Gzip Compression Using Altera OpenCL

Mohamed Abdelfattah *(University of Toronto)*

Andrei Hagiescu

Deshanand Singh

Gzip

- **Widely-used lossless compression program**
- **Gzip = LZ77 + Huffman**

- **Big data needs *fast* compression**

  Gigabyte-per-second

  - Lower disk space in data centers
  - Less power on communication networks

LZ77 Compression Example

- **This sentence is an easy sentence to compress.**

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- This sentence is an easy sentence to compress.

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- This sentence is an easy sentence to compress.

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

ALTERA
MEASURABLE ADVANTAGE™

LZ77 Compression Example

- **This sentence is an easy sentence to compress.**

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

ALTERA
MEASURABLE ADVANTAGE™

LZ77 Compression Example

- This sentence is an easy sentence to compress.

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

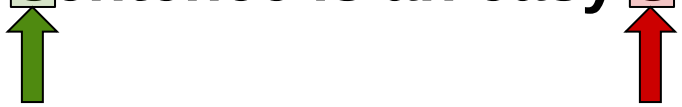
LZ77 Compression Example

- **This sentence is an easy sentence to compress.**

1. Scan file byte by byte
2. Look for matches
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- **This sentence is an easy sentence to compress.**

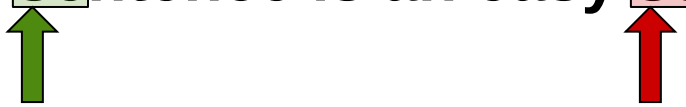


1. Scan file byte by byte
2. Look for matches
    1. Match length
    2. Match offset
3. Replace with a reference to previous occurrence

ALTERA
MEASURABLE ADVANTAGE™

LZ77 Compression Example

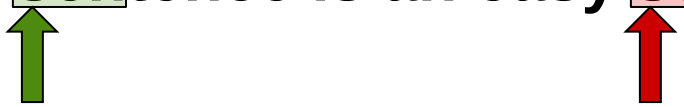
- **This sentence is an easy sentence to compress.**



1. Scan file byte by byte
2. Look for matches
    1. Match length = 2
    2. Match offset
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- This sentence is an easy sentence to compress.



1. Scan file byte by byte
2. Look for matches
    1. Match length = 3
    2. Match offset
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- This sentence is an easy sentence to compress.

1. Scan file byte by byte
2. Look for matches
    1. Match length = 8
    2. Match offset
3. Replace with a reference to previous occurrence

LZ77 Compression Example

- This sentence is an easy sentence to compress.

1. Scan file byte by byte
2. Look for matches
    1. Match length = 8
    2. Match offset = 20
3. Replace with a reference to previous occurrence

LZ77 Compression Example

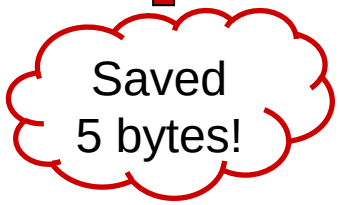
- This sentence is an easy @(8,20) to compress.

1. Scan file byte by byte
2. Look for matches
   - Match length = 8
   - Match offset = 20
3. Replace with a reference to previous occurrence
   - Marker, length, offset

ALTERA
MEASURABLE ADVANTAGE™

LZ77 Compression Example

- **This sentence is an easy sentence to compress.**
- **This sentence is an easy @(8,20) to compress.**



1. Scan file byte by byte
2. Look for matches
   - Match length = 8
   - Match offset = 20
3. Replace with a reference to previous occurrence
   - Marker, length, offset

ALTERA
MEASURABLE ADVANTAGE™

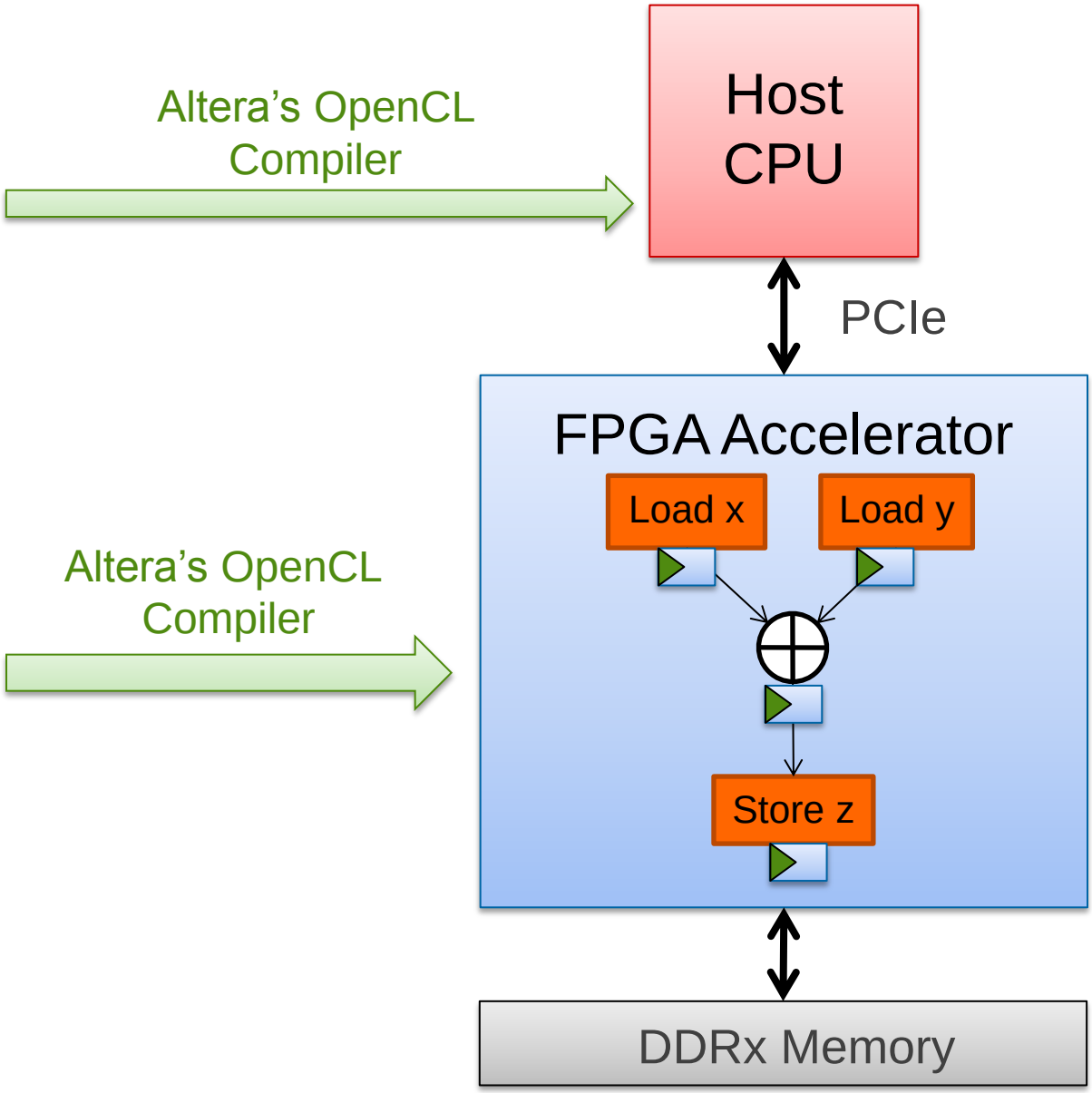
Altera OpenCL Compiler for FPGAs

Host Code

```
//host code
//Enqueue buffer
//Enqueue Kernel(s)
//dequeue buffers
…
```

OpenCL Single-threaded Code

```
void kernel
simple(global int *input,
                int size,
       global int *output)
{
  for(i=1..size)
  {
    int x = input[i];
    int y = input[i+1];
    int z = x + y;
    output[i] = z;
  }
}
```



Altera OpenCL Compiler for FPGAs

Host Code

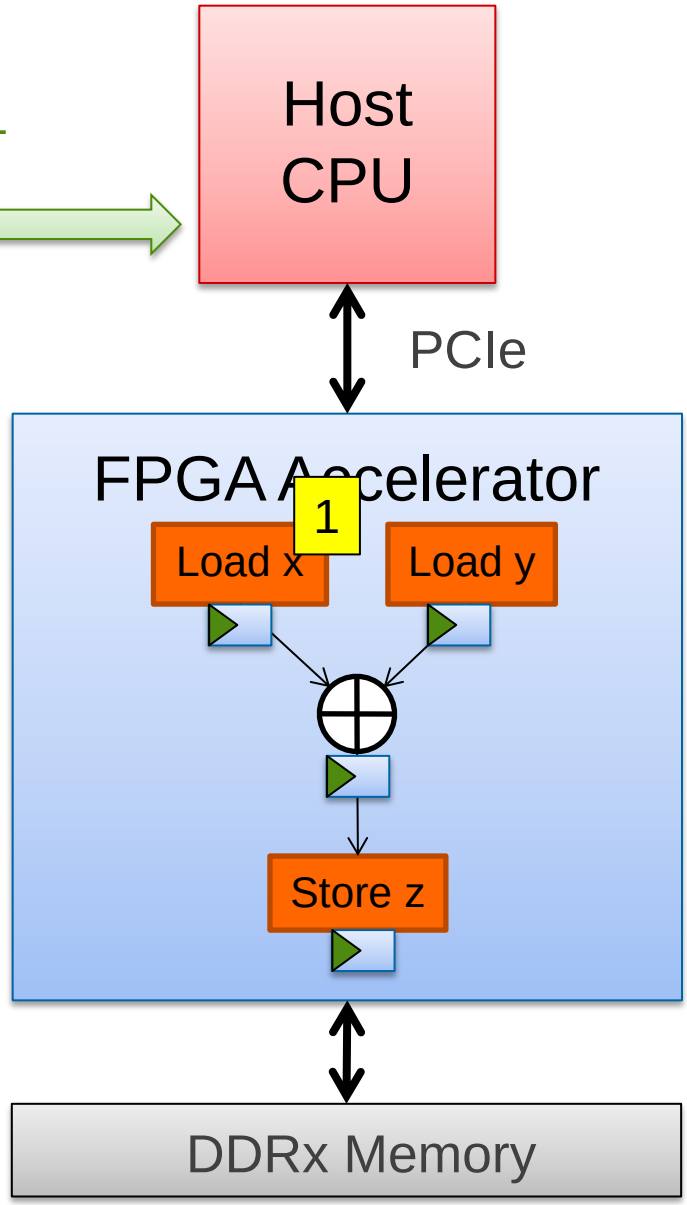
```
//host code
//Enqueue buffer
//Enqueue Kernel(s)
//dequeue buffers
…
```

Altera's OpenCL Compiler

OpenCL Single-threaded Code

```
void kernel
simple(global int *input,
               int size,
       global int *output)
{
  for(i=1..size)
  {
    int x = input[i];
    int y = input[i+1];
    int z = x + y;
    output[i] = z;
  }
}
```

Altera's OpenCL Compiler



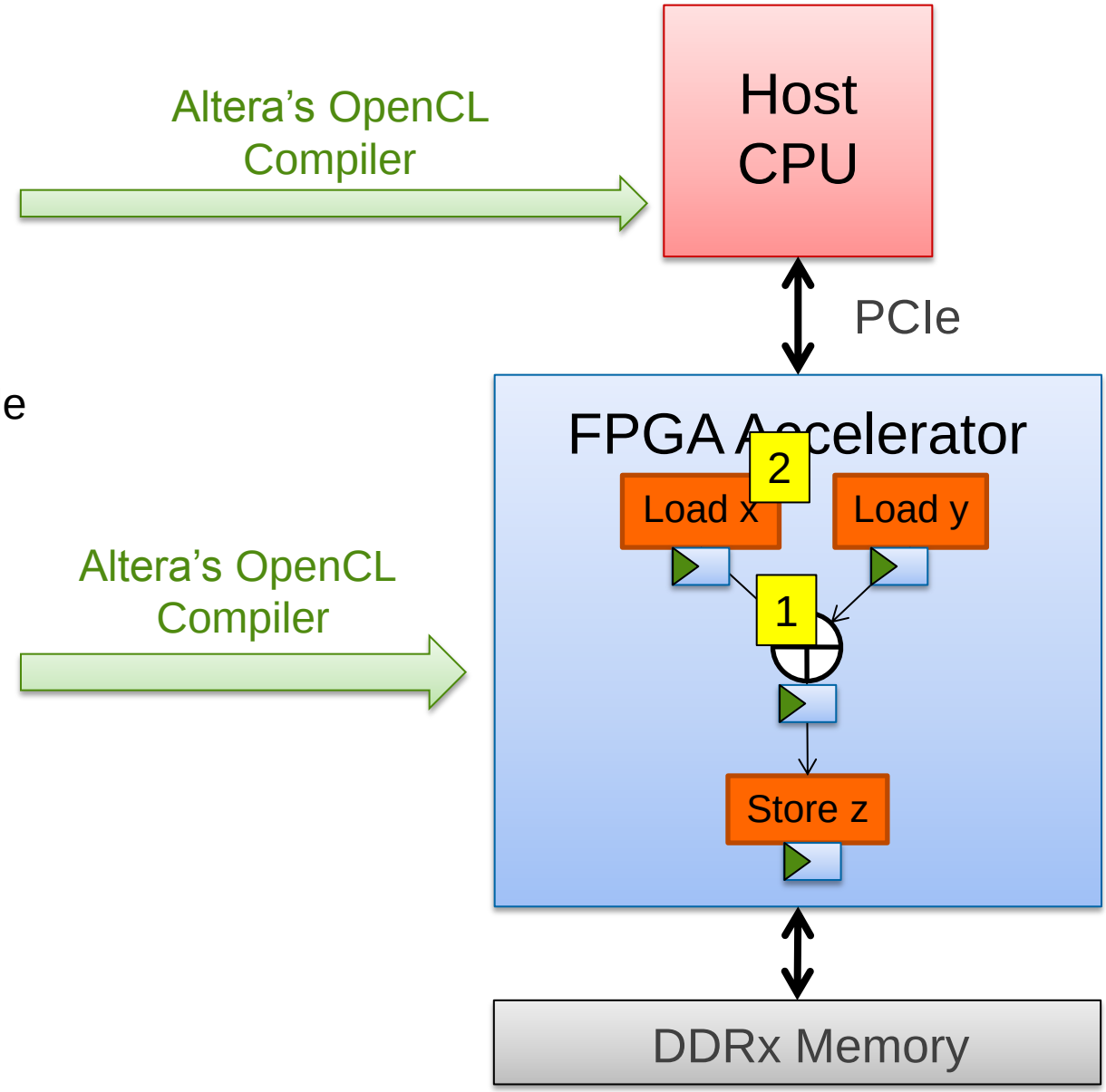
Altera OpenCL Compiler for FPGAs

Host Code

```
//host code
//Enqueue buffer
//Enqueue Kernel(s)
//dequeue buffers
…
```

OpenCL Single-threaded Code

```
void kernel
simple(global int *input,
                int size,
       global int *output)
{
  for(i=1..size)
  {
    int x = input[i];
    int y = input[i+1];
    int z = x + y;
    output[i] = z;
  }
}
```



Altera OpenCL Compiler for FPGAs

Host Code

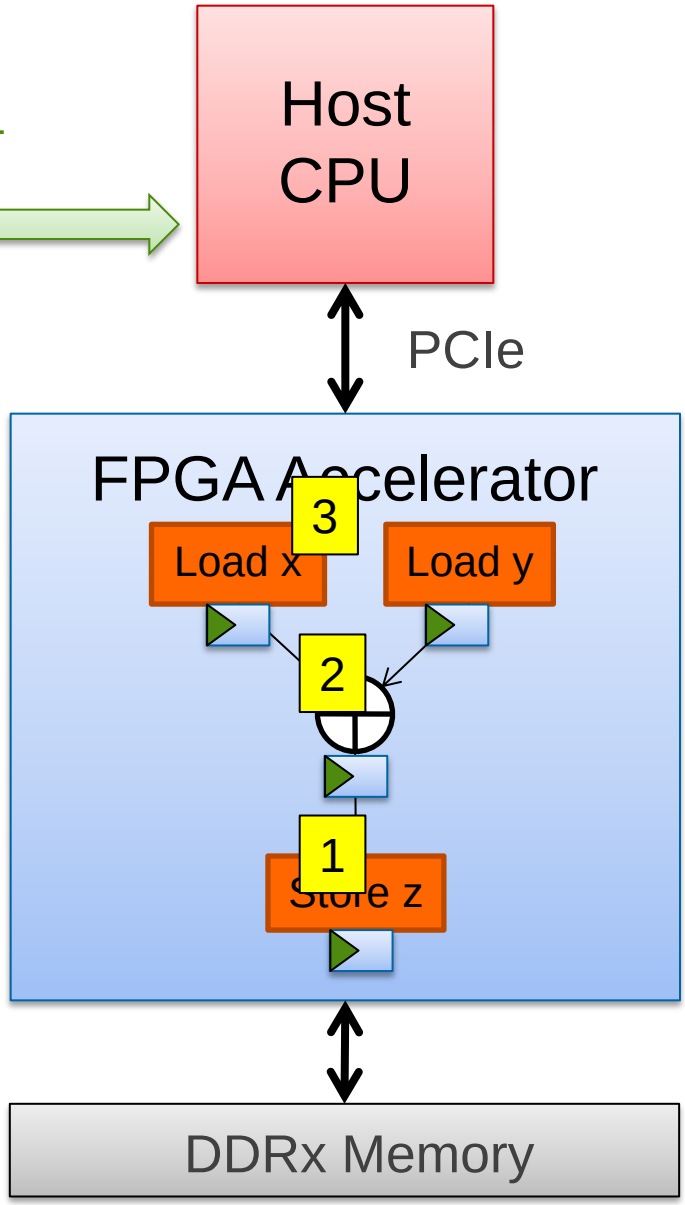
```
//host code
//Enqueue buffer
//Enqueue Kernel(s)
//dequeue buffers
…
```

Altera's OpenCL Compiler

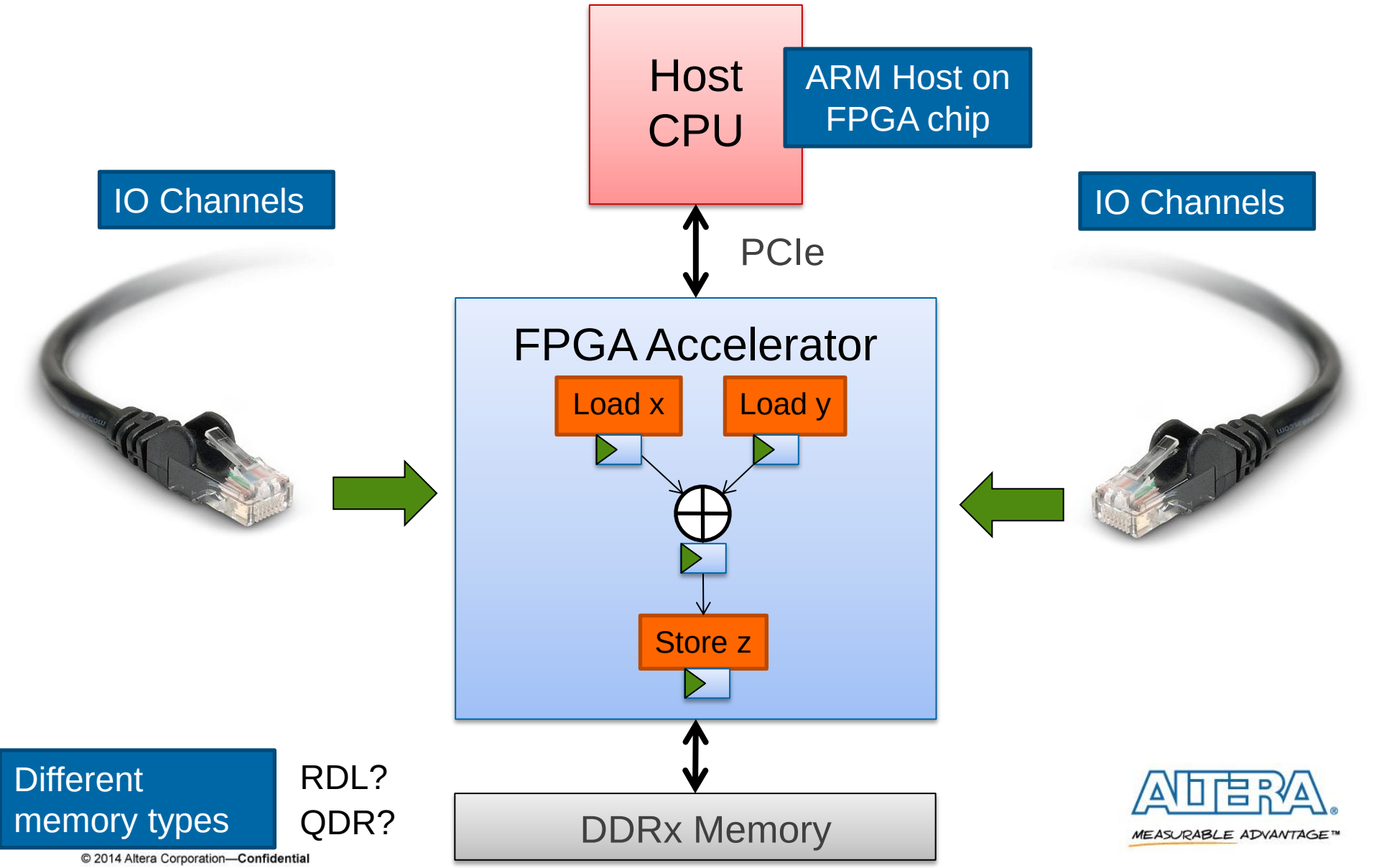
OpenCL Single-threaded Code

```
void kernel
simple(global int *input,
               int size,
       global int *output)
{
  for(i=1..size)
  {
    int x = input[i];
    int y = input[i+1];
    int z = x + y;
    output[i] = z;
  }
}
```

Altera's OpenCL Compiler



FPGAs can be VERY Custom



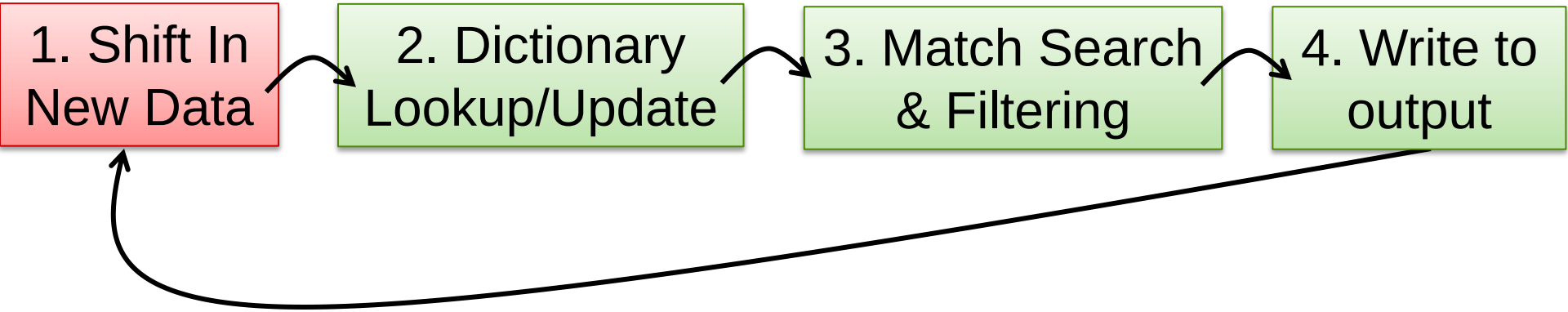
Different memory types

RDL?

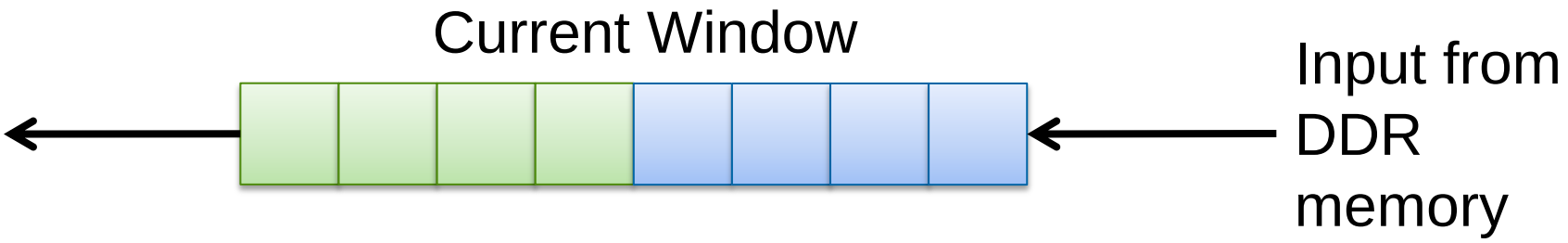
QDR?

ALTERA
MEASURABLE ADVANTAGE™

Implementation Overview

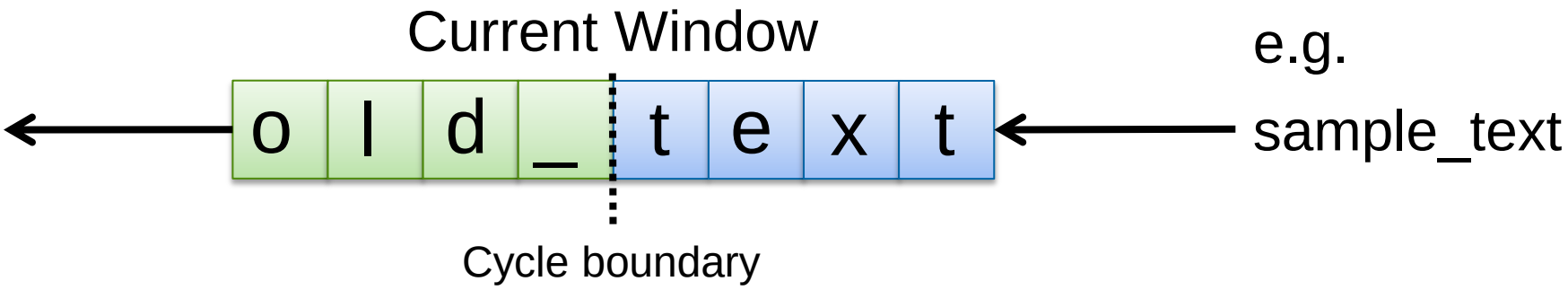


1. Shift In New Data



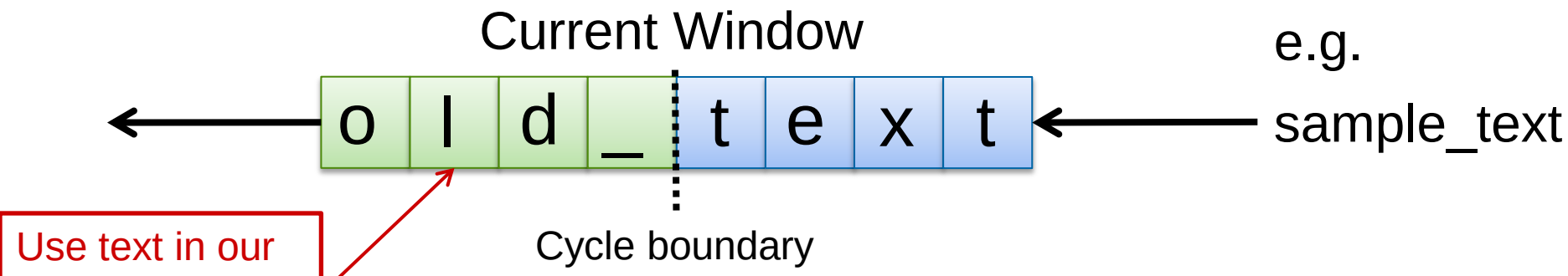

```
//shift current window
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[i] = current_window[i+VEC];

//load in new data
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[VEC+i] = input[inpos+i];
```

1. Shift In New Data




```
//shift current window
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[i] = current_window[i+VEC];

//load in new data
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[VEC+i] = input[inpos+i];
```

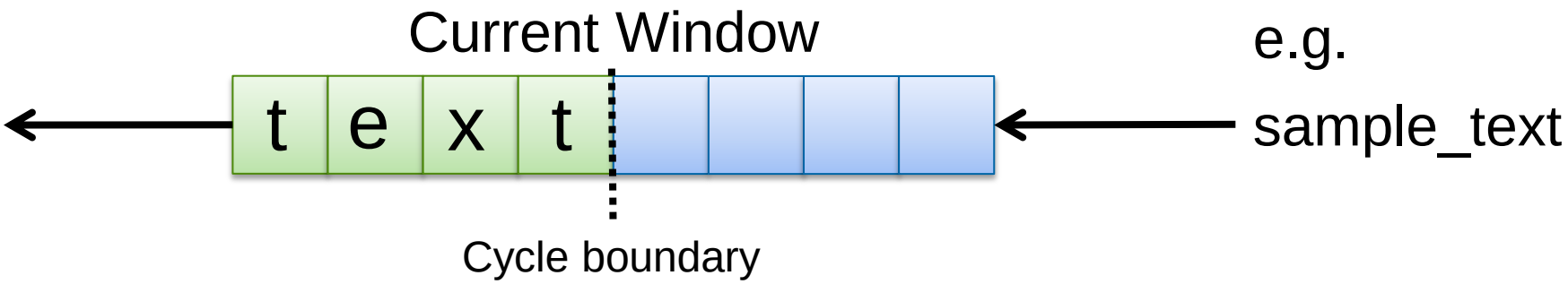
1. Shift In New Data




```
//shift current window
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[i] = current_window[i+VEC];

//load in new data
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[VEC+i] = input[inpos+i];
```

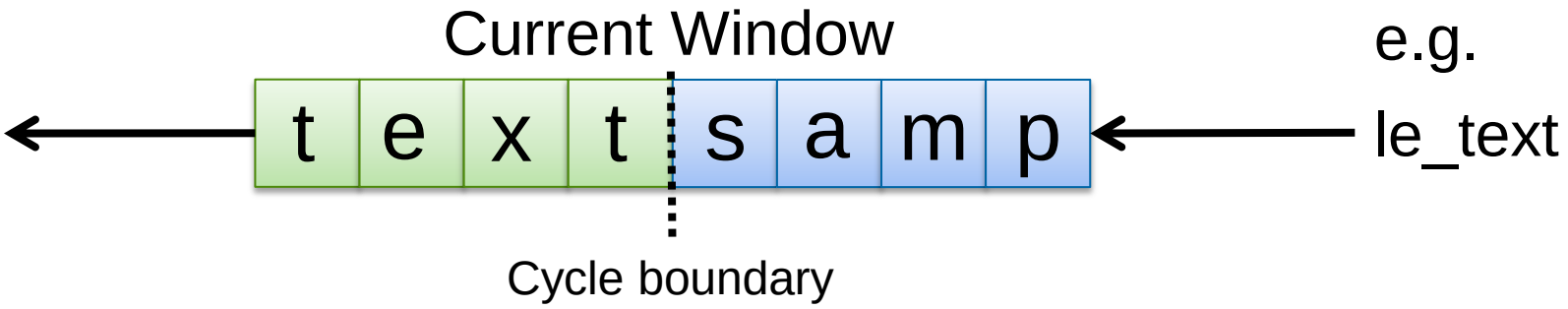
VEC = 4

1. Shift In New Data



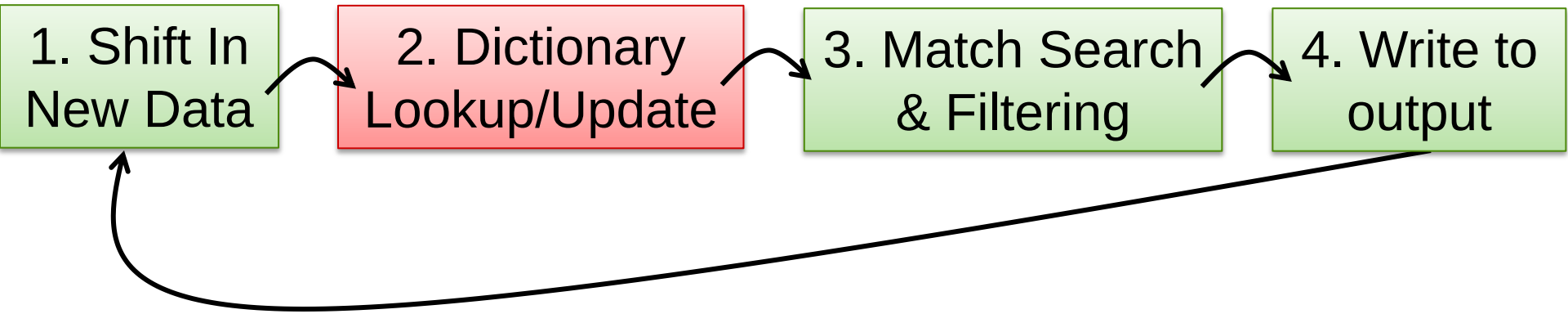

```
//shift current window
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[i] = current_window[i+VEC];

//load in new data
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[VEC+i] = input[inpos+i];
```

1. Shift In New Data

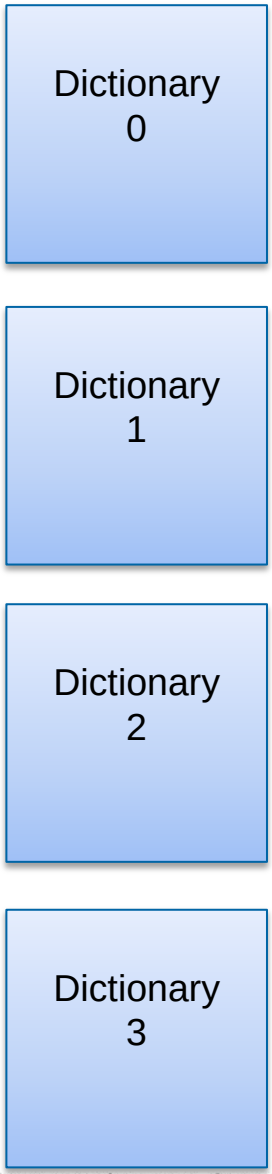



```
//shift current window
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[i] = current_window[i+VEC];

//load in new data
#pragma unroll
for(char i = 0; i < VEC; i++)
    current_window[VEC+i] = input[inpos+i];
```

Implementation Overview



2. Dictionary Lookup/Update



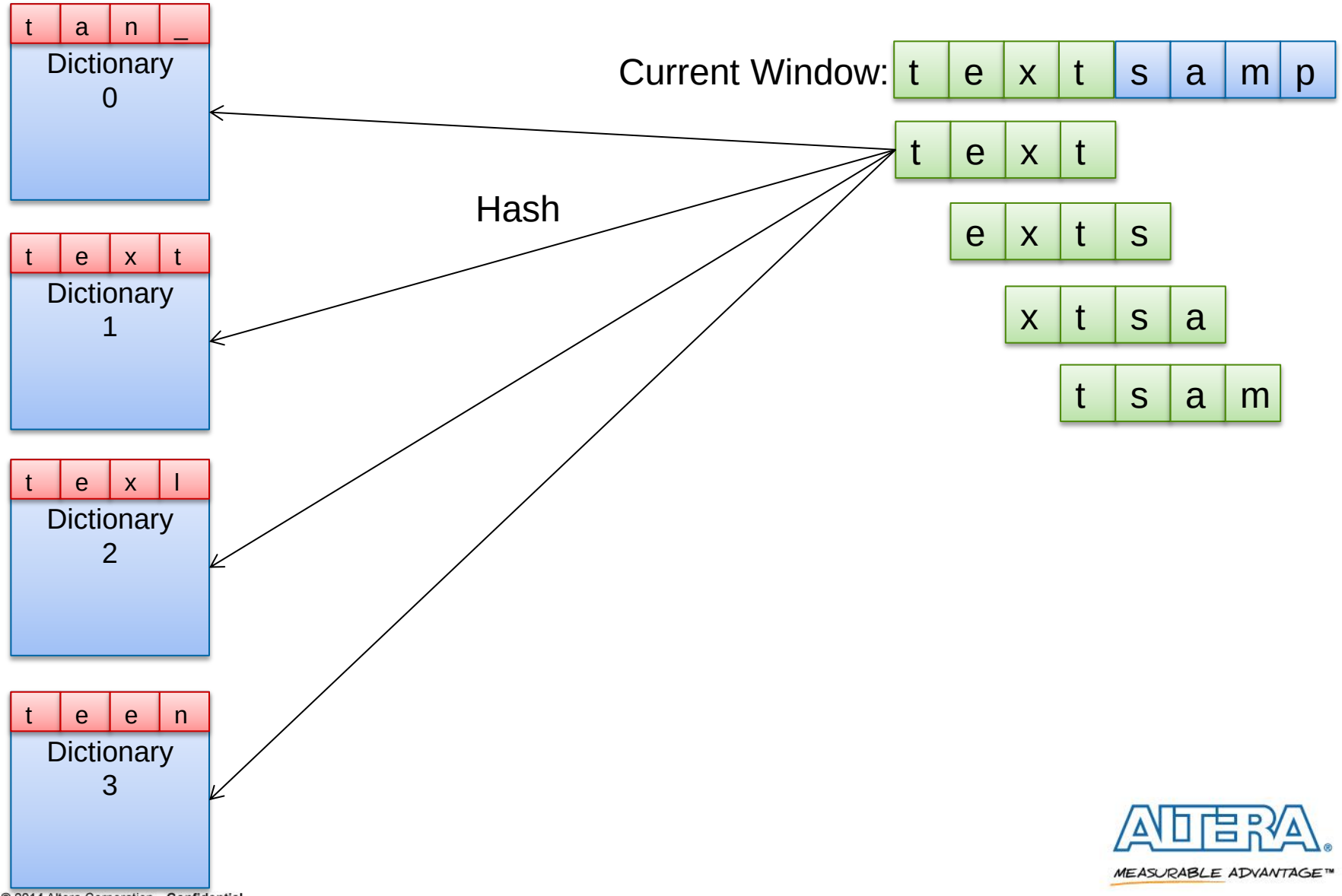
Current Window: t e x t s a m p

1. **Compute hash**
2. **Look for match in 4 dictionaries**
3. **Update dictionaries**

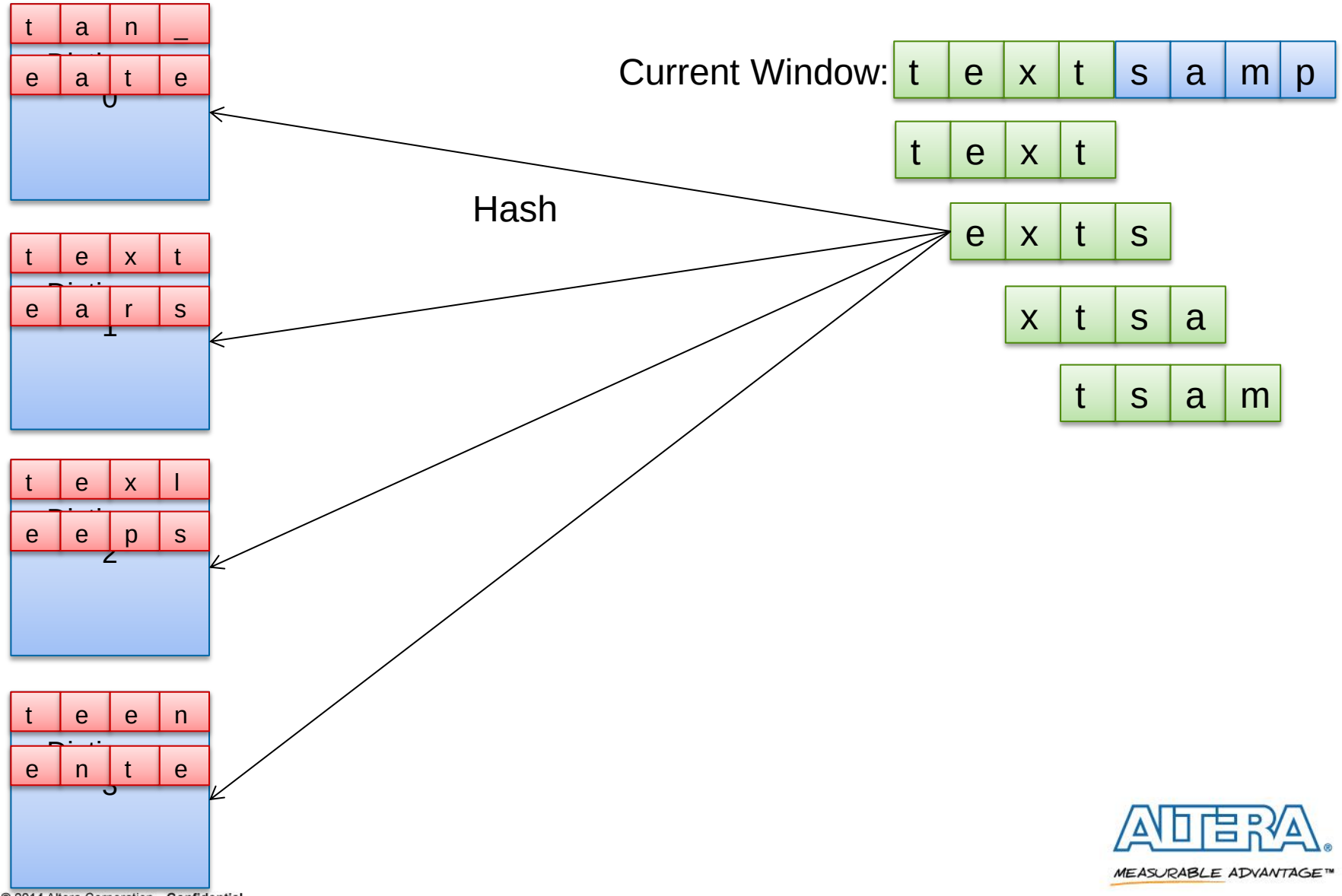
Dictionaries buffer the text that we have already processed, e.g.:

- This sentence is an easy sentence to compress.

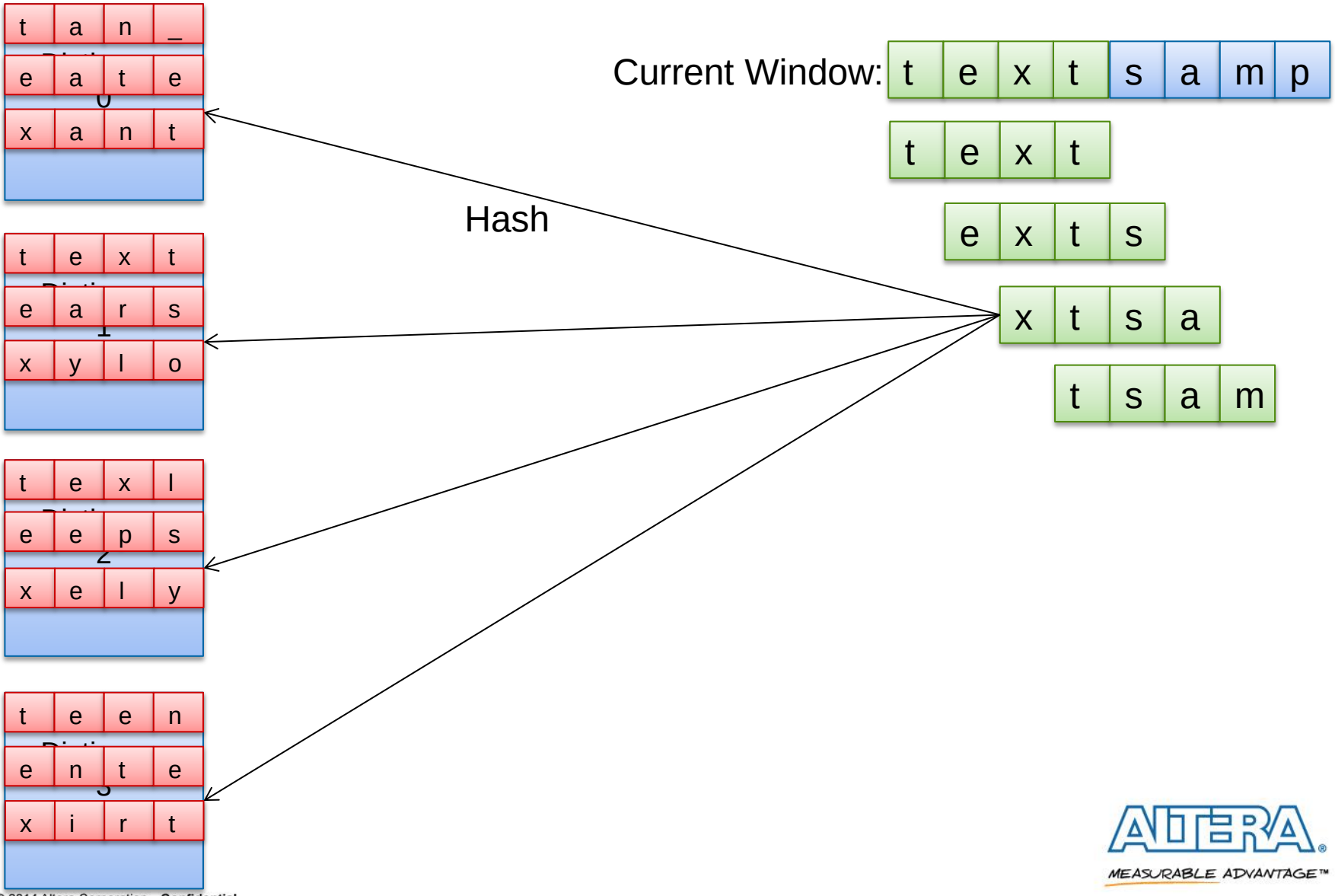
2. Dictionary Lookup/Update



2. Dictionary Lookup/Update

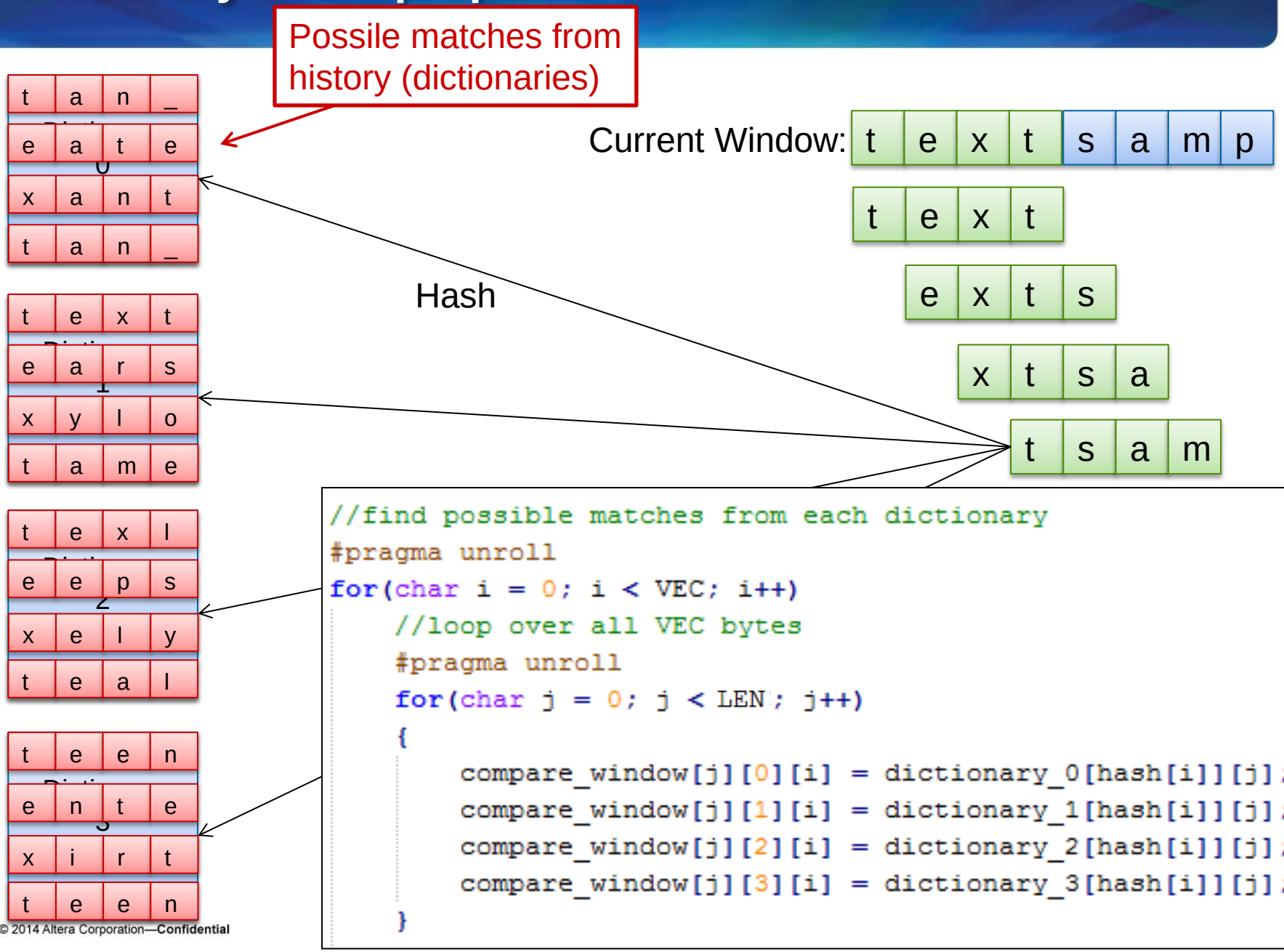


2. Dictionary Lookup/Update



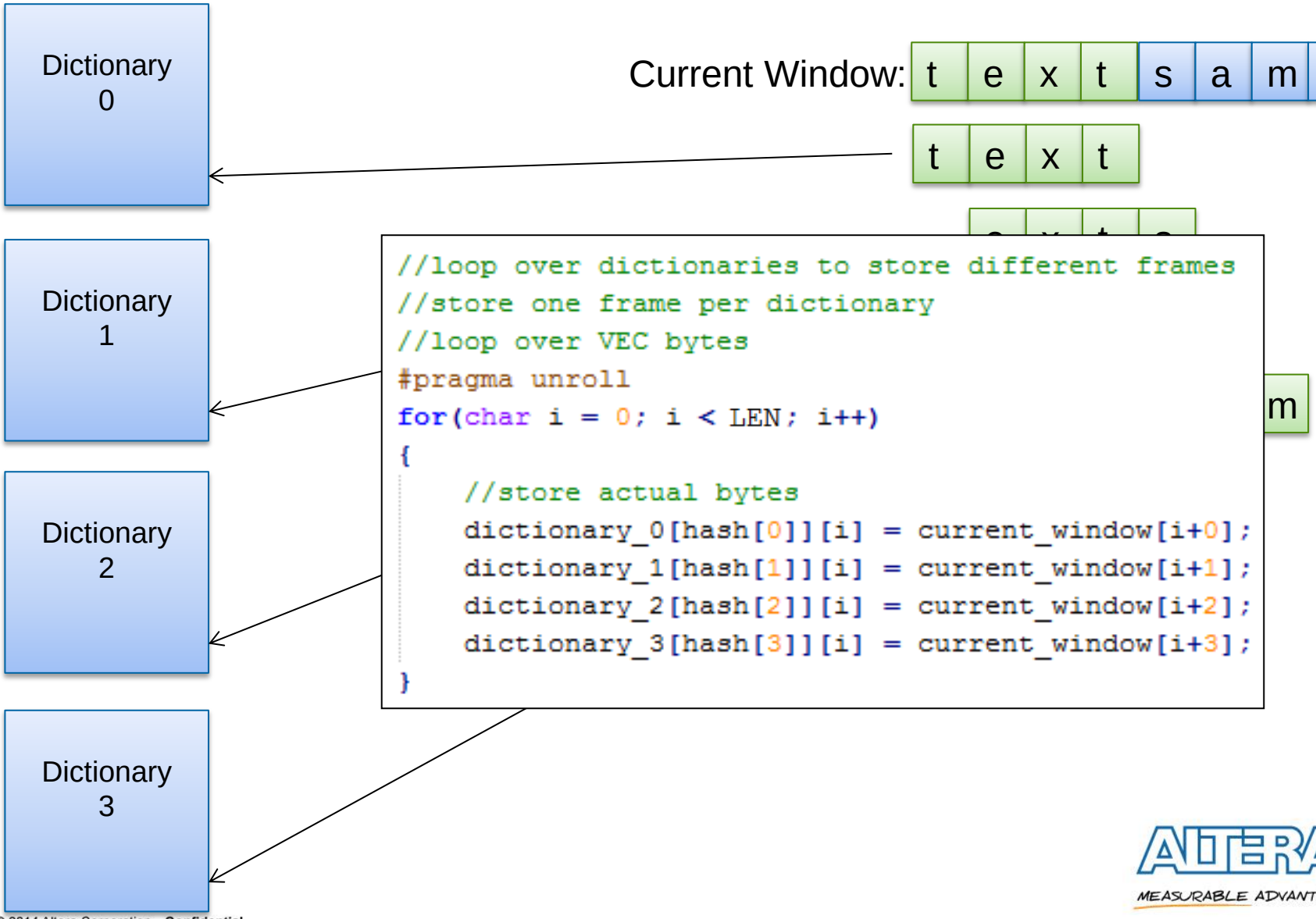
ALTERA
MEASURABLE ADVANTAGE™

2. Dictionary Lookup/Update

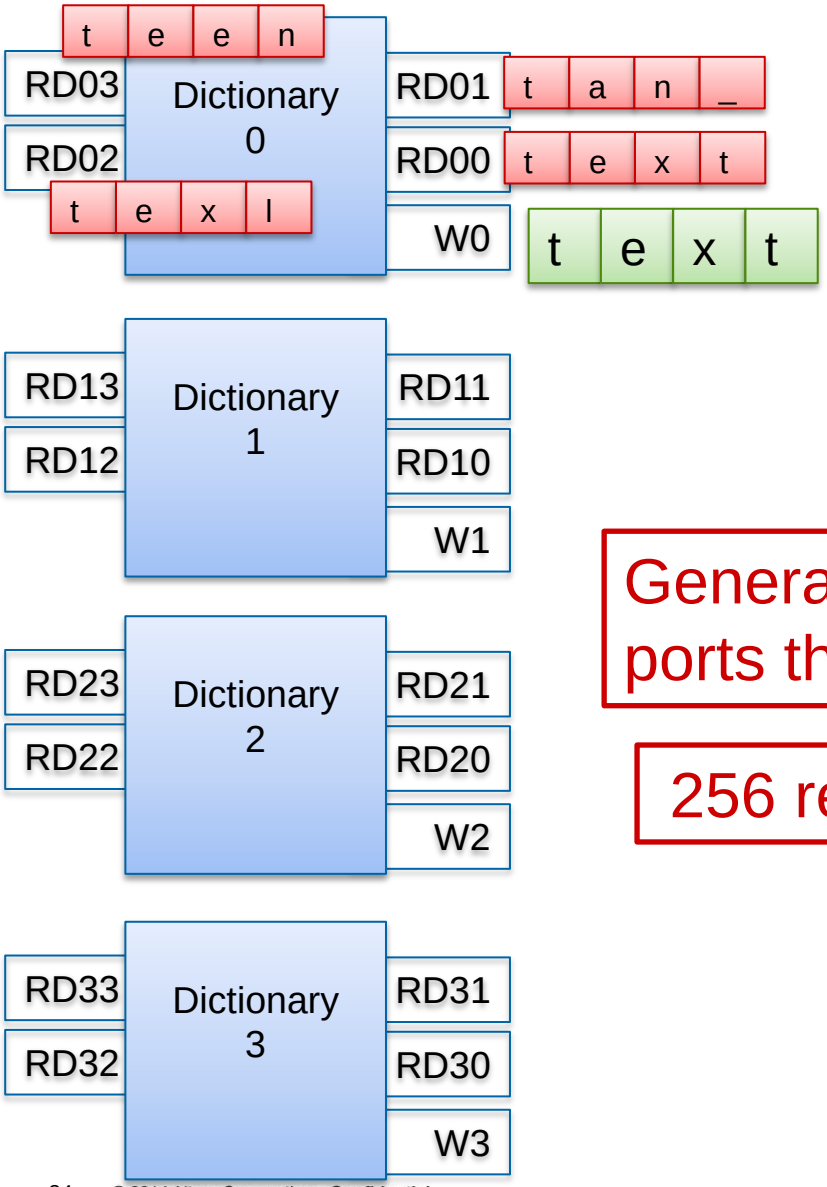



```
//find possible matches from each dictionary
#pragma unroll
for(char i = 0; i < VEC; i++)
    //loop over all VEC bytes
    #pragma unroll
    for(char j = 0; j < LEN; j++)
    {
        compare_window[j][0][i] = dictionary_0[hash[i]][j];
        compare_window[j][1][i] = dictionary_1[hash[i]][j];
        compare_window[j][2][i] = dictionary_2[hash[i]][j];
        compare_window[j][3][i] = dictionary_3[hash[i]][j];
    }
```

2. Dictionary Lookup/Update



2. Dictionary Lookup/Update

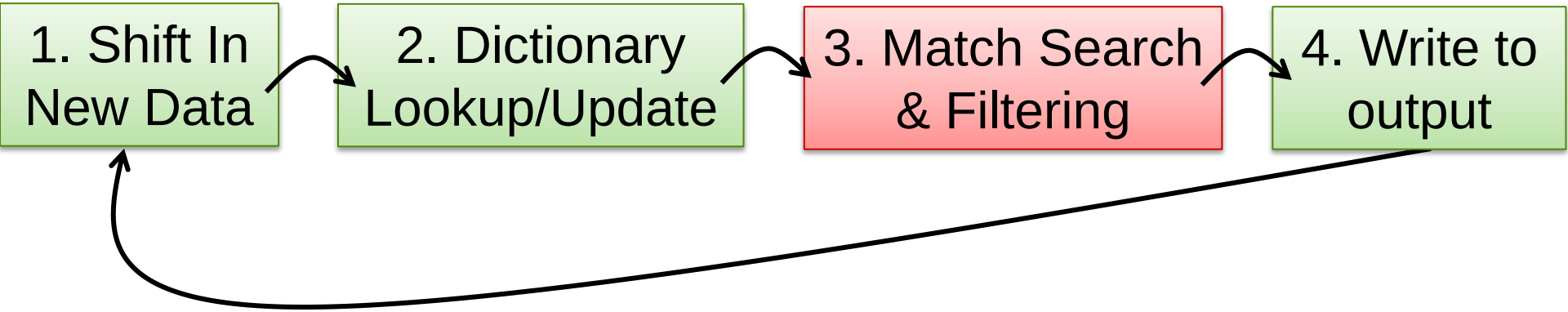


Current Window: t e x t s a m p

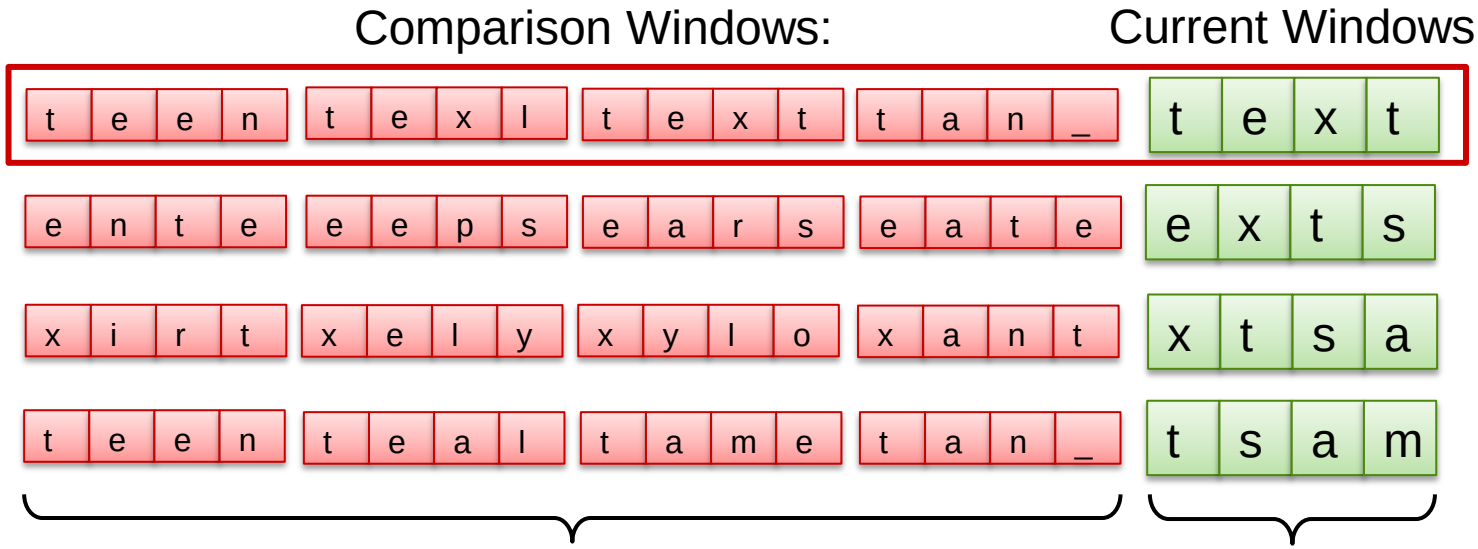
Generate exactly the number of read/write ports that we need and the width

256 read ports, 16 write ports – 128 bits

Implementation Overview

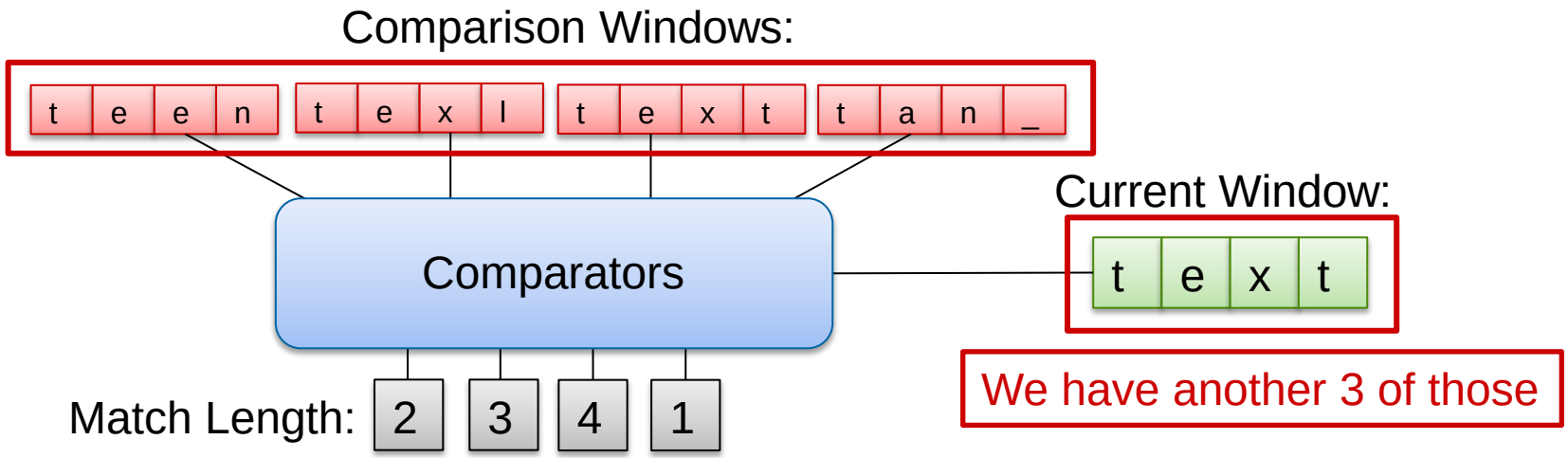


3. Match Search & Filtering



Compare current window against each of its 4 compare windows

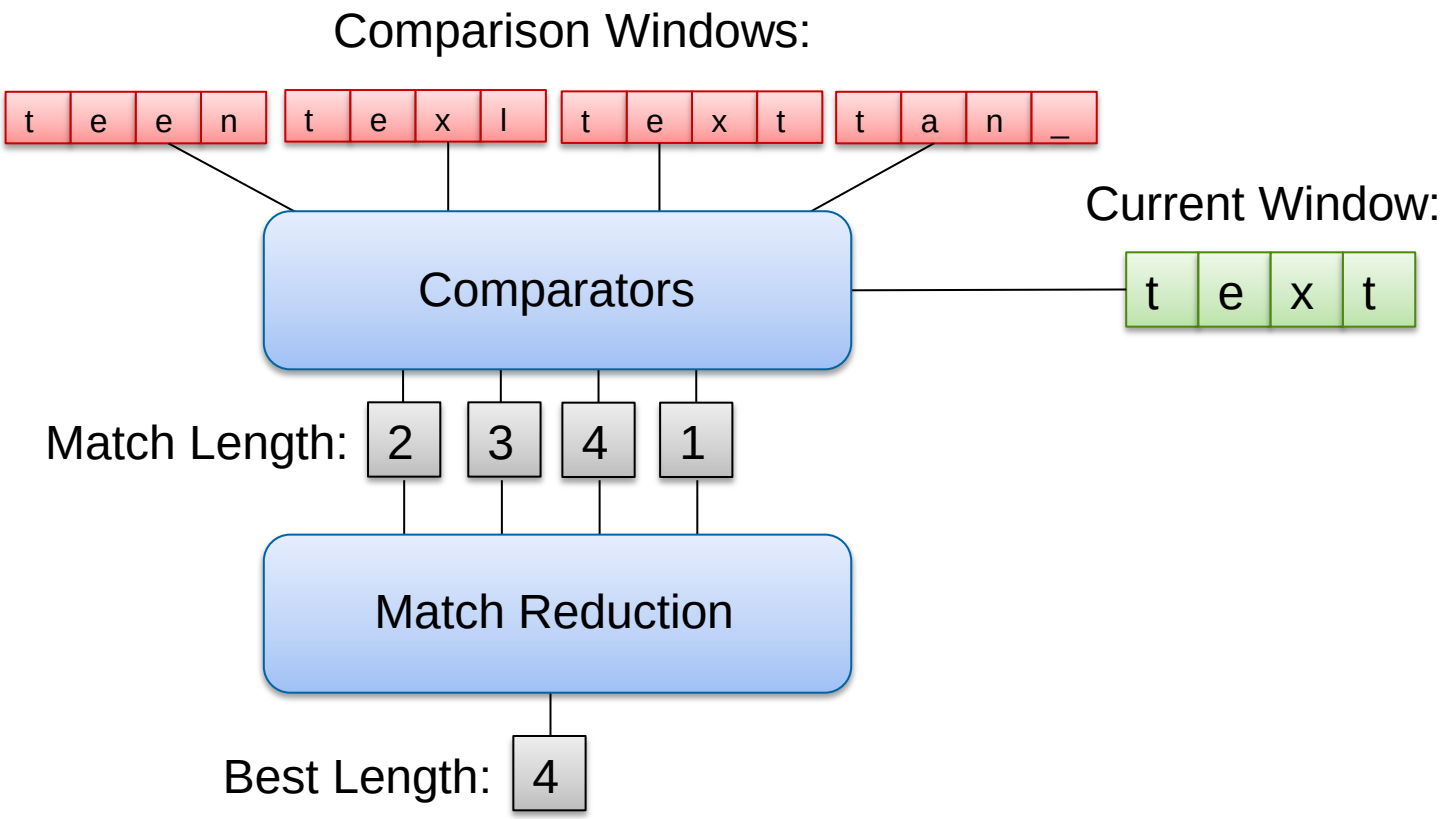
3. Match Search & Filtering




```
//loop over each comparison window frame -- one comes from each dictionary
#pragma unroll
for(char i = 0; i < VEC; i++ )
{
    //loop over each current window
    #pragma unroll
    for(char j = 0; j < VEC ; j++)
    {
        //loop over each byte in current & comparison windows
        #pragma unroll
        for(char k = 0; k < LEN ; k++)
        {
            if(current_window[j+k] == compare_window[k][i][j] && !done)
                length[j] ++;
            else
                done = 1;
```

Compare each byte

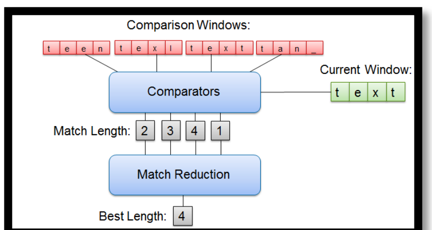
3. Match Search & Filtering

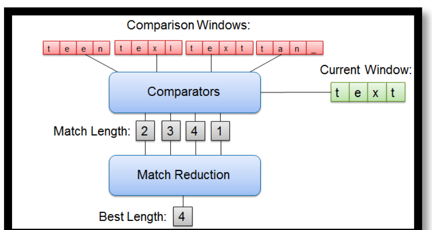


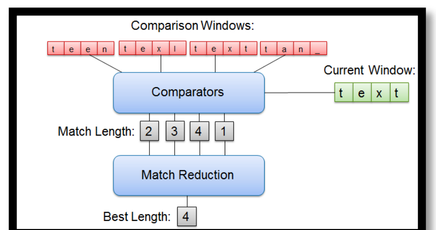

```
    //is this the best length?
    #pragma unroll
    for(char m = 0; m < VEC; m++)
    {
        bestlength[m] = (length[m] > bestlength[m]) ? length[m] : bestlength[m];
    }
}
```

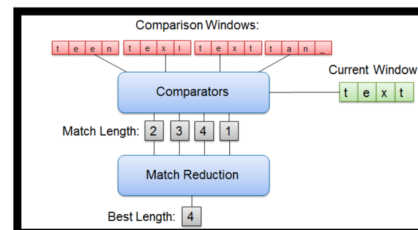
3. Match Search & Filtering

```
//loop over each comparison window frame -- one comes from each dictionary
#pragma unroll
for(char i = 0; i < VEC; i++ )
{
    //loop over each current window
    #pragma unroll
    for(char j = 0; j < VEC ; j++)
    {
        //loop over each byte in current & comparison windows
        #pragma unroll
        for(char k = 0; k < LEN ; k++)
```










```
        }
    }

    //is this the best length?
    #pragma unroll
    for(char m = 0; m < VEC; m++)
    {
        bestlength[m] = (length[m] > bestlength[m]) ? length[m] : bestlength[m];
    }
}
```

3. Match Search & Filtering

```
//loop over each comparison window frame -- one comes from each dictionary
#pragma unroll
for(char i = 0; i < VEC; i++ )
{
    //loop over each current window
    #pragma unroll
    for(char j = 0; j < VEC ; j++)
    {
        //loop over each byte in current & comparison windows
        #pragma unroll
        for(char k = 0; k < LEN ; k++)
        {
            if(current_window[j+k] == compare_window[k][i][j] && !done)
                length[j] ++;
            else
                done = 1;
        }
    }

    //is this the best length?
    #pragma unroll
    for(char m = 0; m < VEC; m++)
    {
        bestlength[m] = (length[m] > bestlength[m]) ? length[m] : bestlength[m];
    }
}
```

3. Match Search & Filtering

```
#pragma unroll
for(char k = 0; k < LEN ; k++)
{
    if(current_window[j+k] == compare_window[k][i][j] && !done)
        length[j] ++;
    else
        done = 1;
}
```

3. Match Search & Filtering

Typical C-code

```
while(current_window[j+k] == comparison_window[k][i][j])
{
    length[j]++;
}
```

```
Full unrolling of the loop is requested but the loop bounds cannot be determined.
```

```
#pragma unroll
for(char k = 0; k < LEN ; k++)
{
    if(current_window[j+k] == compare_window[k][i][j] && !done)
        length[j] ++;
    else
        done = 1;
}
```

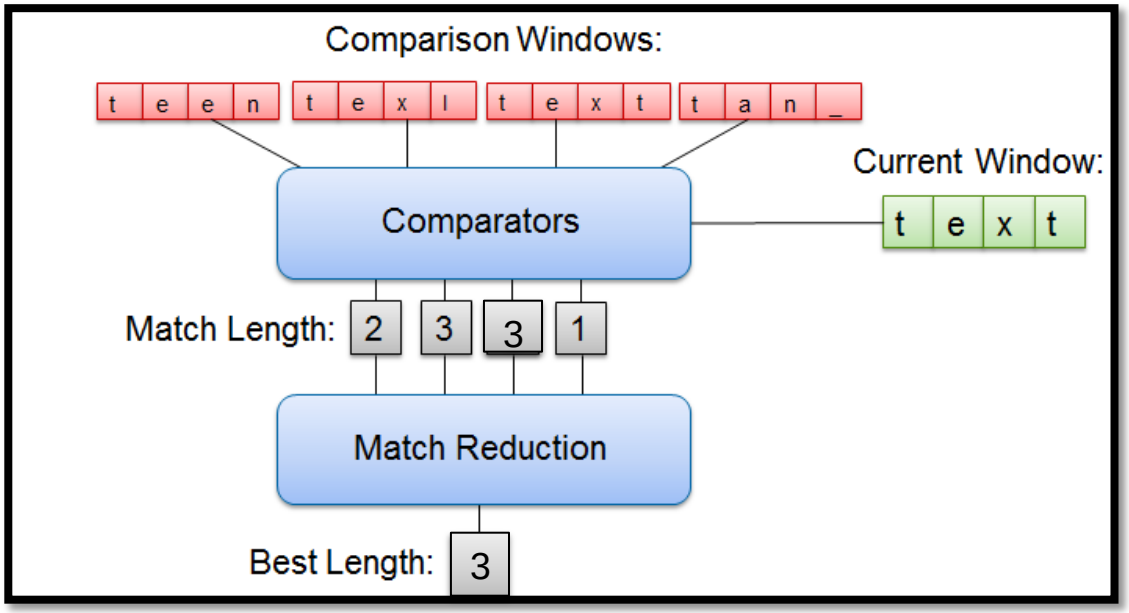
Fixed loop bounds – compiler can unroll loop

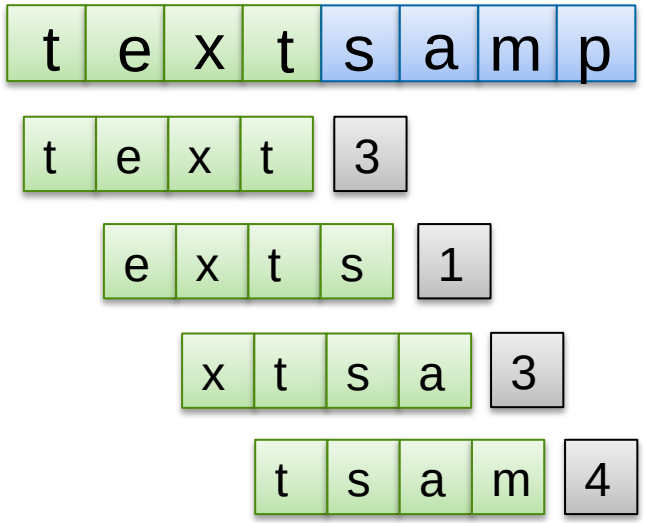
```
Compiler Warning: removing out-of-bounds accesses to
```

ALTERA
MEASURABLE ADVANTAGE™

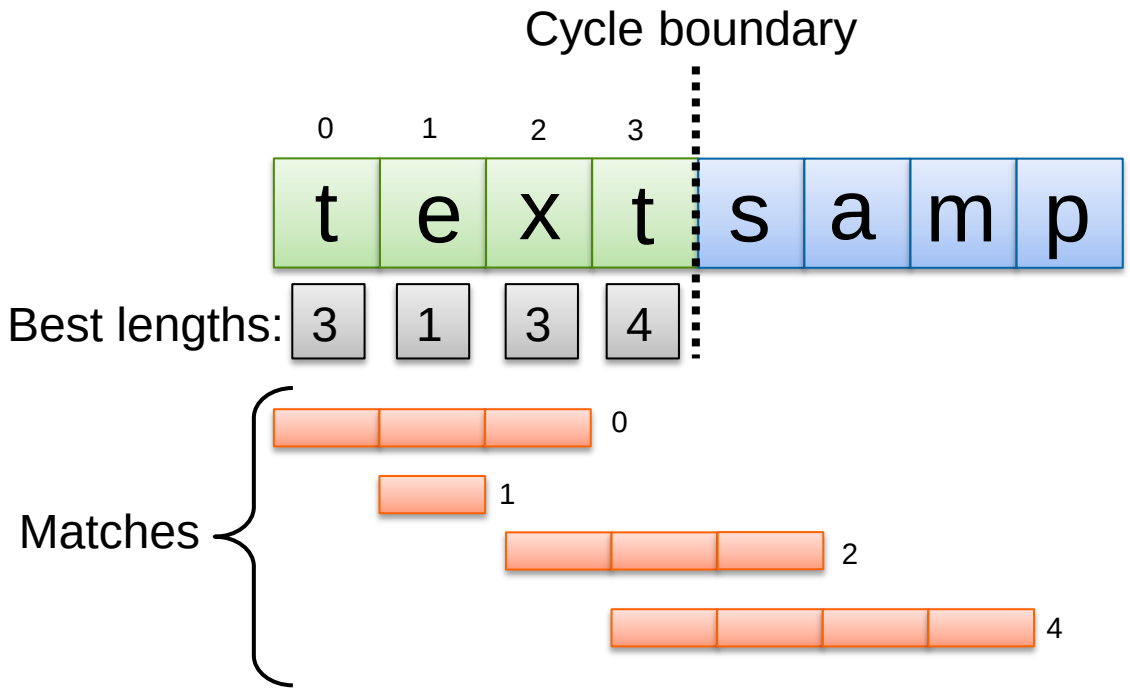
3. Match Search & Filtering

- **One bestlength associated with each current_window**





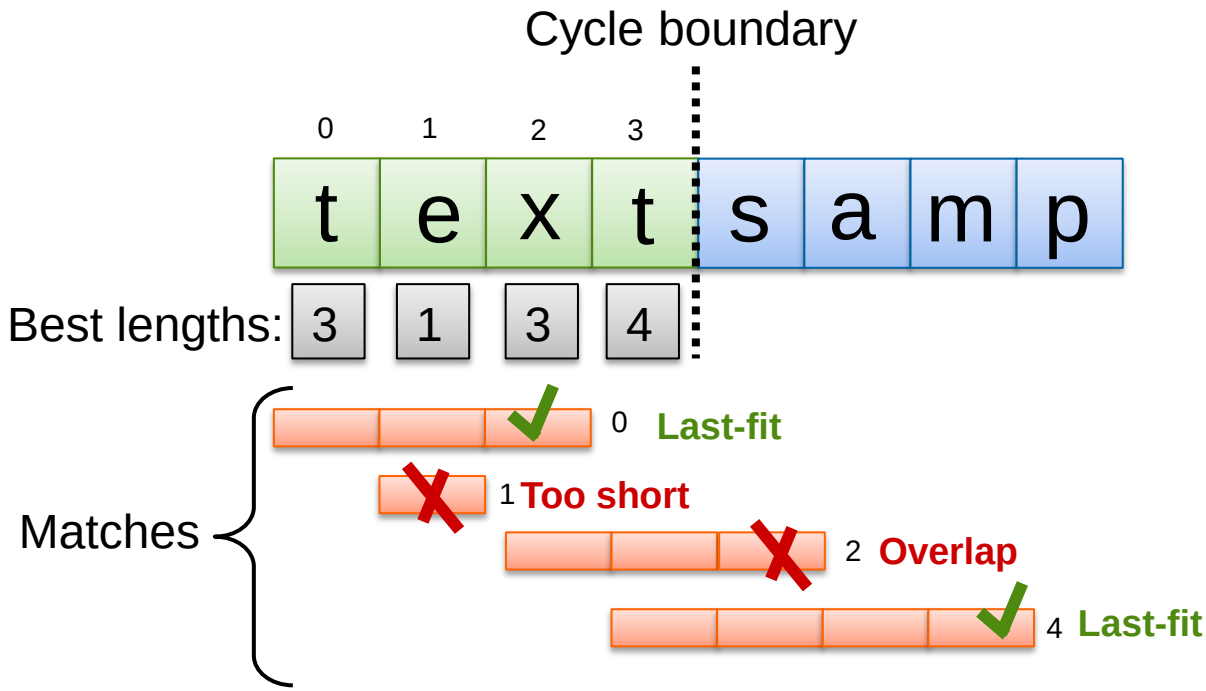
3. Match Search & Filtering



Select the best combination of matches from the set of candidate matches

1. Remove matches that are longer when encoded than original
2. From the remaining set; select the best ones
   - (heuristic for bin-packing) → last-fit
3. Compute “first valid position” for next step

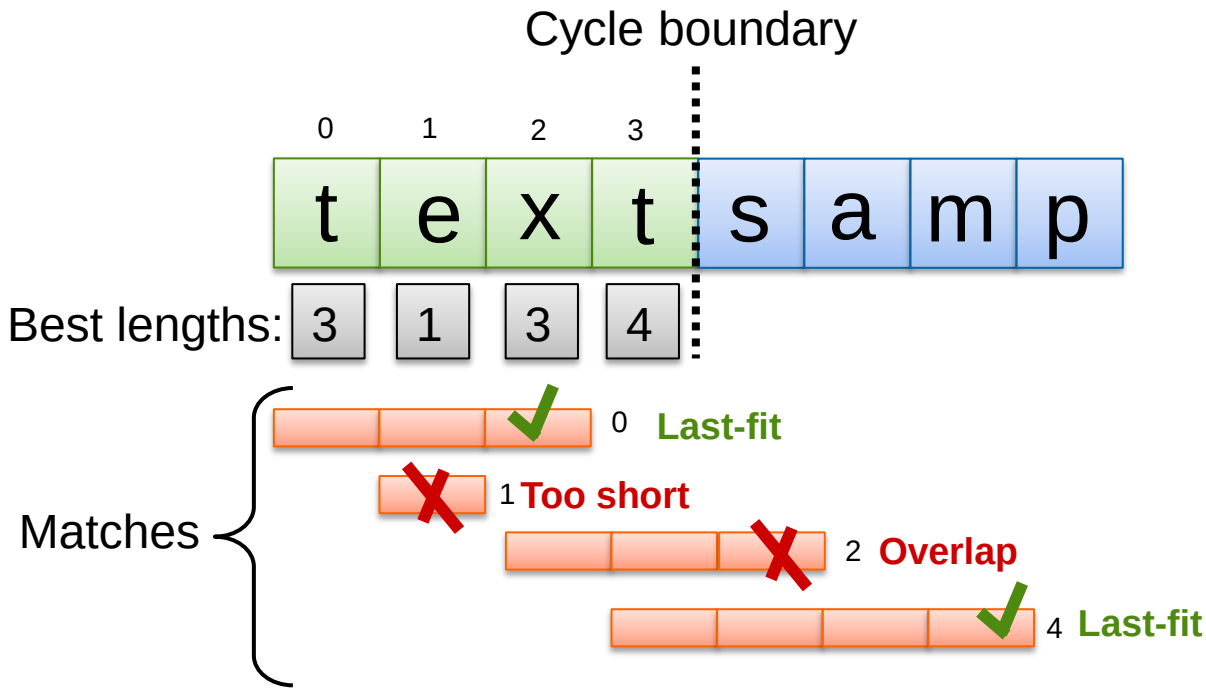
3. Match Search & Filtering



Select the best combination of matches from the set of candidate matches

1. Remove matches that are longer when encoded than original
2. From the remaining set; select the best ones
   - (heuristic for bin-packing) → last-fit
3. Compute “first valid position” for next step

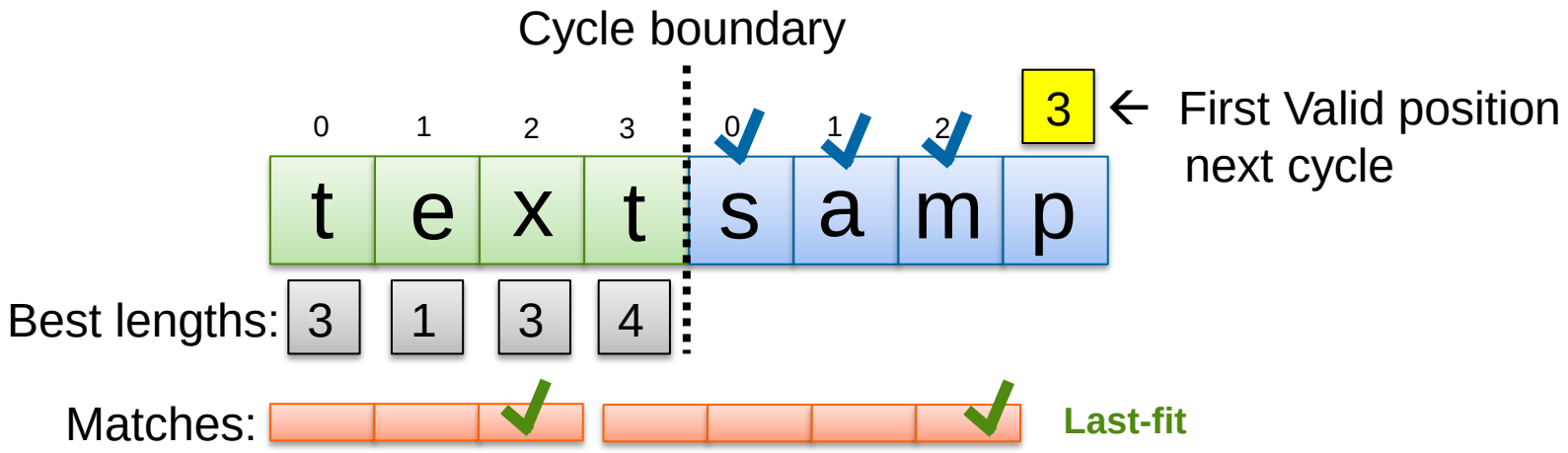
3. Match Search & Filtering



Select the best combination of matches from the set of candidate matches

1. Remove matches that are longer when encoded than original
2. From the remaining set; select the best ones
   - (heuristic for bin-packing) → last-fit
3. Compute “first valid position” for next step

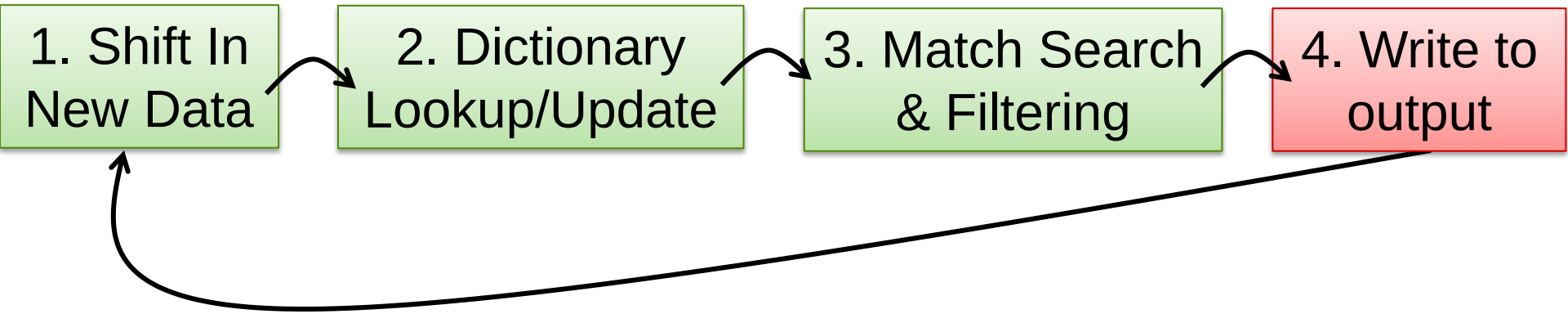
3. Match Search & Filtering



Select the best combination of matches from the set of candidate matches

1. Remove matches that are longer when encoded than original
2. From the remaining set; select the best ones
   - (heuristic for bin-packing) → last-fit
3. Compute “first valid position” for next step

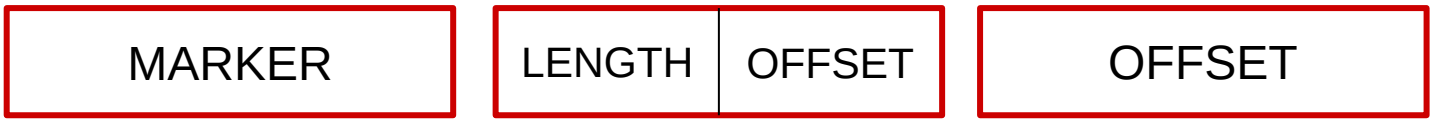
Implementation Overview



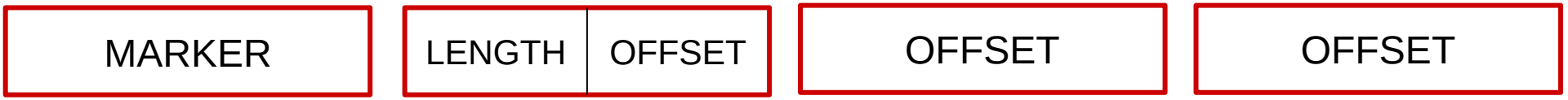
4. Writing to Output

- This sentence is an easy sentence to compress.
- This sentence is an easy @(8,20) to compress.

- **Marker, length, offset**
  - Length is limited by VEC (=16 in our case) – *fits in 4 bits*
  - Offset is limited by 0x40000 (doesn't make sense to be more) – *fits in 21 bits*
- **Use either 3 or 4 bytes for this:**
  - Offset < 2048



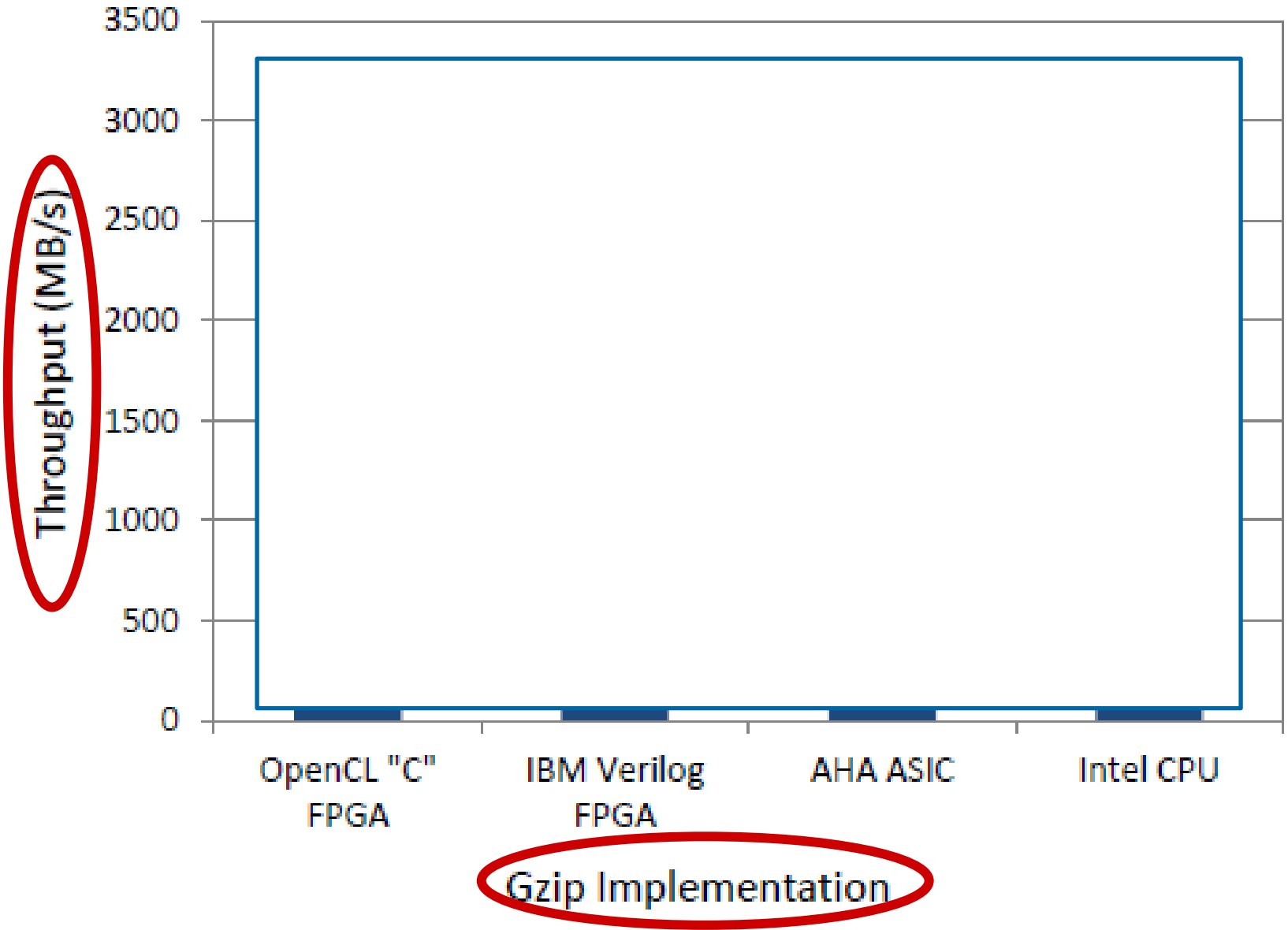
  - Offset = 2048 .. 262144



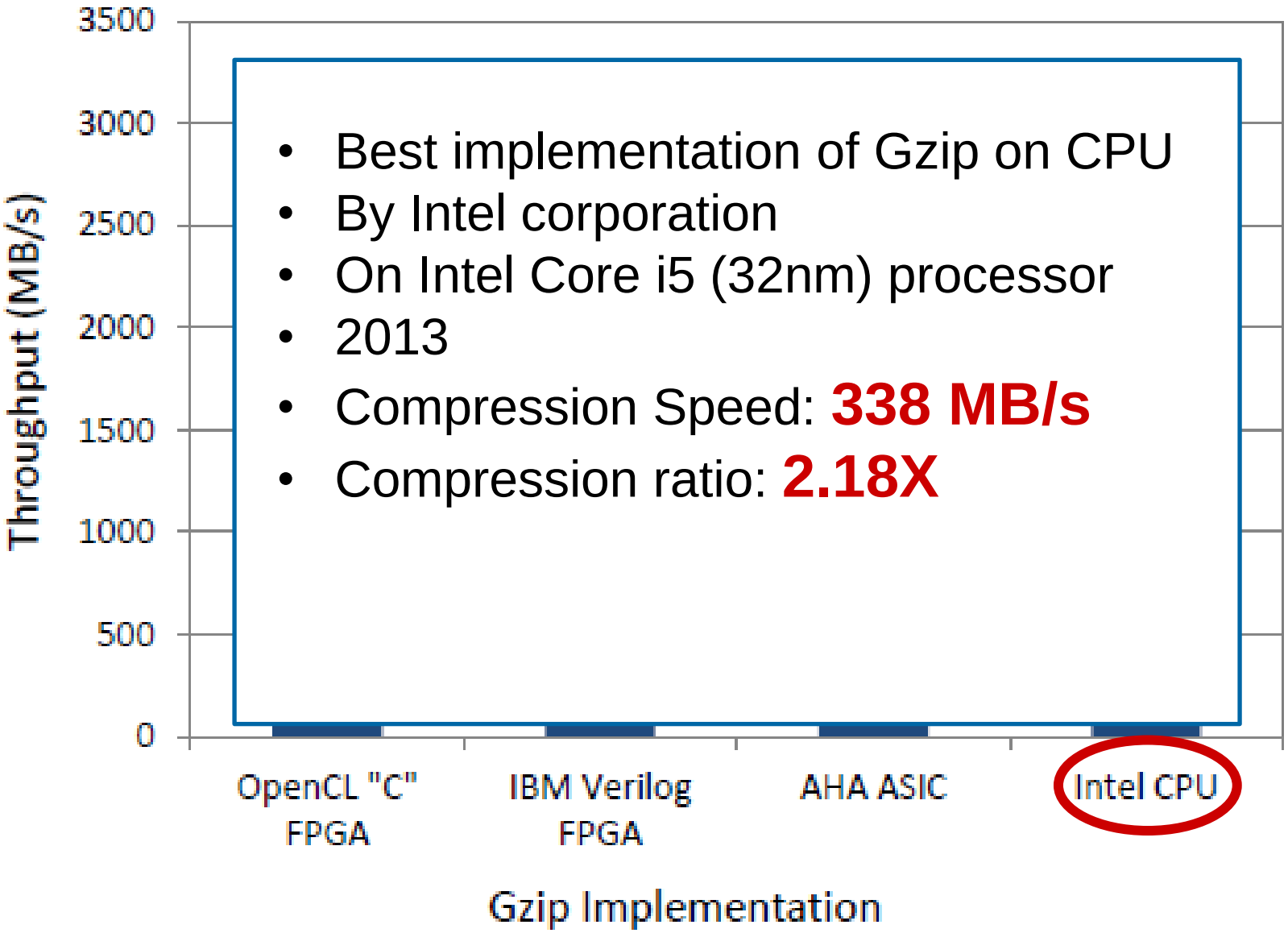
ALTERA
MEASURABLE ADVANTAGE™

```
//loop over all chars in this cycle and compose output
#pragma unroll
for(char i = 0; i < VEC; i++)
{
    if(bestlength[i] == 0)
    {
        output_buffer[output_buffer_size++] = current_window[i];

        if(current_window[i] == marker)
            output_buffer[output_buffer_size++] = 0;
    }
    else if(bestlength[i] > 0)
    {
        //1-output marker
        output_buffer[output_buffer_size++] = marker;

        //2-output bestlength
        output_buffer[output_buffer_size++] = ((bestlength[i] - 3) << 4) | (offset & 0xf);

        //3-output bestoffset
        if(offset < 0x800) //offset needs only one extra byte
        {
            output_buffer[output_buffer_size++] = (offset >> 4) & 0x7f;
        }
        else //offset needs two extra bytes
        {
            output_buffer[output_buffer_size++] =  (offset >> 4)  | 0x80;
            output_buffer[output_buffer_size++] = ((offset >> 11) & 0x7f);
        }
    }
}
```

Comparison against CPU/Verilog – Best Gzips out there!

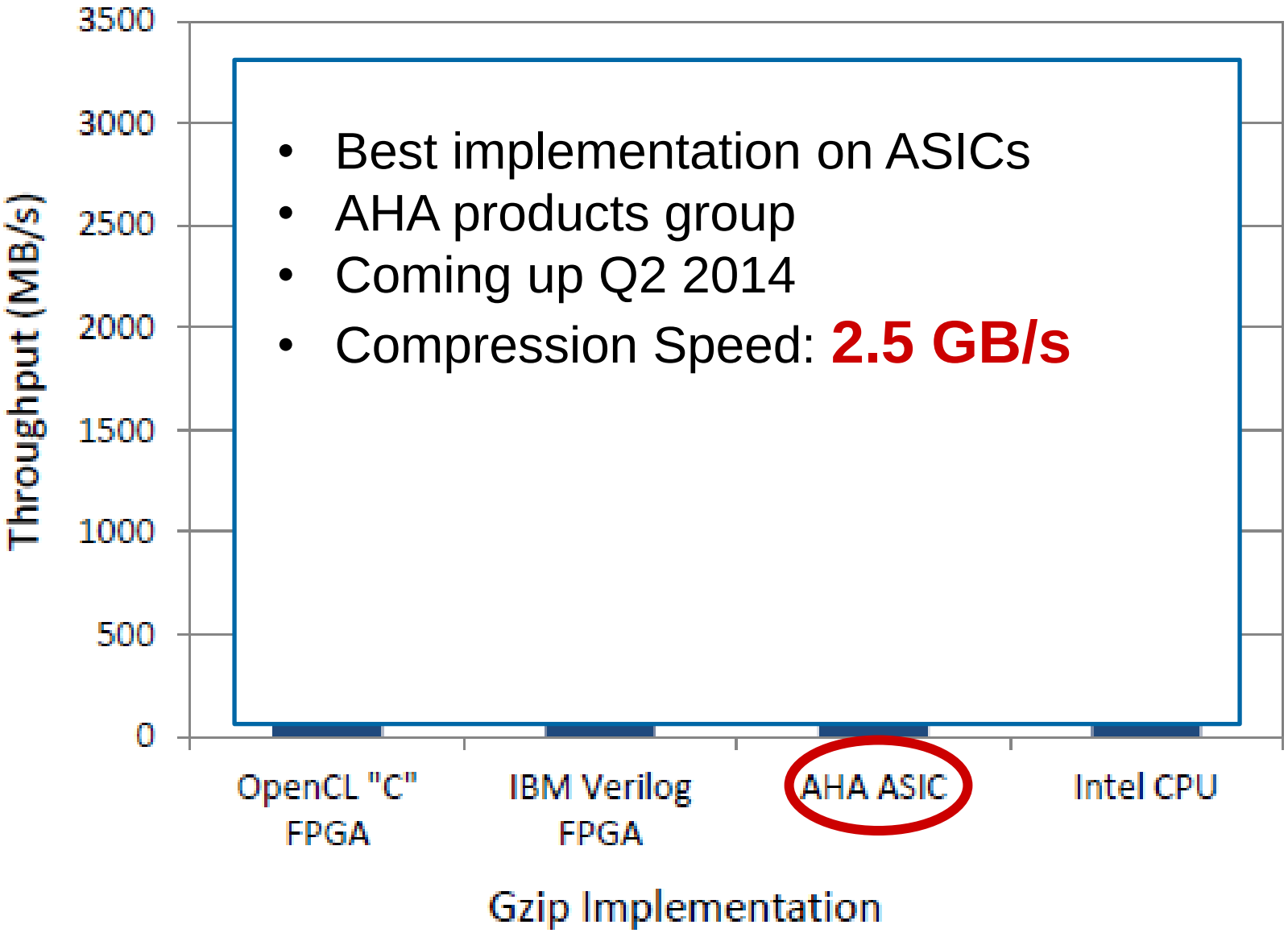


Comparison against CPU/Verilog



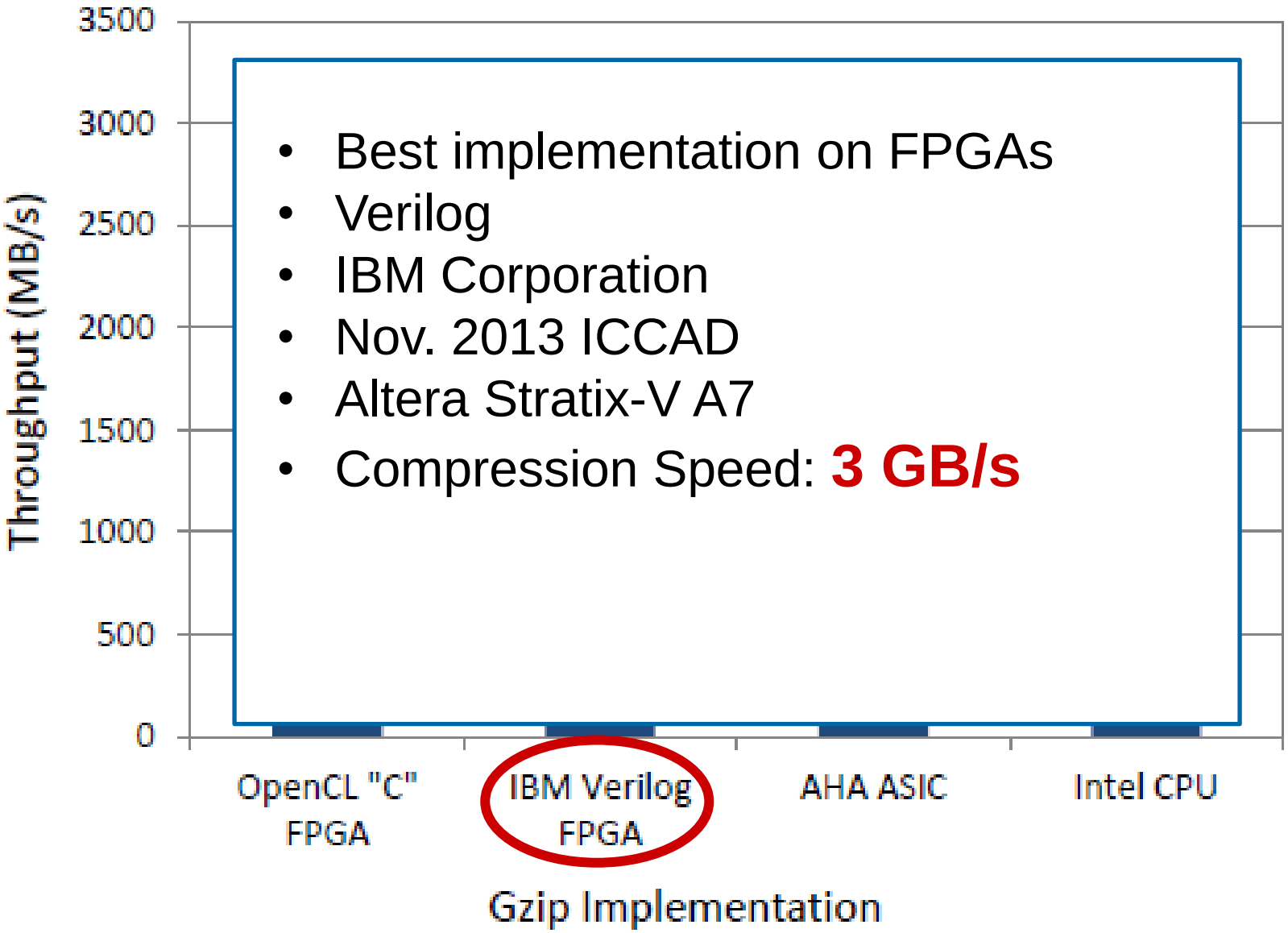
- Best implementation of Gzip on CPU
- By Intel corporation
- On Intel Core i5 (32nm) processor
- 2013
- Compression Speed: **338 MB/s**
- Compression ratio: **2.18X**

Comparison against CPU/Verilog



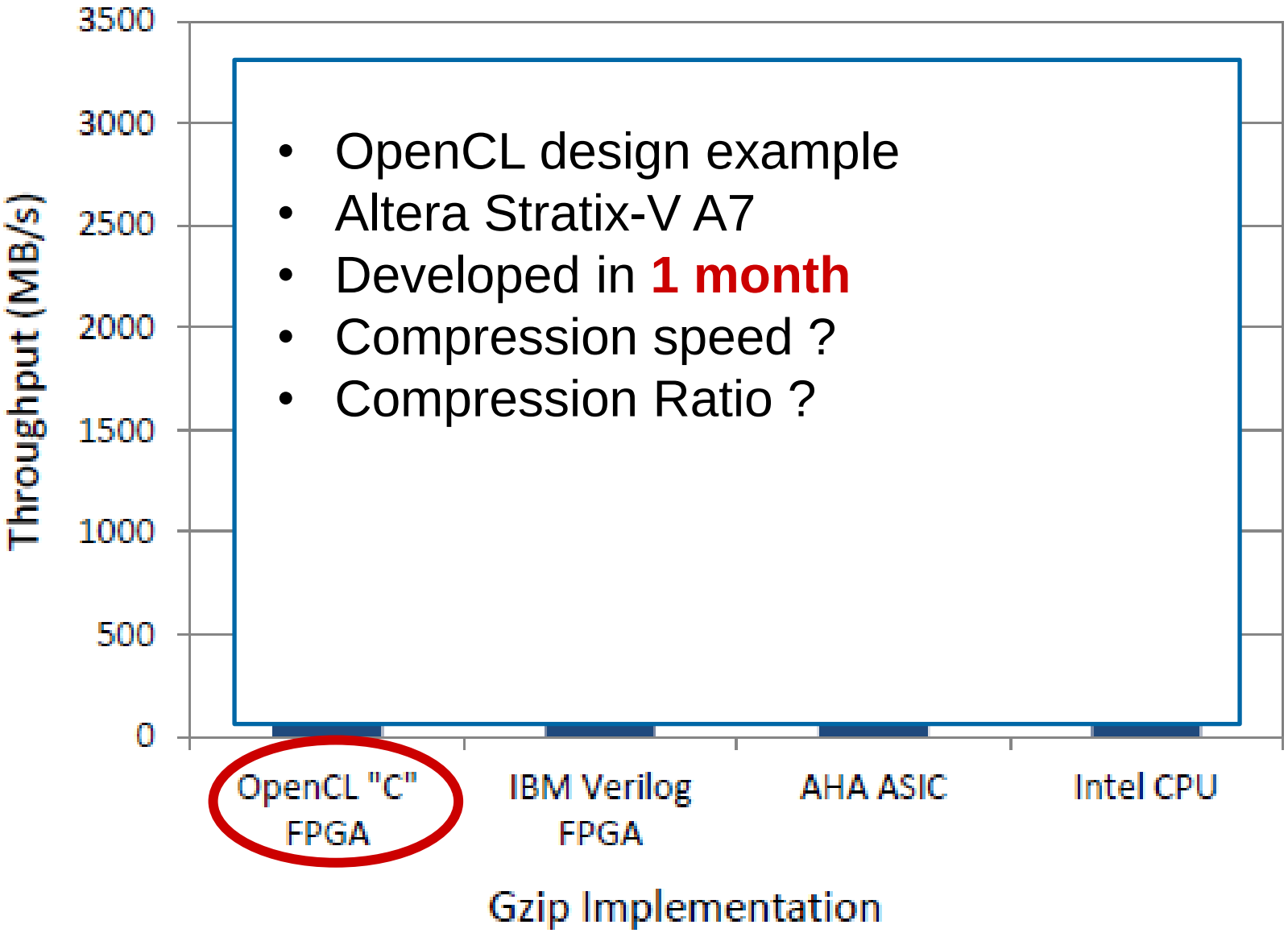
- Best implementation on ASICs
- AHA products group
- Coming up Q2 2014
- Compression Speed: **2.5 GB/s**

Comparison against CPU/Verilog



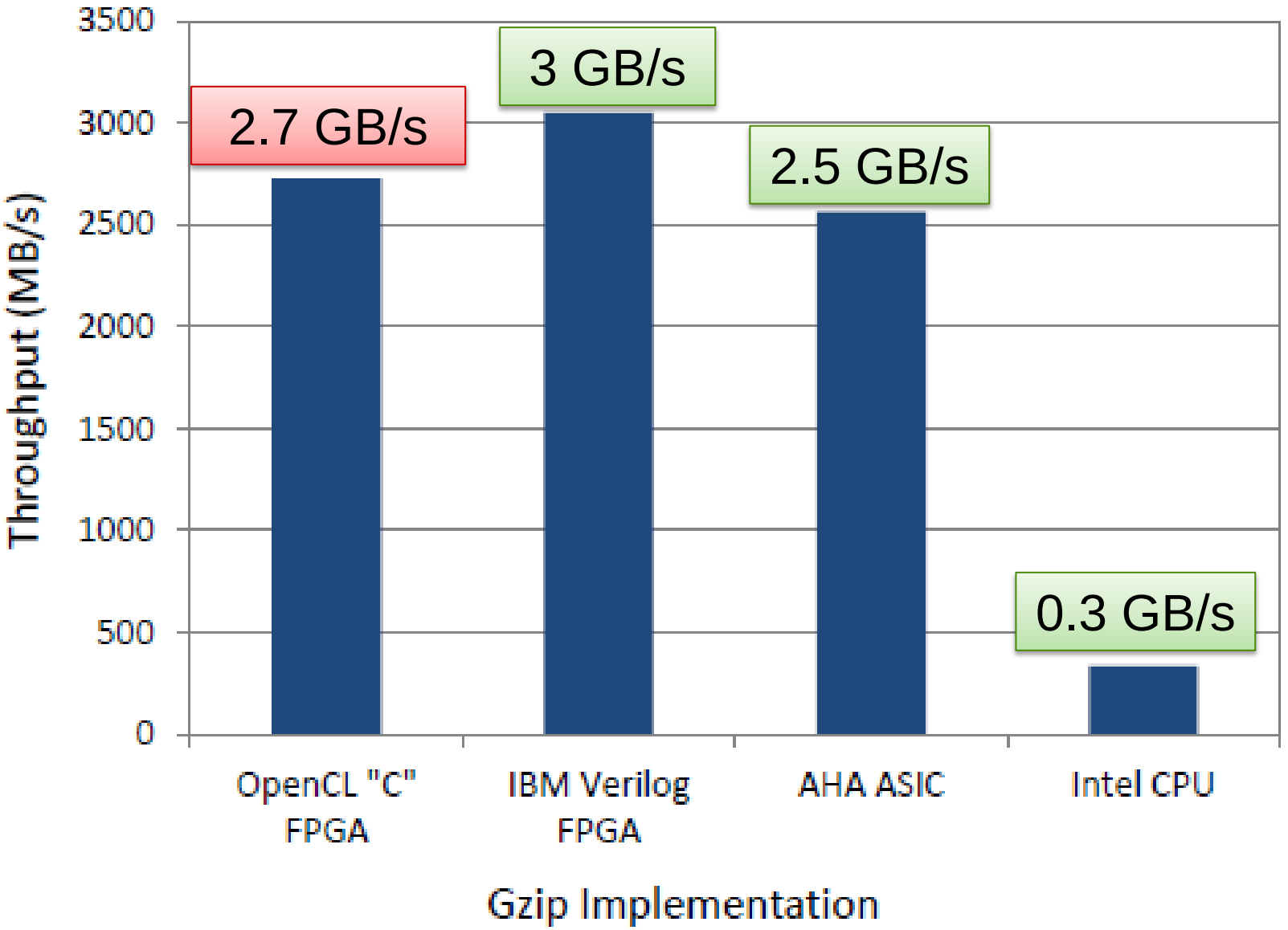
- Best implementation on FPGAs
- Verilog
- IBM Corporation
- Nov. 2013 ICCAD
- Altera Stratix-V A7
- Compression Speed: **3 GB/s**

Comparison against CPU/Verilog

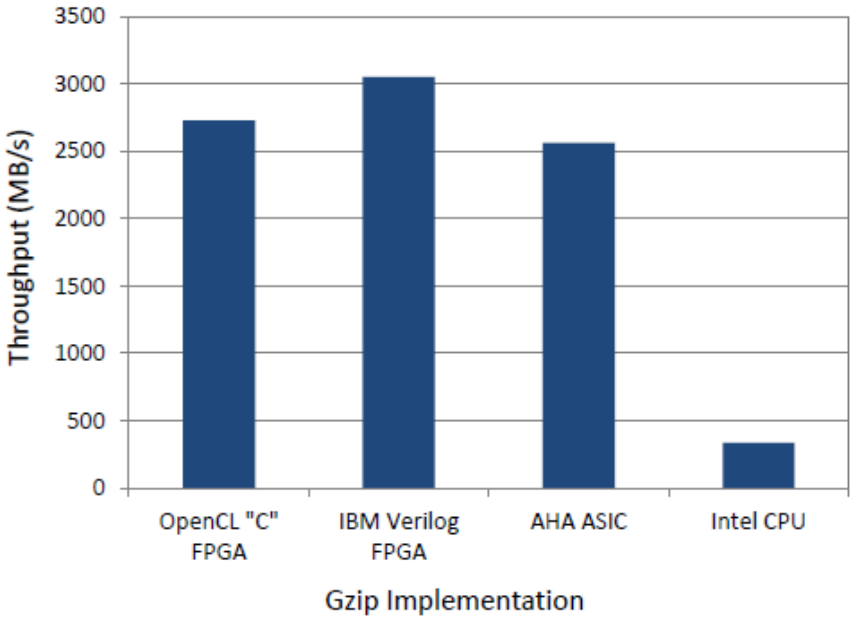


- OpenCL design example
- Altera Stratix-V A7
- Developed in **1 month**
- Compression speed ?
- Compression Ratio ?

Comparison against CPU/Verilog



Comparison against CPU



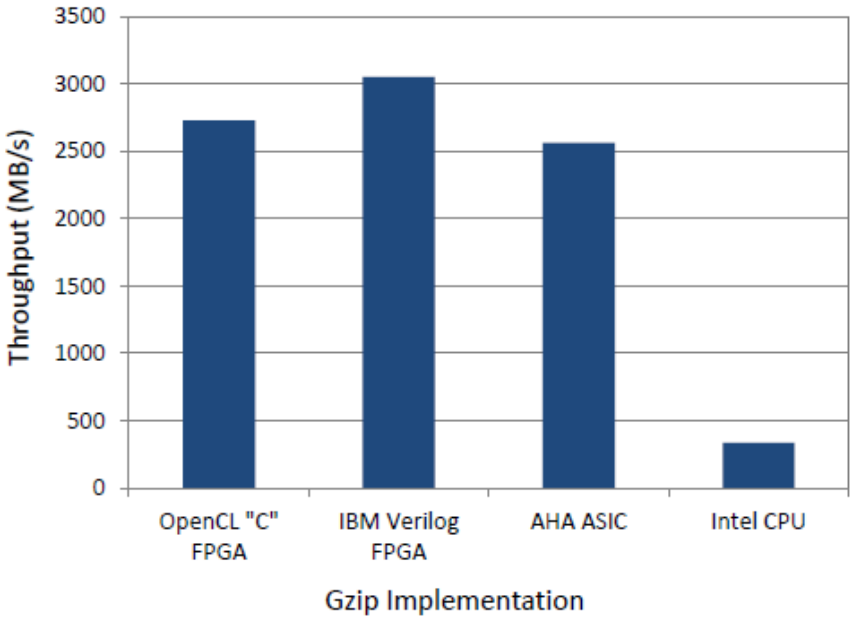
Same compression ratio

12X better performance/Watt

TABLE I: Comparison between our OpenCL FPGA and the best CPU implementation of Gzip.

	Performance	Performance/Watt	Compression Ratio
OpenCL FPGA	2.71 GB/s	111 MB/J	2.17×
Intel Gzip	338 MB/s	9.26 MB/J	2.18×
Gap	8.2× faster	12× better	on par

Comparison against Verilog



10% Slower

12% more resources

Much lower design effort and design time
Days instead of months

	Performance	Efficiency	Productivity
OpenCL Kernel	2.71 GB/s 179 MHz	51% logic 68% RAM	High (1 month)
Verilog	2.98 GB/s 200 MHz	45% logic* 45% RAM*	Low
Gap	10% slower	6% more 23% more	– –

*conservative area estimate based on chip image. [10]

Thank You

ALTERA®

MEASURABLE ADVANTAGE™