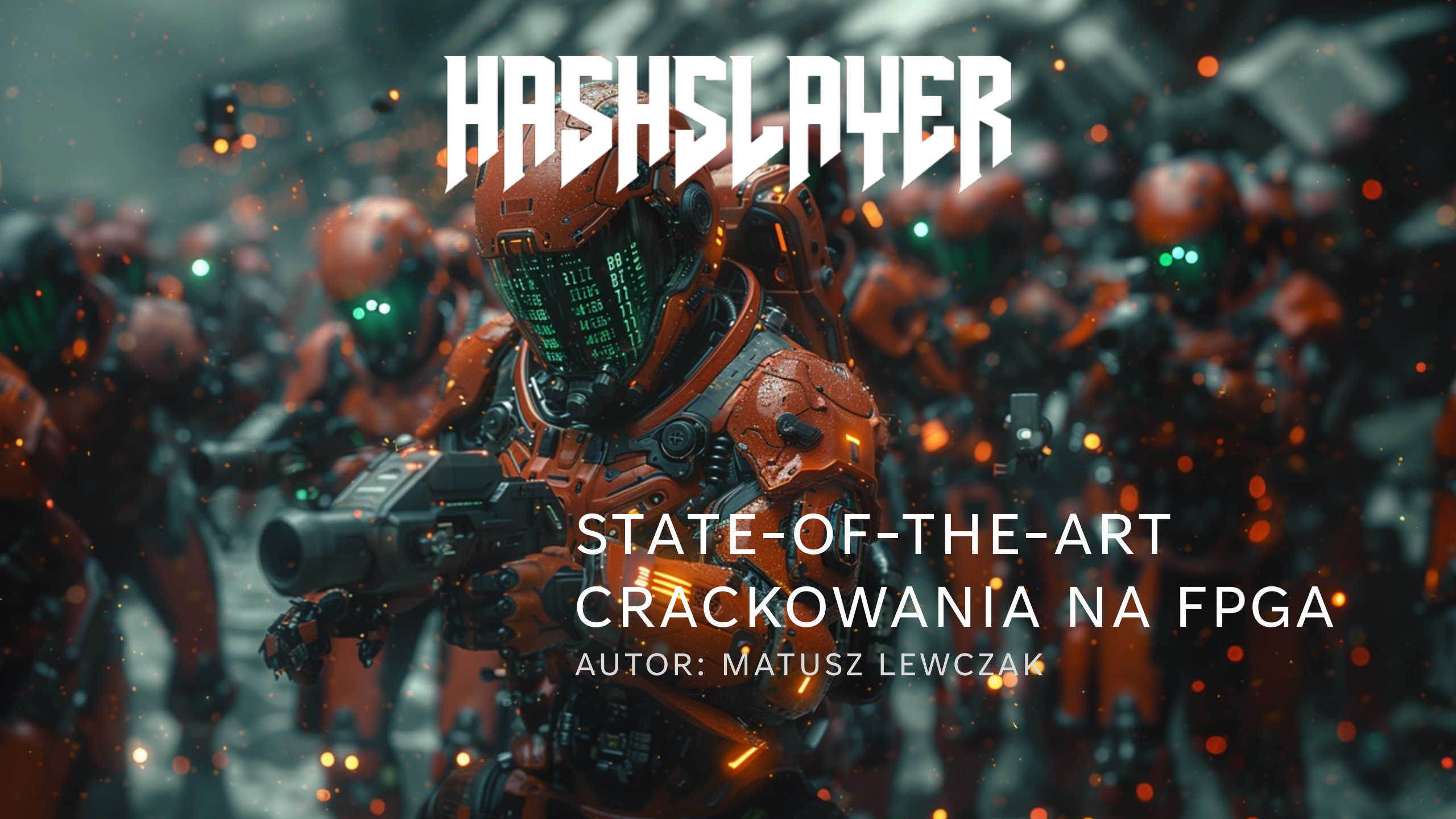


STATE-OF-THE-ART CRACKOWANIA NA FPGA

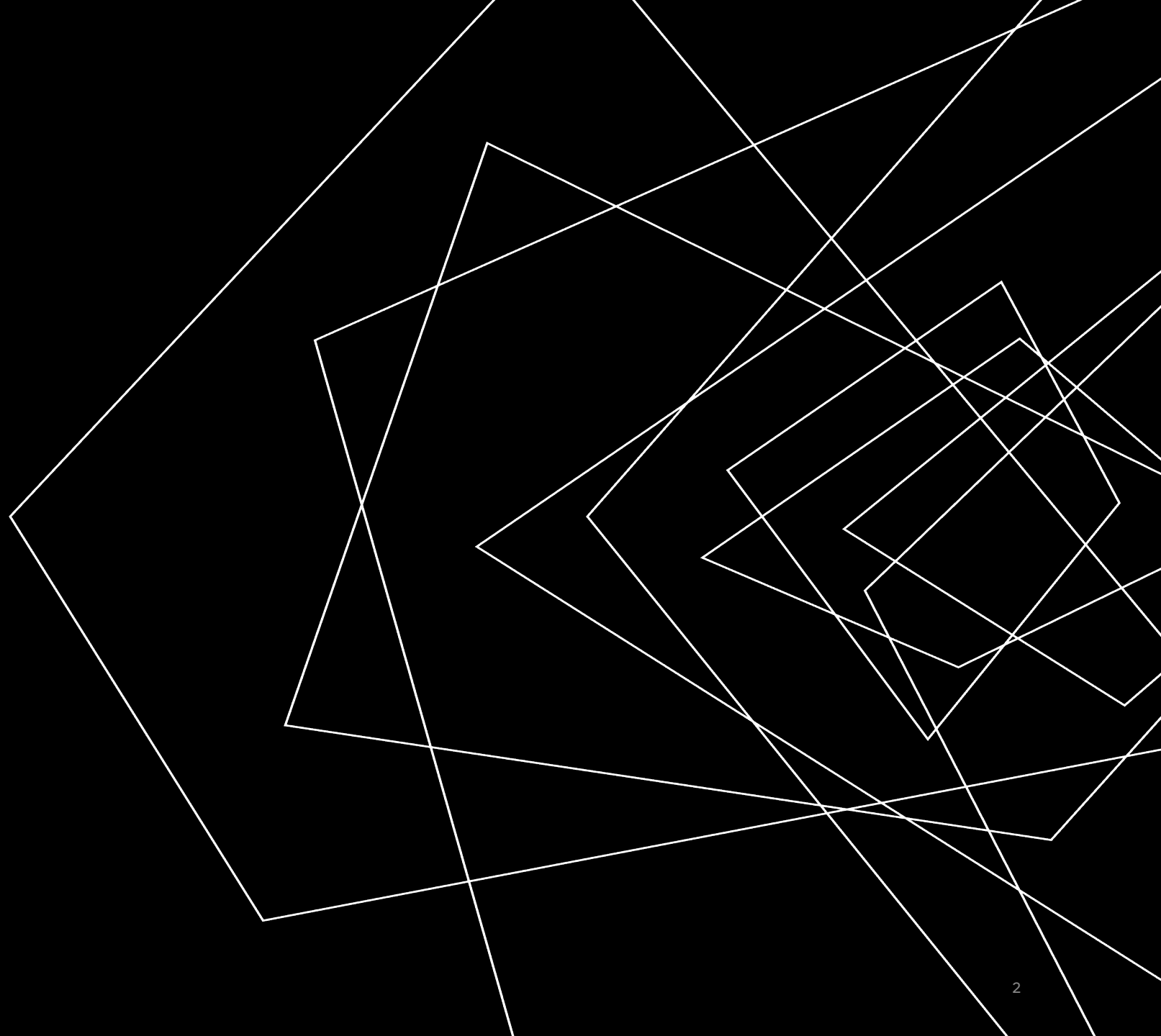
AUTOR: MATUSZ LEWCZAK

AGENDA

Wstęp do FPGA

FPGA w chmurze

Crackowanie na FPGA



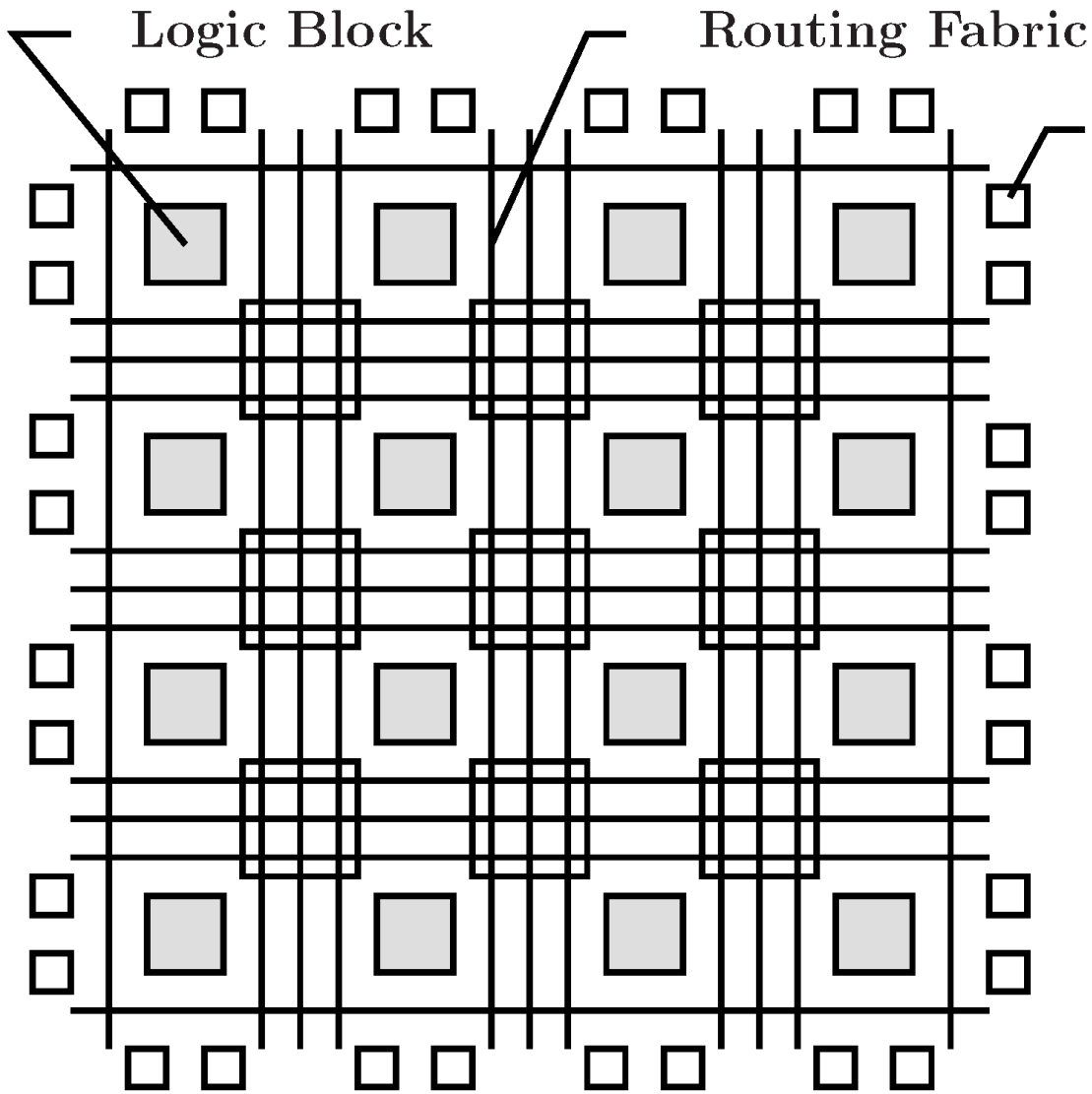
MATEUSZ LEWCZAK

IT SECURITY CONSULTANT @ SECURITUM

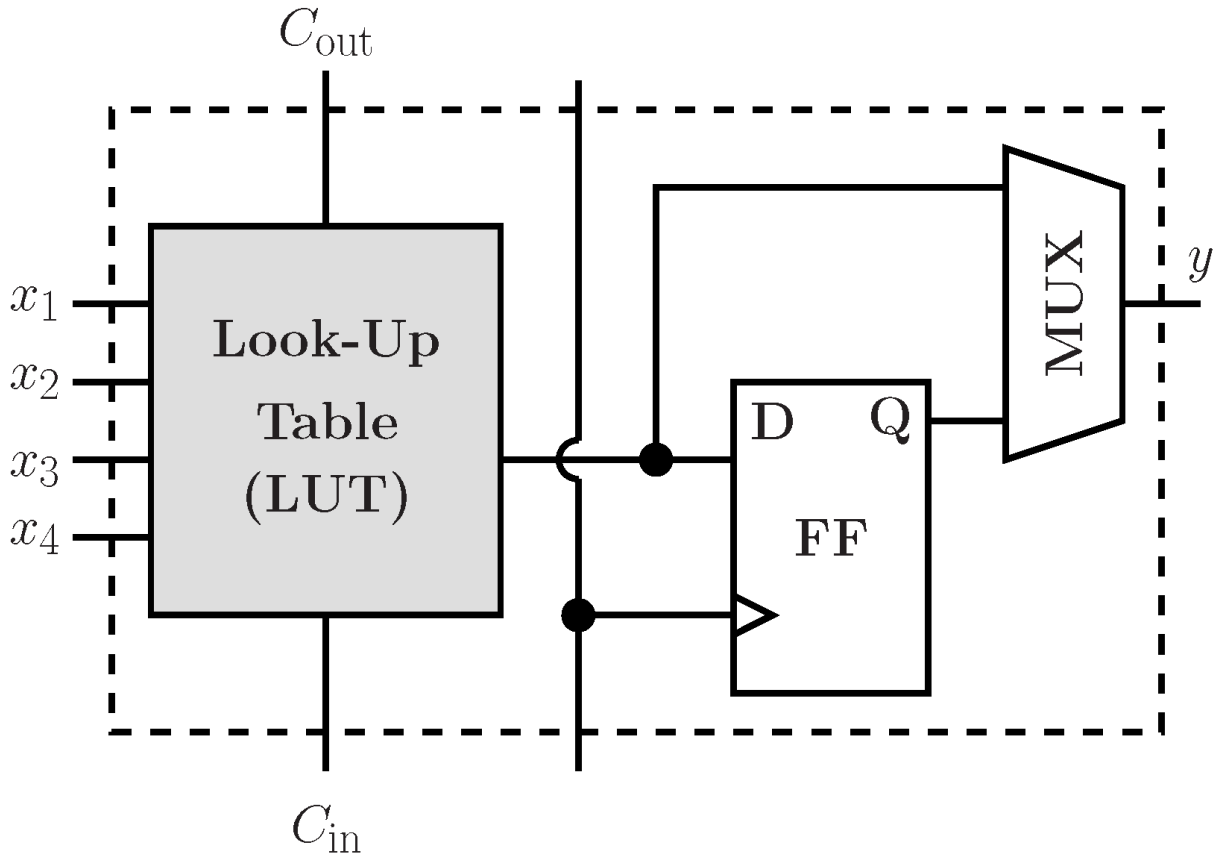
SZYBKA TERMINOLOGIA

- Układ == układ realizujący jakąś logikę.
- Hashcore == układ liczący hashe.
- Jednostka hashująca == Wiele jednostek hashujących w jednym układzie.
- Programowanie FPGA == wgrywanie fizycznej konfiguracji FPGA.

CZYM JEST FPGA?

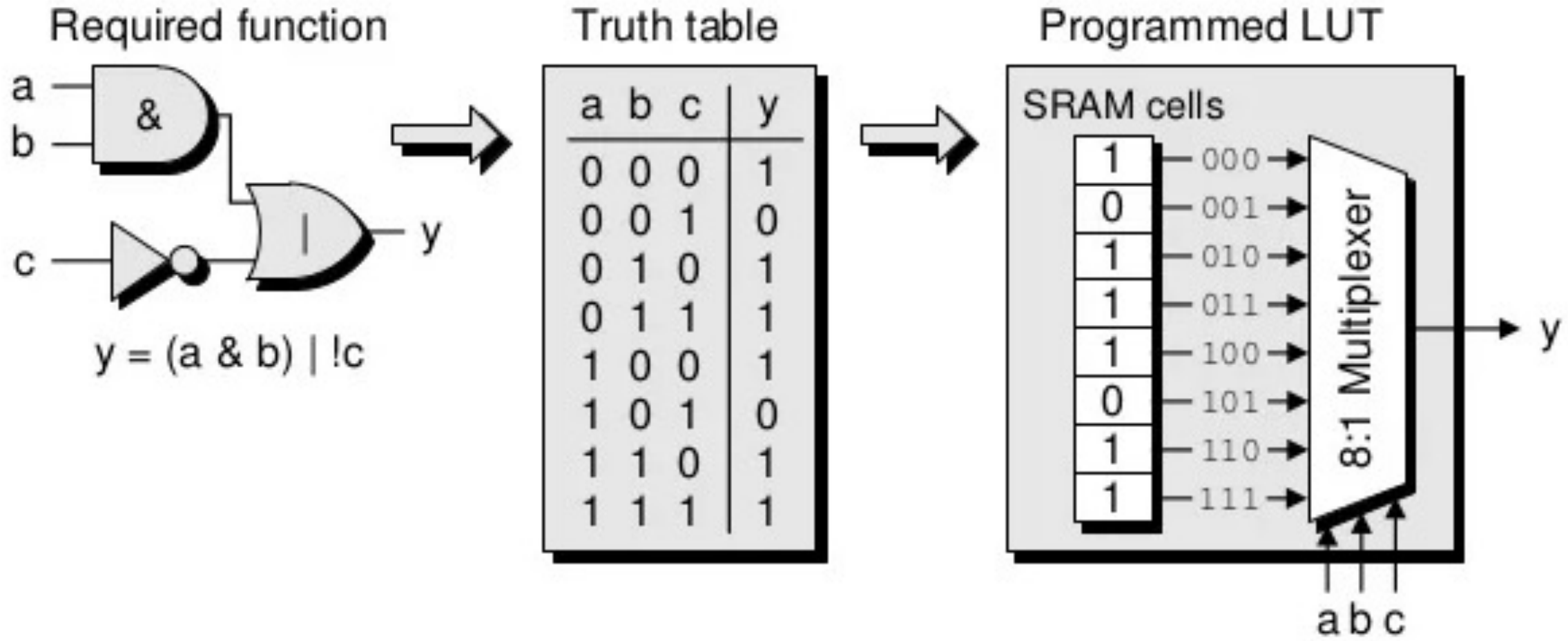


(a)

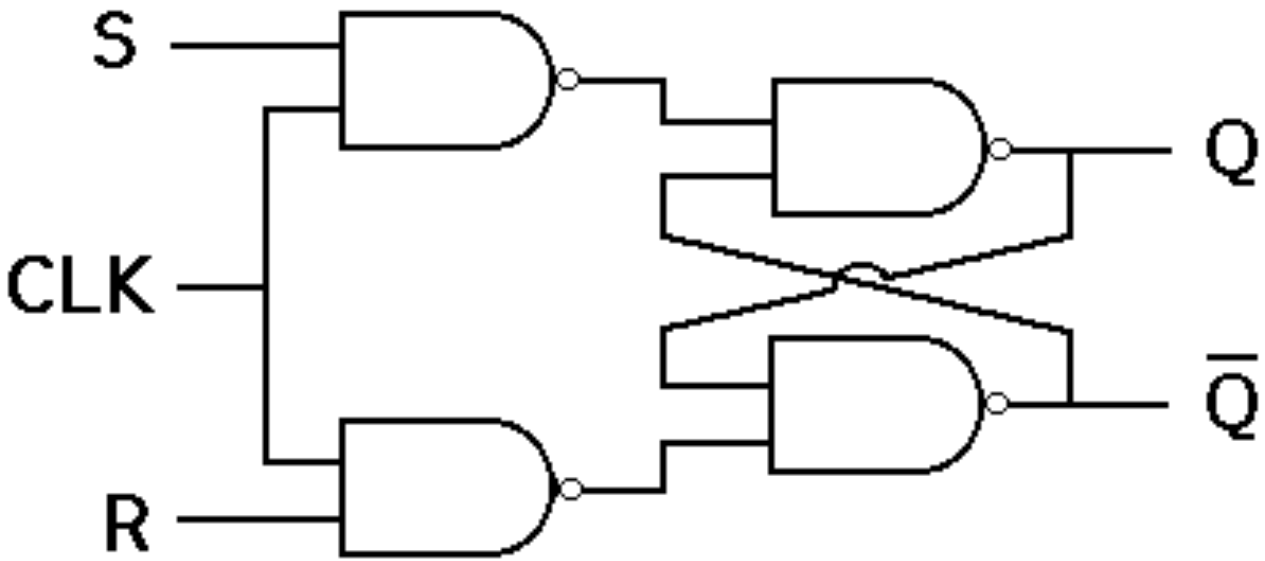


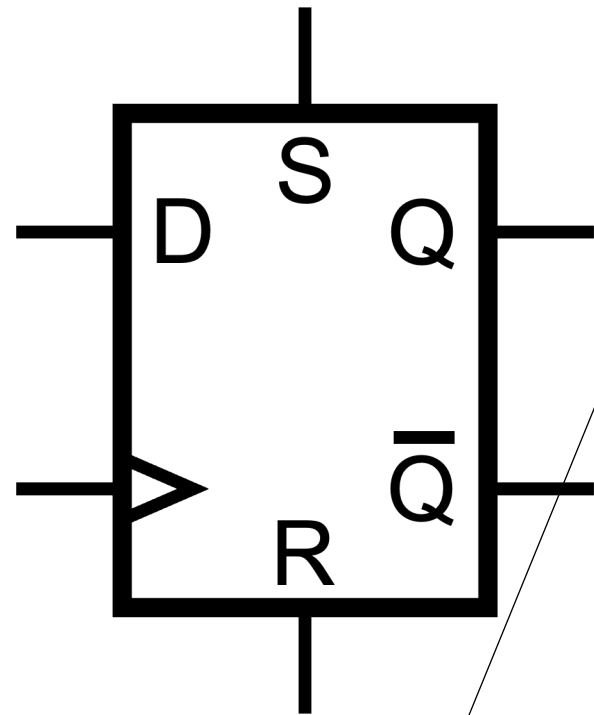
(b)

LOOK-UP TABLE

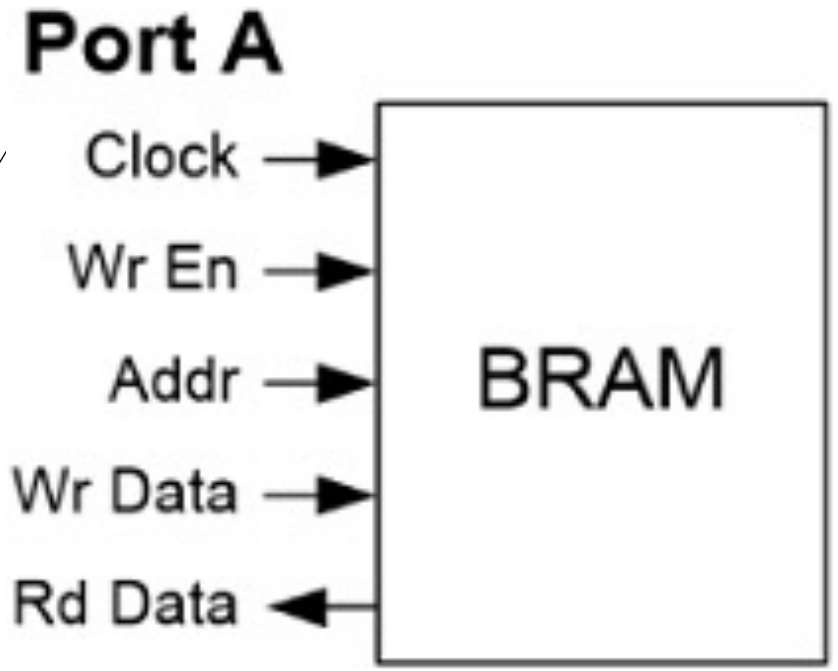


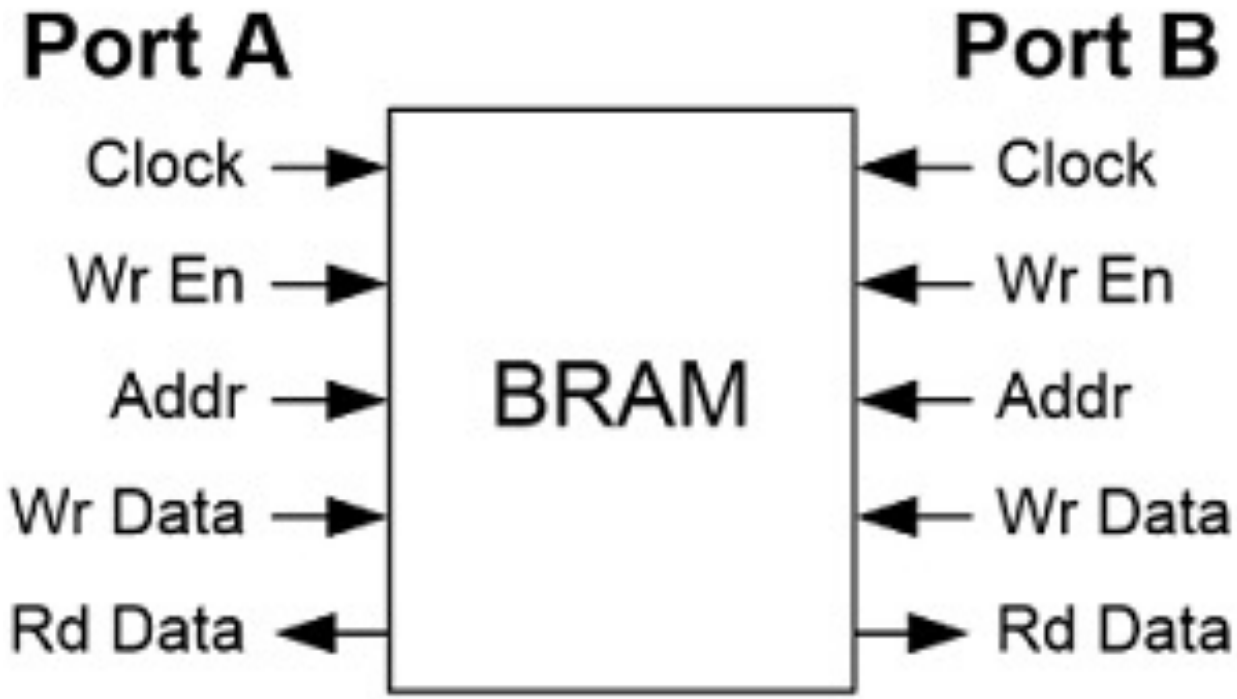
FLIP-FLOP'Y





BLOCK RAM


Port A
Port B
Clock
Wr En
Addr
Wr Data
Rd Data
BRAM
not wired


Port A
Port B
Clock
Wr En
Addr
Wr Data
Rd Data
BRAM
Clock
Wr En
Addr
Wr Data
Rd Data

ZASTOSOWANIE

- Przetwarzanie danych w czasie rzeczywisty.
- Przetwarzanie sygnałów cyfrowych.
- Sztuczna inteligencja (Altera).
- Glue Logic (iPhone 7).
- Automitve.
- Wszędzie...

PROGRAMOWANIE FPGA

„PROGRAMOWANIE” FPGA

VHDL:

```
2  process ({S0,S1},A,B,C,D)
3  begin
4     case {S0,S1}, is
5        when "00" => Y <= A;
6        when "01" => Y <= B;
7        when "10" => Y <= C;
8        when "11" => Y <= D;
9        when others => Y <= A;
10    end case;
11 end process;
```

Verilog:

```
1
2
3  always @({S0,S1}, A, B, C, D)
4        case ({S0,S1})
5           2'b00: Y = A;
6           2'b01: Y = B;
7           2'b10: Y = C;
8           2'b11: Y = D;
9        endcase
10
```

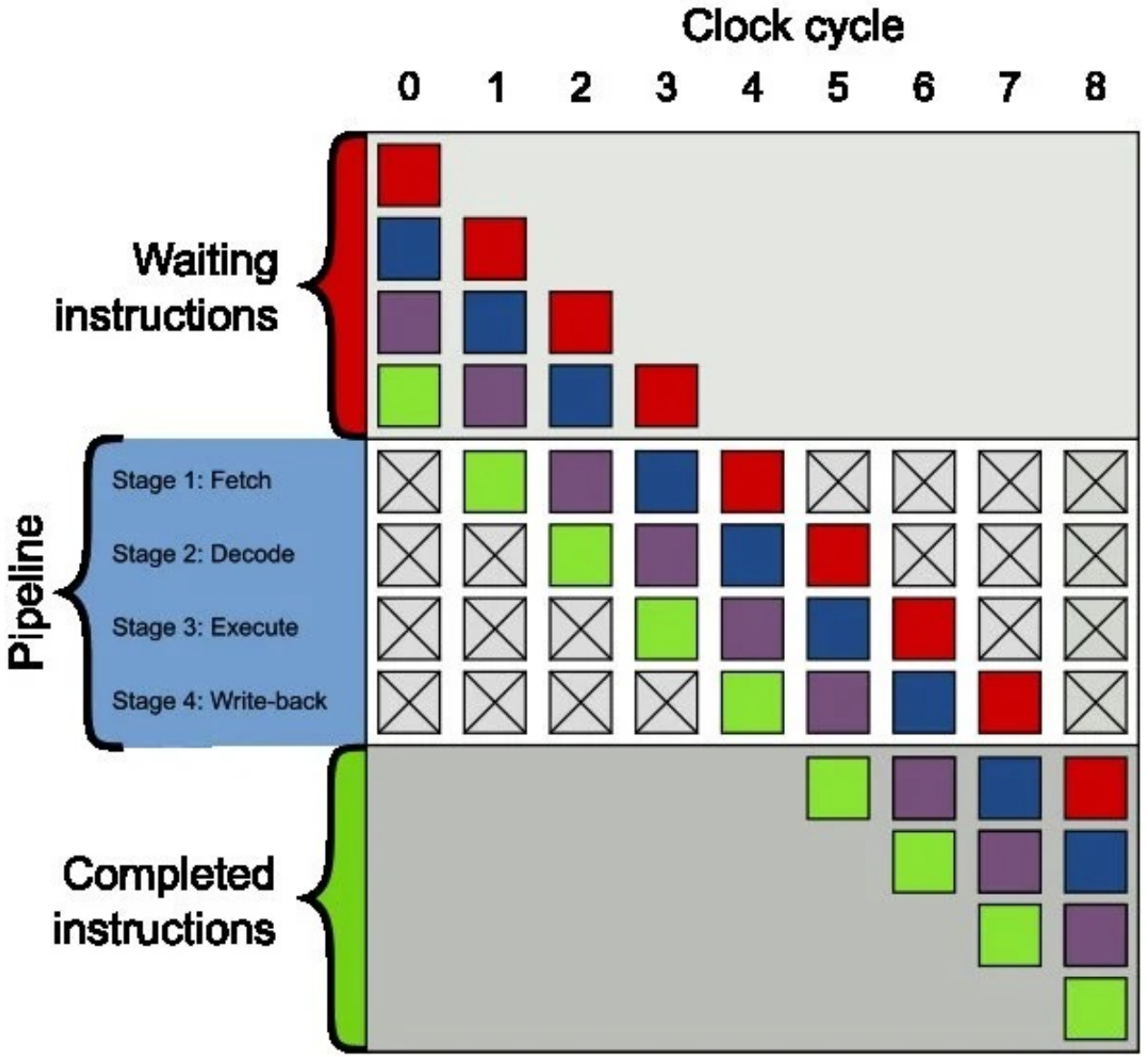
HIGH LEVEL SYNTHESIS

FPGA VS CPU VS GPU

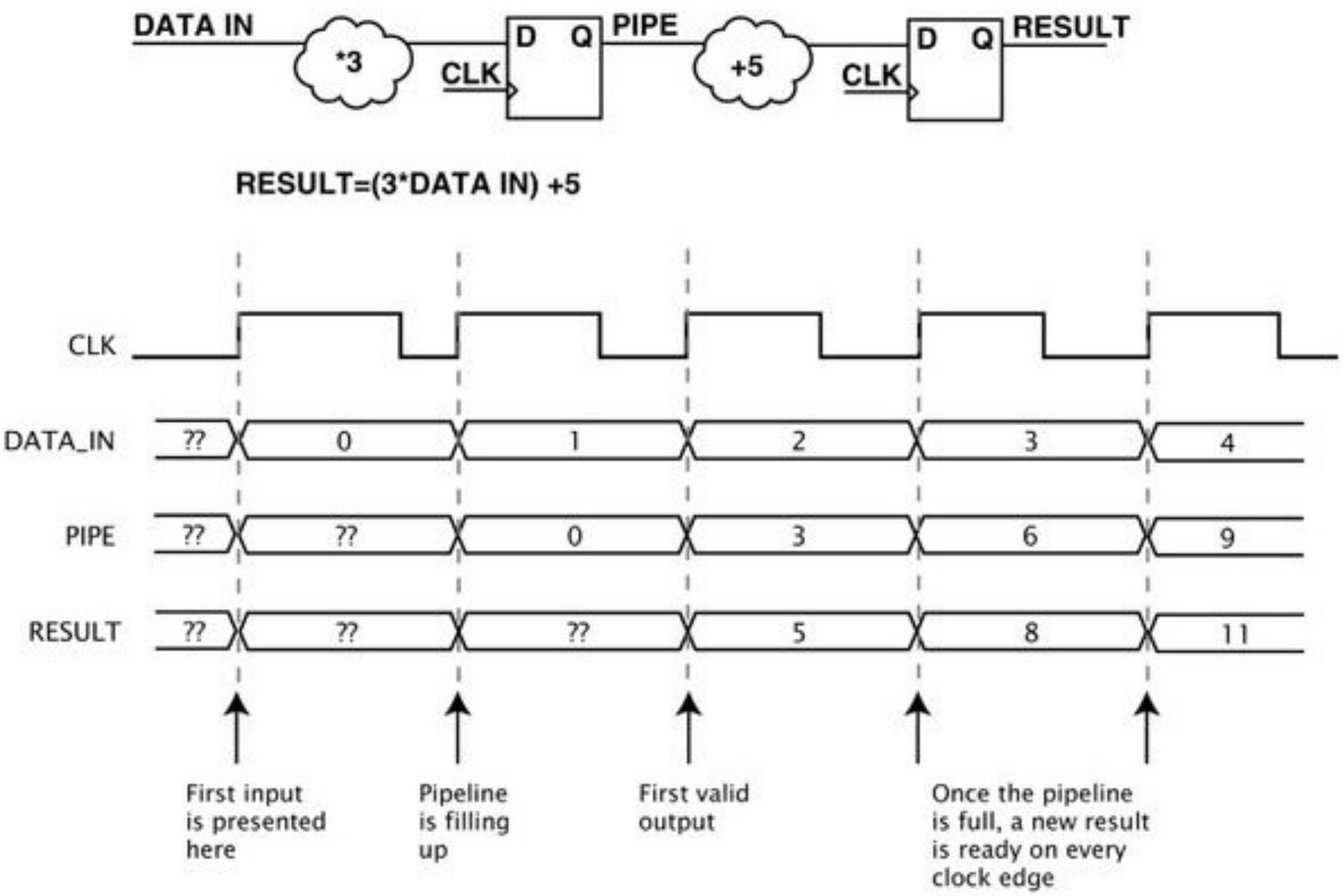
FPGA	CPU i GPU
Bazuje na bramkach i sygnałach	Bazuje na instrukcjach
Specjalistyczne układy skrojone pod potrzebę	Ogólne zastosowanie
Niski zegar (100/200 MHz)	Wysoki zegar (4/5 GHz)
Brak jakiegokolwiek narzutu	Narzut softwareowy
Bardzo małe zużycie energii (10/20 W)	Duże zużycie energii (nawet 600W)

PIPELINING

PIPELINING


Clock cycle
0
1
2
3
4
5
6
7
8
Waiting instructions
Pipeline
Stage 1: Fetch
Stage 2: Decode
Stage 3: Execute
Stage 4: Write-back
Completed instructions

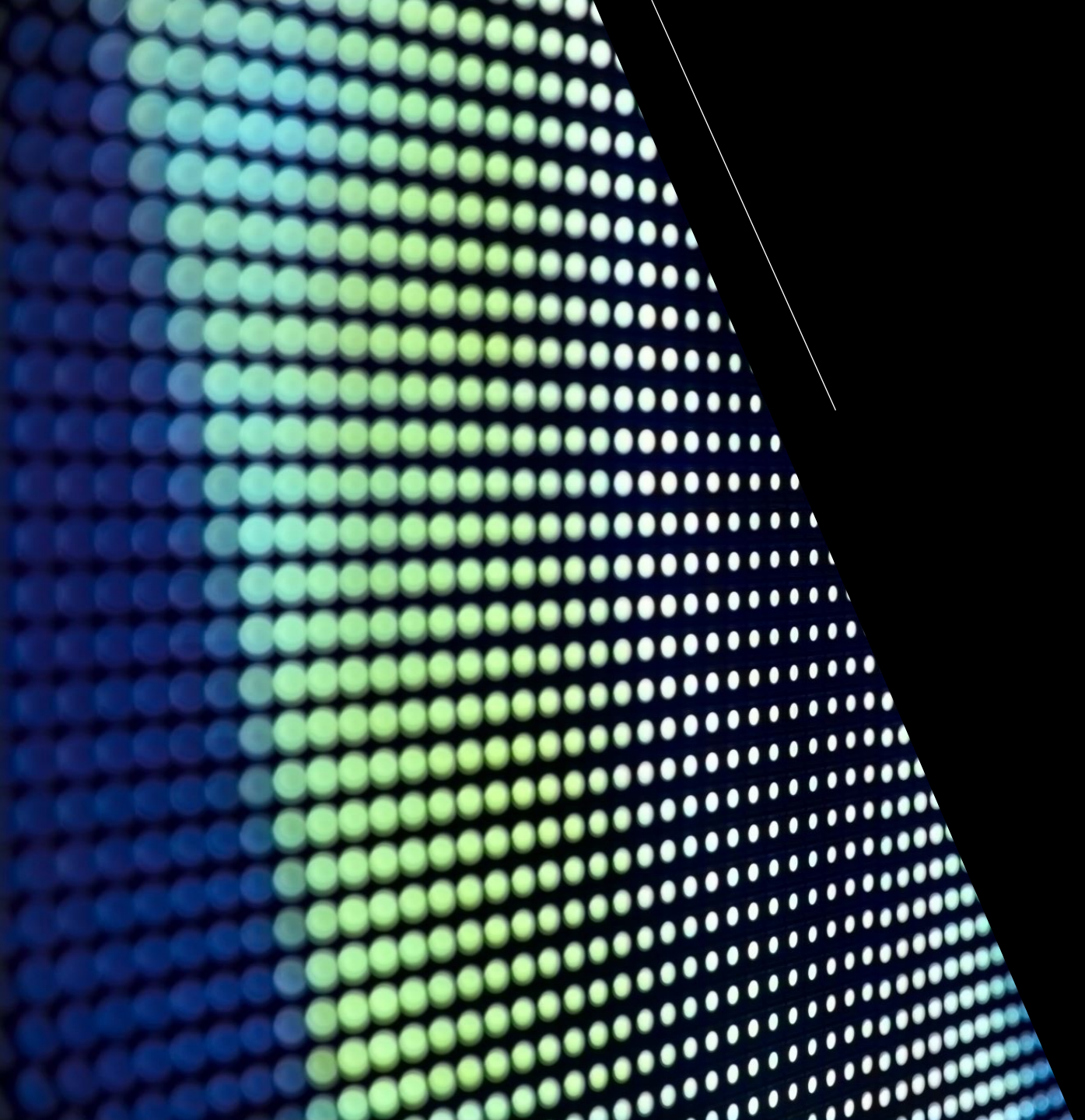
PIPELINING


DATA IN
*3
CLK
D Q
PIPE
+5
CLK
D Q
RESULT
RESULT=(3*DATA IN) +5
CLK
DATA_IN
?? 0 1 2 3 4
PIPE
?? ?? 0 3 6 9
RESULT
?? ?? ?? 5 8 11
First input is presented here
Pipeline is filling up
First valid output
Once the pipeline is full, a new result is ready on every clock edge

PIPELINING – ANALOGIA

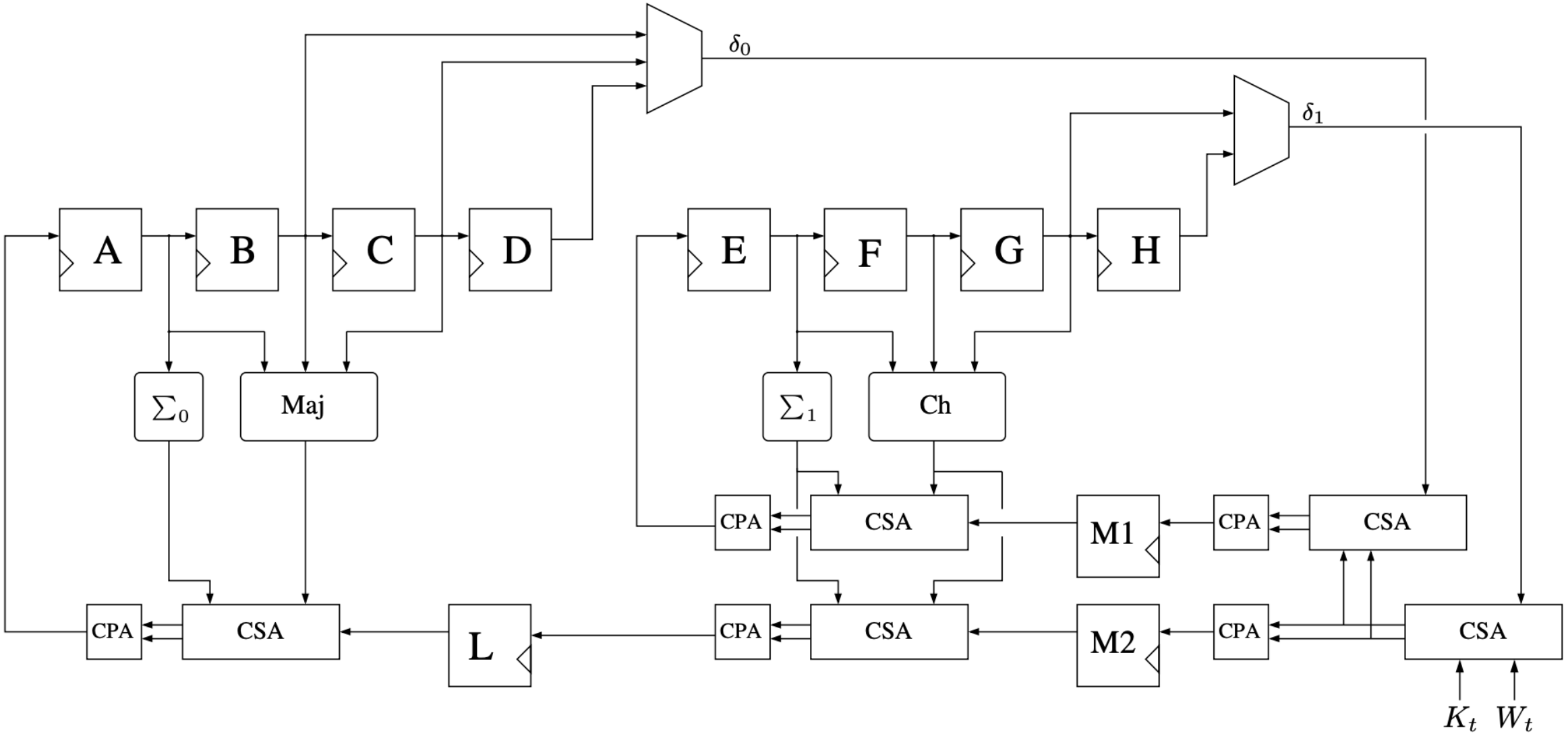
TY

SYN KOLEZANKI ~~TWOJEJ MAMY~~

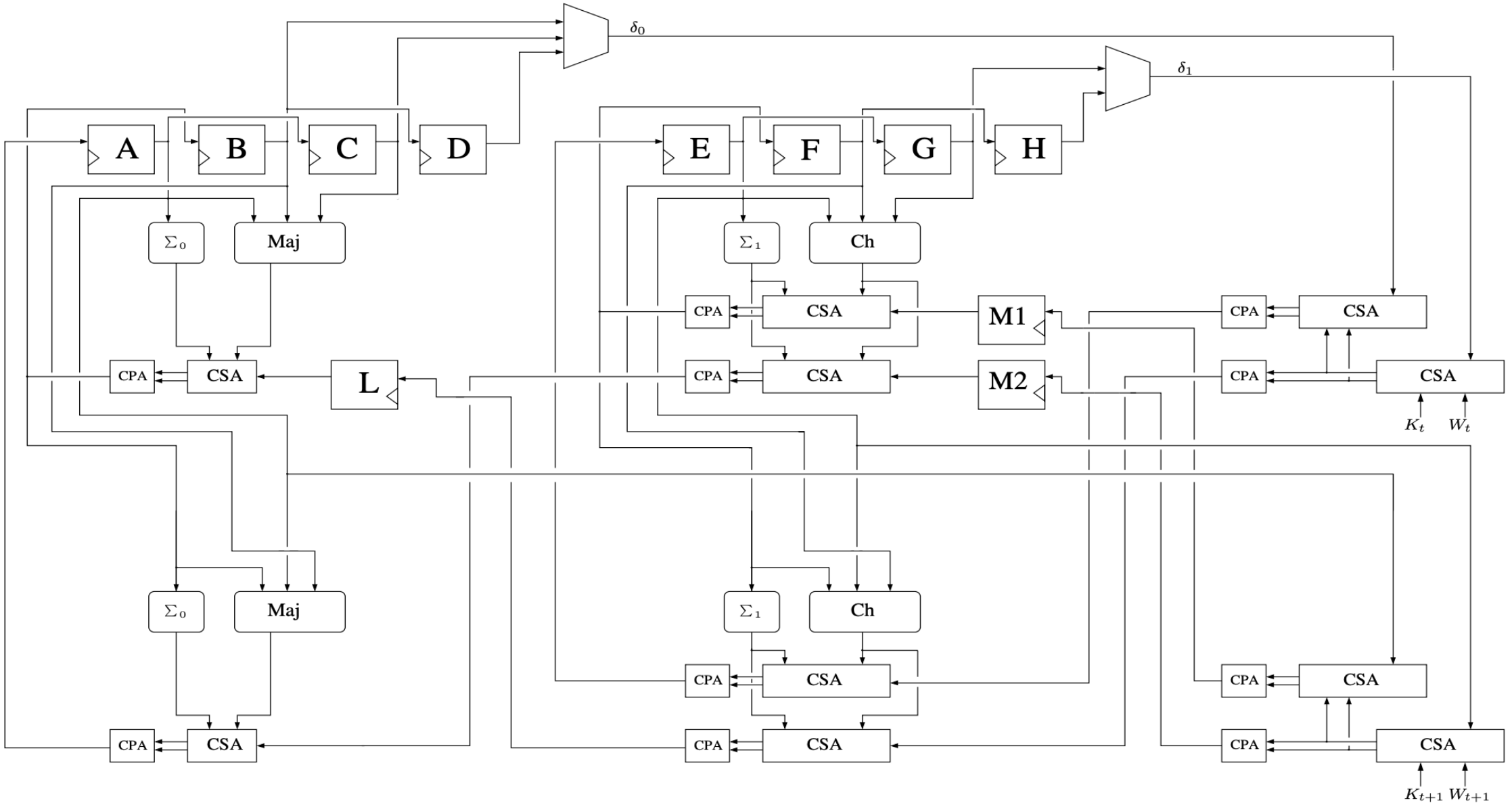


JAK WYGLADA PRZYKLADOWA JEDNOSTKA HASHUJACA?

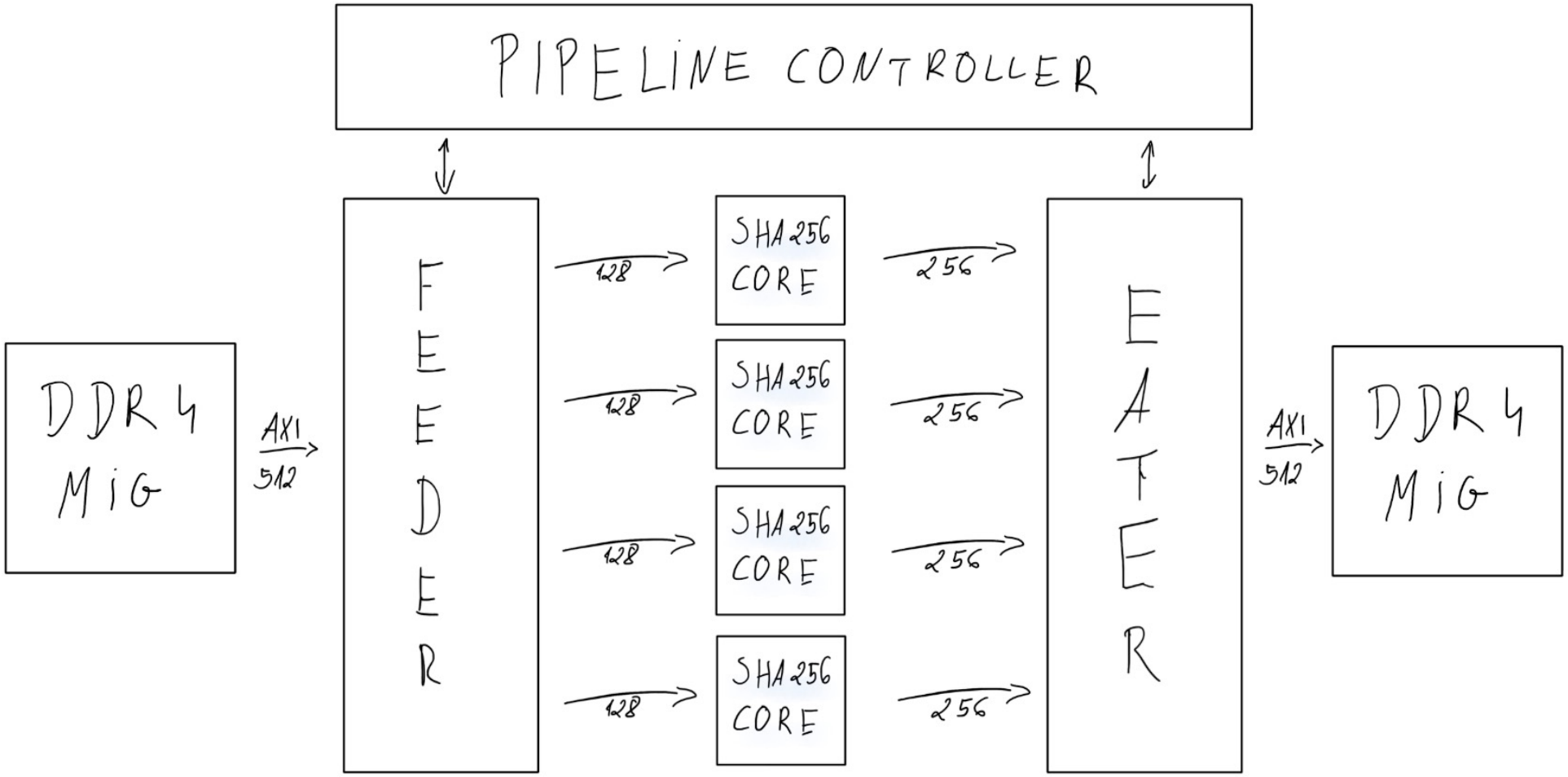
SHA256 – NOT PIPELINED



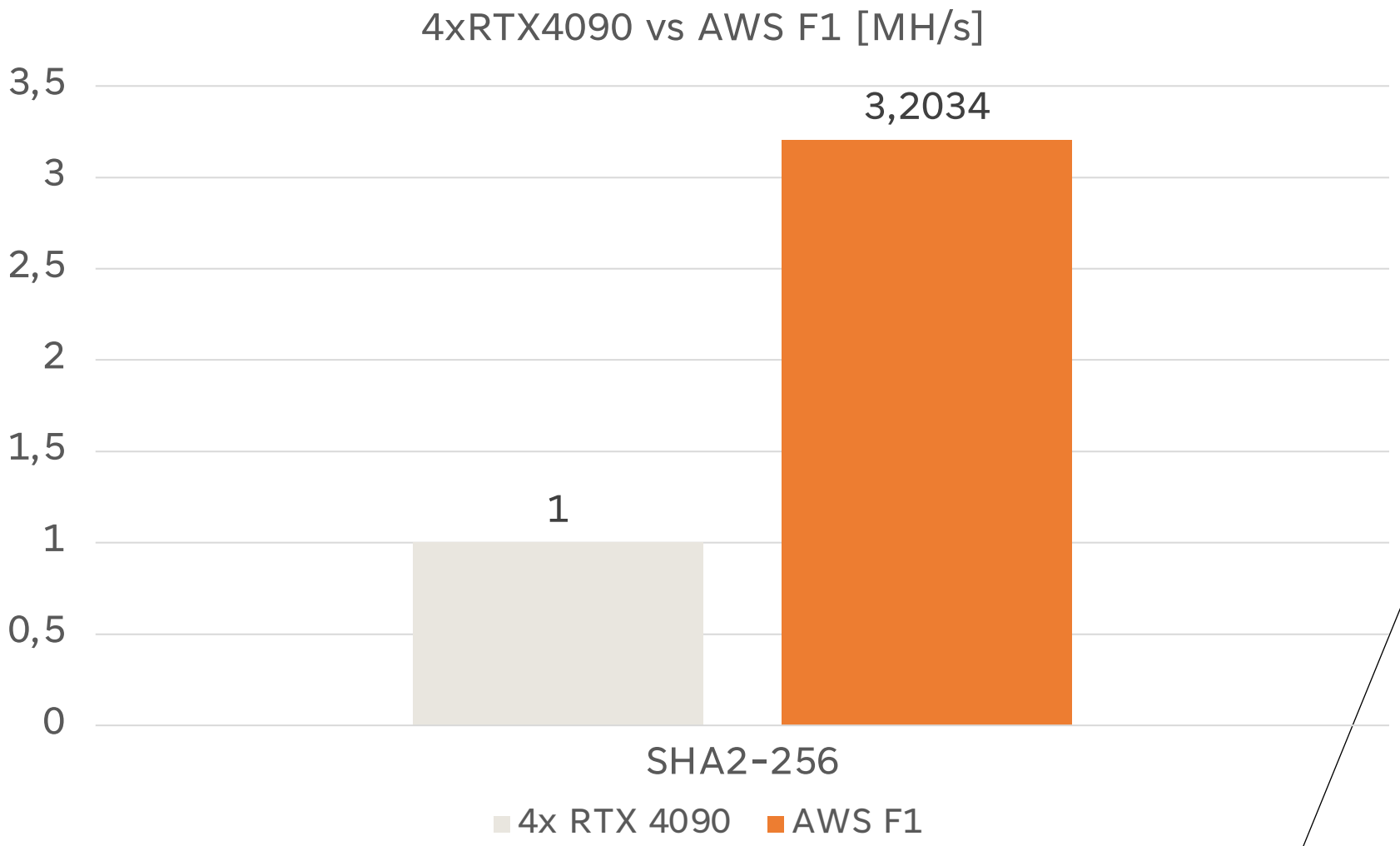
SHA256 – PIPELINED

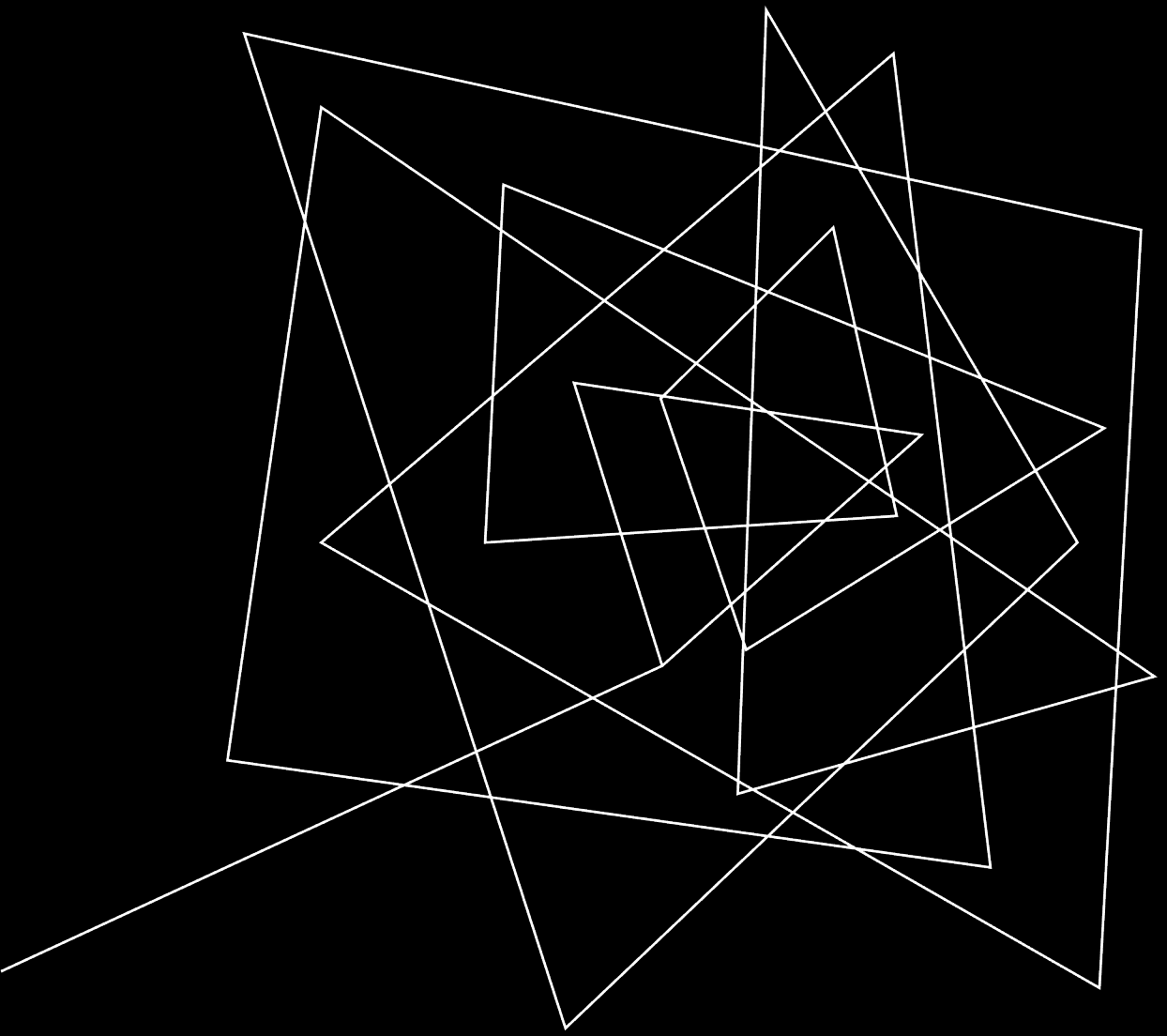


POGLADOWY UKLAD SHA256



WYNIKI


4xRTX4090 vs AWS F1 [MH/s]
3,5
3
2,5
2
1,5
1
0,5
0
1
3,2034
SHA2-256
4x RTX 4090
AWS F1



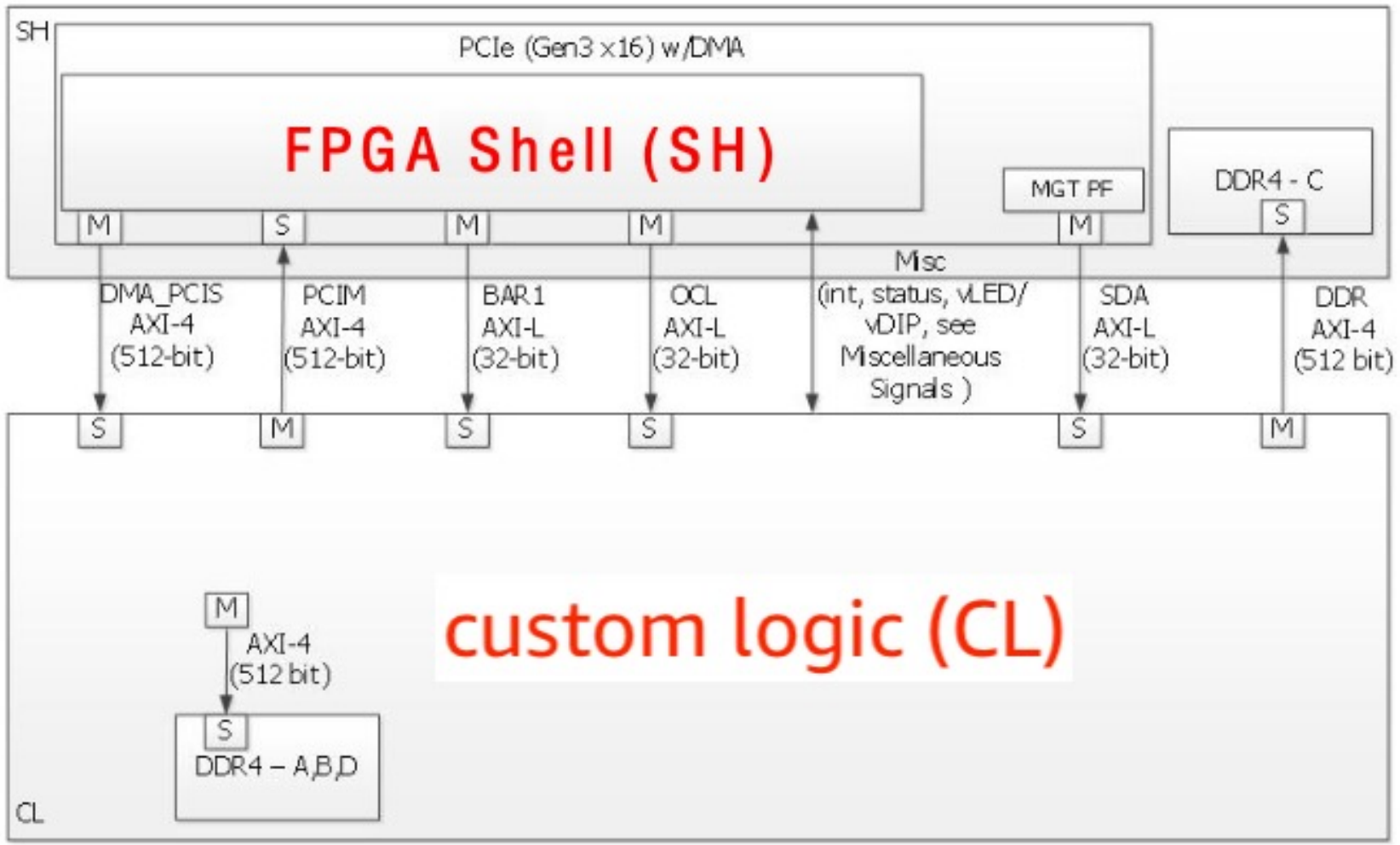
FPGA W CHMURZE – SPECYFIKACJA

SPECYFIKACJA SPRZETOWA

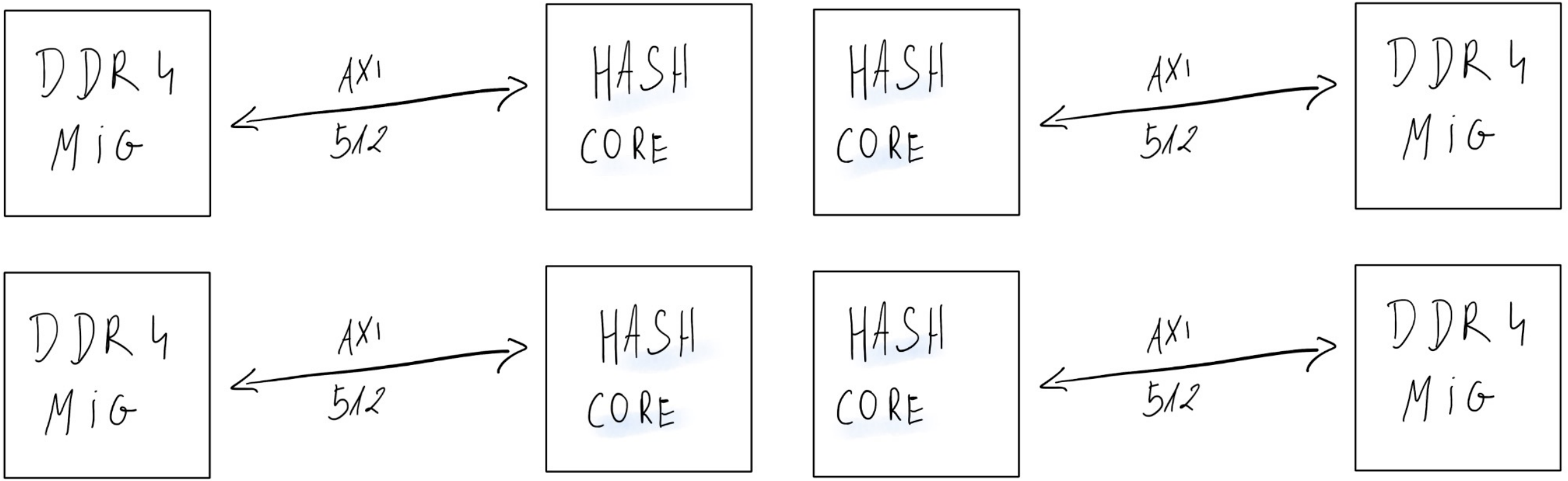
AWS EC2 f1.2xlarge

- Xilinx Virtex UltraScale+ VU9P,
  - 1.2M LUTS,
  - 2.5M FF,
  - 9 MB BRAM,
  - 34 MB ULTRARAM.
- 2x32 GB DDR4 RDIMM (512 bit interface),
- Dedykowany interfejs PCI Express x16.

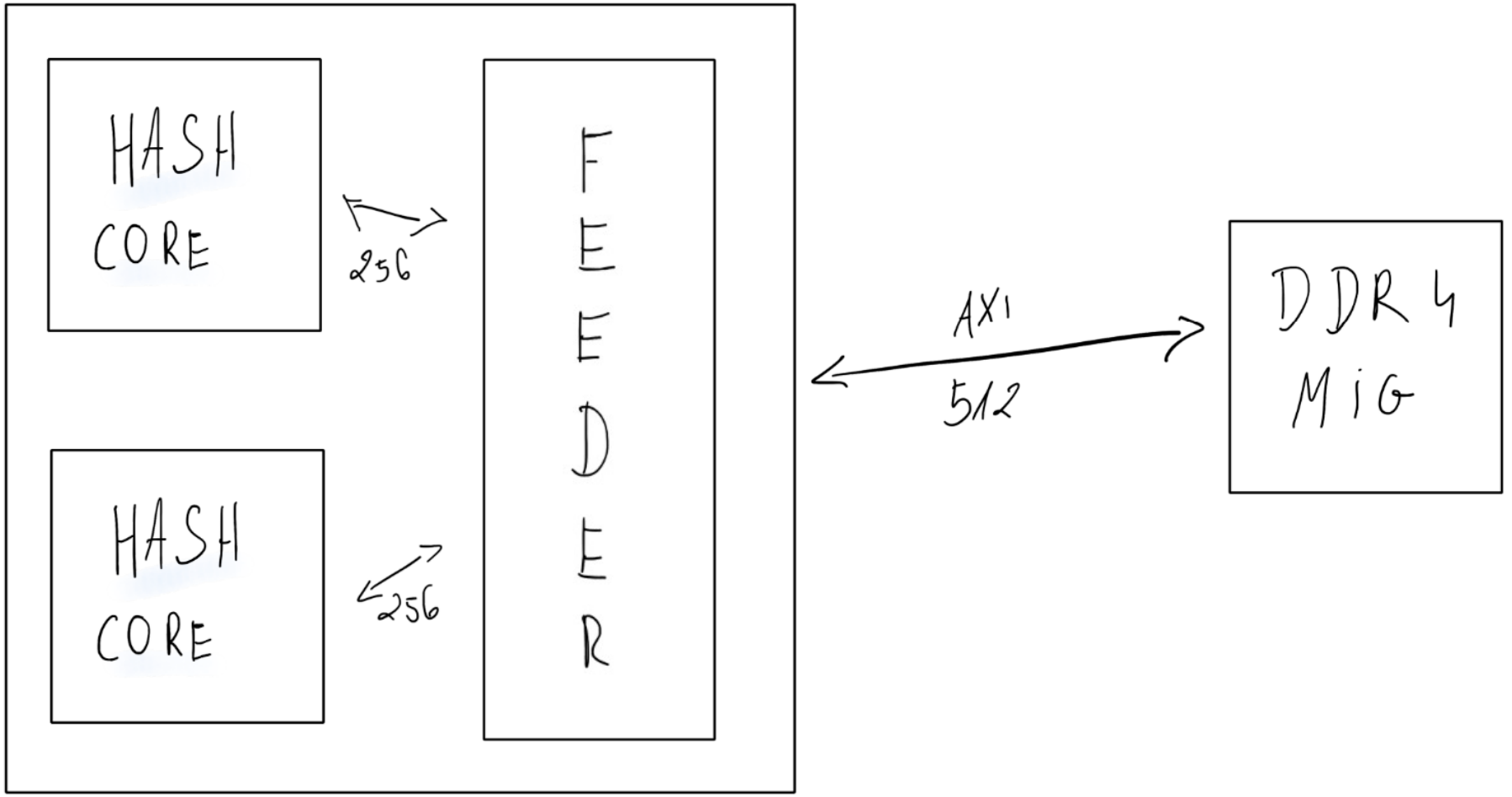
AWS SHELL


SH
PCIe (Gen3 x16) w/DMA
FPGA Shell (SH)
M
S
M
M
MGT PF
M
DDR4 - C
S
Misc
DMA_PCIS
AXI-4
(512-bit)
PCIM
AXI-4
(512-bit)
BAR1
AXI-L
(32-bit)
OCL
AXI-L
(32-bit)
(int, status, vLED/
vDIP, see
Miscellaneous
Signals)
SDA
AXI-L
(32-bit)
DDR
AXI-4
(512 bit)
S
M
S
S
S
M
M
AXI-4
(512 bit)
S
DDR4 – A,B,D
custom logic (CL)
CL

INTERFEJSY PAMIECI W F1 – DOBRE PRAKTYKI

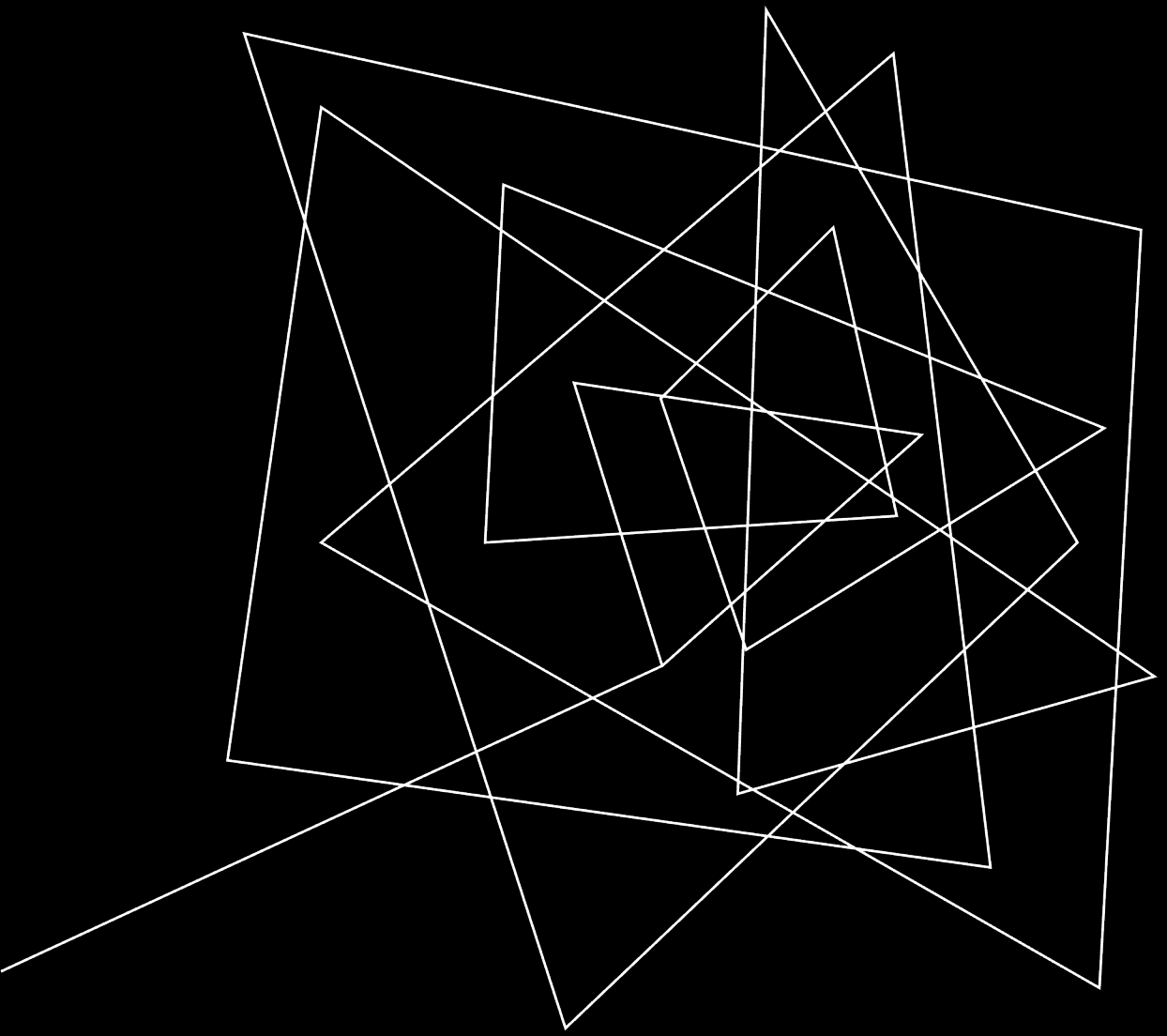


INTERFEJSY PAMIECI W F1 – DOBRE PRAKTYKI



CO MA ZNACZENIE?

- Ilość elementów logicznych.
- Częstotliwość taktowania zegara.
- Ilość i szerokość kanałów DDR4.

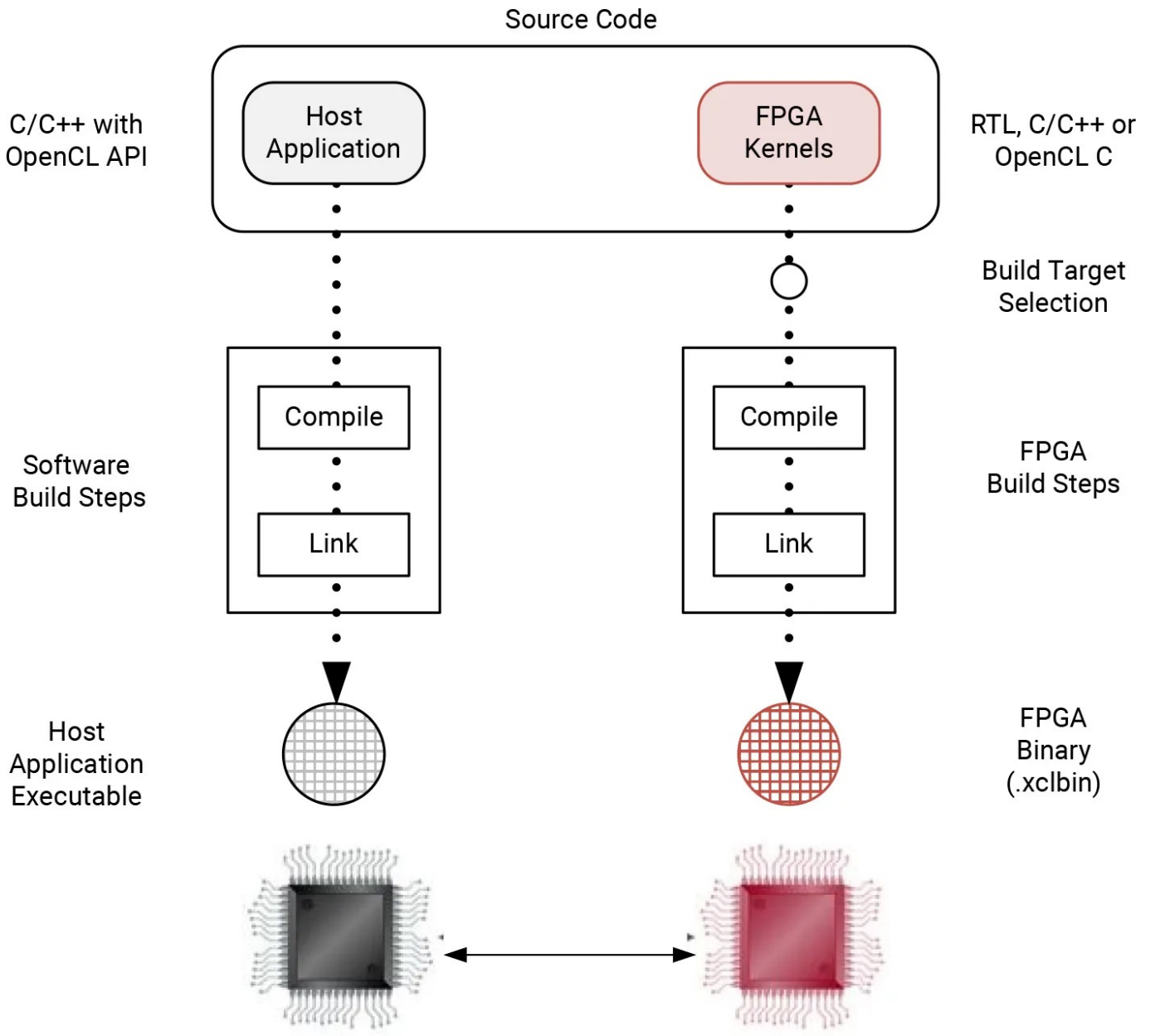


FPGA W CHMURZE - DEVELOPMENT

CZEGO POTRZEBUJEMY?

- Idealnie? Superkomputer lub gotowa instancje C5
- Ubuntu 21.04,
- IDE Vitis 2021.1,
- AWS FPGA SDK.

HOST <-> DEVICE


Source Code
C/C++ with OpenCL API
Host Application
FPGA Kernels
RTL, C/C++ or OpenCL C
Build Target Selection
Software Build Steps
Compile
Link
Compile
Link
FPGA Build Steps
Host Application Executable
FPGA Binary (.xclbin)

PRZYKLADOWY FRAMGMENT KODU KERNELA

```
extern "C" void hmacSha1Kernel_1(ap_uint<512> in, ap_uint<512> out) {
#pragma HLS dataflow
[...]
readIn<_burstLength, _channelNumber>(in, textInStrm, textLengthStrm, textNumStrm, keyInStrm);

splitInput<_channelNumber, _burstLength>(textInStrm, textLengthStrm, textNumStrm, keyInStrm,
keyStrm, msgStrm, msgLenStrm, eMsgLenStrm);

hmacSha1Parallel<_channelNumber>(keyStrm, msgStrm, msgLenStrm, eMsgLenStrm, hshStrm, eHshStrm);

mergeResult<_channelNumber, _burstLength>(hshStrm, eHshStrm, outStrm, burstLenStrm);

writeOut<_burstLength, _channelNumber>(outStrm, burstLenStrm, out);
}
```

KOMPILOWANIE KERNELA

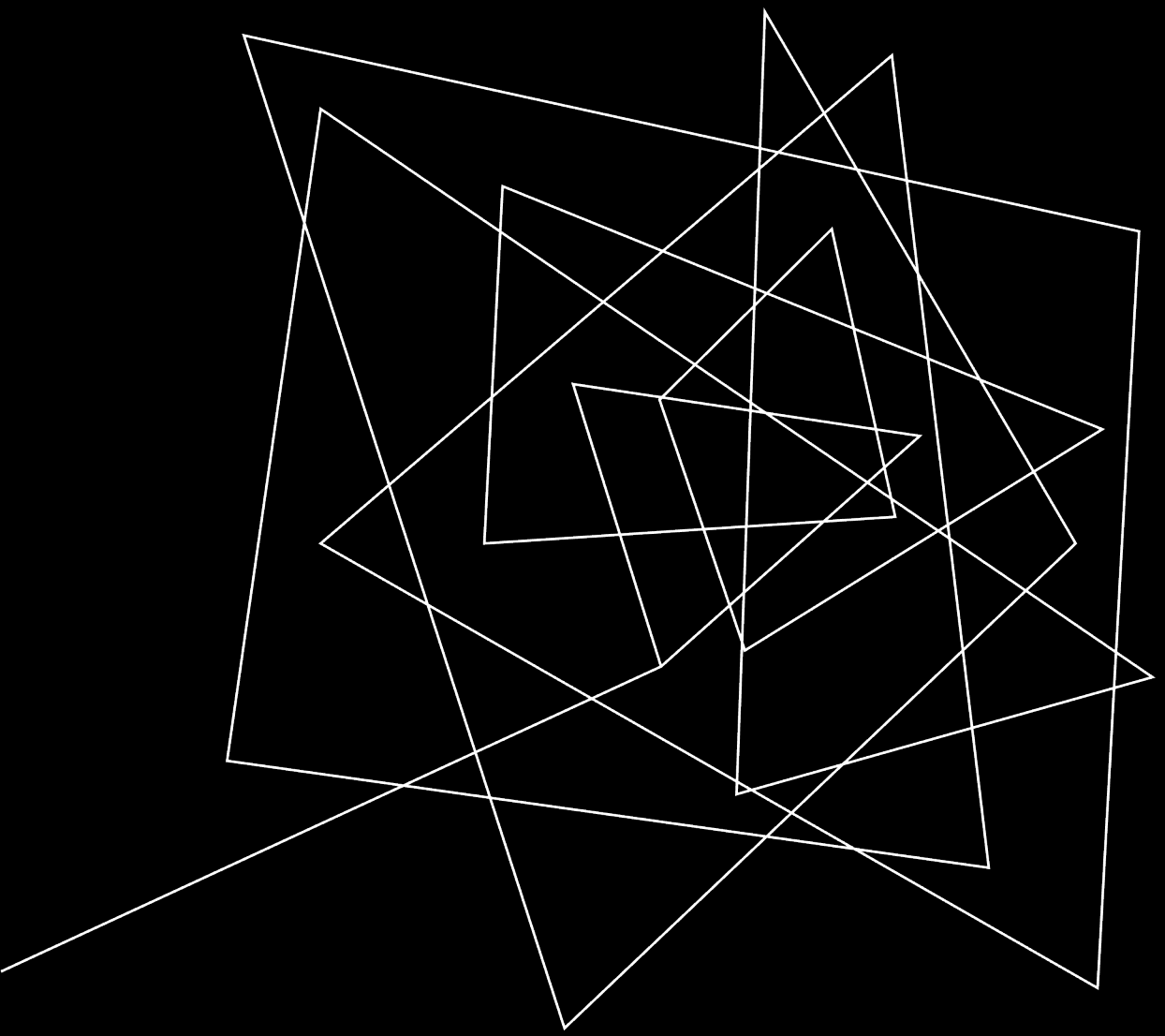
1. Synthesis,
2. Optimalization,
3. Placement,
4. Routing,
5. Implementation.

DLACZEGO WLASNIE TAK?

1. Opóźnienia w transmisji.
2. Duże zużycie zasobów.

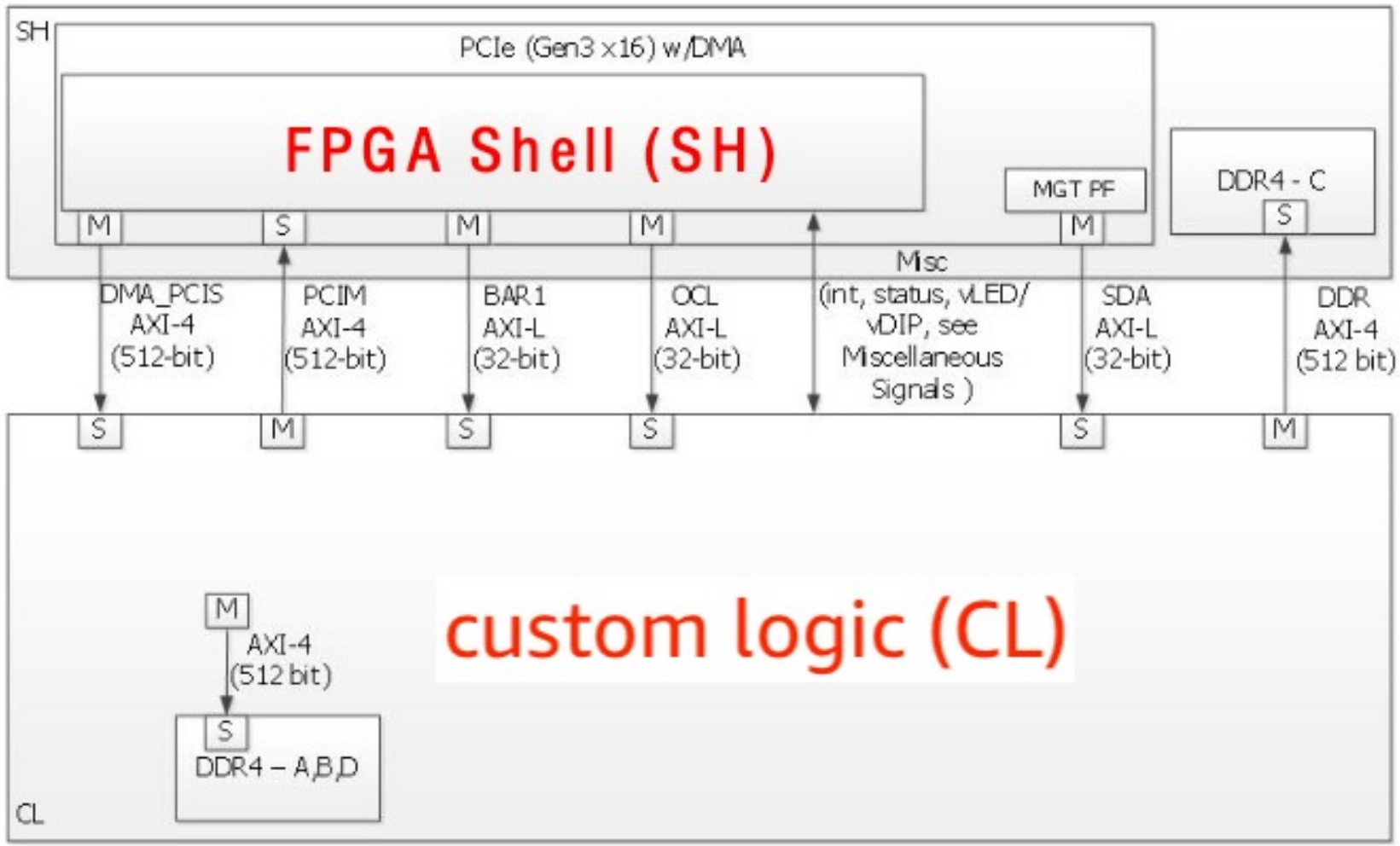
JAK MOZEMY TESTOWAC?

- Lokalnie
  - Software Emulation,
  - Hardware Emulation.
- Zdalnie
  - Hardware



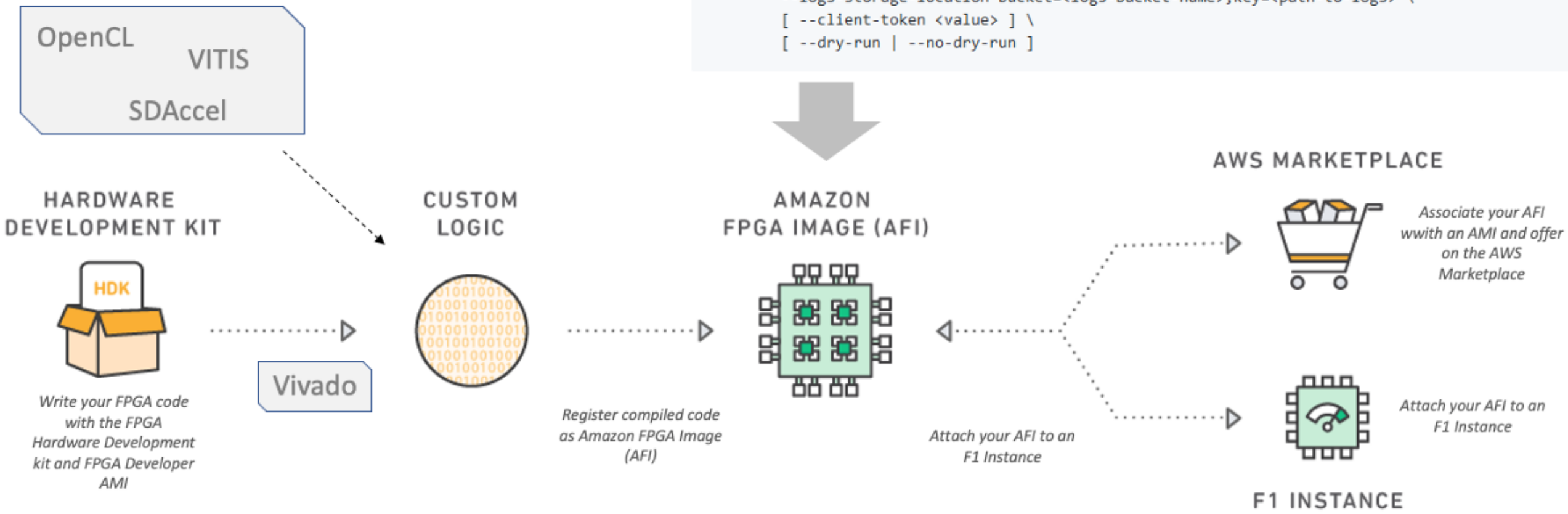
FPGA W CHMURZE - DEPLOYMENT

AWS SHELL


SH
PCIe (Gen3 x16) w/DMA
FPGA Shell (SH)
M
S
M
M
MGT PF
M
DDR4 - C
S
Misc
DMA_PCIS
AXI-4
(512-bit)
PCIM
AXI-4
(512-bit)
BAR1
AXI-L
(32-bit)
OCL
AXI-L
(32-bit)
(int, status, vLED/
vDIP, see
Miscellaneous
Signals)
SDA
AXI-L
(32-bit)
DDR
AXI-4
(512 bit)
S
M
S
S
S
M
M
AXI-4
(512 bit)
S
DDR4 – A,B,D
custom logic (CL)
CL

AMAZON FPGA IMAGE

```
$ aws ec2 create-fpga-image \
    --name <afi-name> \
    --description <afi-description> \
    --input-storage-location Bucket=<dcp-bucket-name>,Key=<path-to-tarball> \
    --logs-storage-location Bucket=<logs-bucket-name>,Key=<path-to-logs> \
    [ --client-token <value> ] \
    [ --dry-run | --no-dry-run ]
```



Amazon FPGA Developer AMI	Custom AMI
C5 / M5 instances	F1 instances

JAK?

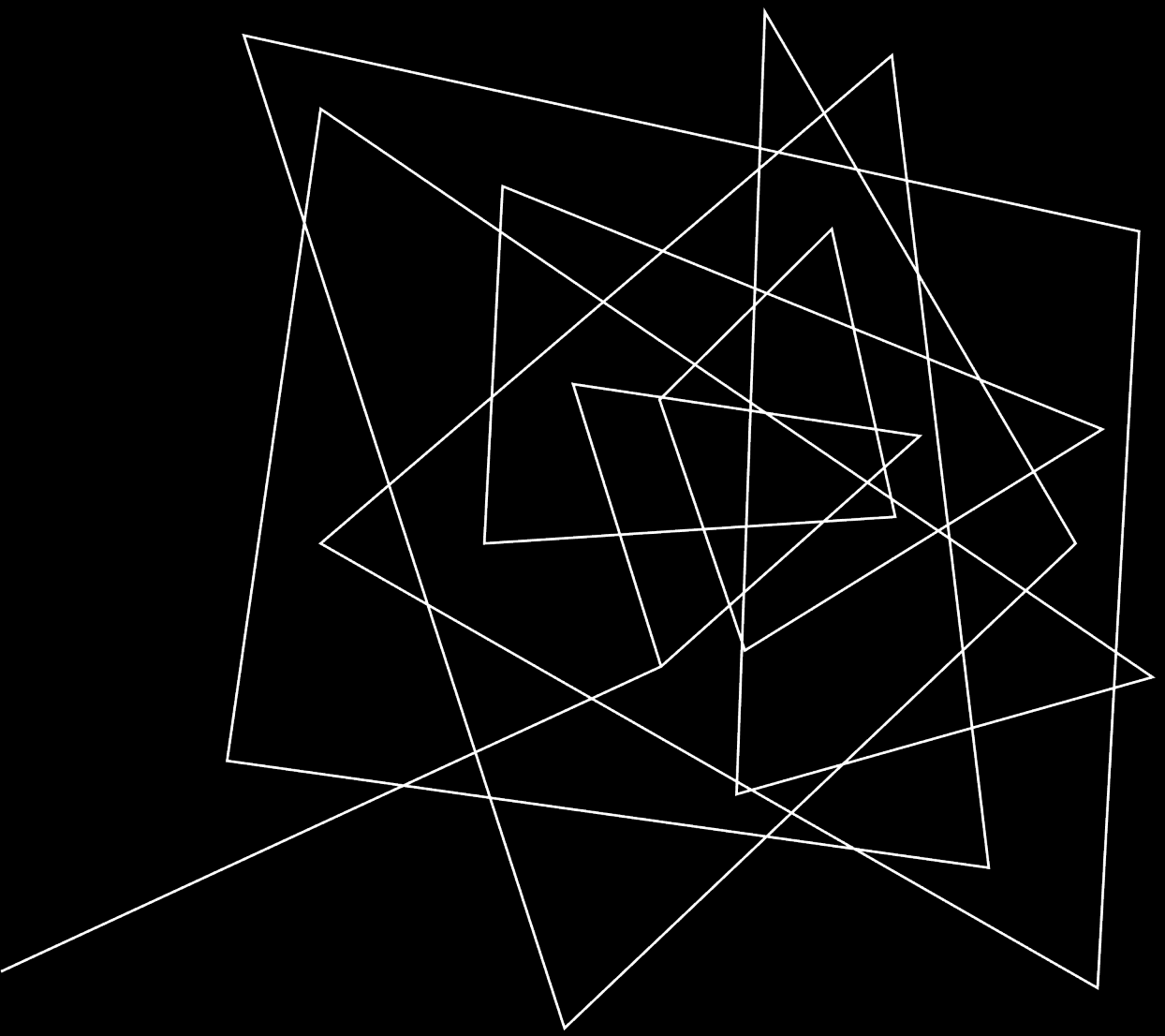
```
$VITIS_DIR/tools/create_vitis_afi.sh \
    -xclbin=<input_xilinx_fpga_binary_xclbin_filename>
    -o=<output_aws_fpga_binary_awsxclbin_filename_root> \
    -s3_bucket=<bckt> -s3_dcp_key=<dcp_dir> -s3_logs_key=<logs_dir>
```

JAK?

```
$VITIS_DIR/tools/create_vitis_afi.sh \
    -xclbin=<input_xilinx_fpga_binary_xclbin_filename>
    -o=<output_aws_fpga_binary_awsxclbin_filename_root> \
    -s3_bucket=<bckt> -s3_dcp_key=<dcp_dir> -s3_logs_key=<logs_dir>

aws ec2 describe-fpga-images --fpga-image-ids <AFI ID>
```

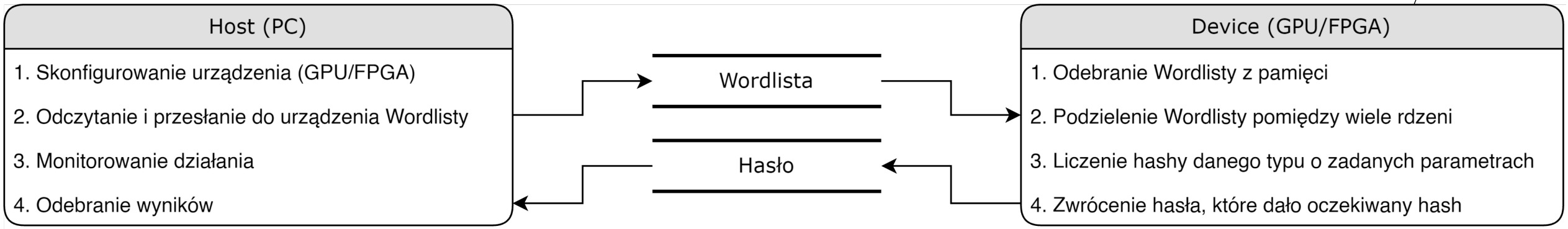
JAK?

```
$VITIS_DIR/tools/create_vitis_afi.sh \
    -xclbin=<input_xilinx_fpga_binary_xclbin_filename>
    -o=<output_aws_fpga_binary_awsxclbin_filename_root> \
    -s3_bucket=<bckt> -s3_dcp_key=<dcp_dir> -s3_logs_key=<logs_dir>

aws ec2 describe-fpga-images --fpga-image-ids <AFI ID>


...
    "State": {
        "Code": "available"
    },
...
```



CRACKOWANIE HASHY NA FPGA

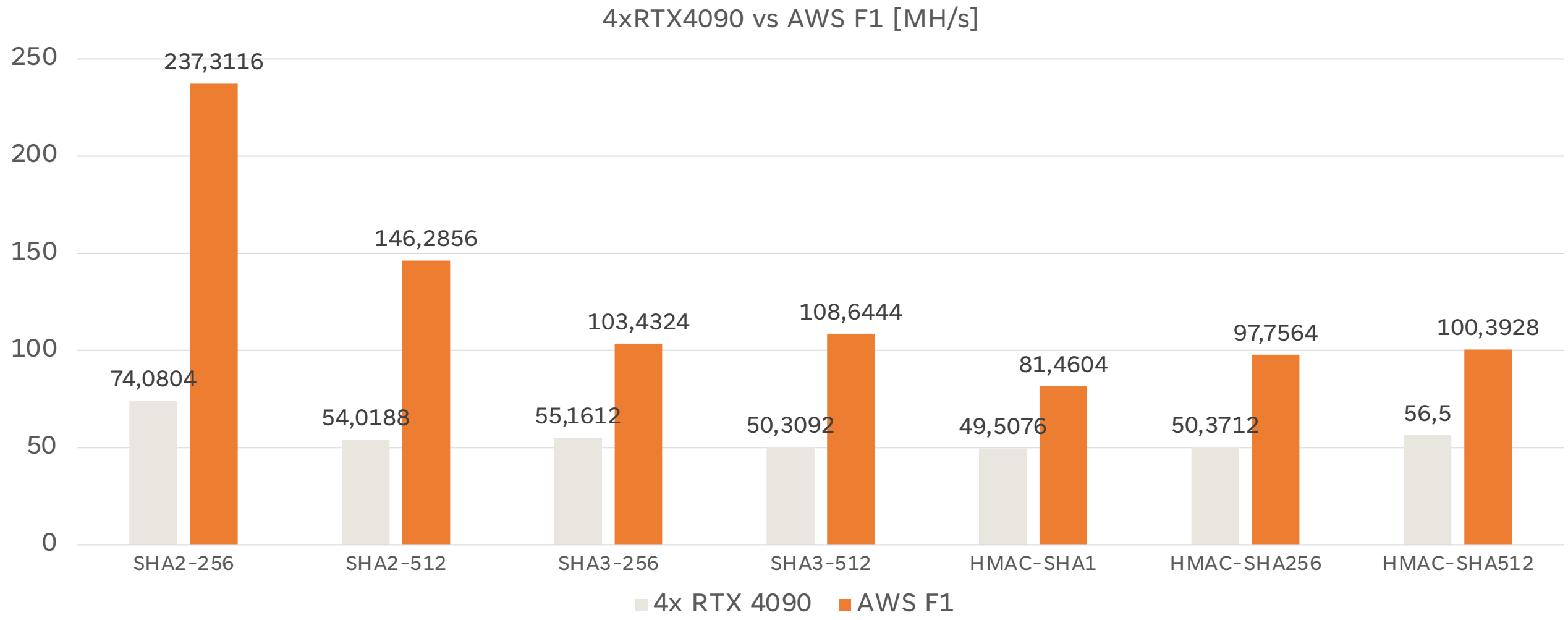
SCHEMAT OGÓLNY LAMANIA HASHY (ATAK SLOWNIKOWY)



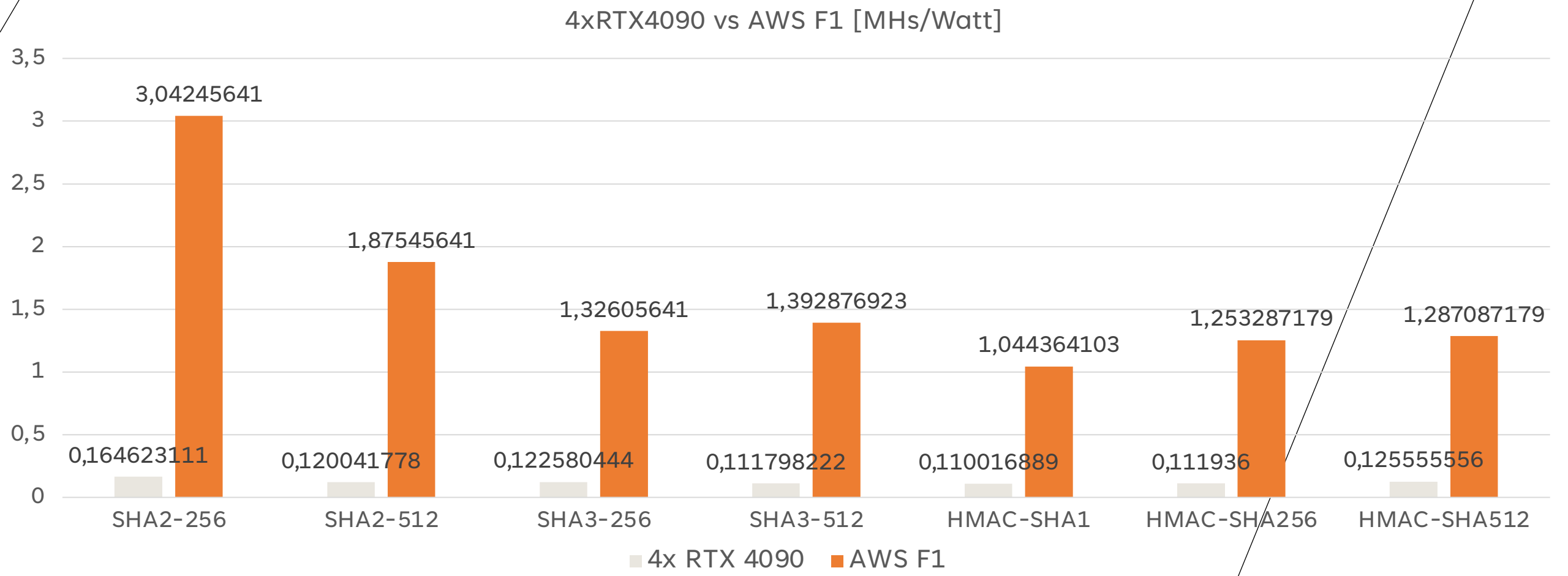
UKLAD IDEALNY?

- Konfiguracja „uszyta na miarę”.
- W 100% wykorzystuje pipelining.
- Wykorzystuje maks. 10% zasobów.
- Wielokrotnego użytku.

WYNIKI

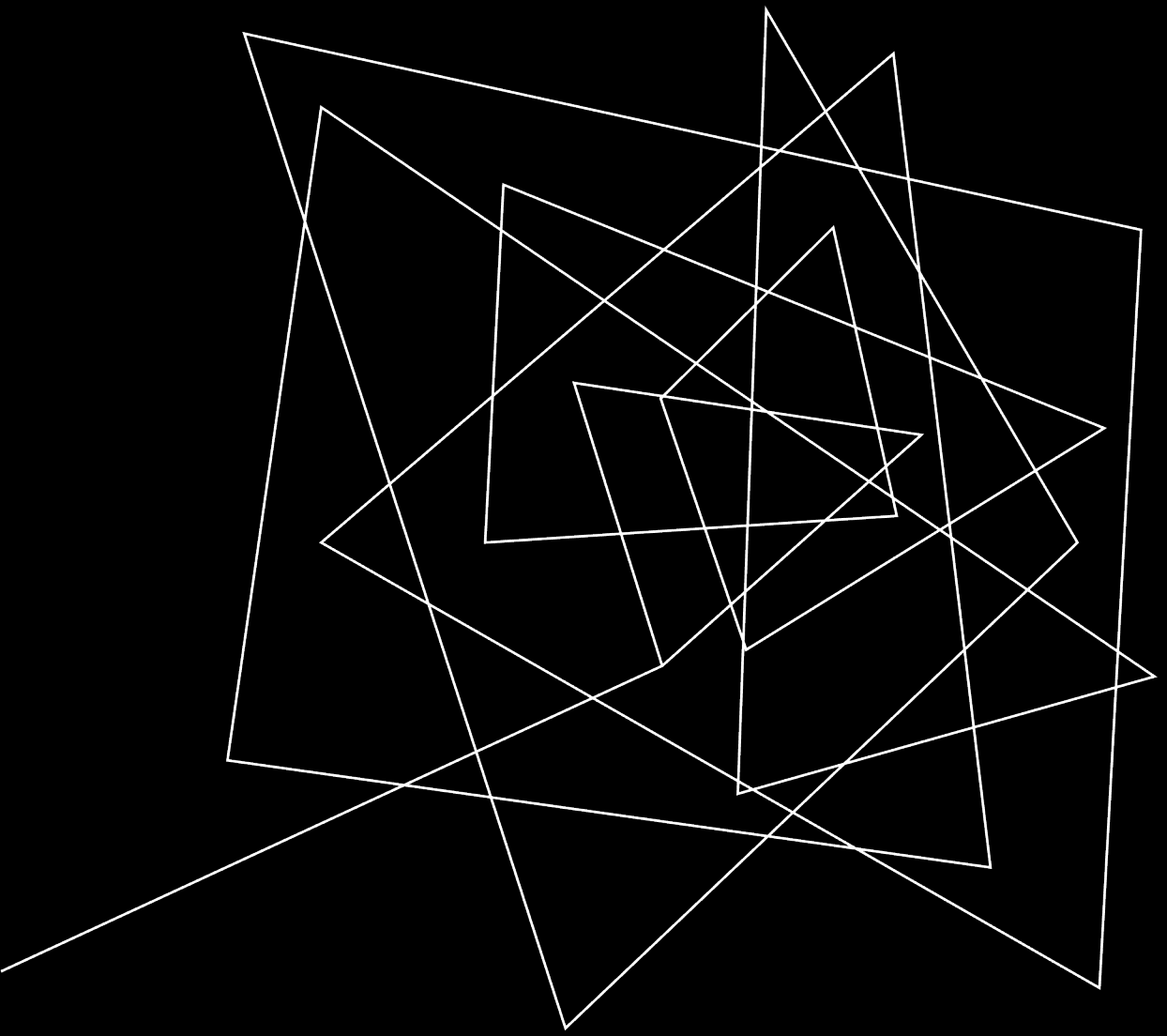

4xRTX4090 vs AWS F1 [MH/s]
250
200
150
100
50
0
237,3116
146,2856
103,4324
108,6444
97,7564
100,3928
74,0804
54,0188
55,1612
50,3092
81,4604
49,5076
50,3712
56,5
SHA2-256
SHA2-512
SHA3-256
SHA3-512
HMAC-SHA1
HMAC-SHA256
HMAC-SHA512
4x RTX 4090
AWS F1

WYNIKI


4xRTX4090 vs AWS F1 [MHs/Watt]
3,5
3
2,5
2
1,5
1
0,5
0
3,04245641
1,87545641
1,32605641
1,392876923
1,044364103
1,253287179
1,287087179
0,164623111
0,120041778
0,122580444
0,111798222
0,110016889
0,111936
0,125555556
SHA2-256
SHA2-512
SHA3-256
SHA3-512
HMAC-SHA1
HMAC-SHA256
HMAC-SHA512
4x RTX 4090
AWS F1

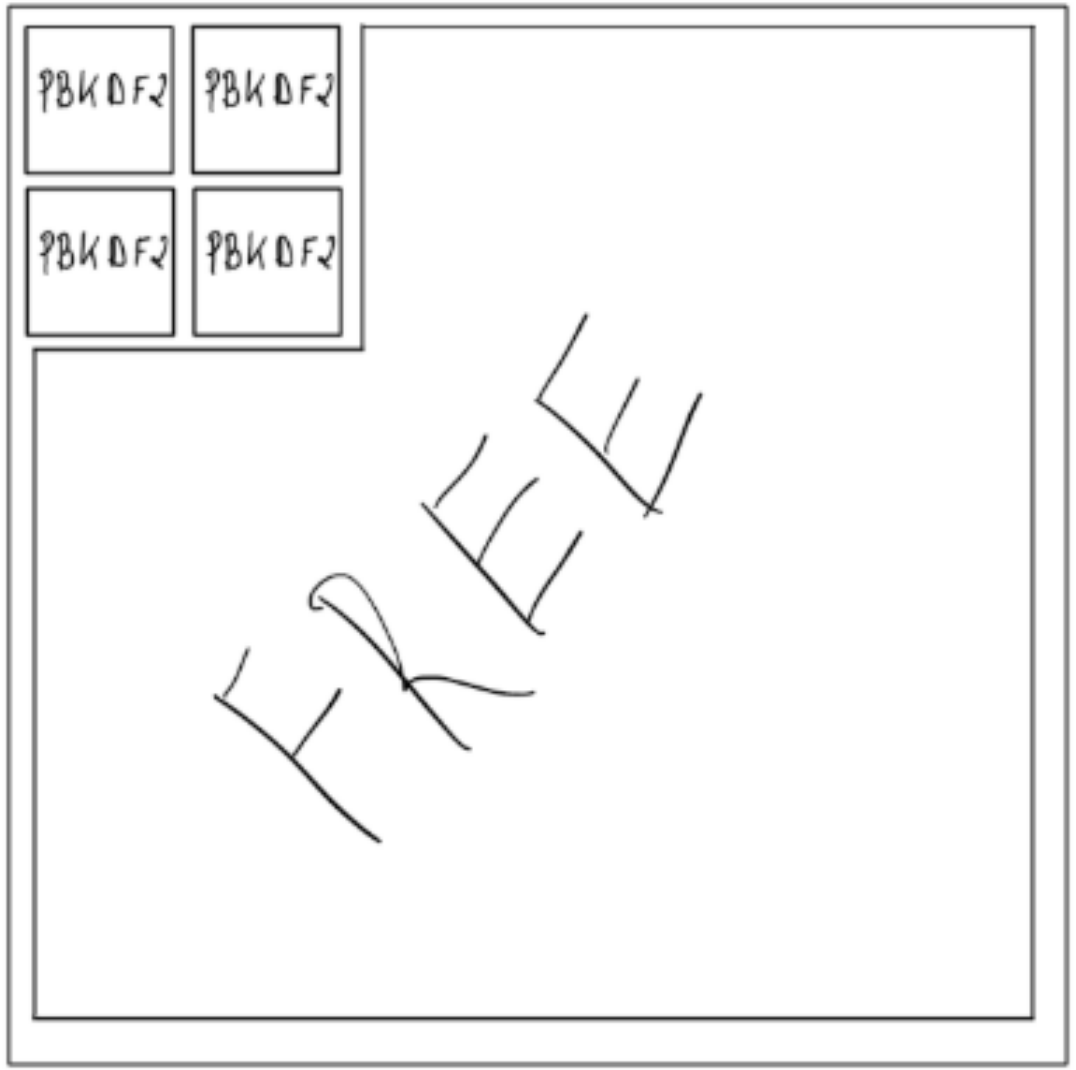
CO DALEJ?

- Skupić się na trudniejszych hashach.
- Zoptymalizować hashcore.
- Może HDL?
- Zrobić porządek w repo :D



SZCZEGÓLNE WLASCIWOSCI

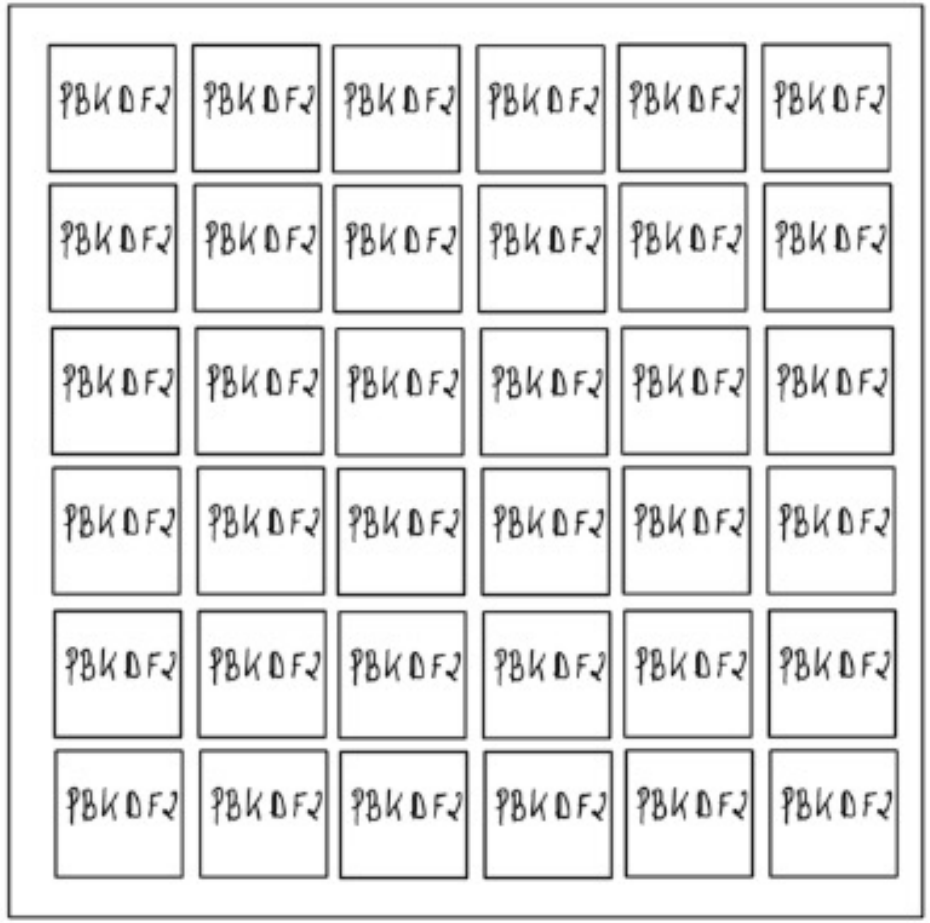
DICTIONARY ATTACK



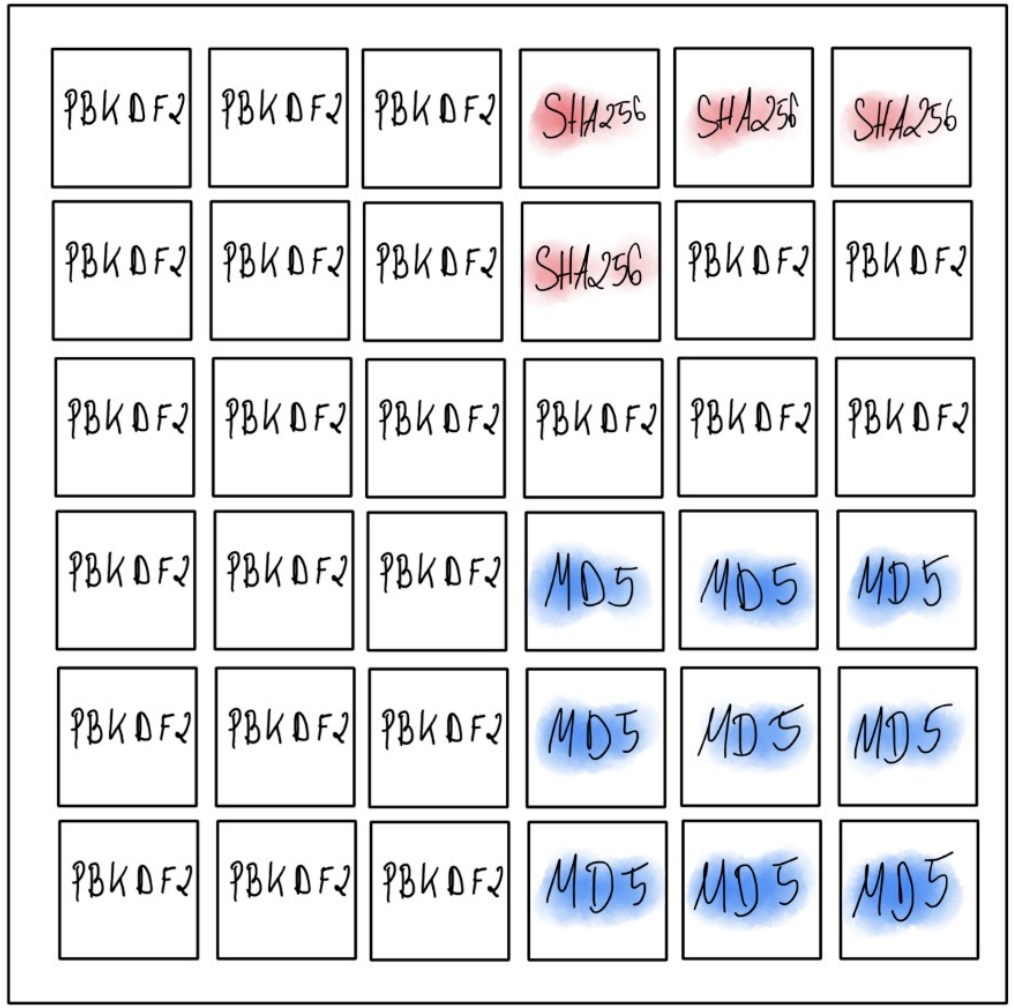
OBEJSCIE PROBLEMU INTERFEJSOW

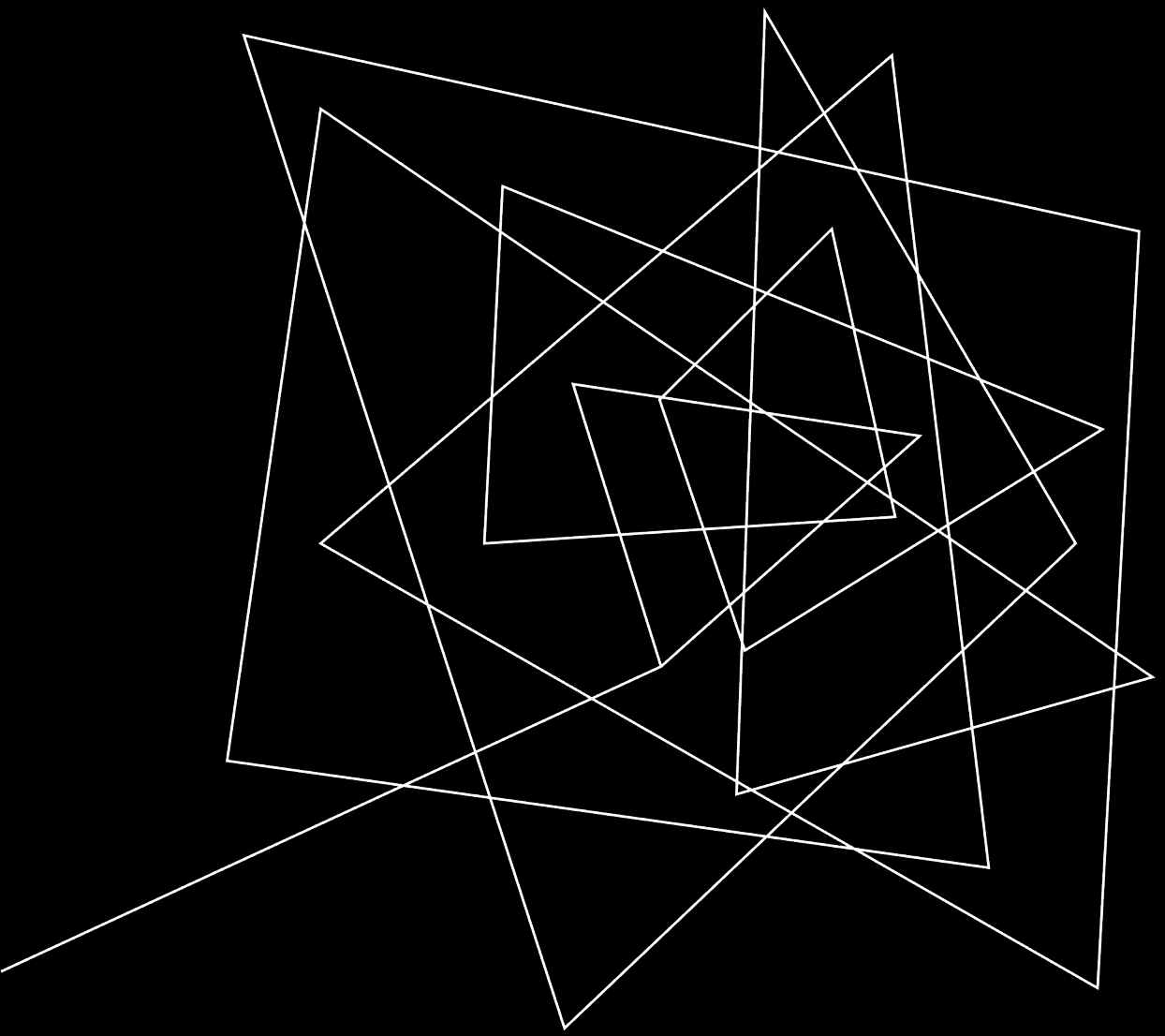
- Kiedy potrzebujemy dostępu do pamięci?
- Jakie istnieją sposoby ataków na hashe?
- Co nam to daje? Jaki jest profit?

BRUTE-FORCE ATTACK



WIELODOSTEPOWOSC





NIE WSZYSTKO JEST TAKIE WESOLE

WADY

- Bardzo skomplikowany development.
- Czas kompilowania.
- Konfiguracja musi być „szyta na miarę”.
- Drogie testowanie.

MATEUSZ LEWCZAK

GITHUB: @LEFTARCODE

BIBLIOGRAFIA

- Bai, Yu & Alawad, Mohammed & DeMara, Ronald & Lin, Mingjie. (2015). Optimally Fortifying Logic Reliability through Criticality Ranking. Electronics. 4. 150-172. 10.3390/electronics4010150.
- McEvoy, Robert & Crowe, Fionuala & Murphy, Colin & Marnane, William. (2006). Optimisation of the SHA-2 family of hash functions on FPGAs. Proceedings - IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures 2006. 2006. 317-322. 10.1109/ISVLSI.2006.70.
- Układy FPGA – Języki programowania – Politechnika Krakowska
  - http://mysinski.wieik.pk.edu.pl/MiUP/Uklady%20programowlane%20czesc%20II%20FPGA.pdf
- https://en.wikipedia.org/wiki/Instruction_pipelining