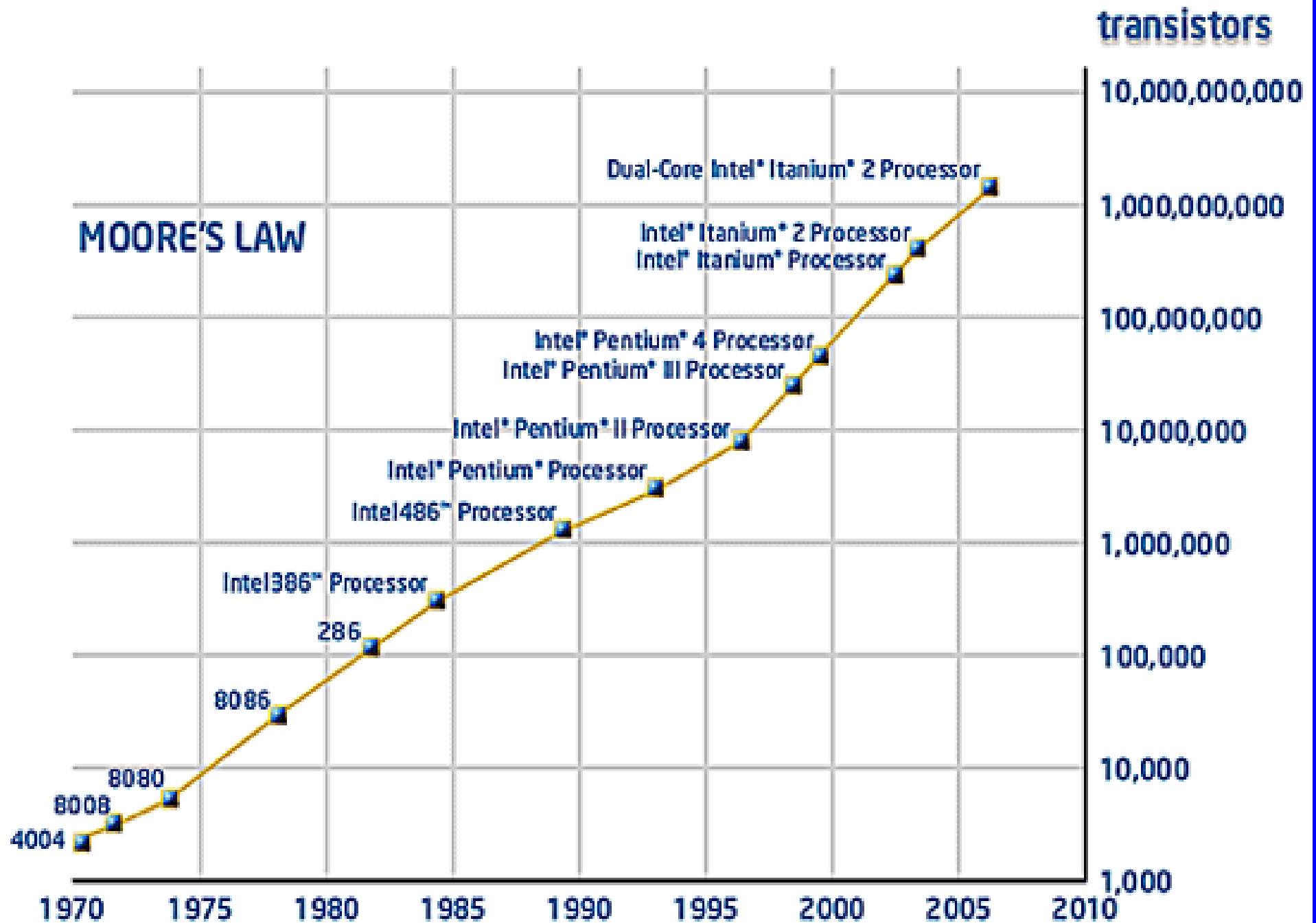




Music, Nirvana and Moore's Law

Nicholas J. Macias
24 May 2011

MOORE'S LAW



Avogadro's Number

$$A_n = 6.02 \times 10^{24}$$

of atoms in 12g of carbon (“1 mole”)

1 mole of any element contains 6.02×10^{24}
atoms

A Mole of Transistors (or Cores/CPUs/Computing Elements)

- 10^{24} = one trillion trillion (nice round number)

Basic Research Question:

How do you use one trillion trillion computing elements?

A Mole of Transistors (or Cores/CPUs/Computing Elements)

- 10^{24} = one trillion trillion (nice round number)

Basic Research Question:

How do you use one trillion trillion computing elements?

One answer: Make a Cell Matrix

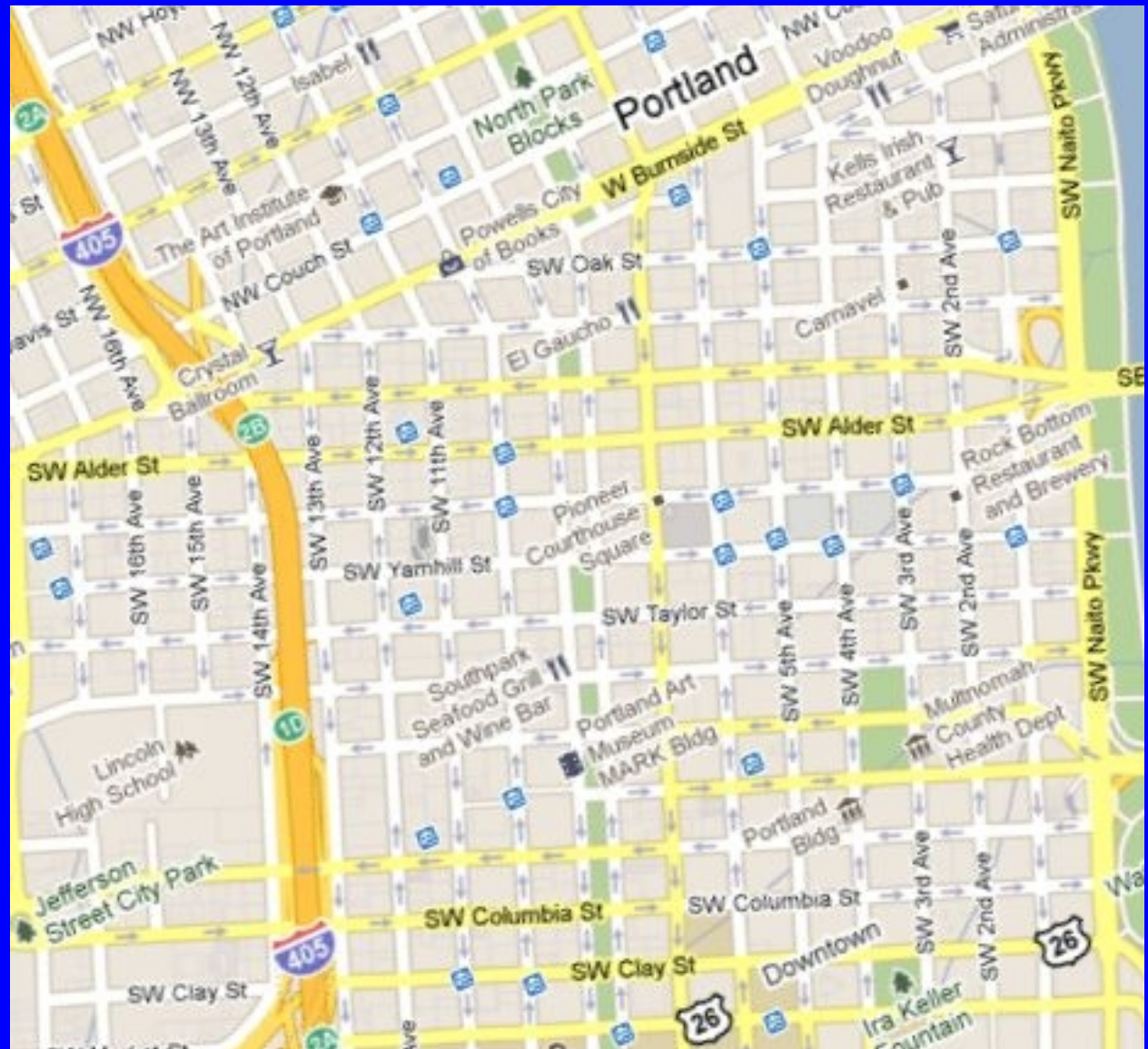
DENSITY

- Assume 1nm x 1nm transistors
- 10^{24} transistors

DENSITY

- Assume 1nm x 1nm transistors
 - 10^{24} transistors
- 

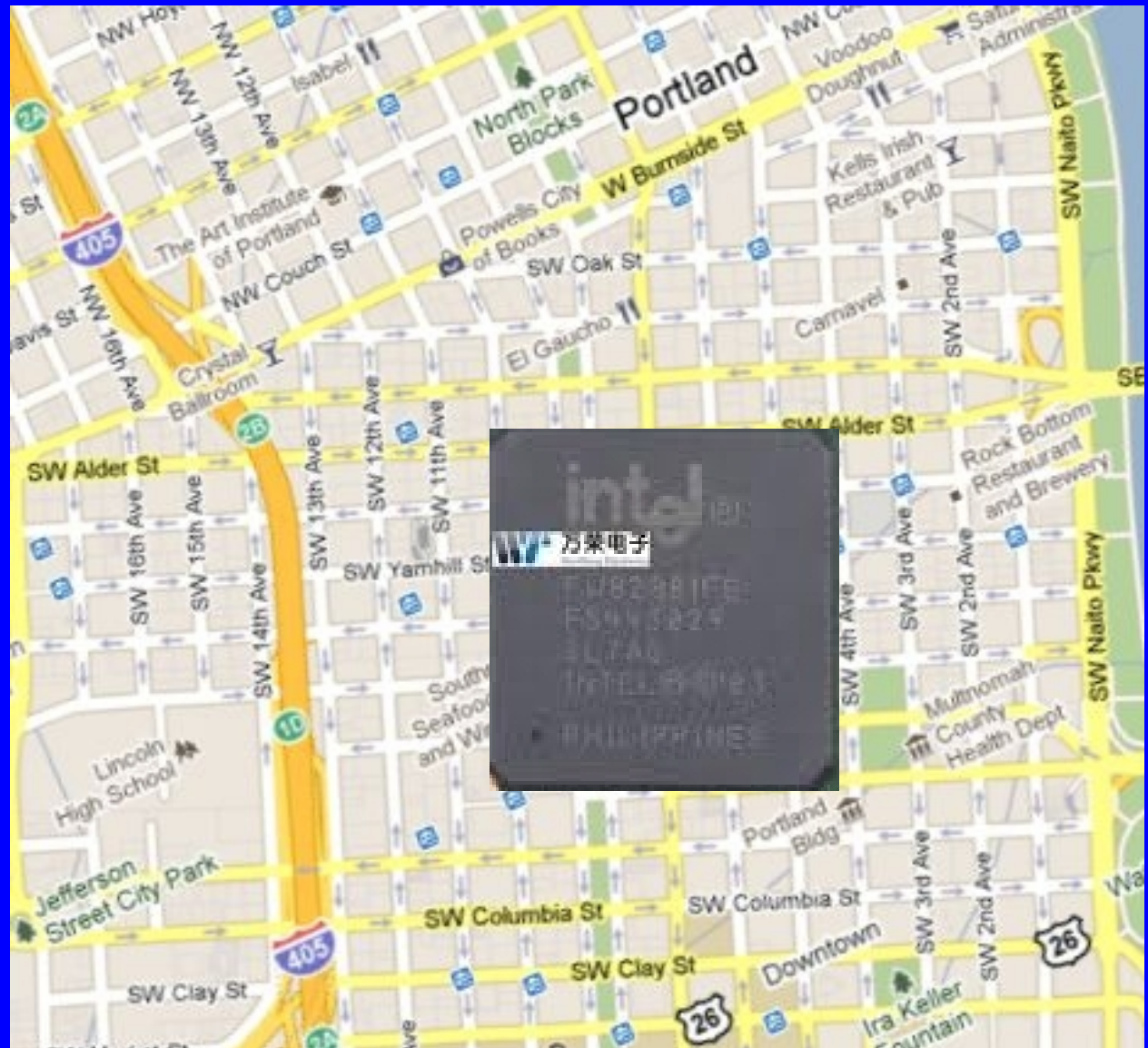
area=1 km²



DENSITY

- Assume 1nm x 1nm transistors
 - 10^{24} transistors
- 

area=1 km²



DENSITY (3-D)

- Assume 1nm x 1nm x 1nm transistors
- 10^{24} transistors

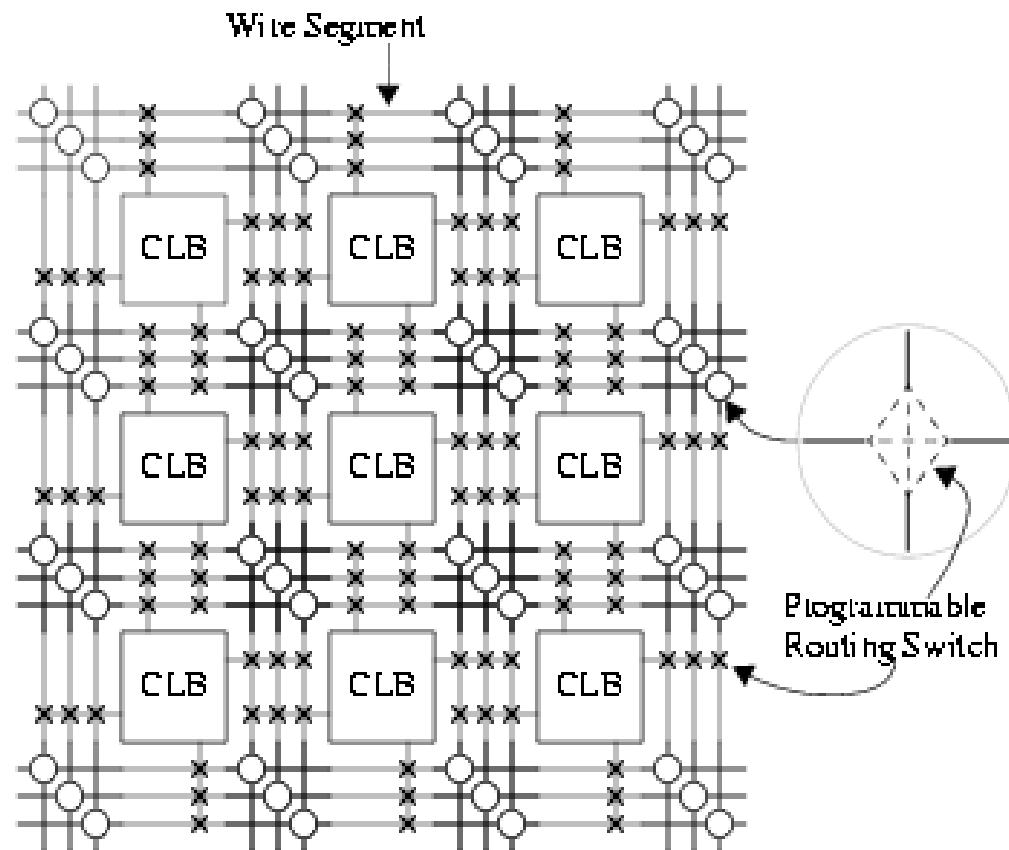
volume=1,000 cc



May want to mostly avoid custom ASIC

- Huge NRE charges (time and \$)
- Validation difficult
- Single-function

Field Programmable Gate Array (FPGA)



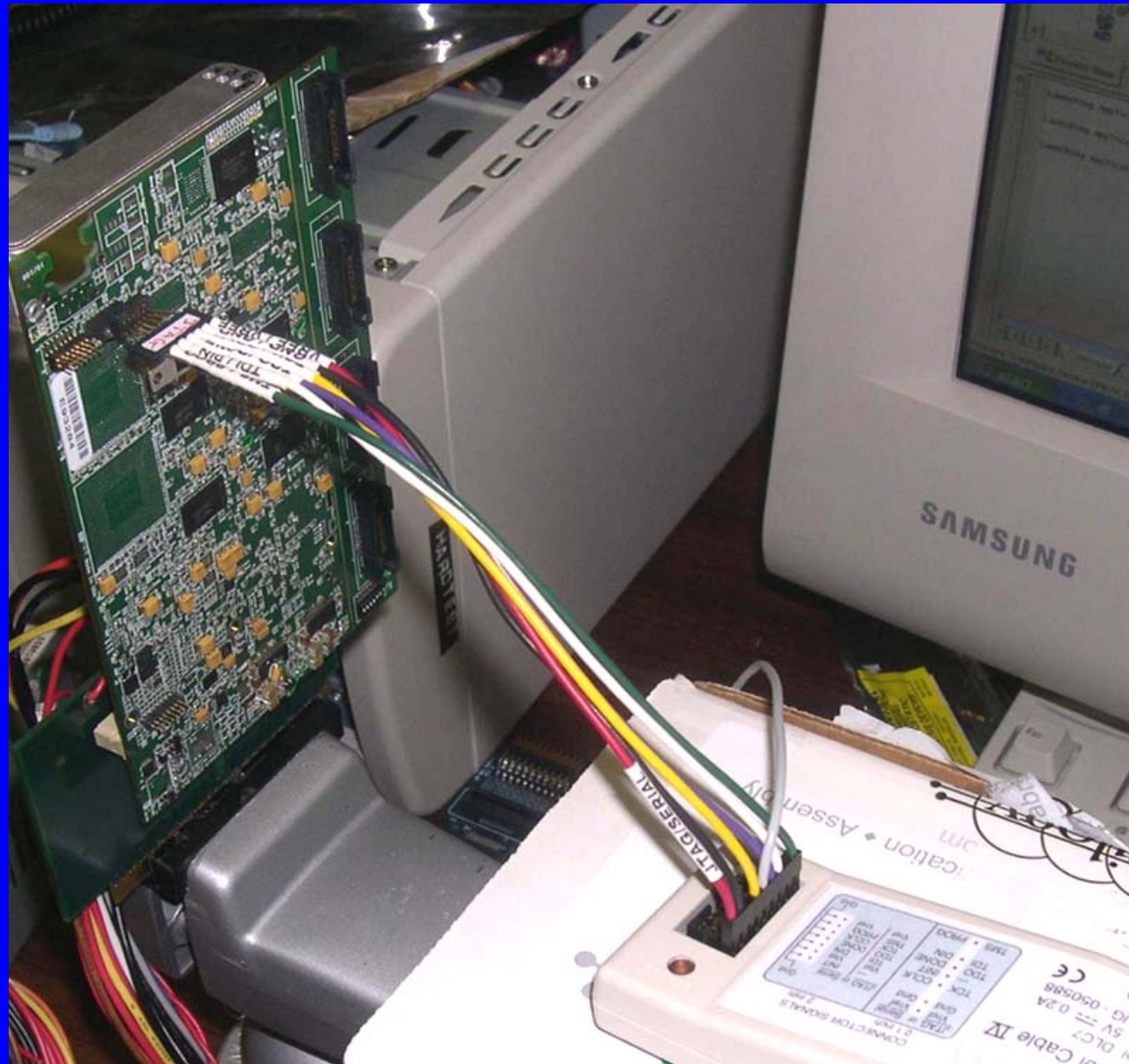
Reconfigurable Logic - GOOD!

- Design once, use for many circuits
- Lower NRE costs
- Faster time to market
- More flexible once in the field
- Particularly beneficial if architecture is scalable

Hypothesis About Future Computing

1. Based on increasingly-large (scalable) reconfigurable devices

How We Use FPGAs Today

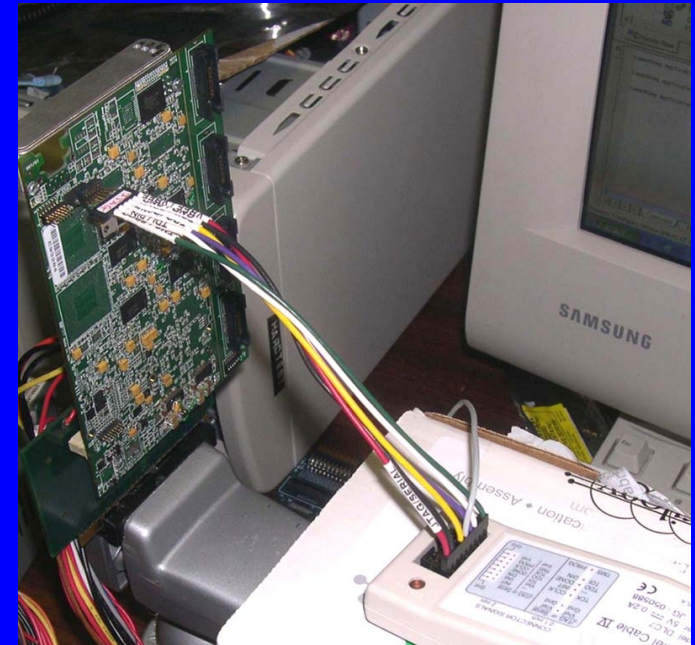


How We Might Use FPGAs Tomorrow...

- Millions of distinct circuits
- Dynamic behavior – register adjustment etc.
- On-the-fly reconfiguration
- Relocation of circuits
- Testing of underlying hardware
- and so on...

How We Might Use FPGAs Tomorrow...

- Millions of distinct circuits
- Dynamic behavior – register adjustment etc.
- On-the-fly reconfiguration
- Relocation of circuits
- Testing of underlying hardware
- and so on...



Hypothesis About Future Computing

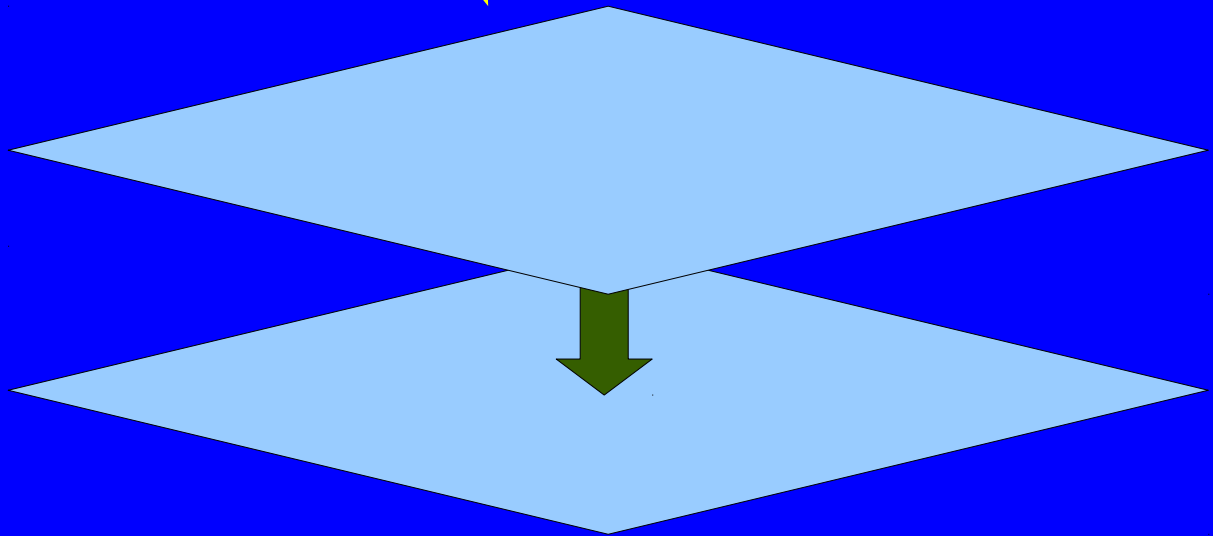
1. Based on increasingly-large (scalable) reconfigurable devices
2. Helpful to have control and management located *inside* the device

- How do we design this layer?
- Custom ASIC?

Control and
Management
System (CMS)

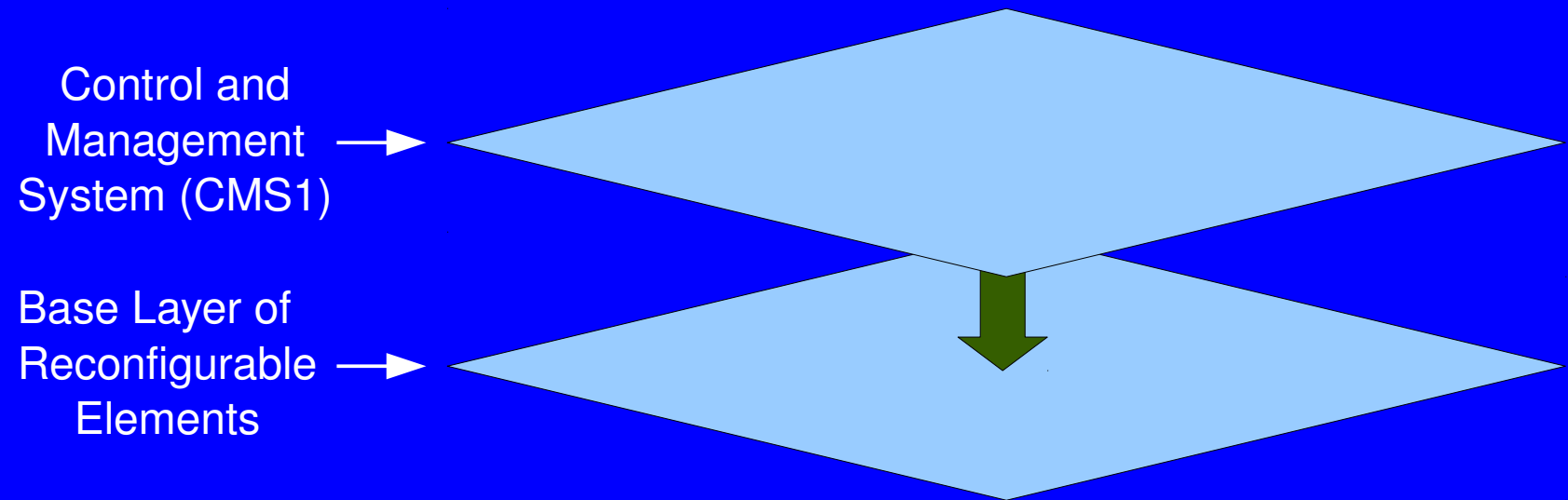


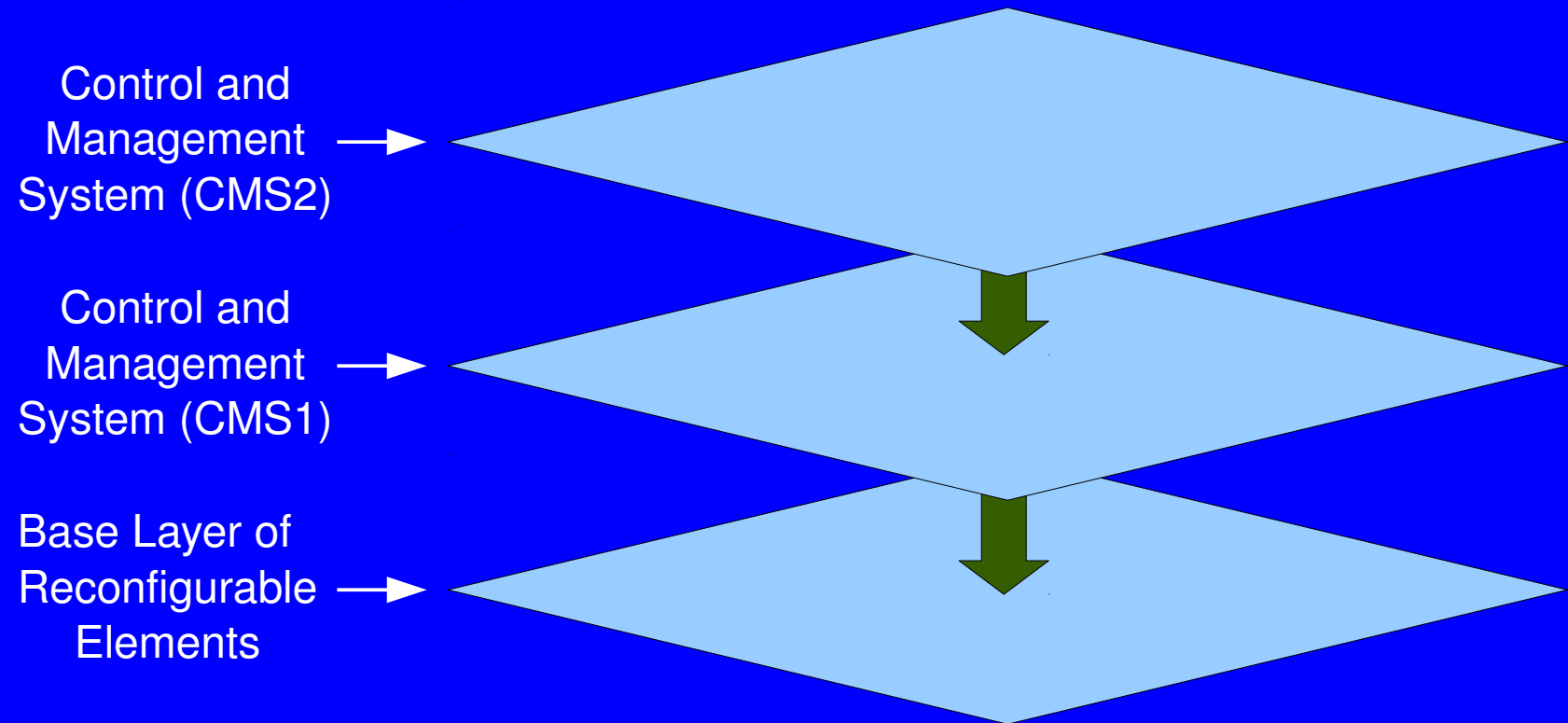
Base Layer of
Reconfigurable
Elements

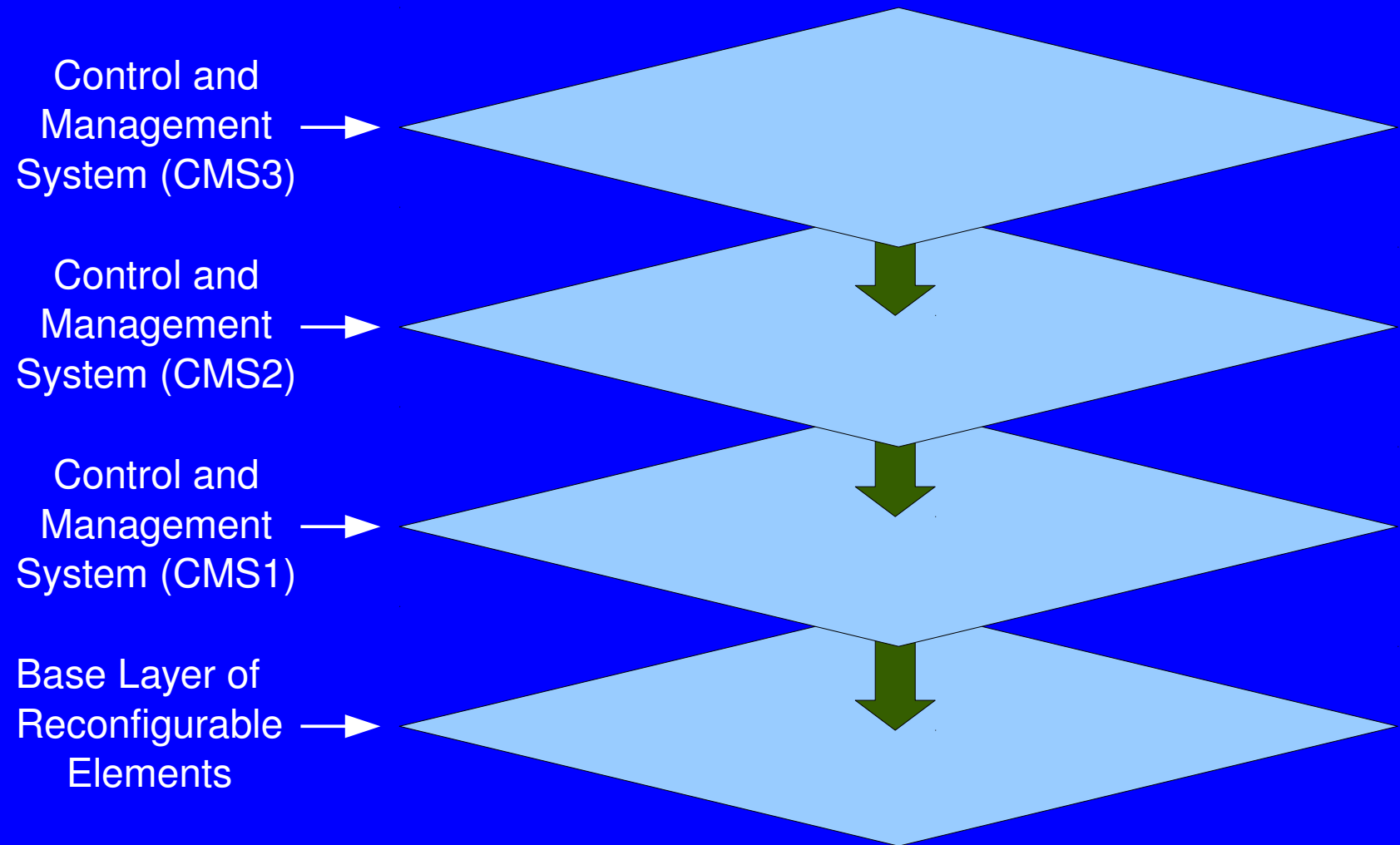


Hypothesis About Future Computing

1. Based on increasingly-large (scalable) reconfigurable devices
2. Helpful to have control and management located *inside* the device
3. As system scales, it's useful to have the control and management system be reconfigurable







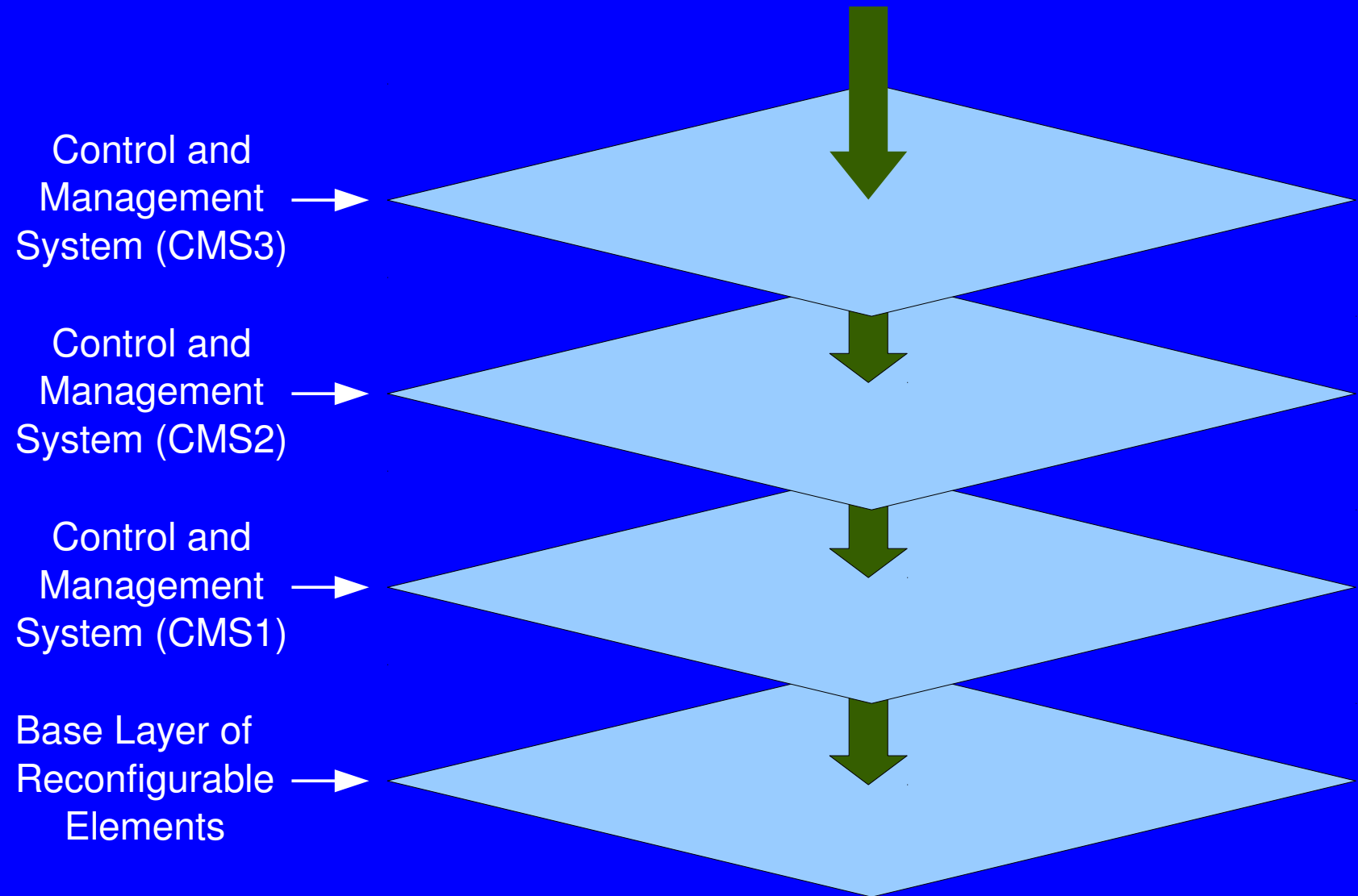
CUSTOM ASIC ???

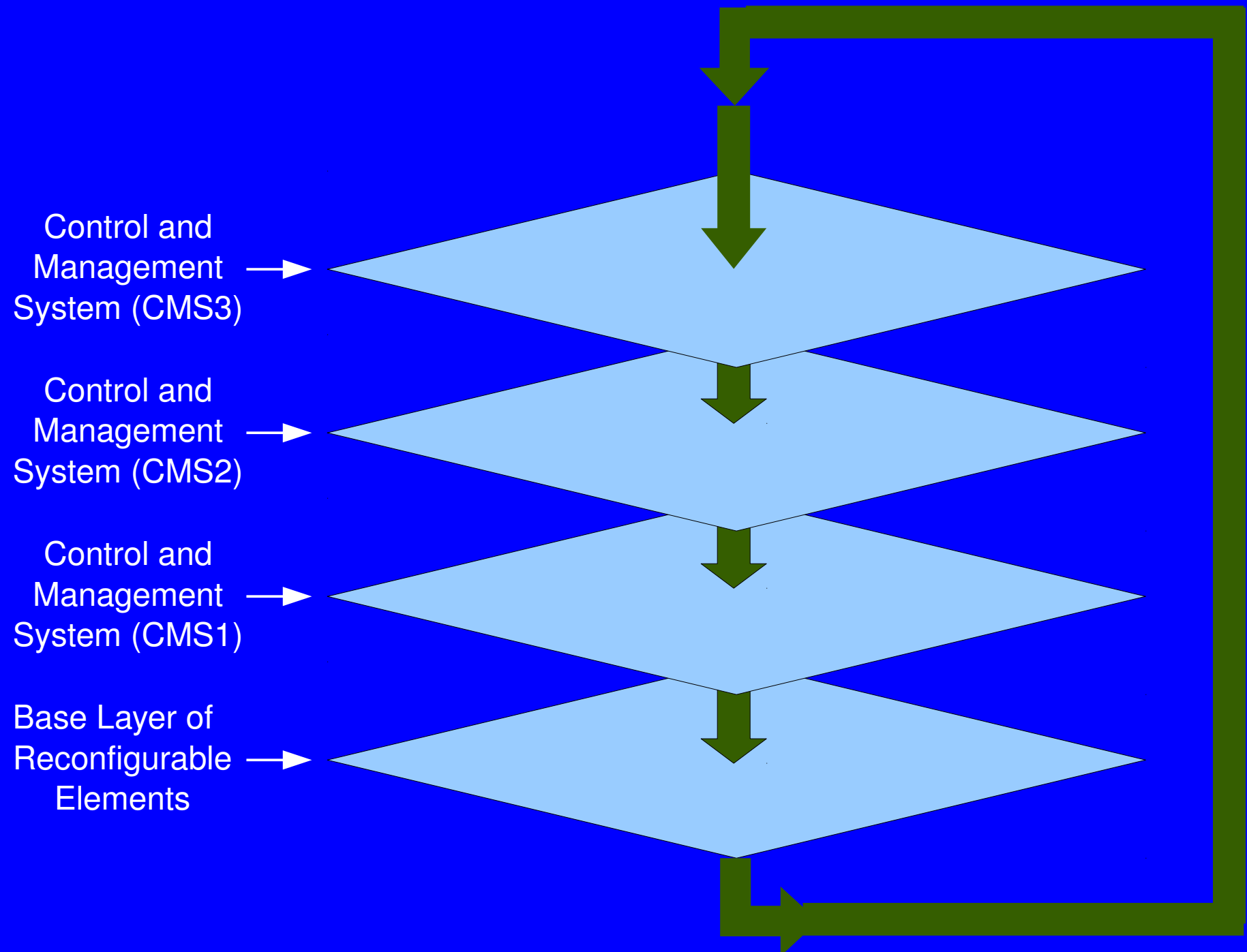
Control and
Management
System (CMS3)

Control and
Management
System (CMS2)

Control and
Management
System (CMS1)

Base Layer of
Reconfigurable
Elements





Hypothesis 4:
A hierarchy-free arrangement of
***controlling* and *controlled* objects**
may be interesting.

“Closing the loop”

Hypothesis 4: “Closing The Loop”

Subject/Object Dualism

Hypothesis 4: “Closing The Loop”

I kick the ball

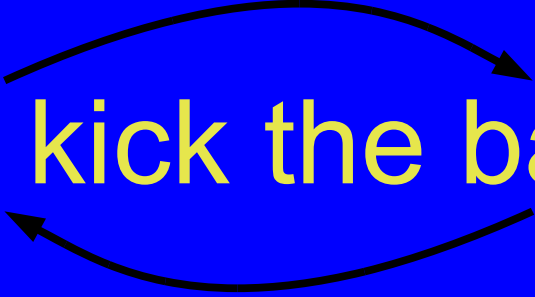
Hypothesis 4: “Closing The Loop”



I kick the ball

Hypothesis 4: “Closing The Loop”

I kick the ball

A diagram illustrating a feedback loop. Two curved arrows form a circle around the text "I kick the ball". The top arrow points from the left side of the text to the right side, and the bottom arrow points from the right side of the text back to the left side, creating a continuous loop.

Hypothesis 4: “Closing The Loop”

Non-Dualism: Interdependence between
subject and object

Less-Philosophical Example

A program that prints its own source code

1. create an array of strings
2. read that array and print each string
3. initialize the array with the program's source code


```
static char p[100][1024]={  
    (fill this in later)  
};
```

```
main()  
{  
    int i;  
    for (i=0;i<100;i++)  
        printf("%s\n",p[i]);  
}
```

```
static char p[100][1024]={  
    "static char p[100][1024]={",  
};
```

```
main()  
{  
    int i;  
    for (i=0;i<100;i++)  
        printf("%s\n",p[i]);  
}
```

```
static char p[100][1024]={  
    "static char p[100][1024]={",  
    "\"static char p[100][1024]={\"",  
};
```

```
main()  
{  
    int i;  
    for (i=0;i<100;i++)  
        printf("%s\n",p[i]);  
}
```

```
static char p[100][1024]={
    "static char p[100][1024]={",
    "\static char p[100][1024]={\",",
    "\\static char p[100][1024]={\\\"\",",
};
```

```
main()
{
    int i;
    for (i=0;i<100;i++)
        printf("%s\n",p[i]);
}
```



```
static char p[100][1024]={
    "static char p[100][1024]={",
    "\"static char p[100][1024]={\"",
    "\\\"static char p[100][1024]={\\\"\"",
    "\\\"\\\"static char p[100][1024]={\\\"\\\"\\\"\",\",\",
};
```

```
main()
{
    int i;
    for (i=0;i<100;i++)
        printf("%s\n",p[i]);
}
```

We're treating this dualistically, as two separate pieces

- a program (the *subject*) that reads
- an array of characters (the *object*)

We're treating code and data as distinct

In fact, code and data are interrelated

- When the code (program) changes, the data (string array) needs to also change
- If the data changes, the code needs to change accordingly

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

```
        for (j=0;j<strlen(p[i]);j++){
```

```
            if ((p[i][j]=='\" || p[i][j]=='\\') printf("\\");
```

```
            printf("%c",p[i][j]);
```

```
        }
```

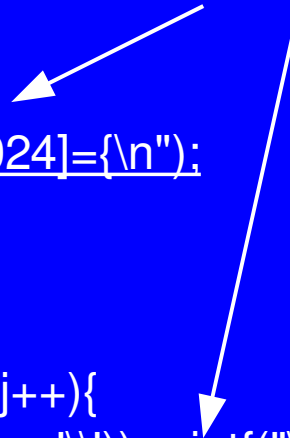
```
    }
```

```
    printf("\n};\n\n");
```

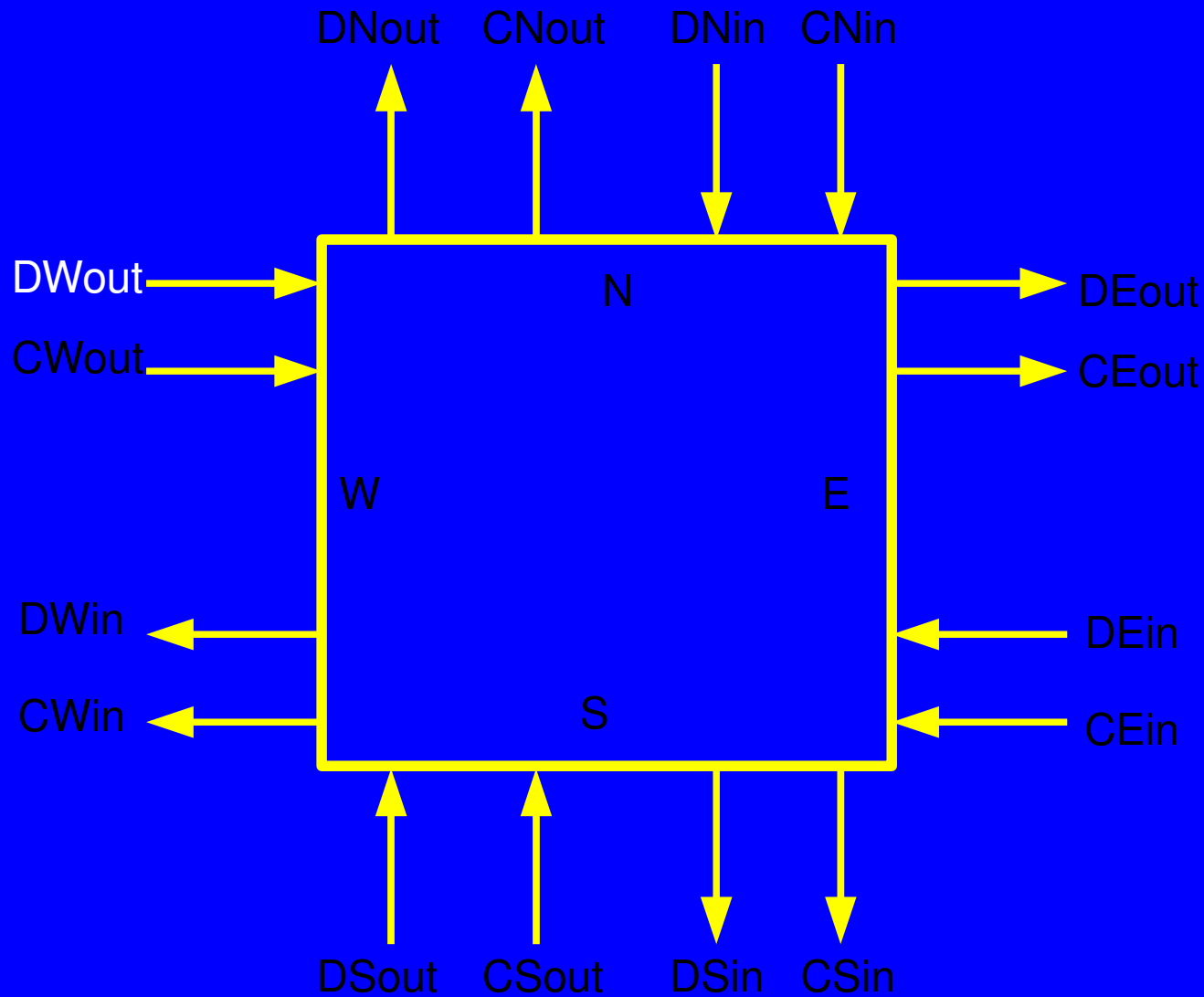
```
    for (i=0;i<20;i++) printf("%s\n",p[i]);
```

```
}
```

Part of the data has been
moved into the code



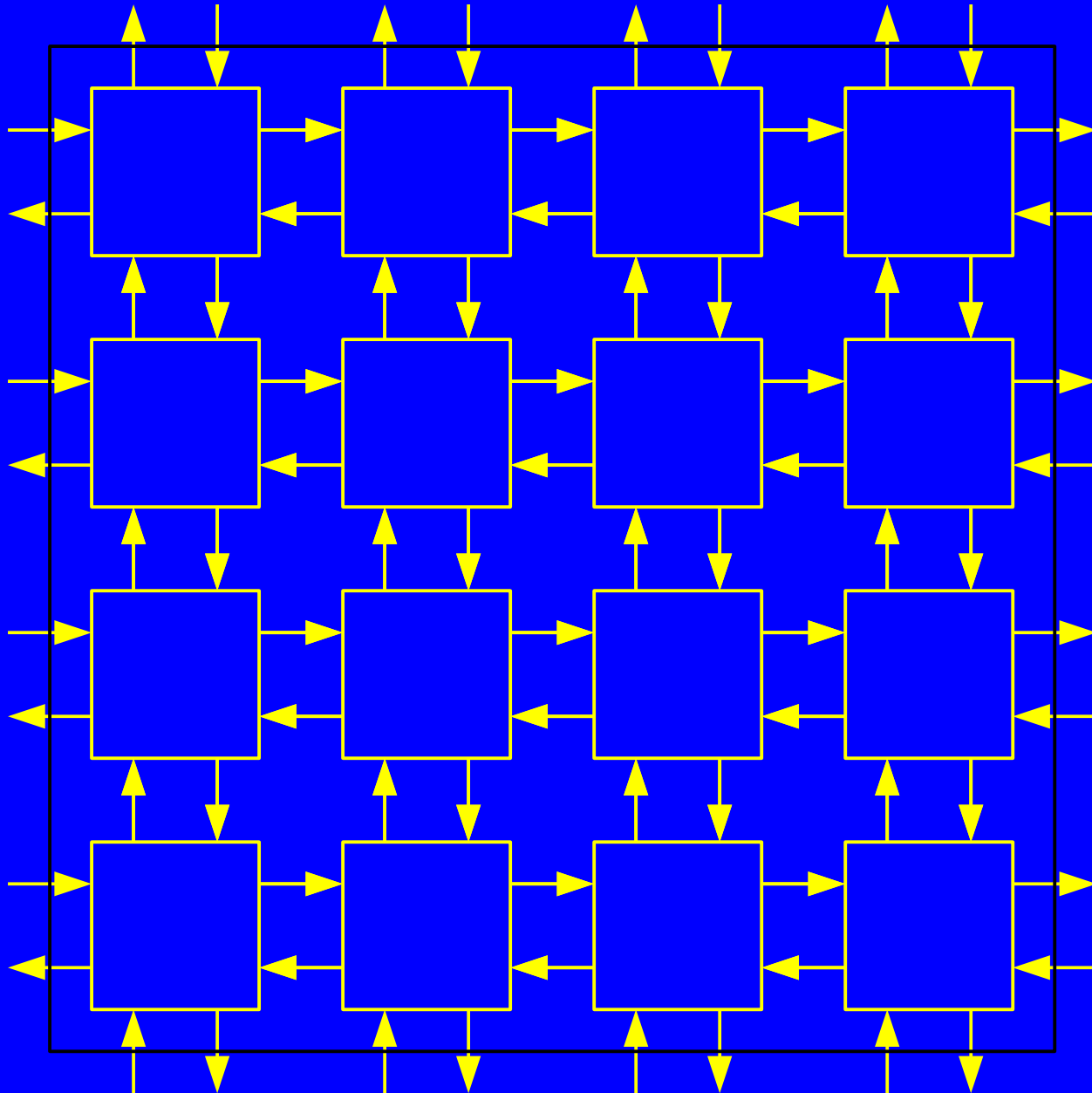
Single Element (4-Sided)



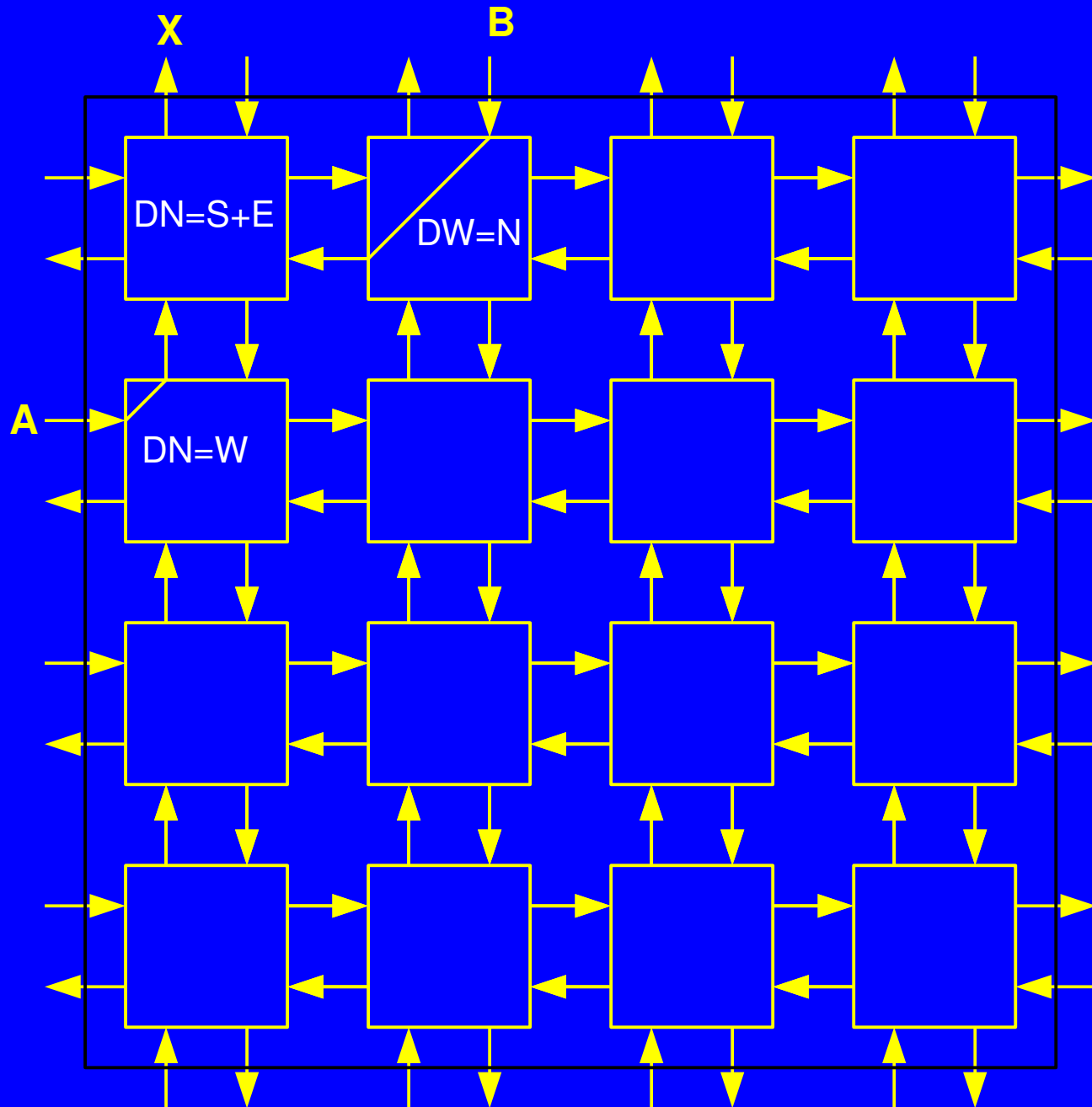
Each element's behavior is controlled by its configuration memory

DNin	DSin	DWin	DEin	CNout	CSout	CWout	CEout	DNout	DSout	DWout	DEout
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1	0	0
1	0	0	1	1	0	0	0	1	1	0	0
1	0	1	0	1	0	0	0	1	1	0	0
1	0	1	1	1	0	0	0	1	1	0	0
1	1	0	0	1	0	0	0	1	1	0	0
1	1	0	1	1	0	0	0	1	1	0	0
1	1	1	0	1	0	0	0	1	1	0	0
1	1	1	1	1	0	0	0	1	1	0	0

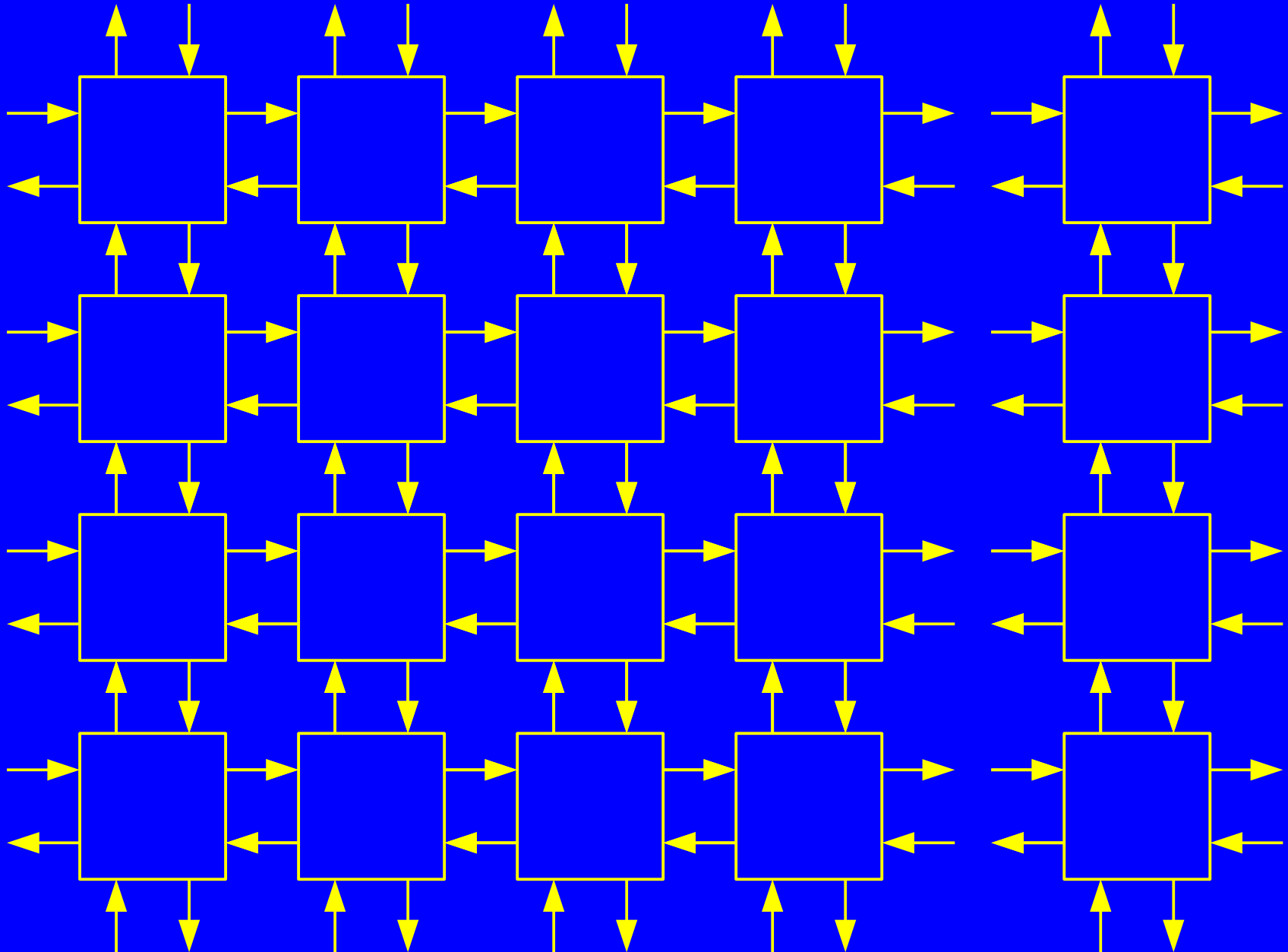
Reconfigurable array is composed of many elements



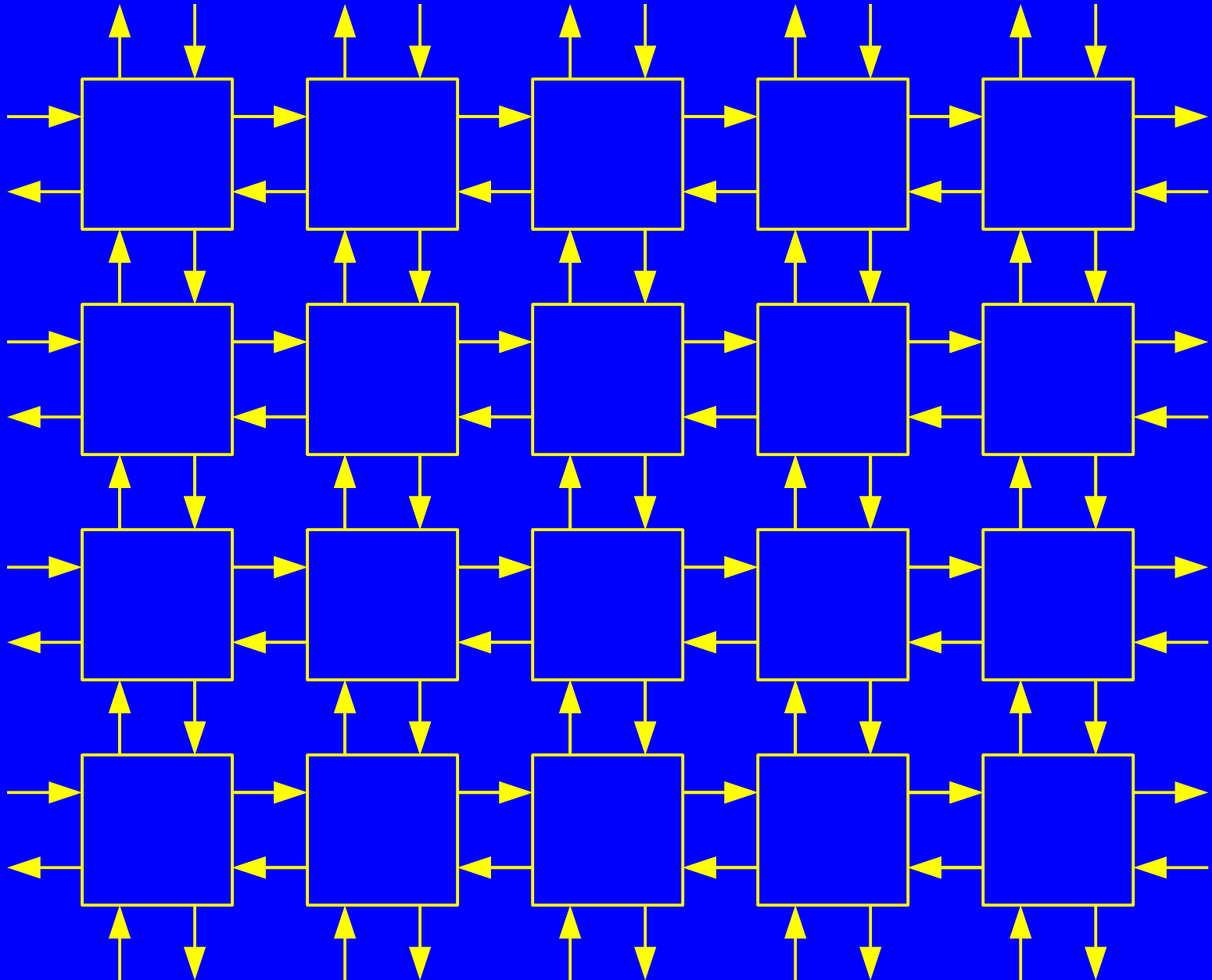
$X=A \text{ Or } B$



**Scalable – can grow by
adding to edges**



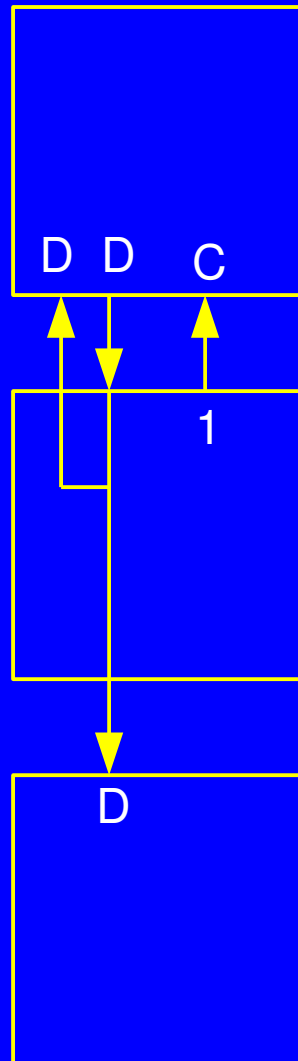
**Scalable – can grow by
adding to edges**



Configuring an Element

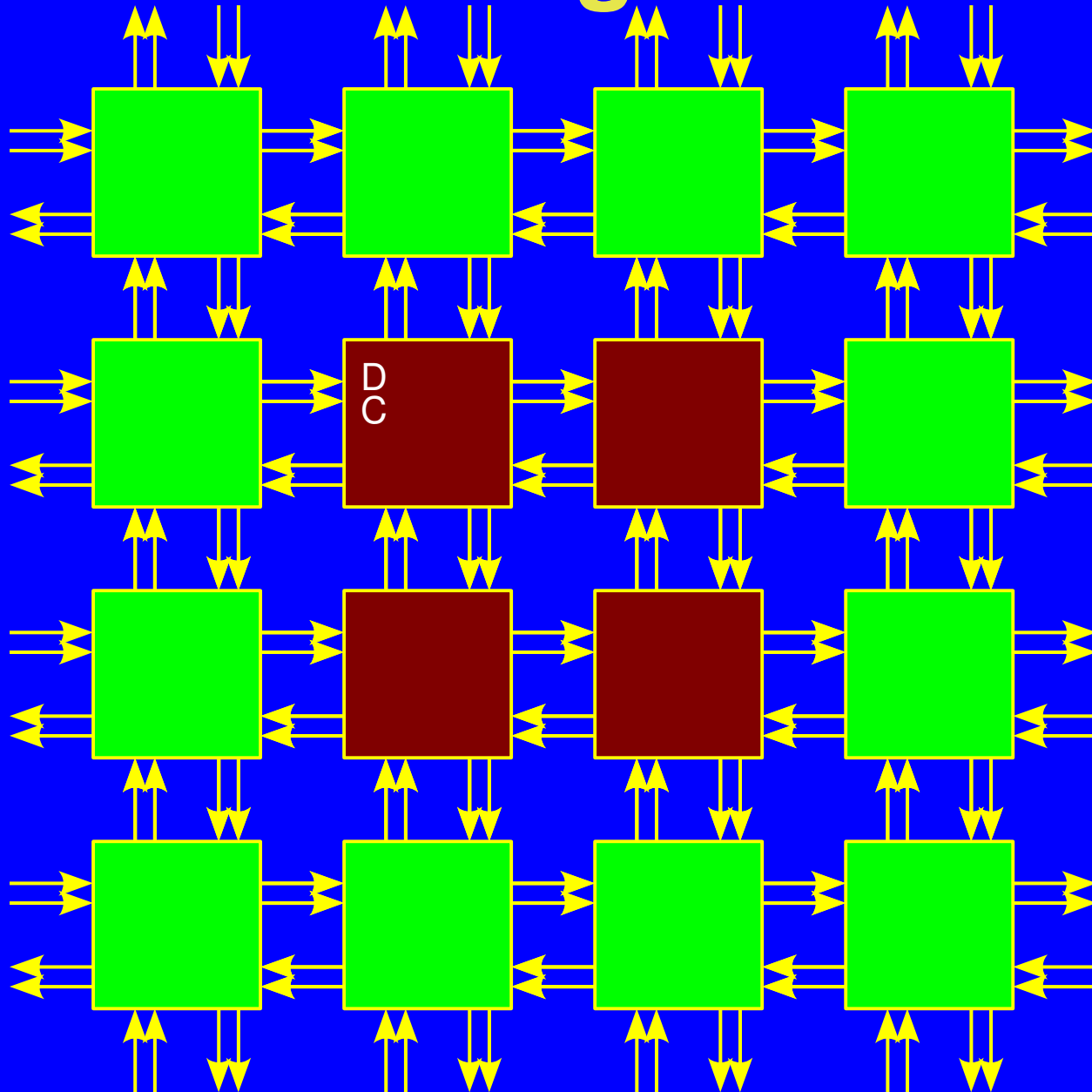
- Set C=1 and shift in a TT (in sync with a system clock) on the corresponding D input
- Can read the previous TT on the D output
- Called “C-Mode” (as opposed to “D-Mode”)

This allows read/write access to TTs for elements *if we can access their C and D inputs*

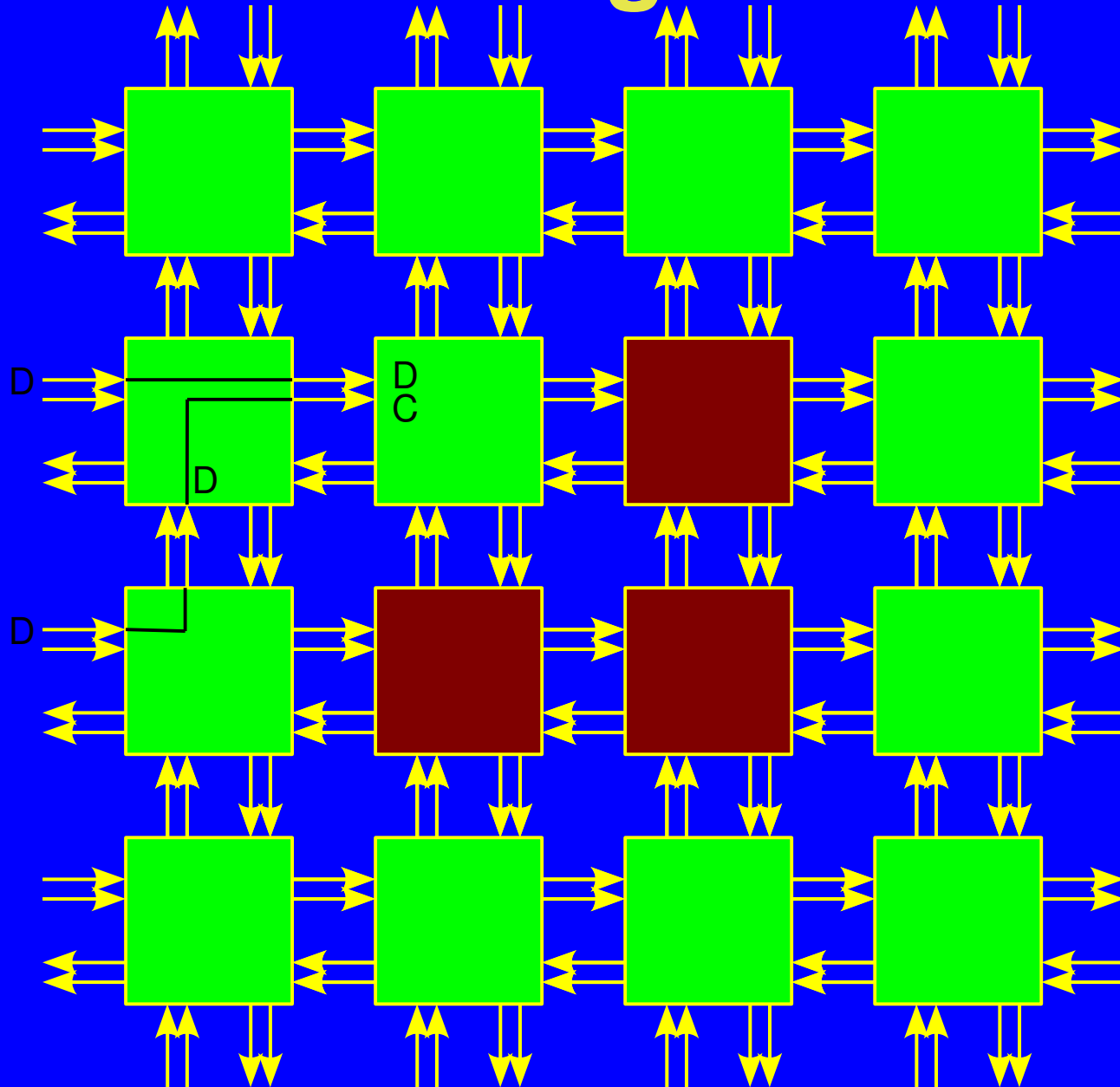


$CN=1 ; DN=N ; DS=N$

How are Non-Edge Elements Configured?

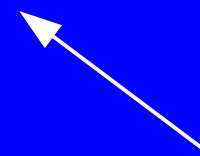
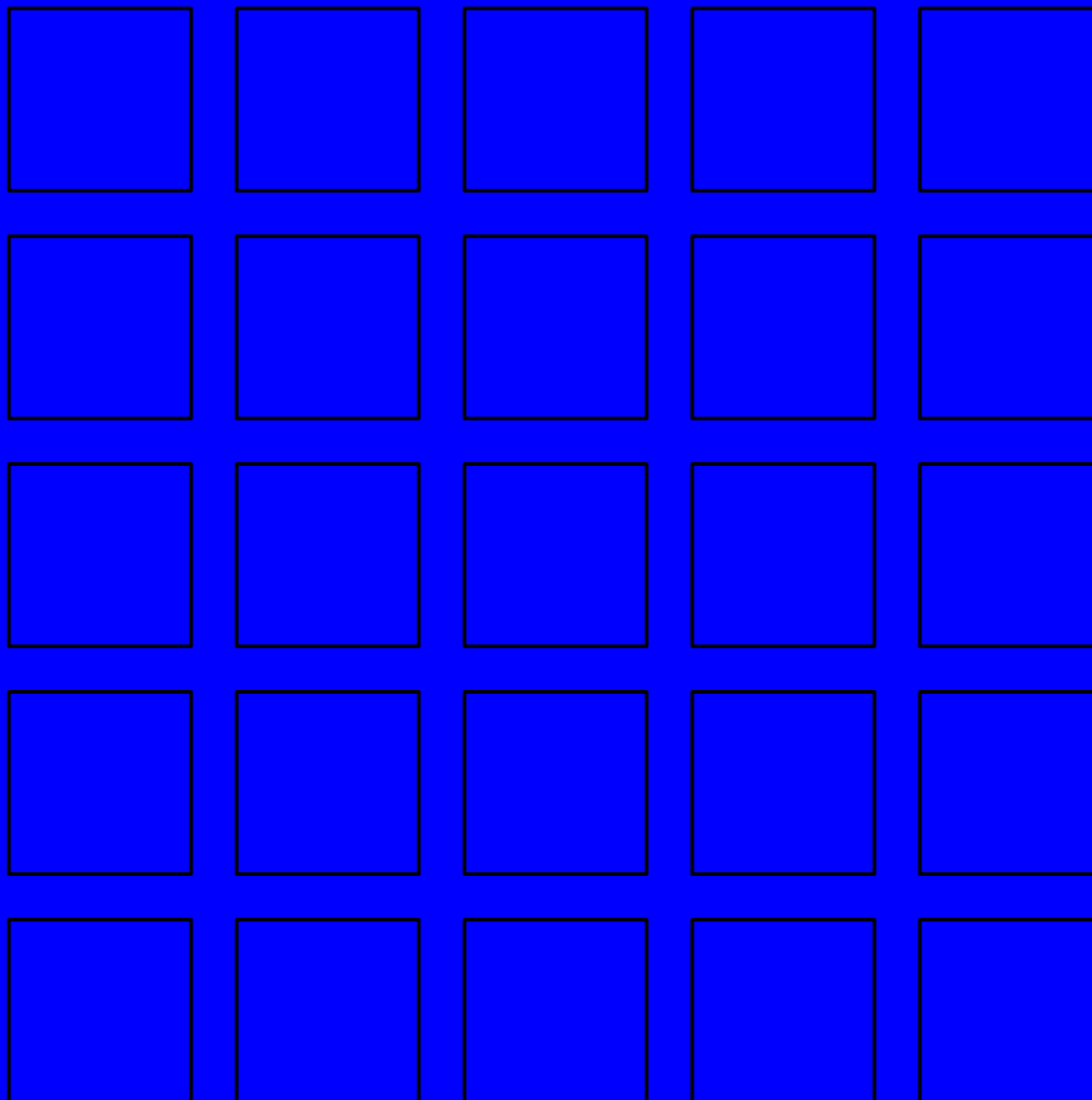


How are Non-Edge Elements Configured?

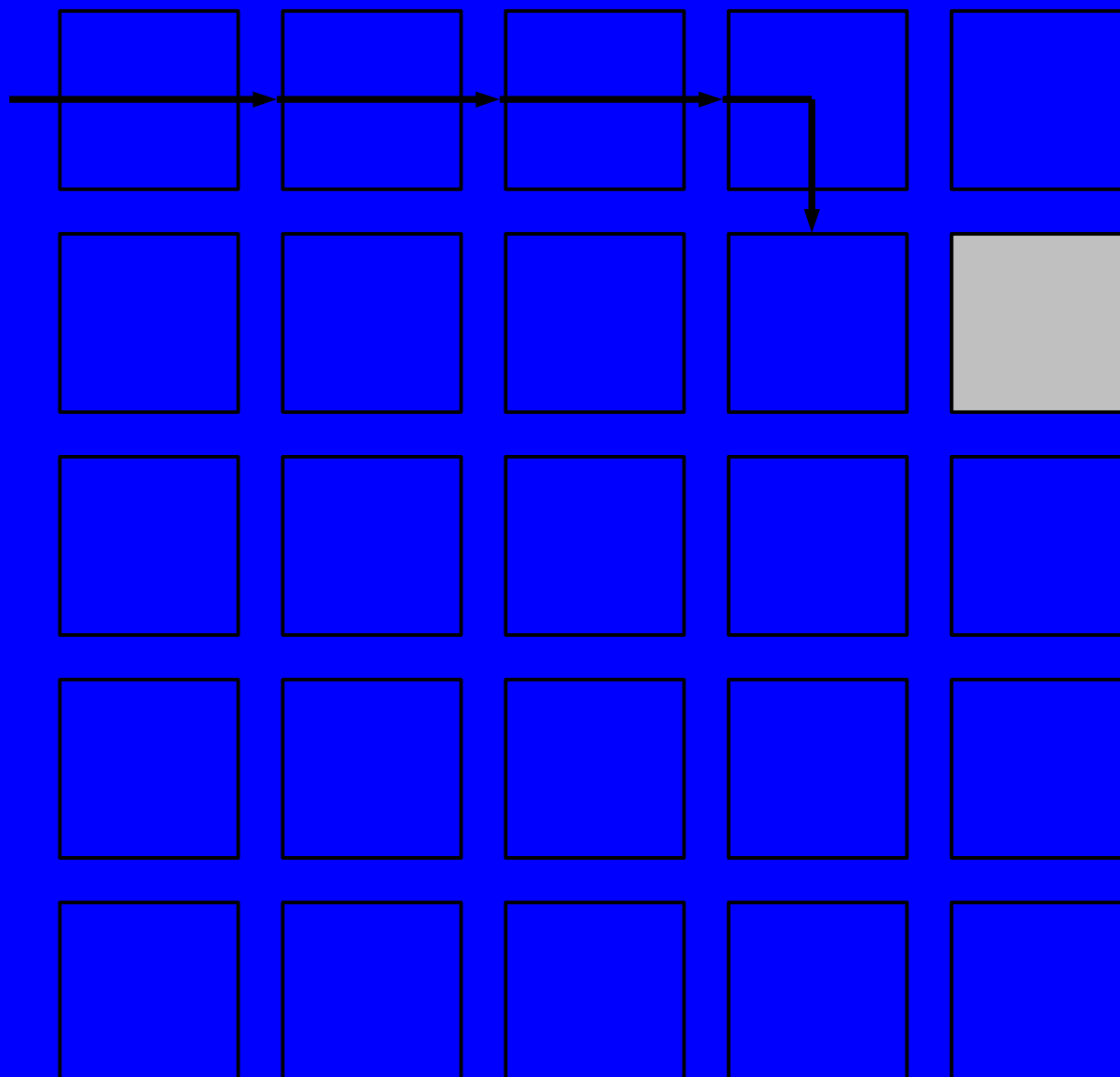


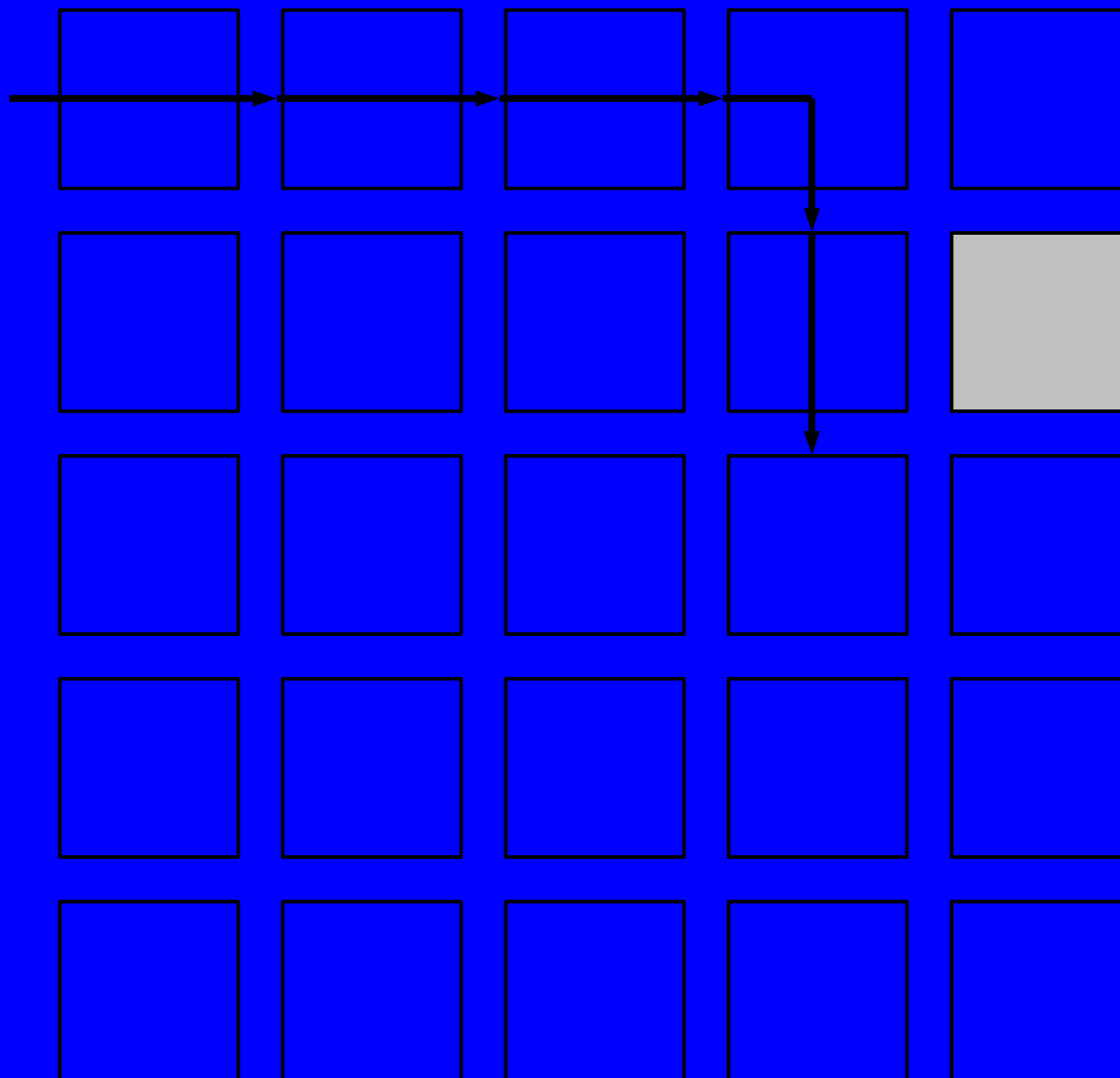
SEQUENCES

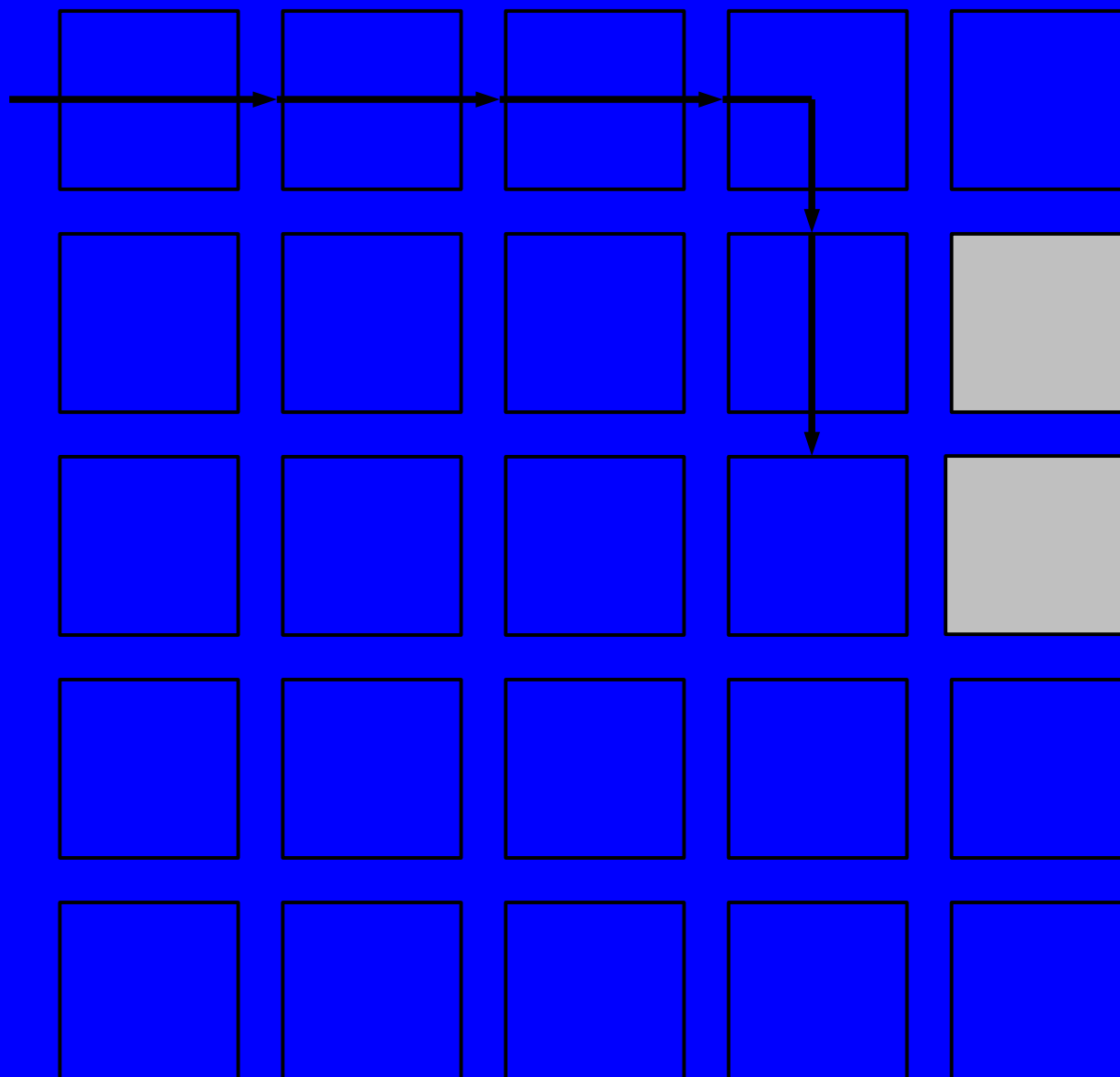
- Collections of D- and C- inputs that result in some desired set of configuration operations
- Examples:
 - building a W->E wire
 - building a corner wire from E->S
 - configuring an element to the east of a N->S wire
 - extend a N->S wire
- Supersequence: a sequence of sequences
 - Example: Bootstrapping

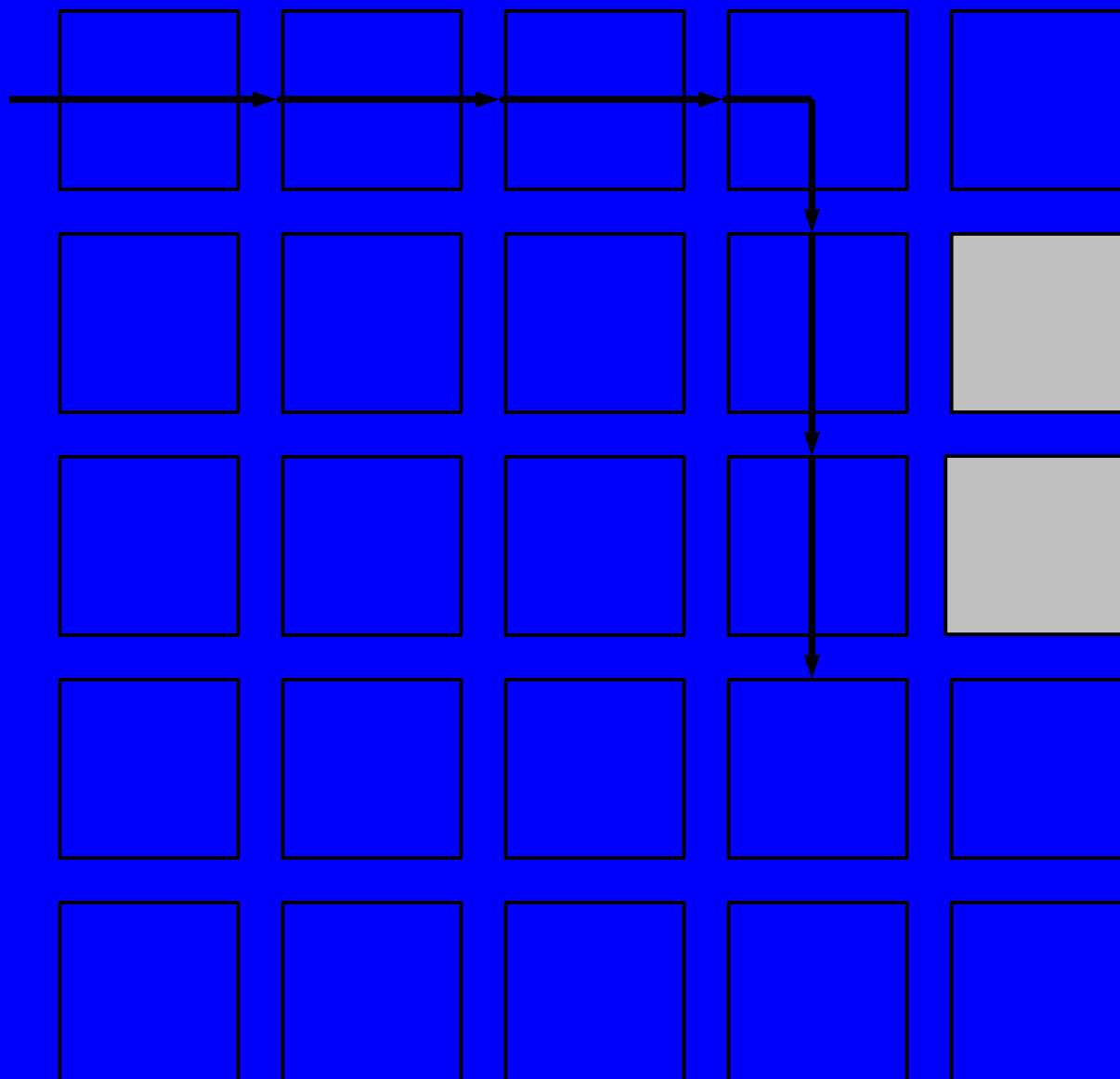


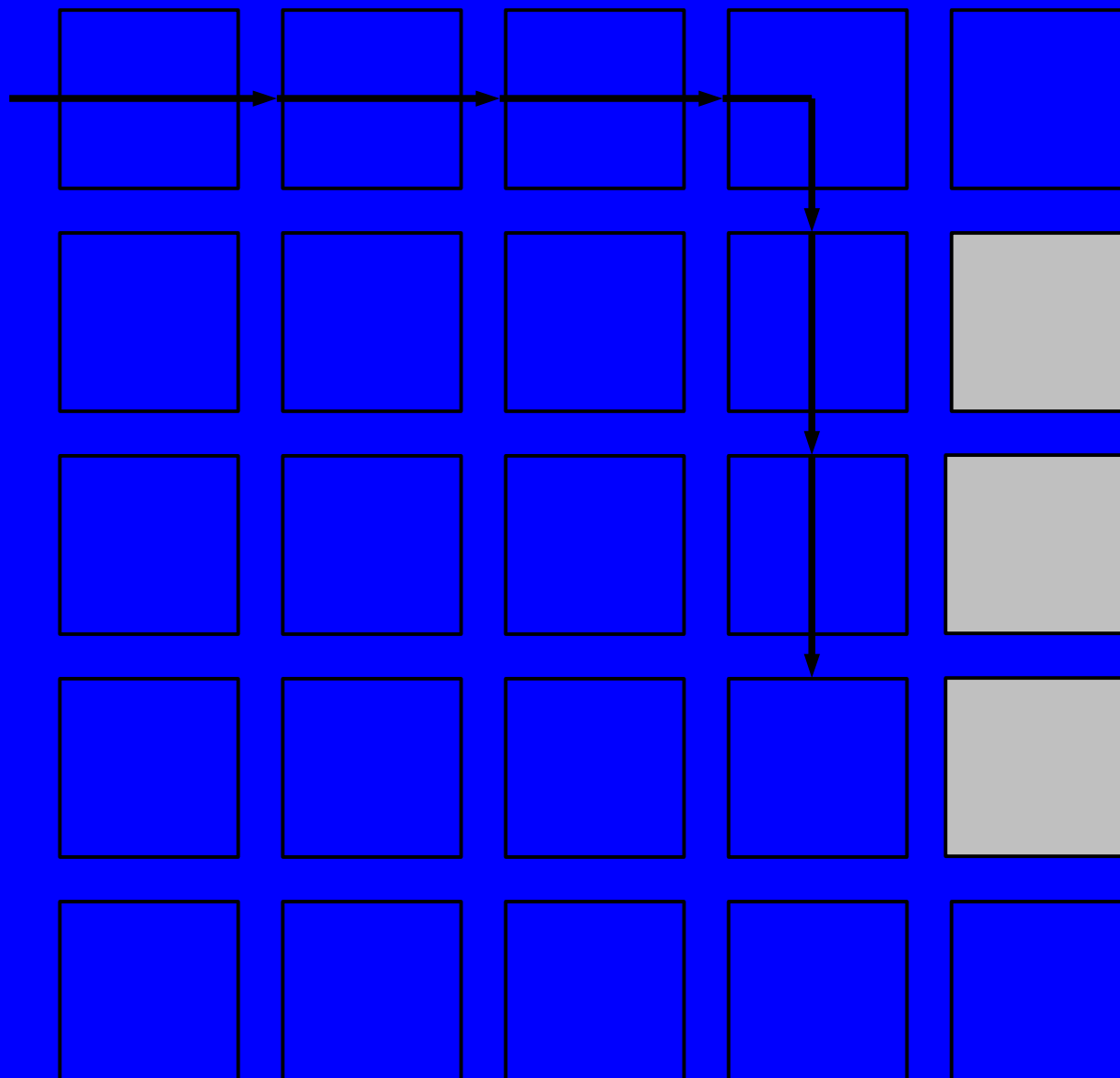
Several
Elements

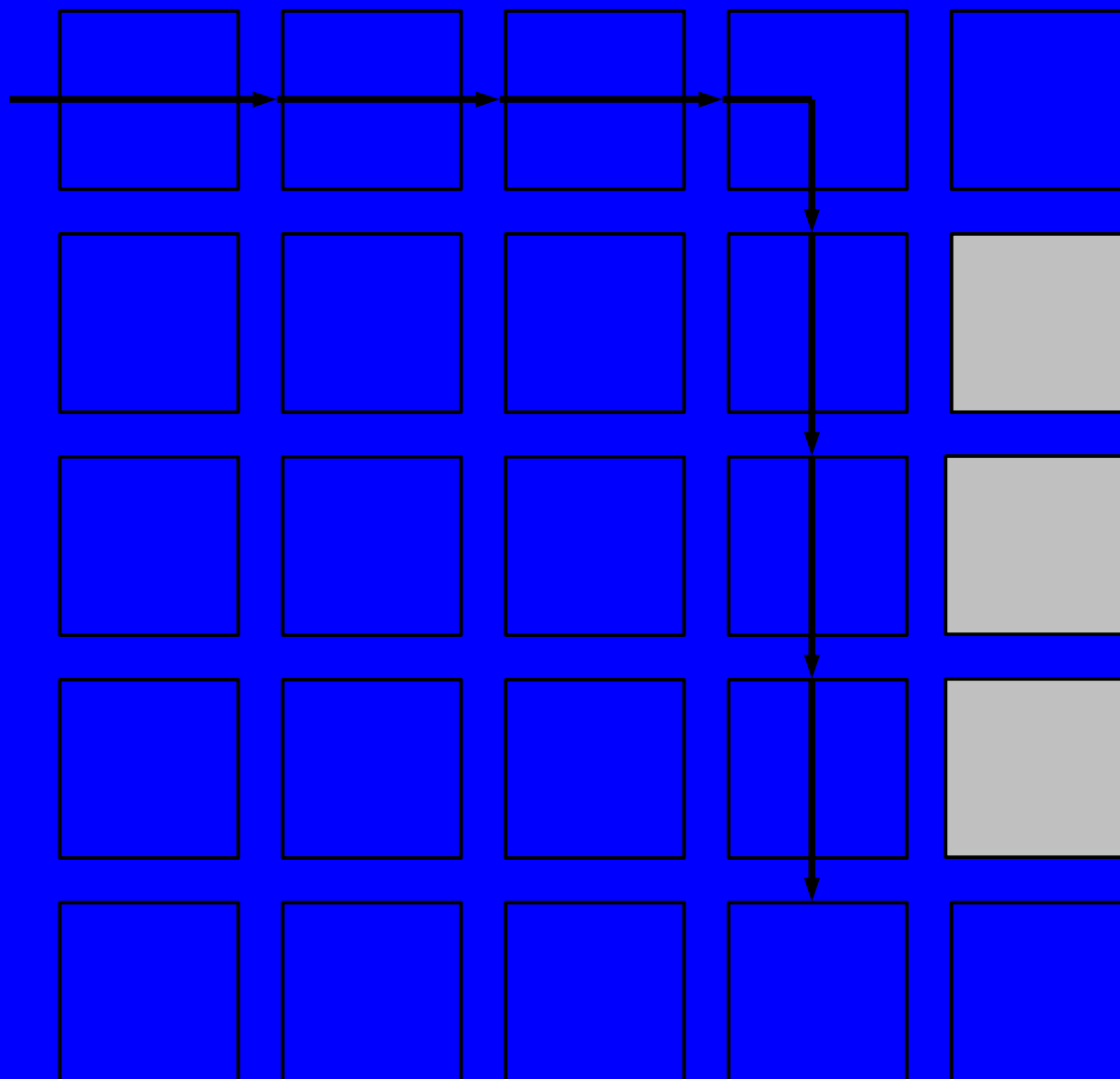


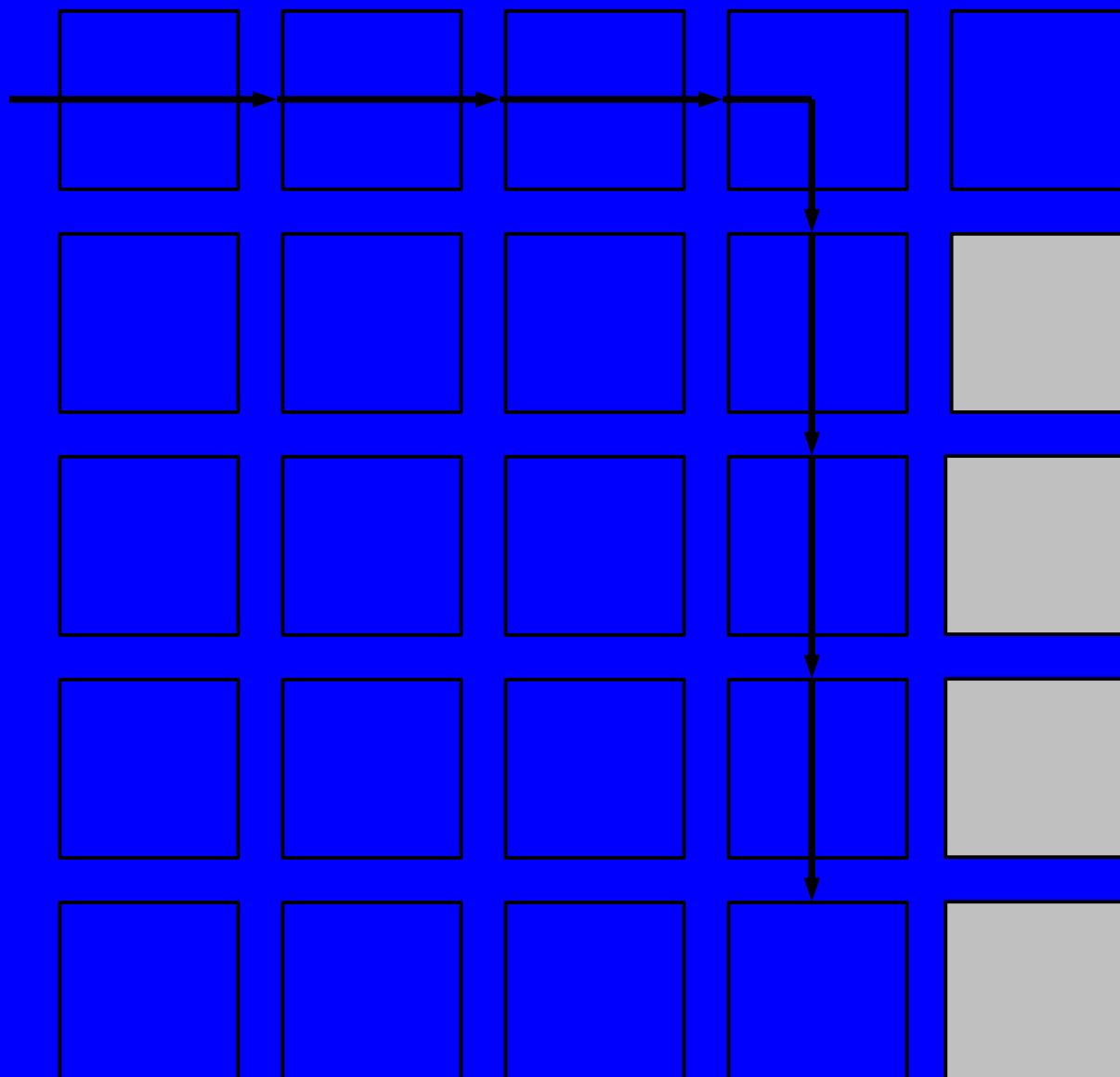






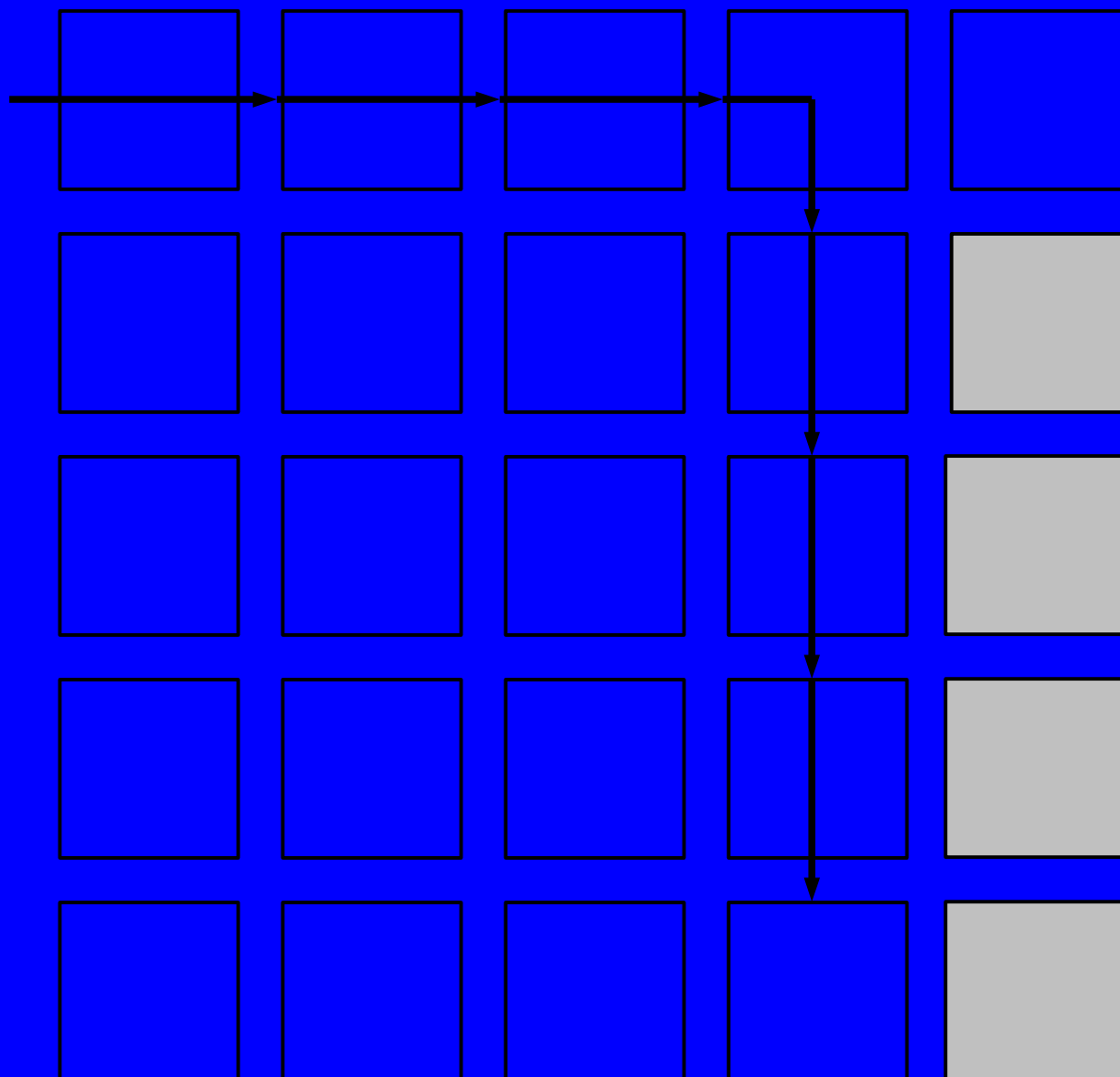


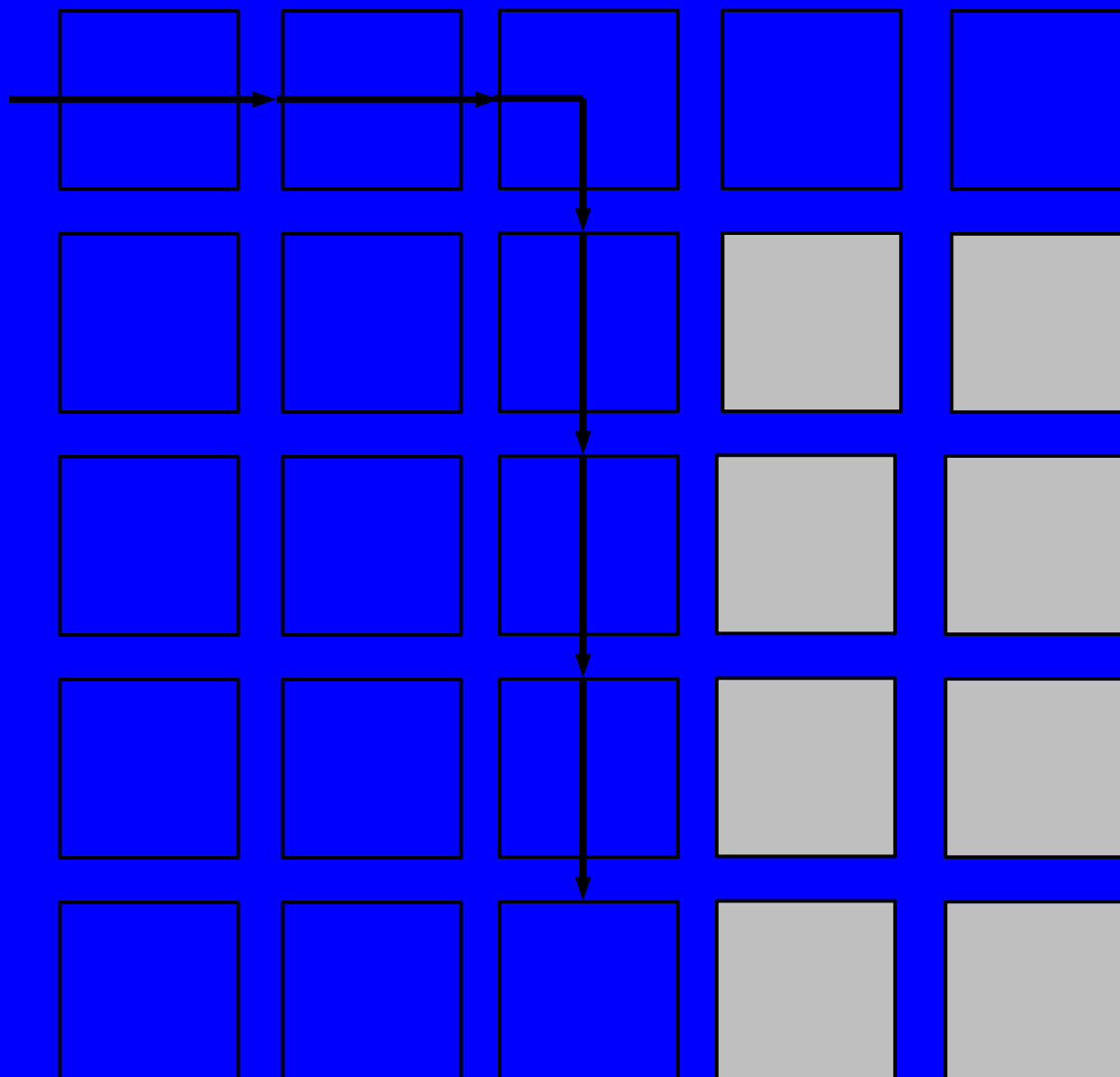


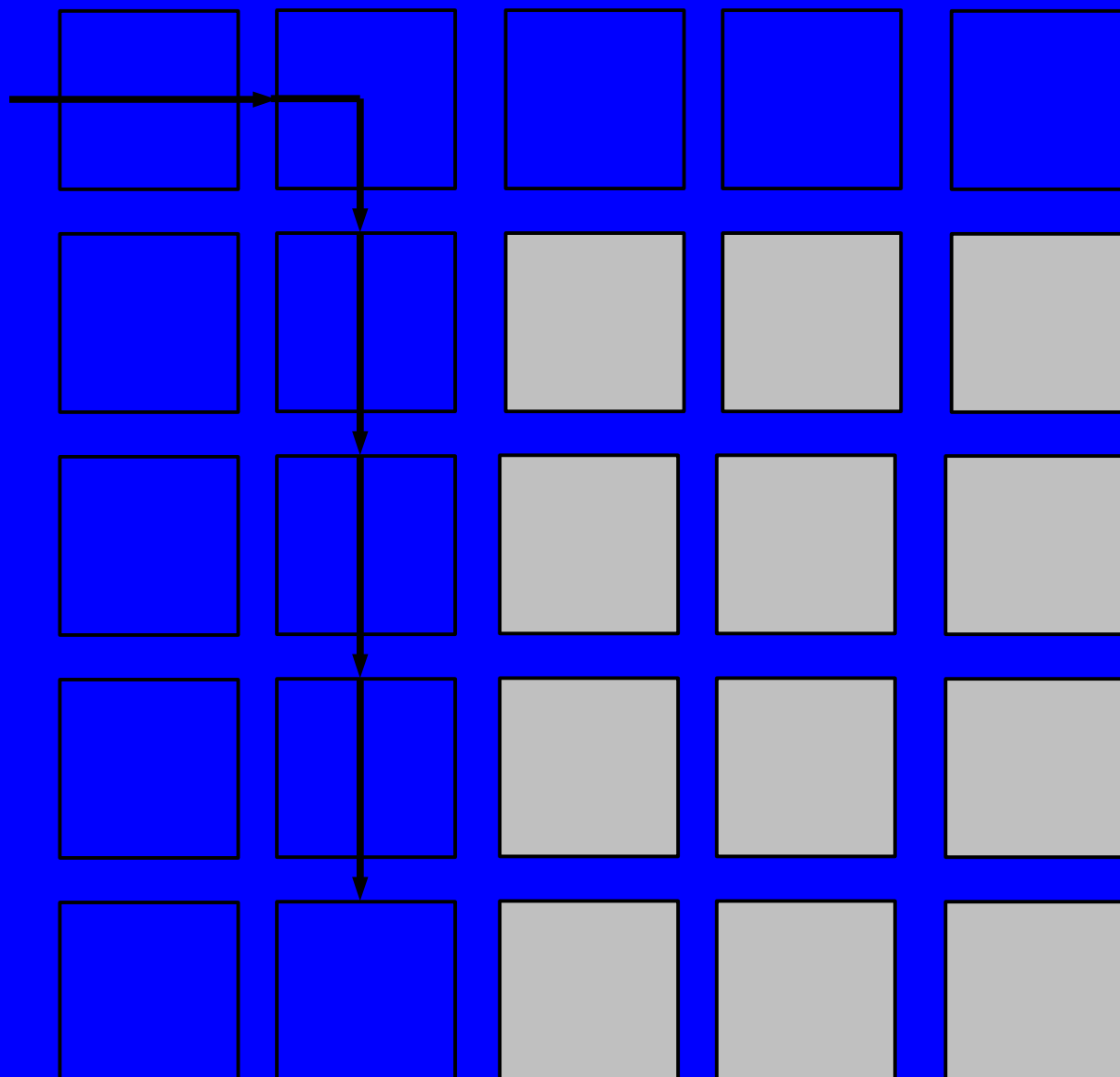


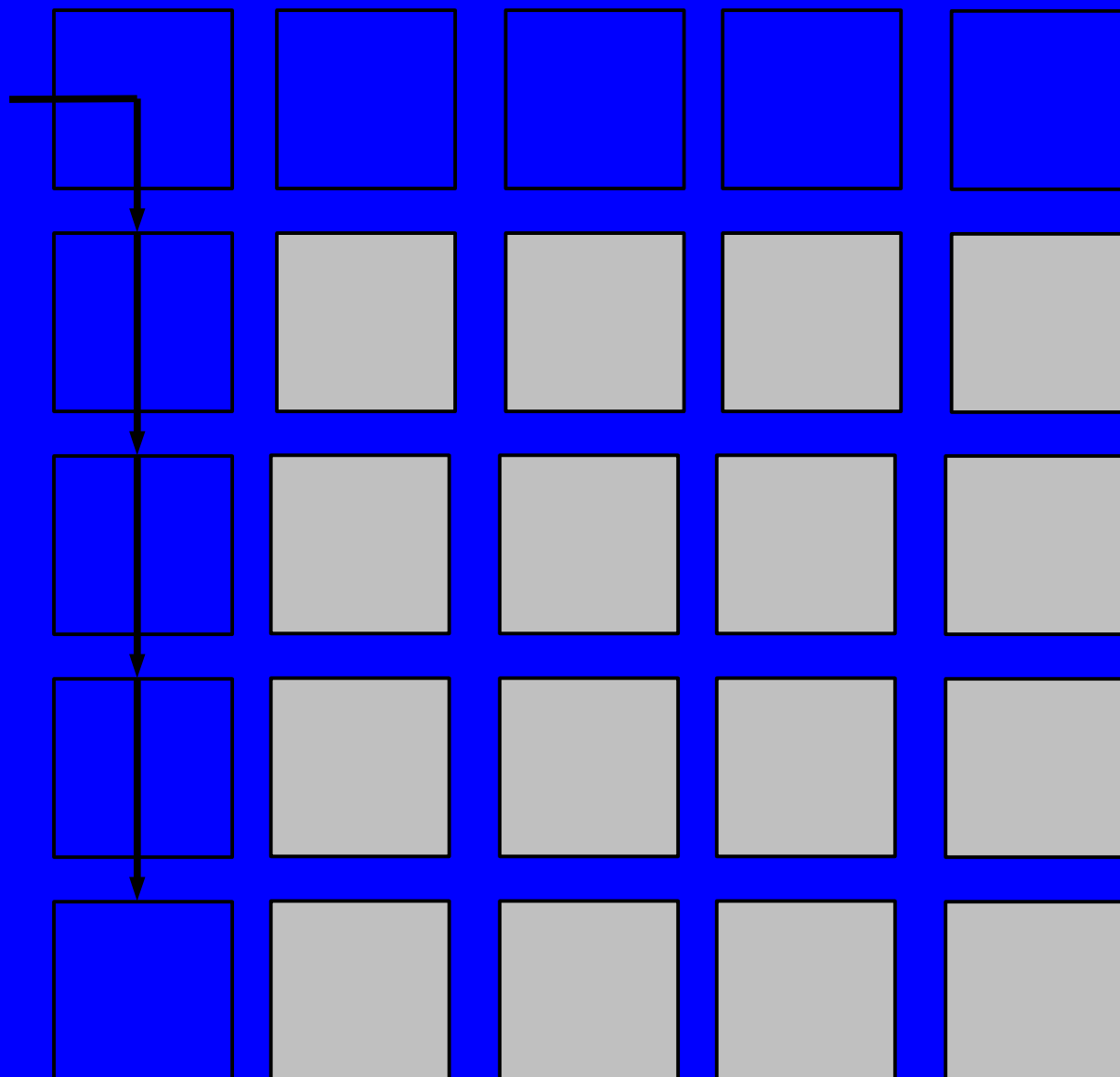
Full Set Of Sequences

- extend to east
- extend to east
- extend to east
- turn east->south
- configure to east;extend south
- configure to east;extend south
- configure to east;extend south
- configure to east







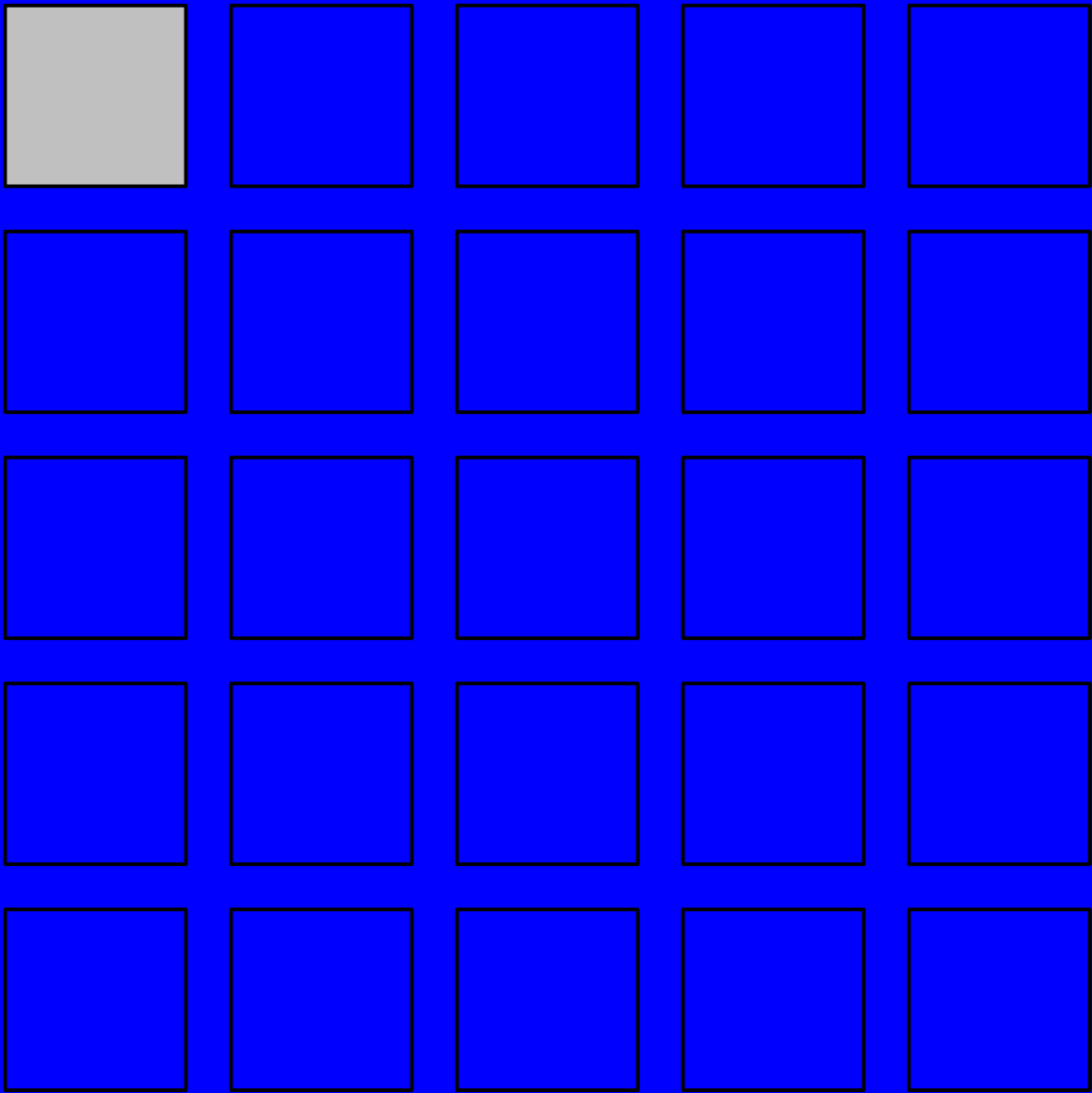


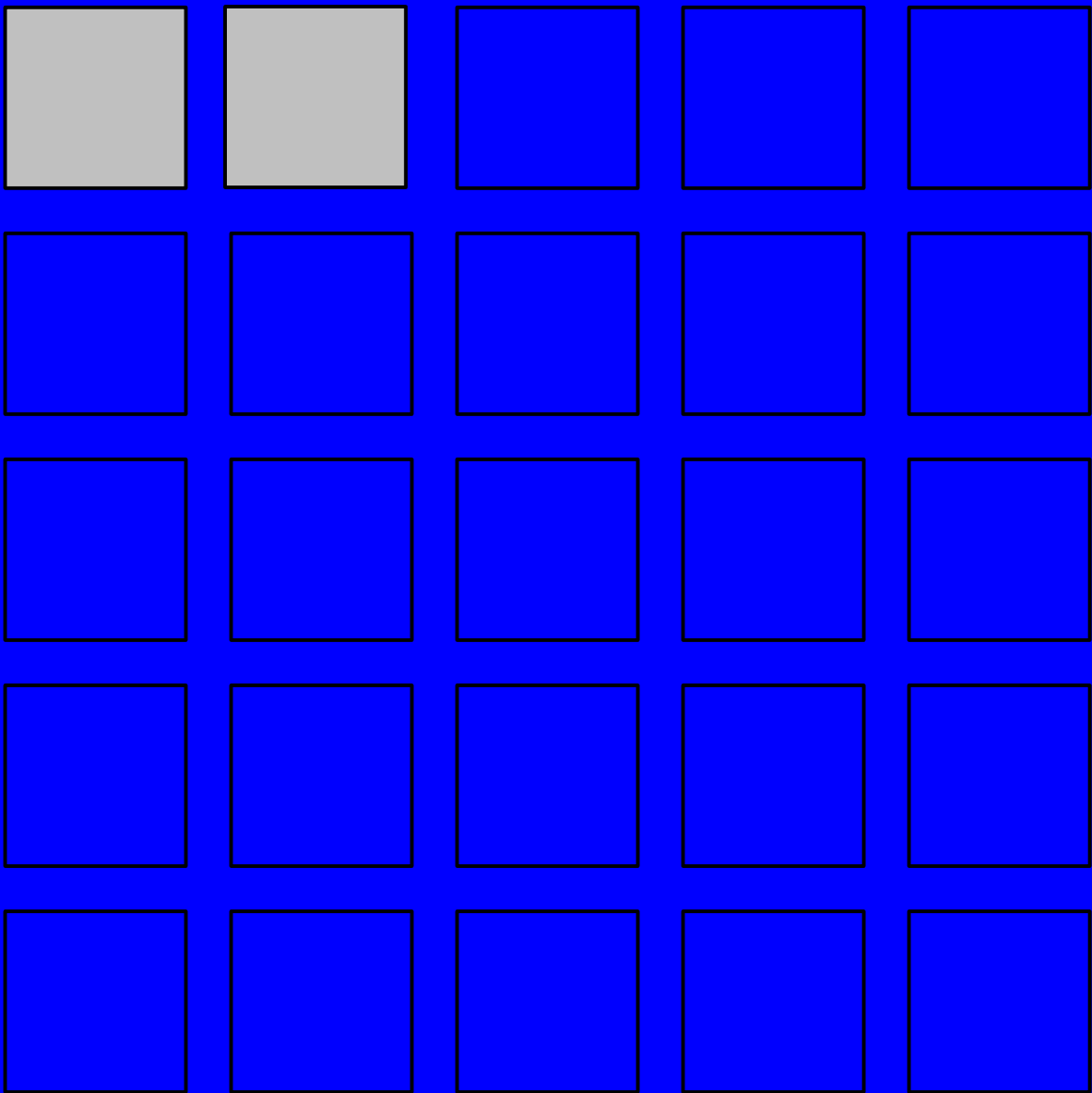
...but is this useful?

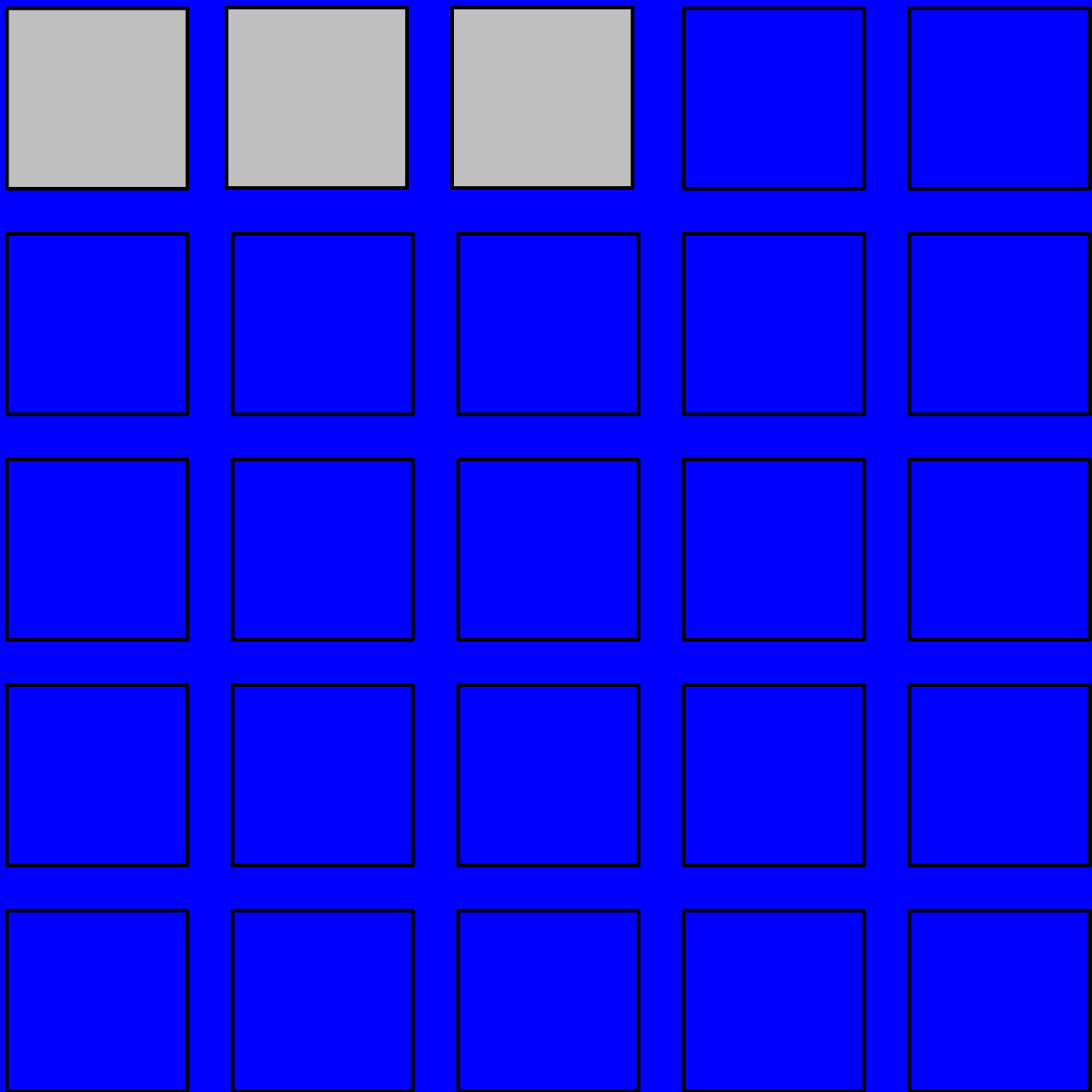
- So far, this sounds like a pain to work with
- Is there a practical, desirable application of this non-dualism?

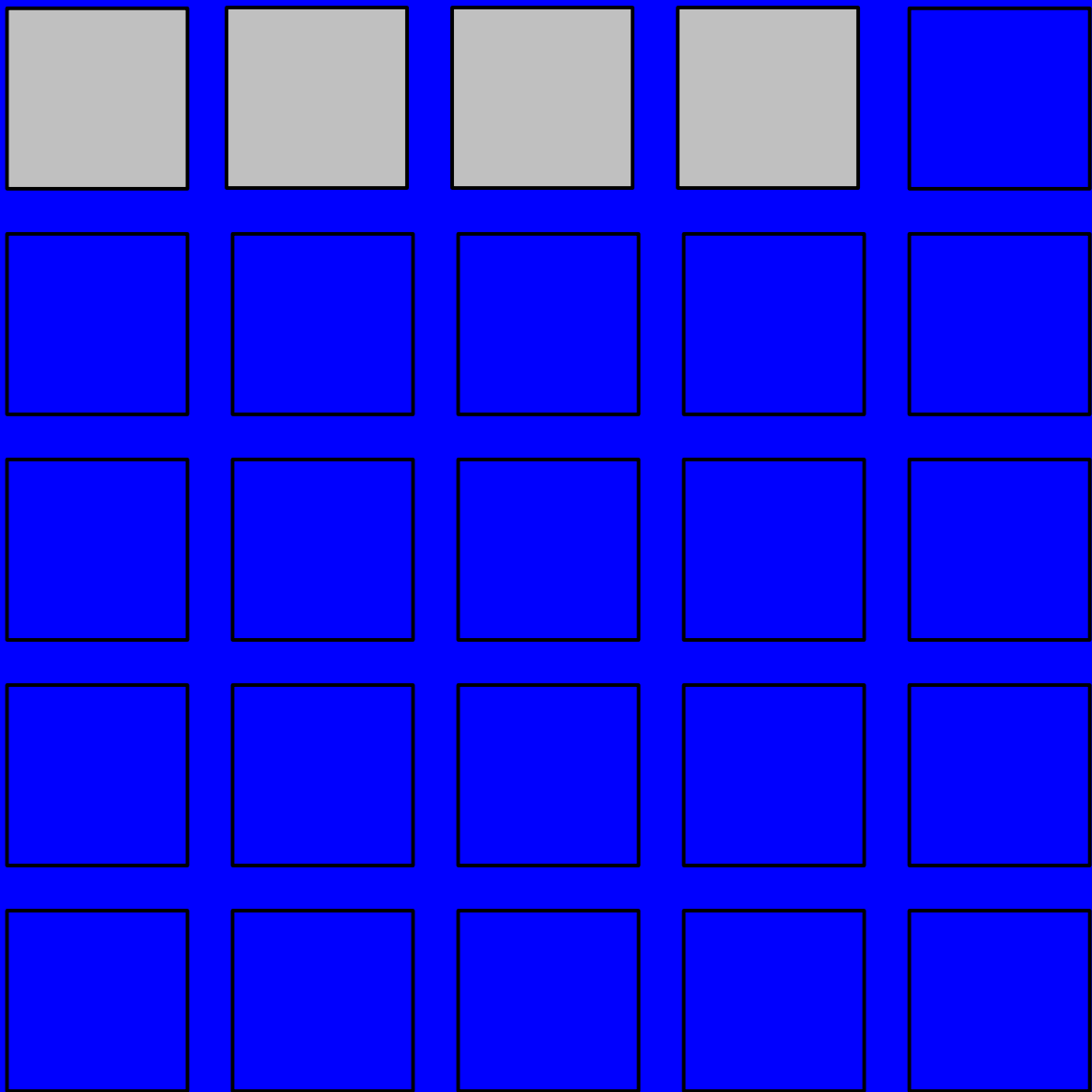
Parallel Configuration

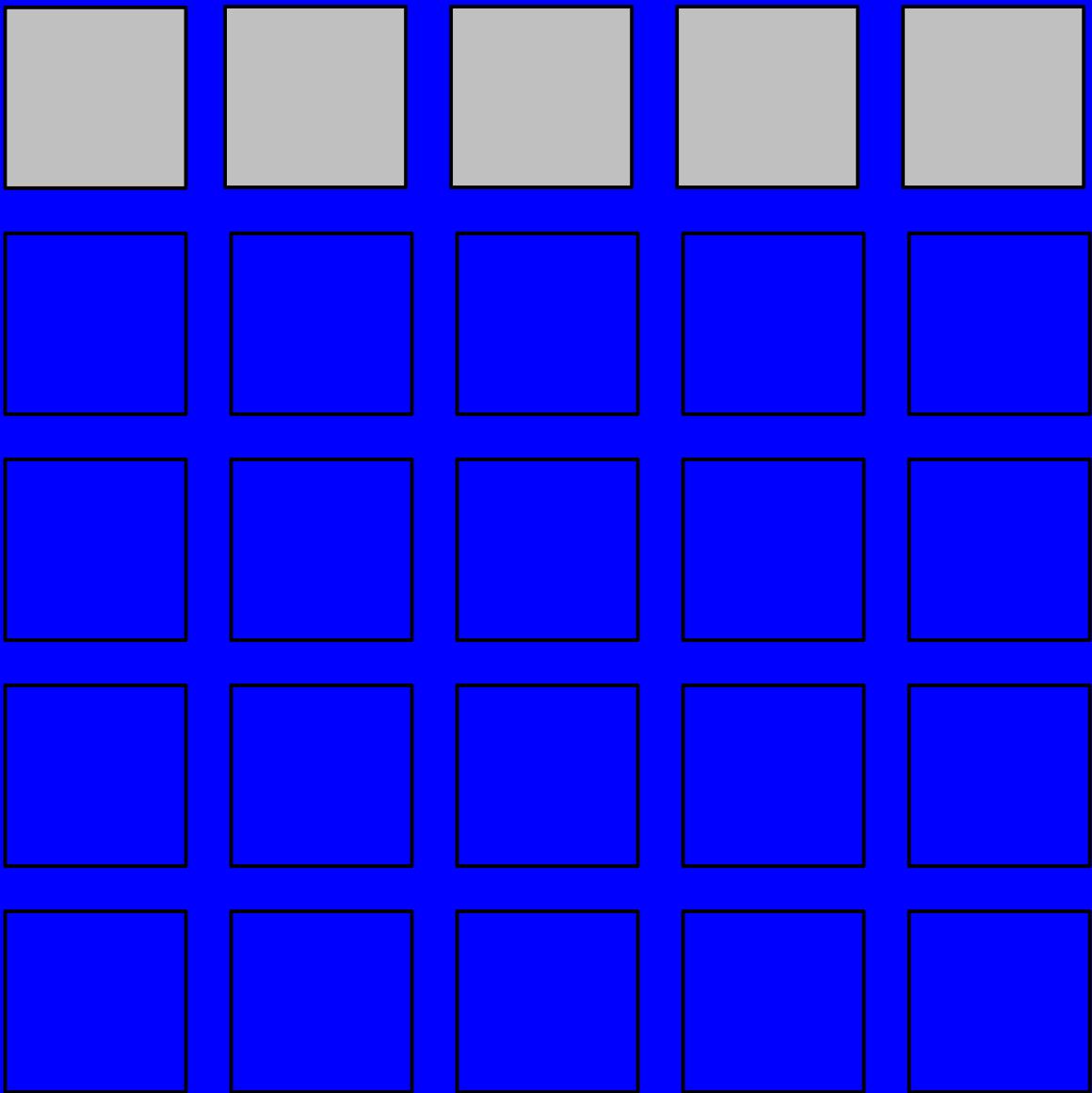
- Medusa Supersequence
- Medusa Circuit
- Useful for tiling an array with sub-circuits that are identical or very similar to each other

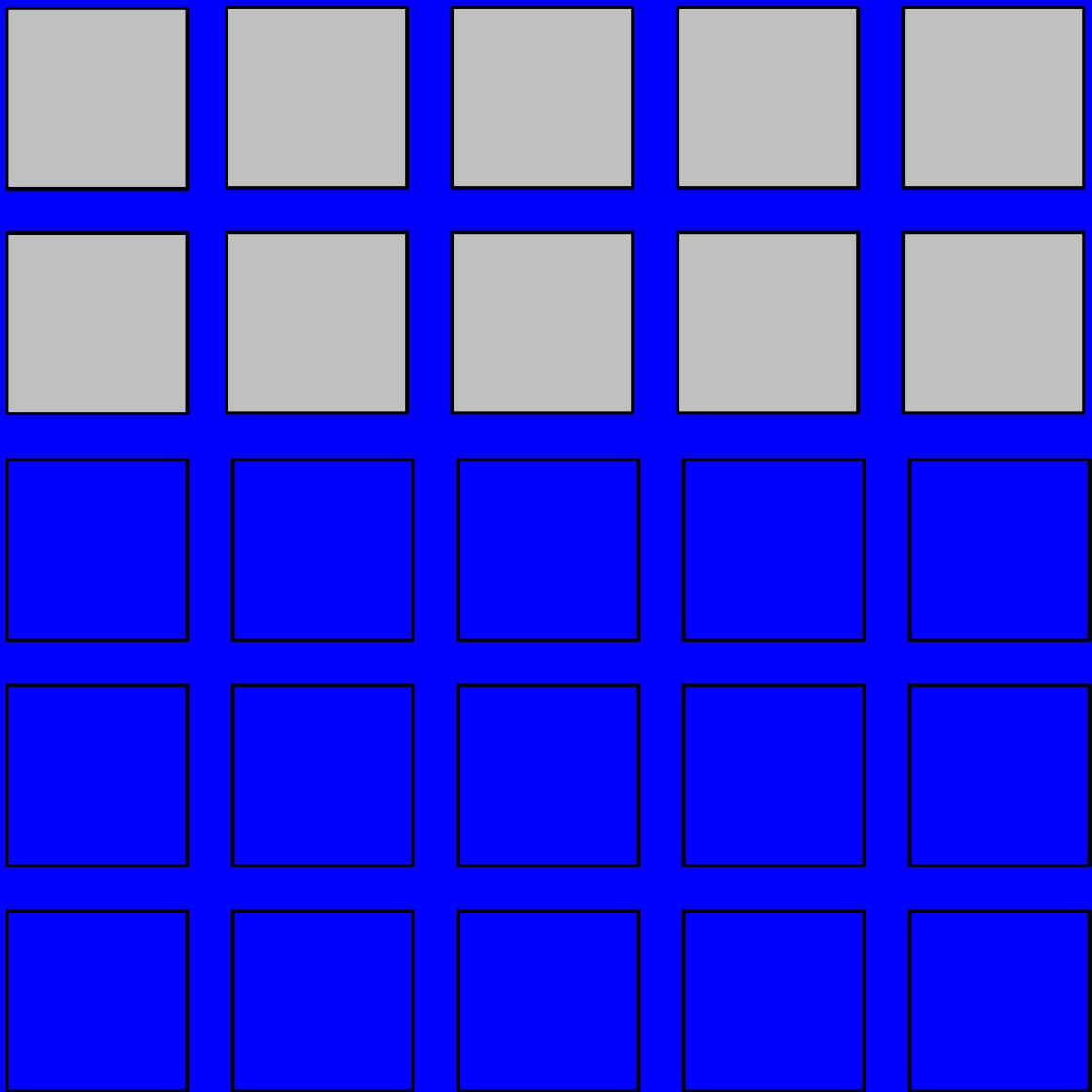


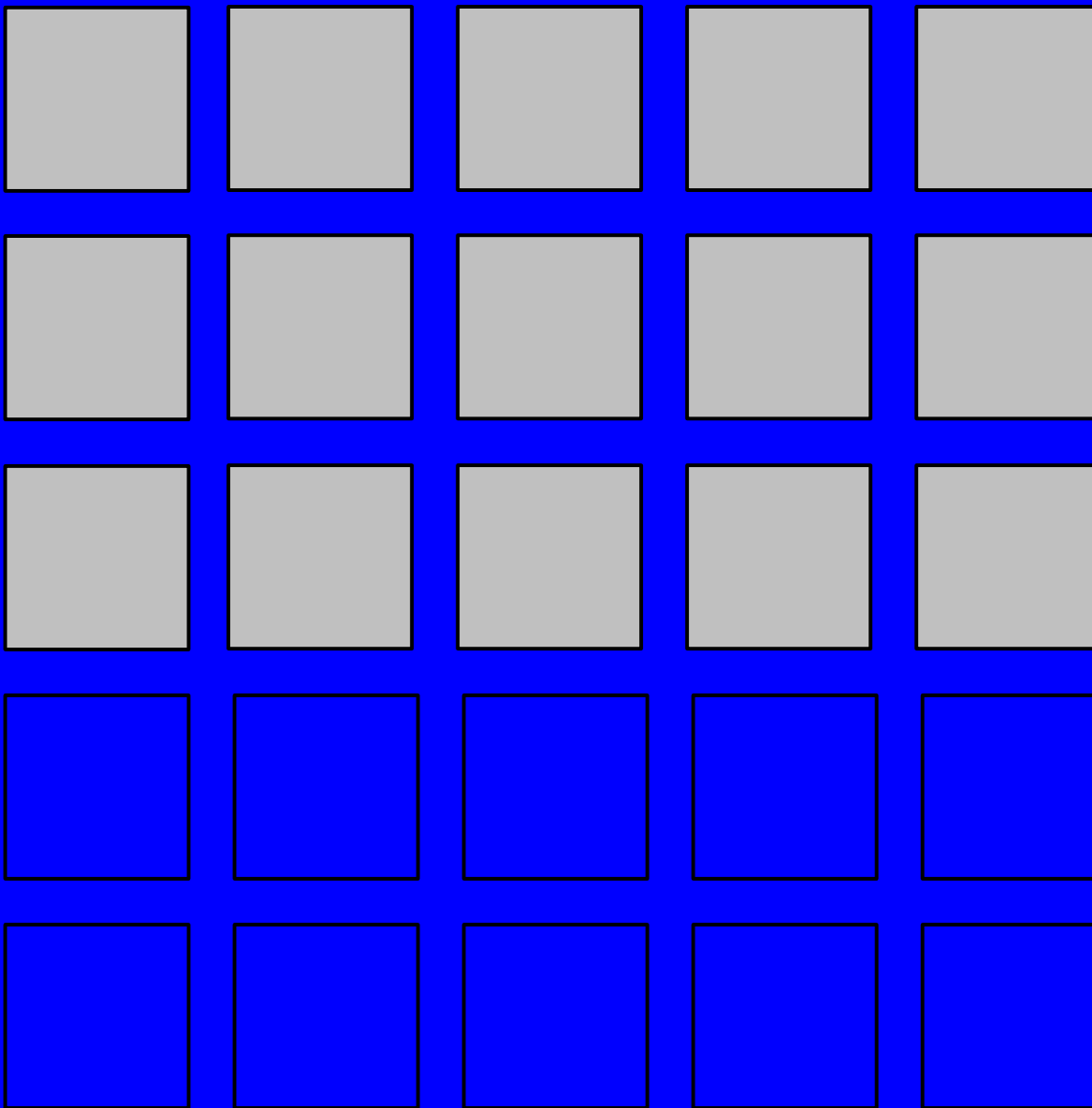


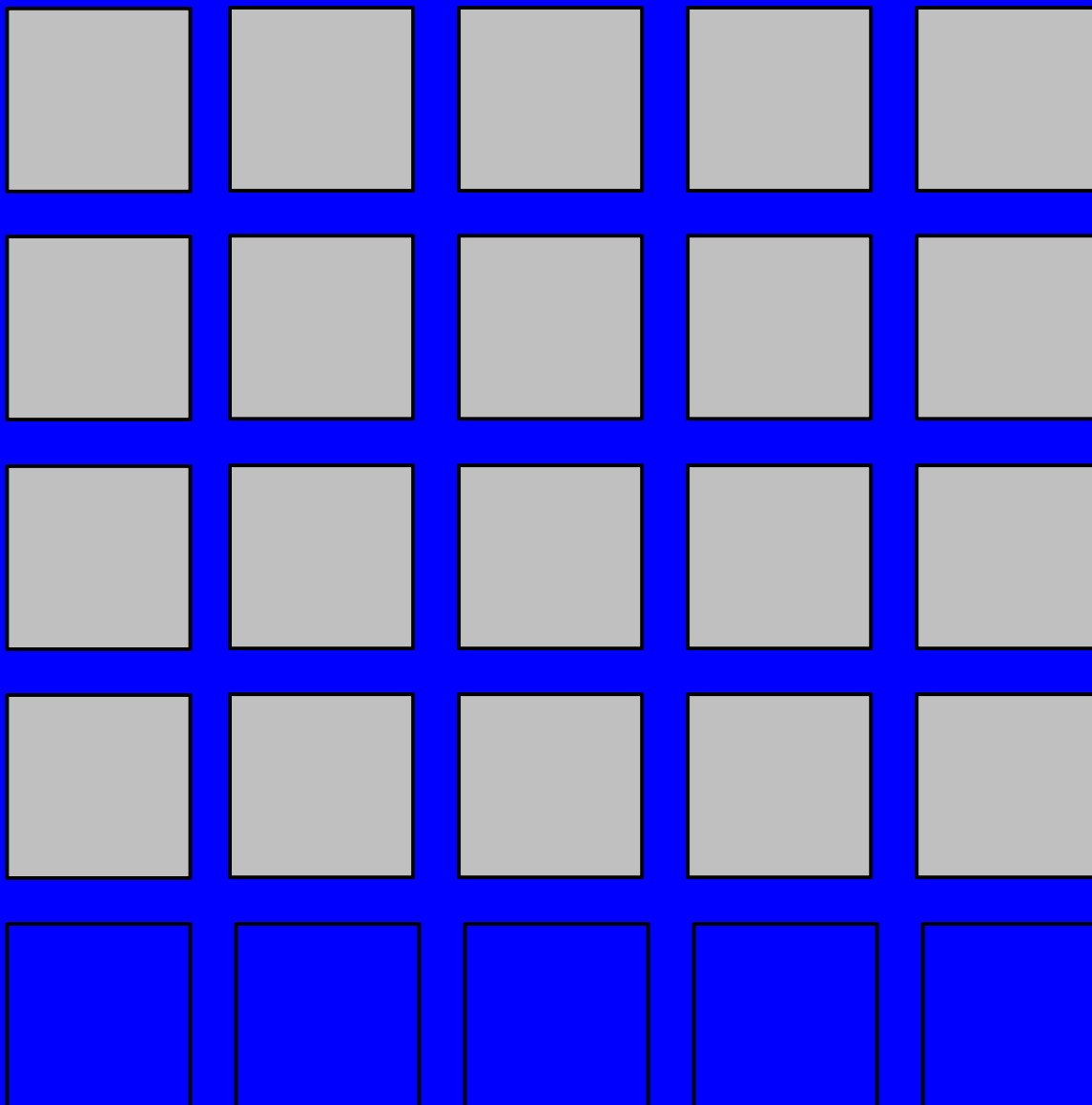


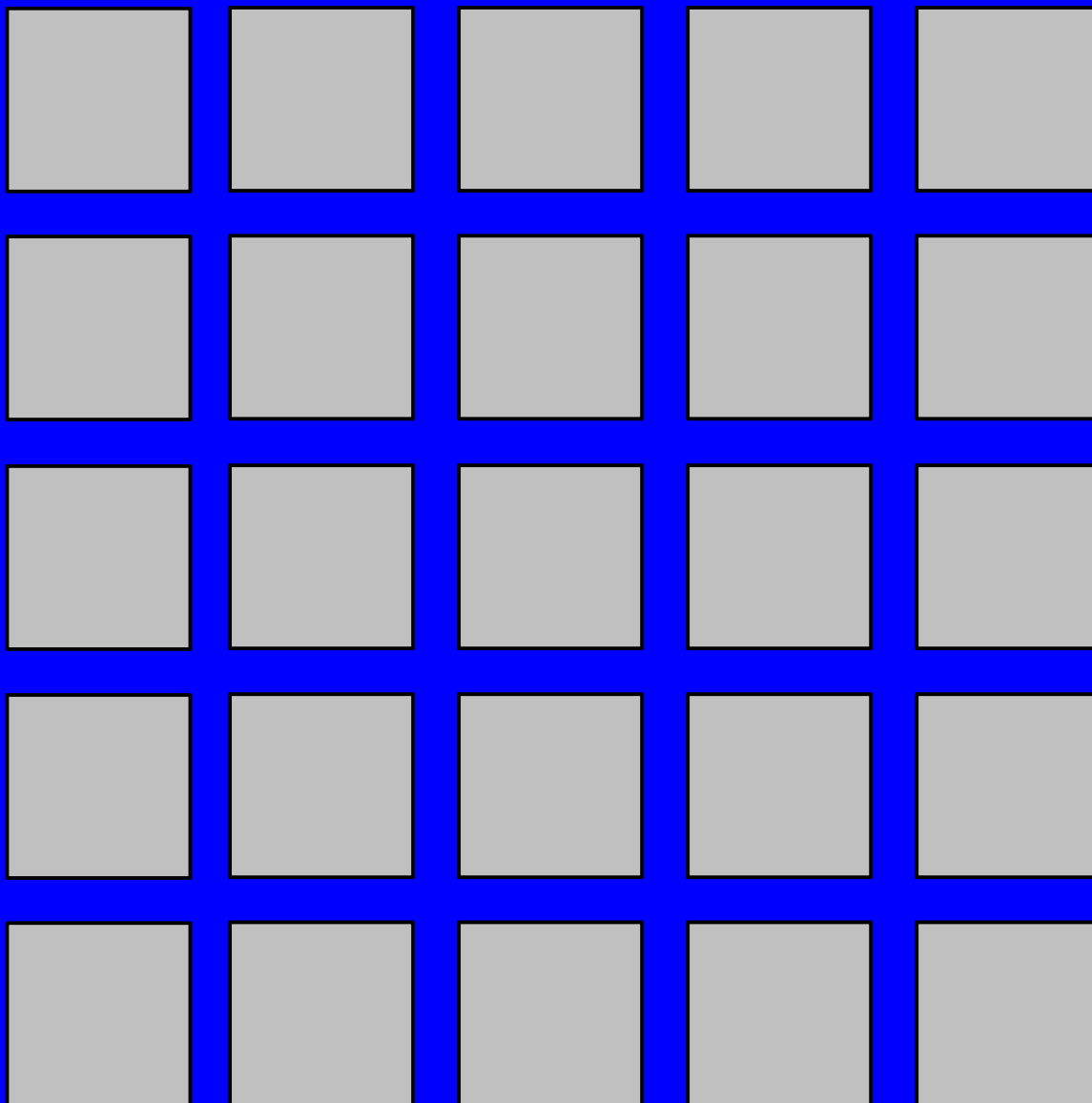












Medusa Supersequence

1. Extend to the east
2. If you made any progress, goto 1
3. Extend to the south *in parallel*
4. If you made any progress, goto 3

This can be a very efficient way to configure a large array of elements

- $O(n^2)$ elements in $O(n)$ steps
- Can extend to 3-D...final 2-D sheet configures a second 2-D sheet in one step
- Can adjust granularity ($K \times L$ sub-regions)
- General parallelizing scheme:
e.g. can also do testing/analysis in parallel

Startup/Configuration Time

10^{24} elements

1 THz clock

Startup/Configuration Time

10^{24} elements

1 THz clock

FPGA	1,000,000,000,000 seconds
------	---------------------------

Startup/Configuration Time
 10^{24} elements
1 THz clock

FPGA

31,709 Years

Startup/Configuration Time
 10^{24} elements
1 THz clock

FPGA	31,709 Years
2-D, Self-Configurable	1 Second

Startup/Configuration Time

10^{24} elements

1 THz clock

FPGA	31,709 Years
2-D, Self-Configurable	1 Second
3-D, Self-Configurable	100 μ Sec

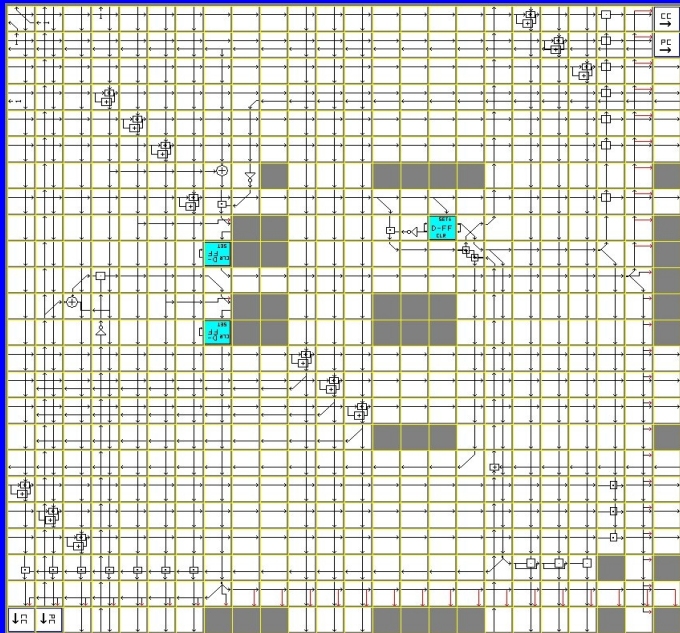
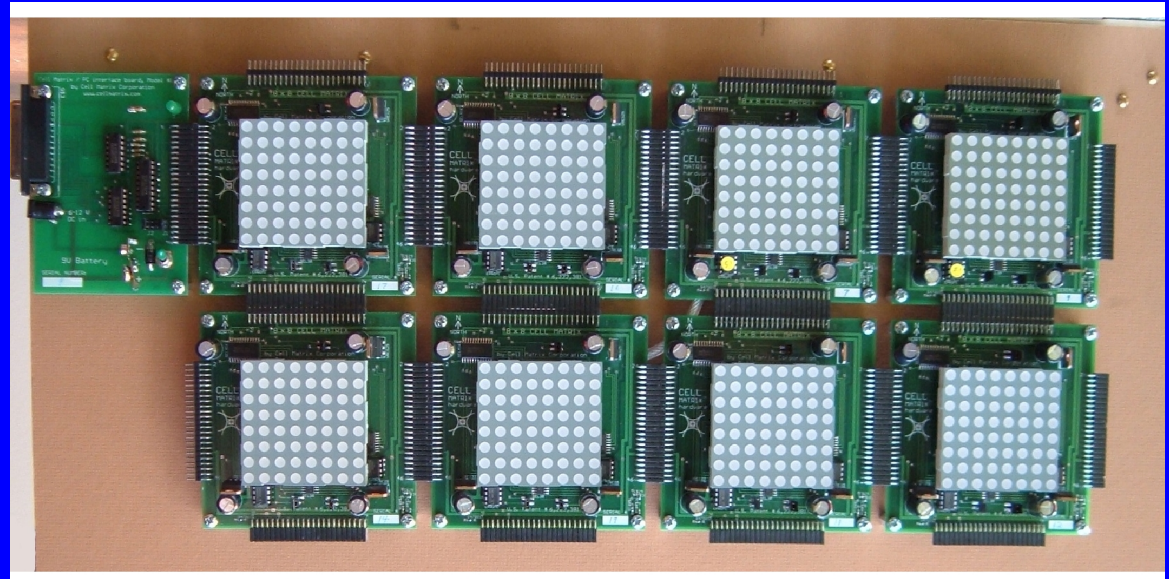
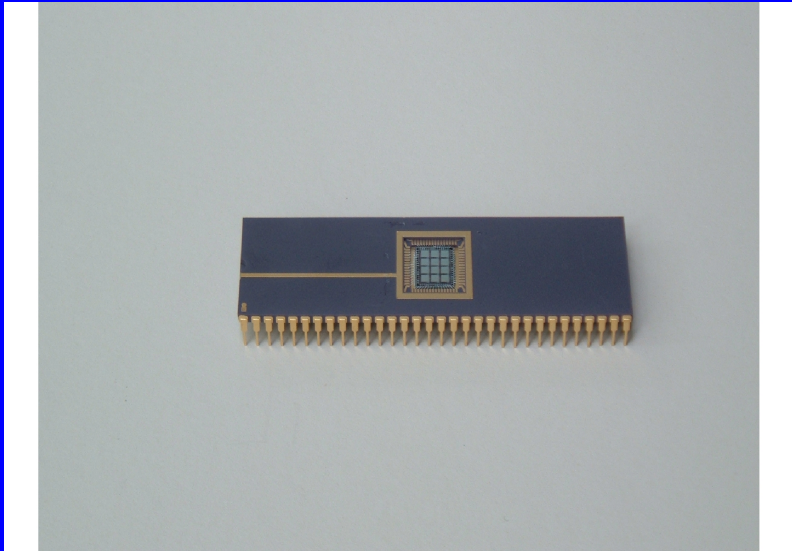
We're basically extending the engineering design space

- No longer building circuits that only process data
- Can build circuits that, in some sense, process *other circuits*

Very useful

- parallel configuration/fault detection/avoidance
- self-replicating circuitry
- scrubbing of circuits to correct runtime upsets
- process driver
- correction of manufacturing defects

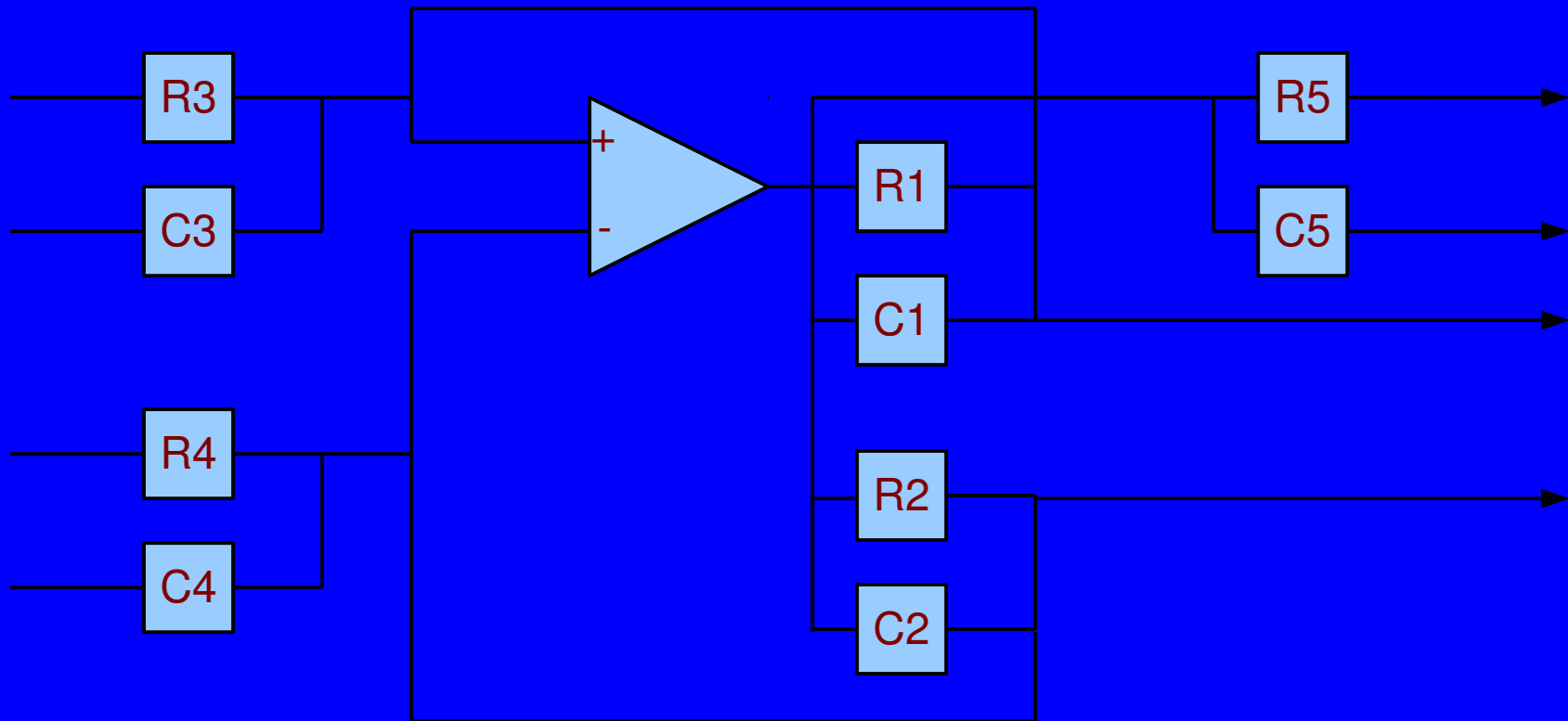
This is Generation I



The world isn't binary...

- Can you make an analog version of this?

Generation II?



Truth Table stores (digitized) values for R1-R5, C1-C5

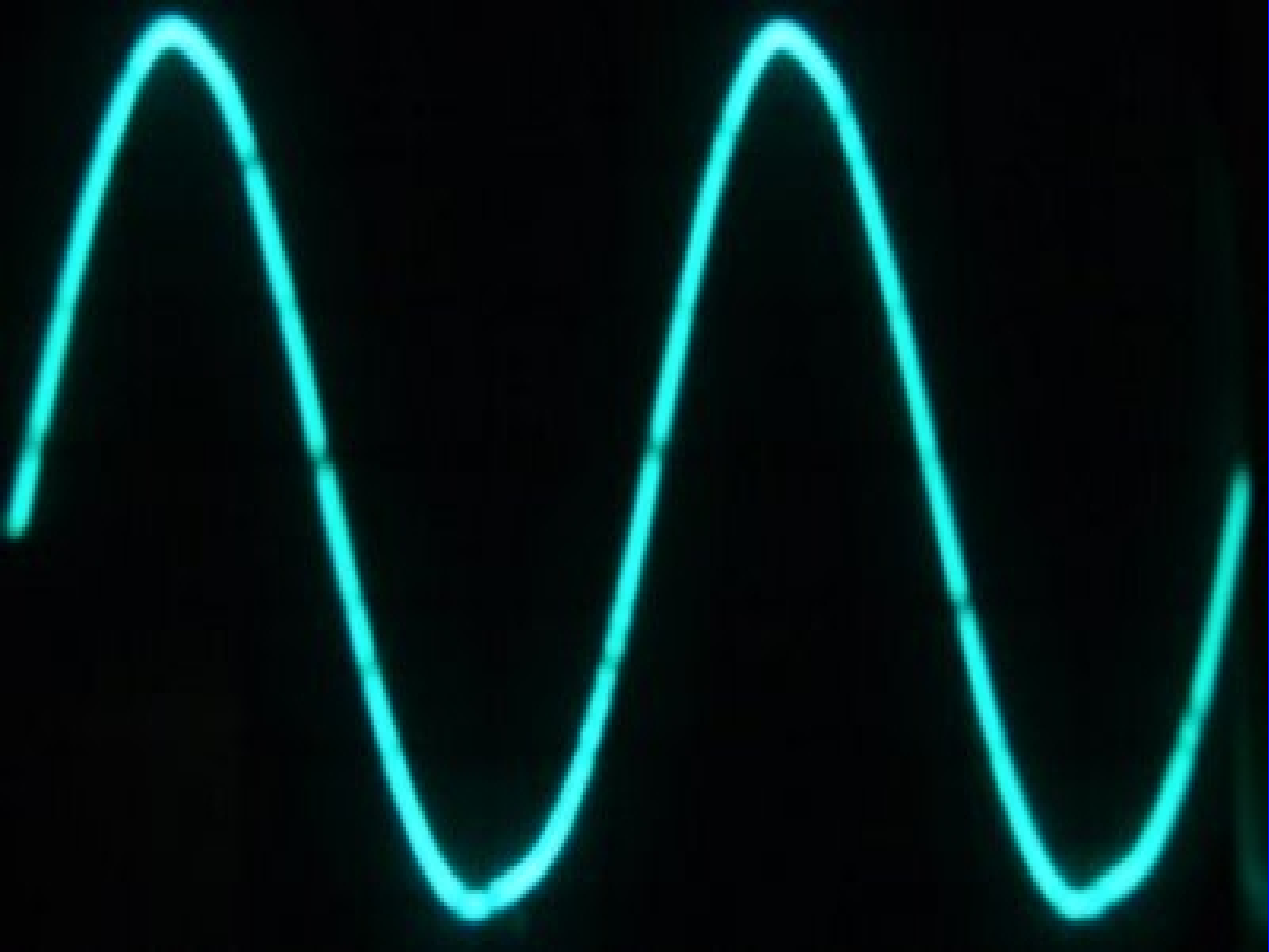
GENERATION III

Desert Island Question

- How do you turn a scanner into a scientific calculator?

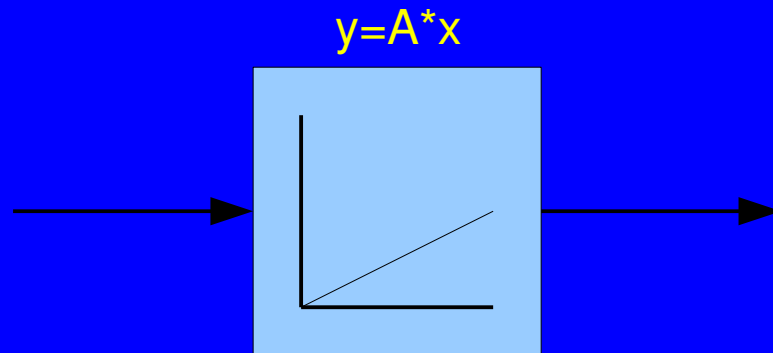
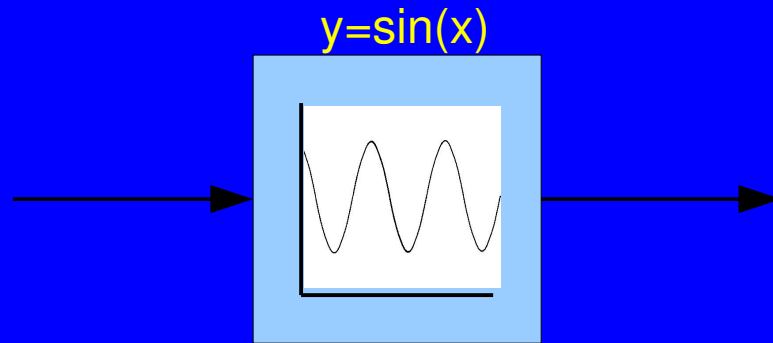
Desert Island Question

- How do you turn a scanner into a scientific calculator?
- ANSWER:
 - Cards with pictures of $y=\sin(x)$, $y=\exp(x)$, etc.
 - Put picture on scanner
 - Move scan mechanism to location “x”
 - Scan
 - See which scan element picks up a dot



**What if we store a function
 $y=f(x)$ inside a cell?**

What if we store a function $y=f(x)$ inside a cell?





How would you implement this?



C-Mode is pretty simple

- Read what's coming out of the source element
- Use that to setup the vibrations in the target element

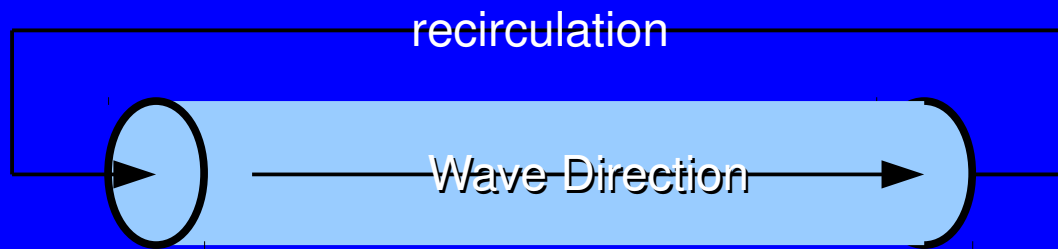
C-Mode is pretty simple

- Read what's coming out of the source element
- Use that to setup the vibrations in the target element

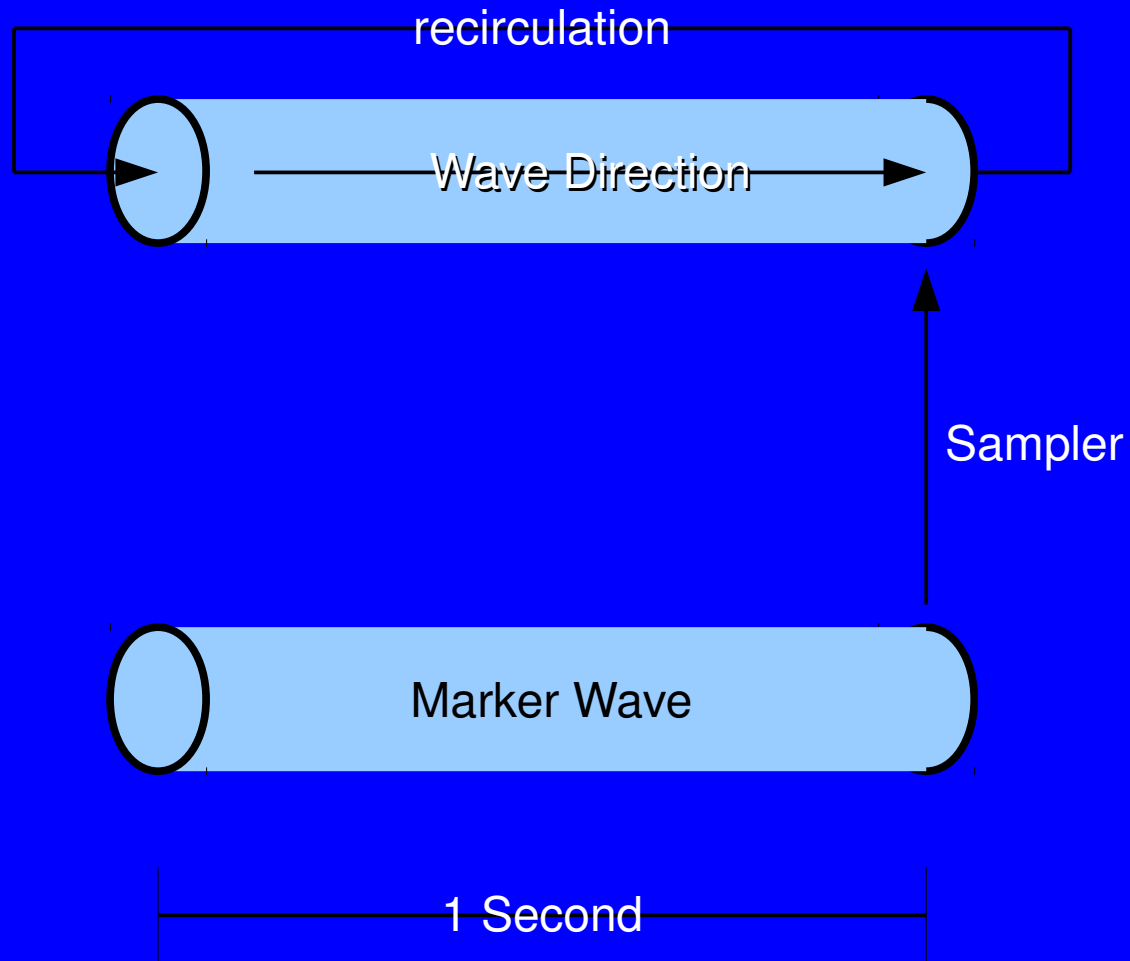
Music is just a pattern of vibrations
(isn't everything?)

**Is this like singing a song that someone else
hears and learns?**

D-Mode is trickier...

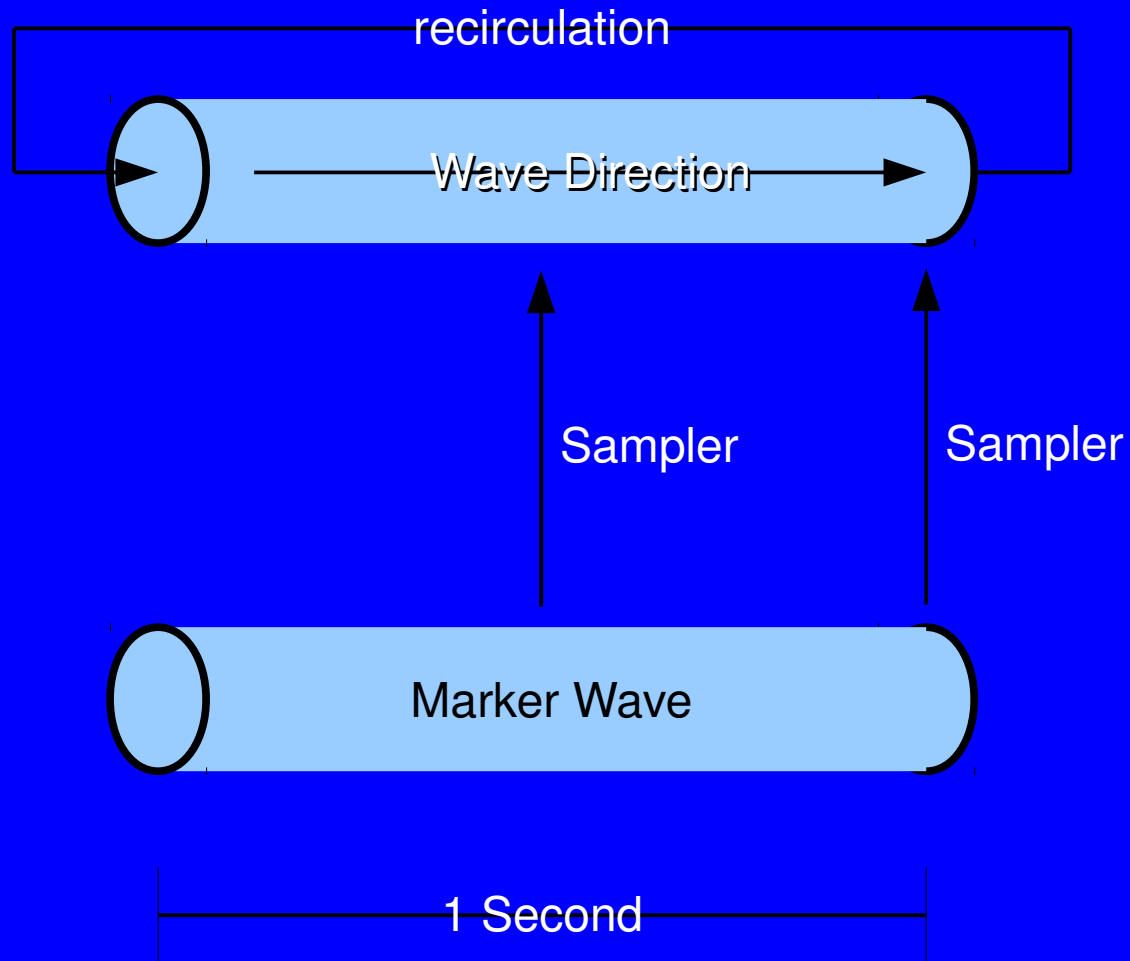


D-Mode is trickier...



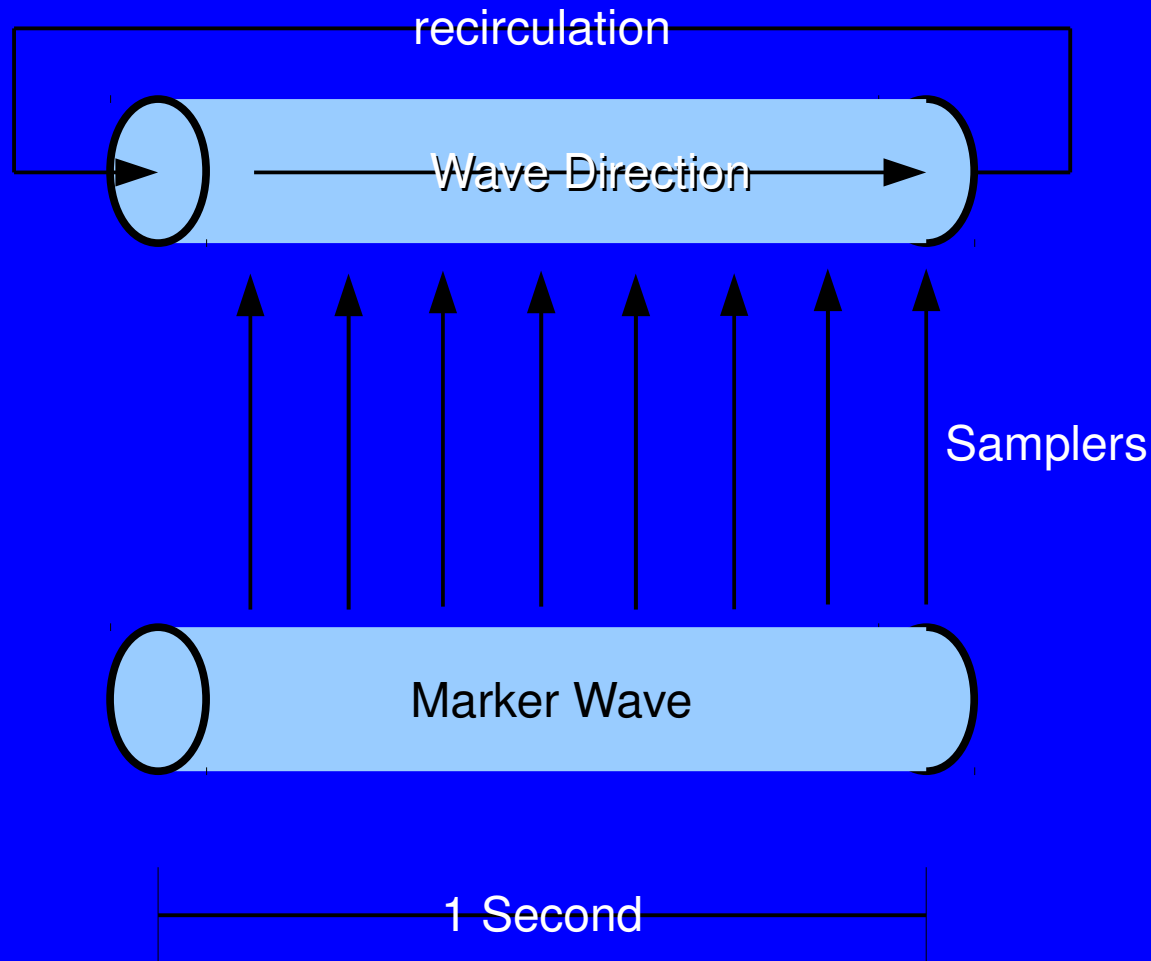
May take 1 second to read your desired value

D-Mode is trickier...



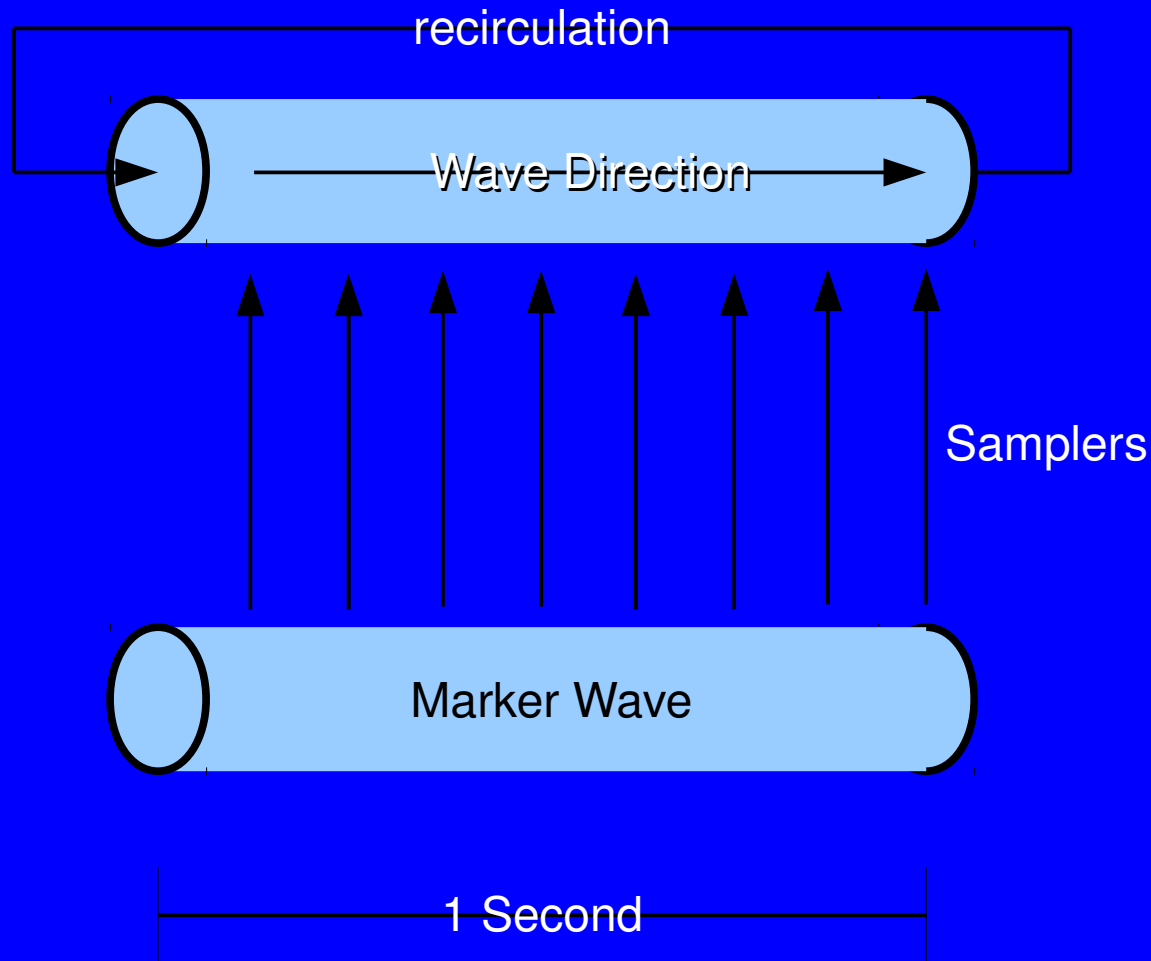
May take 1/2 second to read your desired value

D-Mode is trickier...



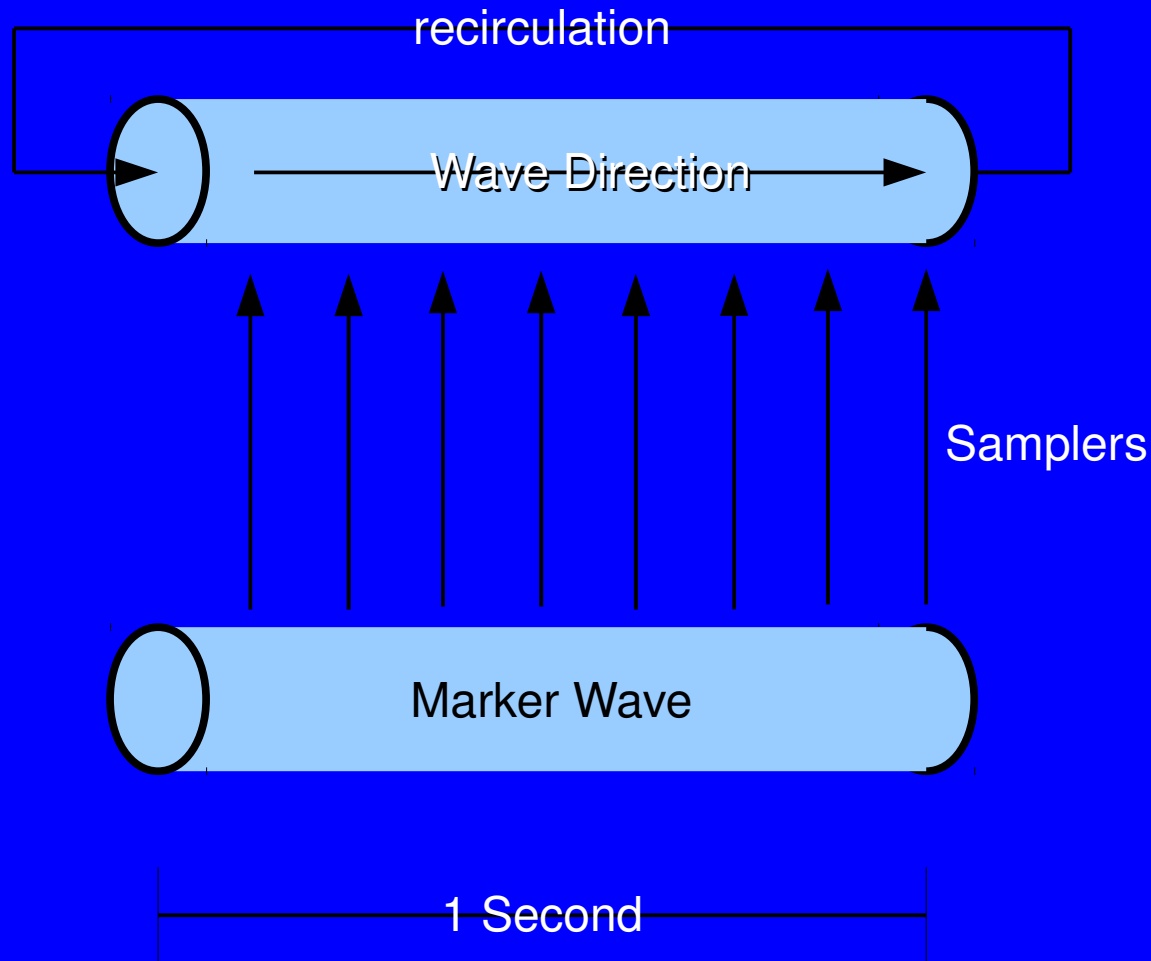
May take $1/8$ second to read your desired value

D-Mode is trickier...



May take $\frac{1}{8}$ second to read your desired value
OR: Read instantly, with up to a 12.5% error in X

D-Mode is trickier...



May take $1/8$ second to read your desired value

OR: Read instantly, with up to a 12.5% error in X

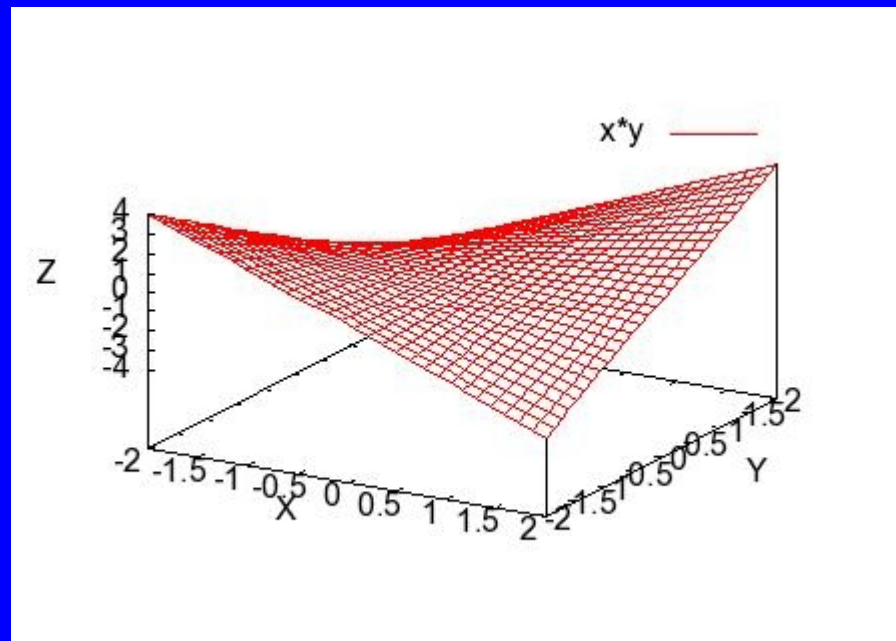
OR: Up to $1/16^{\text{th}}$ second delay and 6.25% error in X

This starts to feel a bit like

$$\Delta x \Delta p \geq \hbar/2$$

What about a 2-input/2-output device?

- 2 outputs are easy: just store 2 functions
- But how do you store $y=f(x_1, x_2)$?
- Deformable membrane?



Now C-Mode is tricky too...

- How do you read this out and copy it into another cell? (and how do you refresh it?)
- Establish a scan-order to serialize the surface?
- Space-filling curve?

Or use a static, mechanical storage system?



D-Mode gets even more complex

- Need to be able to read membrane height at point (x_1, x_2)
- 2-D traveling wave?
- 2-D scan of a fixed deformed membrane?

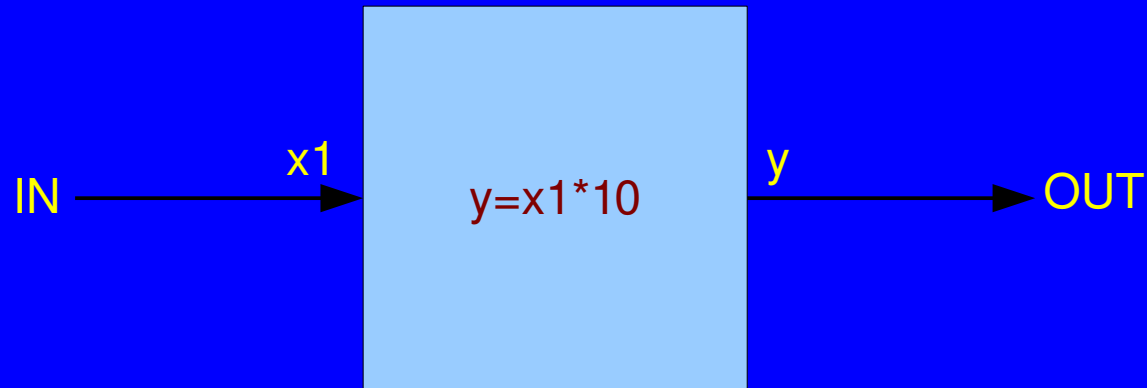
$$y=f(x_1,x_2,x_3)$$

- Example: temperature inside a 3-D region
- Better example: phase of light?
- Hologram-based storage?
- D- and C- modes both seem fairly complex here
- May suffice for 2-D Matrix

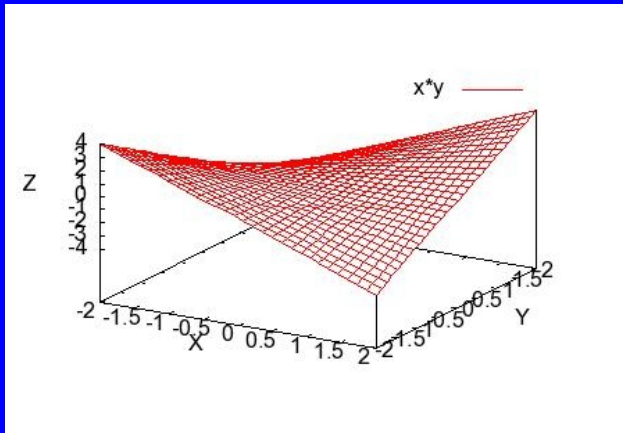
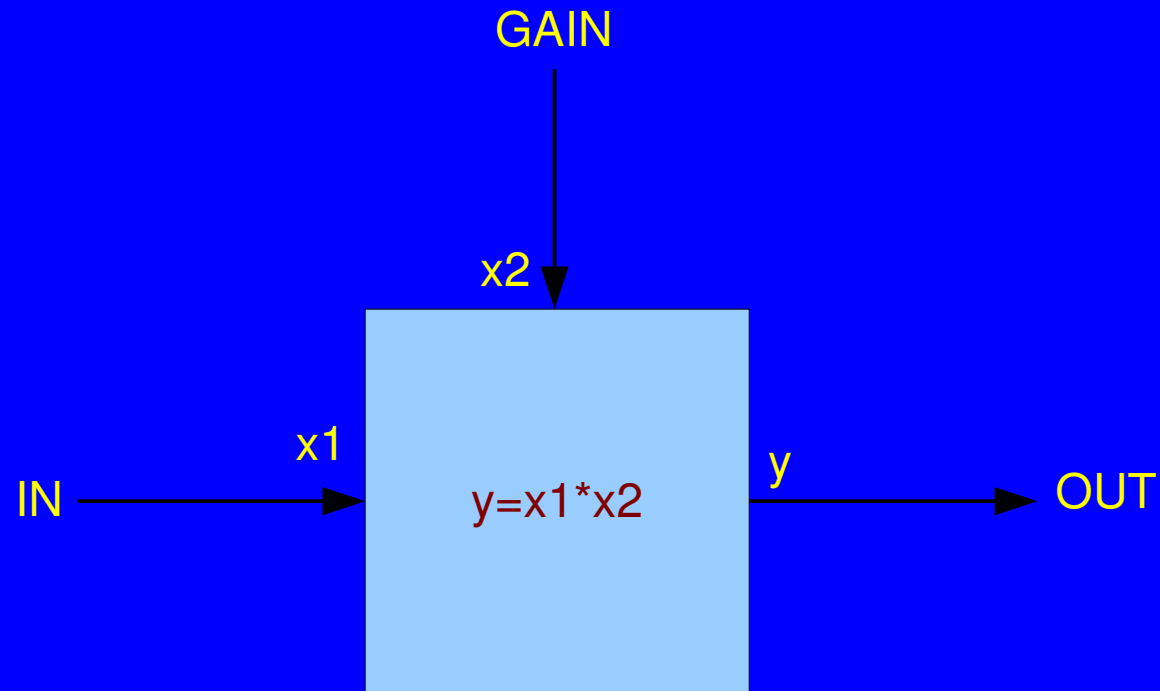
$$y=f(x_1,x_2,x_3,x_4)$$

- Ideal for 2-D Matrix
- May suffice for 3-D Matrix
- Example: change temperature at (x_1,x_2,x_3) , use time for x_4
- Better example: change phase over time (holographic movie?)
- Something else???
- “Singularity beyond the singularity”

Sample Application: Fixed-Gain Amplifier

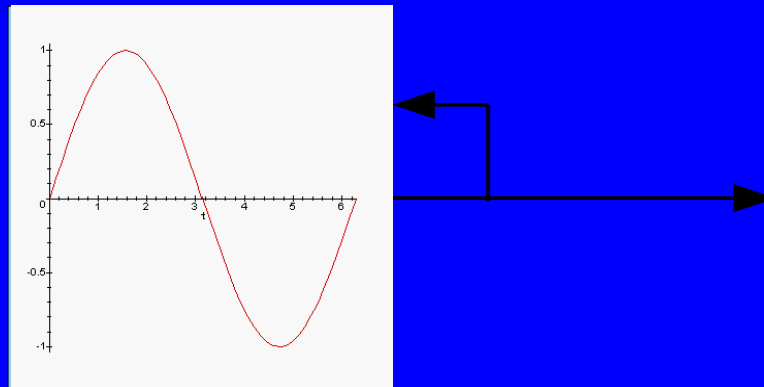


Variable-Gain Amplifier



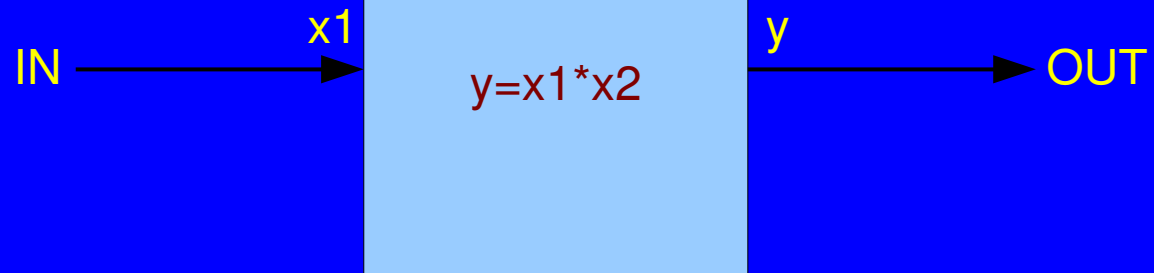
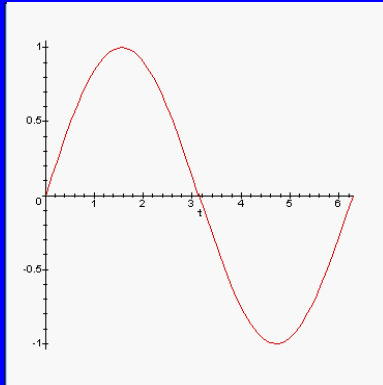
Sine Wave Generator

CELL IS IN C-MODE



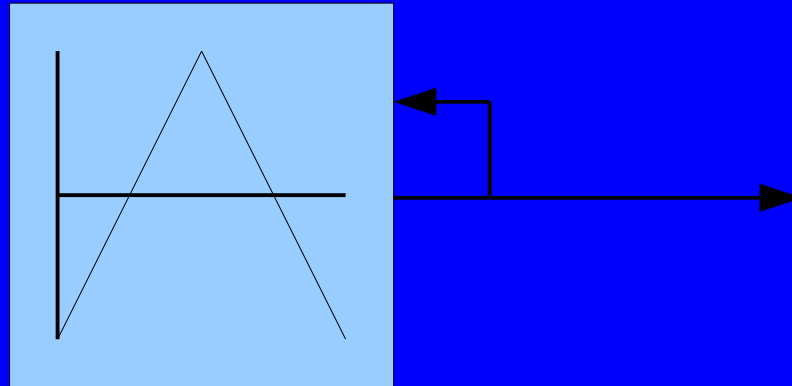
AM Encoder

CELL IS IN C-MODE



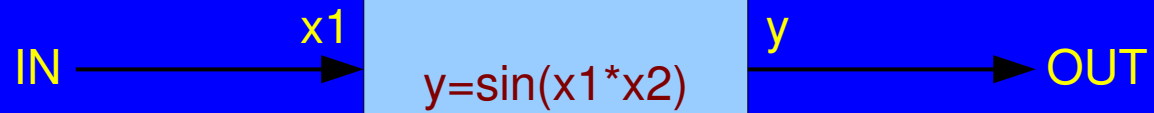
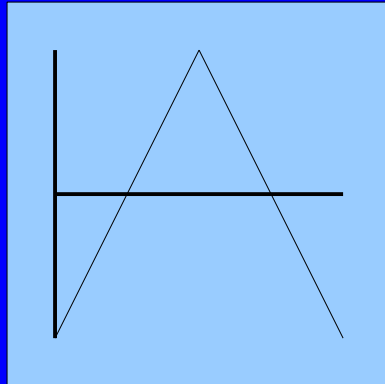
Sawtooth Generator

CELL IS IN C-MODE

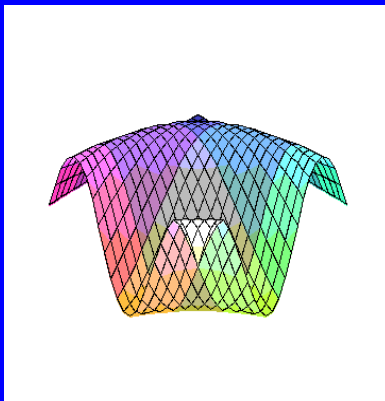


FM Encoder

CELL IS IN C-MODE

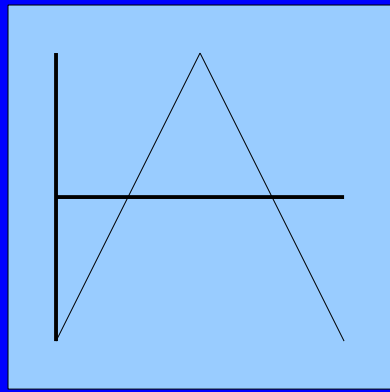


$x2$



Better AM Encoder

CELL IS IN C-MODE



x1

$$y = \sin(x1 * x2)$$

y

x1

$$y = x1 * x2$$

y

OUT

x2

BASE
FREQUENCY

x2

IN

These only use simple 2-input blocks

- Much broader scope of applications with 3- and 4-input devices
- Very different from DSP
- More in-tune with the real world?

Are we re-inventing music?

- Aborigines: Dreamtime/Creation
- World was sung into existence



Are we re-inventing music?

- Aborigines: Dreamtime/Creation
- World was sung into existence
- Passing on of these songs: a bit like C-Mode?

Are we re-inventing music?

- Aborigines: Dreamtime/Creation
- World was sung into existence
- Passing on of these songs: a bit like C-Mode?
- Songlines: playback of creation songs
- Information about the land is extracted from the songs

Are we re-inventing music?

- Aborigines: Dreamtime/Creation
- World was sung into existence
- Passing on of these songs: a bit like C-Mode?
- Songlines: playback of creation songs
- Information about the land is extracted from the songs
- A bit like D-mode?

Is the singularity after the
singularity actually 40,000
years old???

Is the singularity after the
singularity actually 40,000
years old???

For more information:
Cell Matrix Corporation
<http://www.cellmatrix.com>
nick4@cellmatrix.com
(540) 446-0692

For more information...

Cell Matrix Corporation

<http://www.cellmatrix.com>

nick4@cellmatrix.com

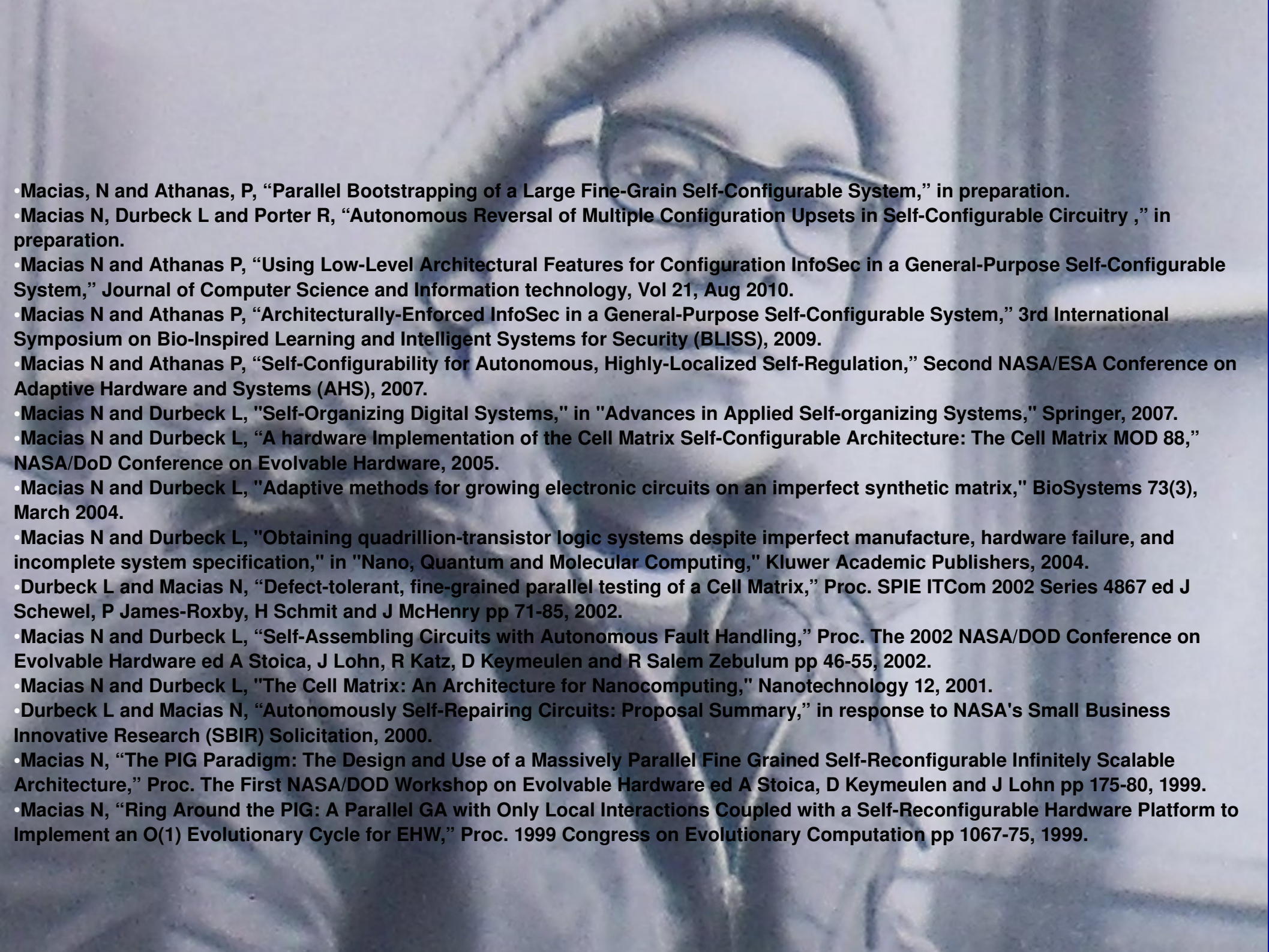
(540) 446-0692

SUMMARY/CONCLUSIONS

- Architecture addresses H1-H4
- Successful management of VLRS
- Internal configuration useful for fast bootstrap, including defect detection and avoidance
- Parallel test/parallel config
- In-vivo implementation is supported
- Sample problem analyzed, including 3-D sim
- Utility of thin 3-D array discovered

FUTURE WORK

- Simulation
- Design tools
- Representation of dynamic circuitry
- Analog version
- Self-assembly
- A-matter
- Continuous TTs

- 
- Macias, N and Athanas, P, "Parallel Bootstrapping of a Large Fine-Grain Self-Configurable System," in preparation.
 - Macias N, Durbeck L and Porter R, "Autonomous Reversal of Multiple Configuration Upsets in Self-Configurable Circuitry ," in preparation.
 - Macias N and Athanas P, "Using Low-Level Architectural Features for Configuration InfoSec in a General-Purpose Self-Configurable System," Journal of Computer Science and Information technology, Vol 21, Aug 2010.
 - Macias N and Athanas P, "Architecturally-Enforced InfoSec in a General-Purpose Self-Configurable System," 3rd International Symposium on Bio-Inspired Learning and Intelligent Systems for Security (BLISS), 2009.
 - Macias N and Athanas P, "Self-Configurability for Autonomous, Highly-Localized Self-Regulation," Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2007.
 - Macias N and Durbeck L, "Self-Organizing Digital Systems," in "Advances in Applied Self-organizing Systems," Springer, 2007.
 - Macias N and Durbeck L, "A hardware Implementation of the Cell Matrix Self-Configurable Architecture: The Cell Matrix MOD 88," NASA/DoD Conference on Evolvable Hardware, 2005.
 - Macias N and Durbeck L, "Adaptive methods for growing electronic circuits on an imperfect synthetic matrix," BioSystems 73(3), March 2004.
 - Macias N and Durbeck L, "Obtaining quadrillion-transistor logic systems despite imperfect manufacture, hardware failure, and incomplete system specification," in "Nano, Quantum and Molecular Computing," Kluwer Academic Publishers, 2004.
 - Durbeck L and Macias N, "Defect-tolerant, fine-grained parallel testing of a Cell Matrix," Proc. SPIE ITCOM 2002 Series 4867 ed J Schewel, P James-Roxby, H Schmit and J McHenry pp 71-85, 2002.
 - Macias N and Durbeck L, "Self-Assembling Circuits with Autonomous Fault Handling," Proc. The 2002 NASA/DOD Conference on Evolvable Hardware ed A Stoica, J Lohn, R Katz, D Keymeulen and R Salem Zebulum pp 46-55, 2002.
 - Macias N and Durbeck L, "The Cell Matrix: An Architecture for Nanocomputing," Nanotechnology 12, 2001.
 - Durbeck L and Macias N, "Autonomously Self-Repairing Circuits: Proposal Summary," in response to NASA's Small Business Innovative Research (SBIR) Solicitation, 2000.
 - Macias N, "The PIG Paradigm: The Design and Use of a Massively Parallel Fine Grained Self-Reconfigurable Infinitely Scalable Architecture," Proc. The First NASA/DOD Workshop on Evolvable Hardware ed A Stoica, D Keymeulen and J Lohn pp 175-80, 1999.
 - Macias N, "Ring Around the PIG: A Parallel GA with Only Local Interactions Coupled with a Self-Reconfigurable Hardware Platform to Implement an O(1) Evolutionary Cycle for EHW," Proc. 1999 Congress on Evolutionary Computation pp 1067-75, 1999.

Avogadro-Scale System

- reconfigurable element: 10,000 transistors
- “smart switch”/cell: 10^6 elements
- “organism”: 10^{14} cells = 10^{24} transistors

Avogadro-Scale System

- reconfigurable element: 10,000 transistors
- “smart switch”/cell: 10^6 elements
- “organism”: 10^{14} cells= 10^{24} transistors
- 15 orders of magnitude increase over today
- Optimistic estimate: $1.5 \cdot \log_2(10^{15}) = 75$ years

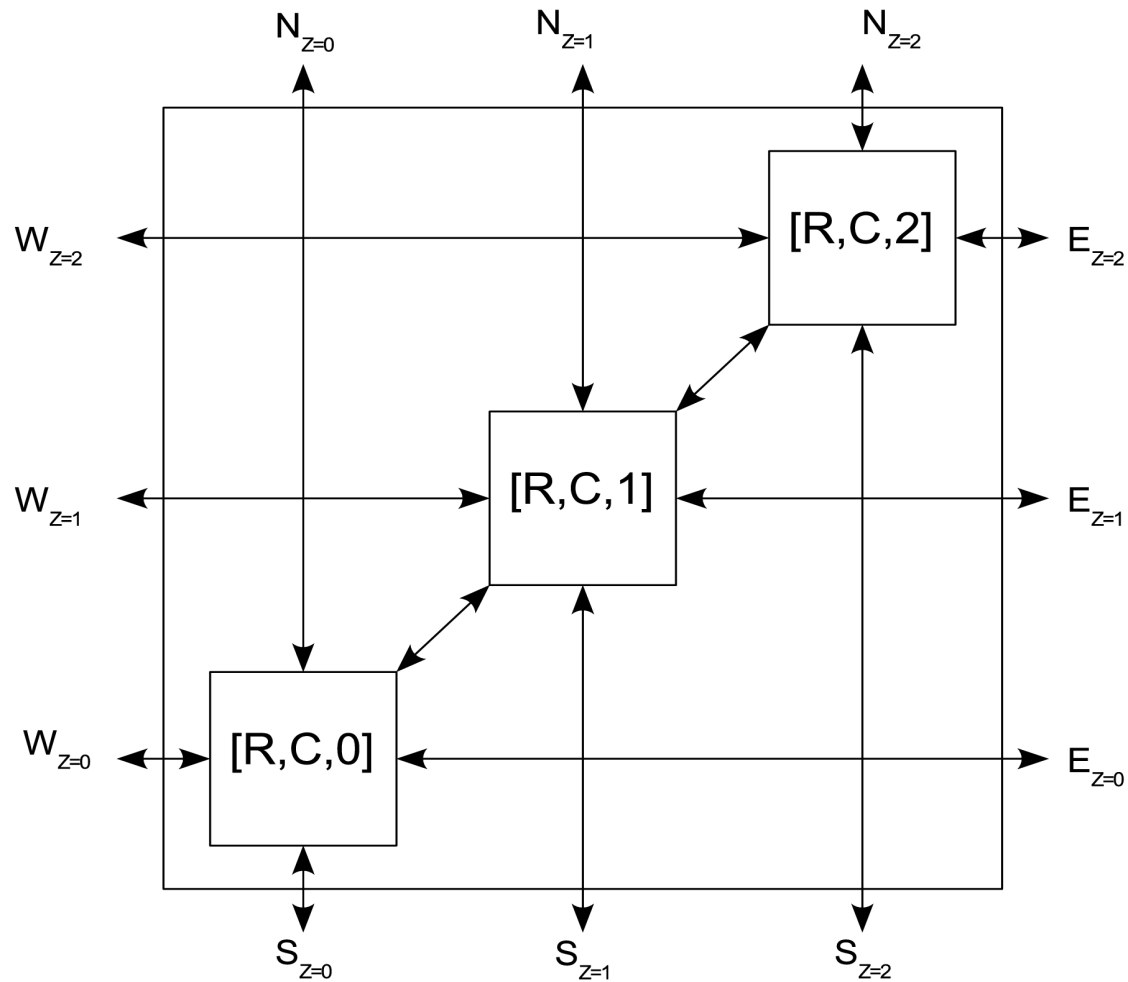
Traditional Criteria

- Smaller transistors
this is, in some sense, **the** key
- Faster switching speeds
- Lower-power transistors
- Yield improvement

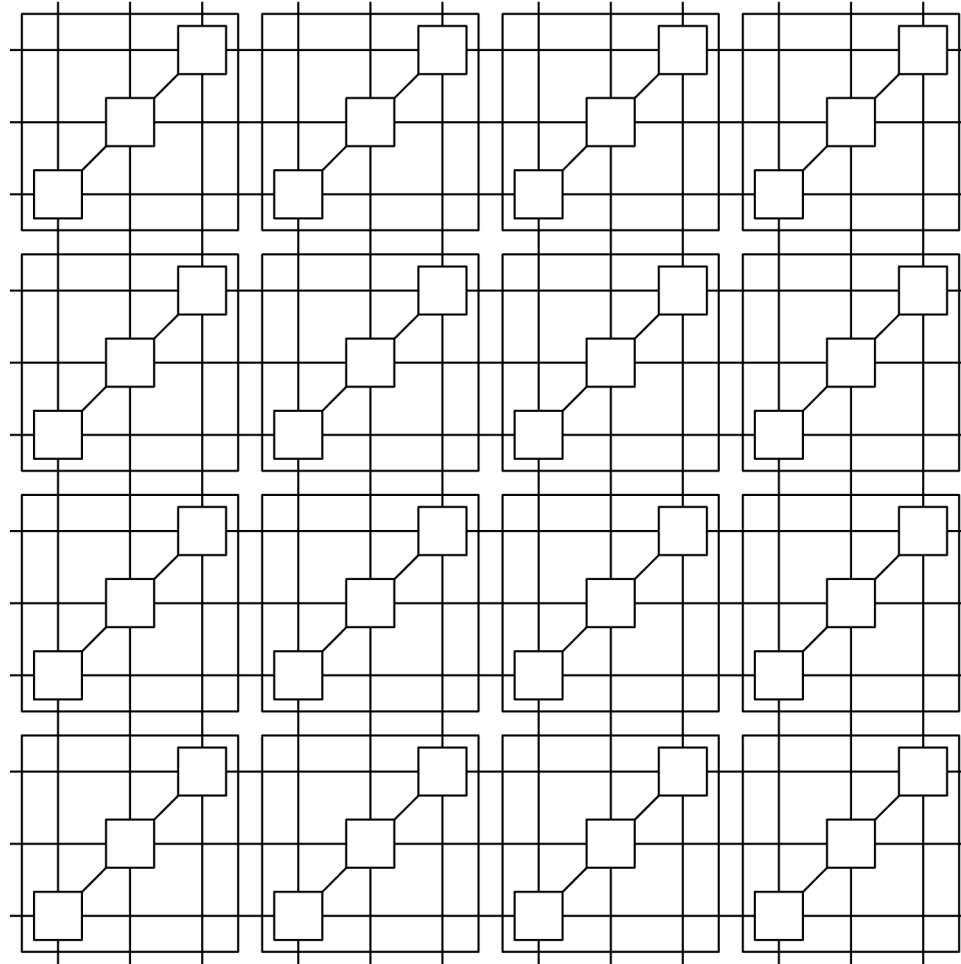
Modified Criteria

- Smaller transistors – important, but **not sufficient**
- Slower switches/greater parallelism
- Low clock frequency -> low power
- Defects can be tolerated post-manufacture

“Thin” 3-D (2.5-D)



Still scalable with 2-D assembly



Order of Cross-Sectional Bandwidth

Scalar	1
Multicore	$\min(\# \text{ Cores, Network BW})$
Cluster	$\min(\# \text{ Nodes, Network BW})$
FPGA	10^{12}
3-D Self-Configurable	10^{16}