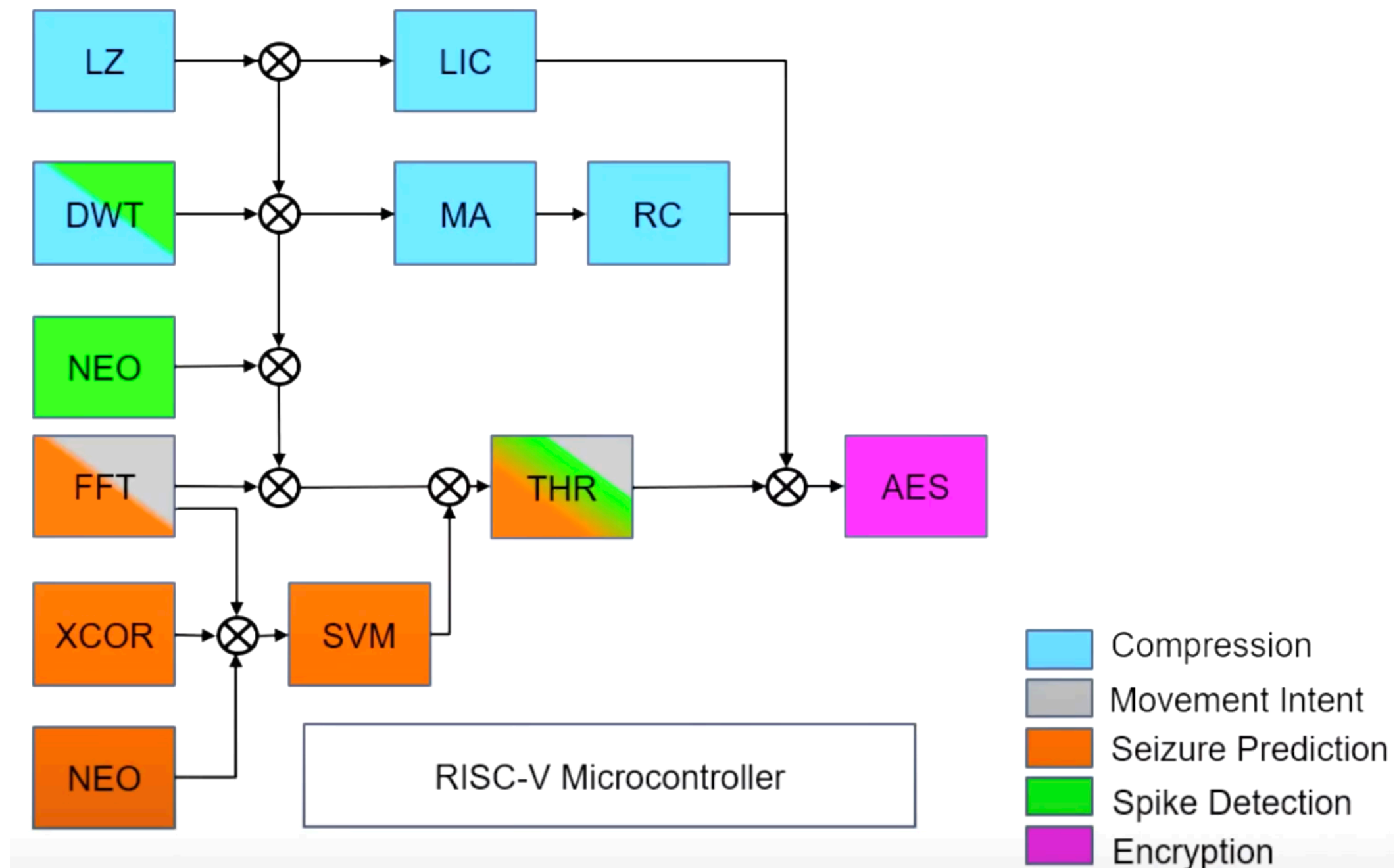
Software Level

Hardware Level



One processing element

General Idea

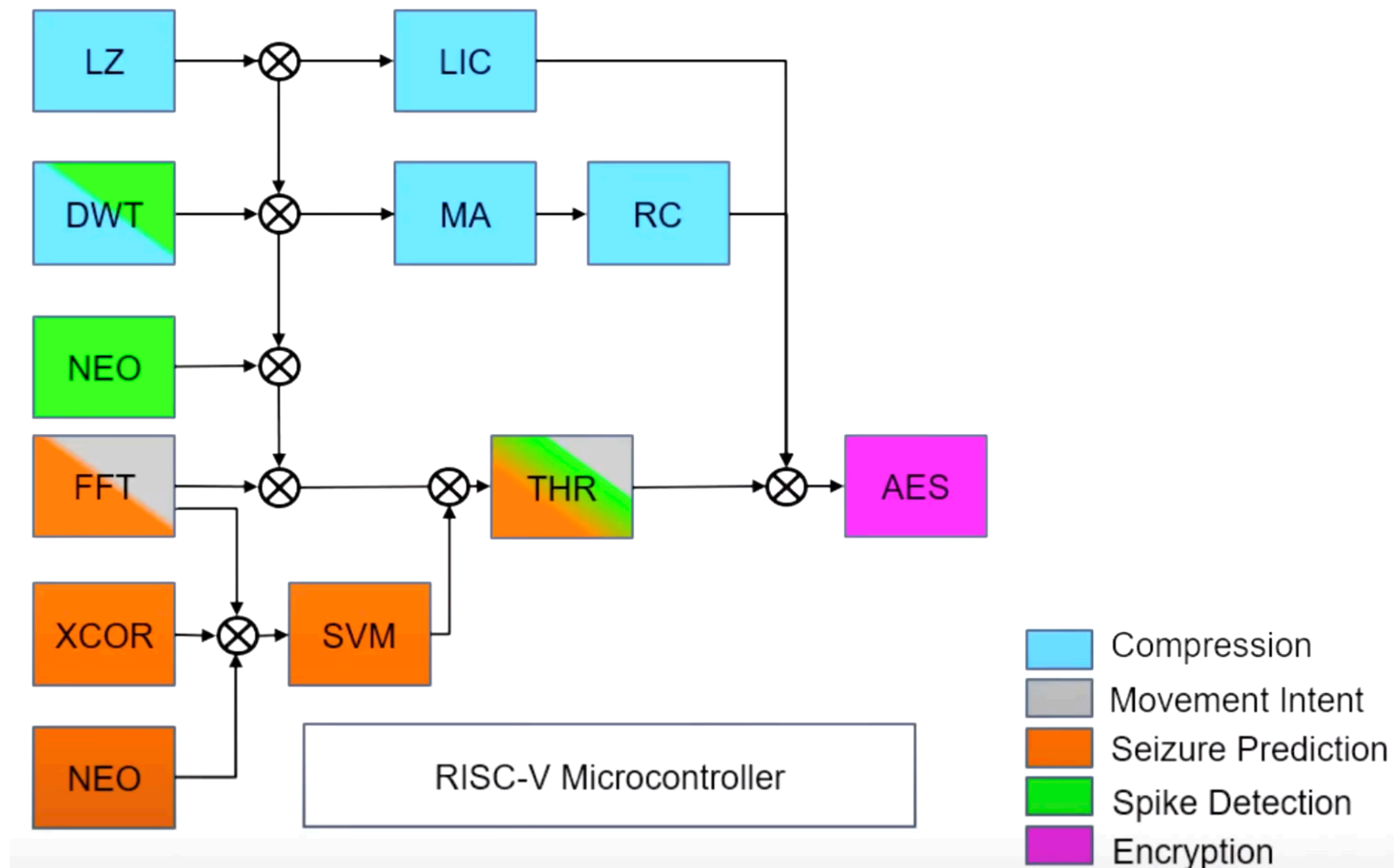


While executing LZMA:

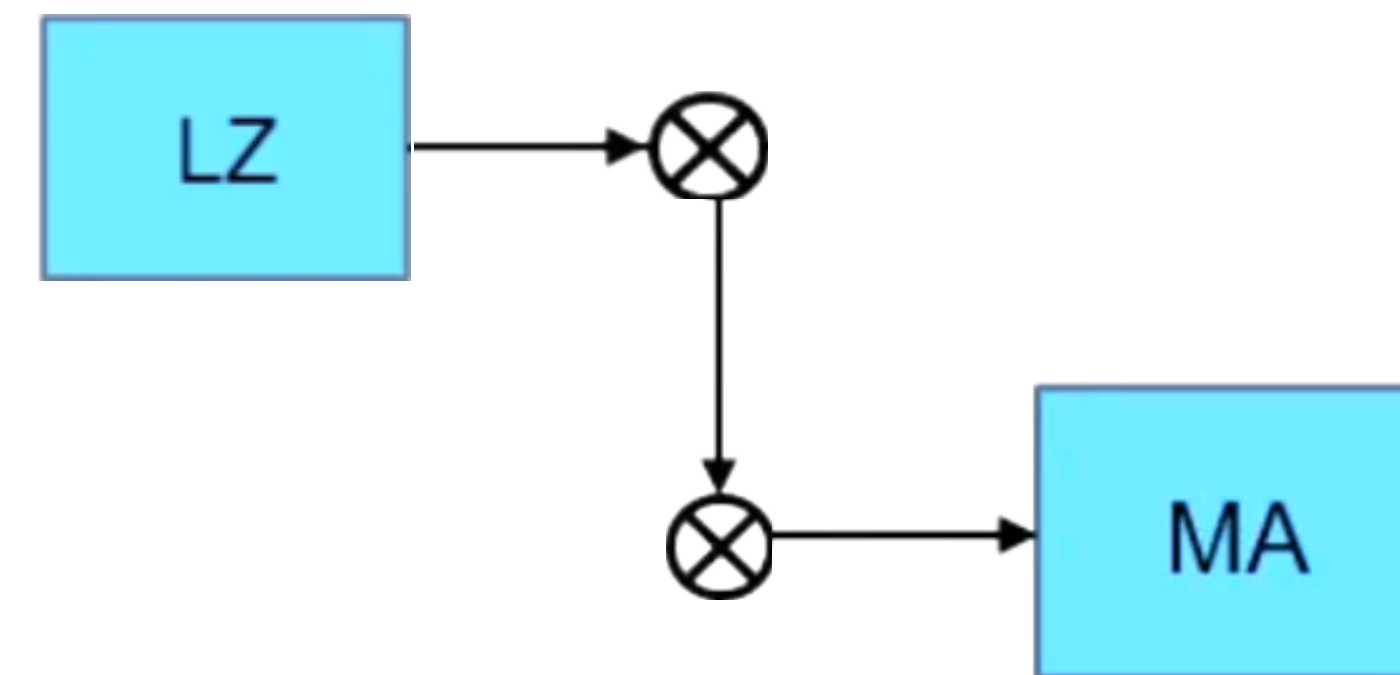


Routing Switch

General Idea

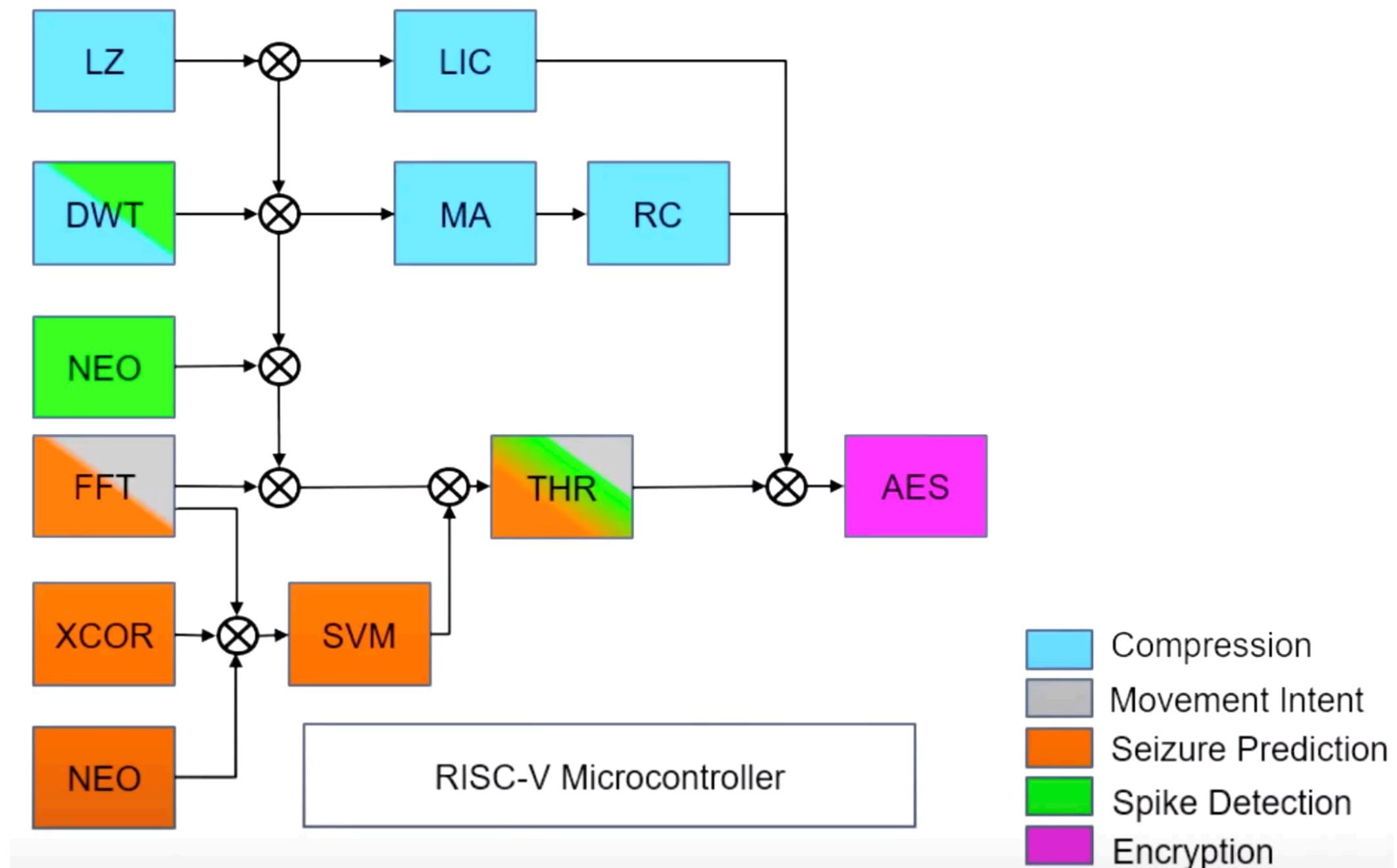


While executing LZMA:

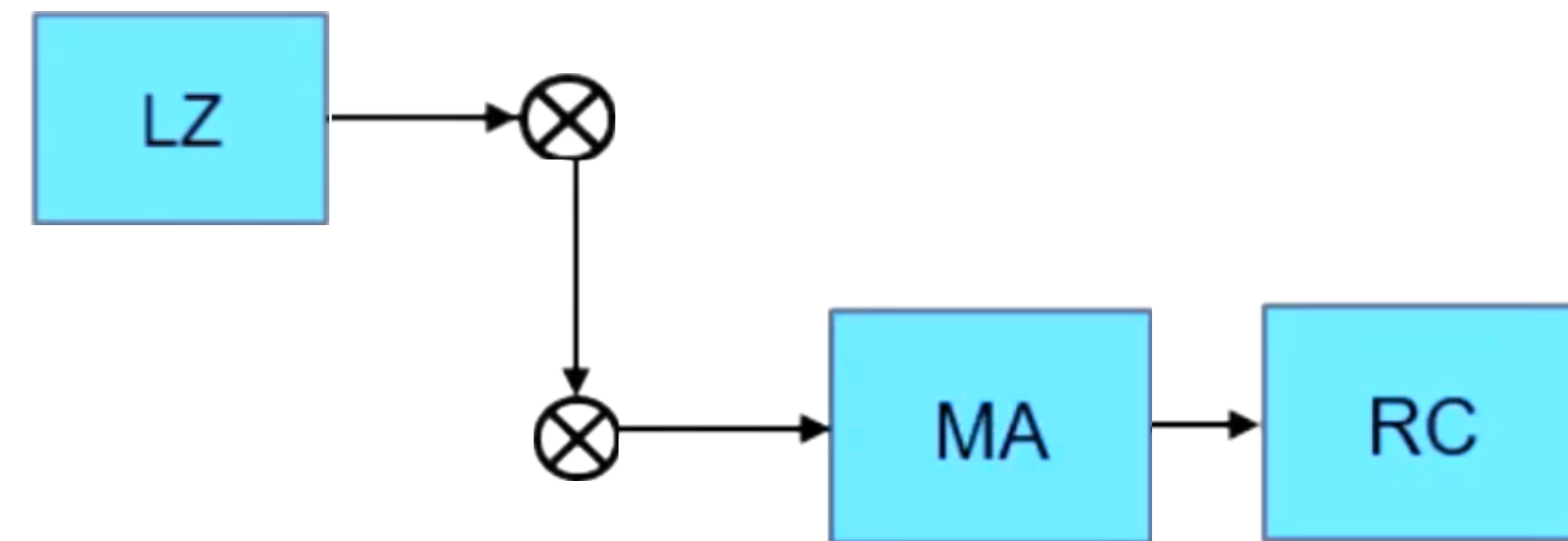


⊗ Routing Switch

General Idea

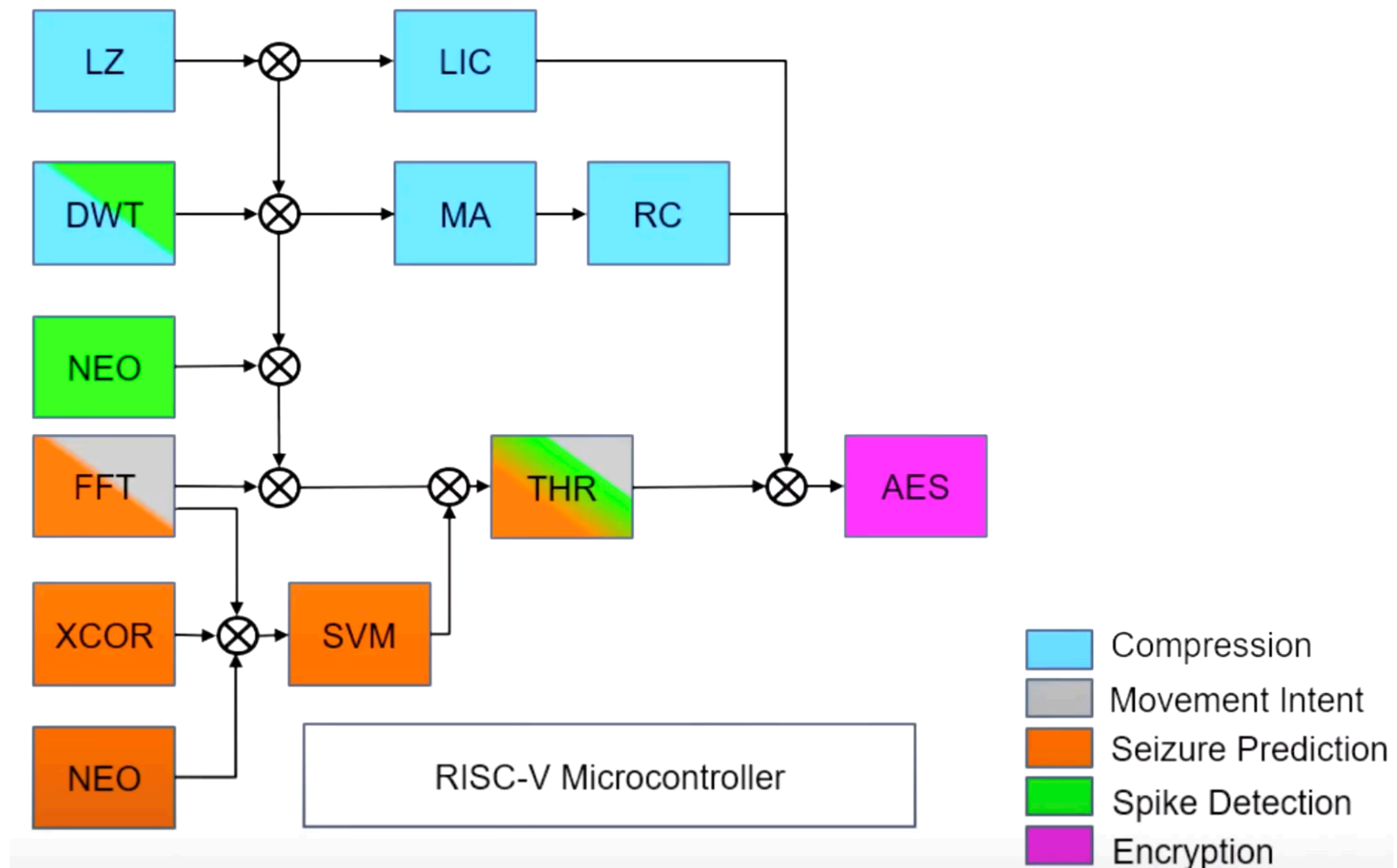


While executing LZMA:

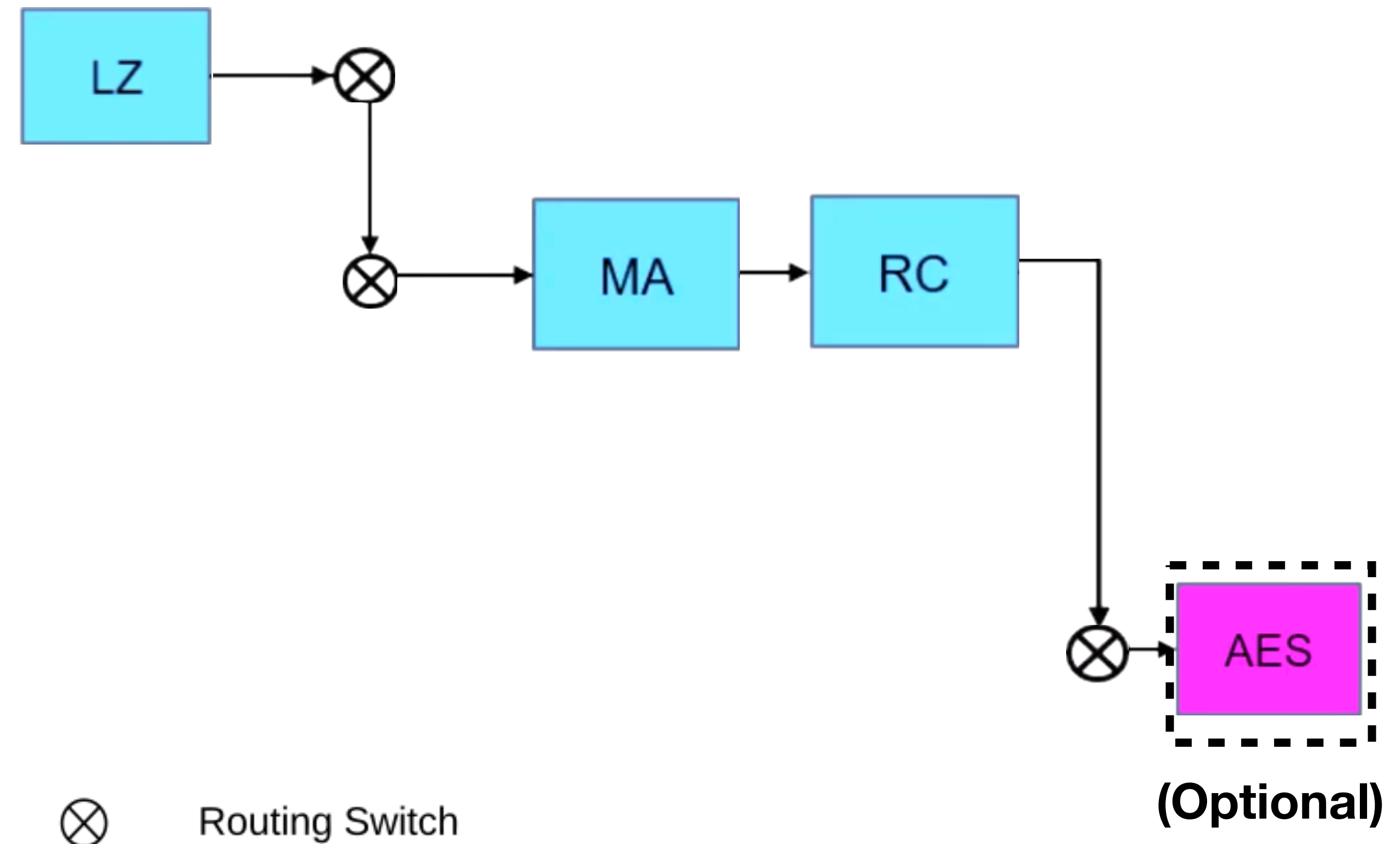


⊗ Routing Switch

General Idea



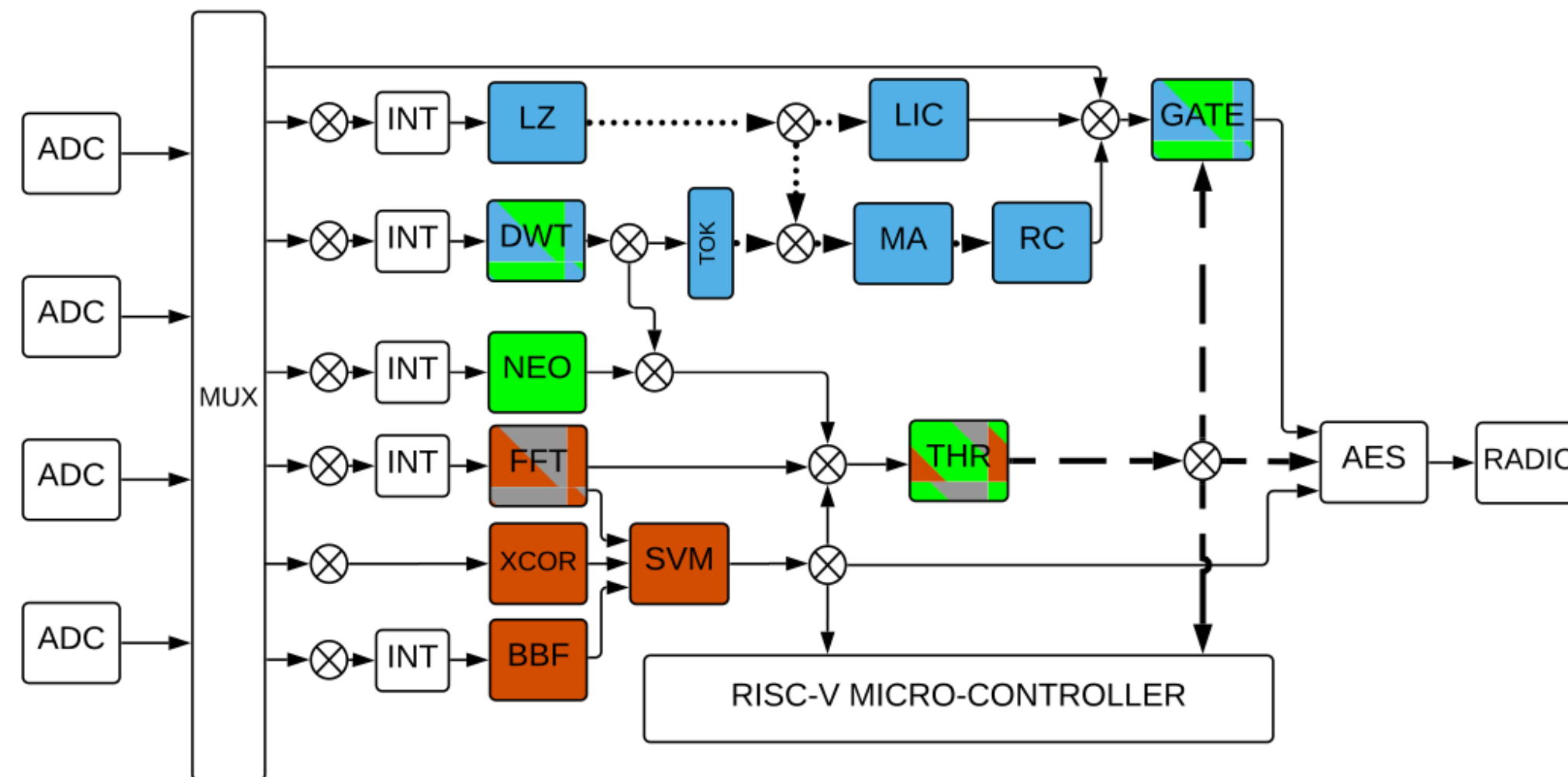
While executing LZMA:



Outline

- Background
- Problems
- Goals
- Key Mechanism
- **Implementation**
- Evaluation
- Strengths
- Weaknesses
- Discussion

Basic structure of HALO



Frequently used BCI tasks pipelines

Compression (LZ4): ADC → MUX → INT → LZ → LIC → [AES]

Compression (LZMA): ADC → MUX → INT → LZ → MA → RC → [AES]

Compression (DWT): ADC → MUX → INT → DWT → TOK → MA → RC → [AES]

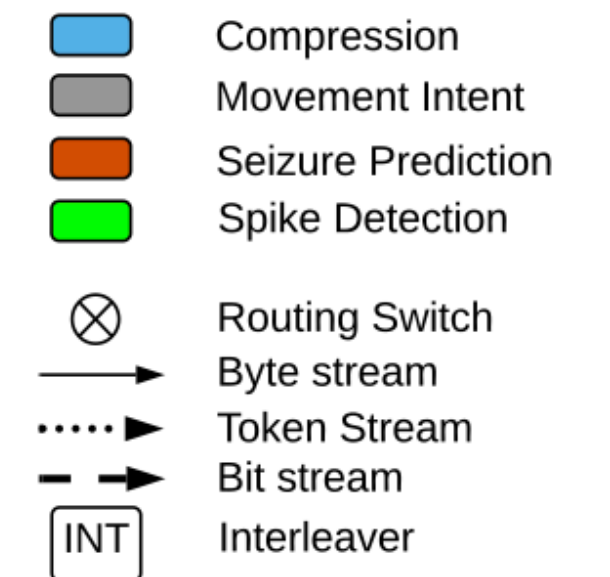
Movement Intent: ADC → MUX → INT → FFT → THR → [GATE] → [AES]

Encryption: ADC → MUX → GATE → AES

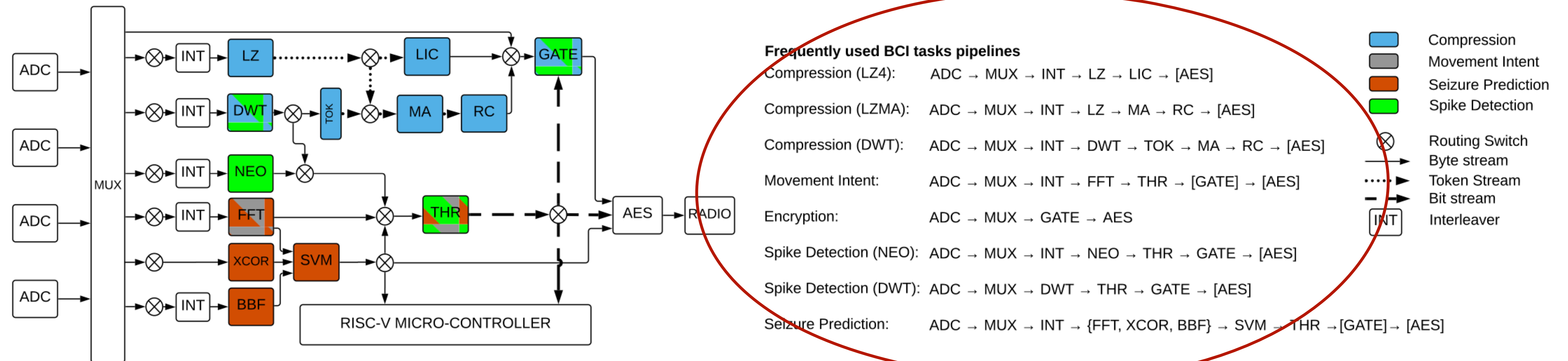
Spike Detection (NEO): ADC → MUX → INT → NEO → THR → GATE → [AES]

Spike Detection (DWT): ADC → MUX → DWT → THR → GATE → [AES]

Seizure Prediction: ADC → MUX → INT → {FFT, XCOR, BBF} → SVM → THR → [GATE] → [AES]

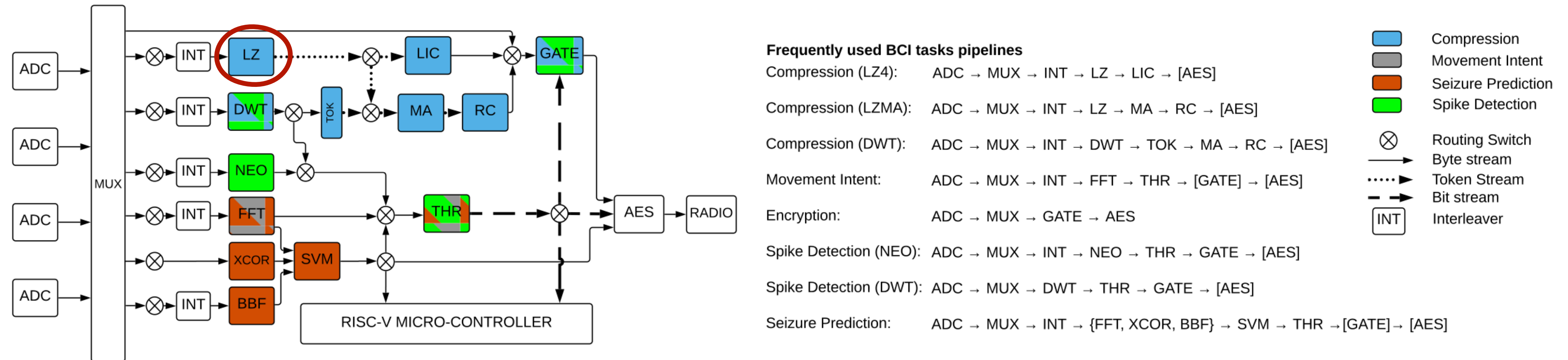


Basic structure of HALO



For each **BCI task** :Controller assembles all required PEs for this task into **pipelines** to execute the task

Basic structure of HALO



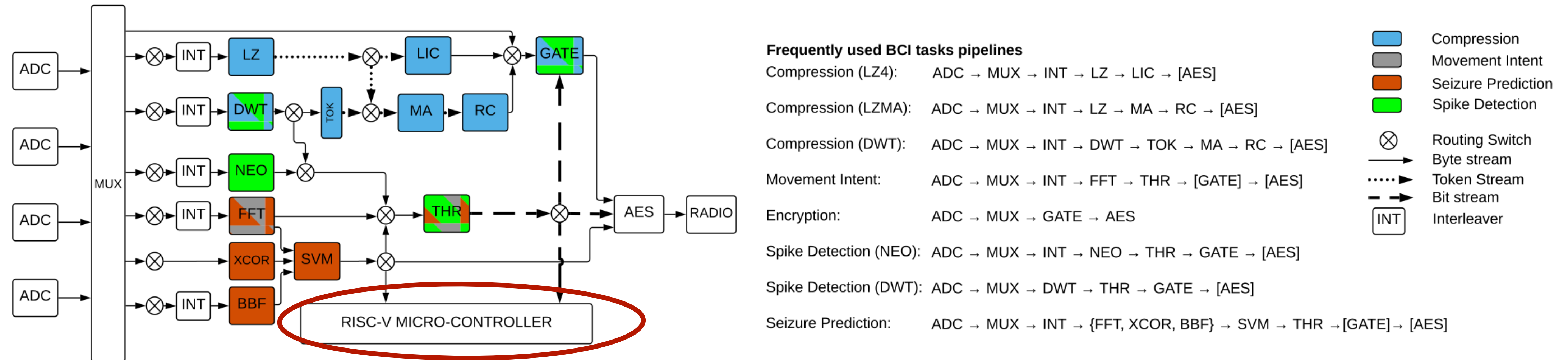
For each **BCI task** :Controller assembles all required PEs for this task into **pipelines** to execute the task

Each **single PE** operates at:

- a frequency **catered to its specific computational needs** => reduces power consumption (while ASIC ran **all** logic at **same** frequency)
- private memory => **cannot share** large amounts of data (“Locality Refactoring ” by PE decomposition)
- adapter to communicate over the interconnect

=> PEs communicate with each other via lower-power circuit-switched network built on an asynchronous communication fabric

Basic structure of HALO



For each **BCI task** :Controller assembles all required PEs for this task into **pipelines** to execute the task

Each **single PE** operates at:

- a frequency **catered to its specific computational needs** => reduces power consumption (while ASIC ran **all** logic at **same** frequency)
- private memory => **cannot share** large amounts of data (“Locality Refactoring ” by PE decomposition)
- adapter to communicate over the interconnect

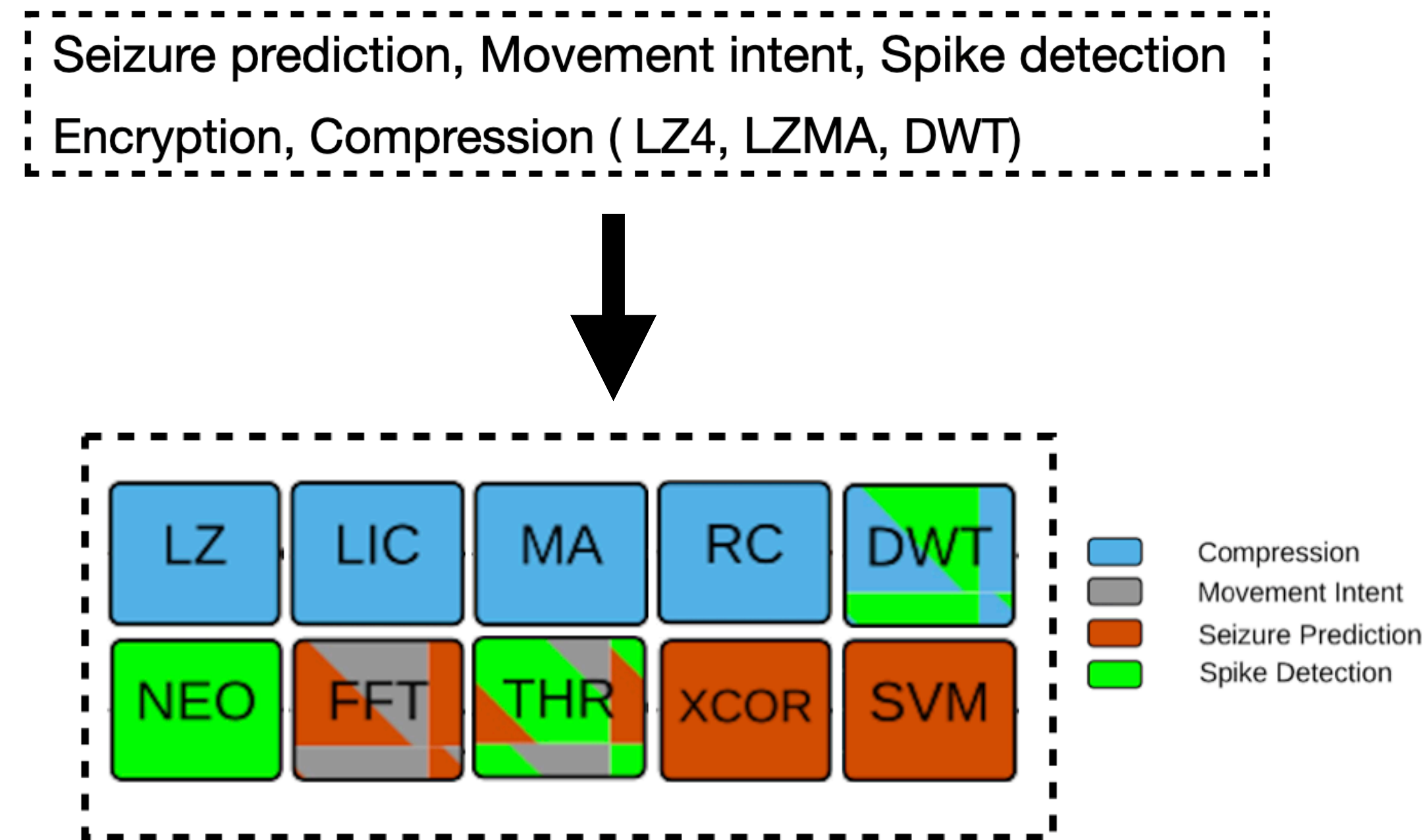
=> PEs communicate with each other via lower-power circuit-switched network built on an asynchronous communication fabric

RISC-V Micro-controller: Assembles PEs into pipelines for each task and **interrupts PEs** by **power overshoot**

PE decomposition

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

PE decomposition



- PE decomposition: Process that **refactoring** underlying algorithm of tasks into distinct pieces and then implement pieces into distinct **hardware blocks** (Processing Elements)
- Complexity depends on **how clearly separated** the algorithmic phases are

PE decomposition: Example

LZMA: One algorithm to realize **data compression**

=> **reduces** radio transmission and useful for high-bandwidth brain interaction

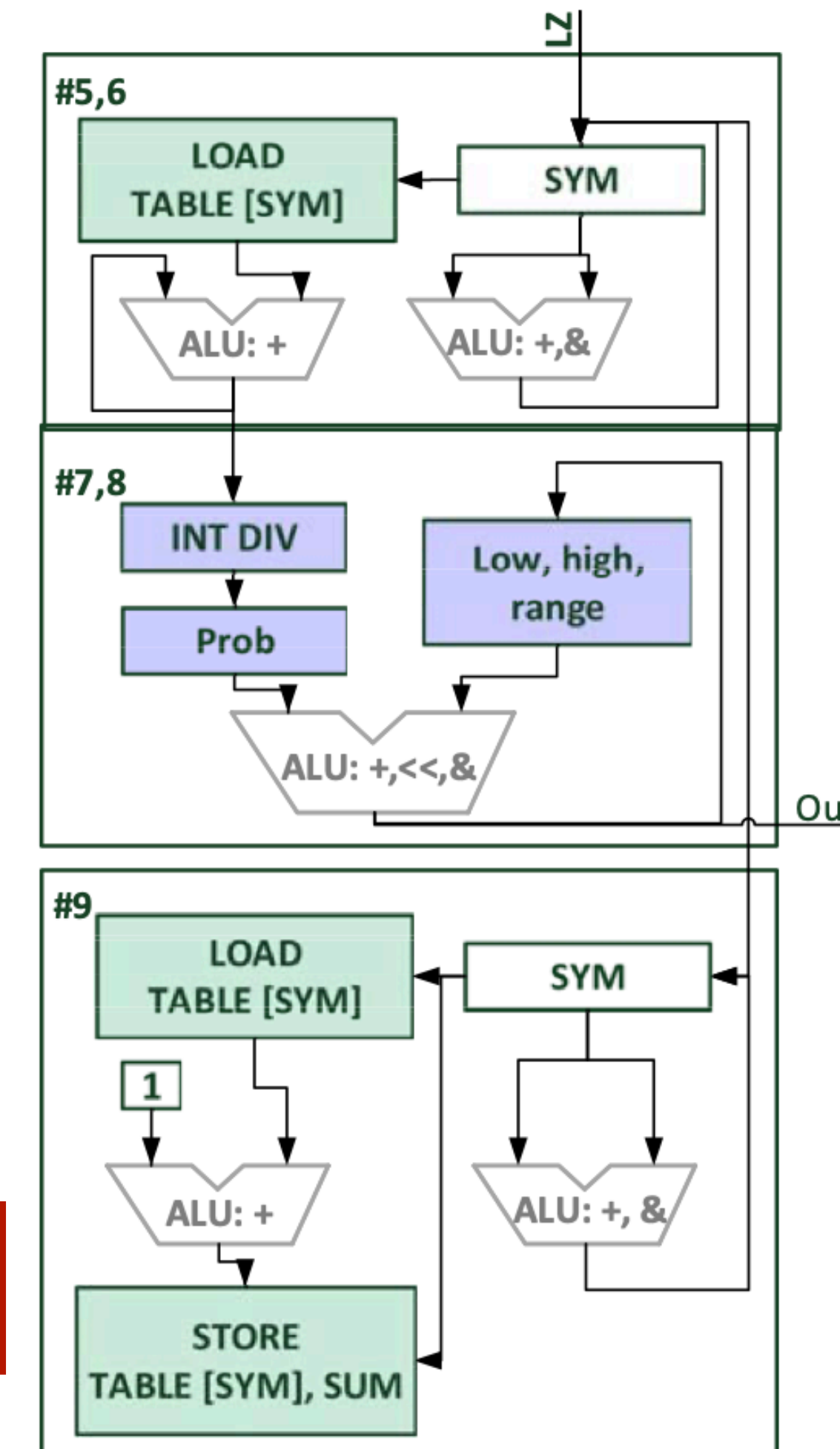
Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:     /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```

PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:       /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```



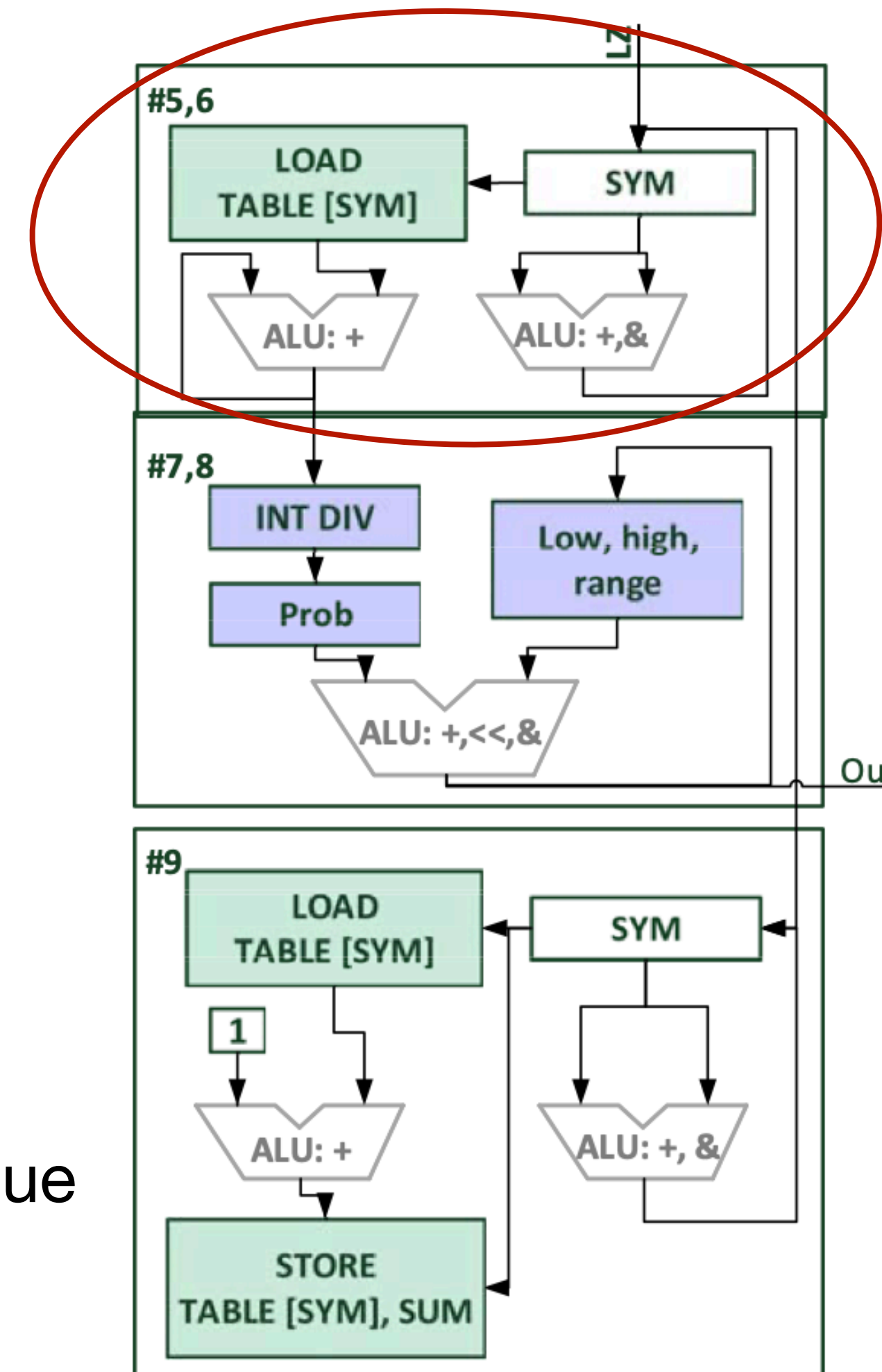
Initial Version: implement Algorithm line by line into hardware blocks

PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:     /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```

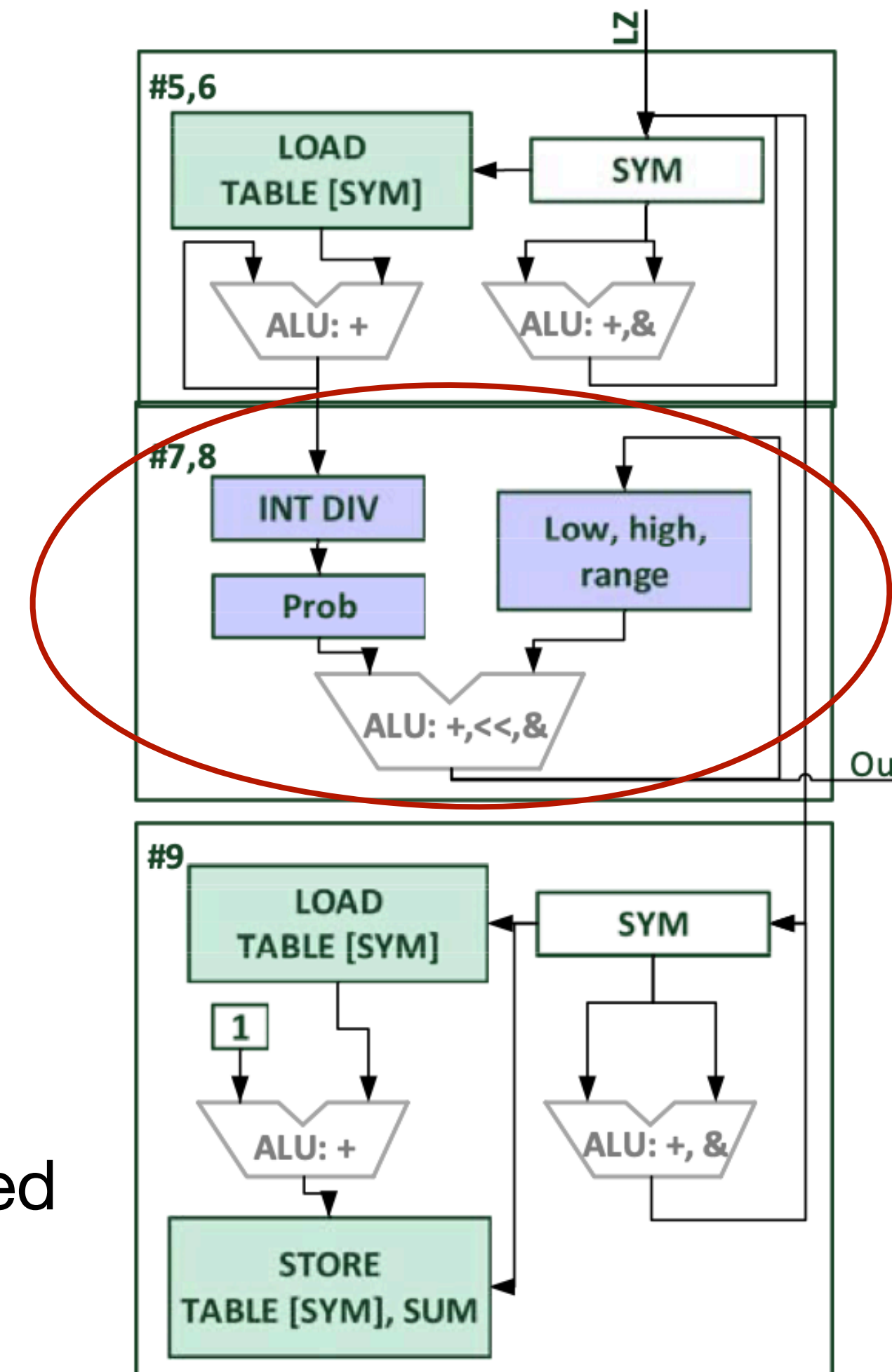
Line 5&6: use Markov (MA) chains to calculate the probability of the current input value based on observed history (frequency table)



PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:     /count total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```



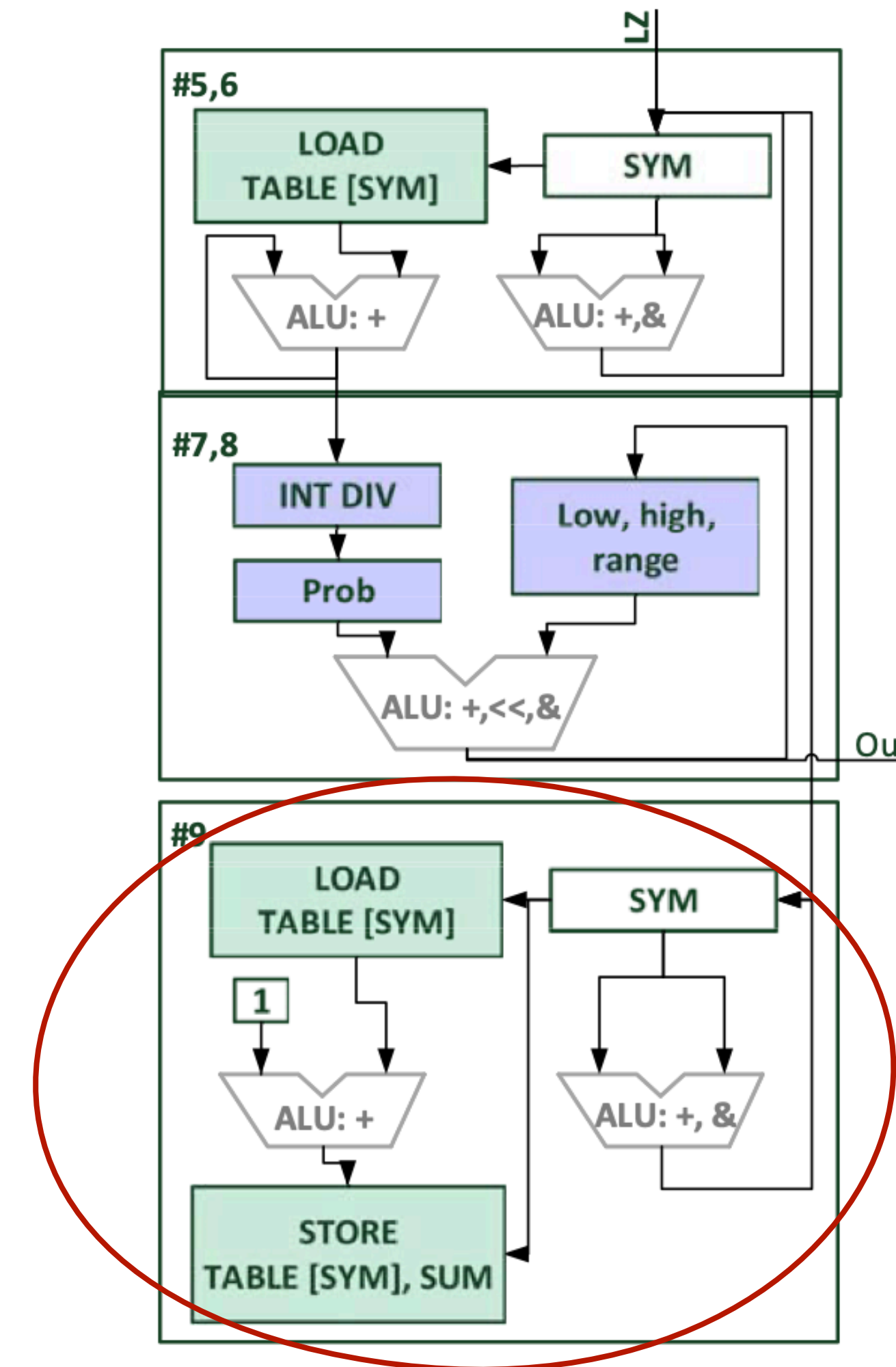
Line 7&8: try to pick more efficient encoding of the input signal based on the calculated probability in the last step

PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:       / counttotal(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:   end while
11:   return output;
12: end function
```

Line 9: update frequency table

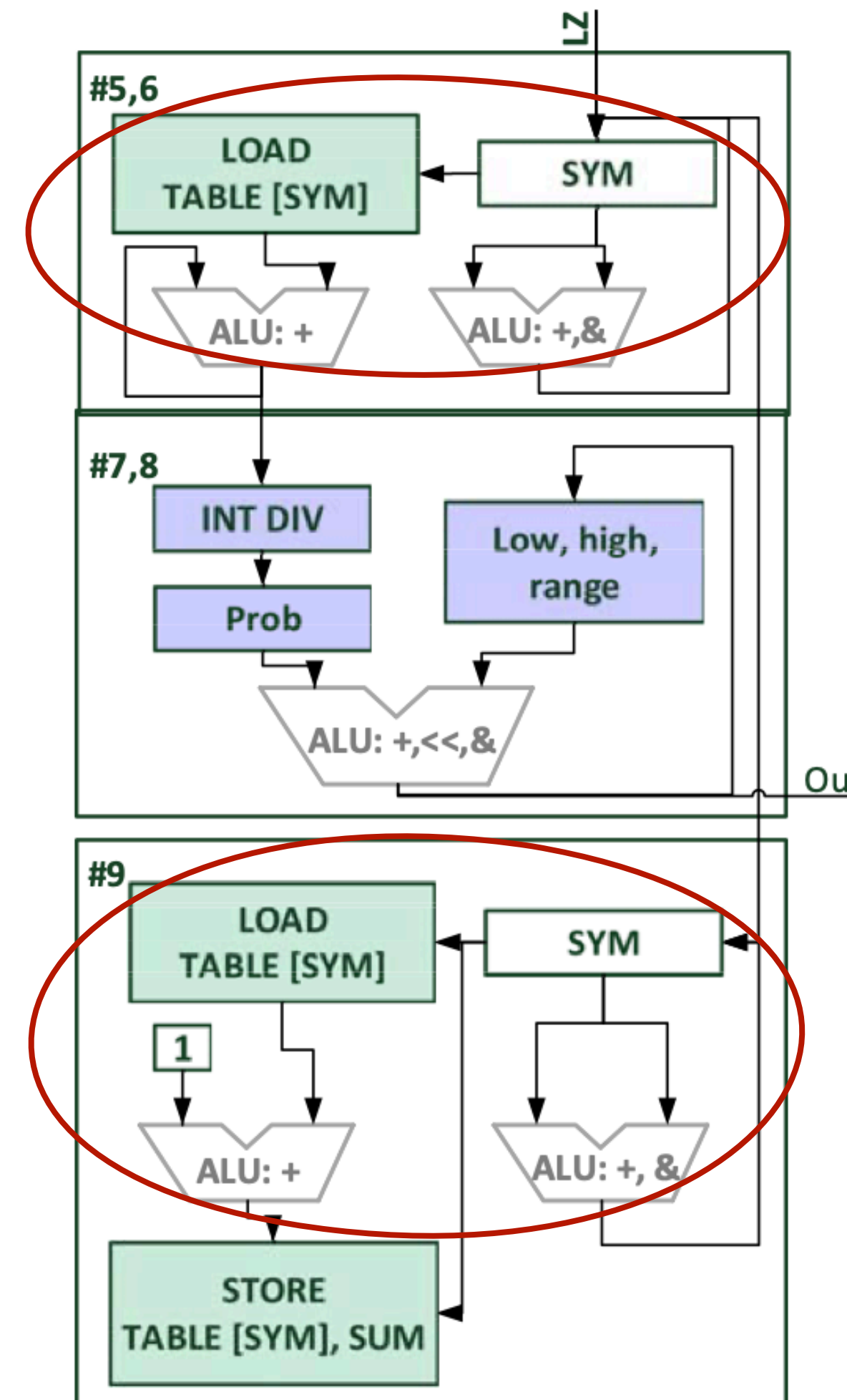


PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:       /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```

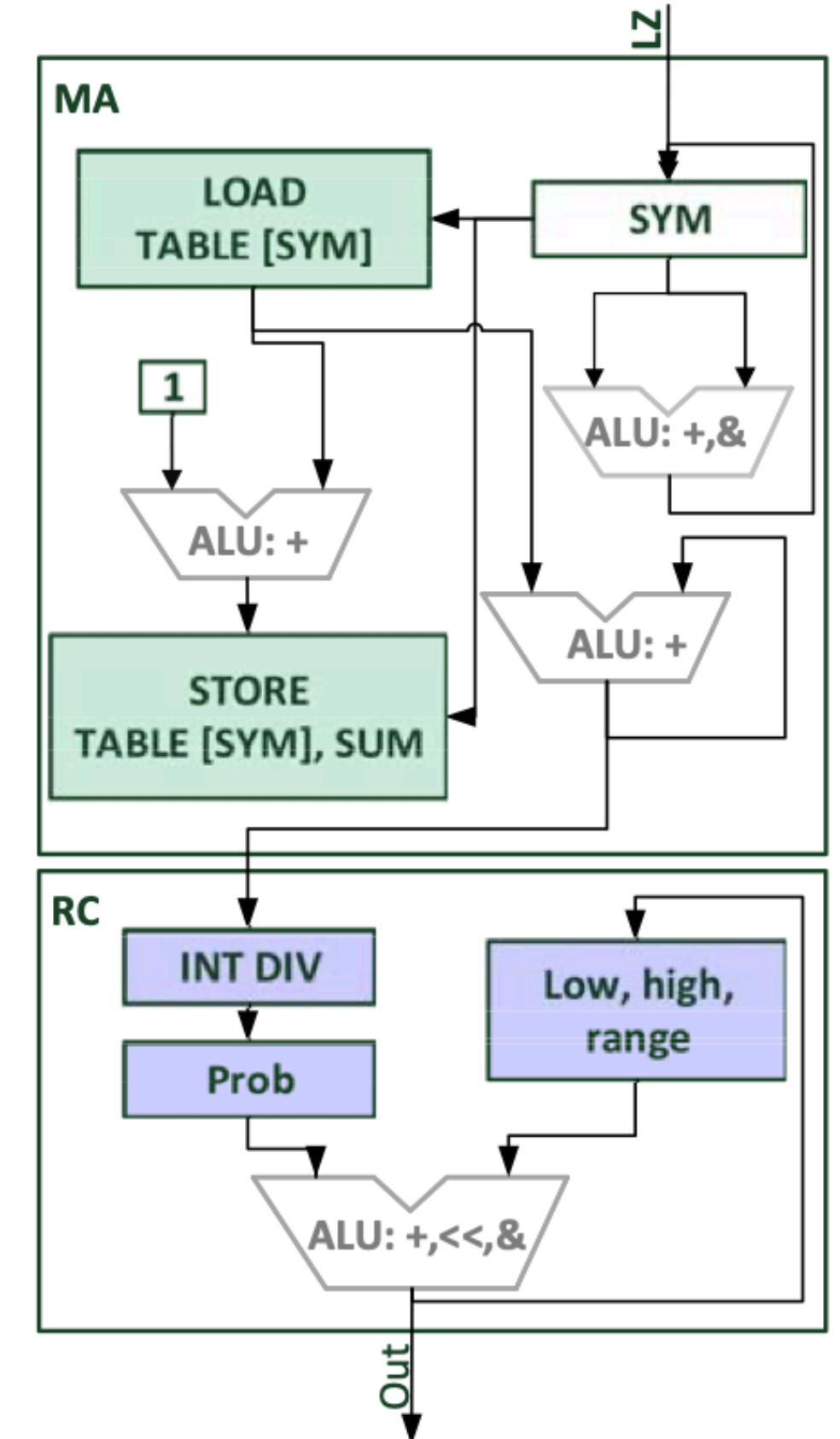
- **Duplication** of hardware component with similar / same functionality
 - Line 5,6 & 9 **share the same** table, but they are separated into **two PEs**
- Since each PE has **private memory**=> unnecessary data movement



PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:       /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```



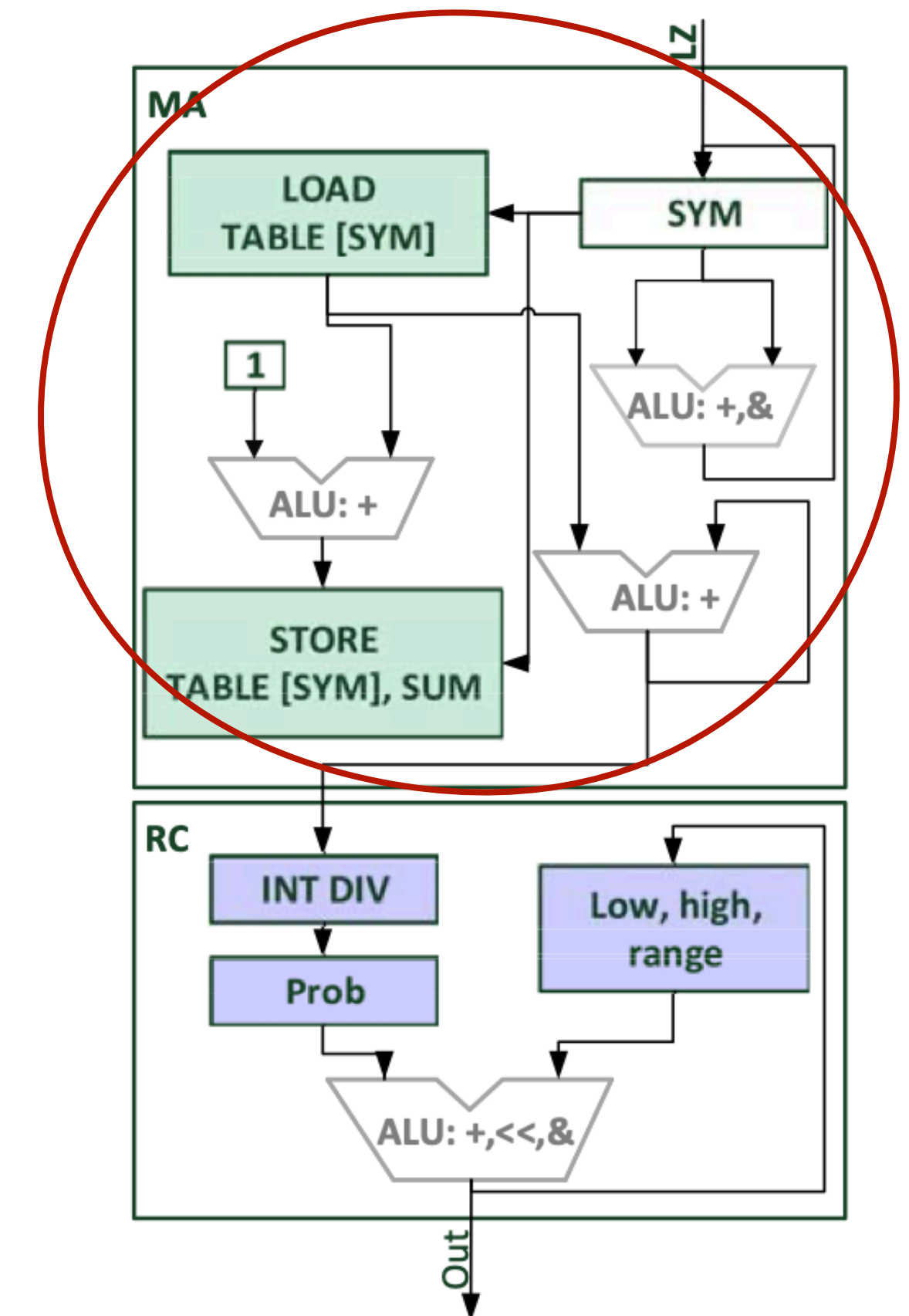
- Bring together phases that operate on the same data structures
- Separate the PEs => operate independently with minimal data movement

PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:     /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```

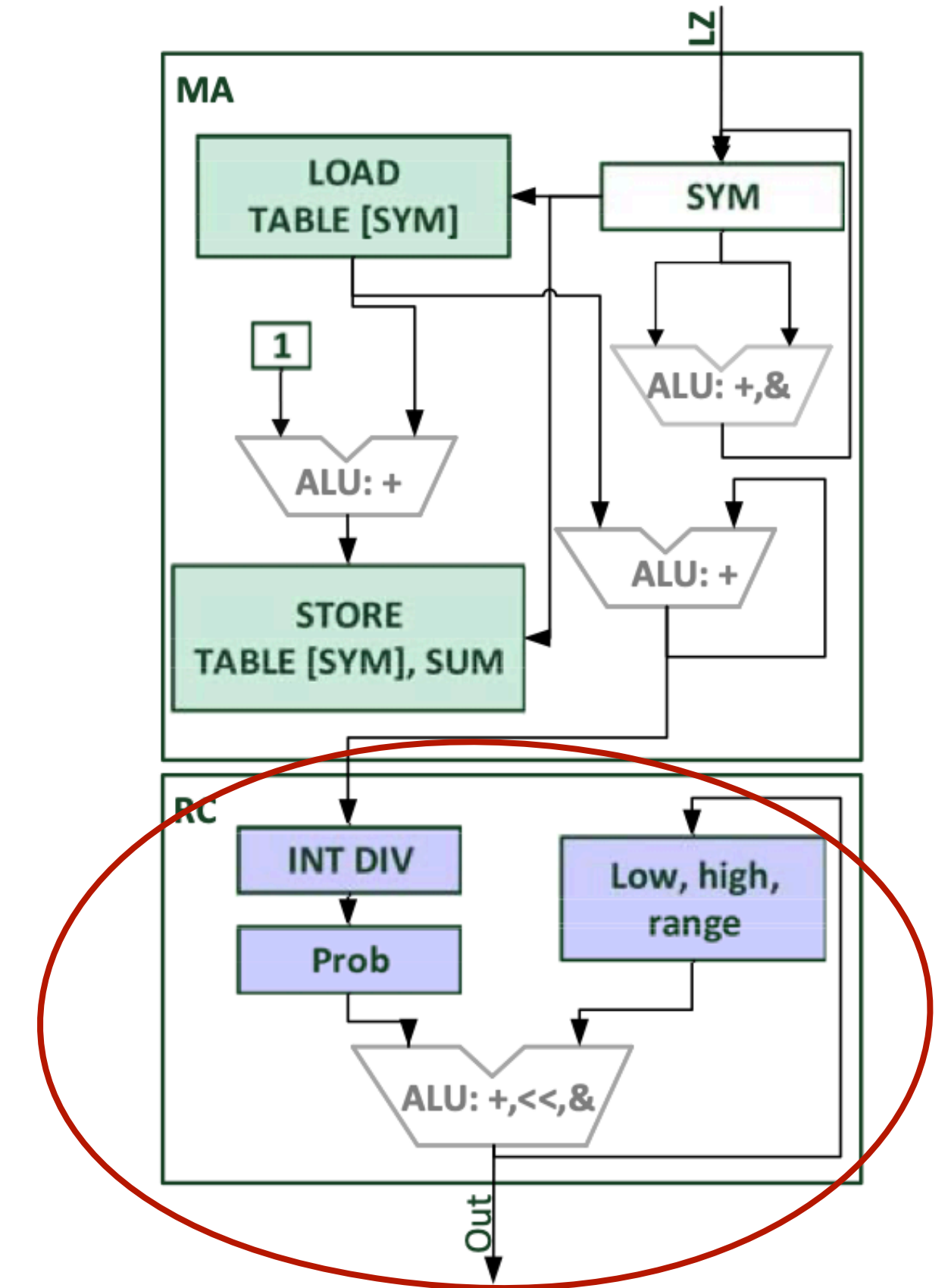
Green part: operations related to frequency table



PE decomposition: Example

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match)
6:       /count_total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
```



Blue part: operations related to encoder state

Key Mechanism: Recap

Seizure prediction, Movement intent, Spike detection

Encryption, Compression (LZ4, LZMA, DWT)

Key Mechanism: Recap

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

①

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)  
2:   output = list(lzma_header);  
3:   while data = input.get() do  
4:     best_match = find_best_match(data);  
5:     Probmatch = count(tablematch, best_match)  
6:       / counttotal(tablematch);  
7:     r1 = range_encode(Probmatch);  
8:     output.push_back(r1);  
9:     increment_counter(tablematch, best_match);  
10:  end while  
11:  return output;  
12: end function
```

① Choose LZMA task and its underlying algorithm to process at first

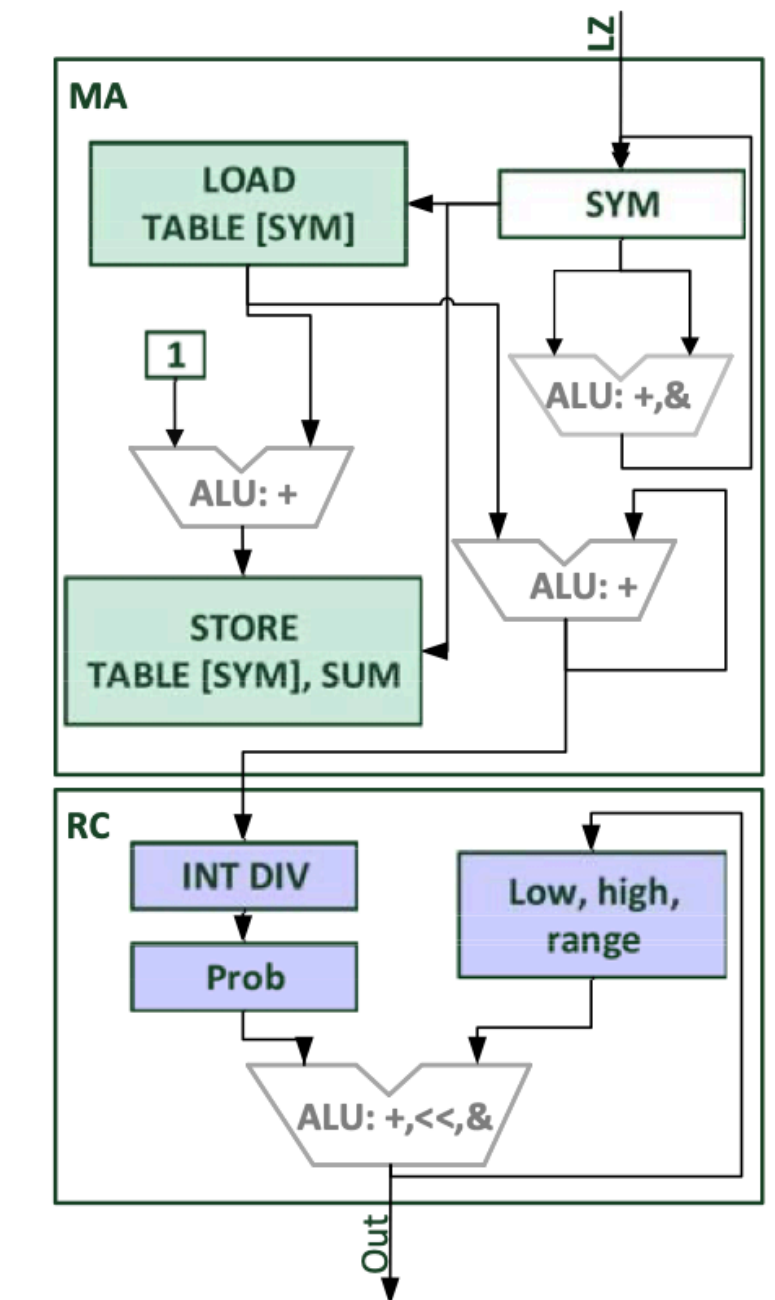
Key Mechanism: Recap

Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

Algorithm 1 LZMA pseudocode

```
1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Prob_match = count(table_match, best_match);
6:     /count total(table_match);
7:     r1 = range_encode(Prob_match);
8:     output.push_back(r1);
9:     increment_counter(table_match, best_match);
10:  end while
11:  return output;
12: end function
```

- ① Choose LZMA task and its underlying algorithm to process at first
- ② & ③ Implement line 5,6,9 into processing element “MA”
- ④ Implement line 9 into processing element “RC”



Key Mechanism: Recap

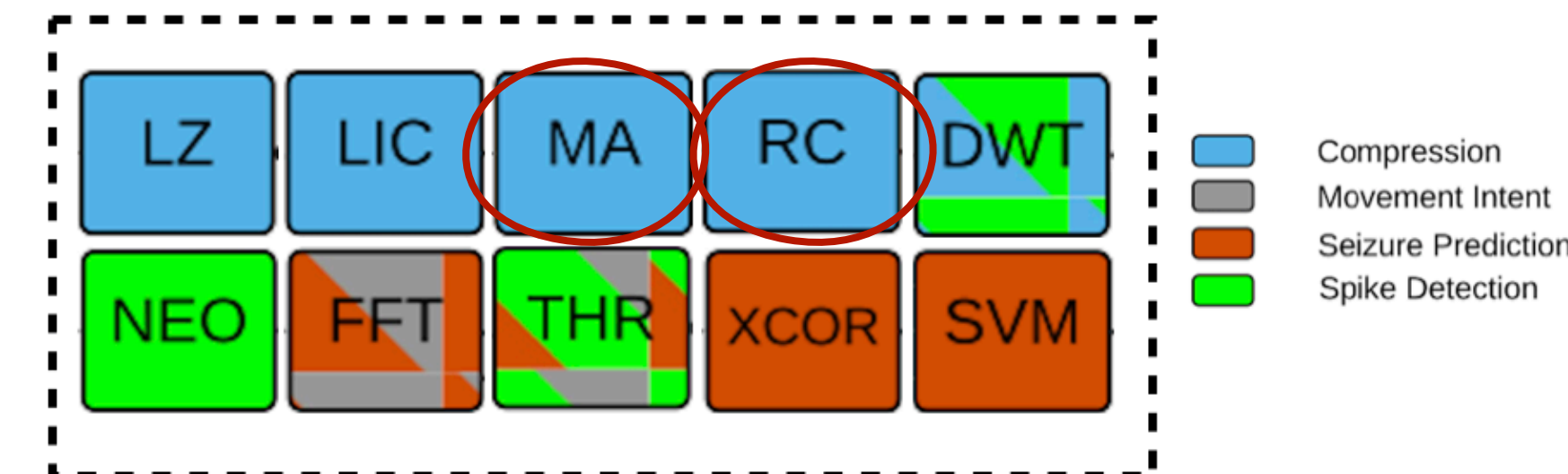
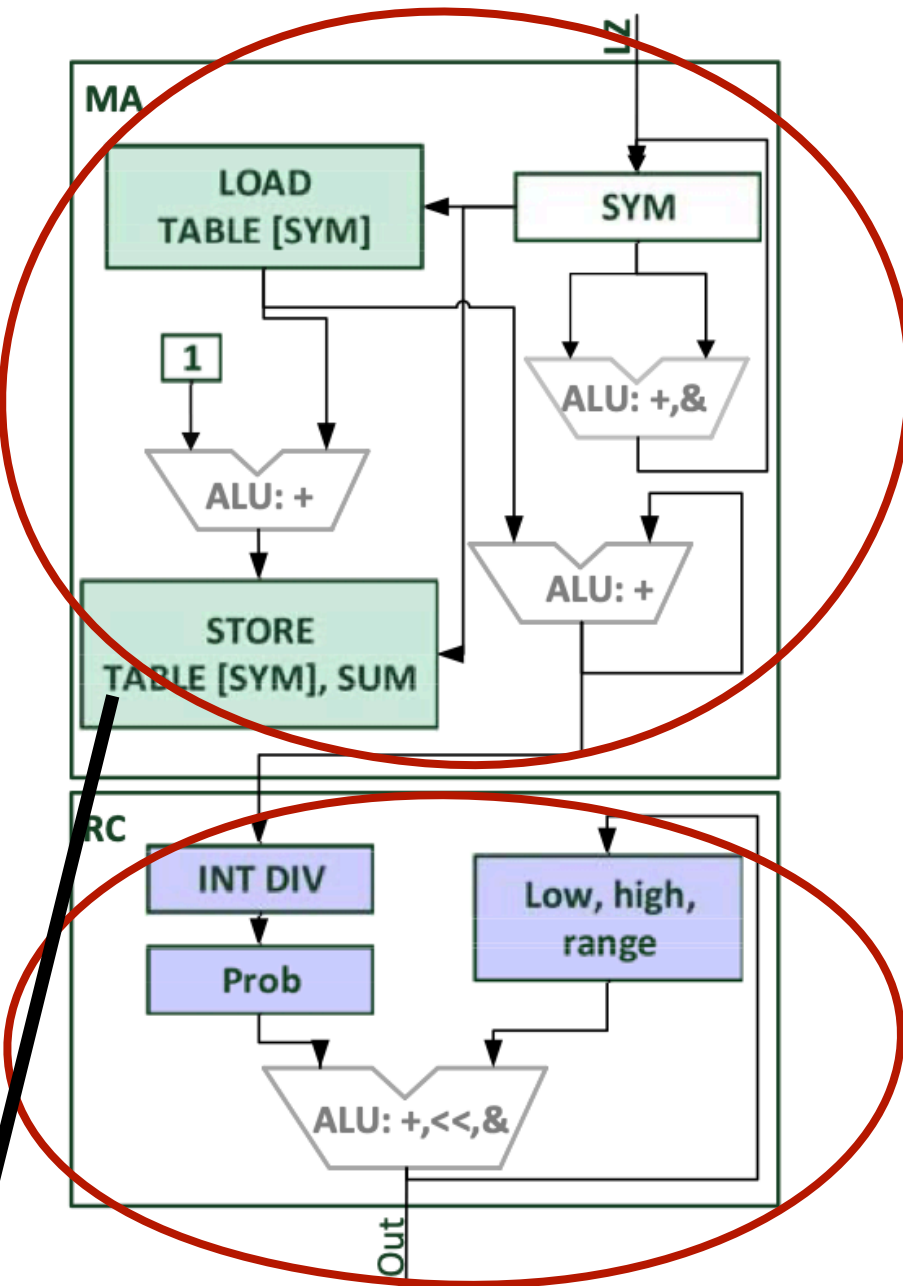
Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

Algorithm 1 LZMA pseudocode

```

1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Prob_match = count(table_match, best_match);
6:     /count total(table_match);
7:     r1 = range_encode(Prob_match);
8:     output.push_back(r1);
9:     increment_counter(table_match, best_match);
10:  end while
11:  return output;
12: end function
    
```

- ① Choose LZMA task and its underlying algorithm to process at first
- ② & ③ Implement line 5,6,9 into processing element “MA”
- ④ Implement line 9 into processing element “RC”
- ⑤ & ⑥ The resulting processing elements corresponds to square blocks MA and RC in the figure



Key Mechanism: Recap

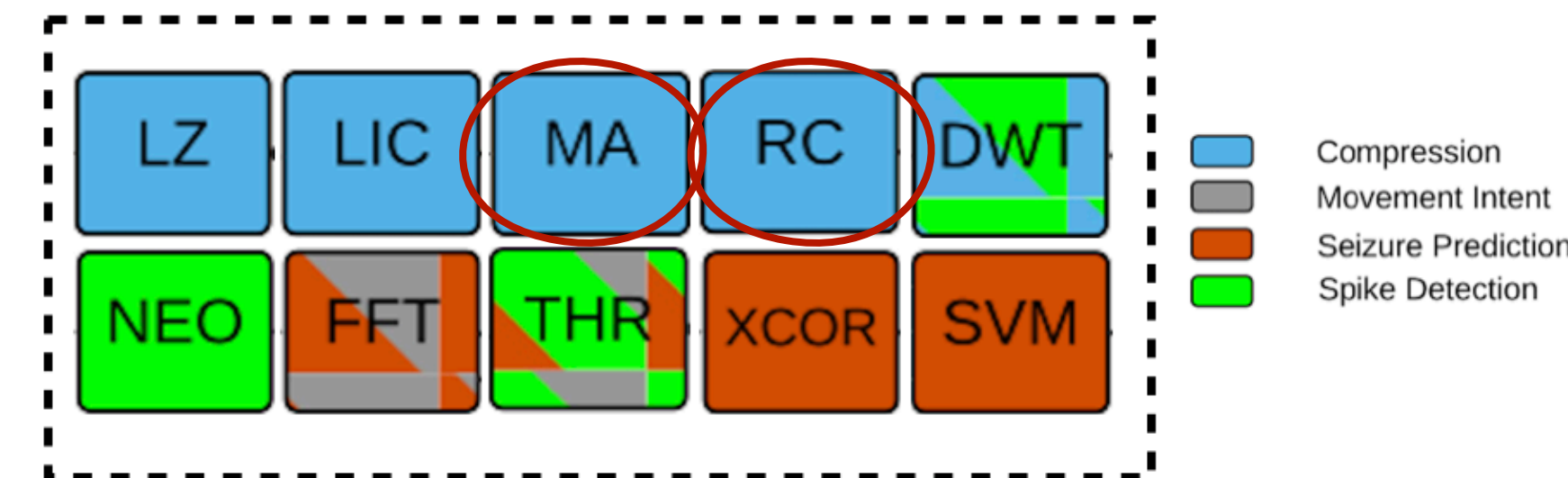
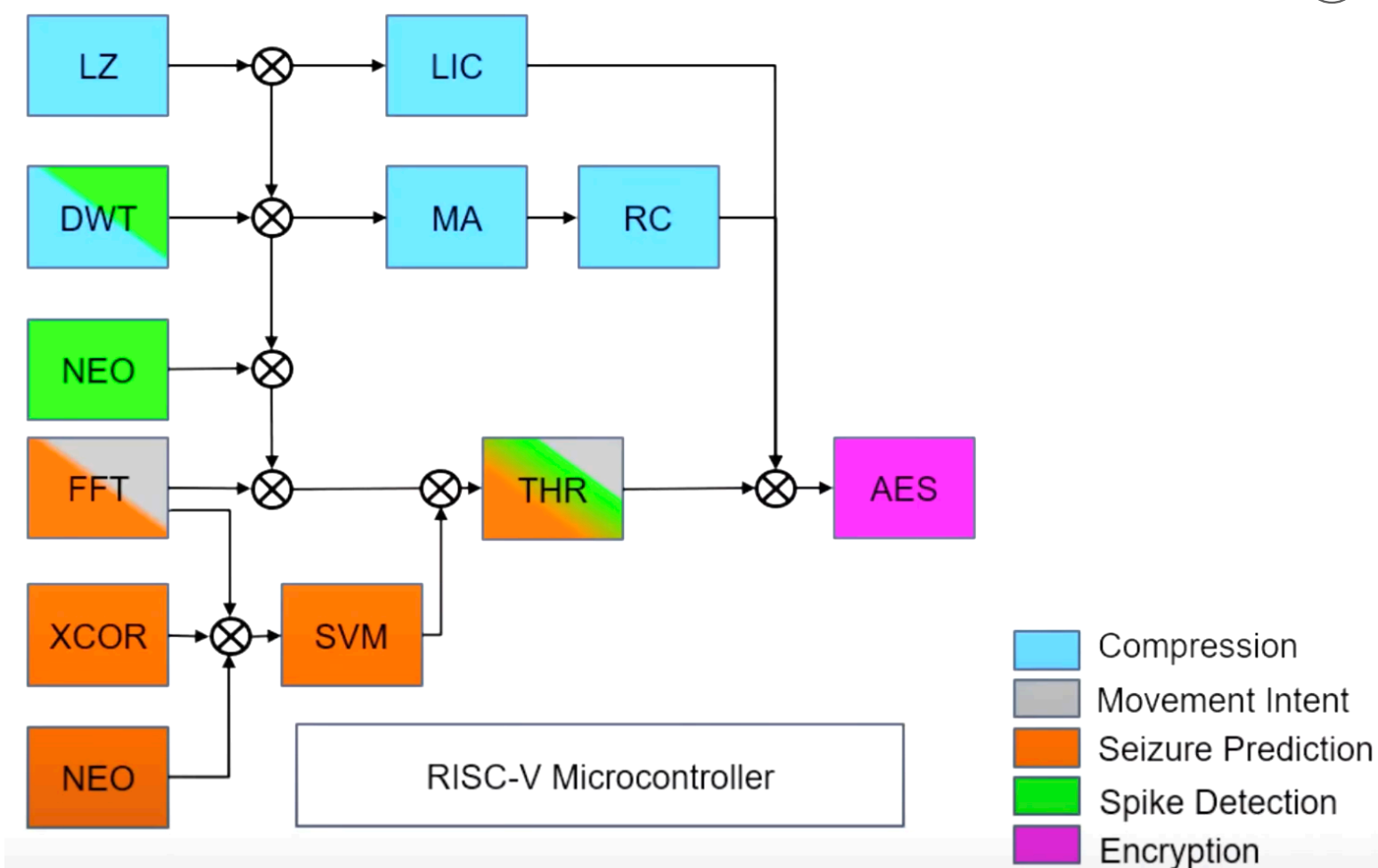
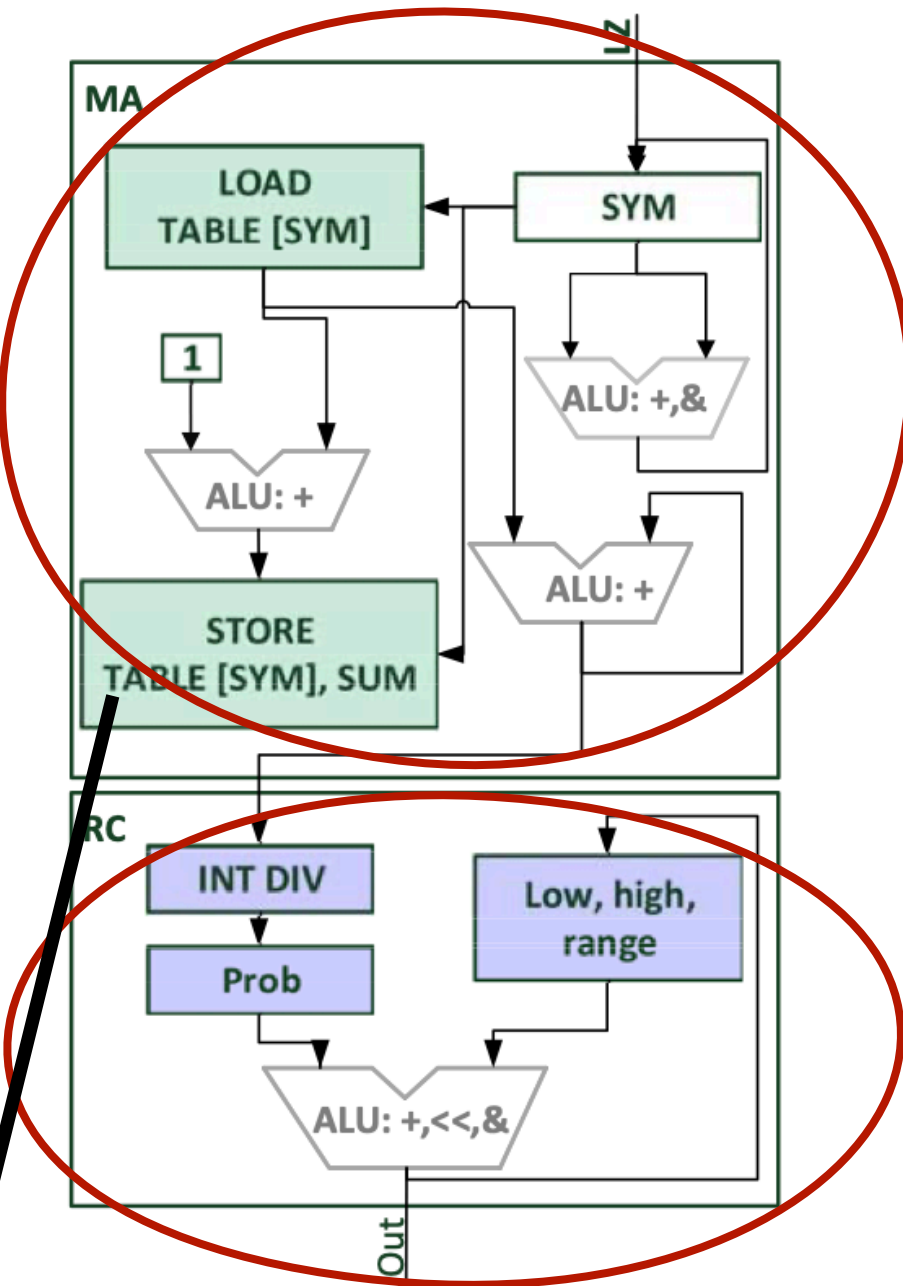
Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

Algorithm 1 LZMA pseudocode

```

1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match);
6:     /count total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
    
```

- ① Choose LZMA task and its underlying algorithm to process at first
- ② & ③ Implement line 5,6,9 into processing element “MA”
- ④ Implement line 9 into processing element “RC”
- ⑤ & ⑥ The resulting processing elements corresponds to square blocks MA and RC in the figure
- ⑦ Arrange the resulting PEs together and get the final architecture



Key Mechanism: Recap

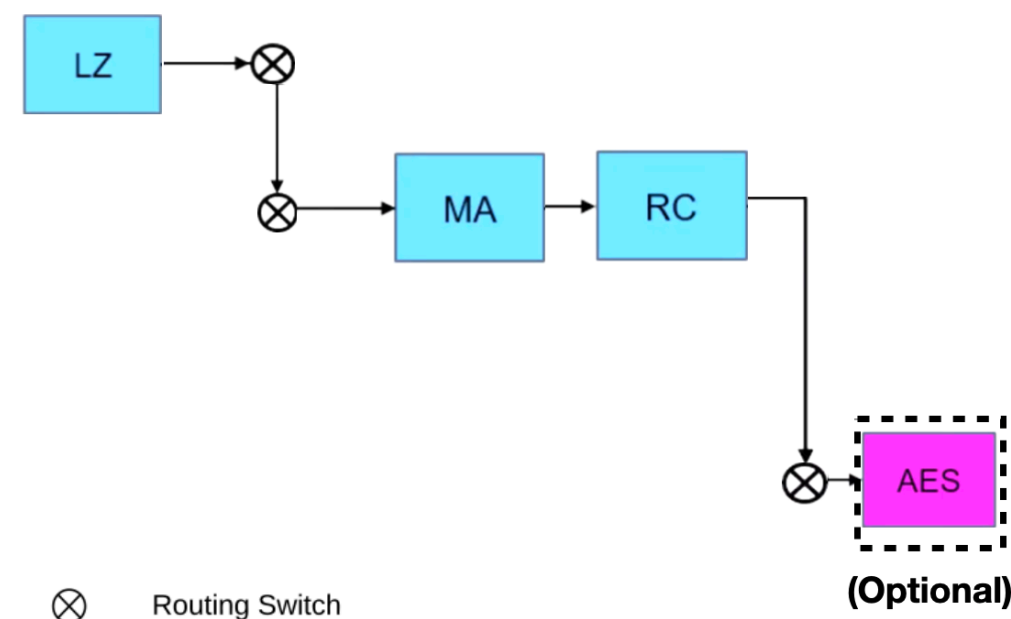
Seizure prediction, Movement intent, Spike detection
Encryption, Compression (LZ4, LZMA, DWT)

Algorithm 1 LZMA pseudocode

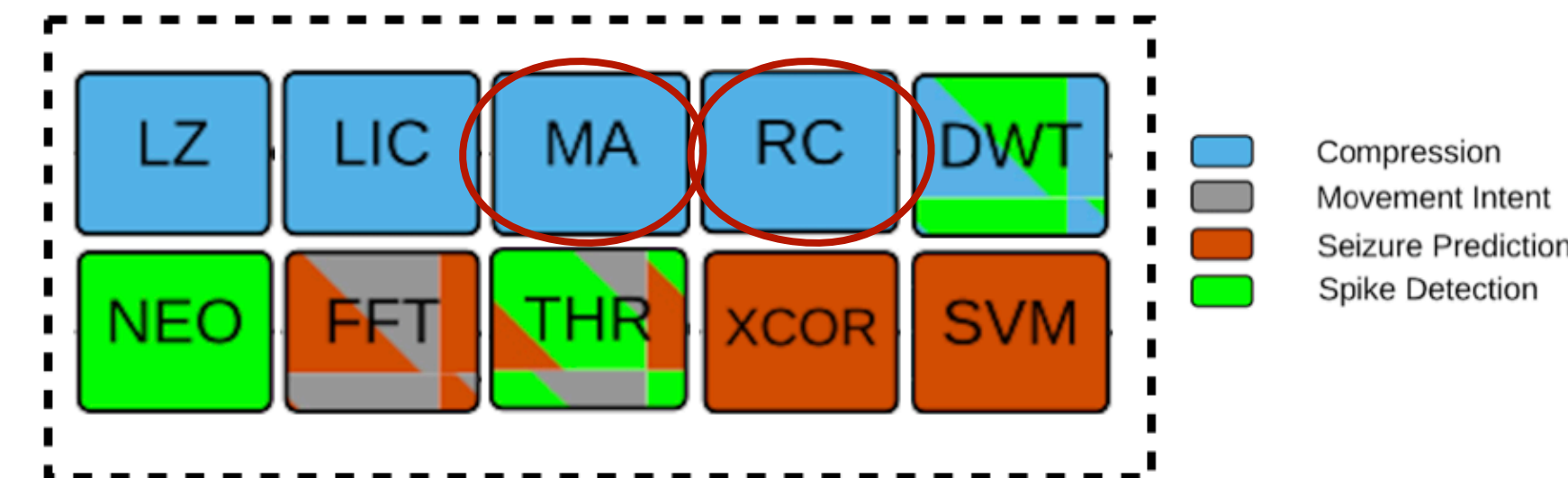
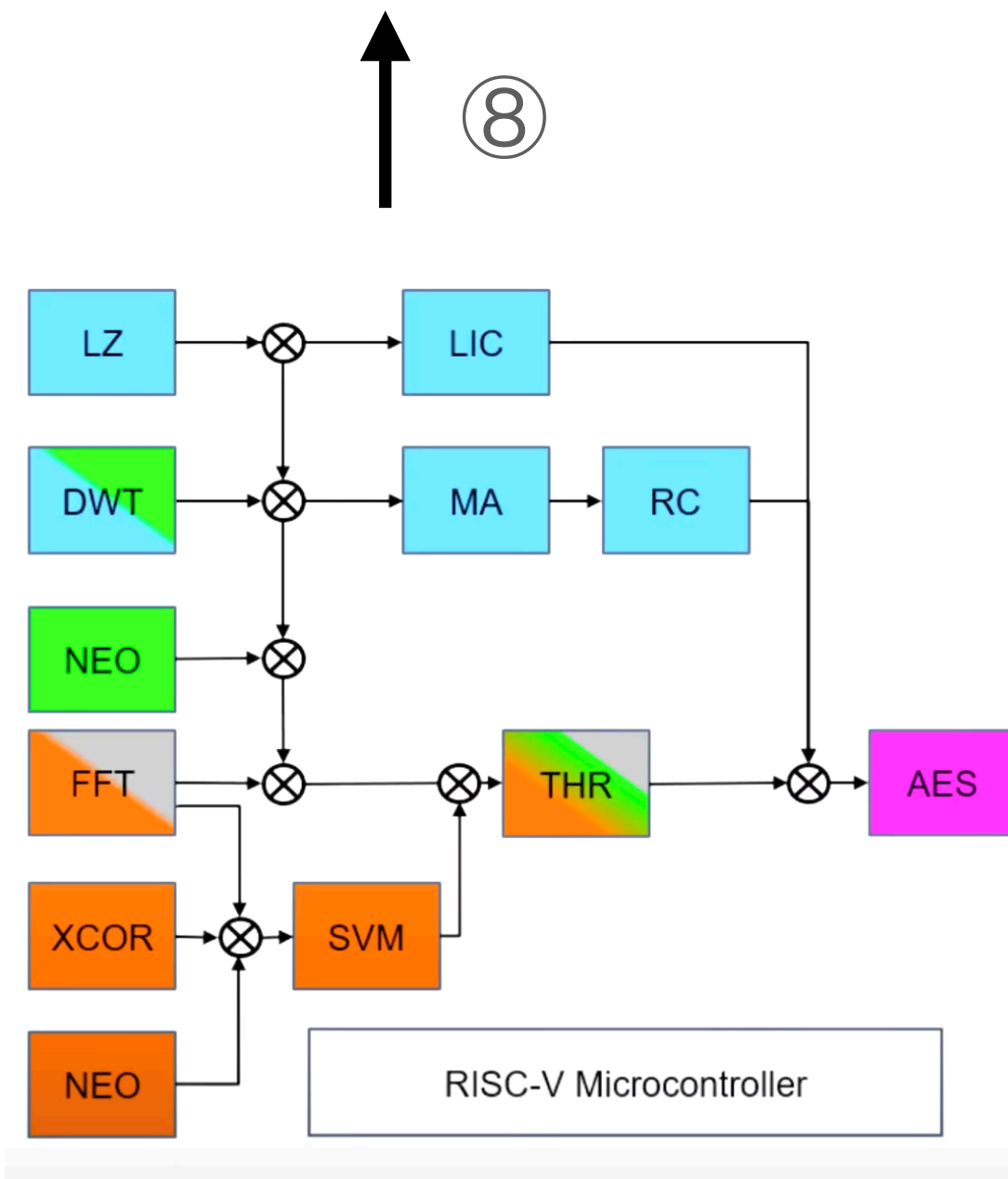
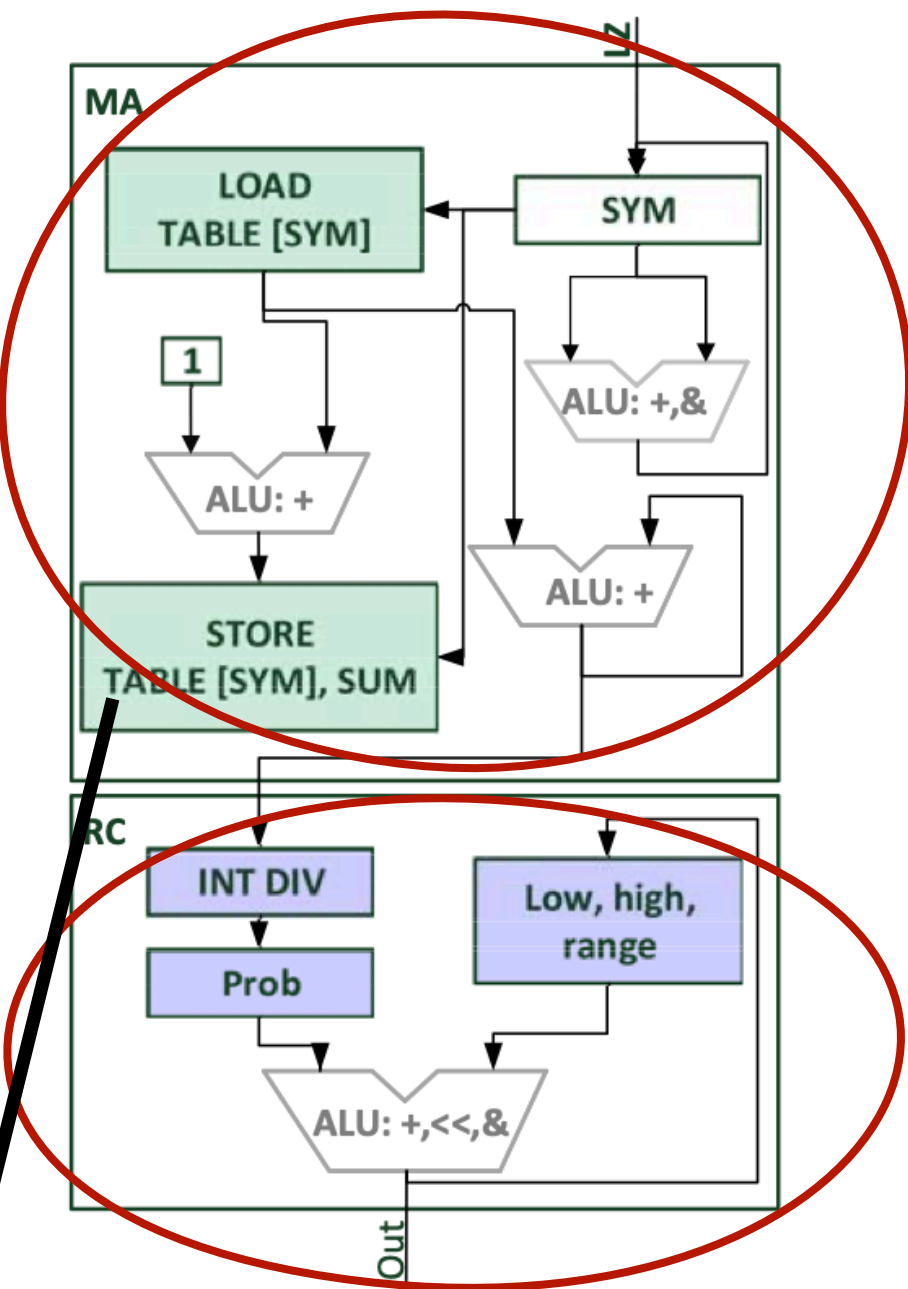
```

1: function LZMA_COMPRESS_BLOCK(input)
2:   output = list(lzma_header);
3:   while data = input.get() do
4:     best_match = find_best_match(data);
5:     Probmatch = count(tablematch, best_match);
6:     /count total(tablematch);
7:     r1 = range_encode(Probmatch);
8:     output.push_back(r1);
9:     increment_counter(tablematch, best_match);
10:  end while
11:  return output;
12: end function
    
```

While executing LZMA:



- ① Choose LZMA task and its underlying algorithm to process at first
- ② & ③ Implement line 5,6,9 into processing element “MA”
- ④ Implement line 9 into processing element “RC”
- ⑤ & ⑥ The resulting processing elements corresponds to square blocks MA and RC in the figure
- ⑦ Arrange the resulting PEs together and get the final architecture
- ⑧ When executing LZMA: configure the corresponding pipeline



Optimization of Processing elements

- Optimization in software level
- Does not change output

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 2 XCOR naive implementation

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // Calculate correlation
5:   if channel.filled() then
6:     for each  $i, j \in \text{channels}$  do
7:       data_i = 0
8:       for each  $\text{data} \in \text{channel}[i]$  do
9:         data_i + = data
10:      end for
11:      data_j = 0
12:      for  $k \in [\text{LAG}, \text{SIZE}]$  do
13:        data_j + = channel[j][k]
14:      end for
15:      avg_i = data_i / SIZE
16:      avg_j = data_j / SIZE
17:      output.push_back(avg_i, avg_j)
18:    end for
19:    return output;
20:  end if
21: end function
```

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 2 XCOR naive implementation

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // Calculate correlation
5:   if channel.filled() then
6:     for each  $i, j \in \text{channels}$  do
7:       data_i = 0
8:       for each data  $\in \text{channel}[i]$  do
9:         data_i + = data
10:      end for
11:      data_j = 0
12:      for  $k \in [\text{LAG}, \text{SIZE}]$  do
13:        data_j + = channel[j][k]
14:      end for
15:      avg_i = data_i / SIZE
16:      avg_j = data_j / SIZE
17:      output.push_back(avg_i, avg_j)
18:    end for
19:    return output;
20:  end if
21: end function
```

- Process data **in blocks** instead of samples
- **Wait for all inputs** in the block to arrive

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 2 XCOR naive implementation

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // Calculate correlation
5:   if channel.filled() then
6:     for each  $i, j \in \text{channels}$  do
7:       data_i = 0
8:       for each data  $\in \text{channel}[i]$  do
9:         data_i += data
10:      end for
11:      data_j = 0
12:      for  $k \in [\text{LAG}, \text{SIZE}]$  do
13:        data_j += channel[j][k]
14:      end for
15:      avg_i = data_i / SIZE
16:      avg_j = data_j / SIZE
17:      output.push_back(avg_i, avg_j)
18:    end for
19:    return output;
20:  end if
21: end function
```

- Process data **in blocks** instead of samples
- **Wait for all inputs** in the block to arrive
- When all inputs arrive, computation **occurs in a burst**
- Requires large buffers to sink the bursts, or high PE frequency

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 3 XCOR spatial programming refactoring

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // data[] stores sums of input received so far
5:   data[count] += input
6:   // data_lag[] stores sums of input till LAG
7:   if count_2 == LAG then
8:     data_lag[count] = data[count]
9:   end if
10:  // Finish correlation computation
11:  if channel.filled() then
12:    for each  $i, j \in \text{channels}$  do
13:      avg_i = data[i]/SIZE
14:      avg_j = (data[j] - data_lag[j])/SIZE
15:      output.push_back(avg_i, avg_j)
16:    end for
17:    return output
18:  end if
19: end function
```

Optimization: **avoid the bursty computation**

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 3 XCOR spatial programming refactoring

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // data[] stores sums of input received so far
5:   data[count] += input
6:   // data_lag[] stores sums of input till LAG
7:   if count_2 == LAG then
8:     data_lag[count] = data[count]
9:   end if
10:  // Finish correlation computation
11:  if channel.filled() then
12:    for each i, j ∈ channels do
13:      avg_i = data[i]/SIZE
14:      avg_j = (data[j] - data_lag[j])/SIZE
15:      output.push_back(avg_i, avg_j)
16:    end for
17:    return output
18:  end if
19: end function
```

Optimization: avoid the bursty computation

- Complete part of computation while reading inputs

Optimization of Processing elements

- Optimization in software level
- Does not change output

Example:

Algorithm 3 XCOR spatial programming refactoring

```
1: function XCOR(input, output)
2:   // channel[][] stores input in appropriate channel location
3:   channel[channel_num][sample_num] = input
4:   // data[] stores sums of input received so far
5:   data[count] += input
6:   // data_lag[] stores sums of input till LAG
7:   if count_2 == LAG then
8:     data_lag[count] = data[count]
9:   end if
10:  // Finish correlation computation
11:  if channel.filled() then
12:    for each  $i, j \in \text{channels}$  do
13:      avg_i = data[i]/SIZE
14:      avg_j = (data[j] - data_lag[j])/SIZE
15:      output.push_back(avg_i, avg_j)
16:    end for
17:    return output
18:  end if
19: end function
```

Optimization: avoid the bursty computation

- Complete part of computation while reading inputs
- Amount of computation needed in the final step is reduced
- Power savings of 2.2x over the original algorithm

Optimization of Processing elements

- Adapting the **precision** => **reduce** power consumption **significantly** while **causing slight error**

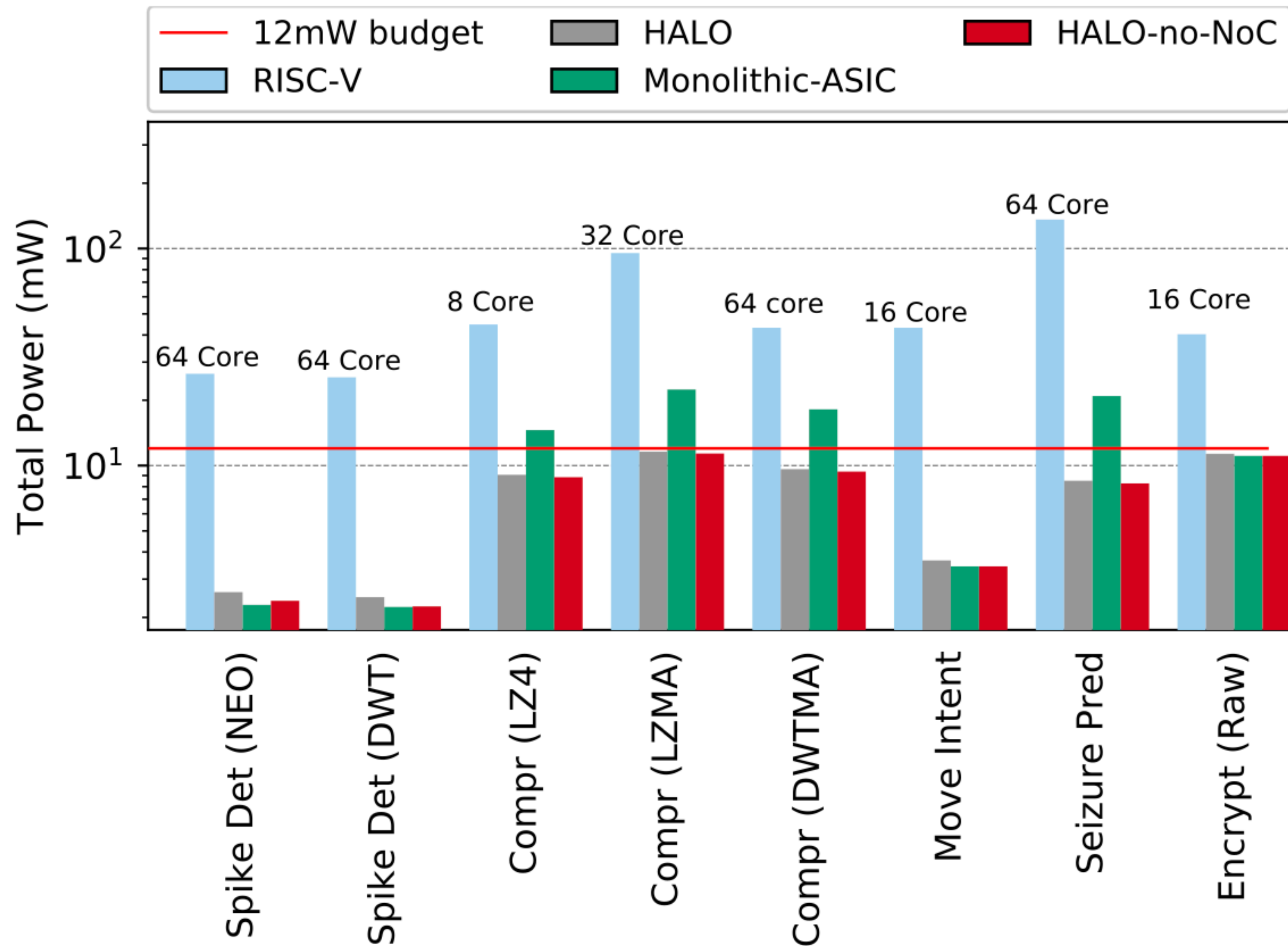
Example:

- Unnecessary high resolution by some of the signal processing algorithms (32 bit integers)
- Replace **floating point** arithmetic with **fixed point** arithmetic (e.g. in BBF PE)
- Results in only **< 0.1% increase in relative error** and an order of magnitude reduction in power

Outline

- Background
- Problems
- Goals
- Key Mechanism
- Implementation
- **Evaluation**
- Strengths
- Weaknesses
- Discussion

HALO versus RISC-V and monolithic ASICs



HALO can satisfy the constraint (red line) for all tasks

Power Analysis of HALO

PE	Freq (MHz)	Logic (mW)		Mem (mW)		Total (mW)	Area (KGE)
		Leak	Dyn	Leak	Dyn		
LZ	129	0.055	1.455	0.095	1.466	3.071	55
LIC	22.5	0.057	0.267	0.006	0.046	0.376	25
MA	92	0.127	2.148	0.067	0.997	3.339	66
RC	90	0.029	0.763	0	0	0.792	12
DWT	3	0.004	0.002	0	0	0.006	2
NEO	3	0.012	0.003	0	0	0.015	5
FFT	15.7	0.057	0.509	0.085	0.356	1.007	22
XCOR	85	0.07	4.182	0.307	0.053	4.612	81
BBF	6	0.066	0.034	0	0	0.1	23
SVM	3	0.018	0.018	0.081	0.033	0.15	8
THR	16	0.002	0.011	0	0	0.013	1
GATE	5	0.003	0.006	0.067	0.054	0.13	17
AES	5	0.053	0.059	0	0	0.112	34
Tasks							
Compr (LZ4)		0.112	1.722	0.101	1.512	3.447	80
Compr (LZMA)		0.211	4.366	0.122	2.463	7.162	133
Compr (DWTMA)		0.16	2.913	0.0123	0.33	3.415	80
Seizure Prediction		0.216	4.760	0.54	0.496	6.012	111
Spike Det (NEO)		0.017	0.02	0.067	0.054	0.158	24
Spike Det (DWT)		0.009	0.019	0.067	0.054	0.149	20
Movement Intent		0.062	0.526	0.152	0.41	1.15	40
Encrypt (Raw)		0.053	0.059	0	0	0.112	34
RISC-V Control _{est}	25	0.341	0.137	0.248	1.080	1.800	70

Compression and seizure prediction: consume the most power

Power Consume of each pipeline under 15 mW

Power Analysis of HALO

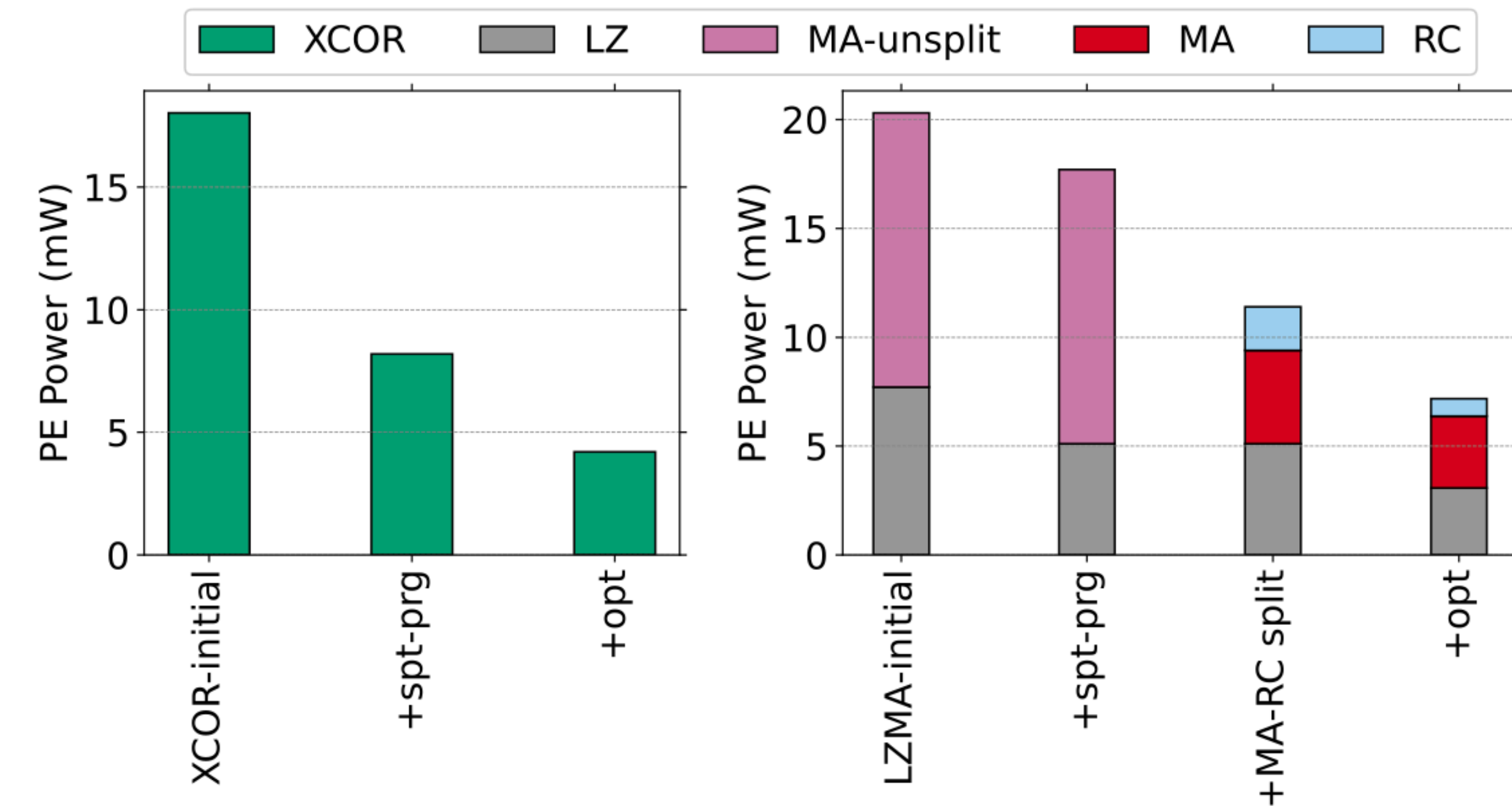
PE	Freq (MHz)	Logic (mW)		Mem (mW)		Total (mW)	Area (KGE)
		Leak	Dyn	Leak	Dyn		
LZ	129	0.055	1.455	0.095	1.466	3.071	55
LIC	22.5	0.057	0.267	0.006	0.046	0.376	25
MA	92	0.127	2.148	0.067	0.997	3.339	66
RC	90	0.029	0.763	0	0	0.792	12
DWT	3	0.004	0.002	0	0	0.006	2
NEO	3	0.012	0.003	0	0	0.015	5
FFT	15.7	0.057	0.509	0.085	0.356	1.007	22
XCOR	85	0.07	4.182	0.307	0.053	4.612	81
BBF	6	0.066	0.034	0	0	0.1	23
SVM	3	0.018	0.018	0.081	0.033	0.15	8
THR	16	0.002	0.011	0	0	0.013	1
GATE	5	0.003	0.006	0.067	0.054	0.13	17
AES	5	0.053	0.059	0	0	0.112	34
Tasks							
Compr (LZ4)		0.112	1.722	0.101	1.512	3.447	80
Compr (LZMA)		0.211	4.366	0.122	2.463	7.162	133
Compr (DWTMA)		0.16	2.913	0.0123	0.33	3.415	80
Seizure Prediction		0.216	4.760	0.54	0.496	6.012	111
Spike Det (NEO)		0.017	0.02	0.067	0.054	0.158	24
Spike Det (DWT)		0.009	0.019	0.067	0.054	0.149	20
Movement Intent		0.062	0.526	0.152	0.41	1.15	40
Encrypt (Raw)		0.053	0.059	0	0	0.112	34
RISC-V Control _{est}	25	0.341	0.137	0.248	1.080	1.800	70

Compression and seizure prediction: consume the most power

Power Consume of each pipeline under 15 mW

In general: higher operating frequency => higher dynamic power

Optimization of PE: Impact



For XCOR (left diagram):

- Before optimization: over 15 mW
- After: Spatial reprogramming saves 50% power

For LZMA (right diagram):

- Spatial reprogramming saves 1.5× power
- locality refactoring: reduces power further to 11.2mW

Conclusion

HARDWARE-SOFTWARE CO-DESIGN concept:

- Provides idea of refactoring the underlying algorithm of each task and implement each piece into a PE
- Provides idea of optimizing each PE in software level (spatial programming by XCOR PE)

HALO meets the safety constraint by realizing each task (under 15mW)

- Run **each** PE at **minimum clock frequency** catered to its **need** while ASICs run all logic at same frequency=> **saves power**
- Optimize each PE separately

HALO realizes many tasks (general-purpose architecture) , not a specific one like in ASIC

- For **each** task, controller **configures** required PEs into **pipeline** to execute it

HALO is extensible

- While realizing new BCI tasks: refactor algorithm, design PE for each piece, **add new PEs** into existing HALO architecture

Outline

- Background
- Problems
- Goals
- Key Mechanism
- Implementation
- Evaluation
- **Strengths**
- Weaknesses
- Discussion

Strengths

Design:

- Great improvement with respect to flexibility => wider BCI adoption
- Low power consumption => safe for chronic use
- Extensible (benefit from its modularity) => can support new tasks by adding PEs into current HALO directly

Paper:

- Can understand the paper without prior background knowledge about BCI
- Highlight the advantage of HALO against BCIs as ASIC through comparison
- Basic structure and concept of HALO well explained

Outline

- Background
- Problems
- Goals
- Key Mechanism
- Implementation
- Evaluation
- Strengths
- **Weaknesses**
- Discussion

Weaknesses

Paper:

- Problems /constraint that HALO meets: not mentioned in the paper => discussion point
- Lack of performance evaluation with respect to processing speed (time to execute a task)
 - Tables focus on evaluation of power consumption and achieved flexibility

Outline

- Background
- Problems
- Goals
- Key Mechanism
- Implementation
- Evaluation
- Strengths
- Weaknesses
- **Discussion**

Discussion

About HALO:

In the paper, author only claims : “ HALO meets a different set of constraints ”, but doesn’t explain it explicitly

Constraints that HALO meets / Problems of HALO/ Places that still need to be improved ?

Discussion

About HALO:

In the paper, author only claims : “ HALO meets a different set of constraints ”, but doesn’t explain it explicitly

Constraints that HALO meets / Problems of HALO/ Places that still need to be improved ?

- **Power constraint** is still a great limitation by designing since HALO belongs to implantable BCI (Key problem of implantable BCI)

Related work: <http://www.cs.yale.edu/homes/abhishek/abhishek-micro17.pdf>

- Need **finer grained** design than monolithic ASICs
- Does not solve the bottleneck problem of application of BCI in “From computer to brain” direction

Discussion

About HALO:

In the paper, author only claims : “ HALO meets a different set of constraints ”, but doesn’t explain it explicitly

Constraints that HALO meets / Problems of HALO/ Places that still need to be improved ?

- **Power constraint** is still a great limitation by designing since HALO belongs to implantable BCI (Key problem of implantable BCI)

Related work: <http://www.cs.yale.edu/homes/abhishek/abhishek-micro17.pdf>

- Need **finer grained** design than monolithic ASICs
- Does not solve the bottleneck problem of application of BCI in “From computer to brain” direction

About BCI in general:

Future prospects of brain-computer interface?

- Fields in which BCI can be applied?
- As headsets vs. implantable (as chips embedded on brain) : tradeoff ?
- Problems/Difficulties that BCI will meet ?

Thank you for Listening and Participating