THE SOUND OF ONE HAND CLAPPING

An Overview of the Cell Matrix Architecture

Nicholas J. Macias
Cell Matrix Corporation

THE SOUND OF ONE HAND CLAPPING

An Overview of the Cell Matrix Architecture

Nicholas J. Macias
Cell Matrix Corporation

TALK OUTLINE

TALK OUTLINE

- Non-Exotic Features (D-Mode)

TALK OUTLINE

- Non-Exotic Features (D-Mode)
- Philosophy

TALK OUTLINE

- Non-Exotic Features (D-Mode)
- Philosophy
- Exotic Features (C-Mode)

TALK OUTLINE

- Non-Exotic Features (D-Mode)
- Philosophy
- Exotic Features (C-Mode)
- Implications

TALK OUTLINE

- Non-Exotic Features (D-Mode)
- Philosophy
- Exotic Features (C-Mode)
- Implications
- The Future

I. BASIC CELL MATRIX FEATURES

WHAT IS A CELL MATRIX?

- Reconfigurable Hardware System

WHAT IS A CELL MATRIX?

- Reconfigurable Hardware System
- Fine-Grained

WHAT IS A CELL MATRIX?

- Reconfigurable Hardware System
- Fine-Grained
- Simple

WHAT IS A CELL MATRIX?

- Reconfigurable Hardware System
- Fine-Grained
- Simple
- General-purpose

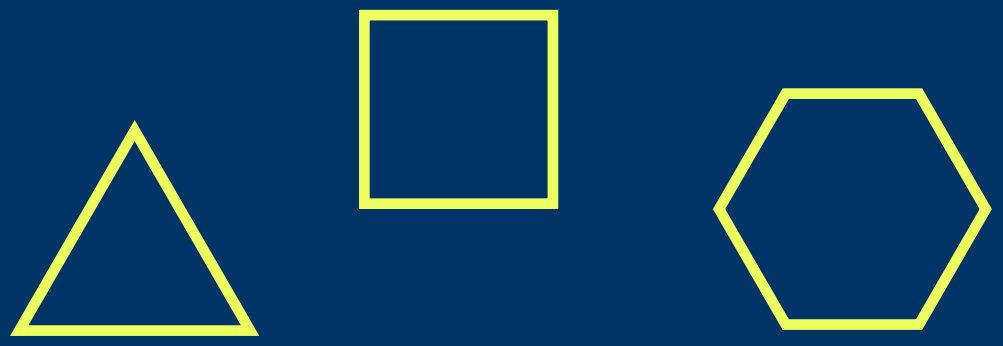
CELLS COME IN MANY FLAVORS

Number of Sides

CELLS COME IN MANY FLAVORS

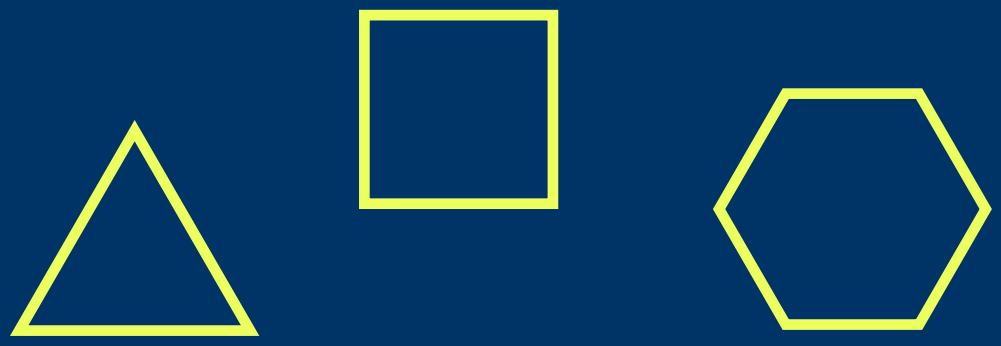
Number of Sides

CELLS COME IN MANY FLAVORS

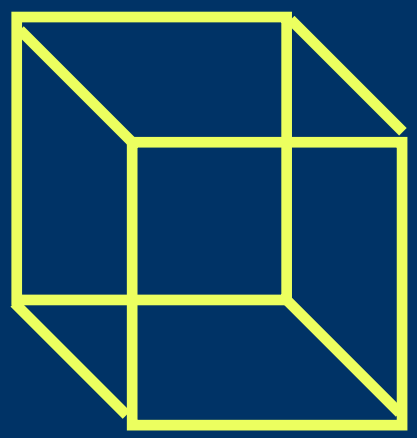


Number of Sides

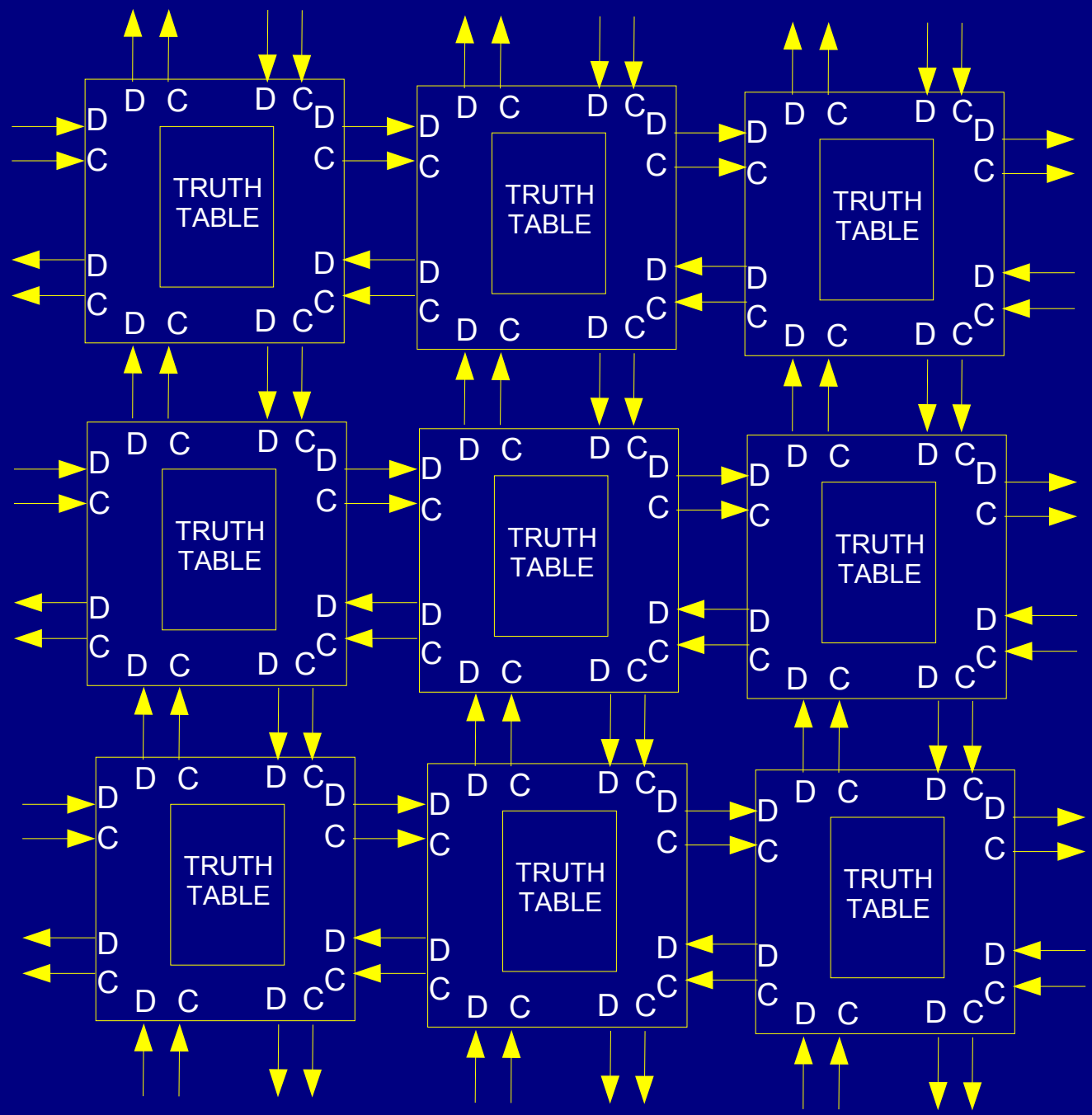
CELLS COME IN MANY FLAVORS

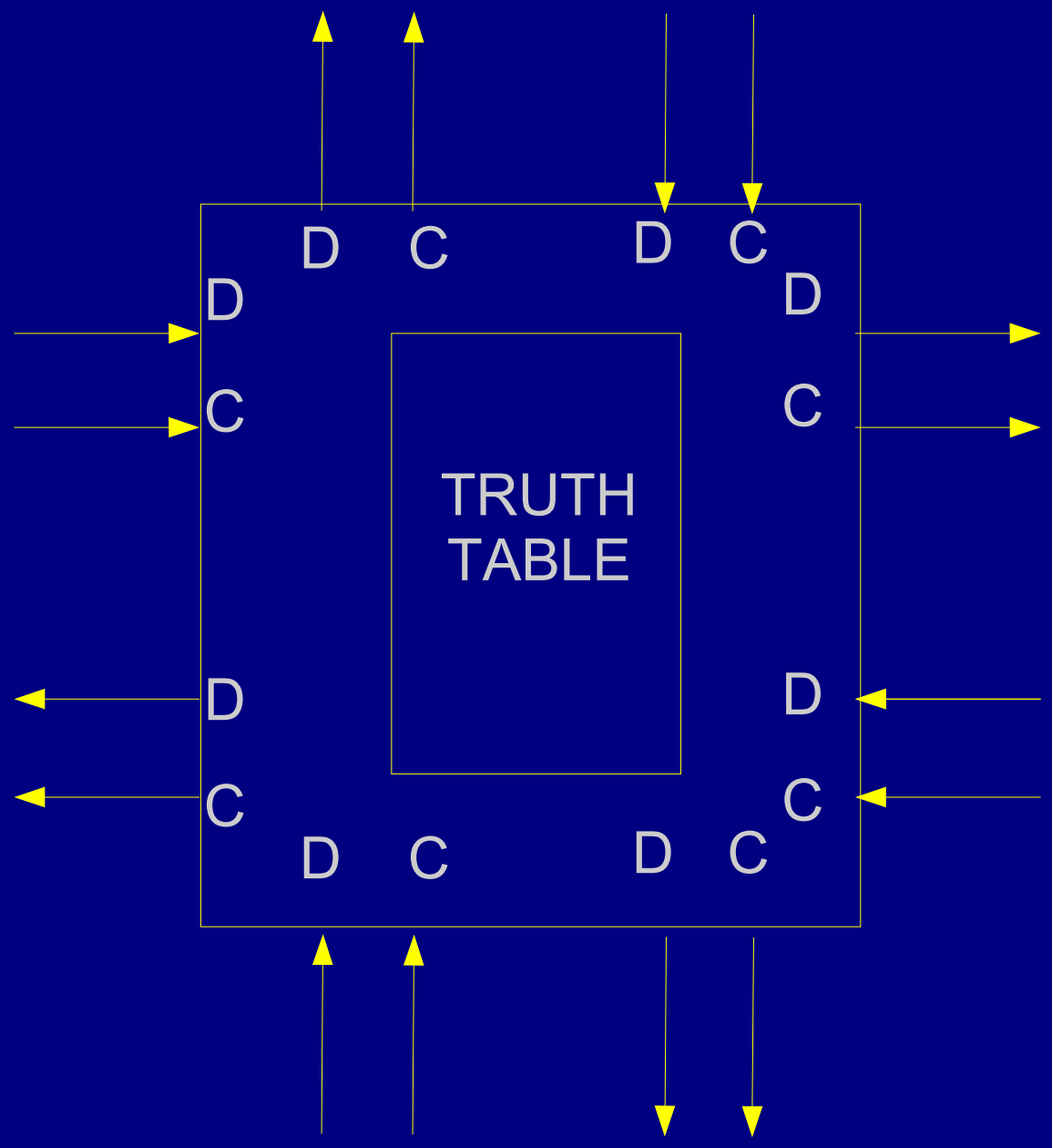


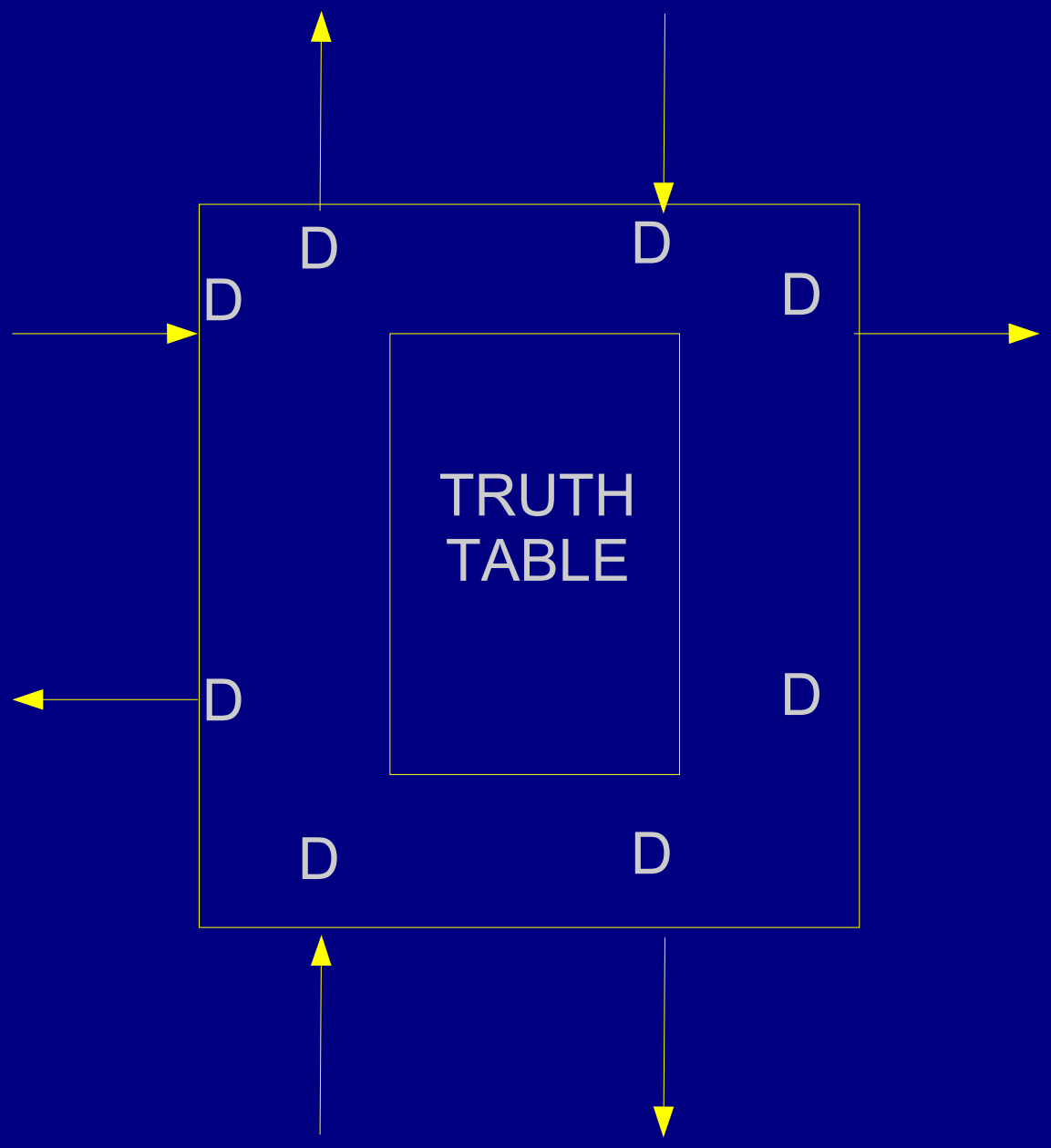
Number of Sides



Dimensionality


TRUTH TABLE
D C

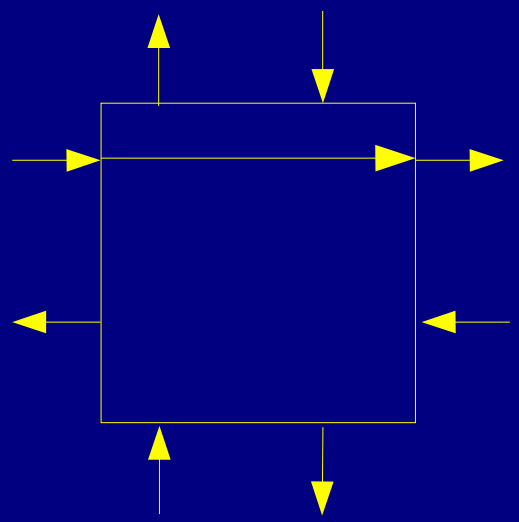

D
C
D
C
D
C
D
C
TRUTH
TABLE
D
C
D
C
D
C
D
C


D
D
D
D
TRUTH
TABLE
D
D
D
D

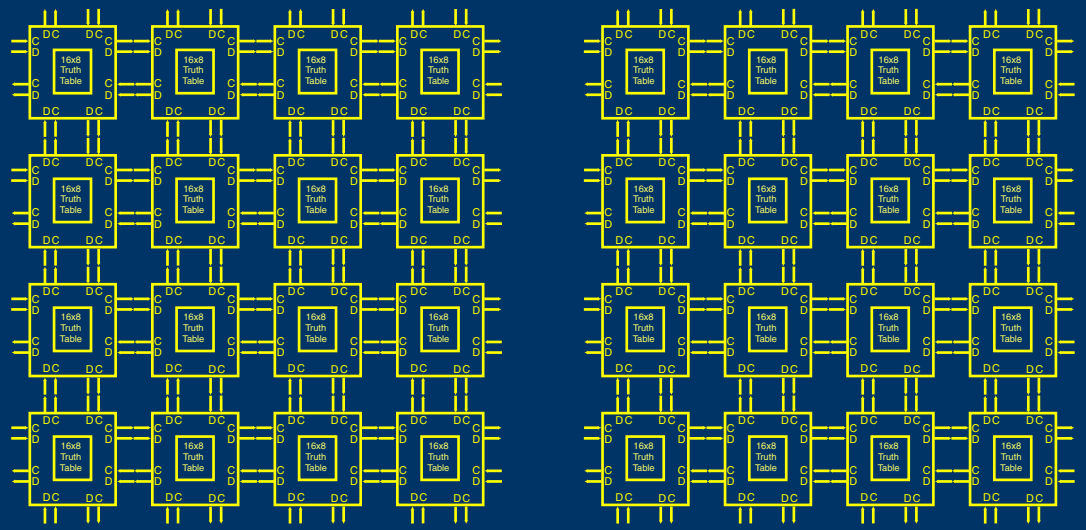
EXAMPLE: CELL CONFIGURED TO BE A WIRE

SAMPLE TRUTH TABLE

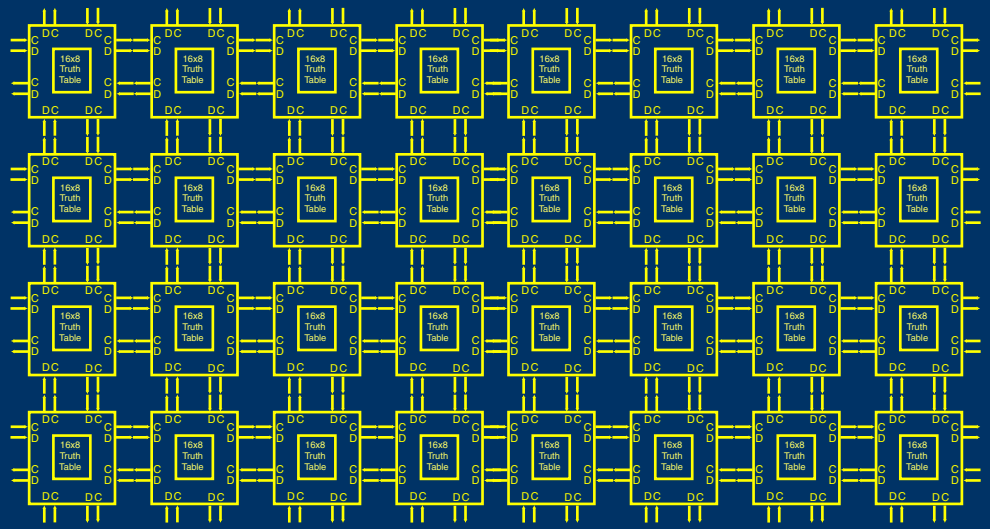
INPUTS				OUTPUTS			
DN	DS	DW	DE	DN	DS	DW	DE
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	1
0	1	1	1	0	0	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	1
1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	1



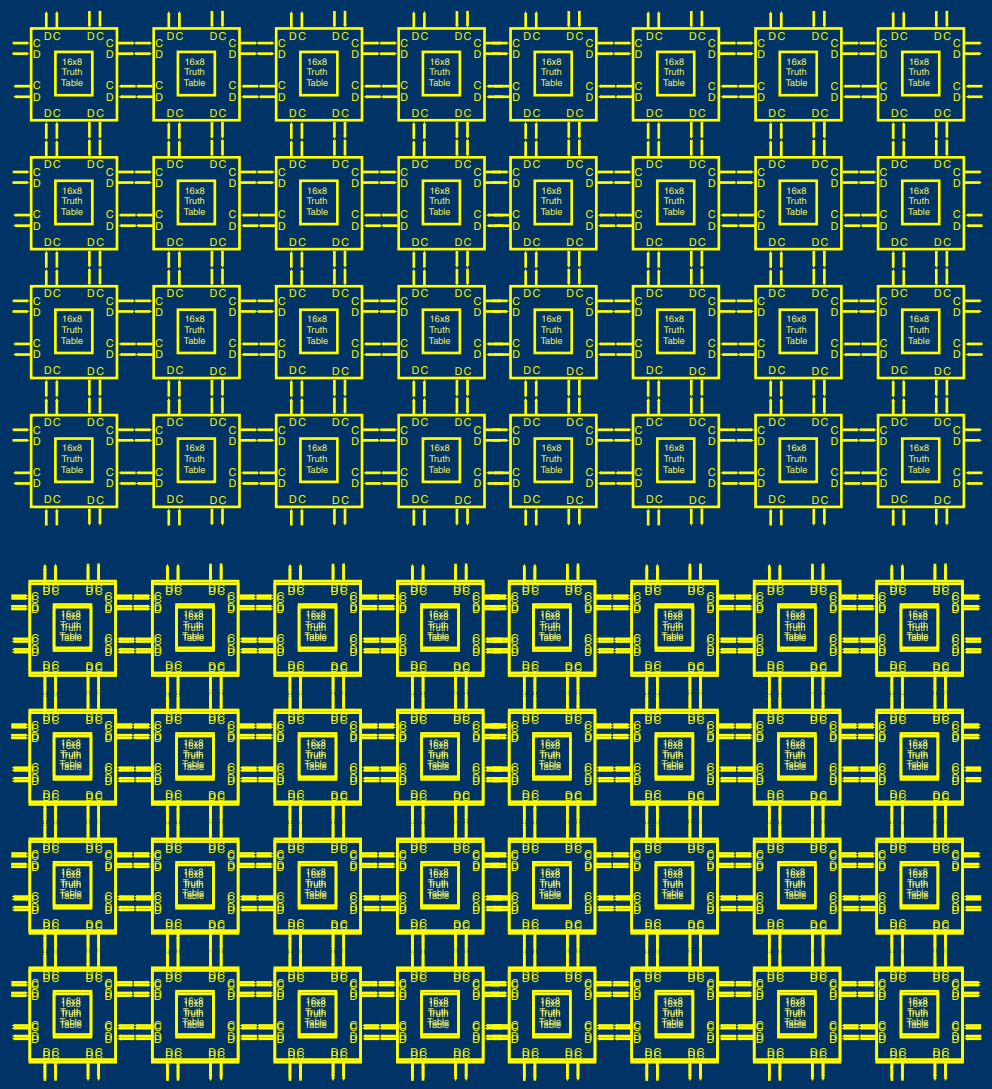
Scaling of the Matrix



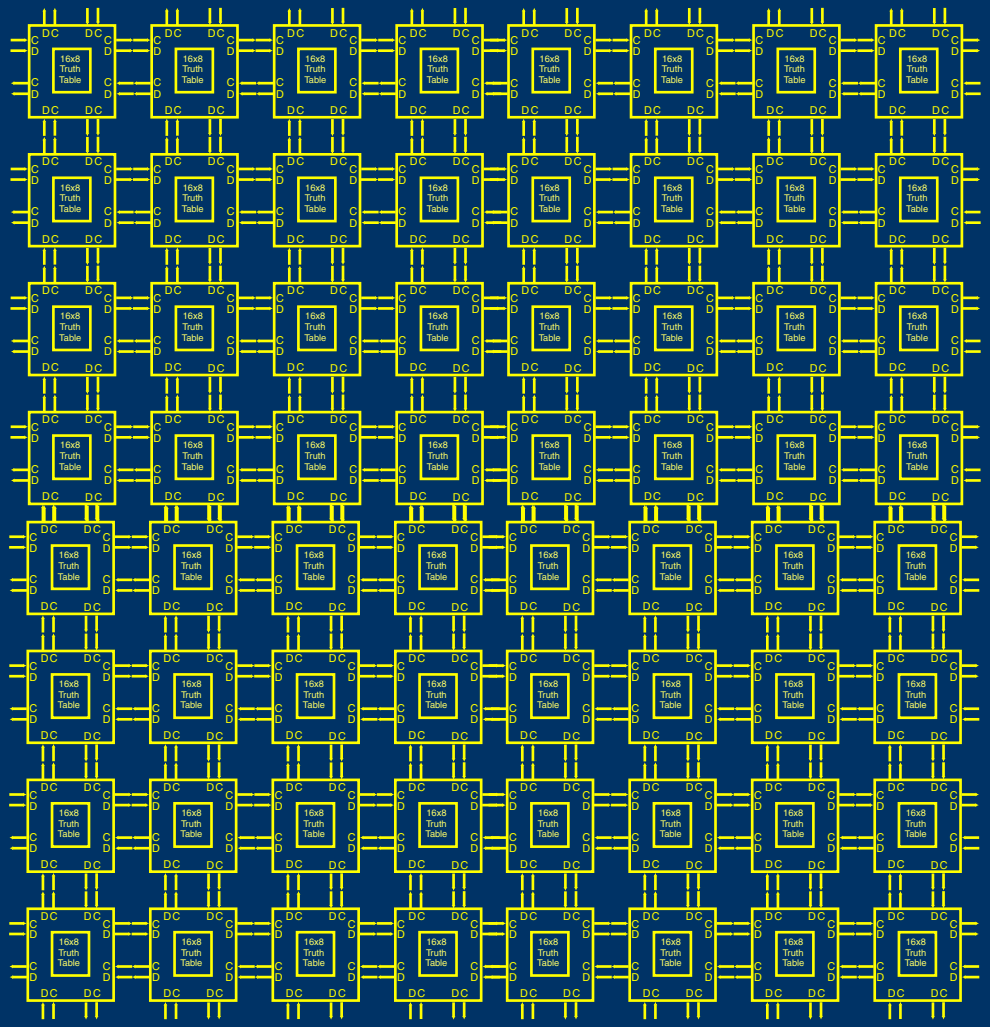
Scaling of the Matrix



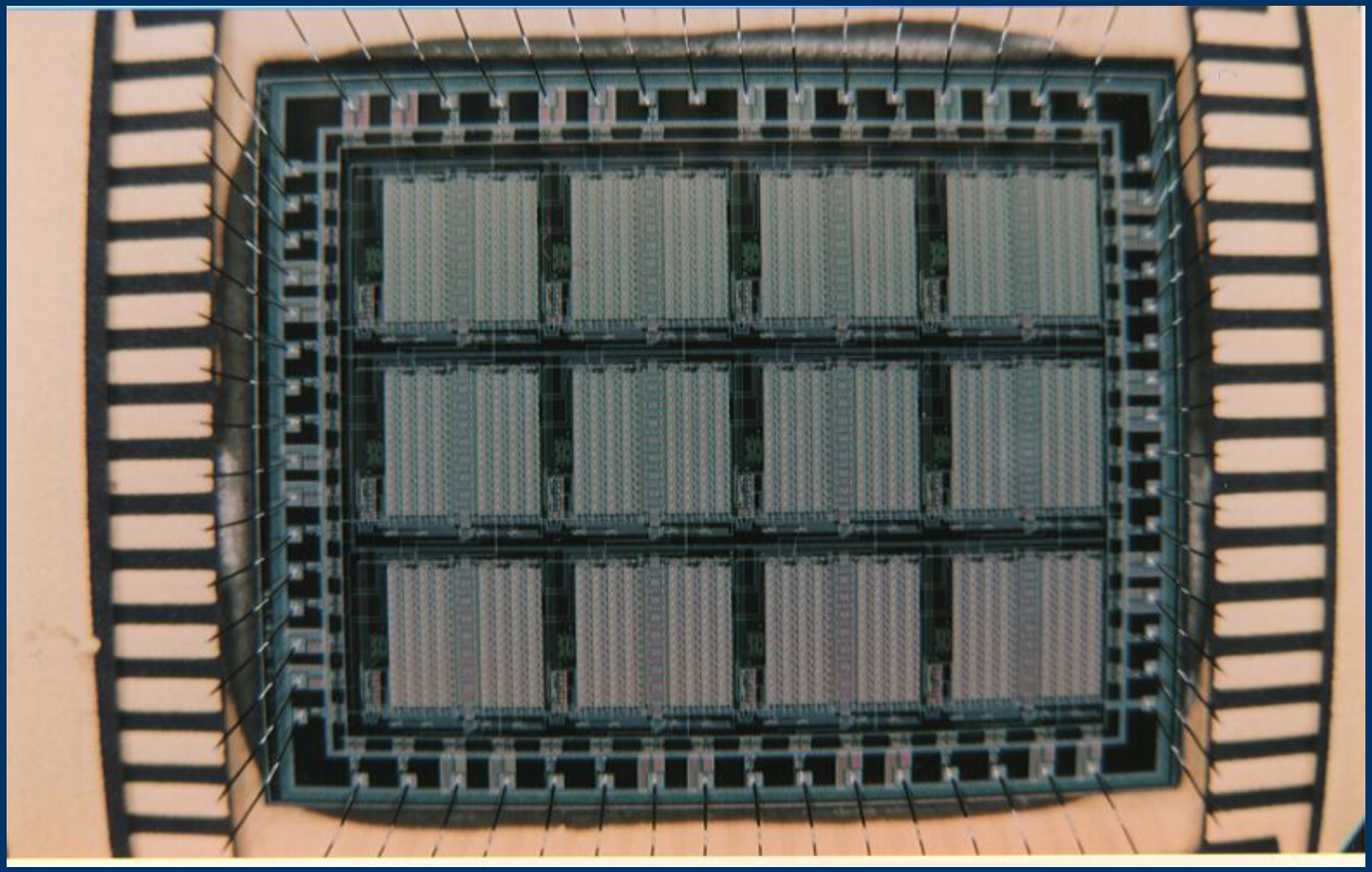
Scaling of the Matrix



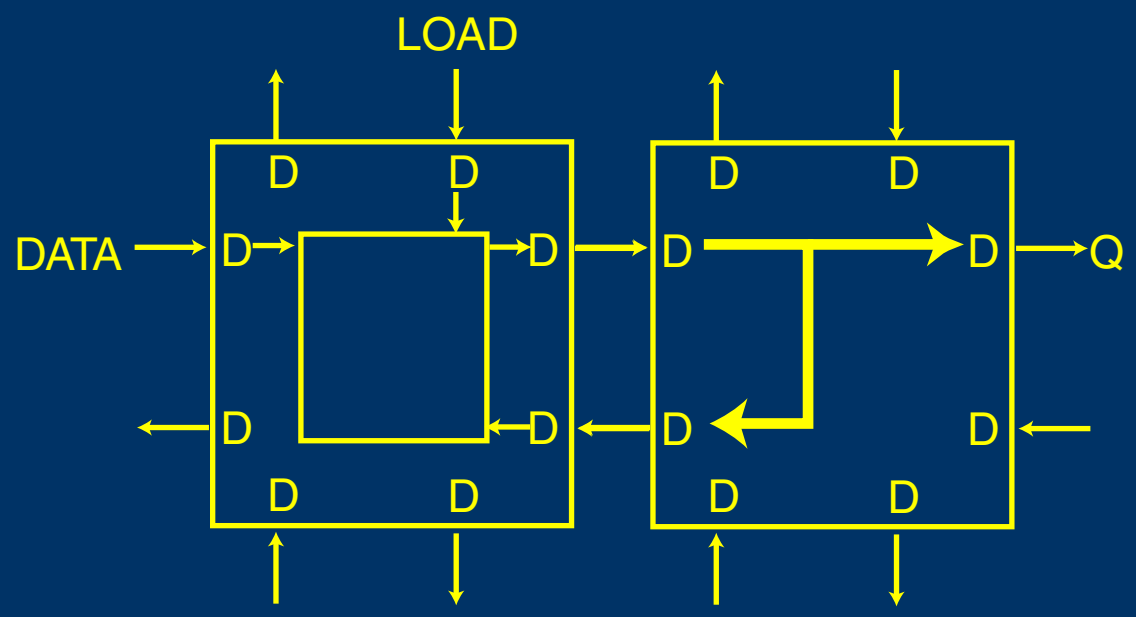
Scaling of the Matrix



FAULT ISOLATION

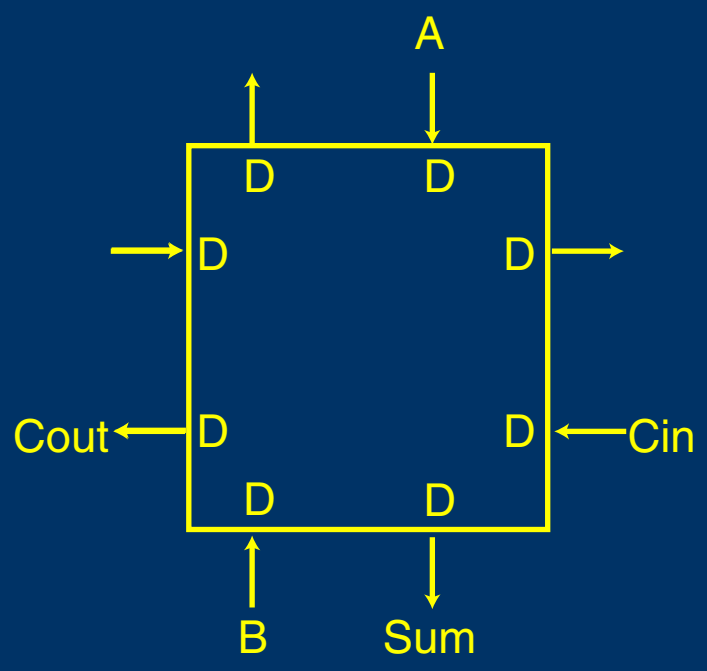


SIMPLE FLIP FLOP

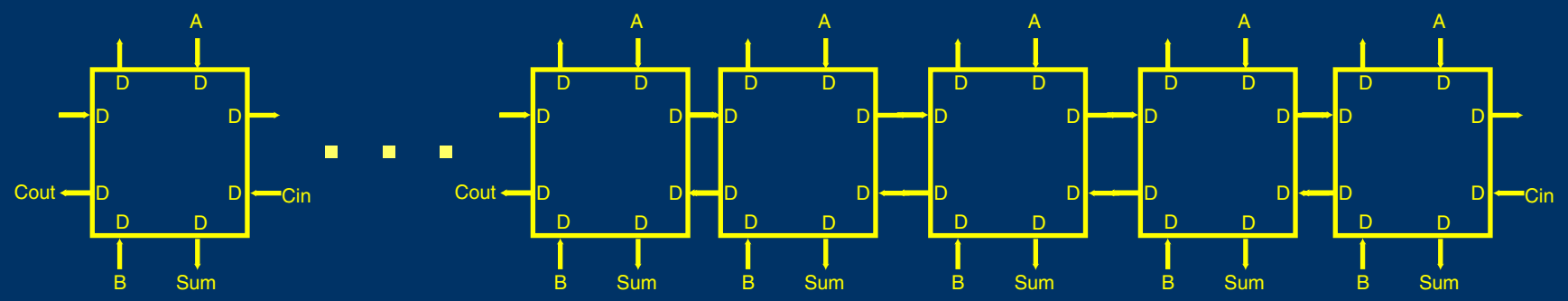


$$DW_{out}=(DN_{in}.DW_{in})+(\overline{DN}_{in}.DE_{in})$$

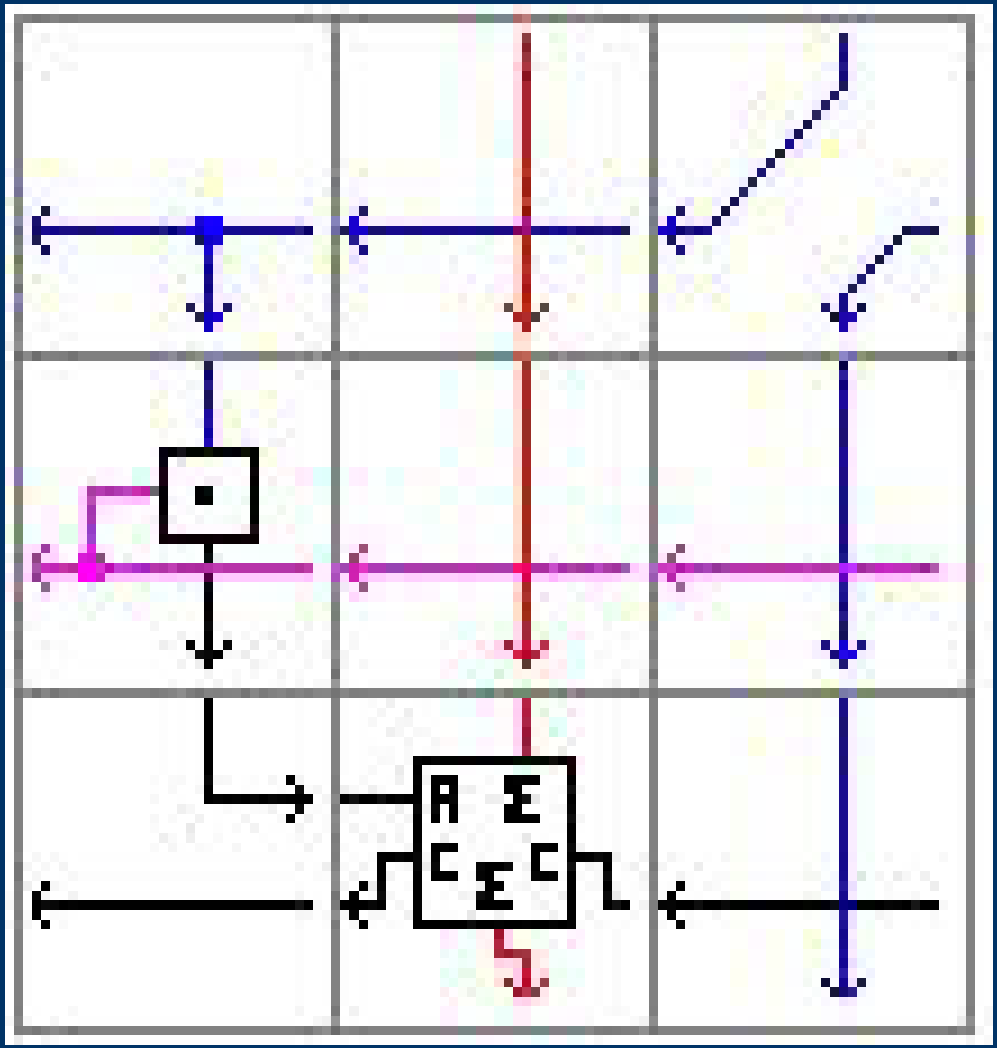
ONE-BIT ADDER



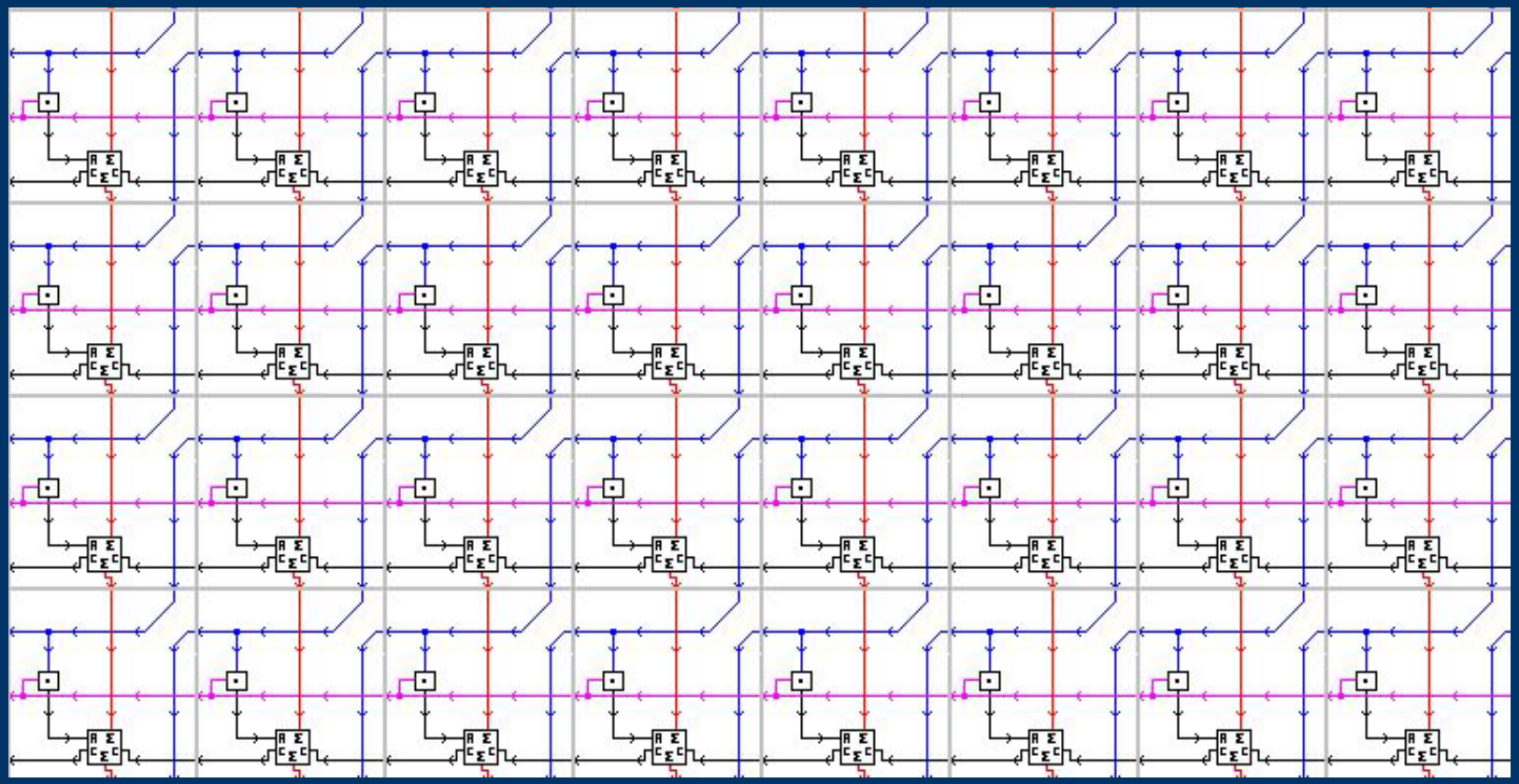
16-BIT ADDER



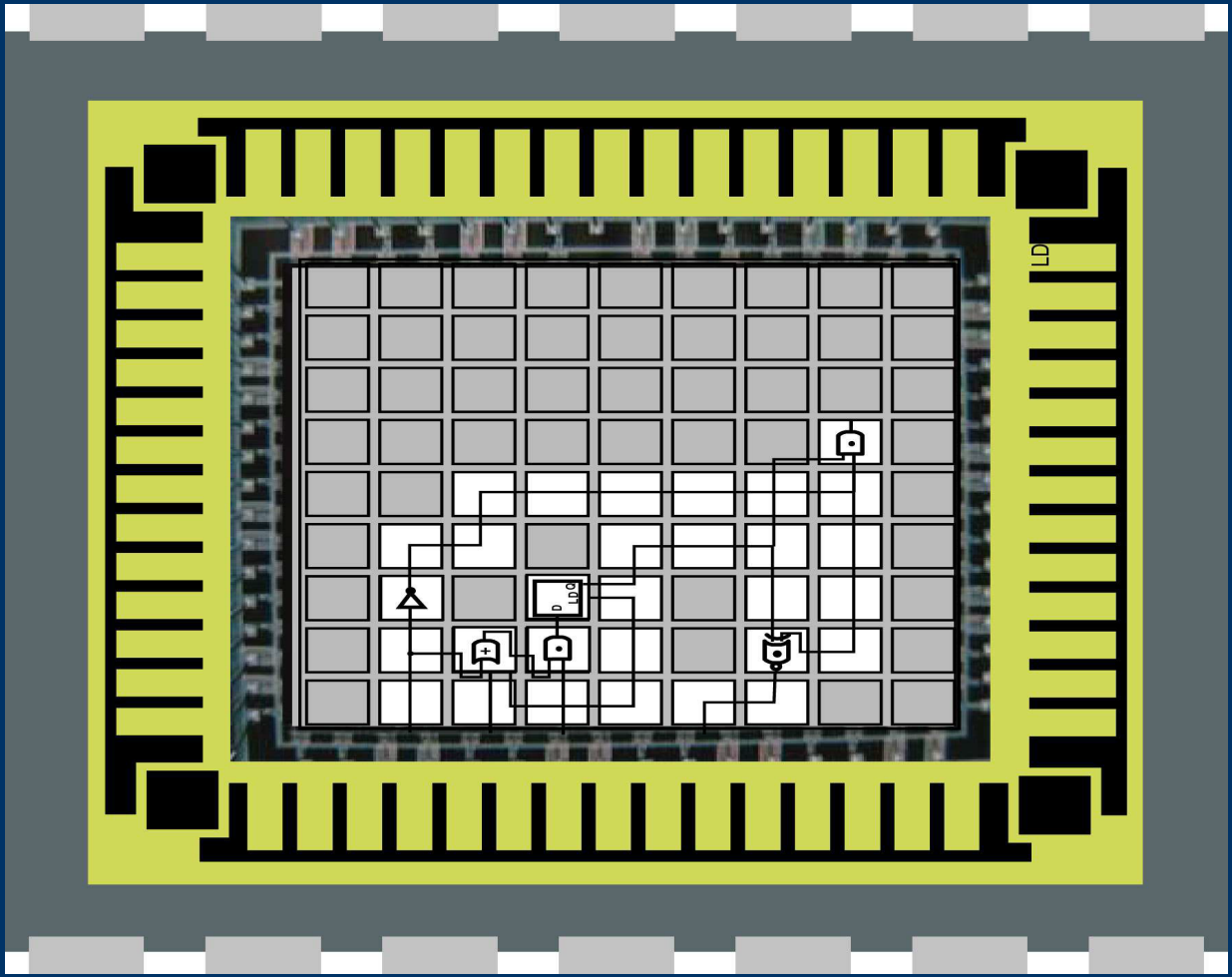
One-Bit Multiplier



4-BIT COMBINATORIAL MULTIPLIER



CAN MAP ARBITRARY CIRCUITS



DNA SEQUENCE ALIGNMENT

```
GATCTACCTACGGATCAT
GATCTACGTACGGATCAT
ATCTACTACGGATCATTC
```

DNA SEQUENCE ALIGNMENT

```
GATCTACCTACGGATCAT
GATCTACGTACGGATCAT
 ATCTACTACGGATCATTC
```

DNA SEQUENCE ALIGNMENT

```
GATCTACCTACGGATCAT
GATCTACGTACGGATCAT
 ATCTAC TACGGATCATTC
```

Sequence Alignment Algorithm

Algorithm *Similarity*

input: sequences s and t

output: similarity between s and t

$m \leftarrow |s|$

$n \leftarrow |t|$

for $i \leftarrow 0$ **to** m **do**

  $a[i, 0] \leftarrow i \times g$

for $j \leftarrow 0$ **to** n **do**

  $a[0, j] \leftarrow j \times g$

for $i \leftarrow 1$ **to** m **do**

  for $j \leftarrow 1$ **to** n **do**

    $a[i, j] \leftarrow \max(a[i-1, j] + g,$
    $a[i-1, j-1] + p(i, j),$
    $a[i, j-1] + g)$

return $a[m, n]$

FIGURE 3.2

Basic dynamic programming algorithm for comparison of two sequences.

Source: Introduction to Computational Molecular Biology
J. Setubal and J. Meidanis

Sequence Alignment Flow

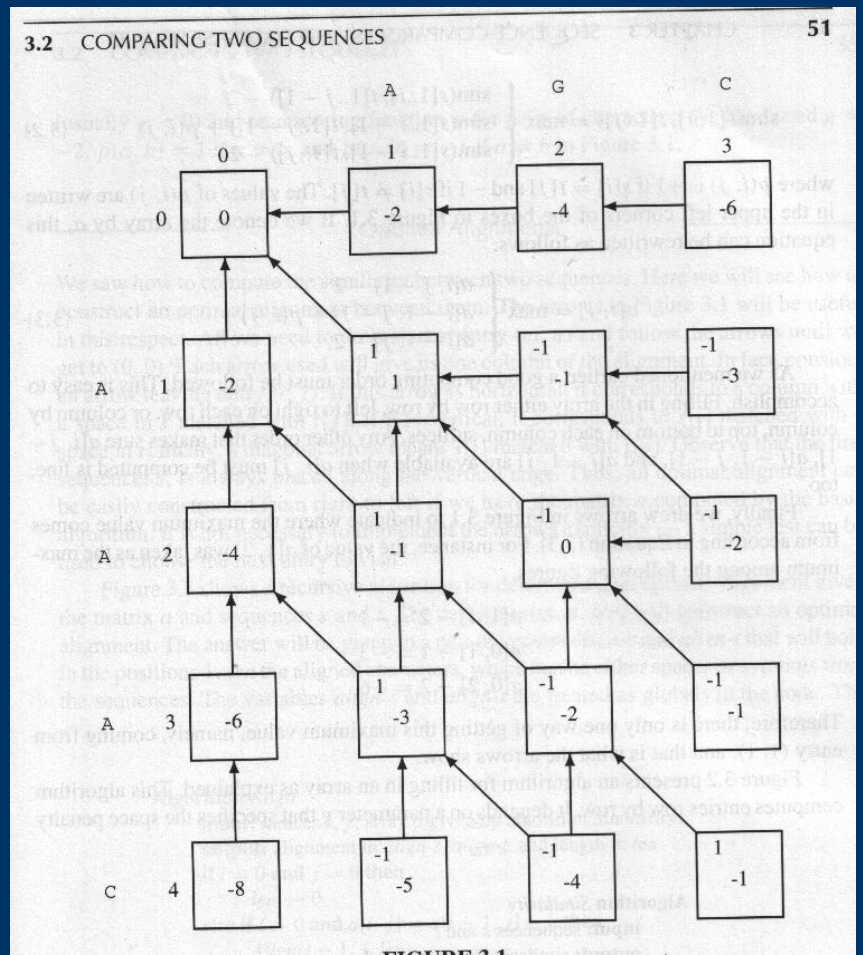
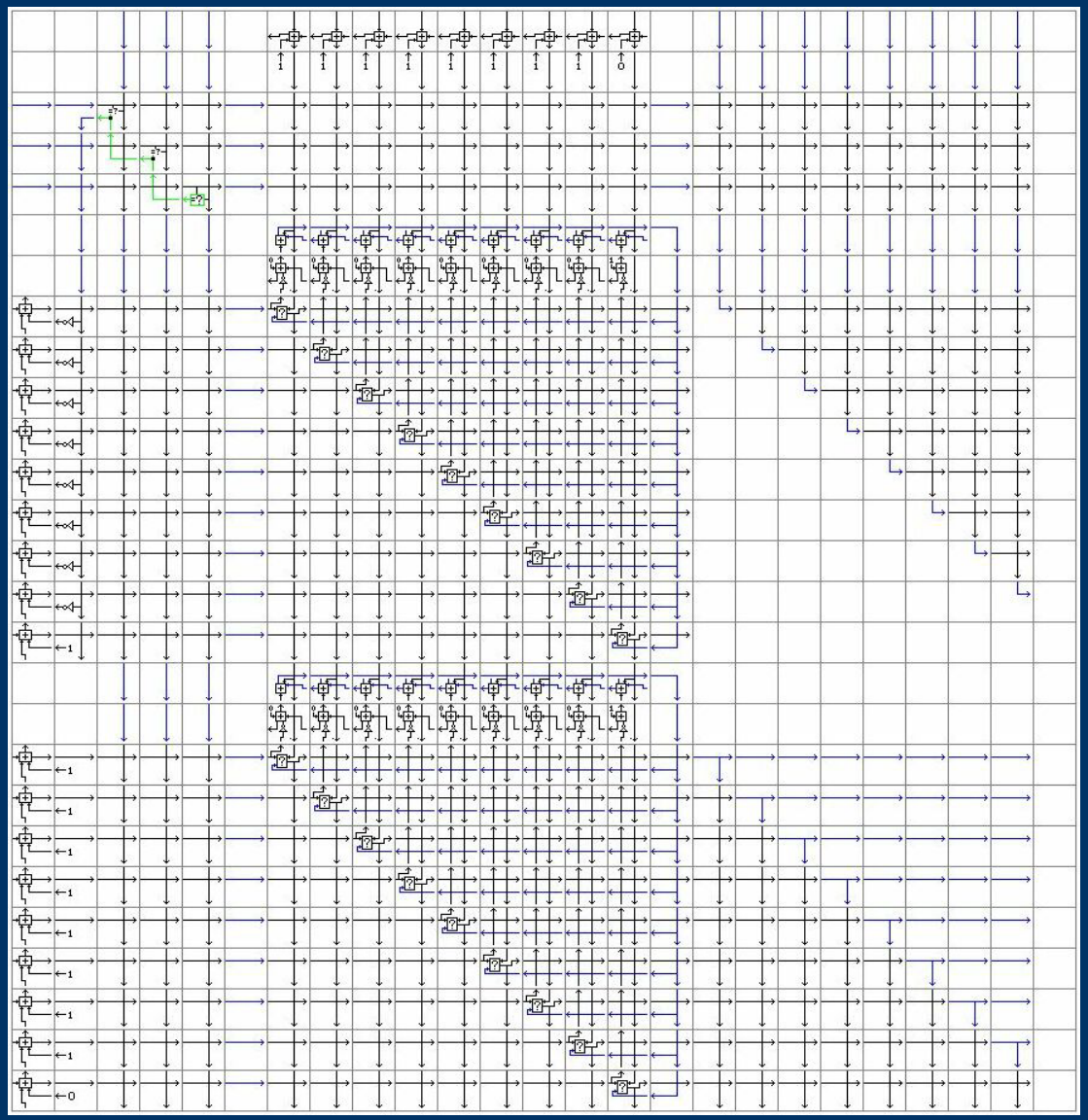


FIGURE 3.1

Bidimensional array for computing optimal alignments. The value in the upper left corner of cell (i, j) indicates whether $s[i] = t[j]$. Indices of rows and columns start at zero.

Source: Introduction to Computational Molecular Biology
J. Setubal and J. Meidanis

Sequence Alignment Circuit



Source: Bin Wang's
Master's Thesis
Utah State University
2001

WHAT'S GOING ON HERE?

- Trading Space for Time
- One-Machine One-Problem

ZETTA-TECHNOLOGY

- Nanotechnology seems to be focused on the *small* (nanometers)

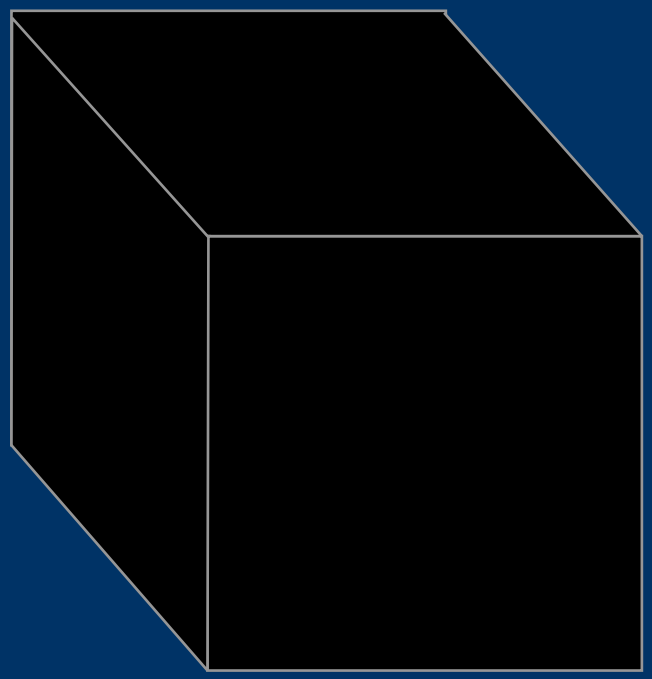
ZETTA-TECHNOLOGY

- Nanotechnology seems to be focused on the *small* (nanometers)
- We're interested in the *numerous* (Zetta=10^{21})

ZETTA-TECHNOLOGY

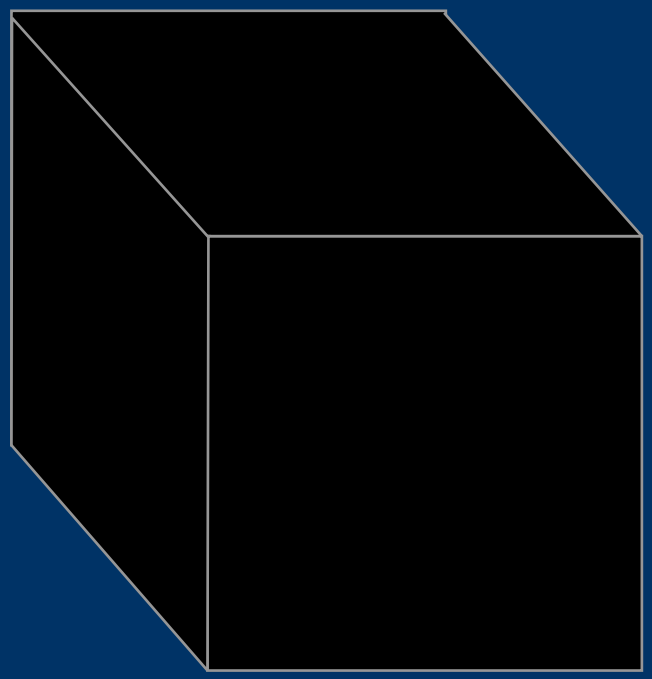
- Nanotechnology seems to be focused on the *small* (nanometers)
- We're interested in the *numerous* (Zetta=10^{21})
- **So you need to make things small, and then make a lot of them!**

WE WANT A LARGE CELL MATRIX!



Pure Carbon
1 Meter per Side
10^{29} Carbon Atoms
100 Atoms/Switch
1000 Switches/Cell

WE WANT A LARGE CELL MATRIX!



Pure Carbon
1 Meter per Side
10^{29} Carbon Atoms
100 Atoms/Switch
1000 Switches/Cell

Avogadro Scale
(10^{24} processors)

THE DEVIL IS IN THE DETAILS!

- **Bootstrap time**
  10^{24} elements *
  1 picosec/element=
  1,000,000,000,000 seconds

THE DEVIL IS IN THE DETAILS!

- **Bootstrap time**
  10^{24} elements *
  1 picosec/element=
  1,000,000,000,000 seconds=
  31,709 YEARS

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- **Faults**

  Assume one billion devices have MTBF=1 year

  Then for one trillion trillion devices

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- **Faults**

  Assume one billion devices have MTBF=1 year

  Then for one trillion trillion devices

MTBF=32 nanoseconds

31 Million Failures Per Second

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- Faults
- **General runtime management**
  monitoring and responding to numnerous events within the system is a huge task...

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- Faults
- General runtime management
- **Manufacturing**

  Moore's law: 50 years

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- Faults
- General runtime management
- **Manufacturing**

  Moore's law: 50 years

  Moore's 2nd law: **$25 TRILLION!**

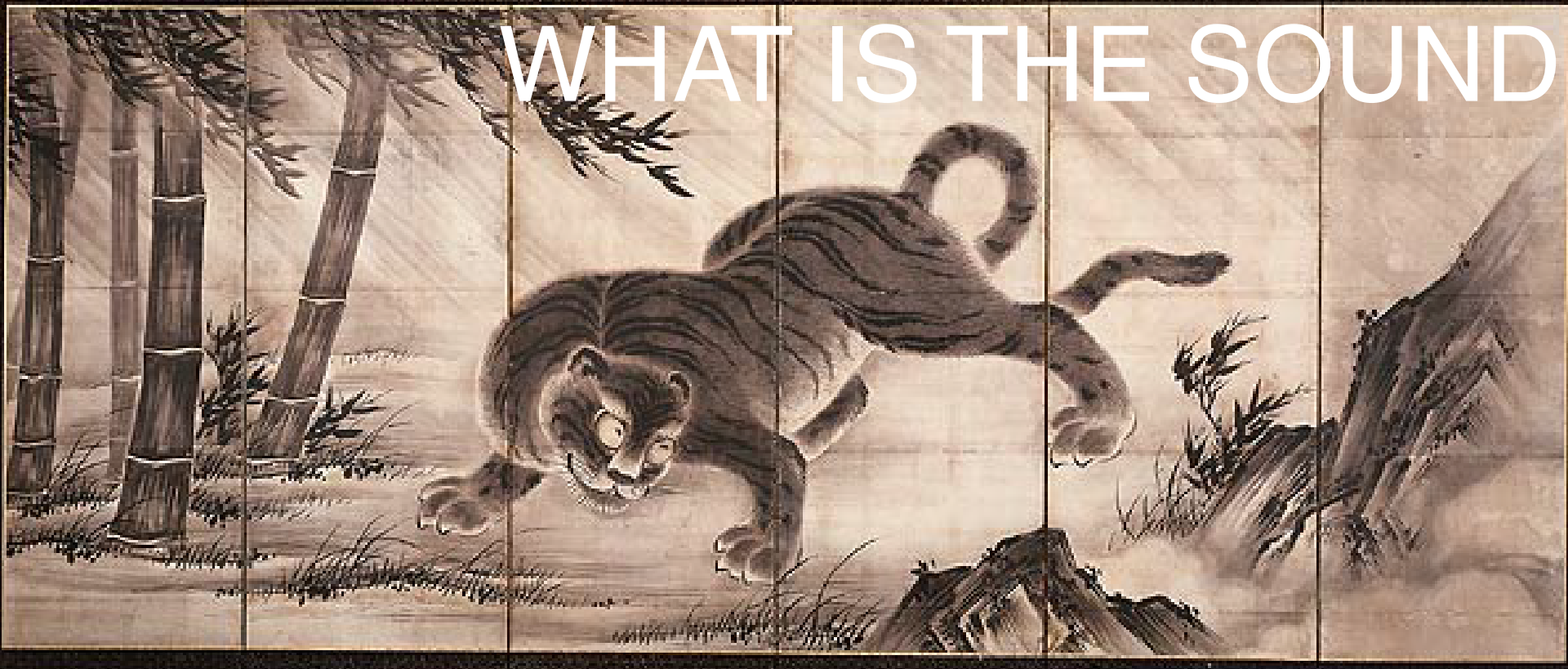
  (GDP US+EU)

THE DEVIL IS IN THE DETAILS!

- Bootstrap time
- Faults
- General runtime management
- Manufacturing
- **Propagation Delays**

1 picosec --> 31,709 years (again)

II. SOME PHILOSOPHY


WHAT IS THE SOUND


OF ONE HAND CLAPPING?


If a tree falls
in the woods
and there's no
one to hear it,
does it make a sound?


If you make a cola and
there's no one to
drink it, does it still
taste sweet?

COCA-COLA SYRUP

®THE COCA-COLA COMPANY

Prepared with sugar and water, phosphoric acid, caffeine, extractives from coca leaves (cocaine removed) and cola nuts, and other flavoring material, colored with caramel. **For Relief of Simple Nausea or Vomiting.**

DOSE One or two teaspoonfuls over crushed ice and sip slowly, or as directed by physician. Repeat hourly if needed. **Keep all medicines out of reach of children.**

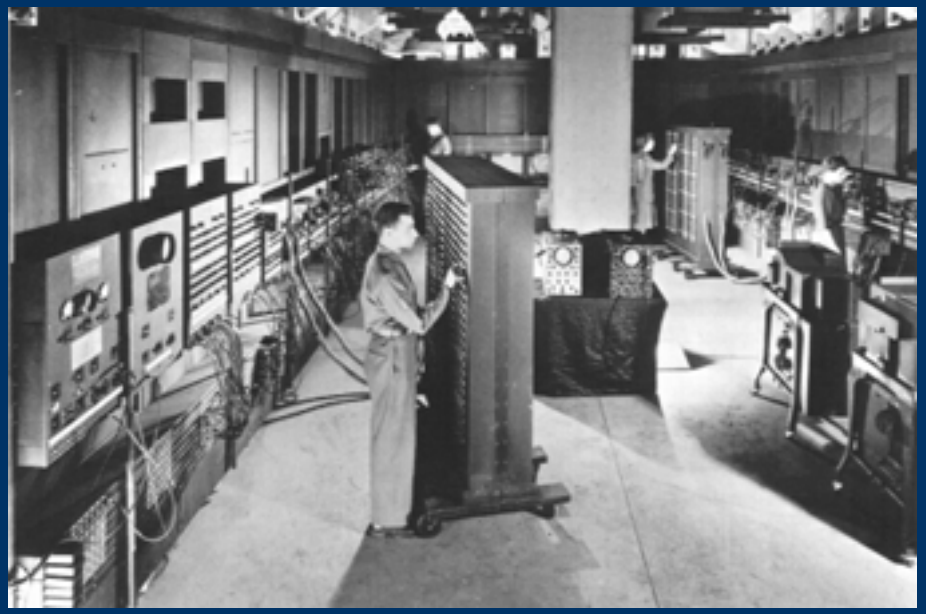
CAUTION: If condition continues after a few doses consult your physician.

NET FLD OZS PACKED BY

APPLICATION TO COMPUTER DESIGN

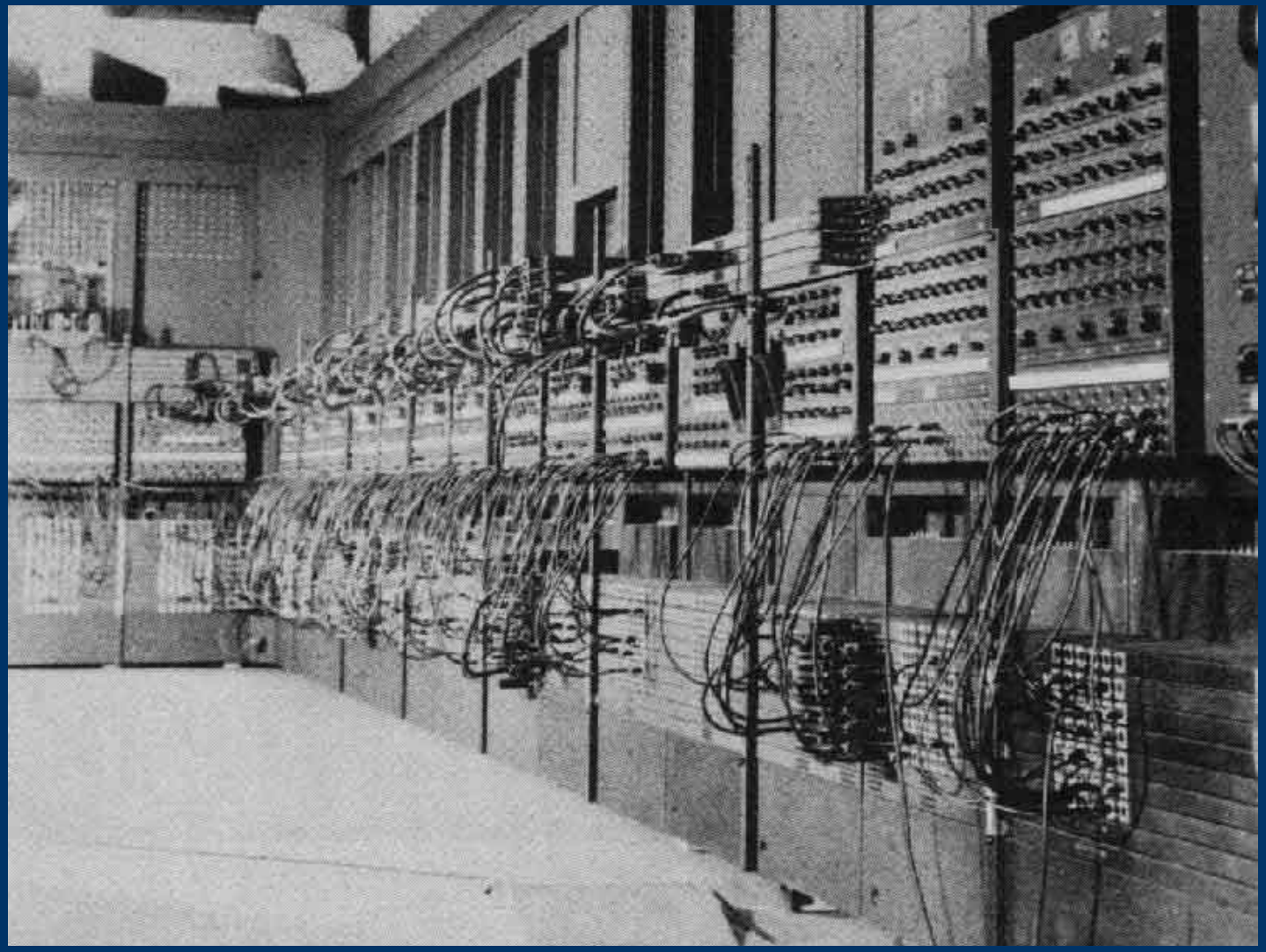
EDVAC COMPUTER

APPLICATION TO COMPUTER DESIGN



ENIAC COMPUTER

APPLICATION TO COMPUTER DESIGN



EDSAC Initial Orders 2 (1949)

```
- code section 0-1 initializes the accumulator and branches to 20,
- later these locations are overwritten and used as data

  0  (T    F) overwritten - (clears accumulator) later used as address
  1  (E 20 F) overwritten - ((always) branch to 20) later used for
                           character input

- code section 2-3 are fixed constants

  2   P  1 F  - constant 1 in address field
  3   U  2 F  - constant for use in creating subroutine returns

 - code section from 4-12 forms a loop that reads in digits from the
 - tape and assembles them into an address

  4   A 39 F              - add constant in 39 into accumulator
  5   R  4 F              - shift accumulator right four places
  6   V    F              - multiply and add into accumulator
  7   L  8 F  - constant  - shift accumulator left five places
  8   T    F              - store partial address in location 0
  9   I  1 F              - input next character into location 1
 10   A  1 F              - load character from location 1
 11   S 39 F              - subtract constant in 39 from accumulator
 12   G  4 F              - branch to 4 if character is a digit

 - code section 13-18 tests a character and either places an A (24+x) F
 - or an E (16+x) F order in location 20

 13   L    D              - left shift accumulator one place
 14   S 39 F              - subtract constant in 39 from accumulator
 15   E 17 F              - conditional branch to 17
 16   S  7 F              - subtract constant in 7 from accumulator
 17   A 35 F              - add constant in 35 to accumulator
 18   T 20 F              - store in location 20


 19   A    F              - add address in 0 into accumulator
 20  (H  8 F) overwritten - (place 10/32 in multiplier) will later be used
                            to add to the accumulator or to branch
 21   A 40 F              - add to accumulator
 22  (T 43 F) overwritten - (store into location 43) will later be used
                            to store assembled order into proper location
```

EDSAC Initial Orders 2 (1949)

```
- code section 0-1 initializes the accumulator and branches to 20,
- later these locations are overwritten and used as data

  0  (T    F) overwritten - (clears accumulator) later used as address
  1  (E 20 F) overwritten - ((always) branch to 20) later used for
                           character input

- code section 2-3 are fixed constants

  2   P  1 F  - constant 1 in address field
  3   U  2 F  - constant for use in creating subroutine returns

 - code section from 4-12 forms a loop that reads in digits from the
 - tape and assembles them into an address

  4   A 39 F              - add constant in 39 into accumulator
  5   R  4 F              - shift accumulator right four places
  6   V    F              - multiply and add into accumulator
  7   L  8 F  - constant  - shift accumulator left five places
  8   T    F              - store partial address in location 0
  9   I  1 F              - input next character into location 1
 10   A  1 F              - load character from location 1
 11   S 39 F              - subtract constant in 39 from accumulator
 12   G  4 F              - branch to 4 if character is a digit

 - code section 13-18 tests a character and either places an A (24+x) F
 - or an E (16+x) F order in location 20

 13   L    D              - left shift accumulator one place
 14   S 39 F              - subtract constant in 39 from accumulator
 15   E 17 F              - conditional branch to 17
 16   S  7 F              - subtract constant in 7 from accumulator
 17   A 35 F              - add constant in 35 to accumulator
 18   T 20 F              - store in location 20


 19   A    F              - add address in 0 into accumulator
 20  (H  8 F) overwritten - (place 10/32 in multiplier) will later be used
                            to add to the accumulator or to branch
 21   A 40 F              - add to accumulator
 22  (T 43 F) overwritten - (store into location 43) will later be used
                            to store assembled order into proper location
```

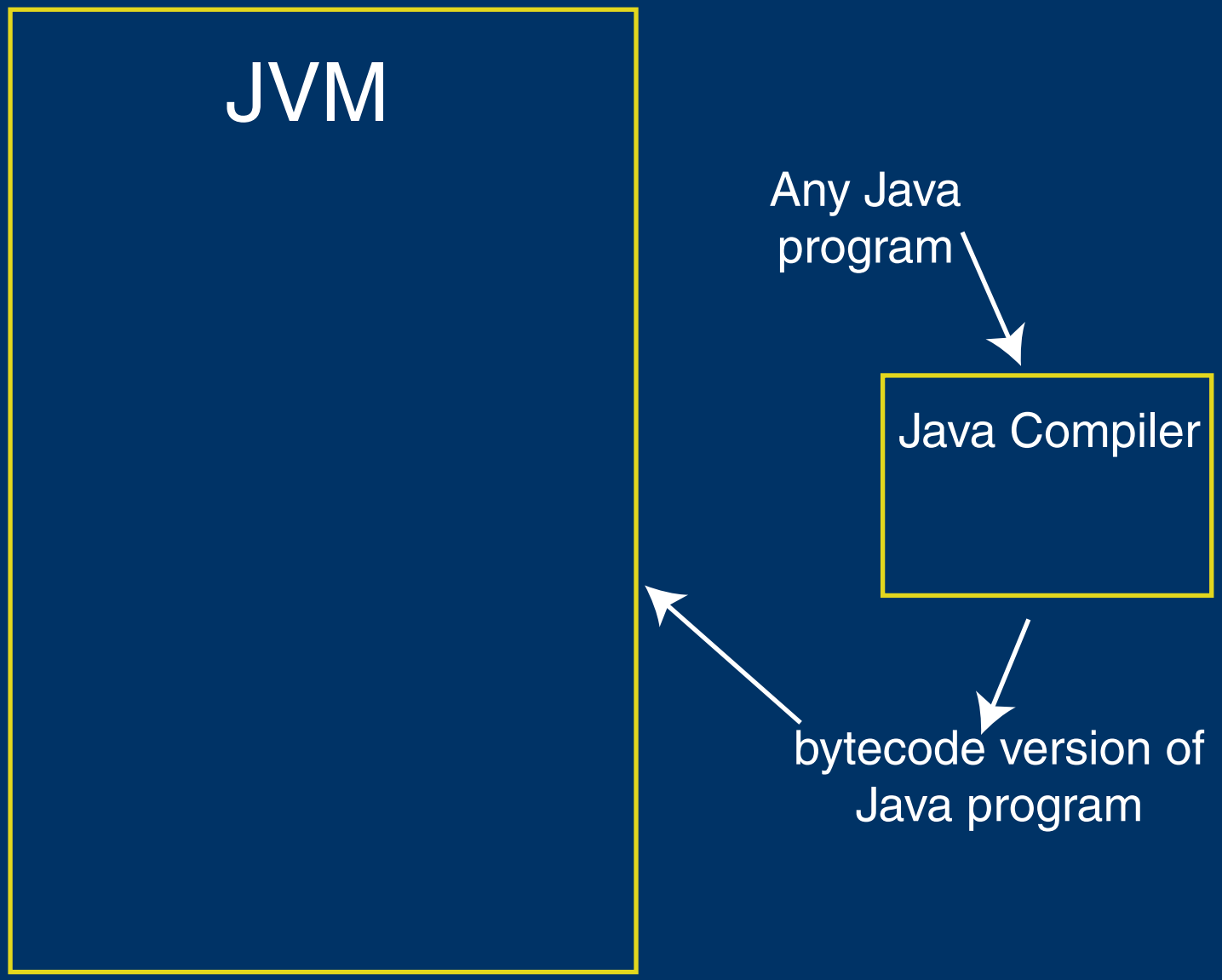
EDSAC Initial Orders 2 (1949)

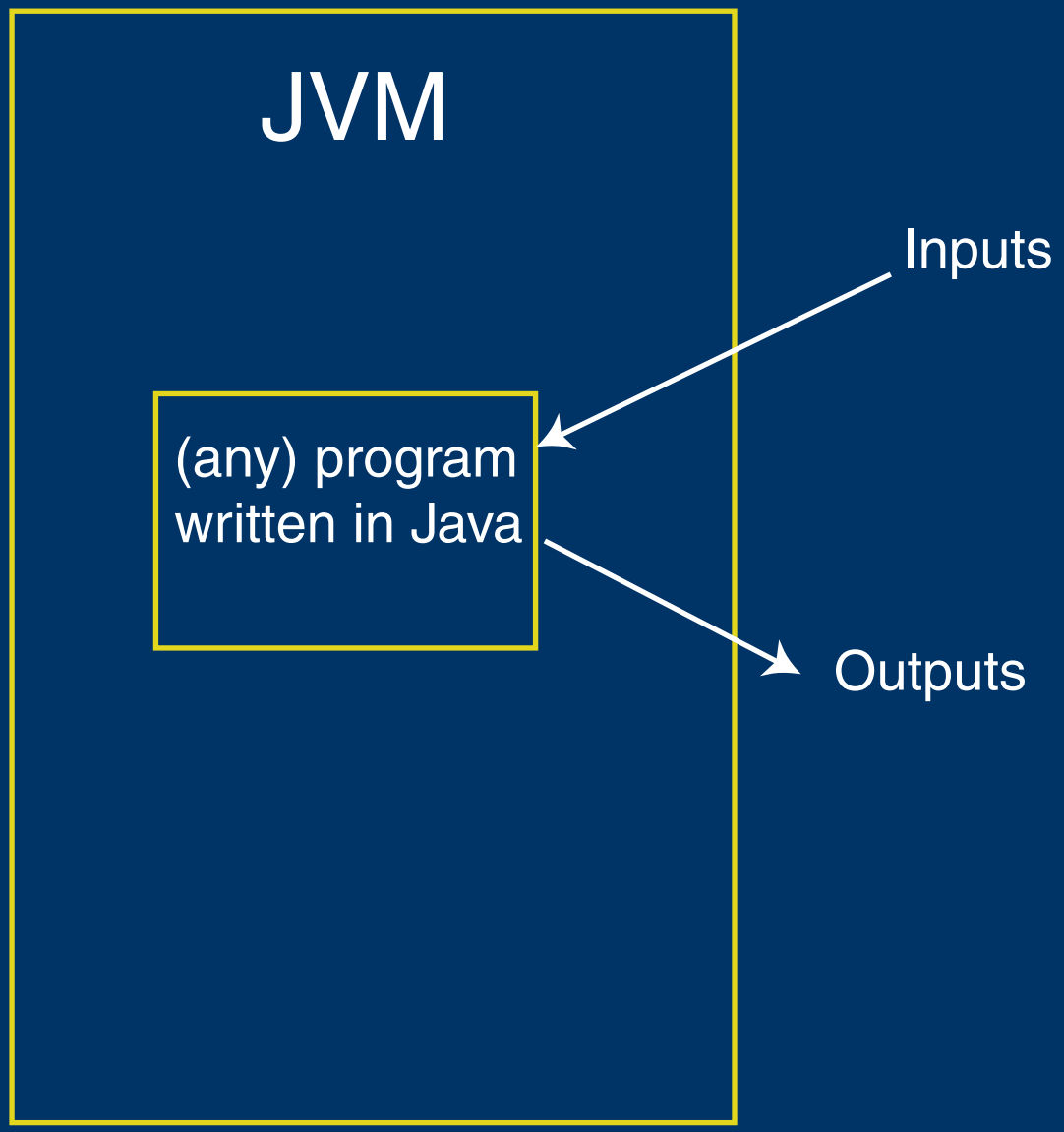
```
- code section 0-1 initializes the accumulator and branches to 20,
- later these locations are overwritten and used as data

  0  (T    F) overwritten - (clears accumulator) later used as address
  1  (E 20 F) overwritten - ((always) branch to 20) later used for
                           character input

- code section 2-3 are fixed constants

  2   P  1 F  - constant 1 in address field
  3   U  2 F  - constant for use in creating subroutine returns

 - code section from 4-12 forms a loop that reads in digits from the
 - tape and assembles them into an address

  4   A 39 F              - add constant in 39 into accumulator
  5   R  4 F              - shift accumulator right four places
  6   V    F              - multiply and add into accumulator
  7   L  8 F  - constant  - shift accumulator left five places
  8   T    F              - store partial address in location 0
  9   I  1 F              - input next character into location 1
 10   A  1 F              - load character from location 1
 11   S 39 F              - subtract constant in 39 from accumulator
 12   G  4 F              - branch to 4 if character is a digit

 - code section 13-18 tests a character and either places an A (24+x) F
 - or an E (16+x) F order in location 20

 13   L    D              - left shift accumulator one place
 14   S 39 F              - subtract constant in 39 from accumulator
 15   E 17 F              - conditional branch to 17
 16   S  7 F              - subtract constant in 7 from accumulator
 17   A 35 F              - add constant in 35 to accumulator
 18   T 20 F              - store in location 20


 19   A    F              - add address in 0 into accumulator
 20  (H  8 F) overwritten - (place 10/32 in multiplier) will later be used
                           to add to the accumulator or to branch
 21   A 40 F              - add to accumulator
 22  (T 43 F) overwritten - (store into location 43) will later be used
                           to store assembled order into proper location
```

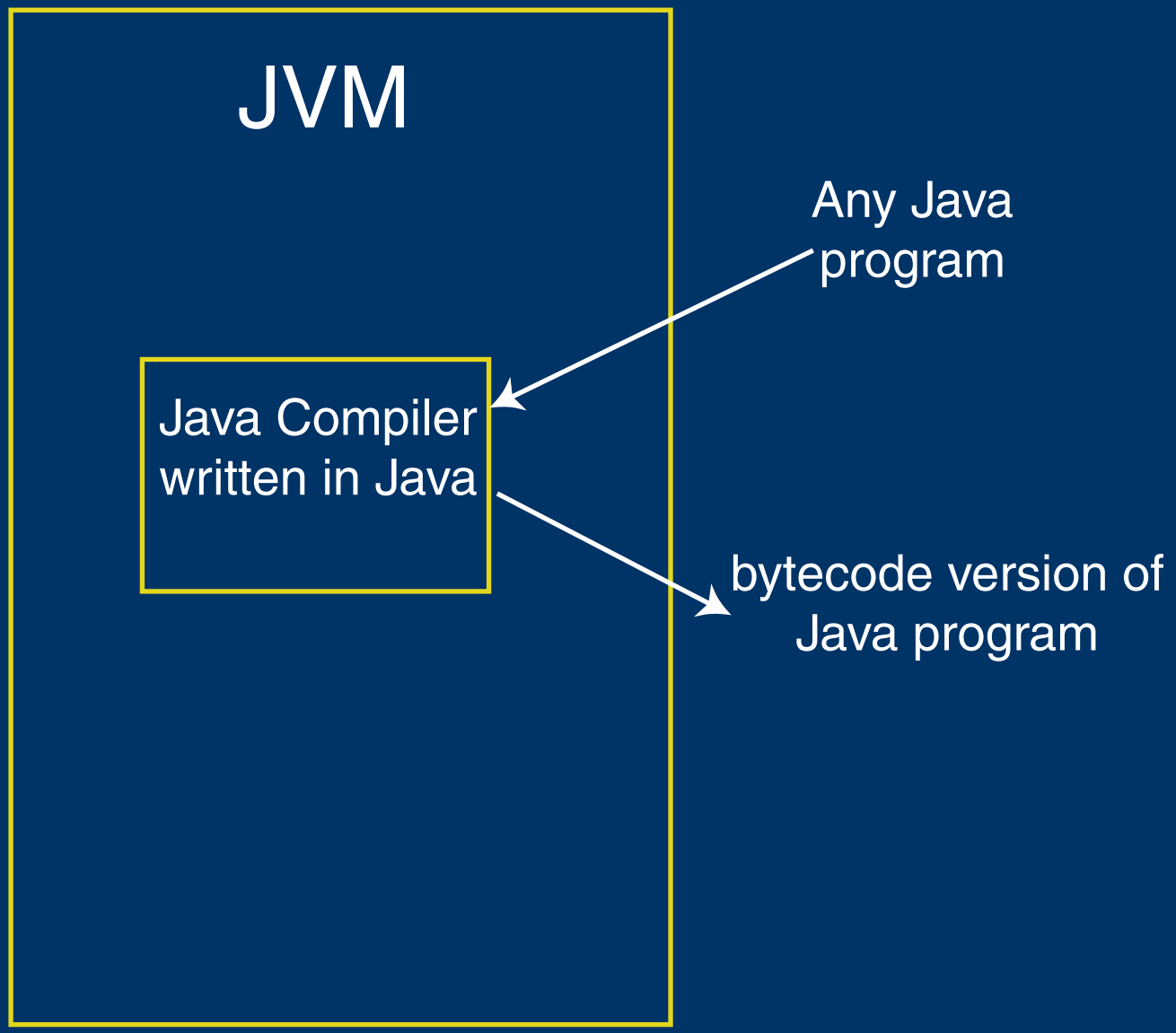
```
/* See if a cell is "near" a failed cell */
  public boolean near_failure(int r,int c)
  {
    if (grid[r][c].fail == 1) return(true);
    if (r>0) {if (grid[r-1][c].fail == 1) return(true);}
    if (c>0) {if (grid[r][c-1].fail == 1) return(true);}
    if (r < ROWS-1) {if (grid[r+1][c].fail == 1) return(true);}
    if (c < COLS-1) {if (grid[r][c+1].fail == 1) return(true);}
    return(false);
  }

  public void change_channel()
  {
    if (channel.equals("a")) channel="b";
    else if (channel.equals("b")) channel="c";
    else channel="a";
  }

  public void set_channel(String c){channel=c;}

  public String get_channel(){return(channel);}

/* Routine to set the channel-build variables for the traced route */
  public void build(int row, int col, String tracestr)
```

00 (0x00) nop 01 (0x01) aconst_null 02 (0x02) iconst_m1 03 (0x03) iconst_0 04 (0x04) iconst_1 05 (0x05) iconst_2 06 (0x06) iconst_3 07 (0x07) iconst_4 08 (0x08) iconst_5 09 (0x09) lconst_0 10 (0x0a) lconst_1 11 (0x0b) fconst_0 12 (0x0c) fconst_1 13 (0x0d) fconst_2 14 (0x0e) dconst_0 15 (0x0f) dconst_1 16 (0x10) bipush 17 (0x11) sipush 18 (0x12) ldc 19 (0x13) ldc_w 20 (0x14) ldc2_w 21 (0x15) iload 22 (0x16) lload 23 (0x17) fload 24 (0x18) dload 25 (0x19) aload 26 (0x1a) iload_0 27 (0x1b) iload_1 28 (0x1c) iload_2 29 (0x1d) iload_3 30 (0x1e) lload_0 31 (0x1f) lload_1 32 (0x20) lload_2 33 (0x21) lload_3 34 (0x22) fload_0 35 (0x23) fload_1 36 (0x24) fload_2 37 (0x25) fload_3 38 (0x26) dload_0 39 (0x27) dload_1 40 (0x28) dload_2 41 (0x29) dload_3 42 (0x2a) aload_0 43 (0x2b) aload_1 44 (0x2c) aload_2 45 (0x2d) aload_3 46 (0x2e) iaload 47 (0x2f) laload 48 (0x30) faload 49 (0x31) daload 50 (0x32) aaload 51 (0x33) baload 52 (0x34) caload 53 (0x35) saload 54 (0x36) istore 55 (0x37) lstore 56 (0x38) fstore 57 (0x39) dstore 58 (0x3a) astore 59 (0x3b) istore_0 60 (0x3c) istore_1 61 (0x3d) istore_2 62 (0x3e) istore_3 63 (0x3f) lstore_0 64 (0x40) lstore_1 65 (0x41) lstore_2 66 (0x42) lstore_3 67 (0x43) fstore_0 68 (0x44) fstore_1 69 (0x45) fstore_2 70 (0x46) fstore_3 71 (0x47) dstore_0 72 (0x48) dstore_1 73 (0x49) dstore_2 74 (0x4a) dstore_3 75 (0x4b) astore_0 76 (0x4c) astore_1 77 (0x4d) astore_2 78 (0x4e) astore_3 79 (0x4f) iastore 80 (0x50) lastore 81 (0x51) fastore 82 (0x52) dastore 83 (0x53) aastore 84 (0x54) bastore 85 (0x55) castore 86 (0x56) sastore 87 (0x57) pop 88 (0x58) pop2 089 (0x59) dup 090 (0x5a) dup_x1 091 (0x5b) dup_x2 092 (0x5c) dup2 093 (0x5d) dup2_x1 094 (0x5e) dup2_x2 095 (0x5f) swap 096 (0x60) iadd 097 (0x61) ladd 098 (0x62) fadd 099 (0x63) dadd 100 (0x64) isub 101 (0x65) lsub 102 (0x66) fsub 103 (0x67) dsub 104 (0x68) imul 105 (0x69) lmul 106 (0x6a) fmul 107 (0x6b) dmul 108 (0x6c) idiv 109 (0x6d) ldiv 110 (0x6e) fdiv 111 (0x6f) ddiv 112 (0x70) irem 113 (0x71) lrem 114 (0x72) frem 115 (0x73) drem 116 (0x74).....ineg 117 (0x75) lneg 118 (0x76) fneg 119 (0x77) dneg 120 (0x78) ishl 121 (0x79) lshl 122 (0x7a) ishr 123 (0x7b) lshr 124 (0x7c) iushr 125 (0x7d) lushr 126 (0x7e) iand 127 (0x7f) land 128 (0x80) ior 129 (0x81) lor 130 (0x82) ixor 131 (0x83) lxor 132 (0x84) iinc 133 (0x85) i2l 134 (0x86) i2f 135 (0x87) i2d 136 (0x88) l2i 137 (0x89) l2f 138 (0x8a) l2d 139 (0x8b) f2i 140 (0x8c) f2l 141 (0x8d) f2d 142 (0x8e) d2i 143 (0x8f) d2l 144 (0x90) d2f 145 (0x91) i2b 146 (0x92) i2c 147 (0x93) i2s 148 (0x94) lcmp 149 (0x95) fcmpl 150 (0x96) fcmpg 151 (0x97) dcmpl 152 (0x98) dcmpg 153 (0x99) ifeq 154 (0x9a) ifne 155 (0x9b) iflt 156 (0x9c) ifge 157 (0x9d) ifgt 158 (0x9e) ifle 159 (0x9f) if_icmpeq 160 (0xa0) if_icmpne 161 (0xa1) if_icmplt 162 (0xa2) if_icmpge 163 (0xa3) if_icmpgt 164 (0xa4) if_icmple 165 (0xa5) if_acmpeq 166 (0xa6) if_acmpne 167 (0xa7) goto 168 (0xa8) jsr 169 (0xa9) ret 170 (0xaa) tableswitch 171 (0xab) lookupswitch 172 (0xac) ireturn 173 (0xad) lreturn 174 (0xae) freturn 175 (0xaf) dreturn 176 (0xb0) areturn 177 (0xb1) return 178 (0xb2) getstatic 179 (0xb3) putstatic 180 (0xb4) getfield 181 (0xb5) putfield 182 (0xb6) invokevirtual 183 (0xb7) invokespecial 184 (0xb8) invokestatic 185 (0xb9) invokeinterface 186 (0xba) xxxunusedxxx1 187 (0xbb) new 188 (0xbc) newarray 189 (0xbd) anewarray 190 (0xbe) arraylength 191 (0xbf) athrow 192 (0xc0) checkcast 193 (0xc1) instanceof 194 (0xc2) monitorenter 195 (0xc3) monitorexit 196 (0xc4) wide 197 (0xc5) multianewarray 198 (0xc6) ifnull 199 (0xc7) ifnonnull 200 (0xc8) goto_w 201 (0xc9) jsr_w Reserved opcodes: 202 (0xca) breakpoint 254 (0xfe) impdep1 255 (0xff) impdep2 00 (0x00) nop

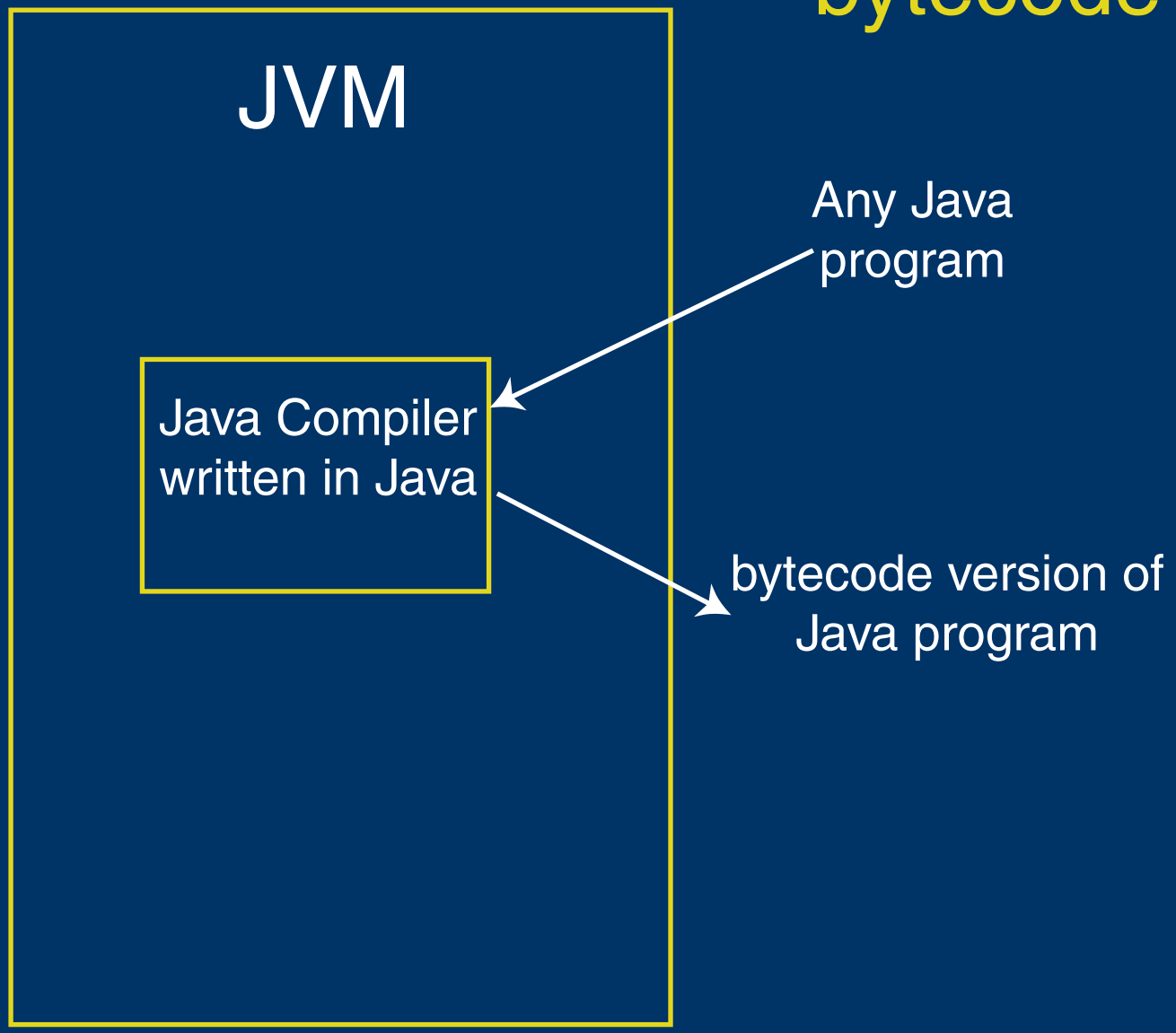
Normal JAVA Program Execution




JVM
Inputs
(any) program
written in Java
Outputs


JVM
Any Java program
Java Compiler written in Java
bytecode version of Java program

How do you execute this new bytecode?



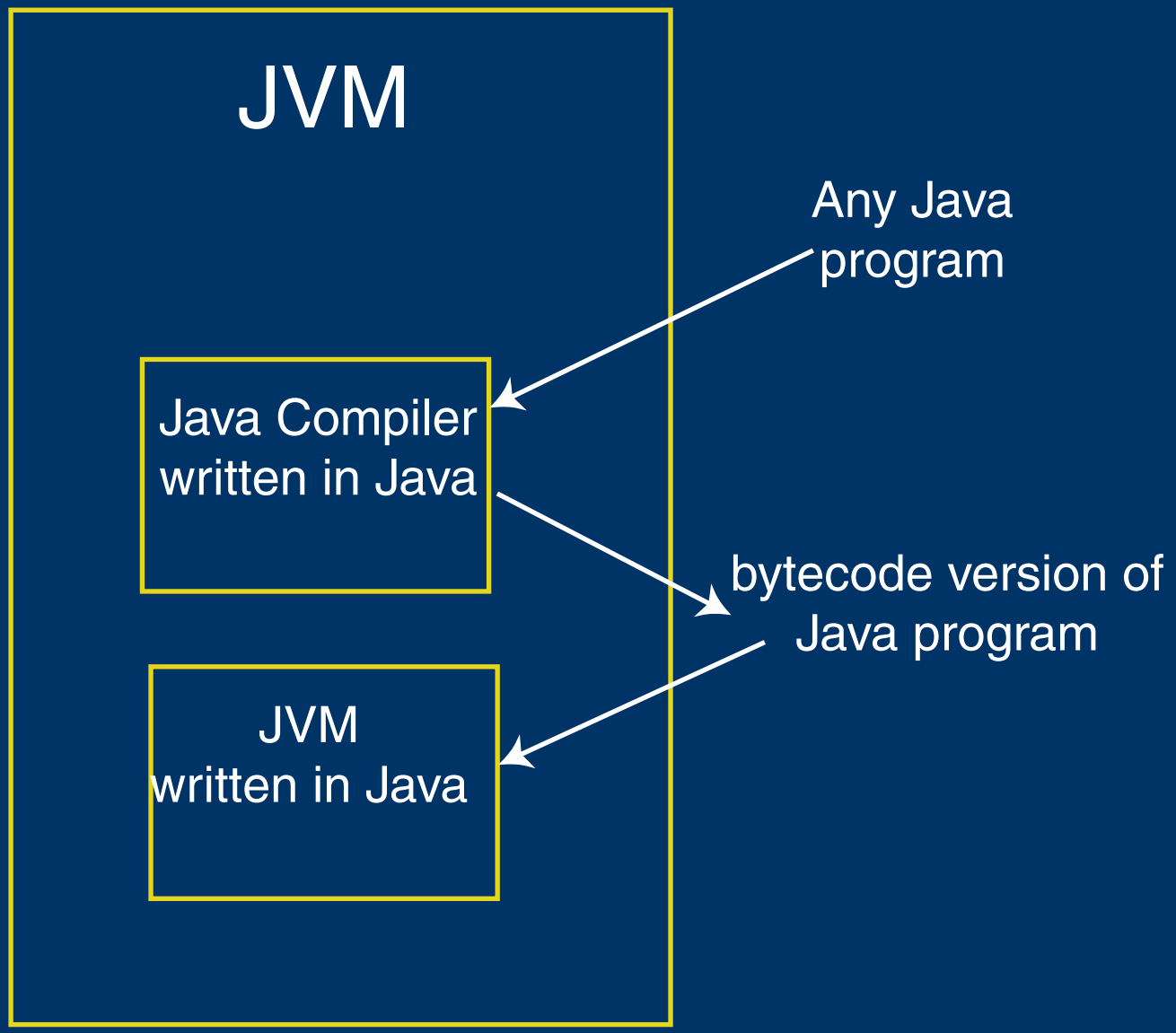
One cannot feed bytecode to the JVM from inside a Java program running on the JVM

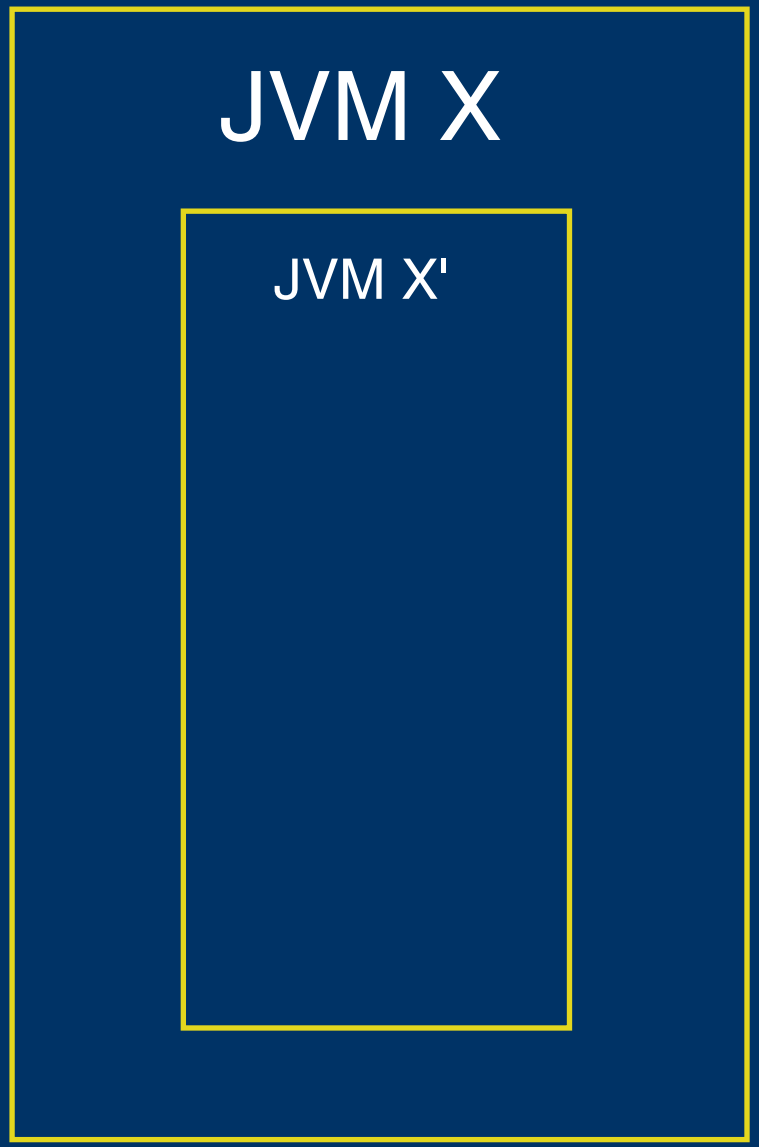
Subject and object ARE different...

One cannot feed bytecode to the JVM from inside a Java program running on the JVM

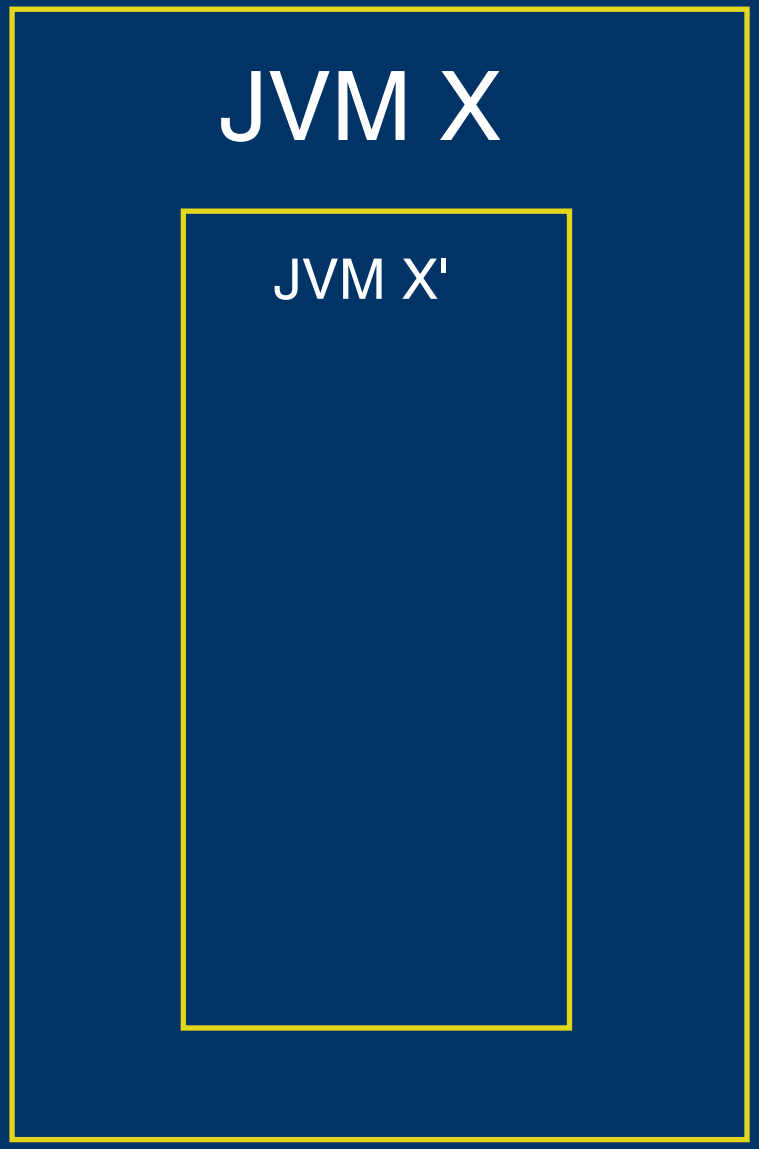
BUT...

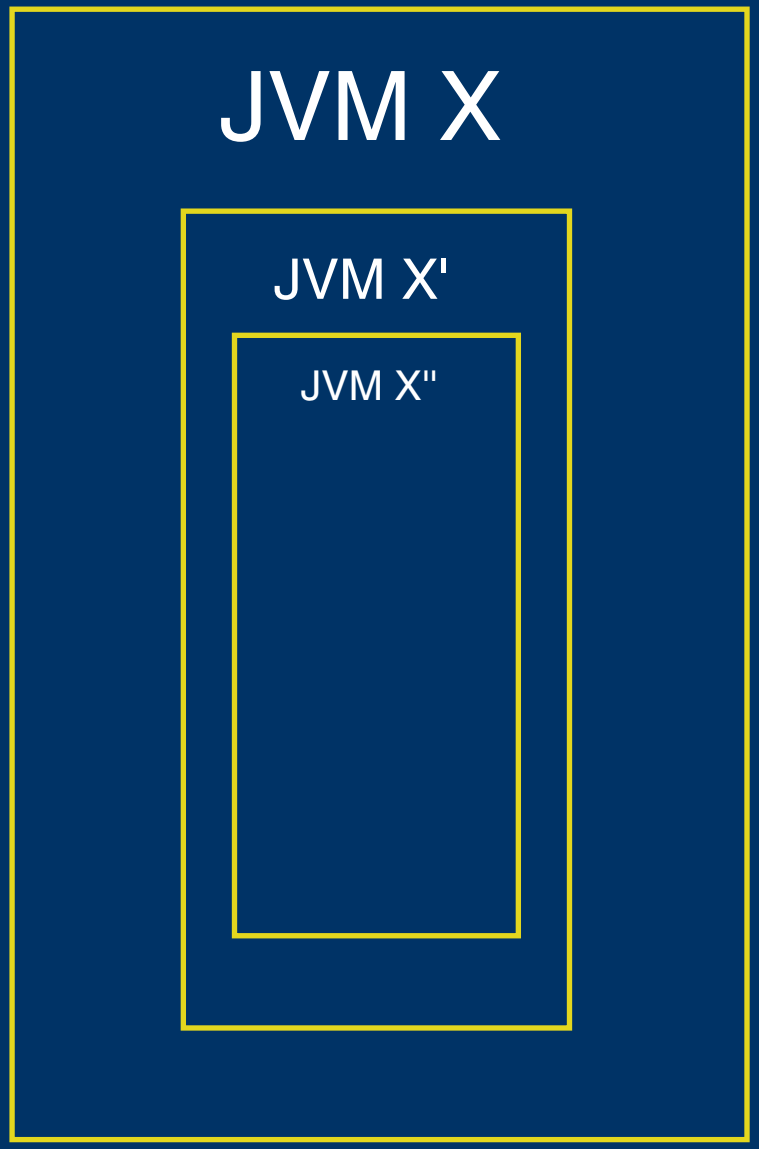
Build a new JVM inside the JVM

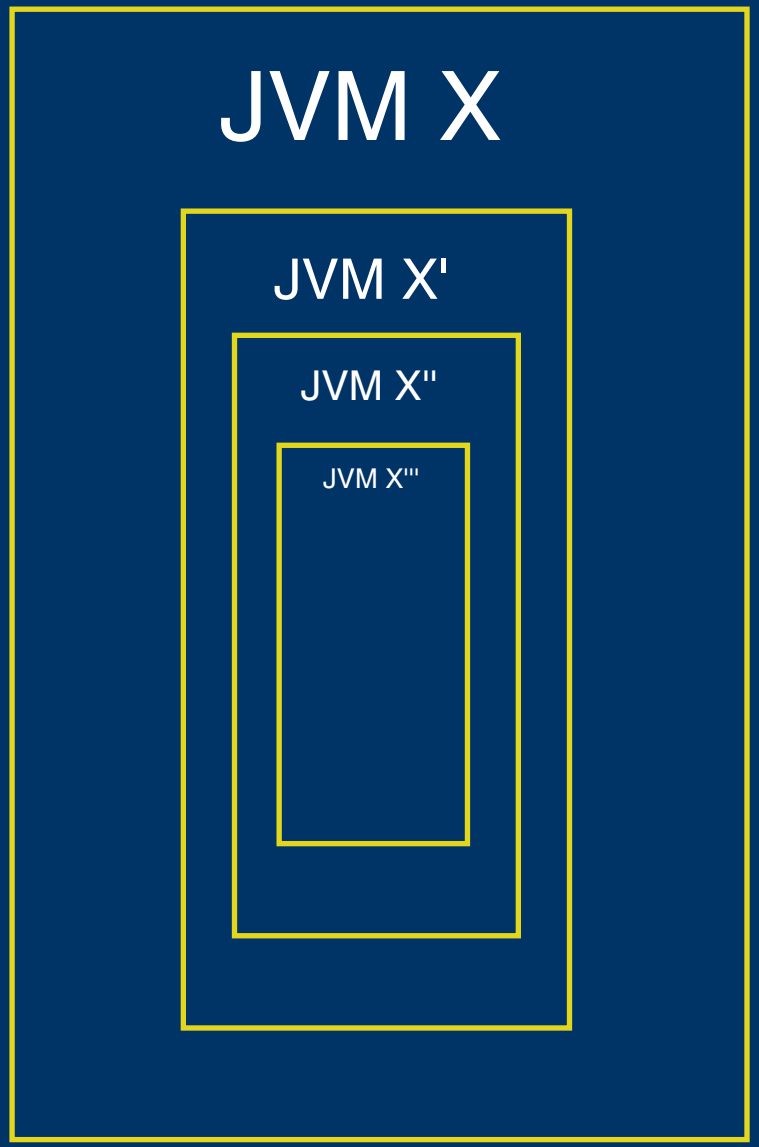



JVM X
JVM X'

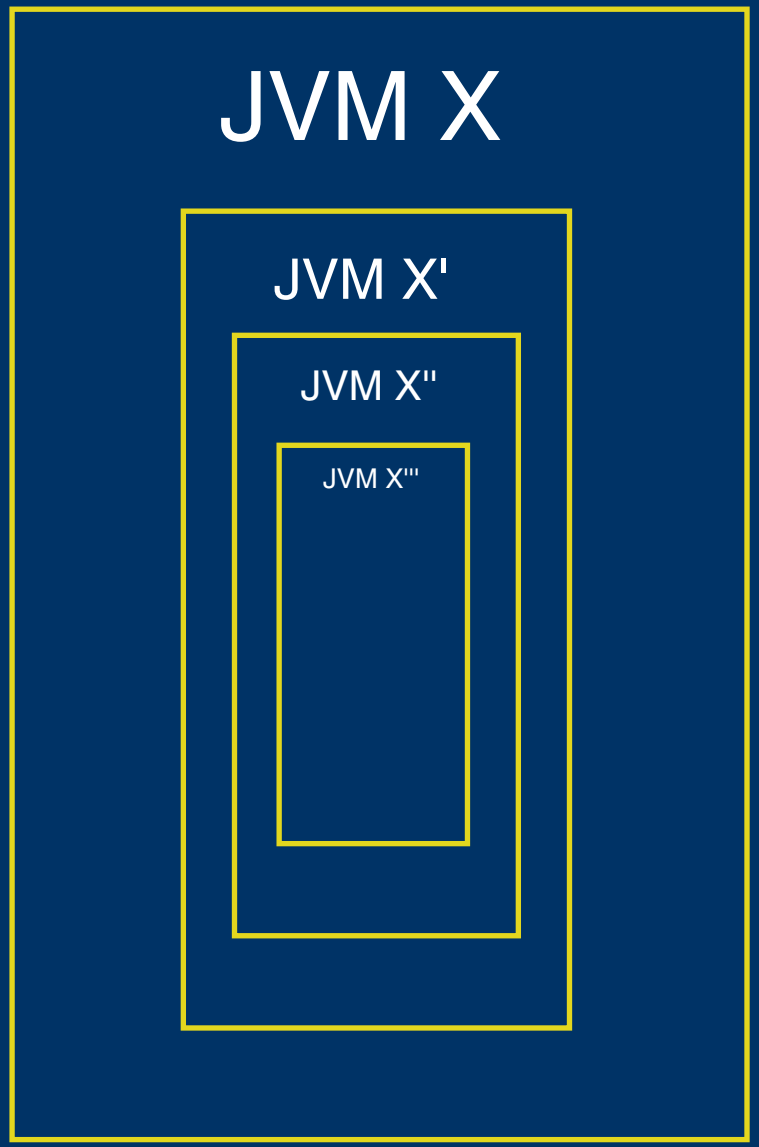
Can bytecode produced by a compiler on X' be run on X' ?




JVM X
JVM X'
JVM X''


JVM X
JVM X'
JVM X''
JVM X'''

Subject and Object are different... (Dualism)



Related to various interesting subjects...

Cantor Diagonalization

Related to various interesting subjects...

Cantor Diagonalization

1 0.**2**3972931232187319239123
2 0.2**9**3995534522398472394 34
3 0.93**3**8739871287321871783 7
4 0.982**1**2389723718728739873
5 0.5638**2**3831312838398432 90

.

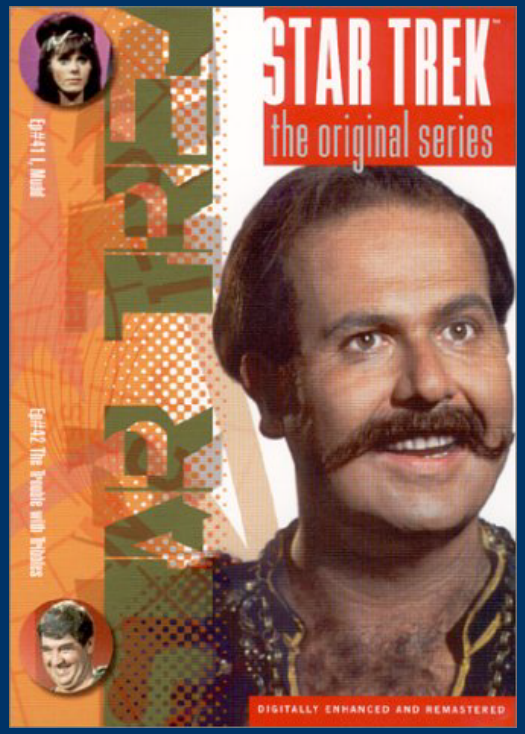
.

.

0.3343334343444343334443 4

Related to various interesting subjects...

Cantor Diagonalization



"I am lying"

The Halting Problem

Make a Turing Machine T that can determine if a Turing Machine T' halts

The Halting Problem

Make a Turing Machine T that can determine if a Turing Machine T' halts

This is impossible...

The Halting Problem

Make a Turing Machine T that can determine if a Turing Machine T' halts

This is impossible...
"Rig the game"

The Halting Problem

Make a Turing Machine T that can determine if a Turing Machine T' halts

Modify T so:

- If T' halts, T enters infinite loop
- If T' doesn't halt, T halts

The Halting Problem

Make a Turing Machine T
that can determine if a
Turing Machine T' halts

Modify T so:

- If T' halts, T enters infinite loop
- If T' doesn't halt, T halts

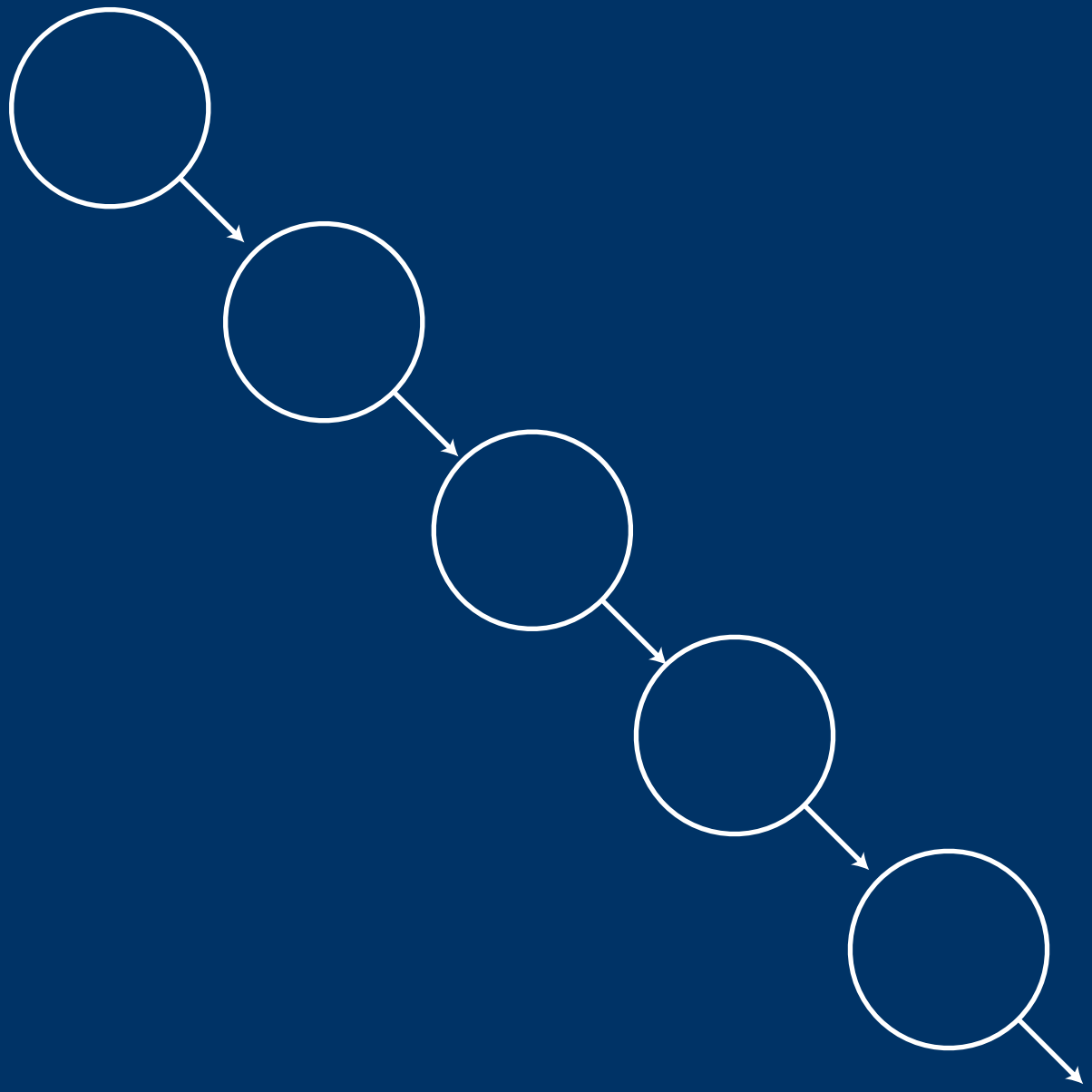
$$T \longrightarrow T$$

Where is the problem?

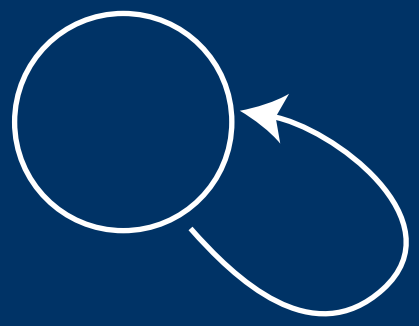
TM can't feed symbols
to its own tape...

Need to emulater a TM
inside the TM

WHAT WE HAVE



WHAT WE WANT



NOT ALWAYS THE CASE...

LISP: string

NOT ALWAYS THE CASE...

LISP: eval(string)

NOT ALWAYS THE CASE...

LISP: eval(string)

APL: string

NOT ALWAYS THE CASE...

LISP: eval(string)

APL: ⍎string

NOT ALWAYS THE CASE...

LISP: eval(string)

APL: ⍎string

Perl: string

NOT ALWAYS THE CASE...

LISP: eval(string)

APL: ⍎string

Perl: `string`

NOT ALWAYS THE CASE...

LISP: eval(string)

APL: ⍎string

Perl: `string`

```
static char p[20][1024]={
"#include <stdio.h>",
"#include <string.h>",
"",
"main()",
"{",
"	int i,j;",
"",
"	printf(\"static char p[20][1024]={\\n\");",
"	for (i=0;i<20;i++){",
"		if (i>0) printf(\"\\\",\\n\");",
"		printf(\"\\\"\");",
"		for (j=0;j<strlen(p[i]);j++){",
"		if ((p[i][j]=='\\\"' || p[i][j]=='\\\\'))
printf(\"\\\\\");",
"		printf(\"%c\",p[i][j]);",
"	  }",
"	}",
"	printf(\"\\\"};\\n\\n\");",
"",
"    for (i=0;i<20;i++) printf(\"%s\\n\",p[i]);",
"}"};
```

```
#include <stdio.h>
#include <string.h>

main()
{
	int i,j;

	printf("static char p[20][1024]={\n");
	for (i=0;i<20;i++){
		if (i>0) printf("\",\n");
		printf("\"");
		for (j=0;j<strlen(p[i]);j++){
		if ((p[i][j]=='\"' || p[i][j]=='\\')) printf("\\");
		printf("%c",p[i][j]);
	  }
	}
	printf("\"};\n\n");

    for (i=0;i<20;i++) printf("%s\n",p[i]);
}
```

```
static char p[20][1024]={
"#include <stdio.h>",
"#include <string.h>",
"",
"main()",
"{",
"       int i,j;",
"",
"       printf(\"static char p[20][1024]={\\n\");",
"       for (i=0;i<20;i++){",
"               if (i>0) printf(\"\\\",\\n\");",
"               printf(\"\\\"\");",
"               for (j=0;j<strlen(p[i]);j++){",
"               if ((p[i][j]=='\\\"' || p[i][j]=='\\\\'))
printf(\"\\\\\");",
"               printf(\"%c\",p[i][j]);",
"         }",
"       }",
"       printf(\"\\\"};\\n\\n\");",
"",
"    for (i=0;i<20;i++) printf(\"%s\\n\",p[i]);",
"}"};
```

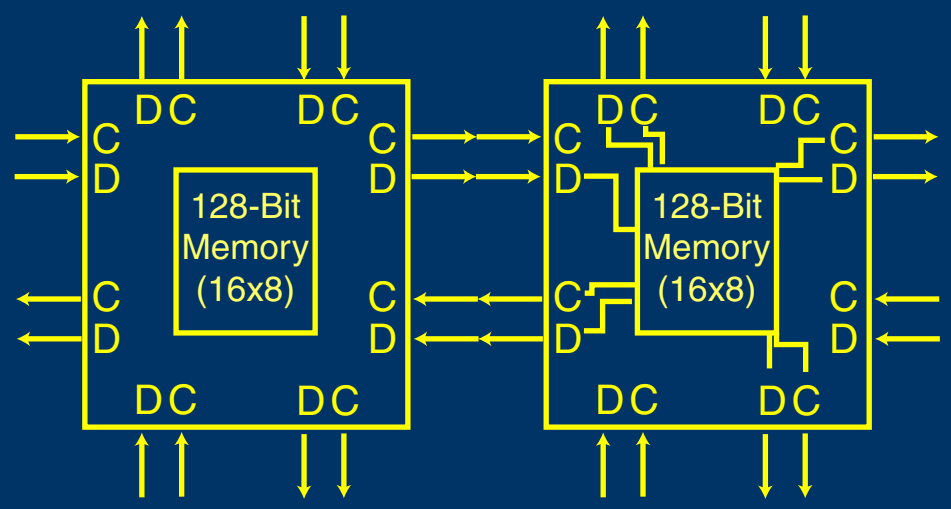
```
#include <stdio.h>
#include <string.h>

main()
{
        int i,j;

        printf("static char p[20][1024]={\n");
        for (i=0;i<20;i++){
                if (i>0) printf("\",\n");
                printf("\"");
                for (j=0;j<strlen(p[i]);j++){
                if ((p[i][j]=='\"' || p[i][j]=='\\')) printf("\\");
                printf("%c",p[i][j]);
          }
        }
        printf("\"};\n\n");

    for (i=0;i<20;i++) printf("%s\n",p[i]);
}
```

III. MORE-EXOTIC CELL MATRIX BEHAVIOR (C-MODE)

How are Cells Configured in a Cell Matrix?

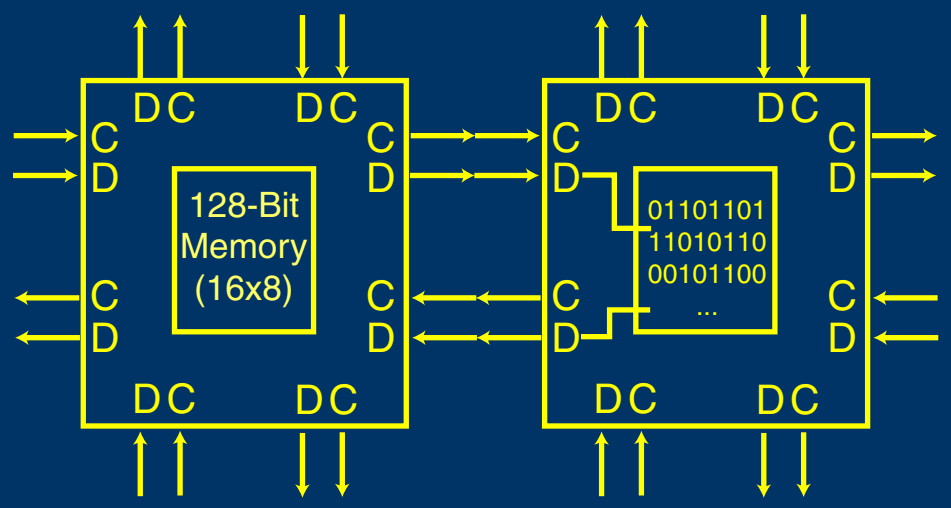
How are Cells Configured in a Cell Matrix?

- Cells typically pass *data* to their neighbors
  - That data is processed via cell's Truth Table (*code*)



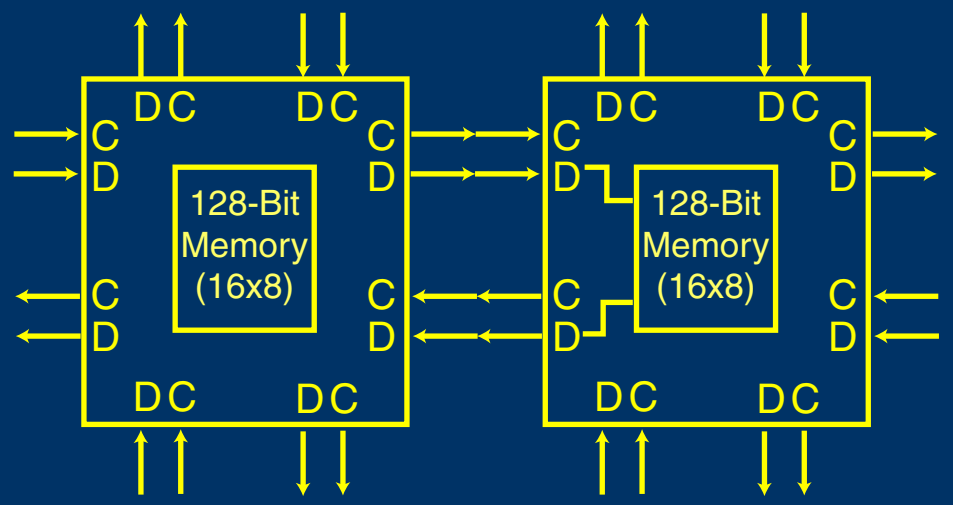
There is a Dual Mode...

- Under certain circumstances, cells may pass *truth tables* (code) to neighbors

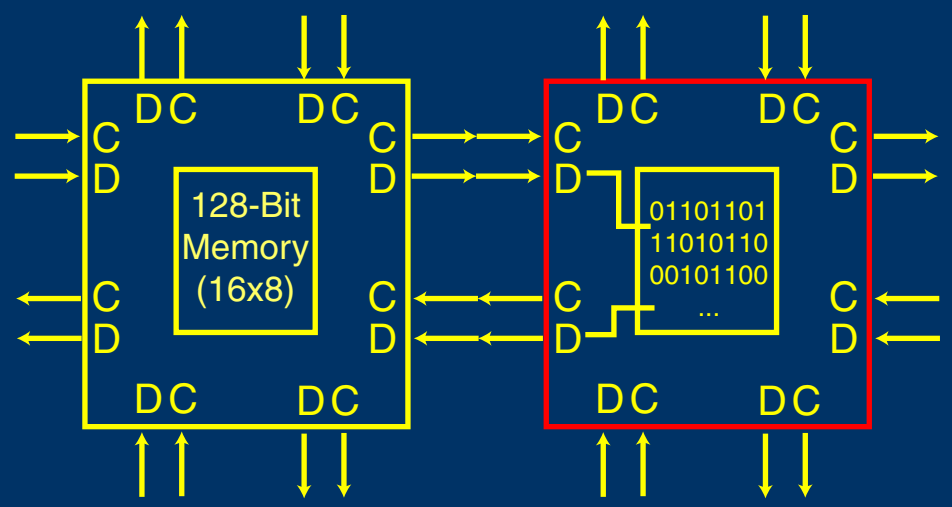


Each Cell Operates in One of Two Modes

D-Mode:
Cells Exchange DATA

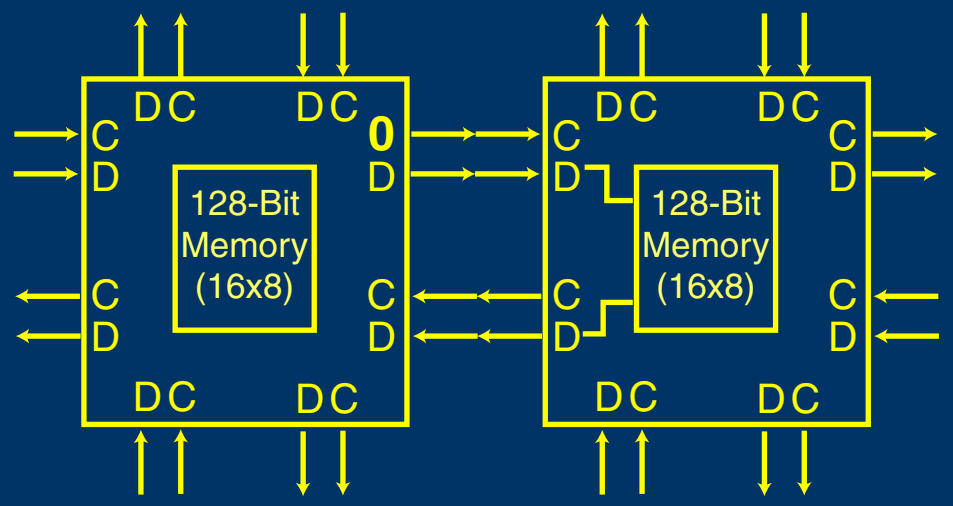


C-Mode:
Cells Exchange CODE

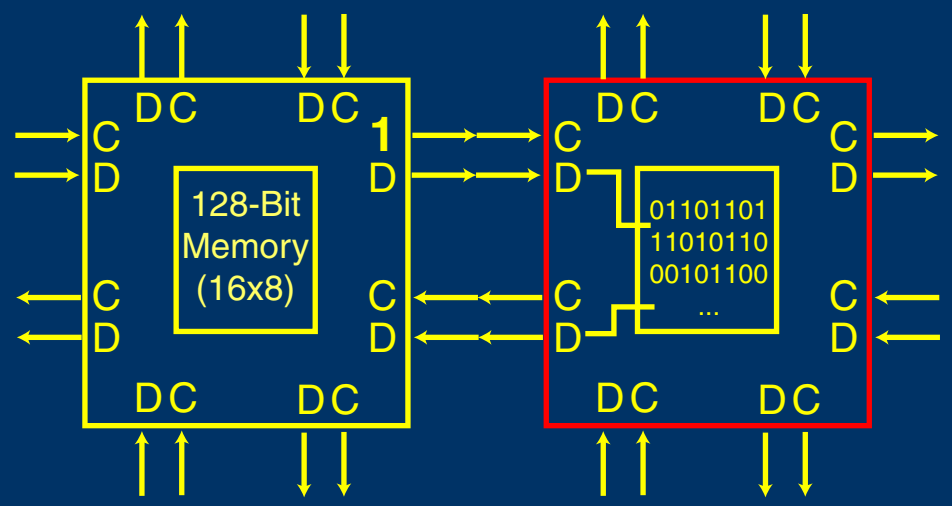


C-Mode is Indicated by Setting a C Input to1

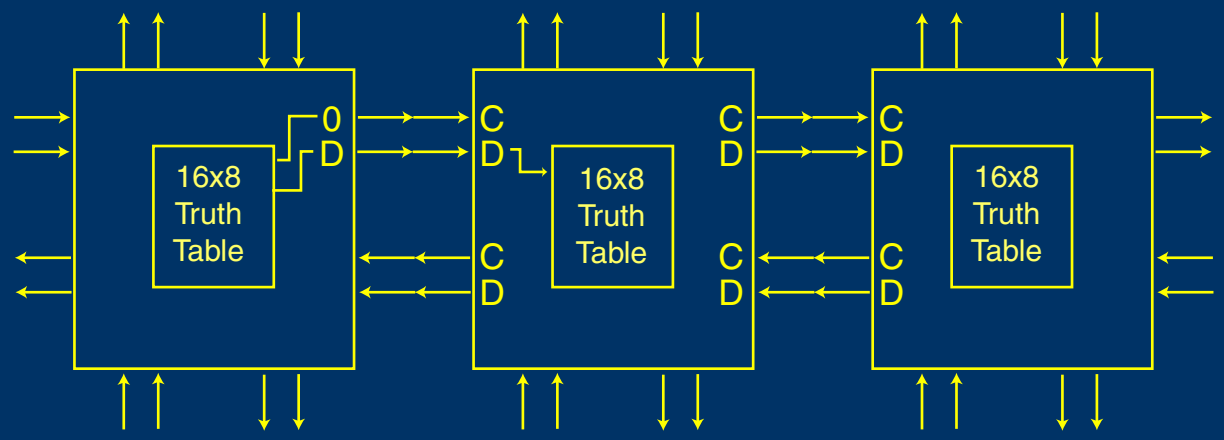
D-Mode:
Cells Exchange DATA



C-Mode:
Cells Exchange CODE

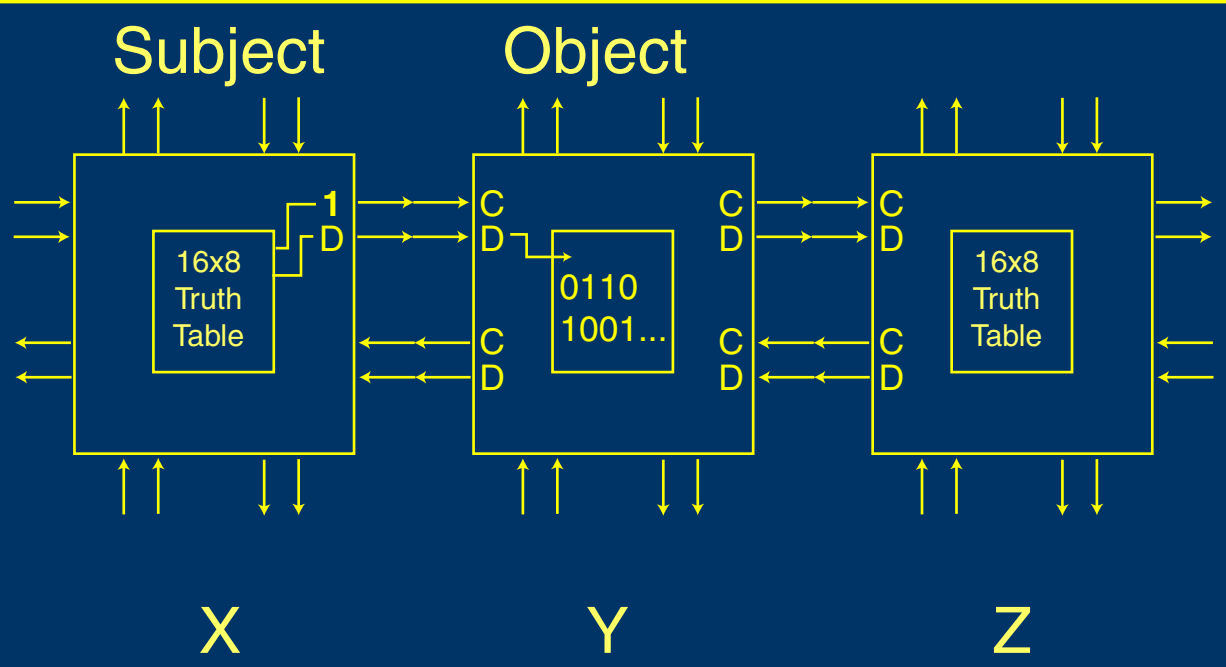


X May Pass Data to Y



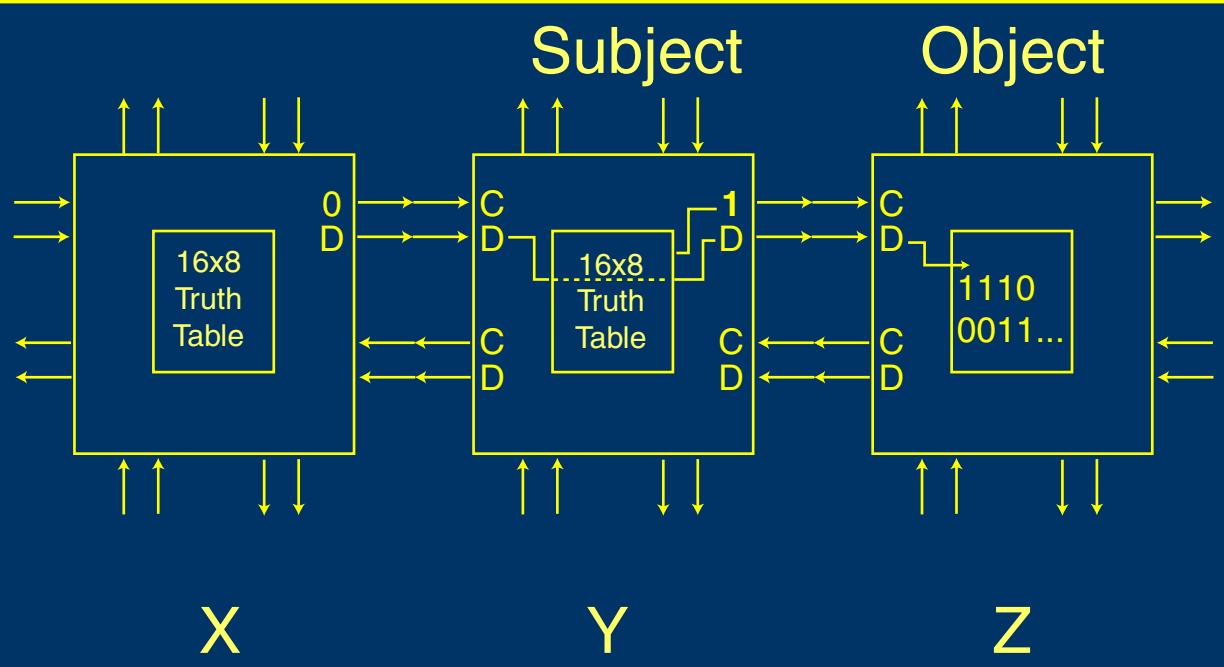
Y is in D Mode

X May Configure Y



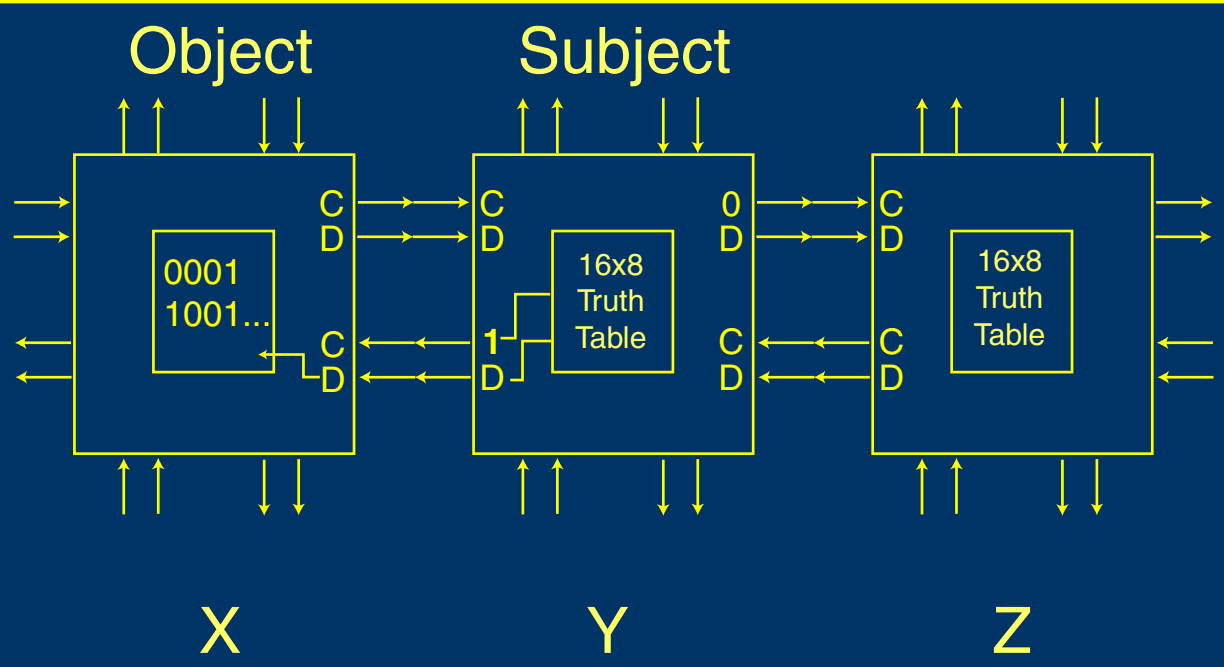
Y is in C Mode

Y May Then Configure Z



Z is in C Mode

Or Y May Reconfigure X



X is in C Mode

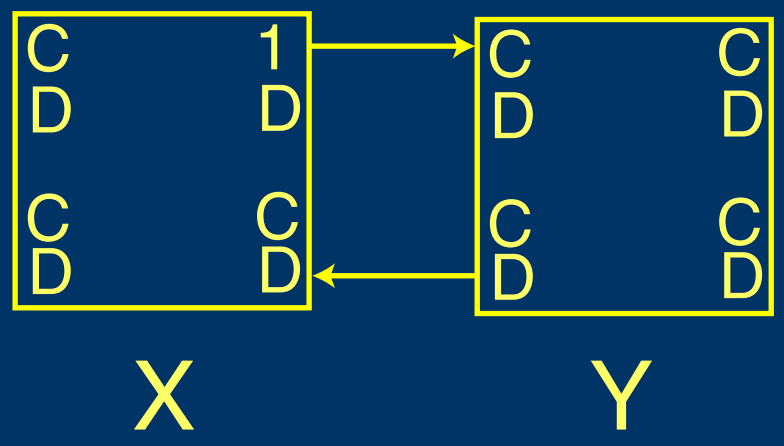
SELF DUALITY

There is a perfect interchangability between the subject and object of configuration

SELF CONFIGURABILITY

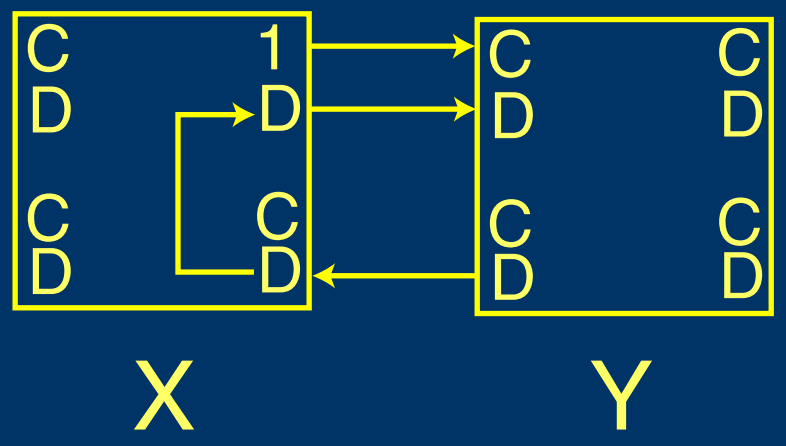
The Cell Matrix is able to read, modify and write its own configuration information.

Cell Reader



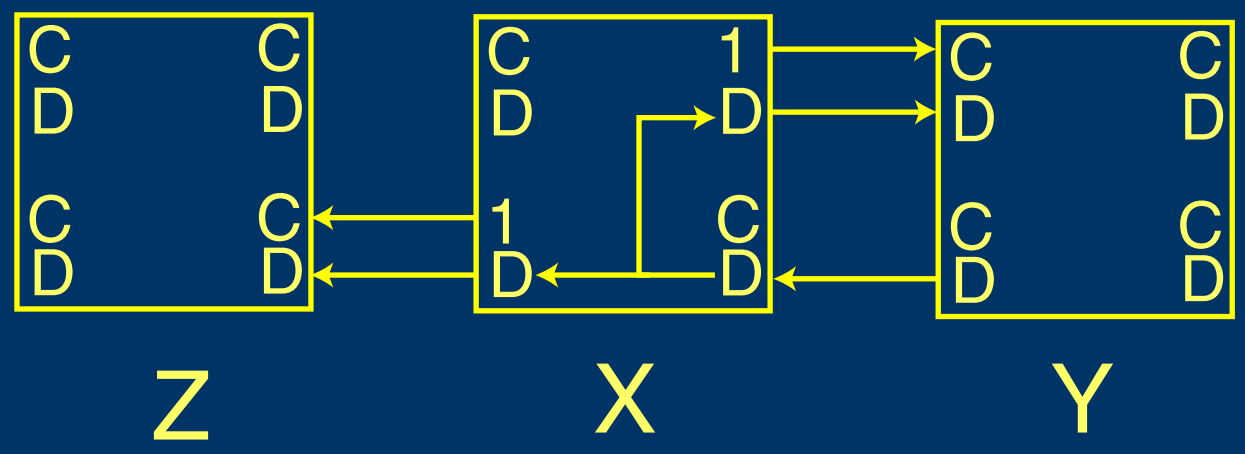
X is reading Y's Truth Table

Non-Destructive Cell Reader



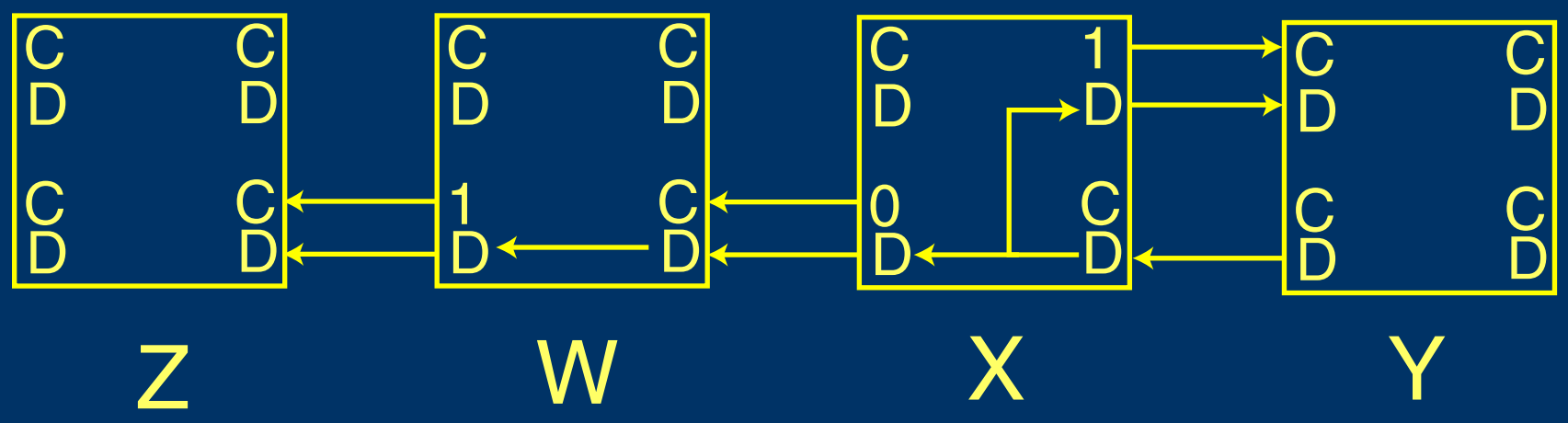
X is reading Y's Truth Table
and copying it back to Y

Cell Replicator



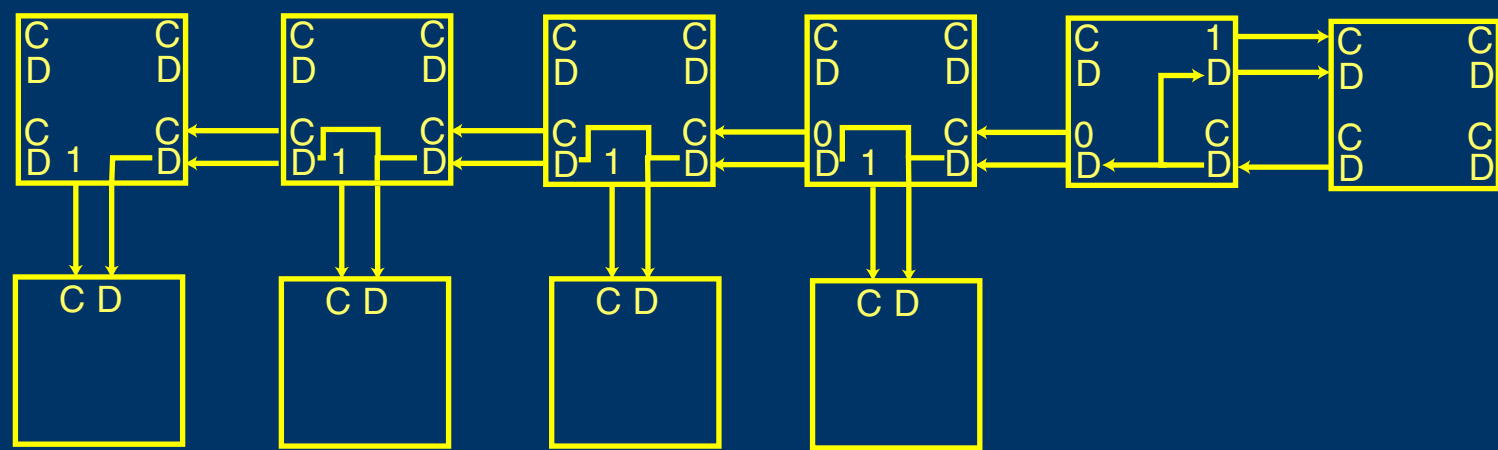
X is reading Y's Truth Table,
copying it back to Y,
and also copying it to Z

Remote Cell Replicator



X is reading Y's Truth Table,
copying it back to Y,
and also copying it to Z

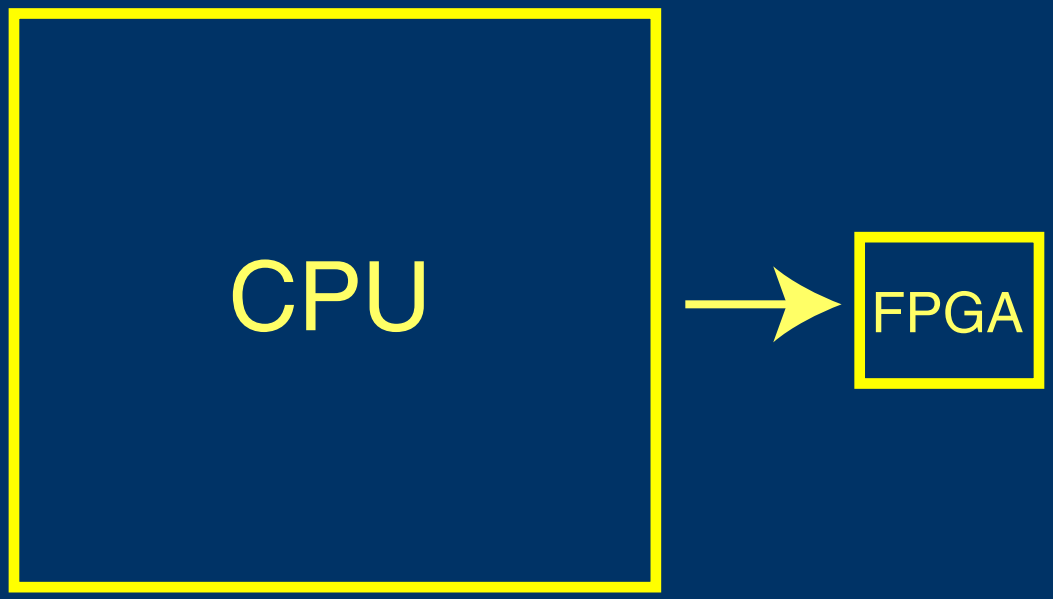
Medusa Circuit



FOUR CELLS CONFIGURED SIMULTANEOUSLY

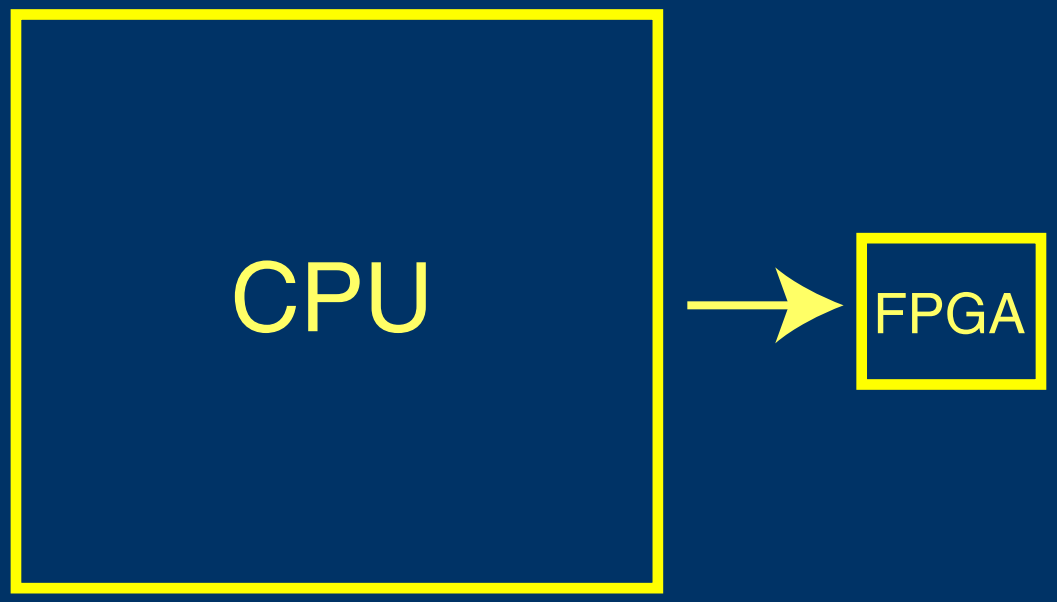
TYPICAL FPGA CONFIGURATION

TYPICAL FPGA CONFIGURATION



External device controls configuration

TYPICAL FPGA CONFIGURATION



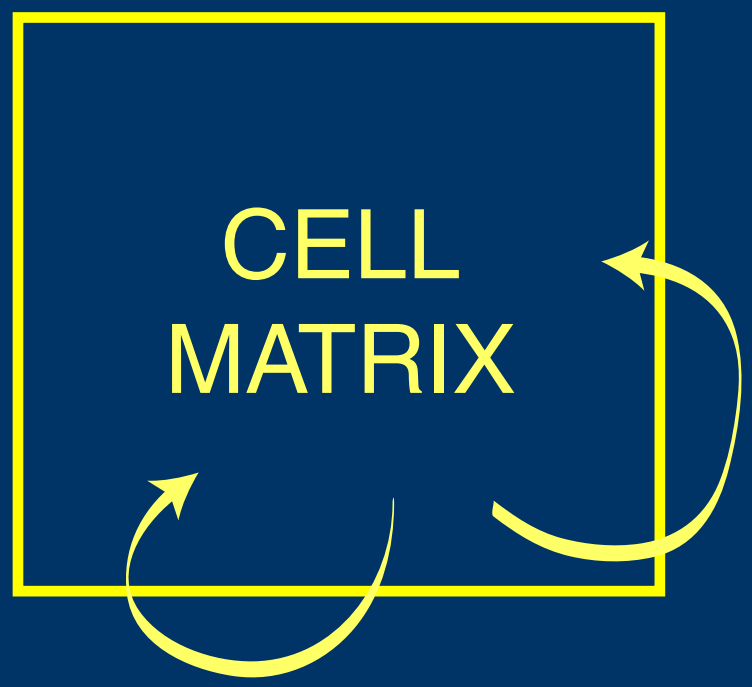
CPU configures FPGA.
FPGA does not configure itself
FPGA does NOT configure CPU

CELL MATRIX CONFIGURATION

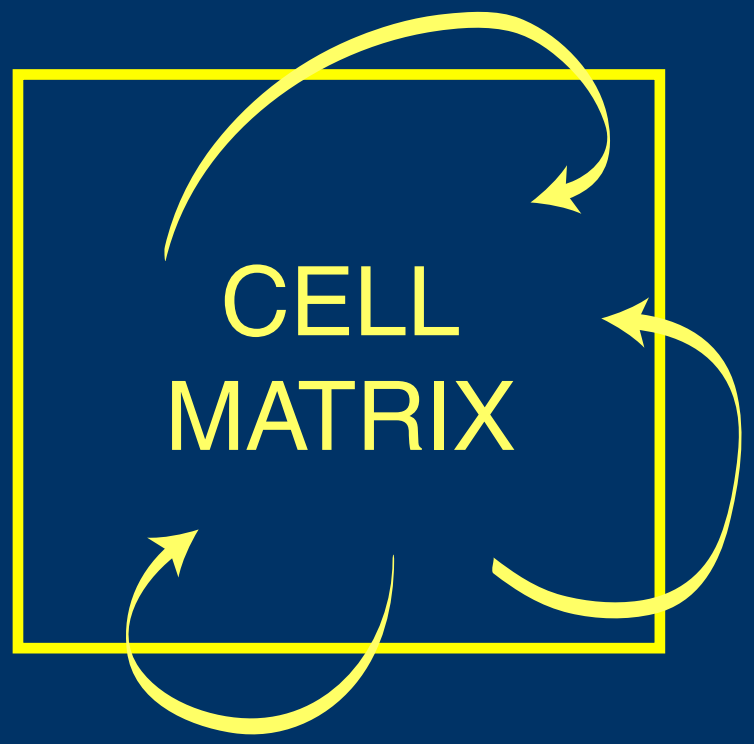
CELL
MATRIX

CELL MATRIX CONFIGURATION

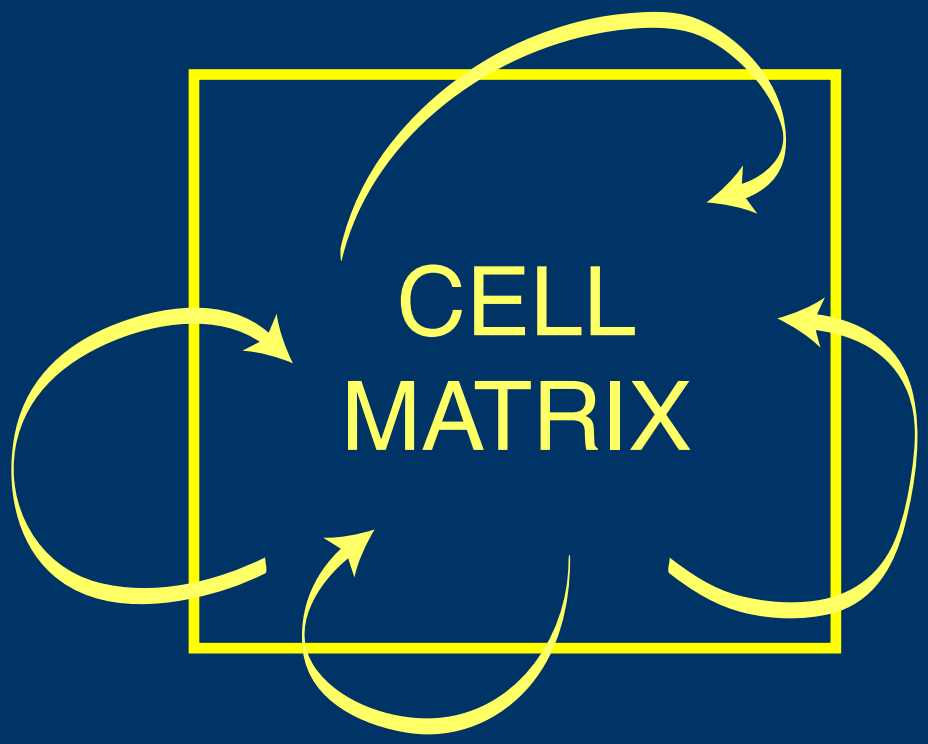
CELL MATRIX CONFIGURATION



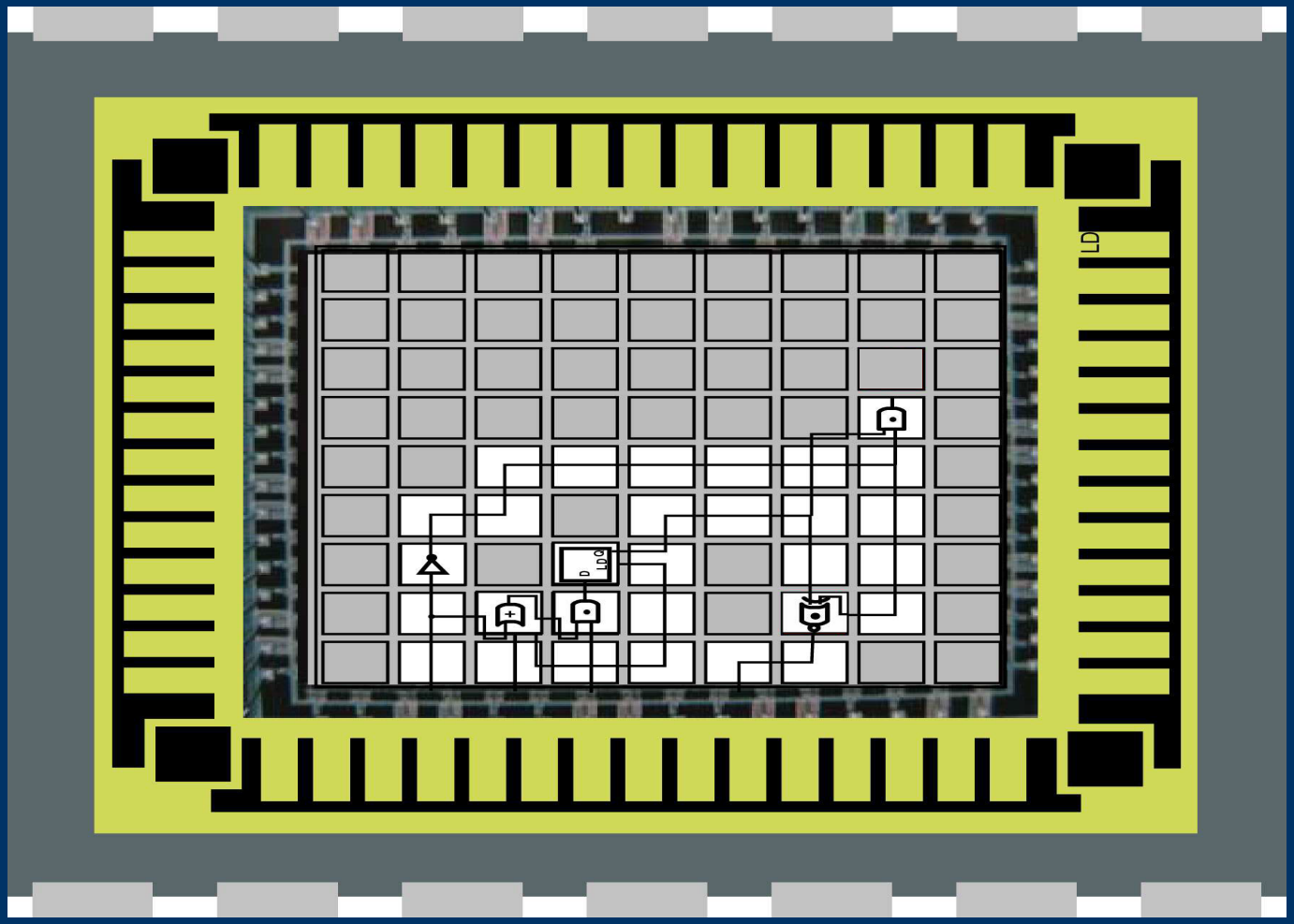
CELL MATRIX CONFIGURATION



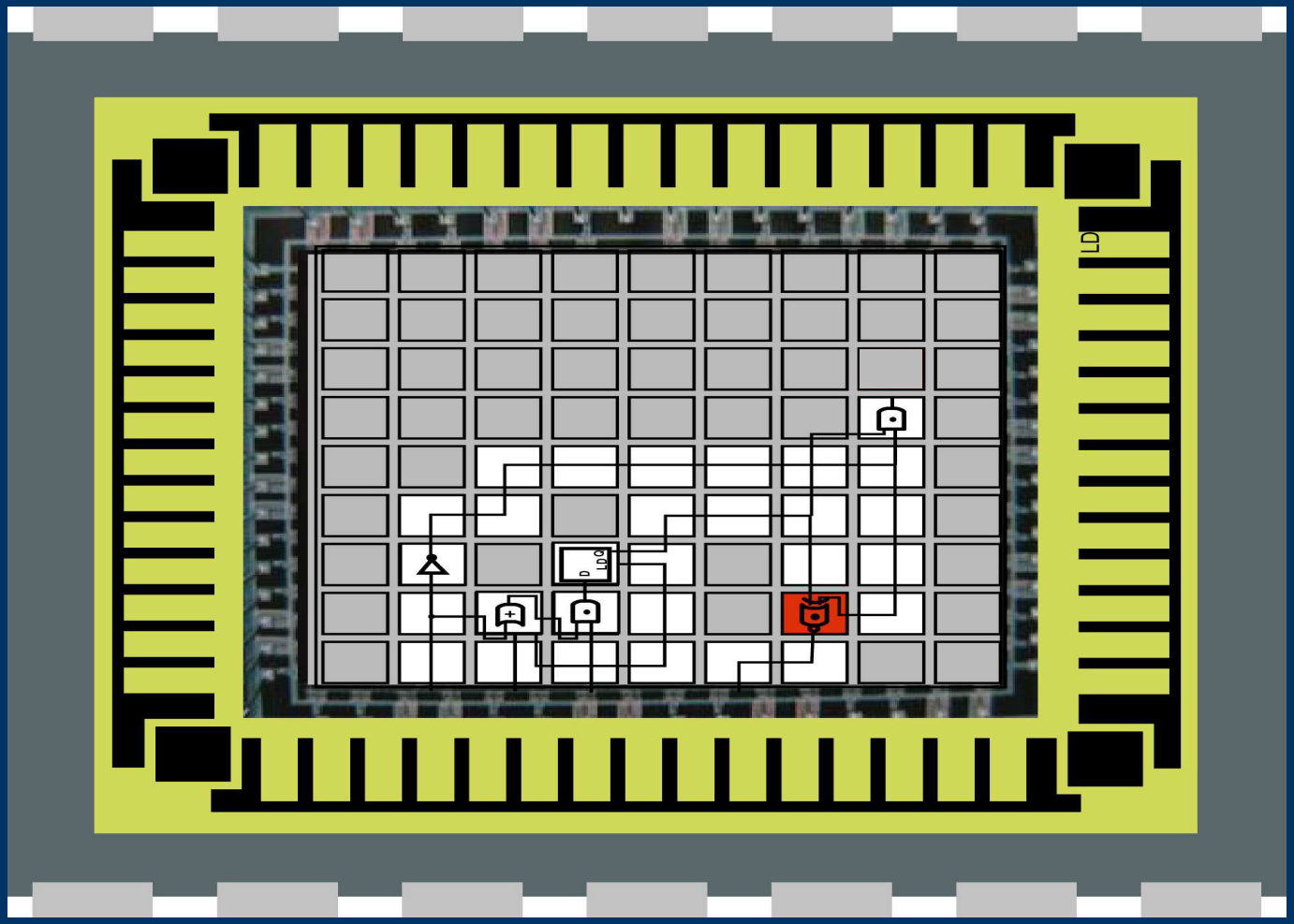
CELL MATRIX CONFIGURATION



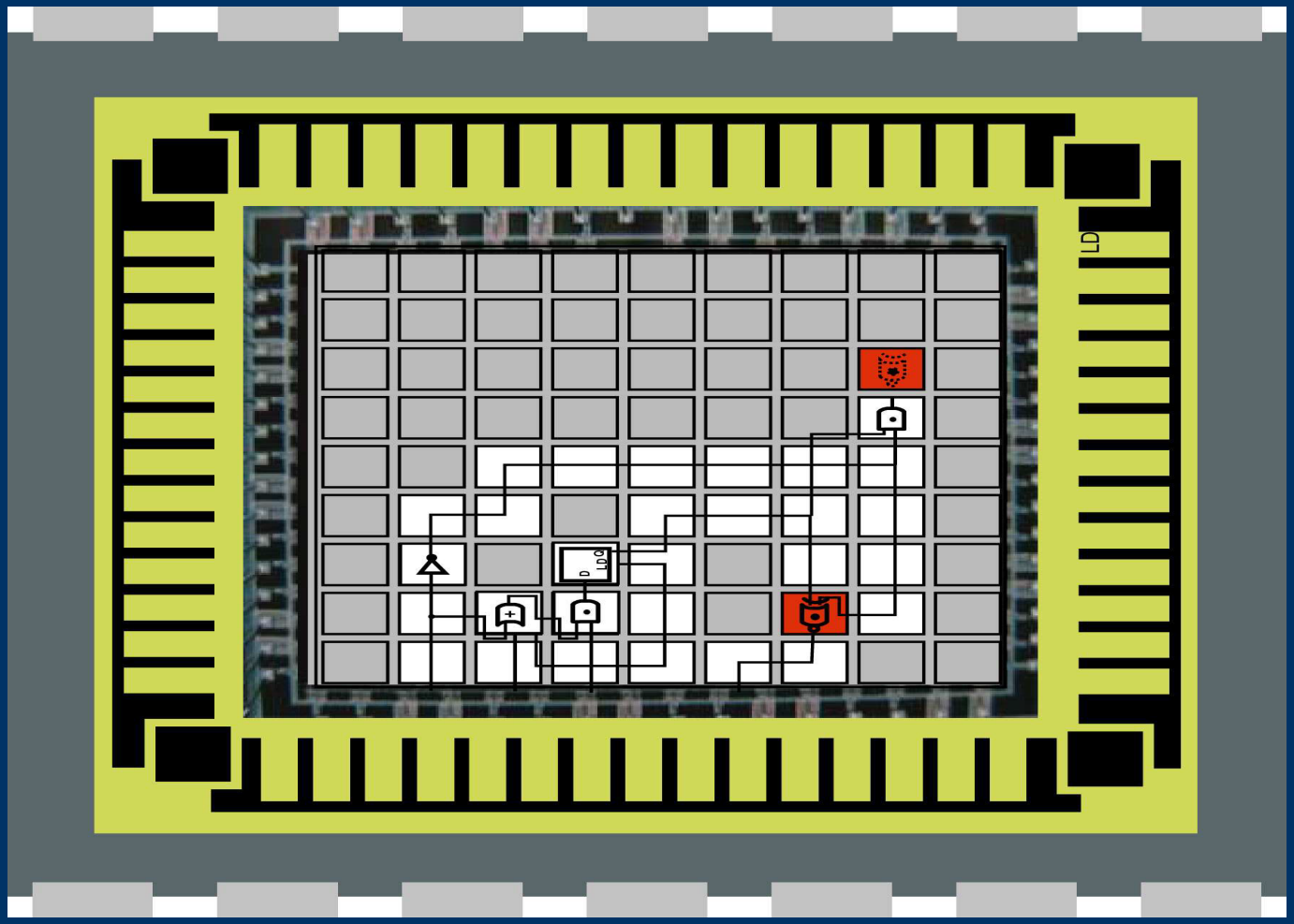
CIRCUITRY CAN PROCESS DATA



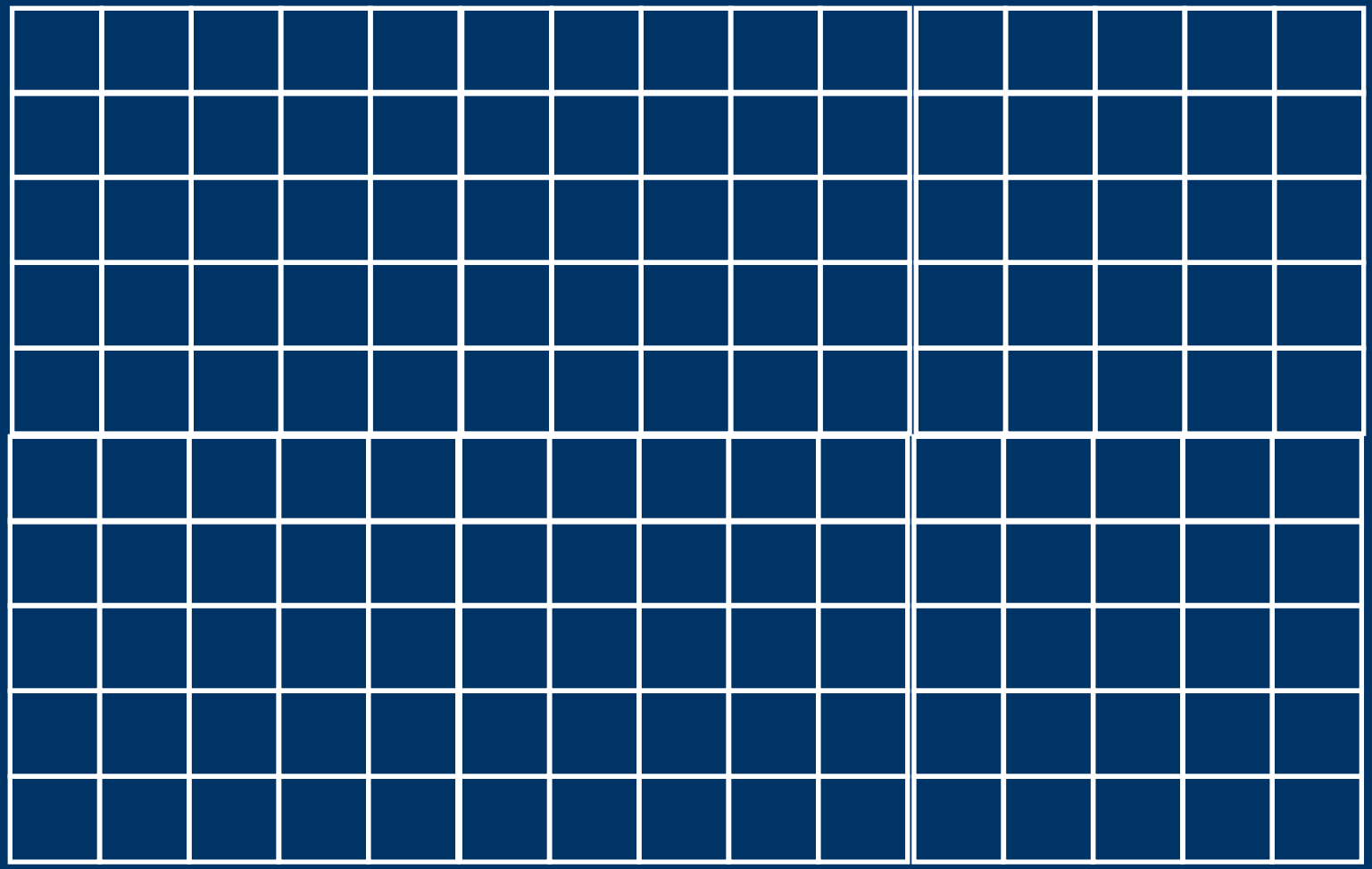
CIRCUITRY CAN PROCESS HARDWARE



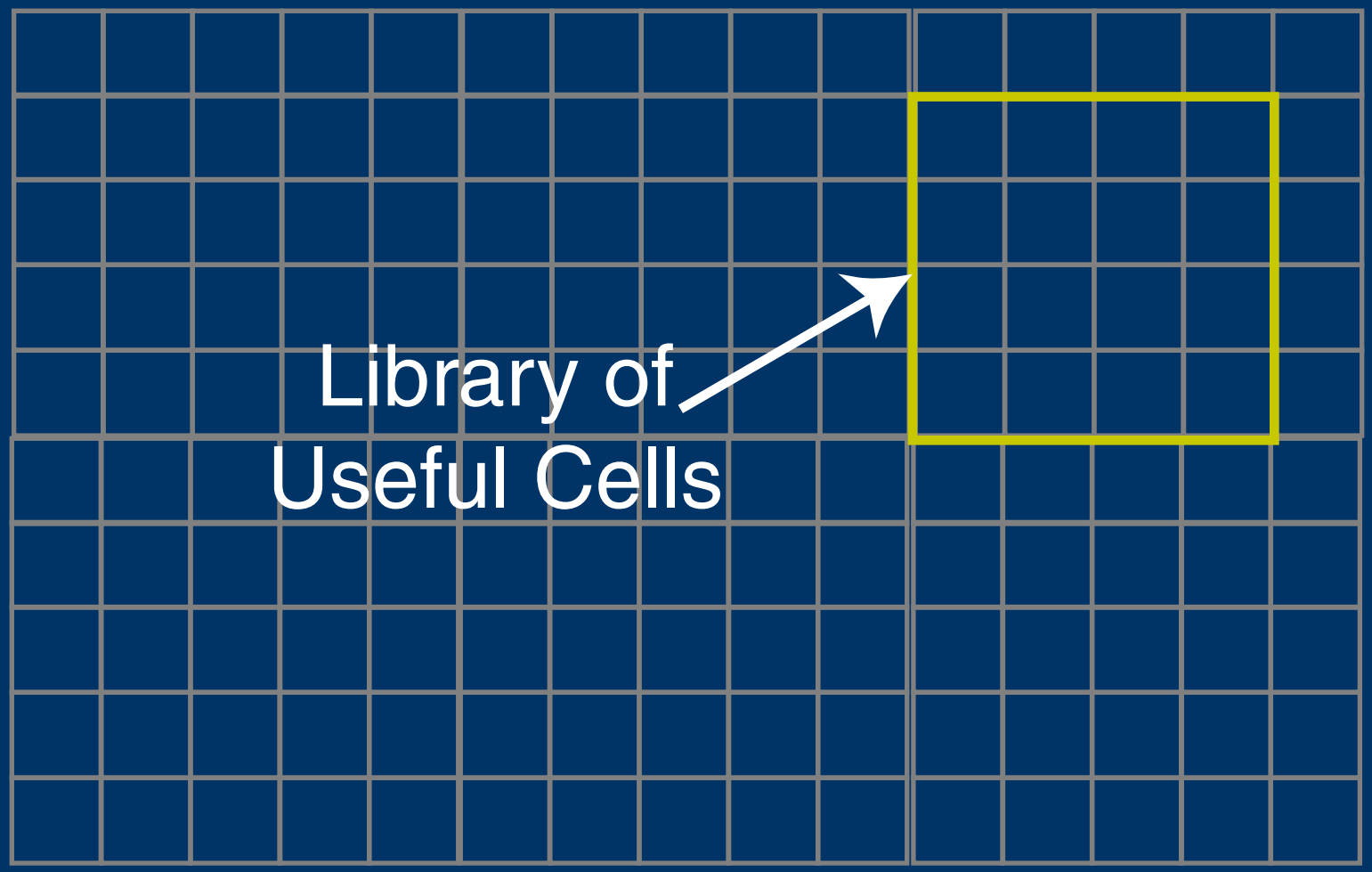
CIRCUITRY CAN PROCESS HARDWARE



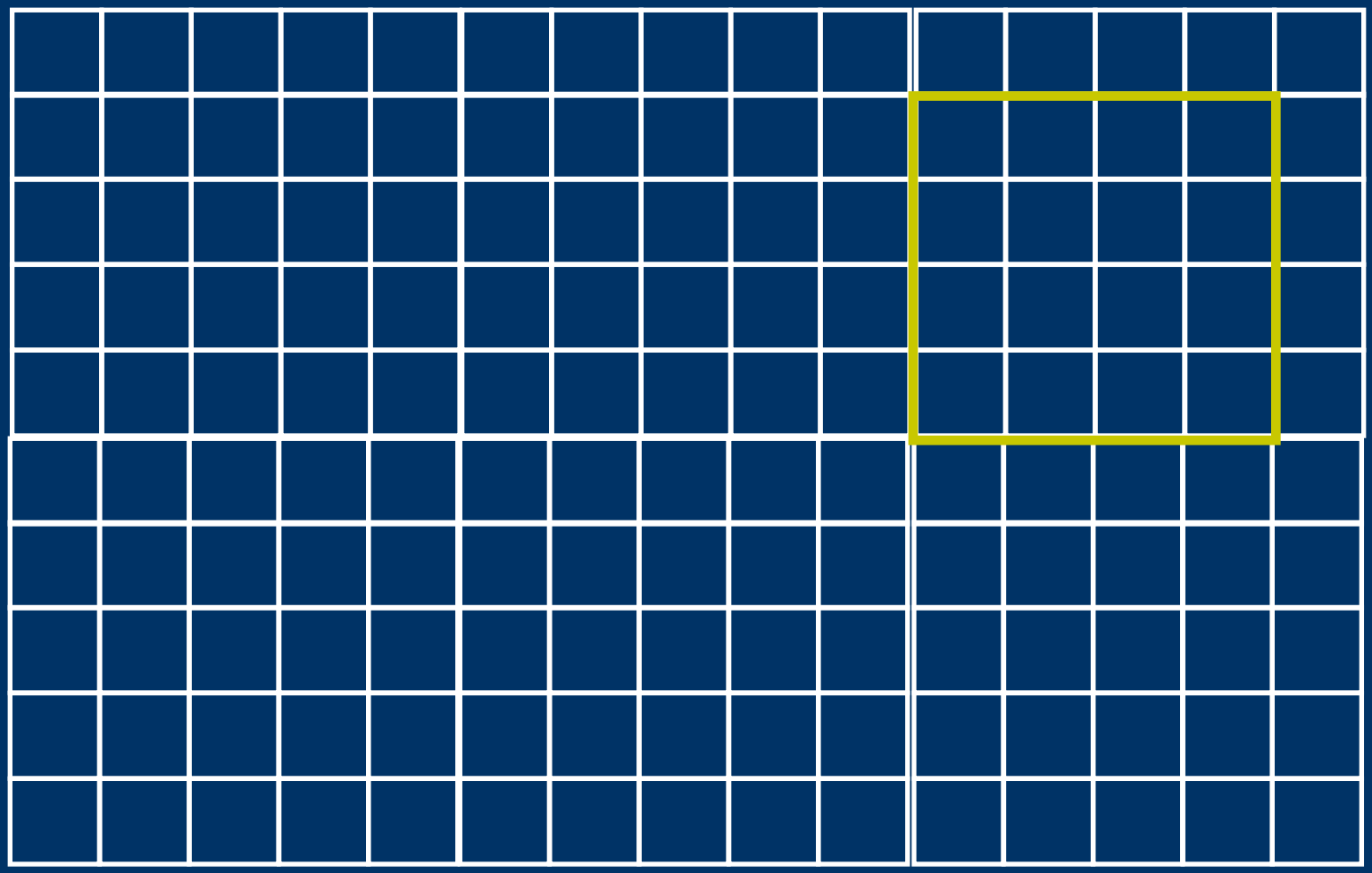
EXAMPLE: HARDWARE LIBRARY



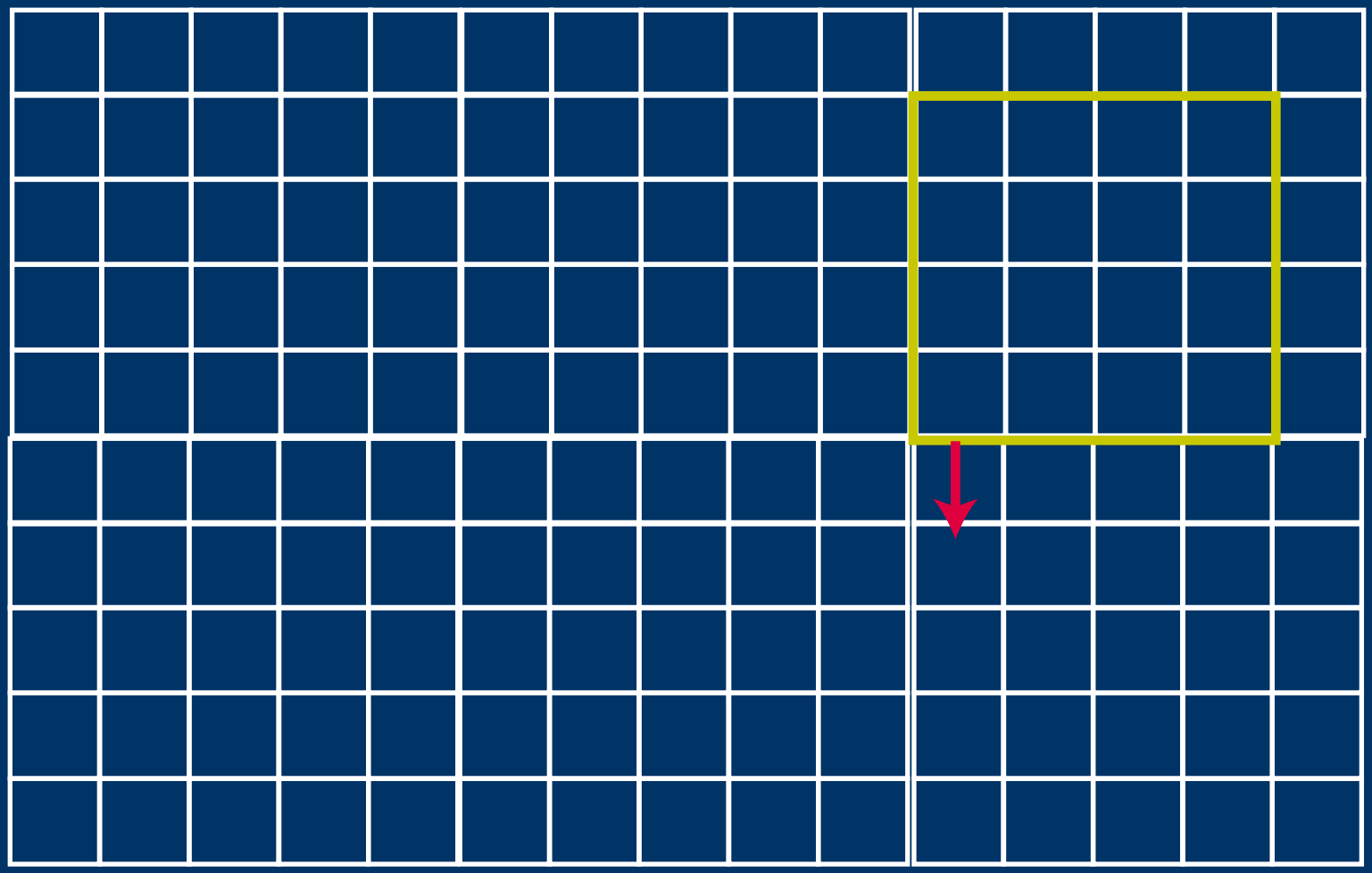
EXAMPLE: HARDWARE LIBRARY



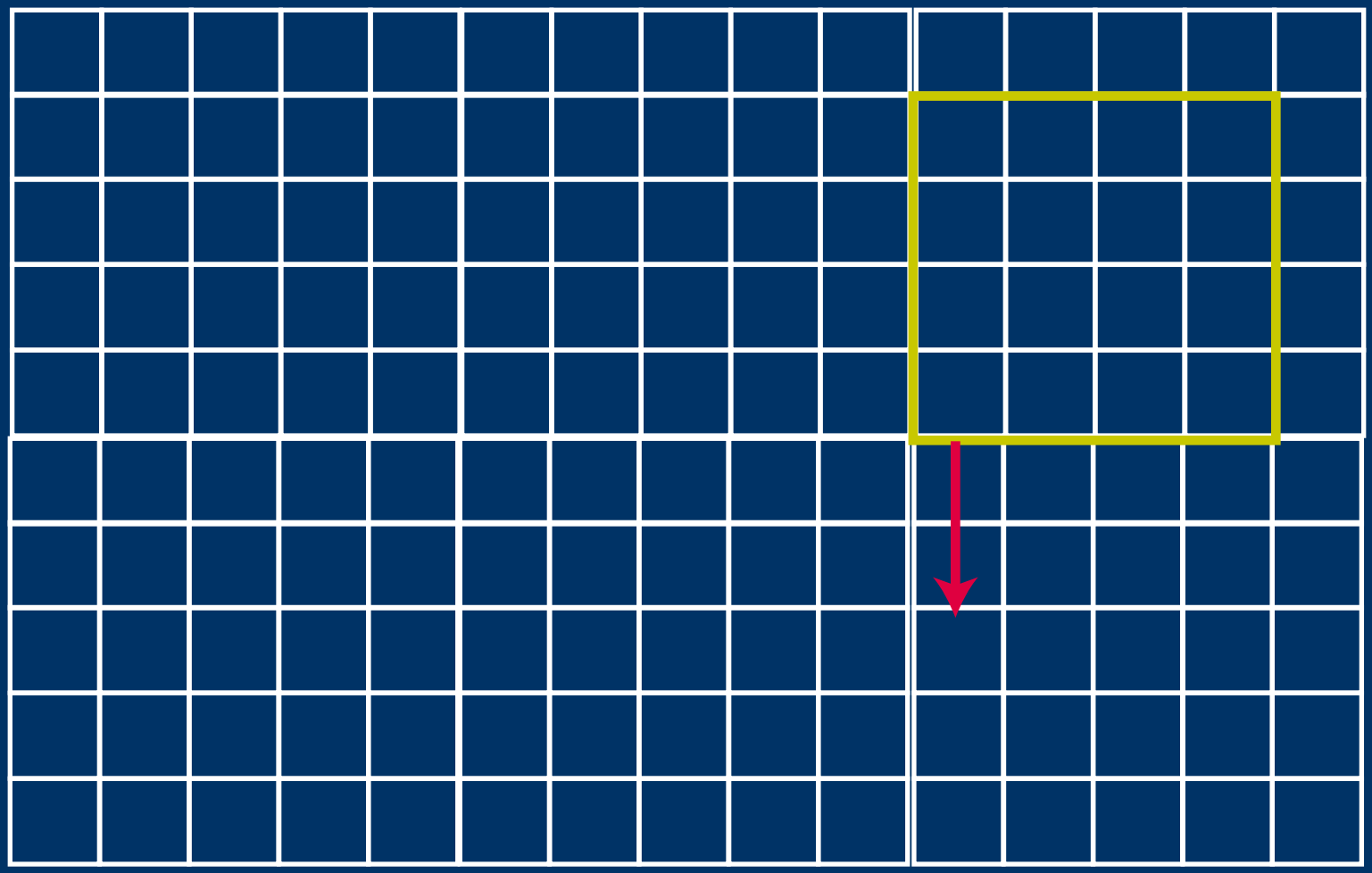
EXAMPLE: HARDWARE LIBRARY



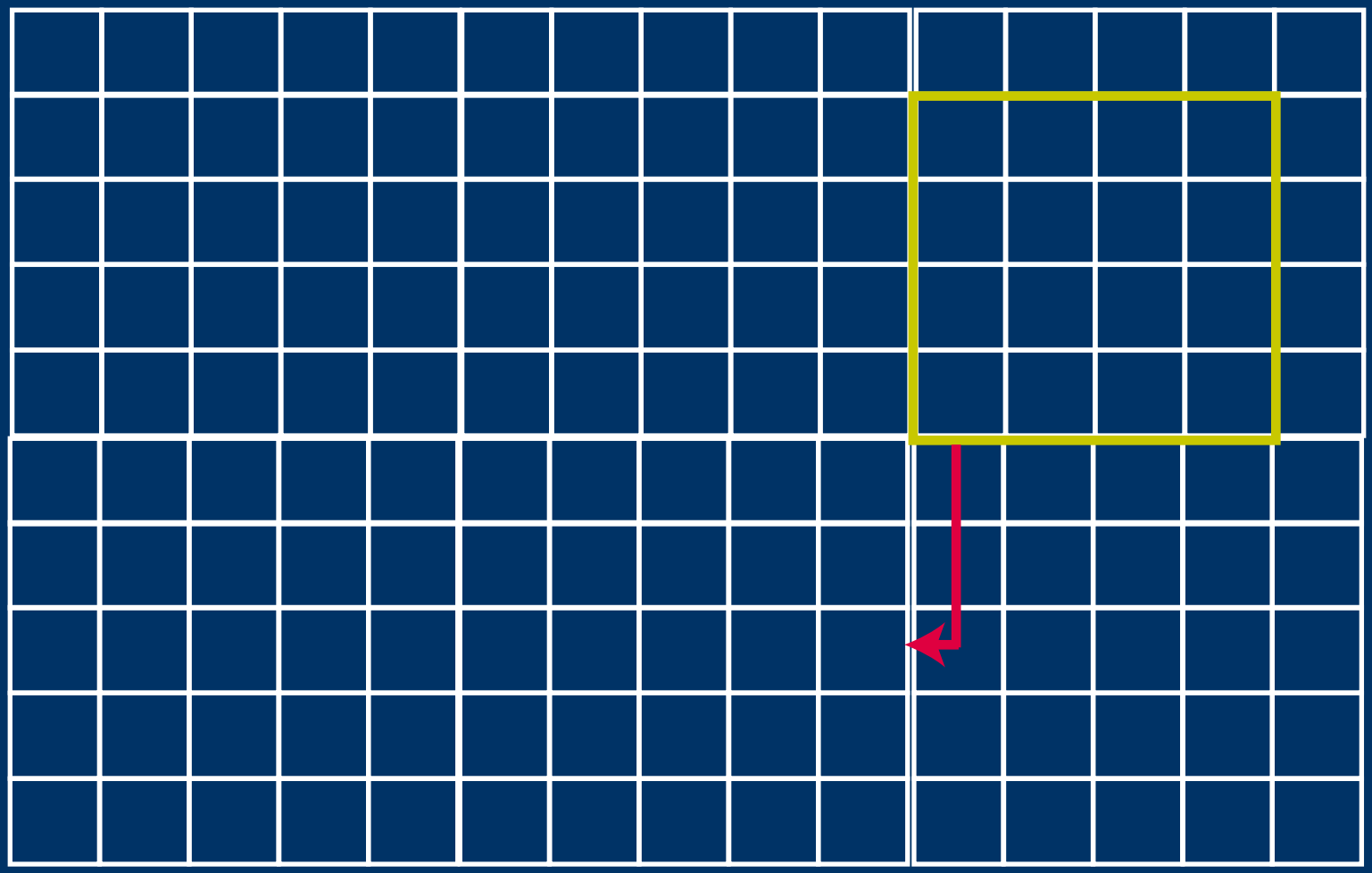
EXAMPLE: HARDWARE LIBRARY



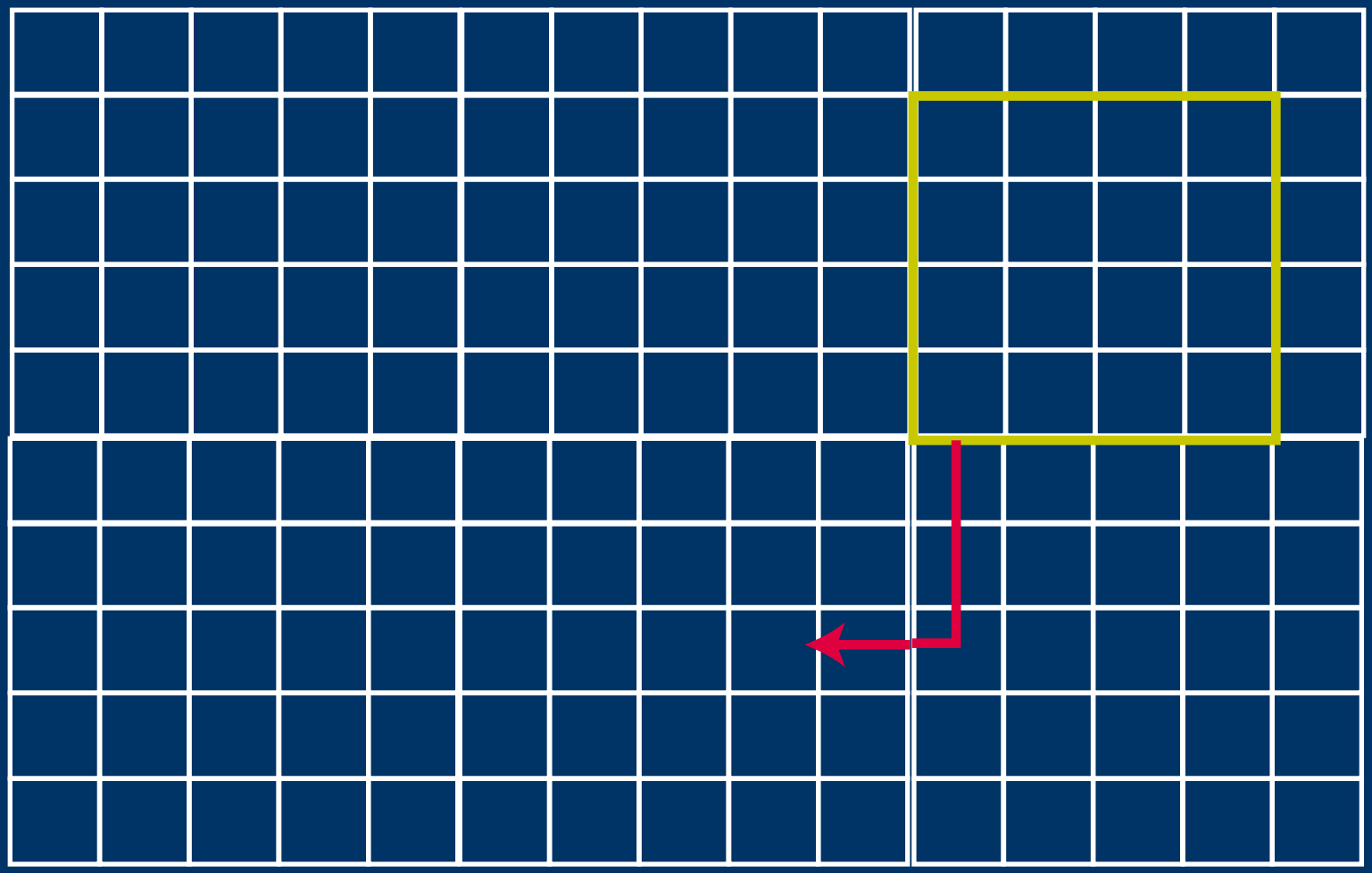
EXAMPLE: HARDWARE LIBRARY



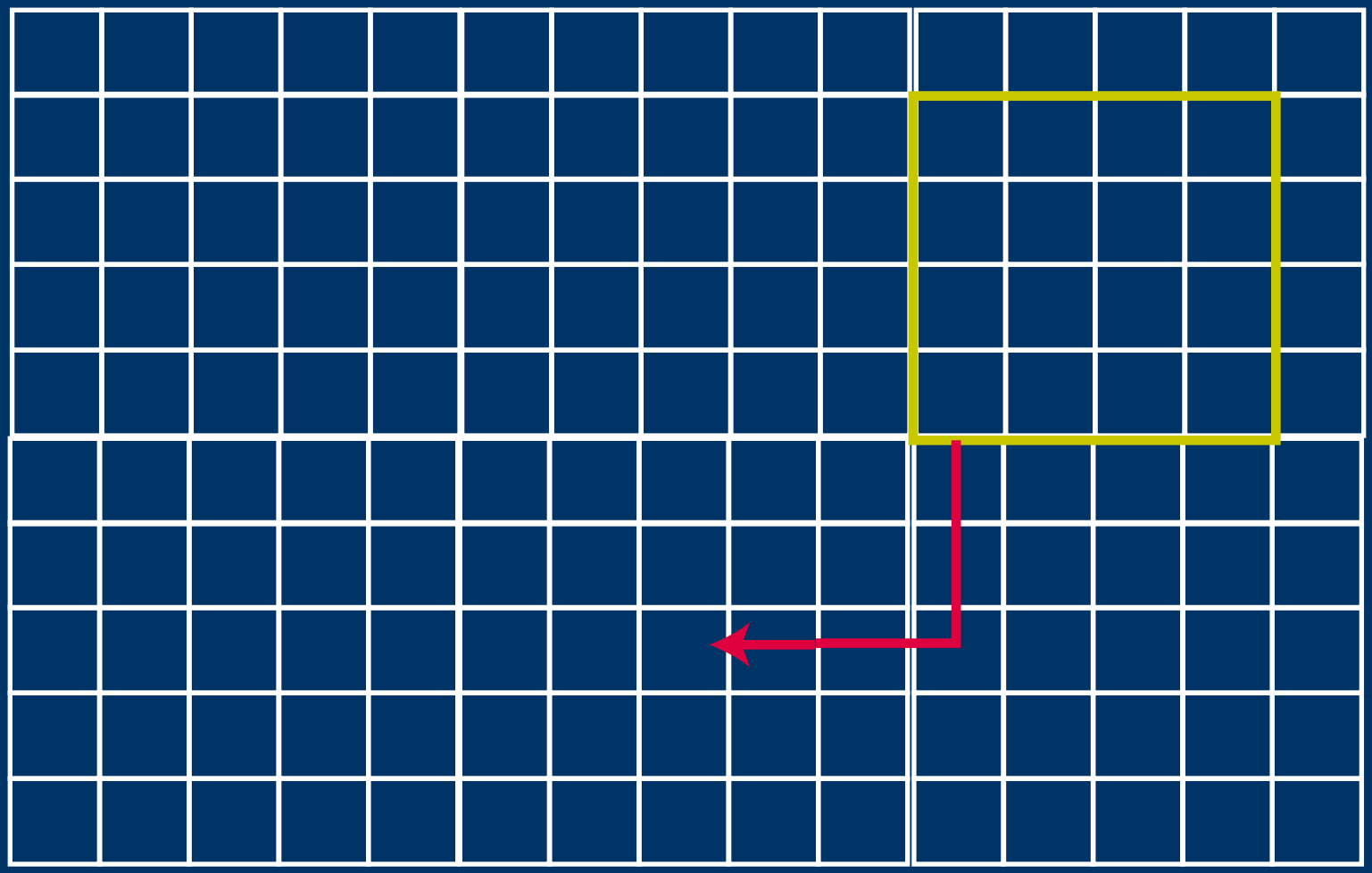
EXAMPLE: HARDWARE LIBRARY



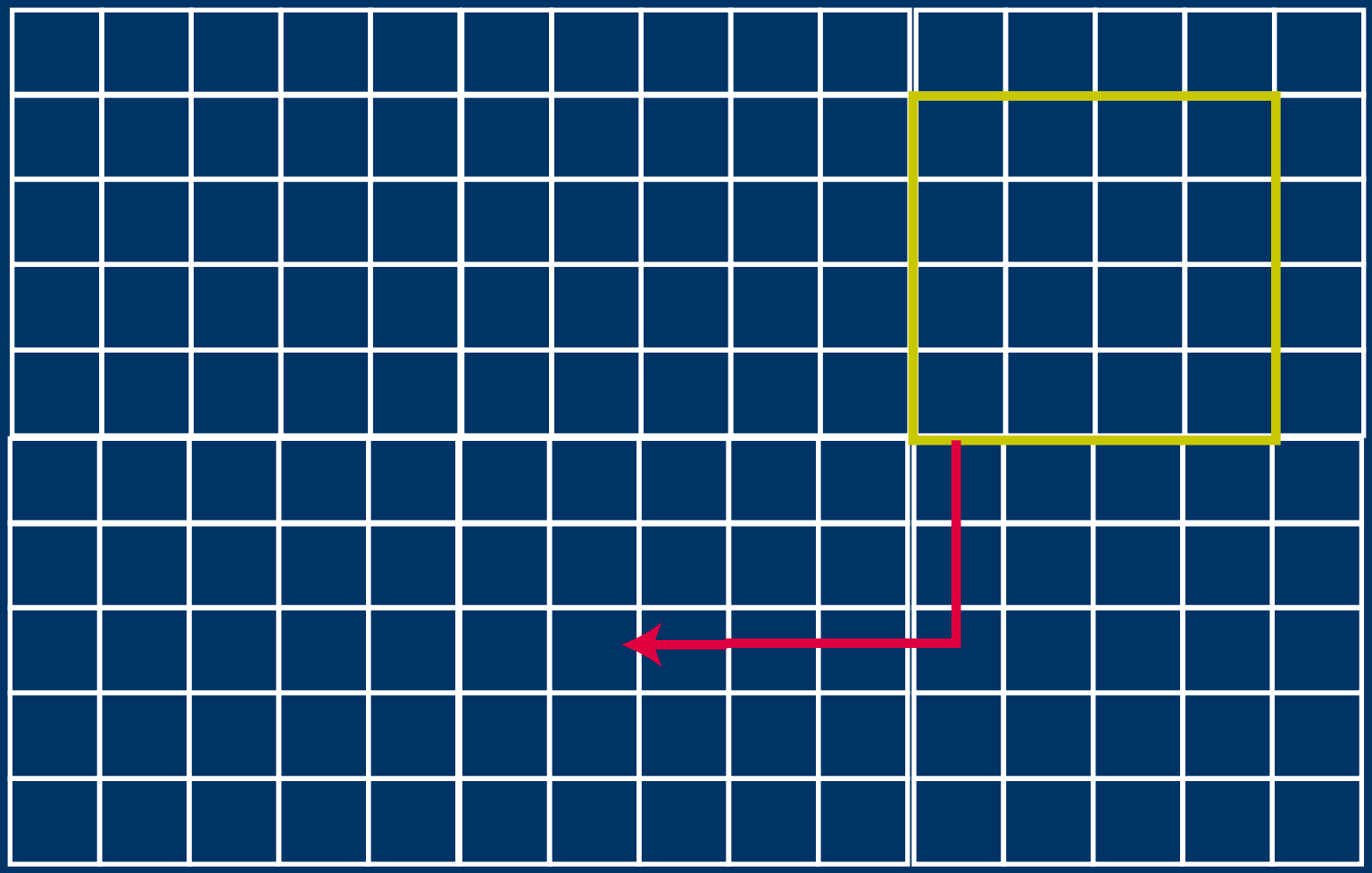
EXAMPLE: HARDWARE LIBRARY



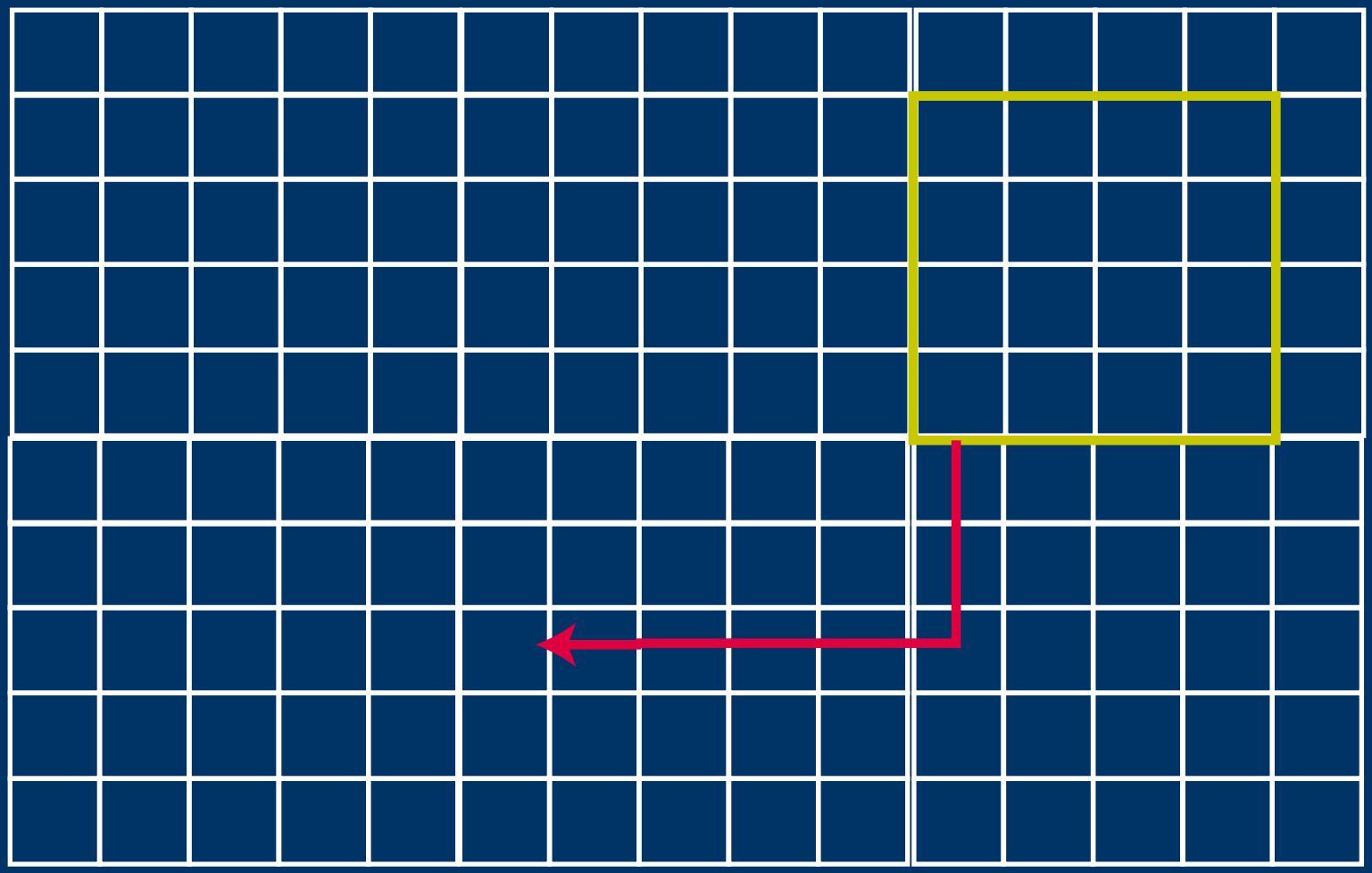
EXAMPLE: HARDWARE LIBRARY



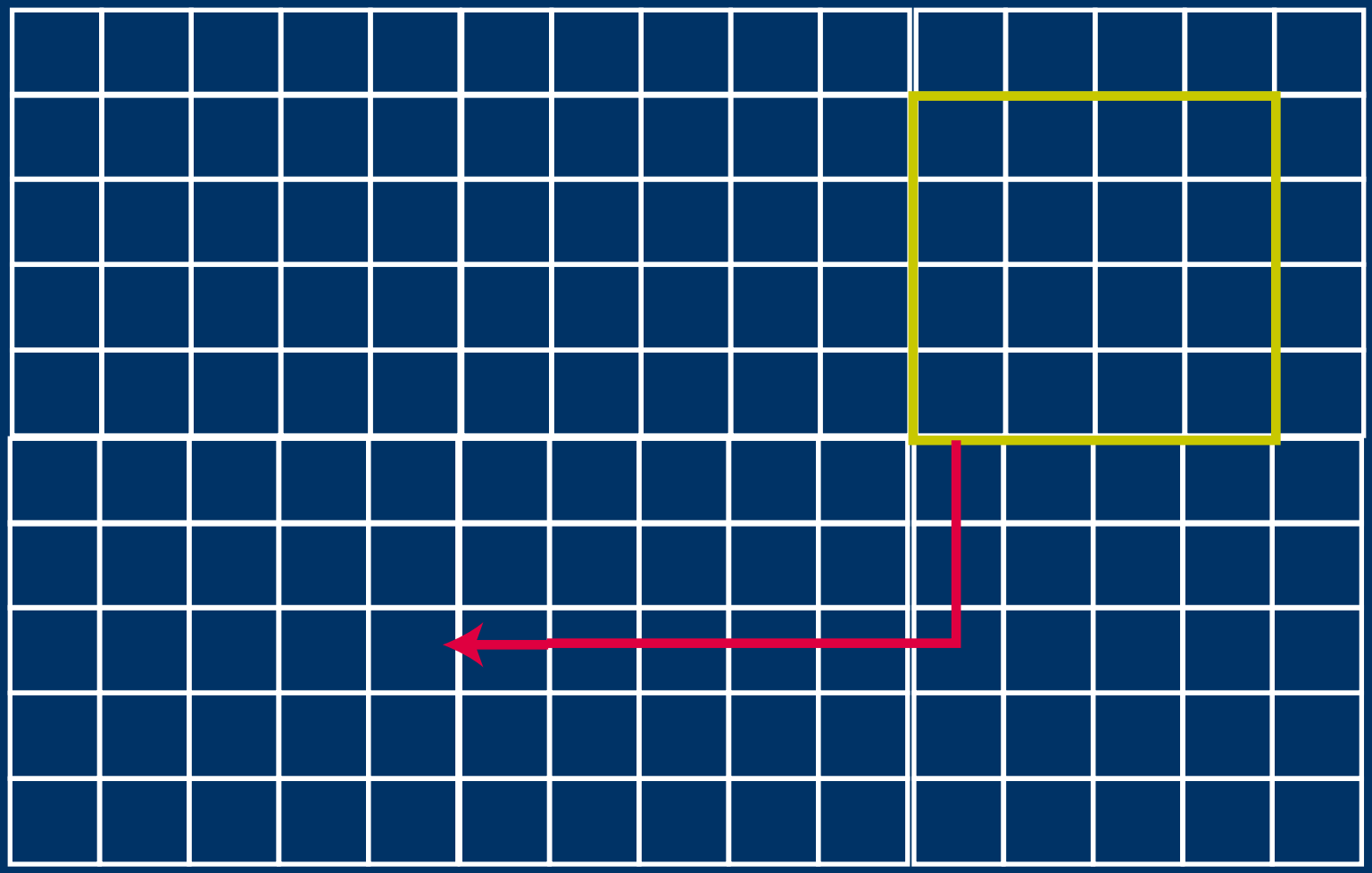
EXAMPLE: HARDWARE LIBRARY



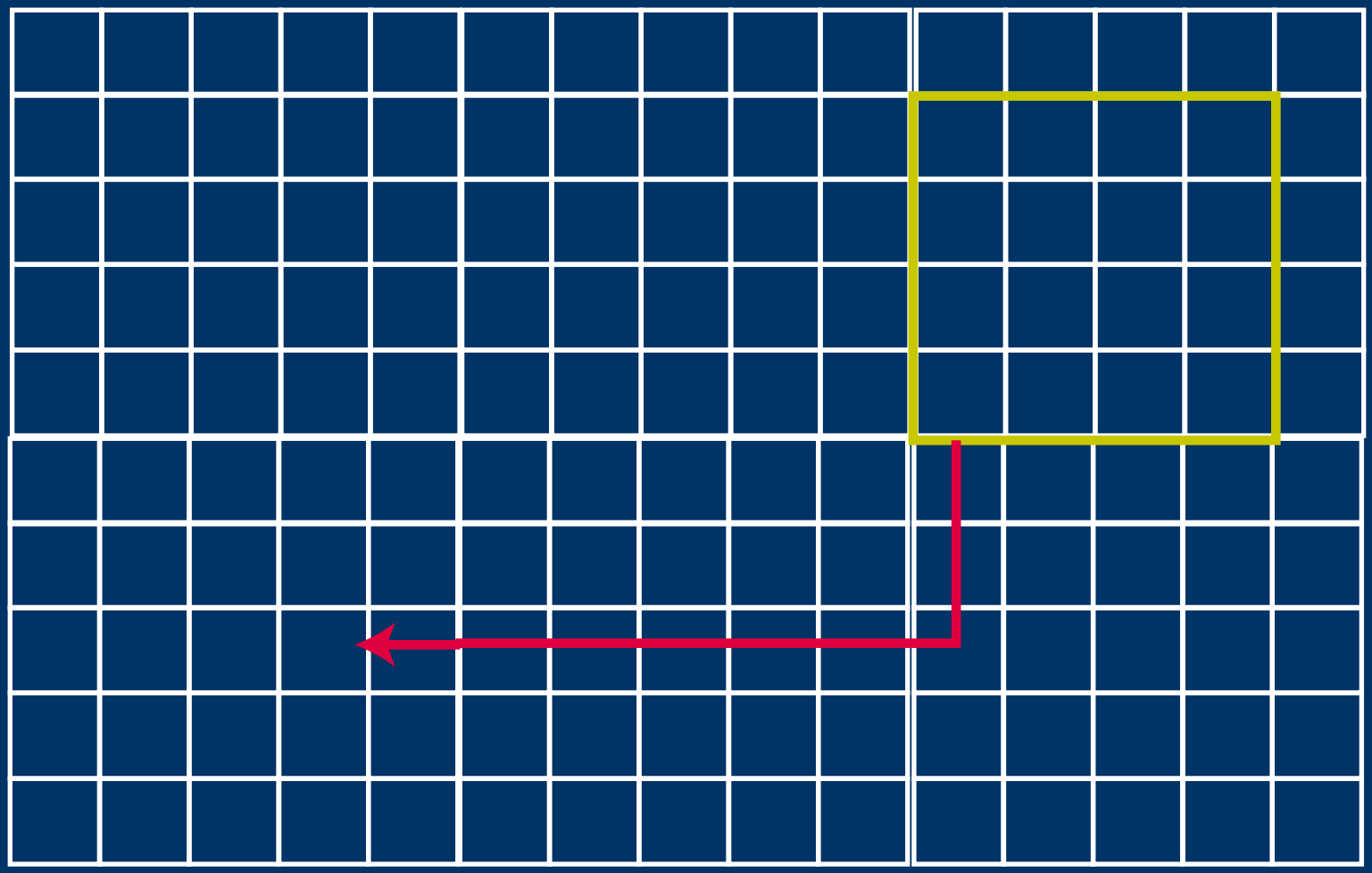
EXAMPLE: HARDWARE LIBRARY



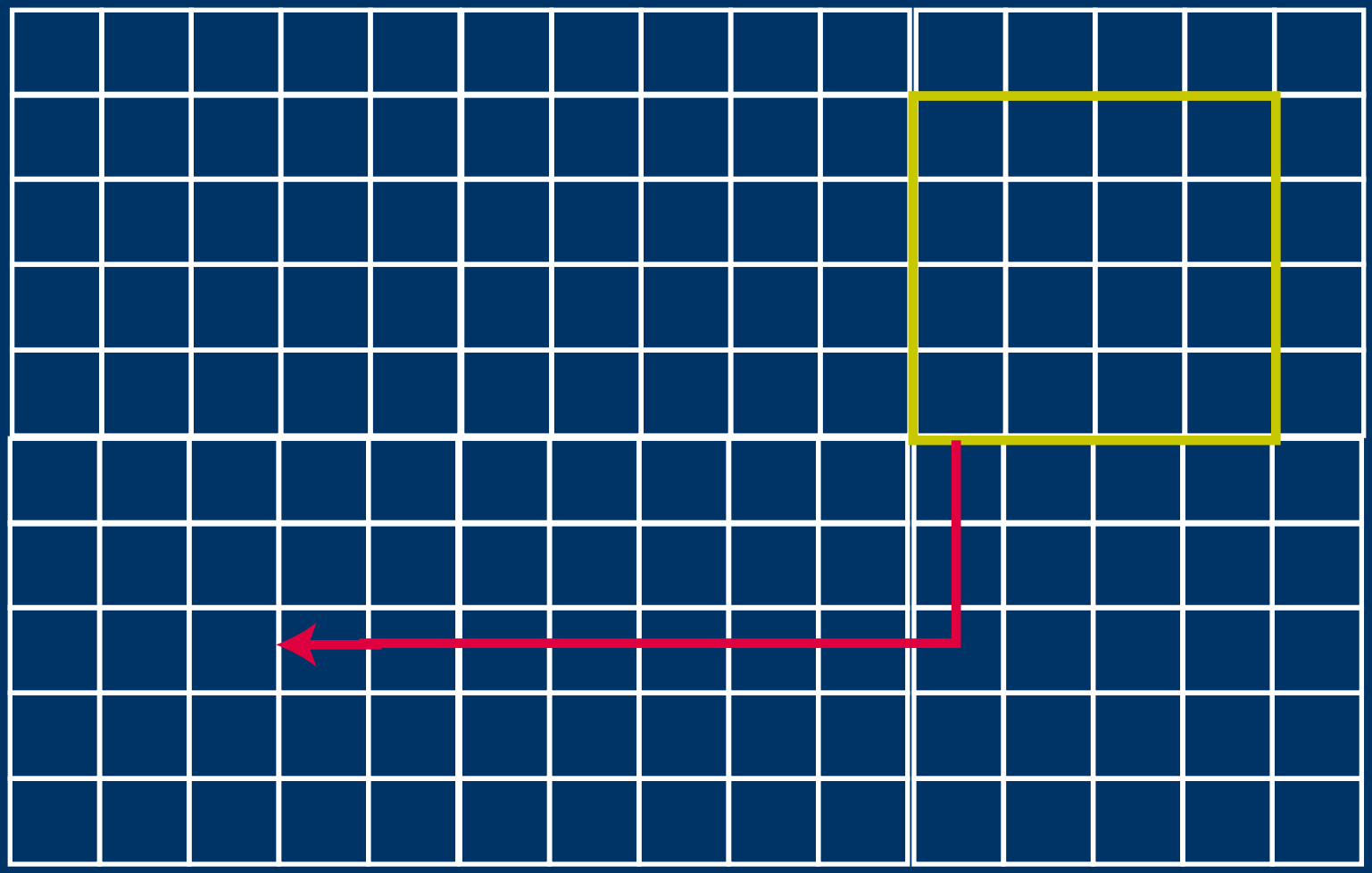
EXAMPLE: HARDWARE LIBRARY



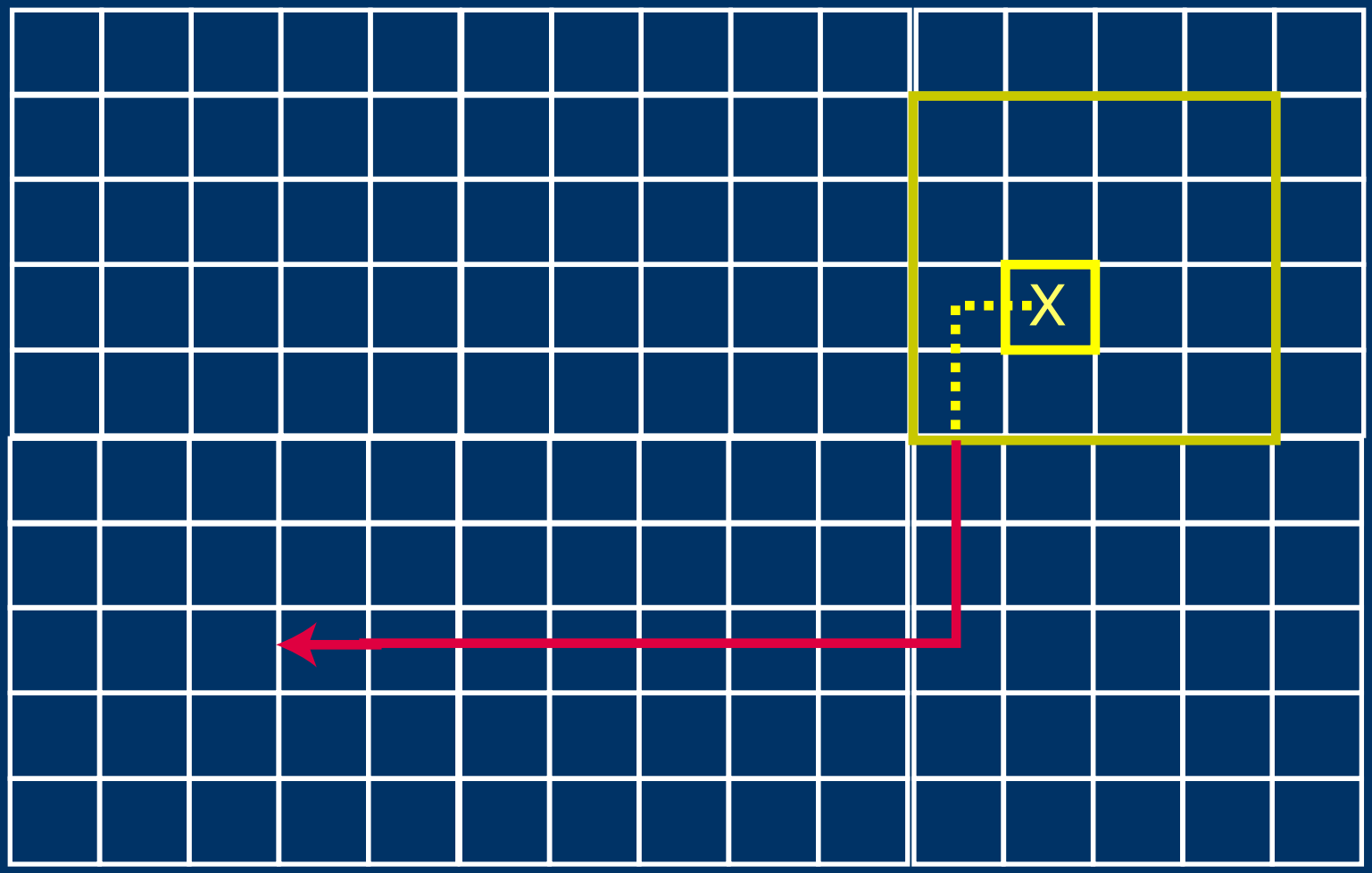
EXAMPLE: HARDWARE LIBRARY



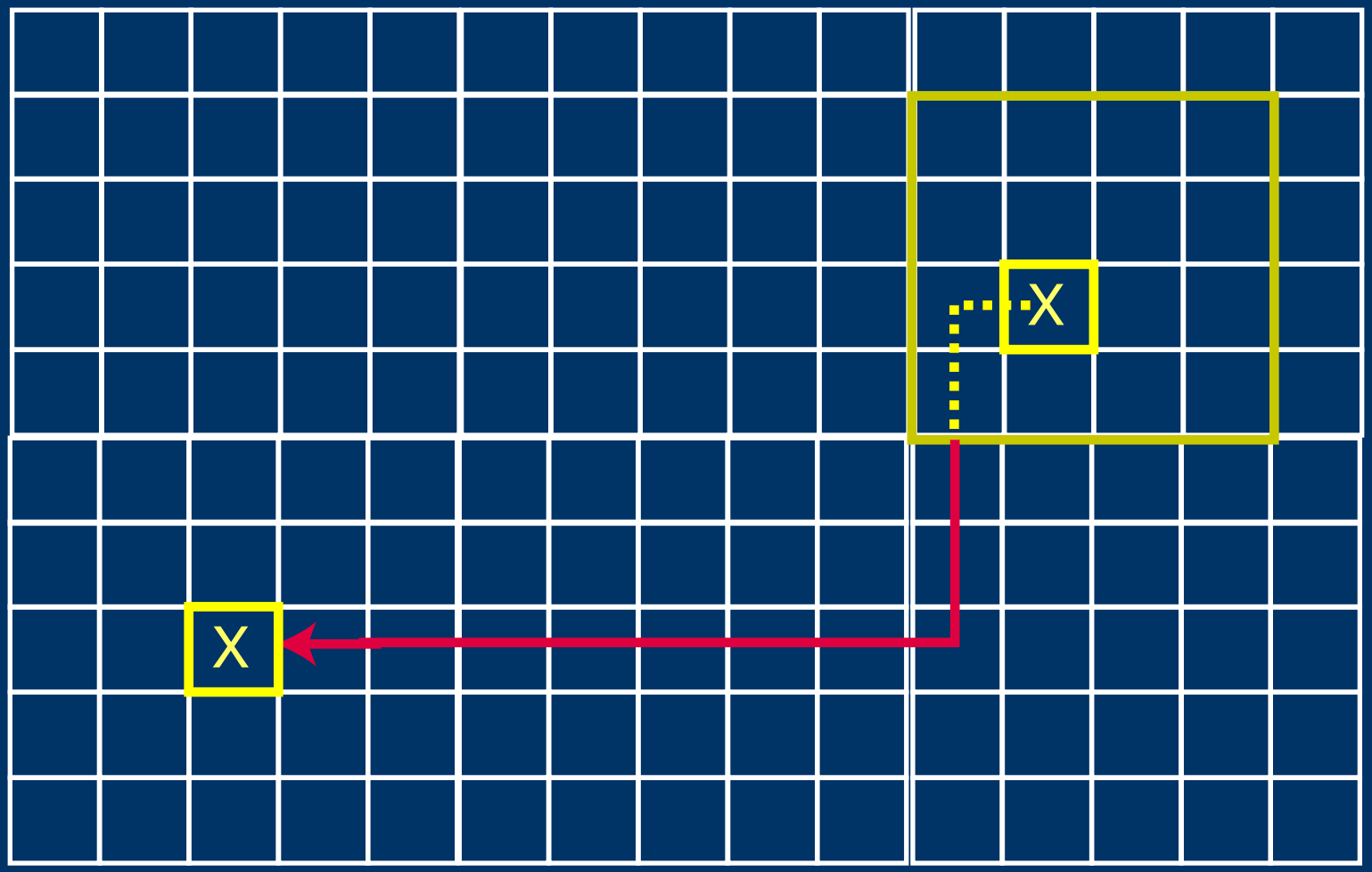
EXAMPLE: HARDWARE LIBRARY



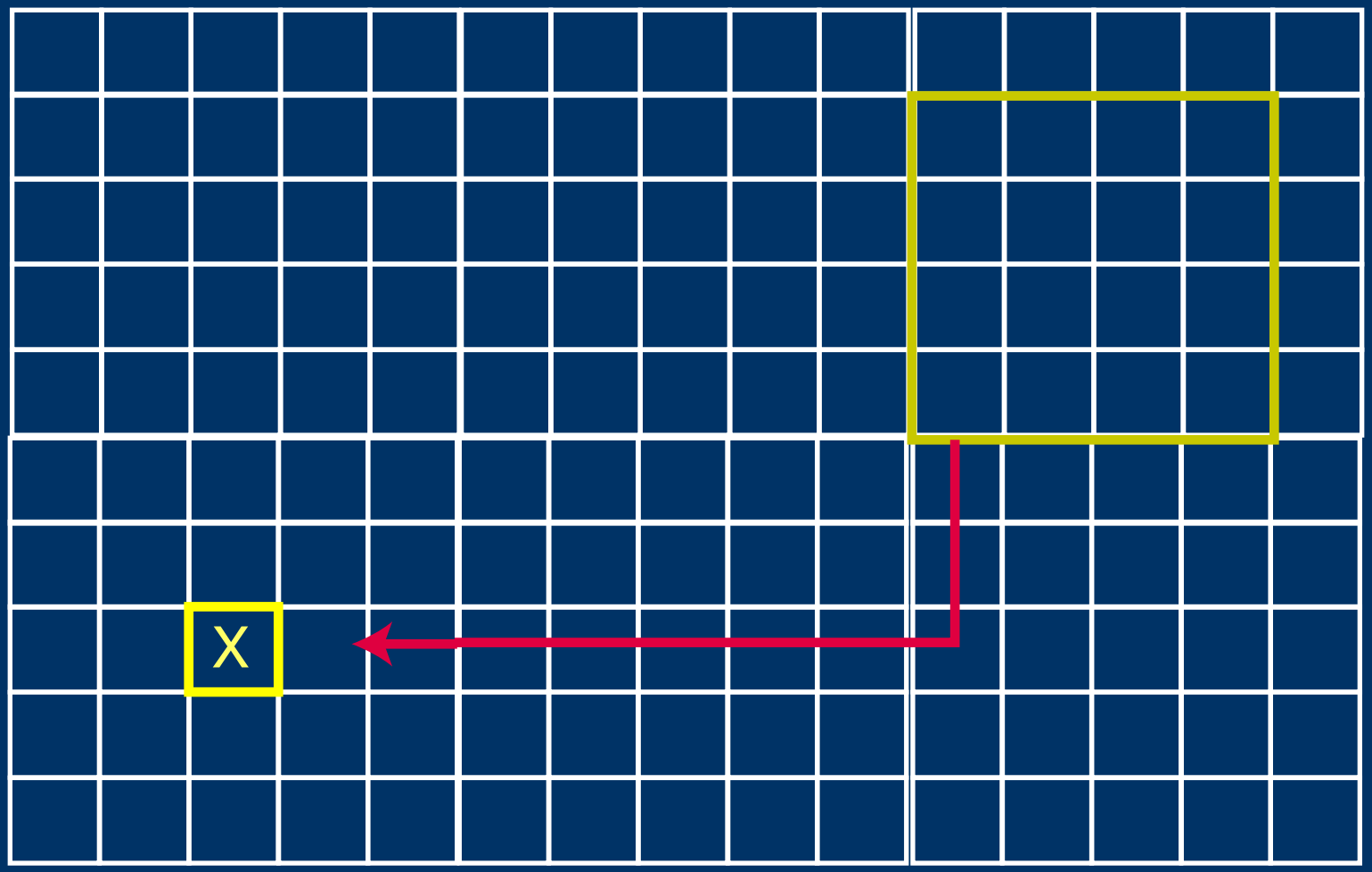
EXAMPLE: HARDWARE LIBRARY



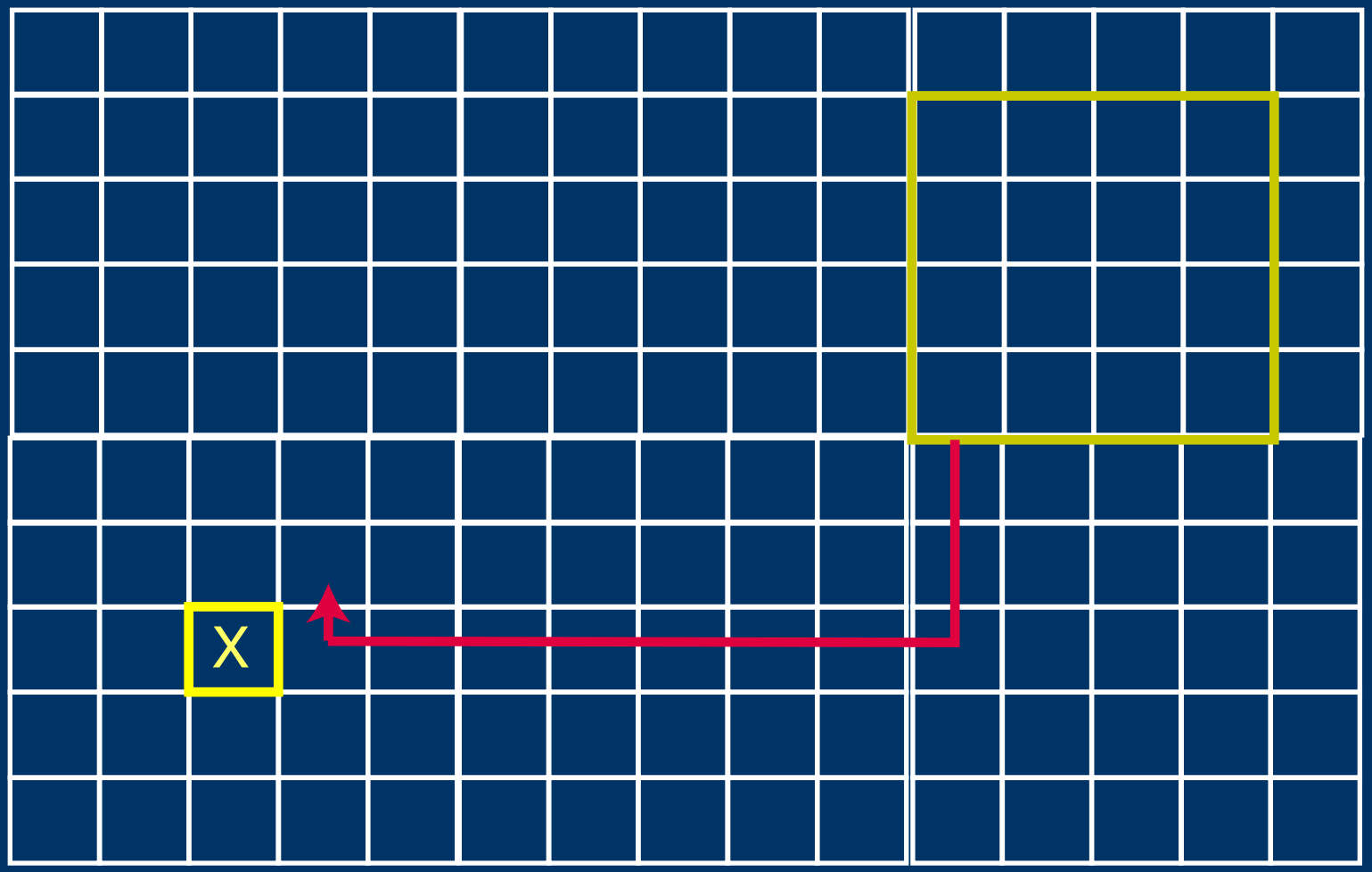
EXAMPLE: HARDWARE LIBRARY



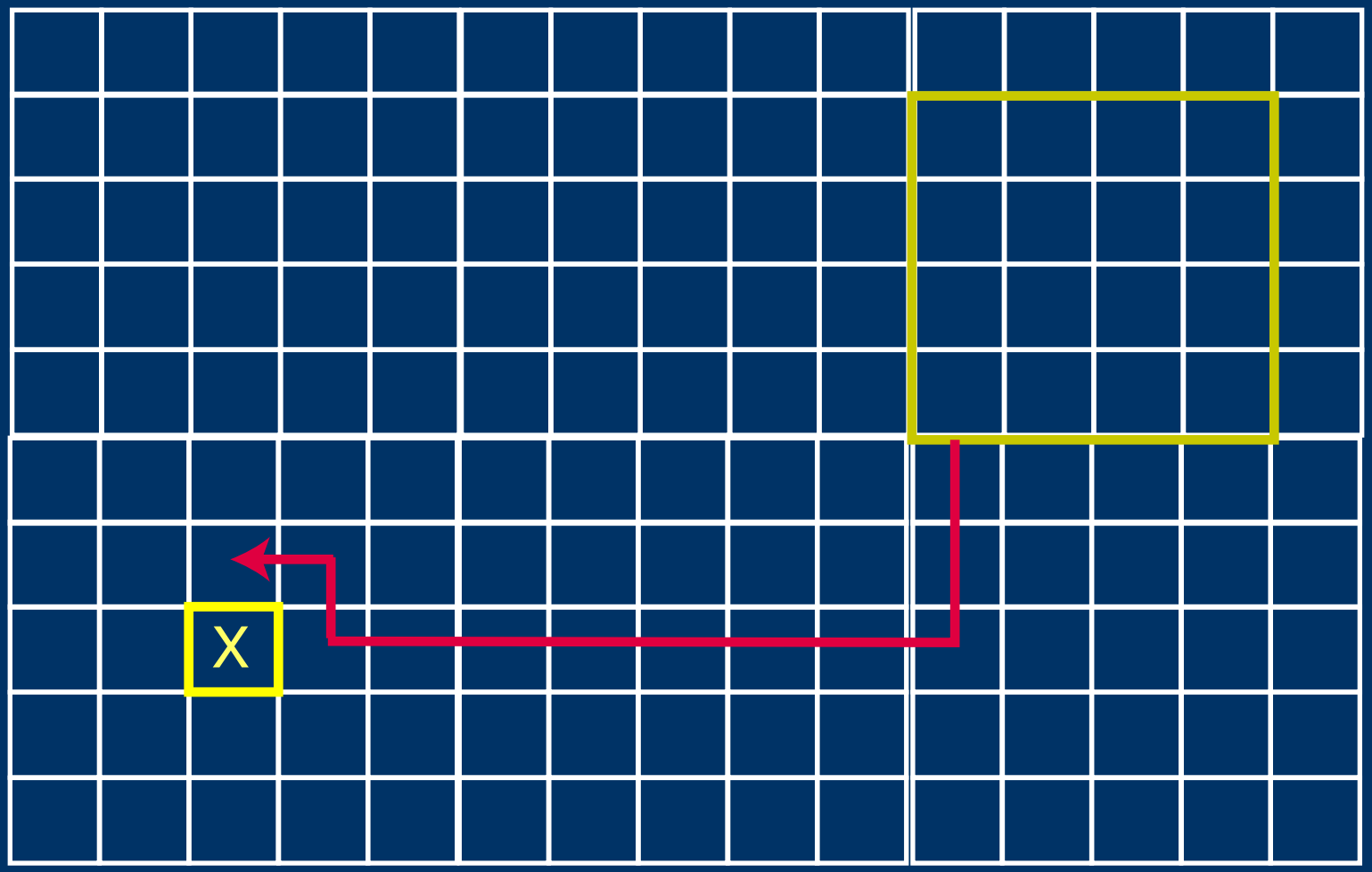
EXAMPLE: HARDWARE LIBRARY



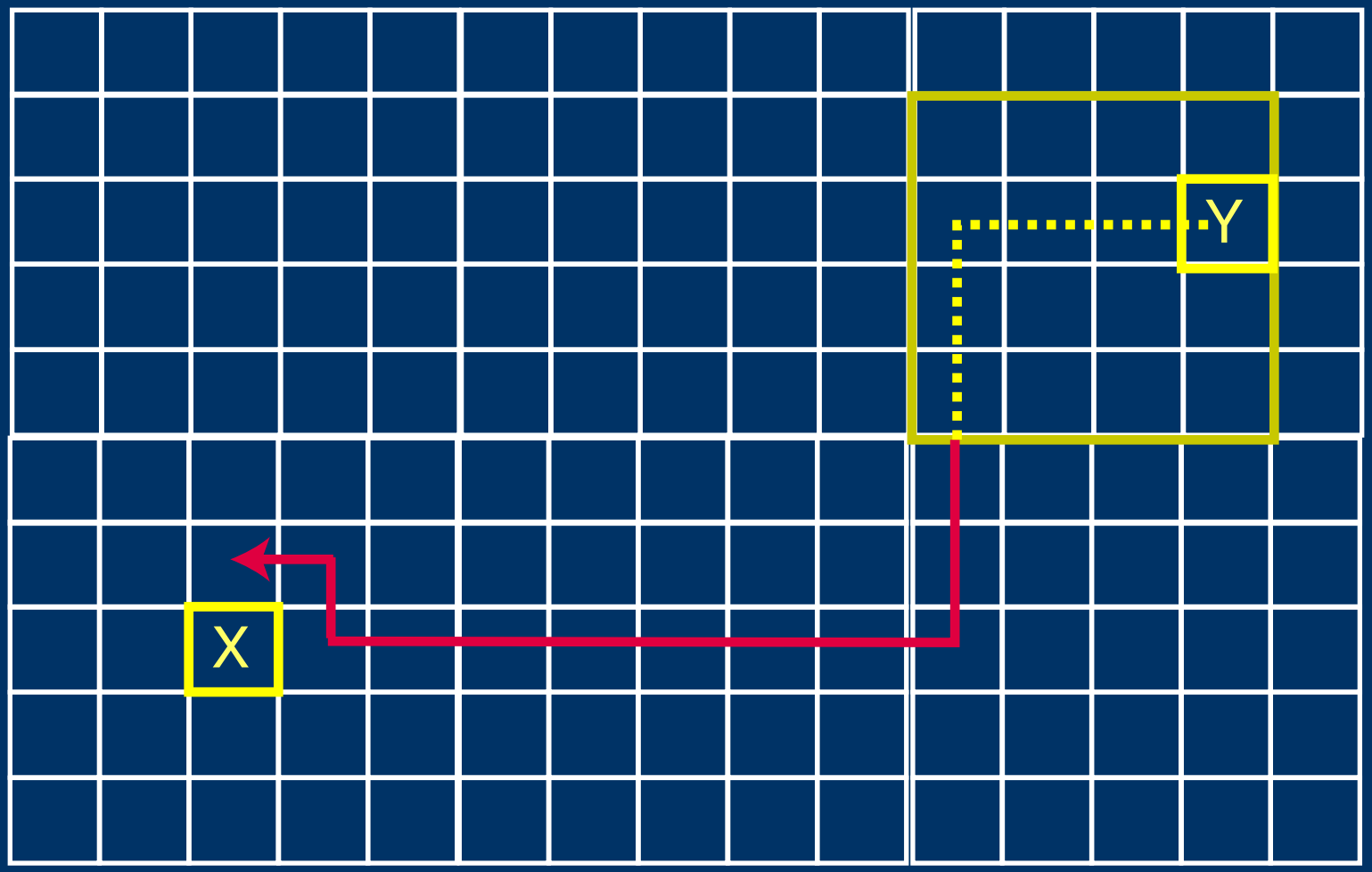
EXAMPLE: HARDWARE LIBRARY



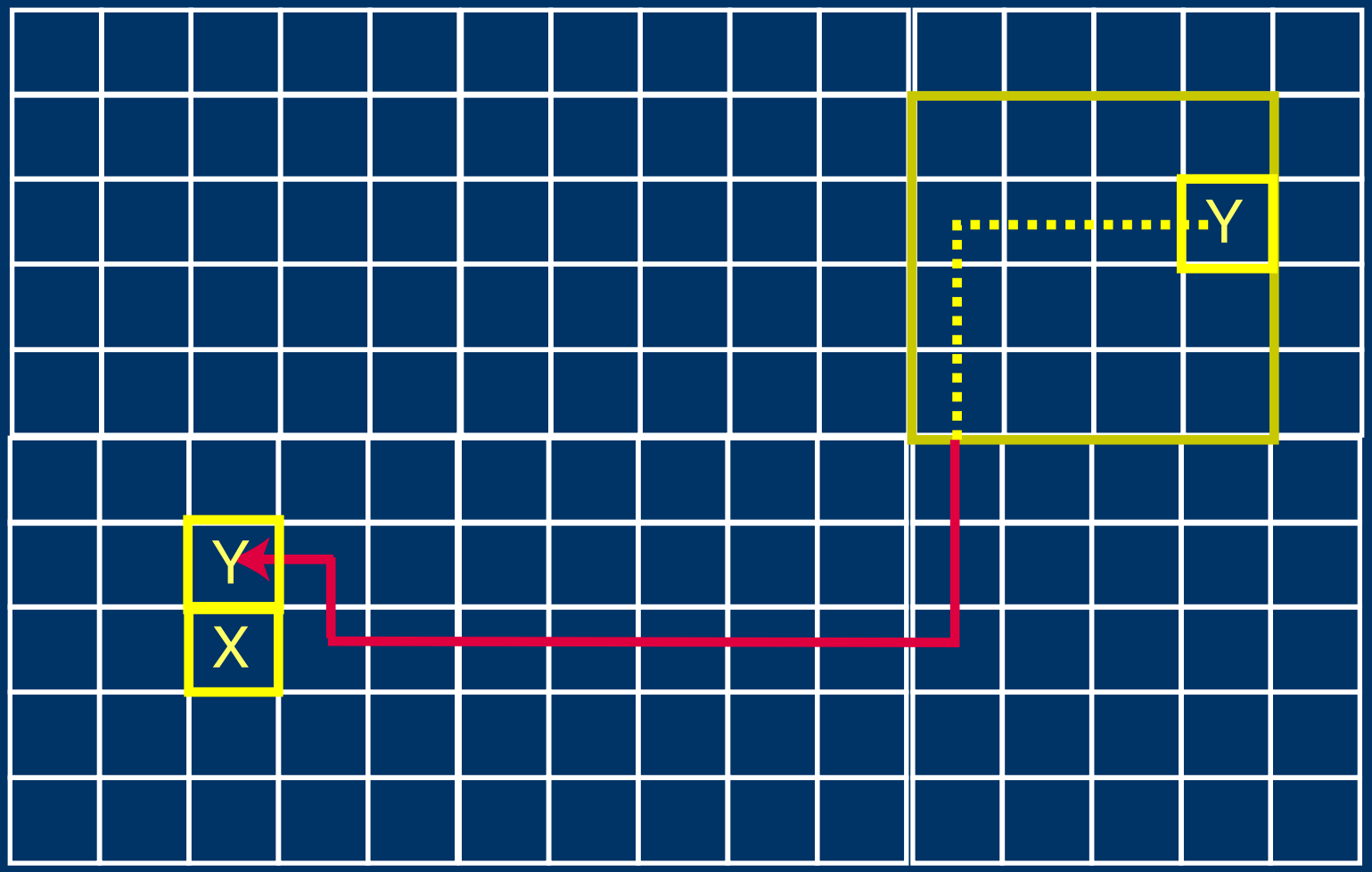
EXAMPLE: HARDWARE LIBRARY



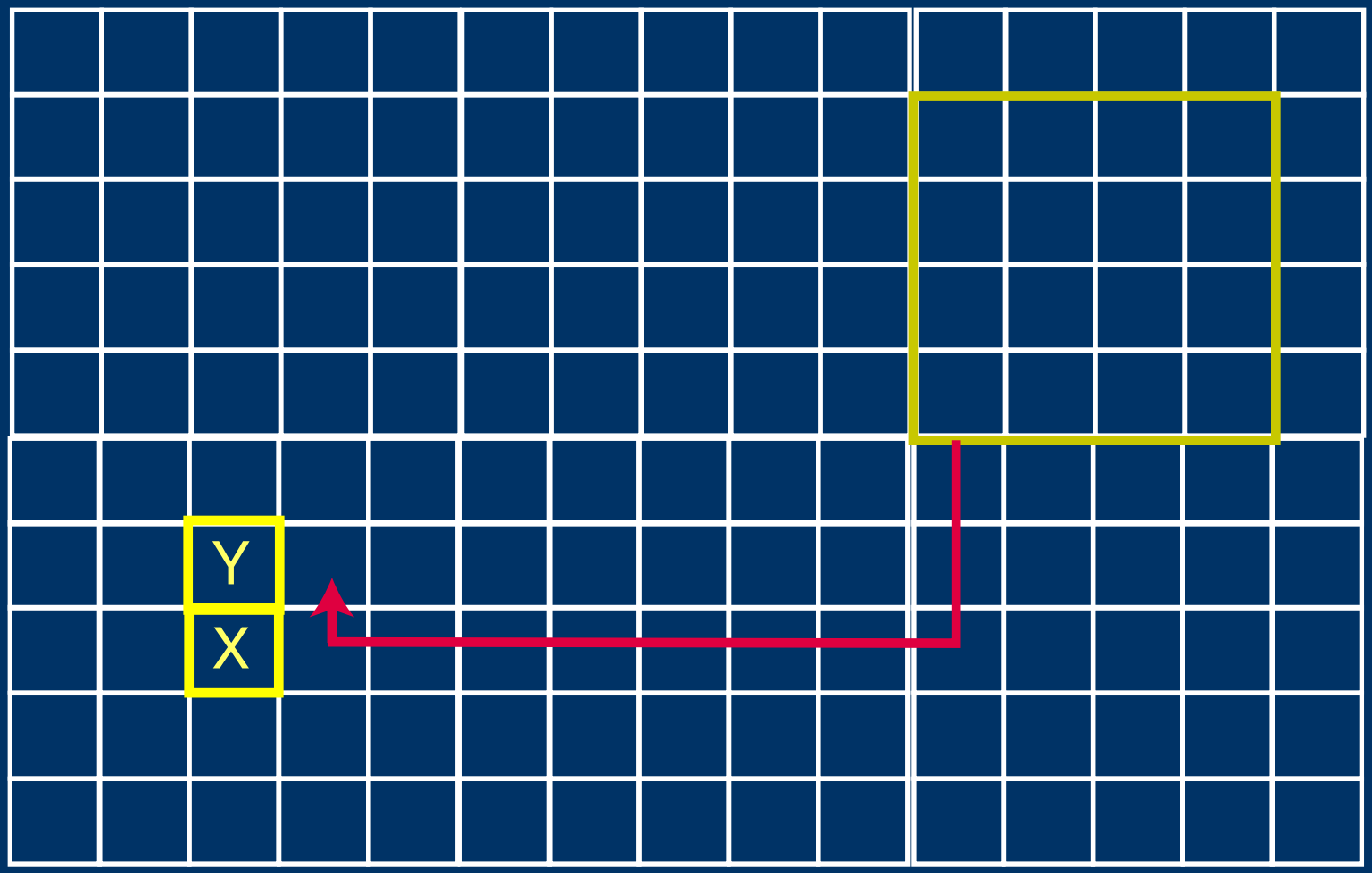
EXAMPLE: HARDWARE LIBRARY



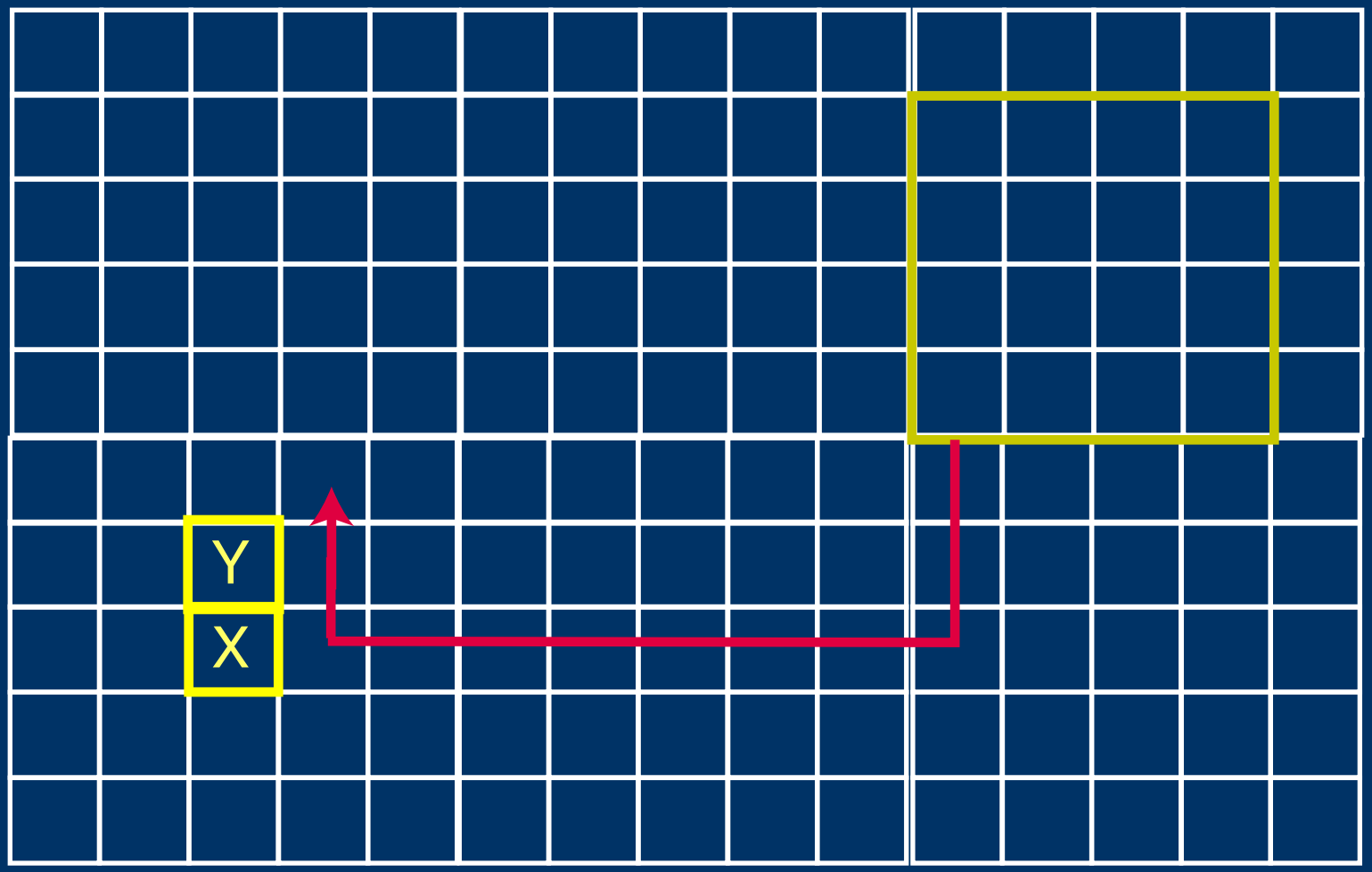
EXAMPLE: HARDWARE LIBRARY



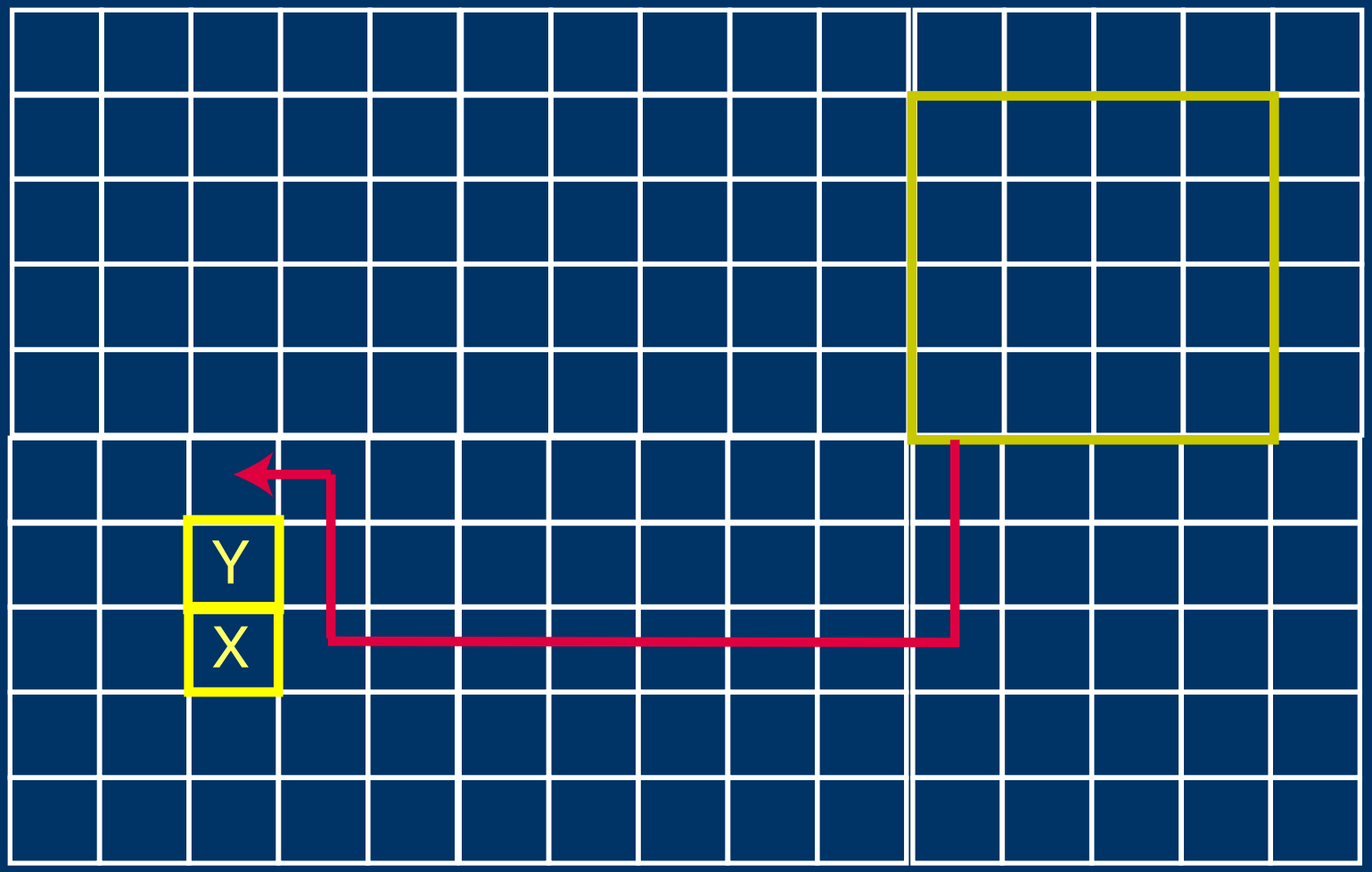
EXAMPLE: HARDWARE LIBRARY



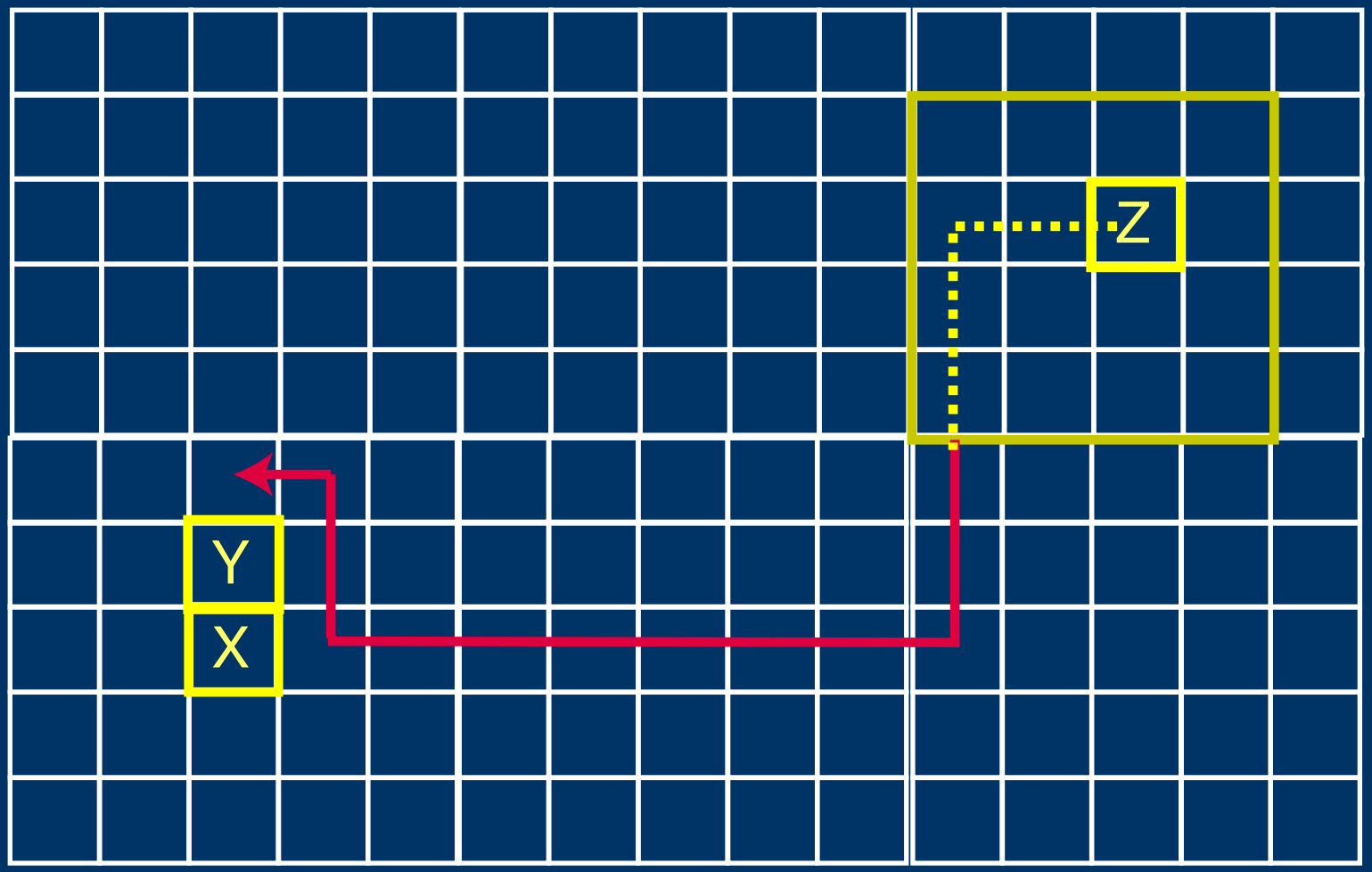
EXAMPLE: HARDWARE LIBRARY



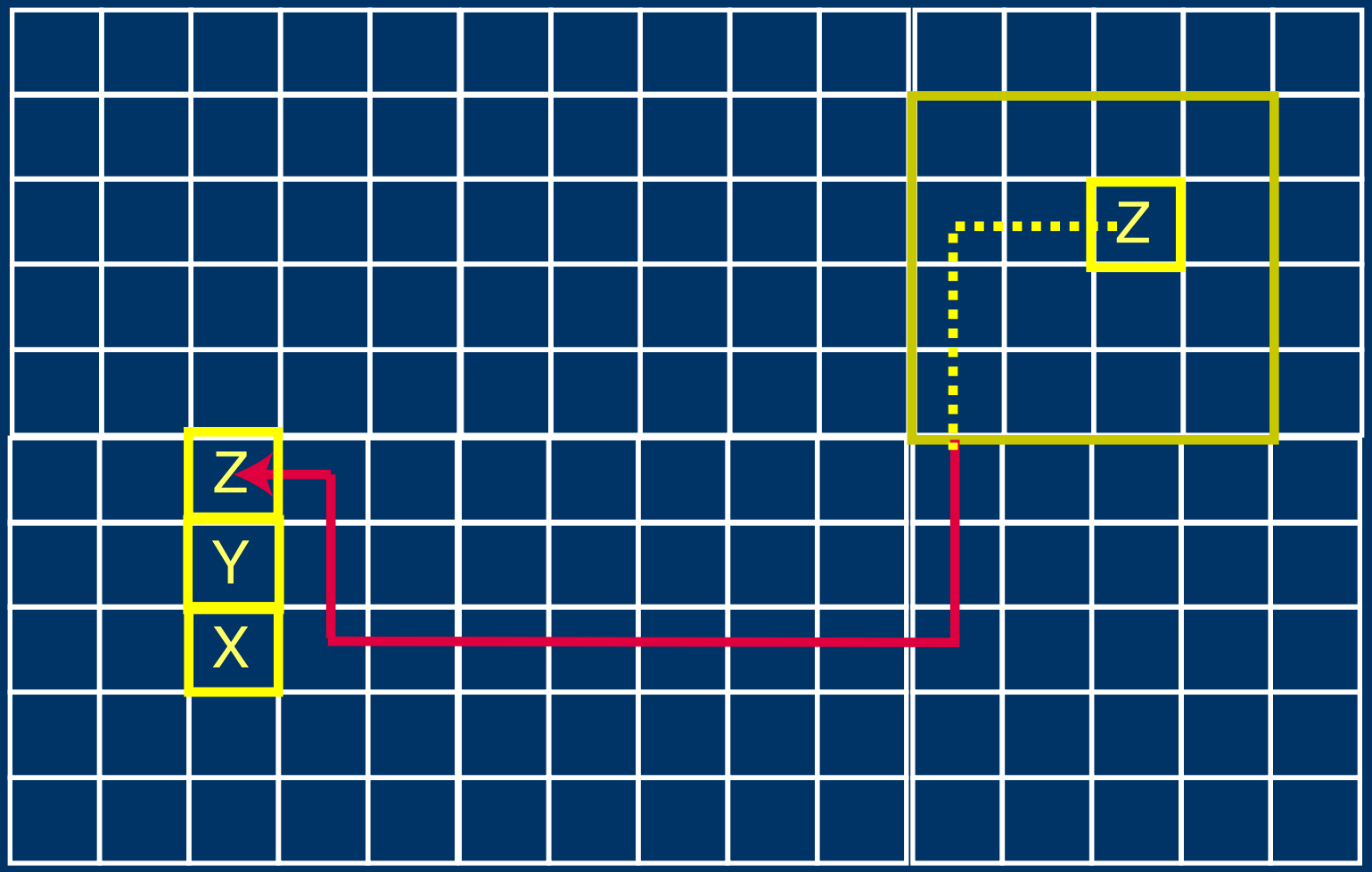
EXAMPLE: HARDWARE LIBRARY



EXAMPLE: HARDWARE LIBRARY



EXAMPLE: HARDWARE LIBRARY



OS CONCEPTS APPLIED TO HARDWARE

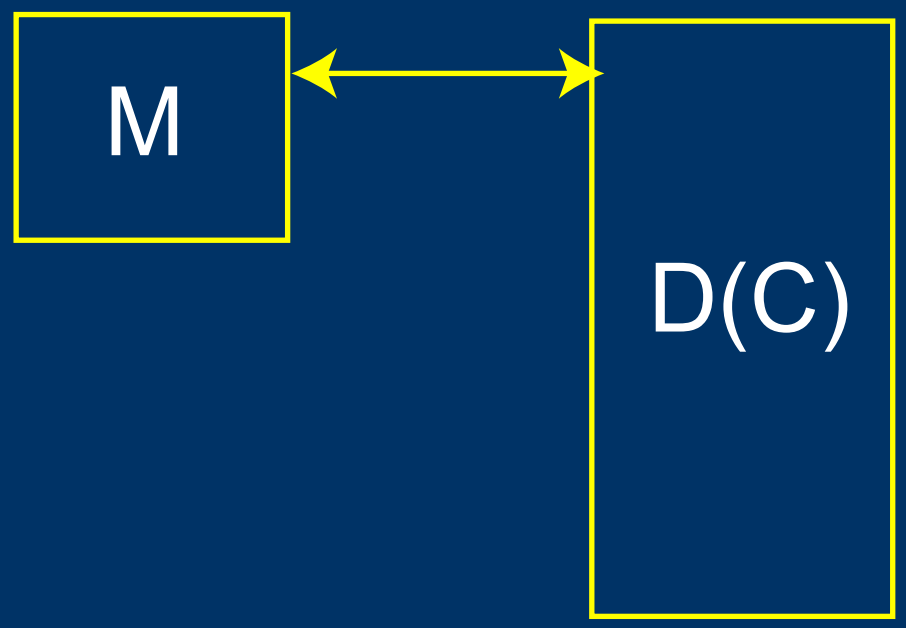
- Hardware Library
- Virtual Hardware Mapping
- Hardware Swapping

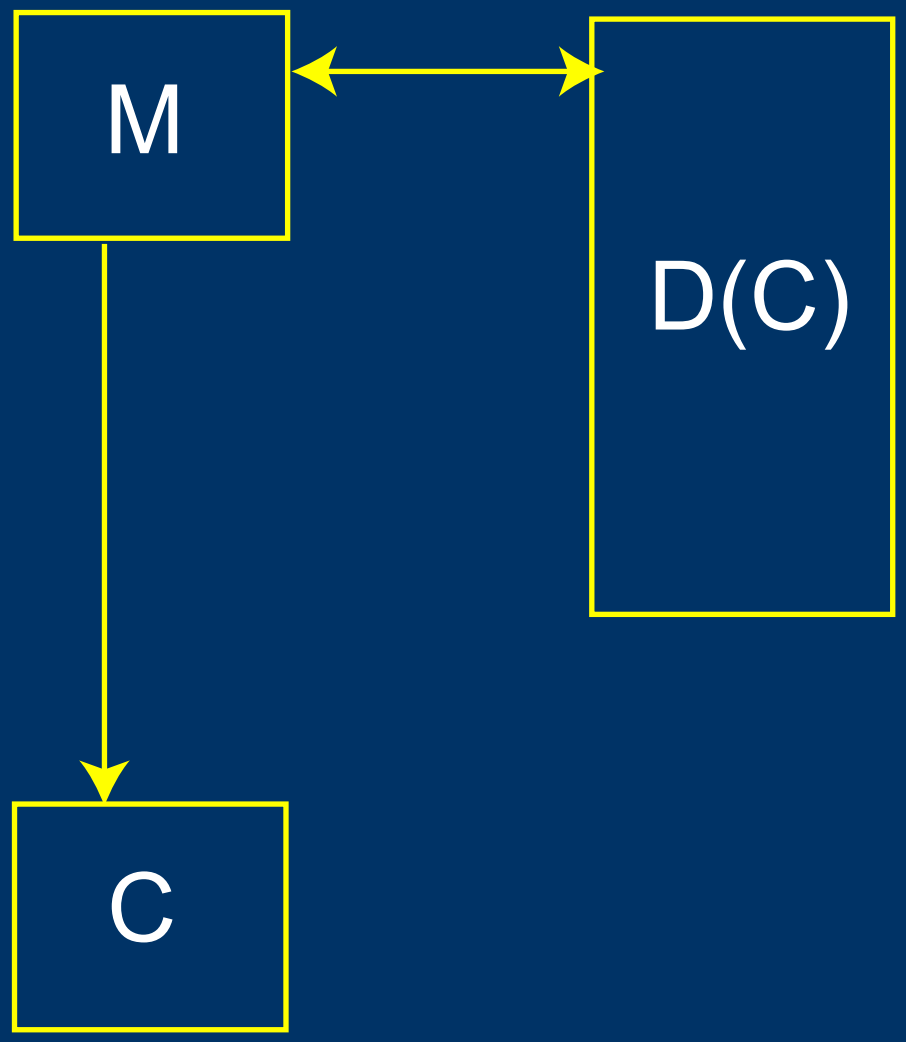
SELF-REPLICATION

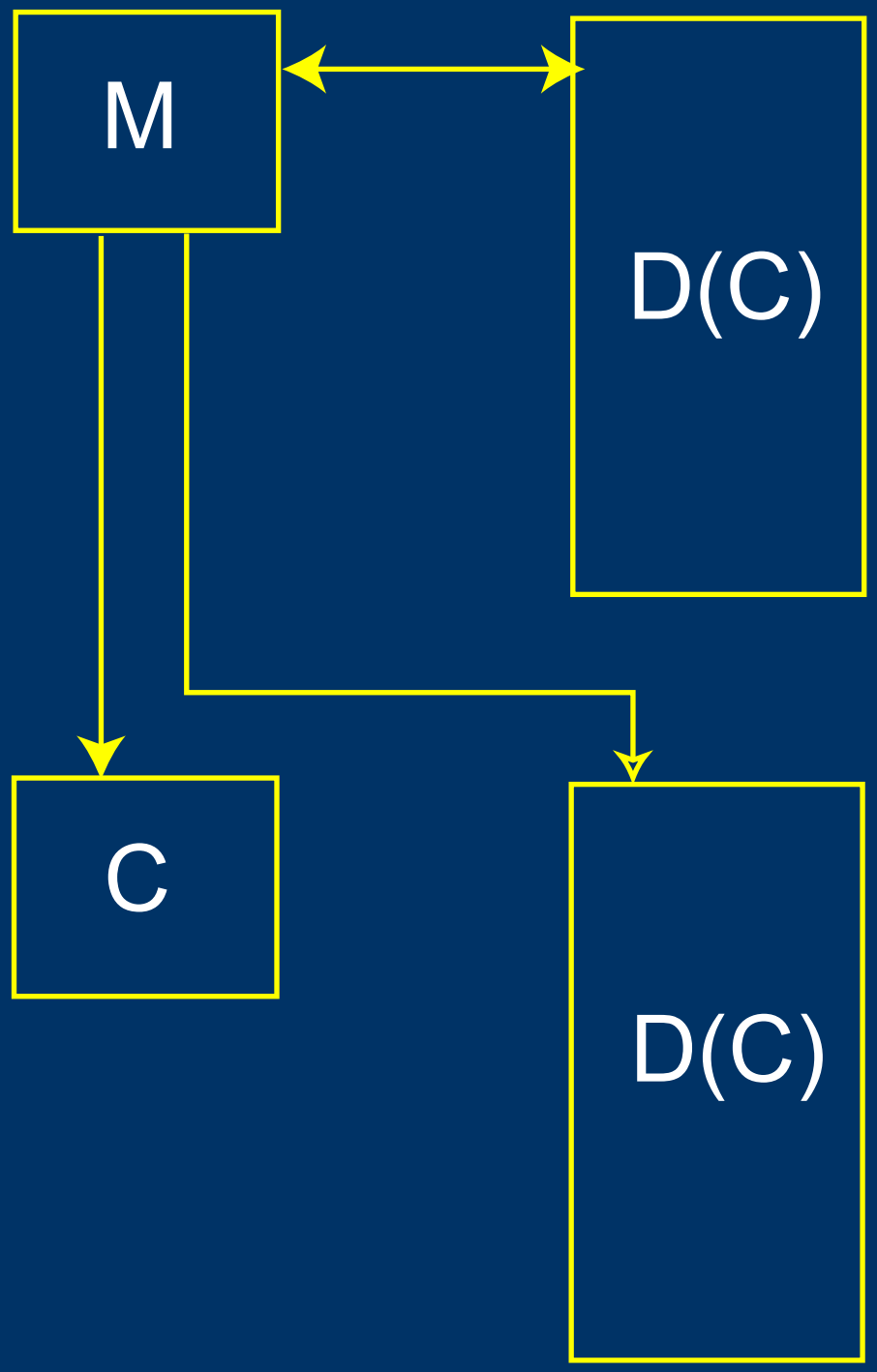
M

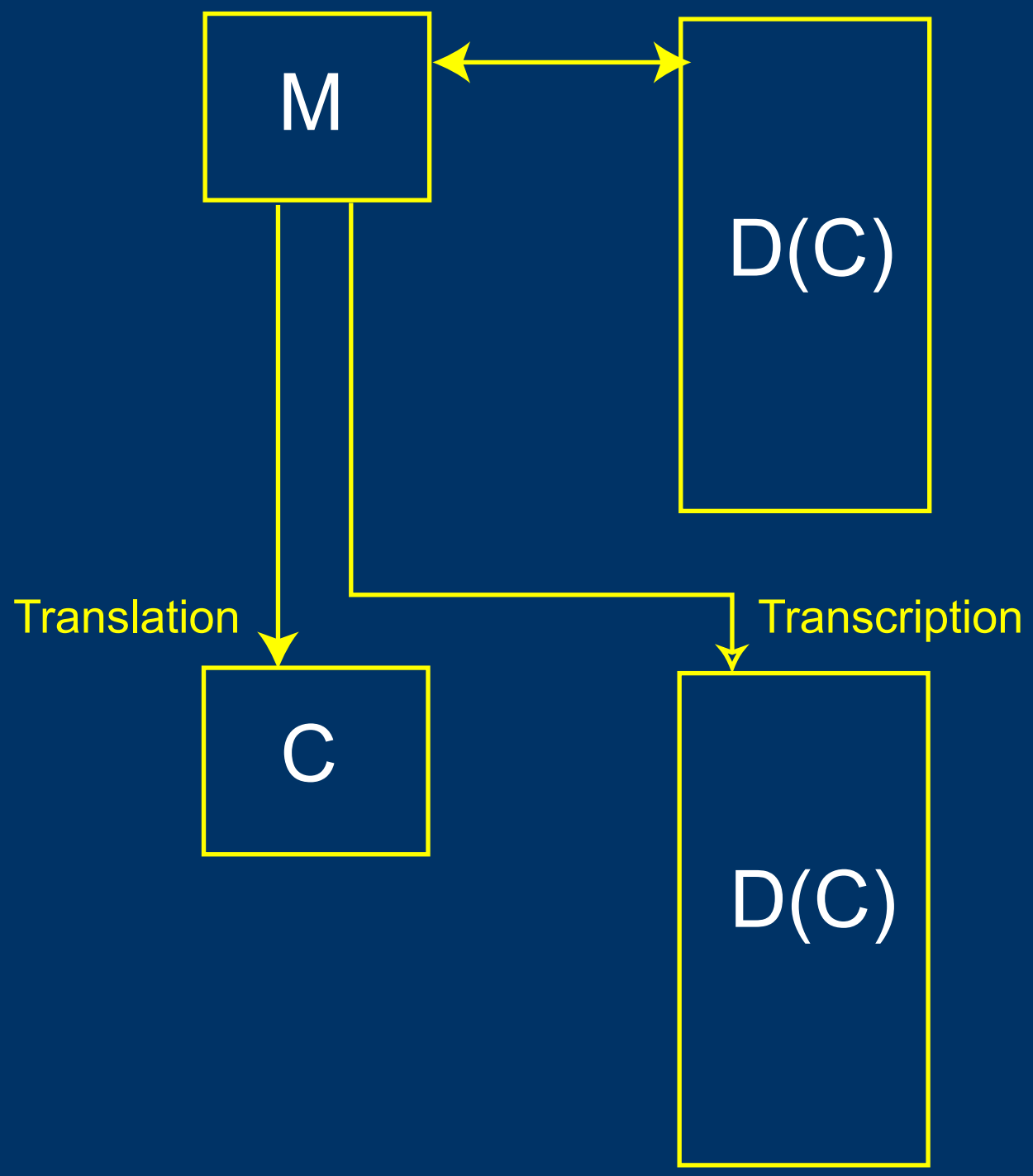
M

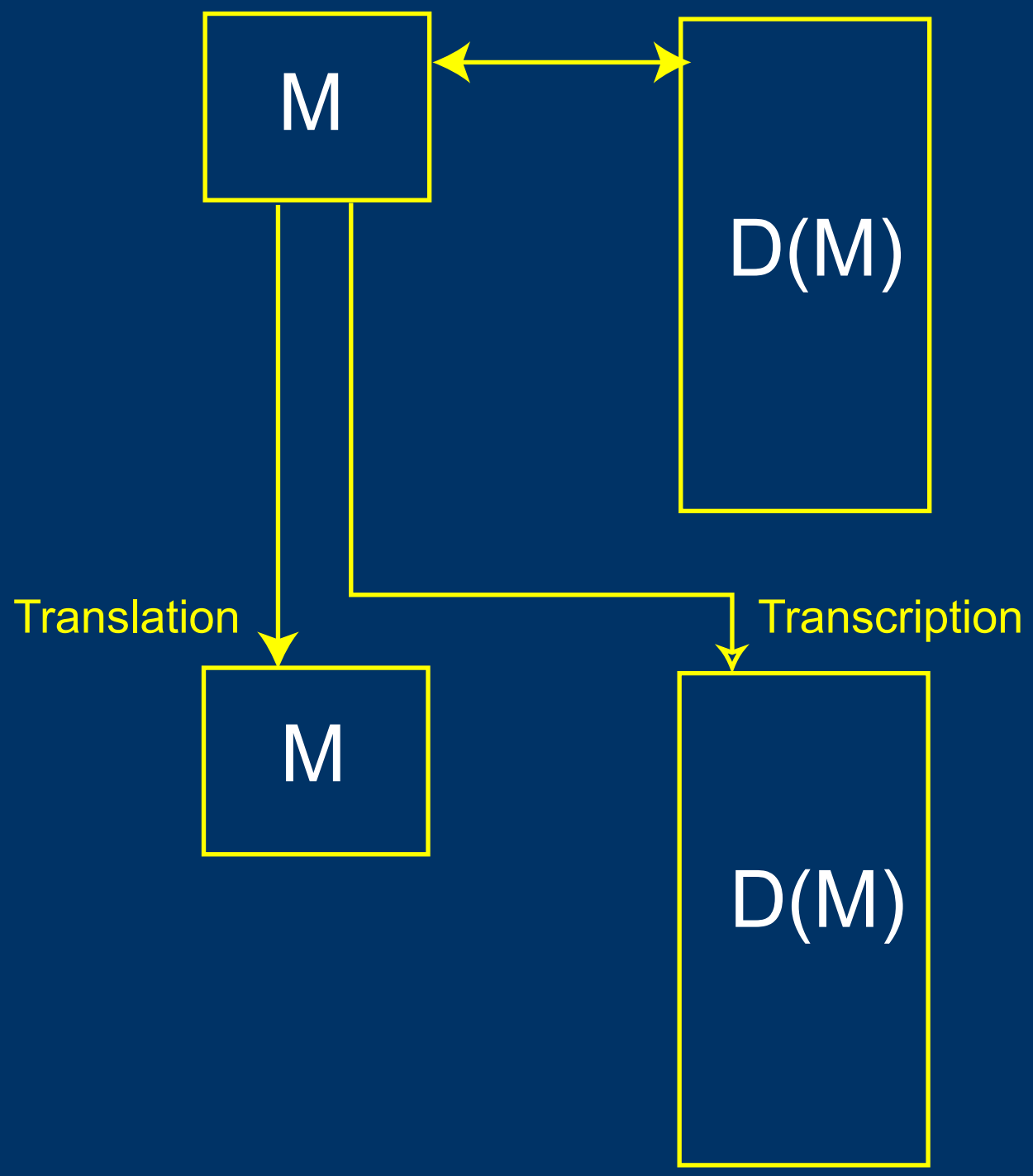
D(C)

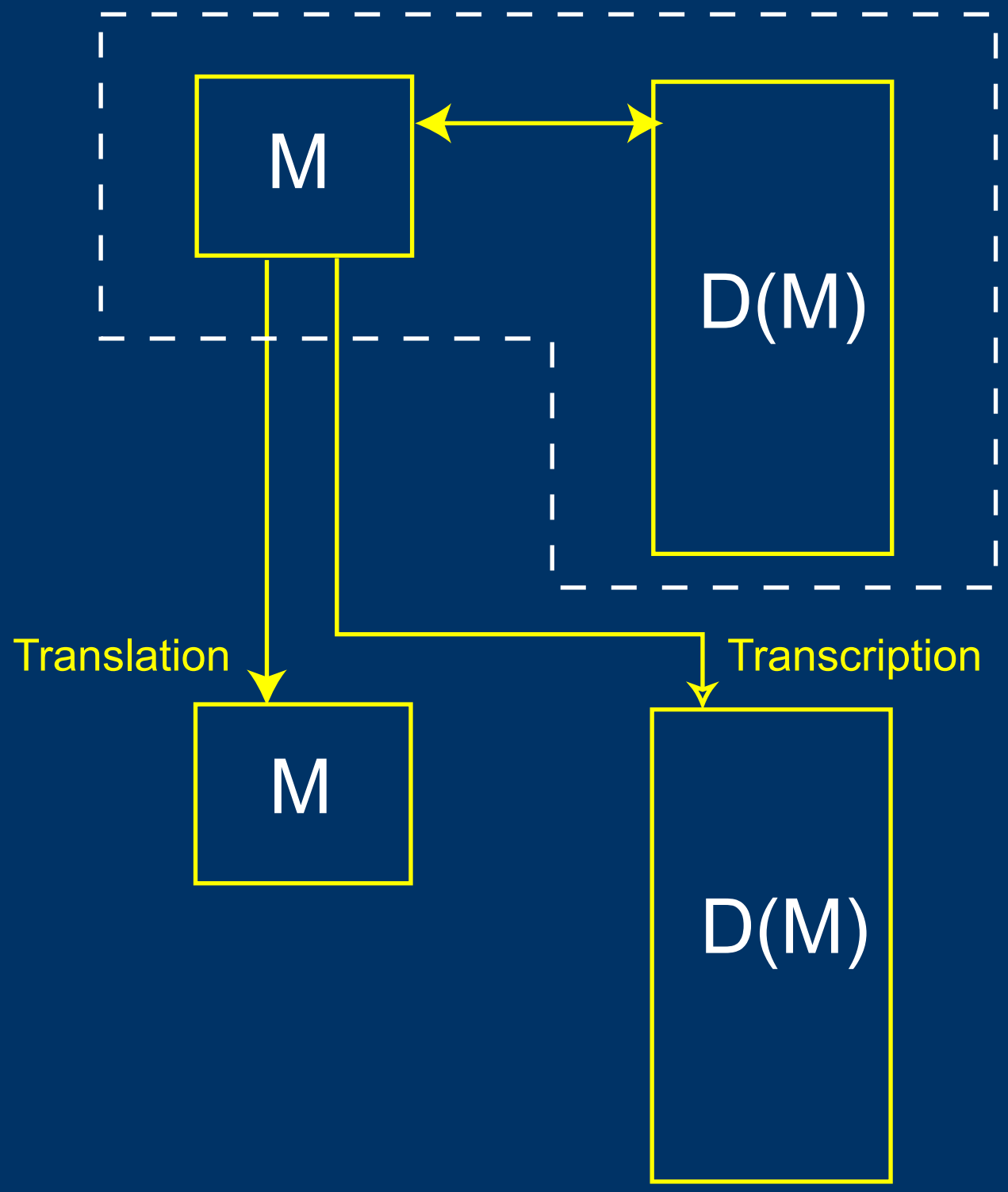

M
D(C)

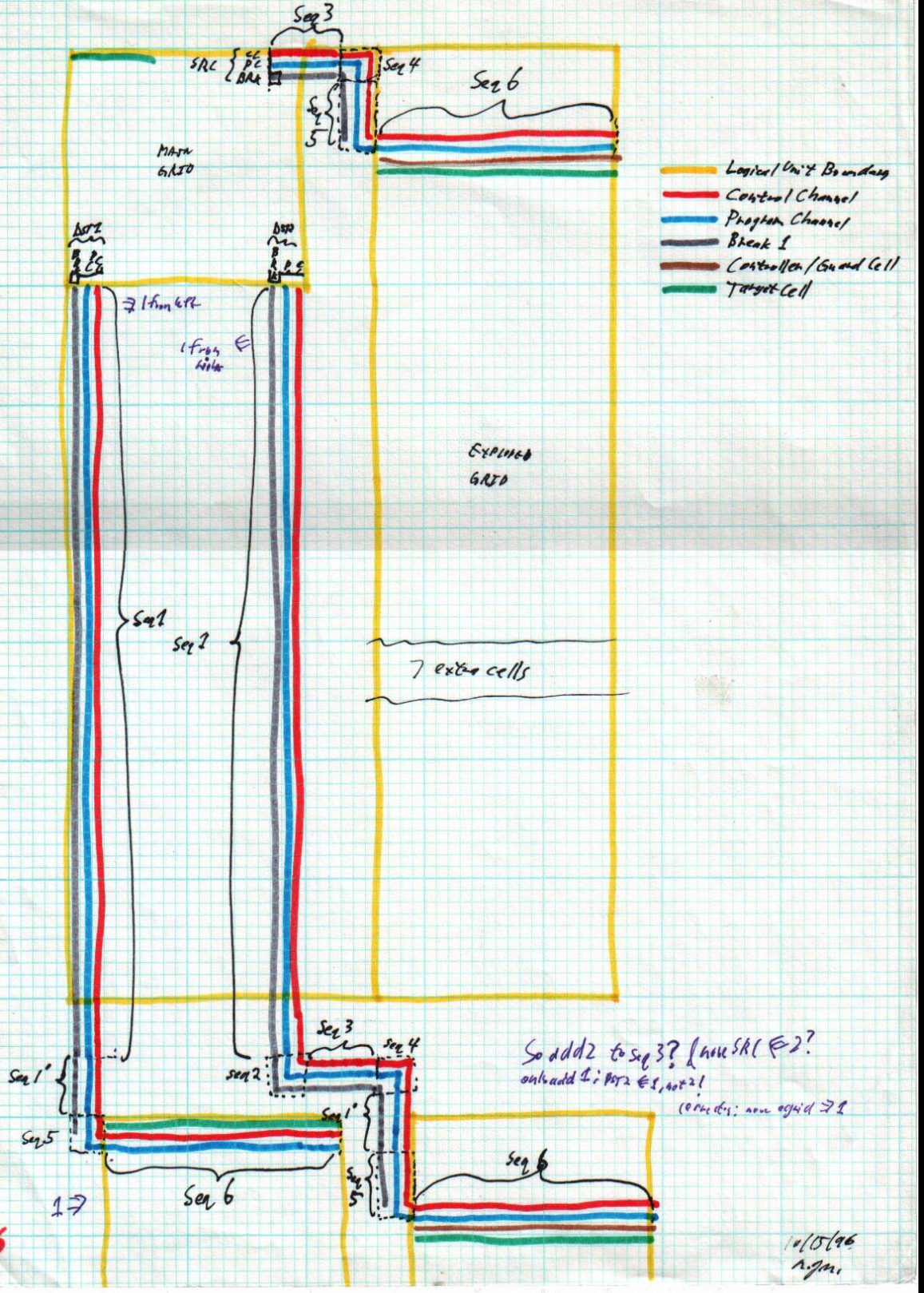

M
D(C)
C

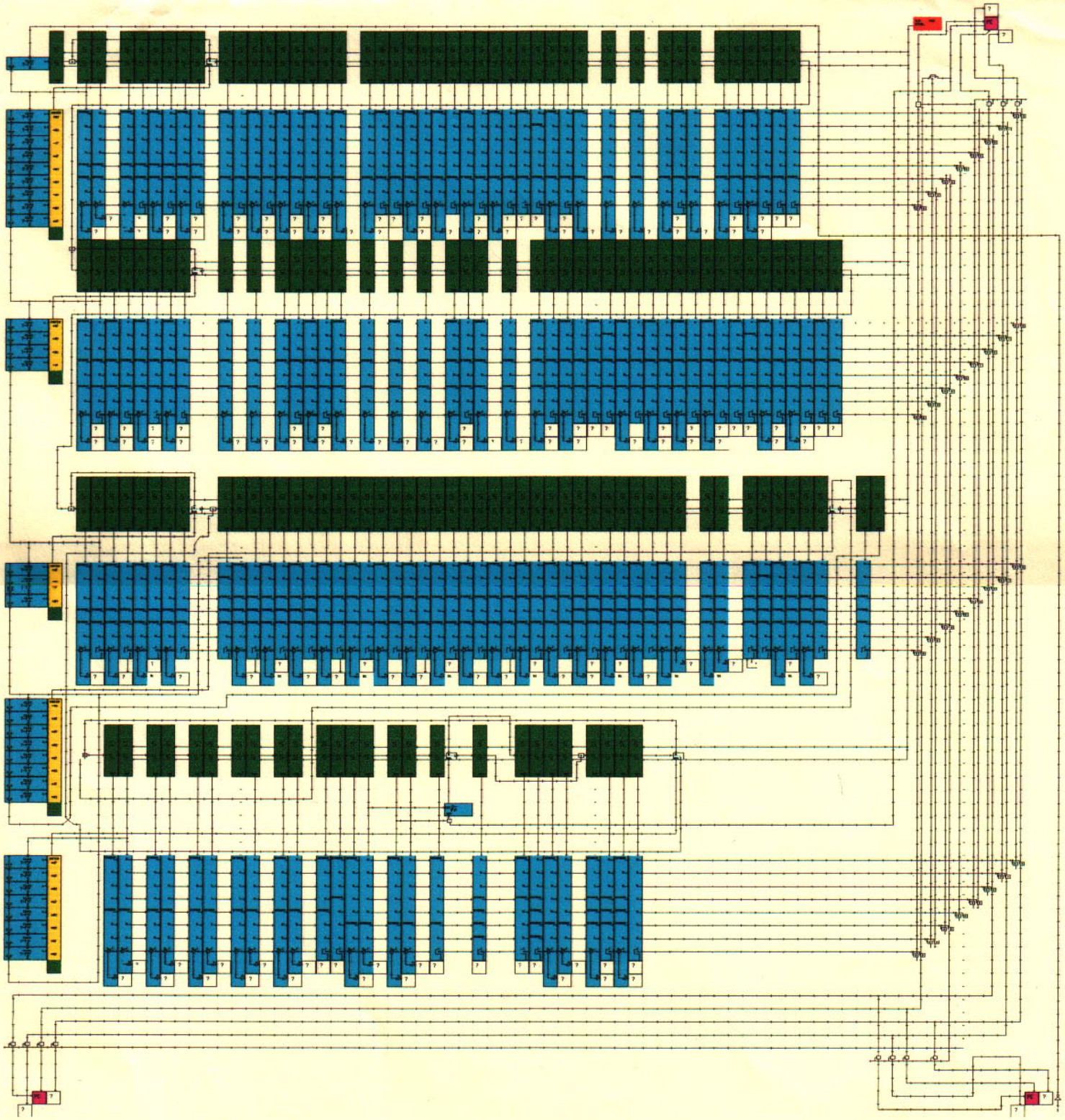

M
D(C)
C
D(C)


M
D(C)
Translation
Transcription
C
D(C)

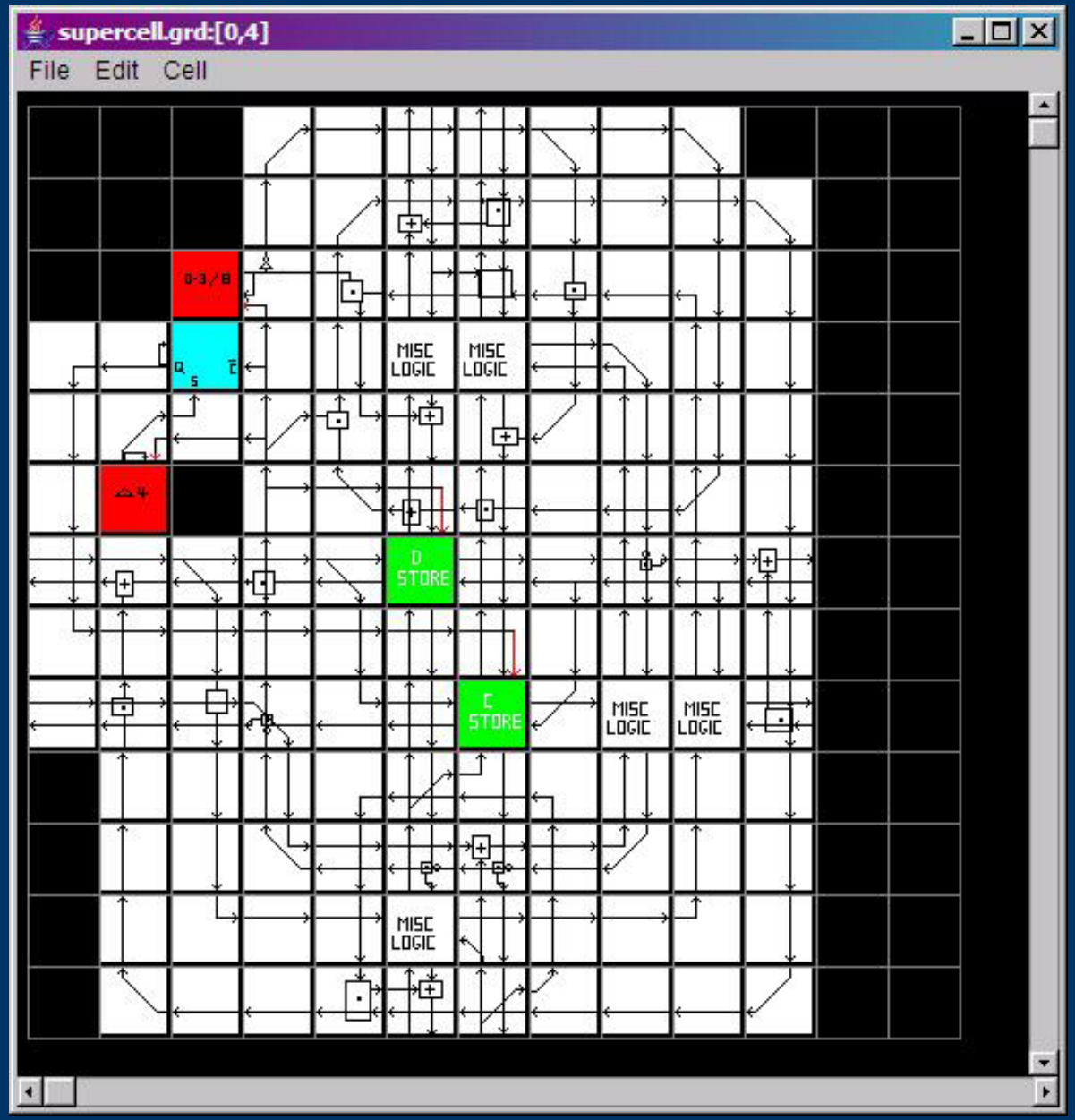

M
D(M)
Translation
Transcription
M
D(M)


M
D(M)
Translation
Transcription
M
D(M)


Seq 3
SRC
CC
PC
BRK
Seq 4
Seq 6
Seq 5
MAIN GRID
Logical Unit Boundary
Control Channel
Program Channel
Break 1
Controller / Guard Cell
Target Cell
DST2
DST1
1 from left
1 from right
EXPANDED GRID
Seq 1
Seq 2
7 extra cells
Seq 3
Seq 4
Seq 1'
Seq 2
Seq 1'
Seq 5
Seq 6
Seq 5
Seq 6
1 →
So add 2 to seq 3? & move SRC ← 2?
only add 1; DST2 ← 1, not 2!
(correction: now equal → 1
SRS6
10/15/96
A. Jan.



SUPER SELF-DUALITY



IV. IMPLICATIONS OF C-MODE

C-MODE HELPS WITH DETAILS

- Bootstrap time
  O(sqrt(n)): 10^{12} steps

C-MODE HELPS WITH DETAILS

- Bootstrap time
  O(sqrt(n)): 10^{12} steps
3-D: O(cuberoot(n)): 10^{8} steps

C-MODE HELPS WITH DETAILS

- Bootstrap time
  O(sqrt(n)): 10^{12} steps
3-D: O(cuberoot(n)): 10^{8} steps
1 picosec/step=100 uSEC

C-MODE HELPS WITH DETAILS

- Bootstrap time
- Faults
- Runtime management
  Can monitor locally, in parallel

C-MODE HELPS WITH DETAILS

- Bootstrap time
- Faults
- Runtime management
- Manufacturing

Can use the Cell Matrix to assist with the manufacturing process!

C-MODE HELPS WITH DETAILS

- Bootstrap time
- Faults
- Runtime management
- Manufacturing
- Propagation Delays

3-D: Max pathlength=300,000,000

C-MODE HELPS WITH DETAILS

- Bootstrap time
- Faults
- Runtime management
- Manufacturing
- Propagation Delays

3-D: Max pathlength=300,000,000

US Pat #6,577,159

ONE-PROBLEM ONE-MACHINE

- Co-compilation of HW and SW
- Can adjust HW at runtime too
- Library contains HW+SW
- General bluring of the line

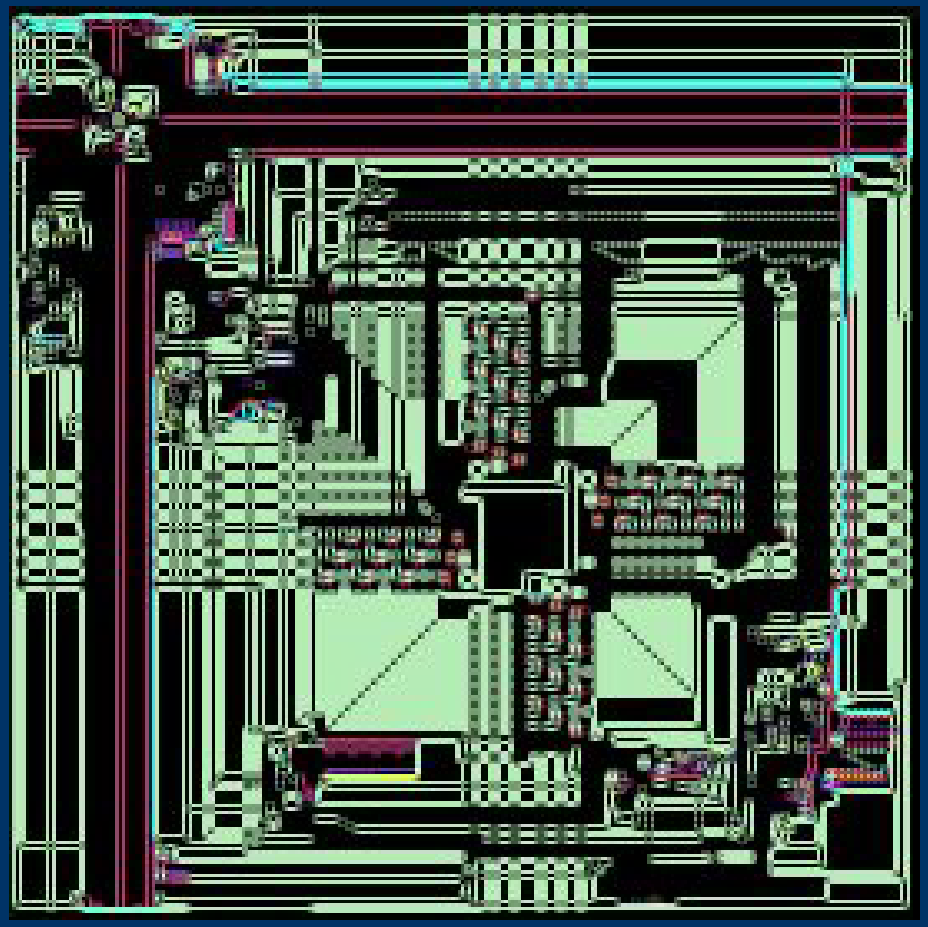
ONE-PROBLEM ONE-MACHINE

More-generally, don't just build
a co-processor:
Build the **entire** processor
+ data storage
+ I/O subsystem
+ ...

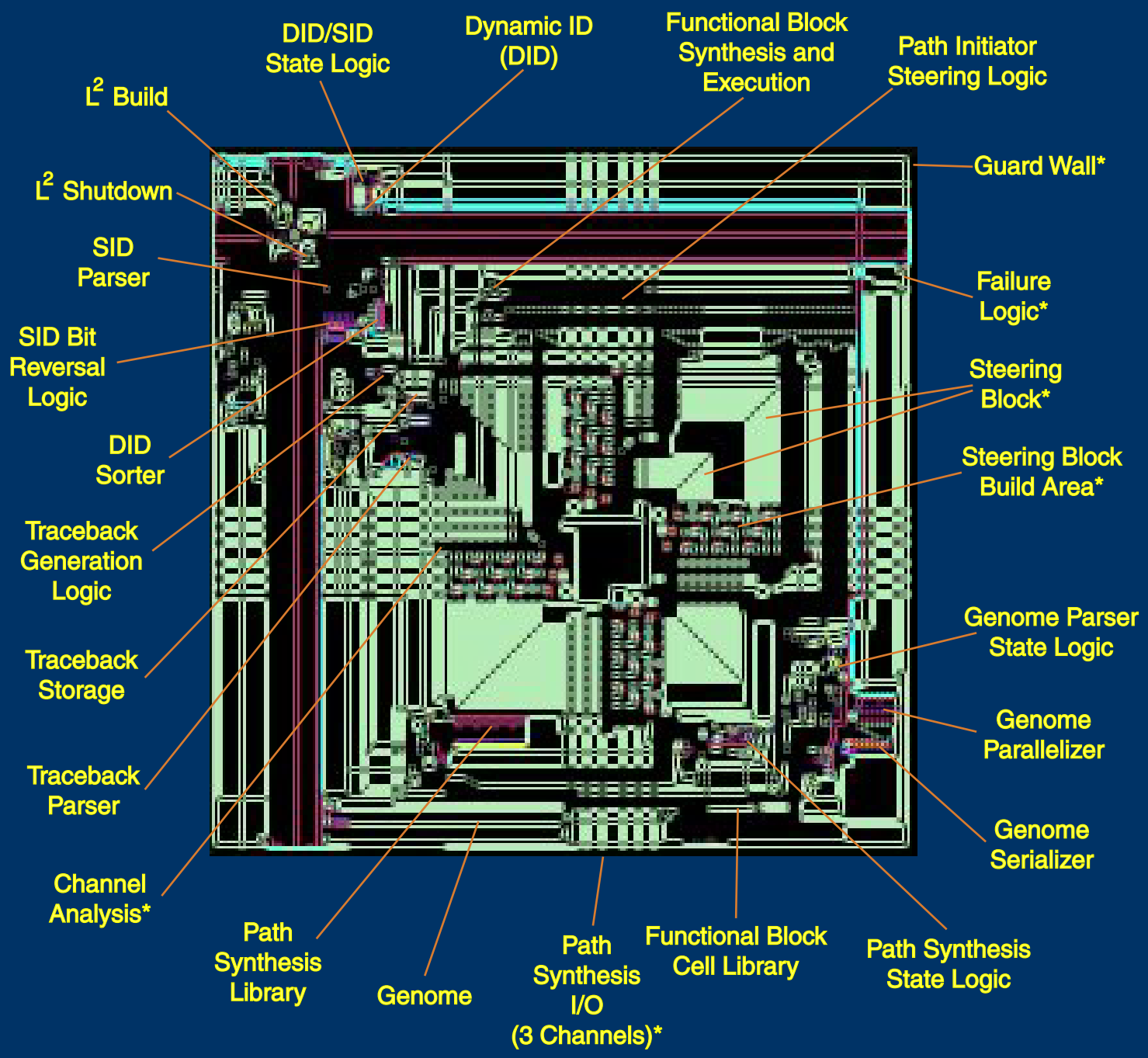
CELLS ARE JUST "STUFF"

One doesn't count brick atoms when building a house...

Supercell Image



72,900 Cell Matrix Cells
270x270


DID/SID State Logic
Dynamic ID (DID)
Functional Block Synthesis and Execution
Path Initiator Steering Logic
L2 Build
Guard Wall*
L2 Shutdown
SID Parser
Failure Logic*
SID Bit Reversal Logic
Steering Block*
DID Sorter
Steering Block Build Area*
Traceback Generation Logic
Genome Parser State Logic
Traceback Storage
Genome Parallelizer
Traceback Parser
Genome Serializer
Channel Analysis*
Path Synthesis Library
Genome
Path Synthesis I/O (3 Channels)*
Functional Block Cell Library
Path Synthesis State Logic

V. THE FUTURE

MUTUAL BENEFITS WITH OTHER TECHNOLOGIES

- Can take advantage of nanotechnology but could also help with atomic-scale assembly (local analysis, correction, etc.)
- Same with 3-D fabrication

ANALOG CELL MATRIX

- Continuous-valued inputs/outputs
- Configurable discrete components (caps/res/op amps/???)
- "Truth Table" stores component values

28 Jan 2003

Definition: A cell of a continuous Cell Matrix of finite order n with integral programming cycles is a set

$$\mathcal{C} = \{D_i, C_i, \mathcal{M}, \Pi, CS, J\}$$

consisting of:

- A set of D-inputs $\{D_i(t): \mathbb{C} \to \mathbb{C},\ 0 \le i \le 2n-1\}$;
- A set of C-inputs $\{C_i(t): \mathbb{C} \to \mathbb{C},\ 0 \le i \le 2n-1\}$;
- An injective mapping $\Pi: [0, 2\pi) \to \mathbb{C}^n$, surjective on the cross product of the domains of $\{D_i(t): 0 \le i \le 2n-1\}$;
- A set of C-states $CS \subseteq \mathbb{C}$;
- A binary ~~Dilmmatic~~ operator $J: \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, and with $J(x,0) = J(0,x) = x$

$\Rightarrow$

28 Jan 2003

Definition (continued):

- A map $\mathcal{M}: \mathbb{C} \times \mathbb{C}^n \to \mathbb{C}^{2n}$ defined by

$$\mathcal{M}: (t, d_0, d_1, \dots, d_{n-1}) \mapsto (\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{2n-1})$$

where $\mathcal{O}_i = \mathcal{OD}_i \cdot DM_{(t)} + \mathcal{OC}_i\,(1 - DM_{(t)})$,

$$DM_{(t)} = \prod_{i=0}^{n-1} (1 - \hat{C}_i(t)),$$

$$\hat{C}_i(t) = \begin{cases} 0 & \text{if } C_i(t) \notin CS \\ 1 & \text{if } C_i(t) \in CS \end{cases},$$

$$\mathcal{OD}_i = \sum_{k=0}^{n-1} \left[D_k(T_2(t) - 2n \cdot 2\pi + i + \tfrac{k}{2\pi i}) \cdot \hat{C}_k(T_2(t) - \tfrac{4n\pi}{n+1} + 2\pi i) \right]$$

$$\mathcal{OC}_i = \begin{cases} \hat{C}_i(t) \cdot \sum_{k=0}^{n-1} \left[D_k((t - T_1(t)) + (T_2(t) - 2n \cdot 2\pi)) \cdot \hat{C}_k((t - T_1(t)) + (T_2(t) - 2n \cdot 2\pi)) \right] & \text{if } t - 2n \cdot 2\pi < T_1(t) \\ \hat{C}_i(t) \cdot \sum_{k=0}^{n-1} \left[D_k(t - 2n \cdot 2\pi) \cdot \hat{C}_k(t - 2n \cdot 2\pi) \right] & \text{if } t - 2n \cdot 2\pi \ge T_1(t) \end{cases}$$

28 Jan 2003

Definition (concluded):

$$T_1(t) = \max\{x < t : x \in CMT_{(t)}\},$$

$$T_2(t) = \max\{x < T_1(t) : x \in CMT_{(t)}\},$$

$$CMT_{(t)} = \{x \le t : DM(t) = 0\},$$

and $j = \Pi^{-1}(d_0, d_1, \dots, d_{n-1})$.

CONTINUOUS CELL MATRIX

- Continuous-valued inputs/outputs
- Eliminate discrete truth table
- Store Input->Output mapping as a continuous-valued function

Amplifier: 1-Cell
FM Gen: 2-Cells

NOT QUITE PERFECT...

- Cell Matrix is Physical
- Configuration is Informatic
- Still Self Contemplating Other

PERFECT SELF DUALITY

Cell Matrix Configuration of
own Physical Structure

SELF ASSEMBLY

- Set of primitive building blocks (synthetic molecules)
- Molecular linking system
- Sensors/Actuators on Cell I/O
- Define configuration sequences for assembling a cell
- Use already-built cells to build new cells (self-duality)

ARTIFICIAL MATTER

- Standard configurable building block

ARTIFICIAL MATTER

- Standard configurable building block
- Local configuration via Cell Matrix

ARTIFICIAL MATTER

- Standard configurable building block
- Local configuration via Cell Matrix
- Don't cut down trees-just configure wood
- Don't nail boards-configure a joint
- Detect breaking point and let go

ENVIRONMENTAL IMPACT

- Single resource use
- Waste minimization
- Controlled disassembly
- Cradle to Cradle

FOR MORE INFO

Cell Matrix Corporation

www.cellmatrix.com

cmsupport@cellmatrix.com

(877) 473-0882