

# Abstraction Layers for Scalable Microfluidic Biocomputers

William Thies\*, J.P. Urbanski†,  
Todd Thorsen† and Saman Amarasinghe\*

---

\* Computer Science and Artificial Intelligence Laboratory

†Hatsopoulos Microfluids Laboratory

Massachusetts Institute of Technology

# Biocomputing Demands Complex Protocols

- **Complex biology experiments have many hazards**
  - Time-consuming laboratory protocols
  - Risk of human error
  - Expensive reagents and equipment
- **Biocomputing: complexity grows with problem size**
  - 20-variable 3-SAT problem: >96 hours to complete [[Adleman02](#)]
  - Larger instances will be even more challenging
  - Need a scalable approach
- **Our approach:**  
**Write protocols as programs, run on microfluidic chips**
  - Write once, run anywhere
  - This talk: how do you “program” a biological protocol?

# Microfluidic Chips

- **Idea: a whole biology lab on a single chip**

- Input/output
- **Sensors:** luminescence, pH, glucose, etc.
- **Actuators:** mixing, PCR, electrophoresis, cell lysis, etc.

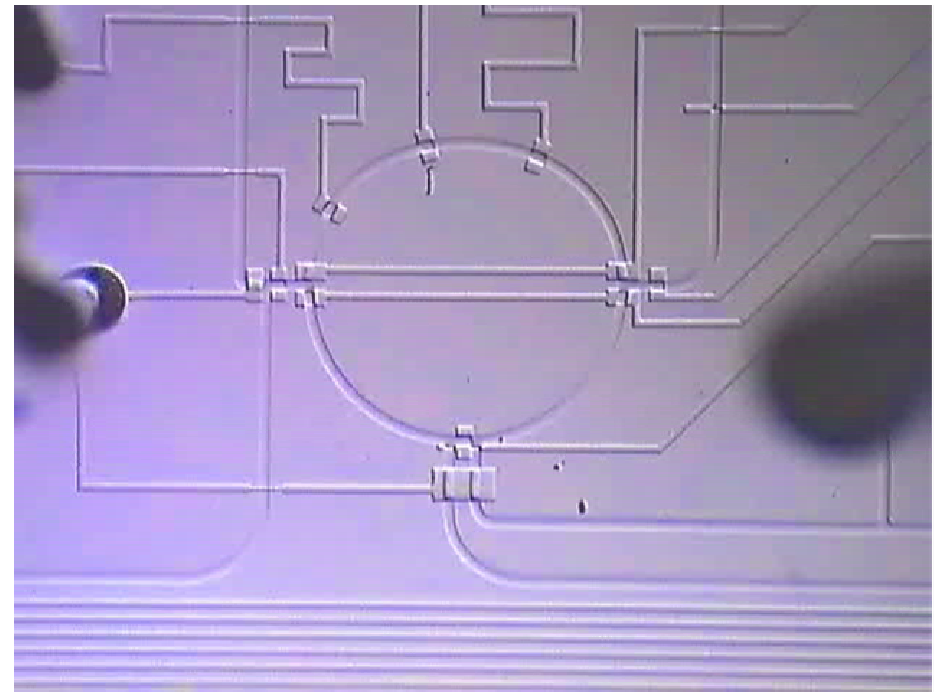
- **Benefits:**

- Small sample volumes
- High throughput
- Geometrical manipulation

- **Applications:**

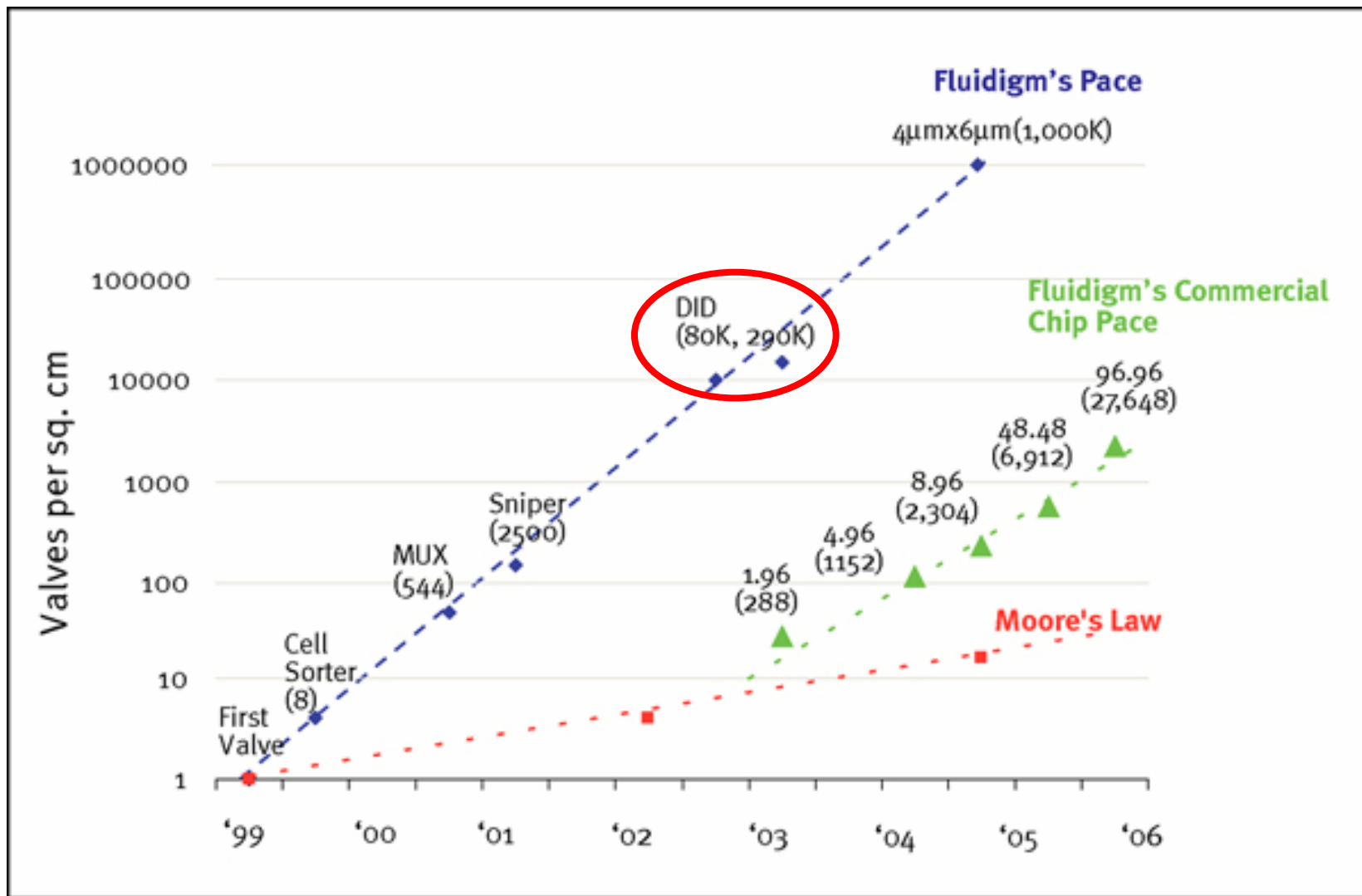
- Biochemistry - Cell biology
- Biological computing

[Livstone/Landweber] [van Noort] [Grover/Mathies] [McCaskill]  
[Gehani/Reif] [Farfel/Stefanovic] [Somei/Kaneda/Fujii/Murata]



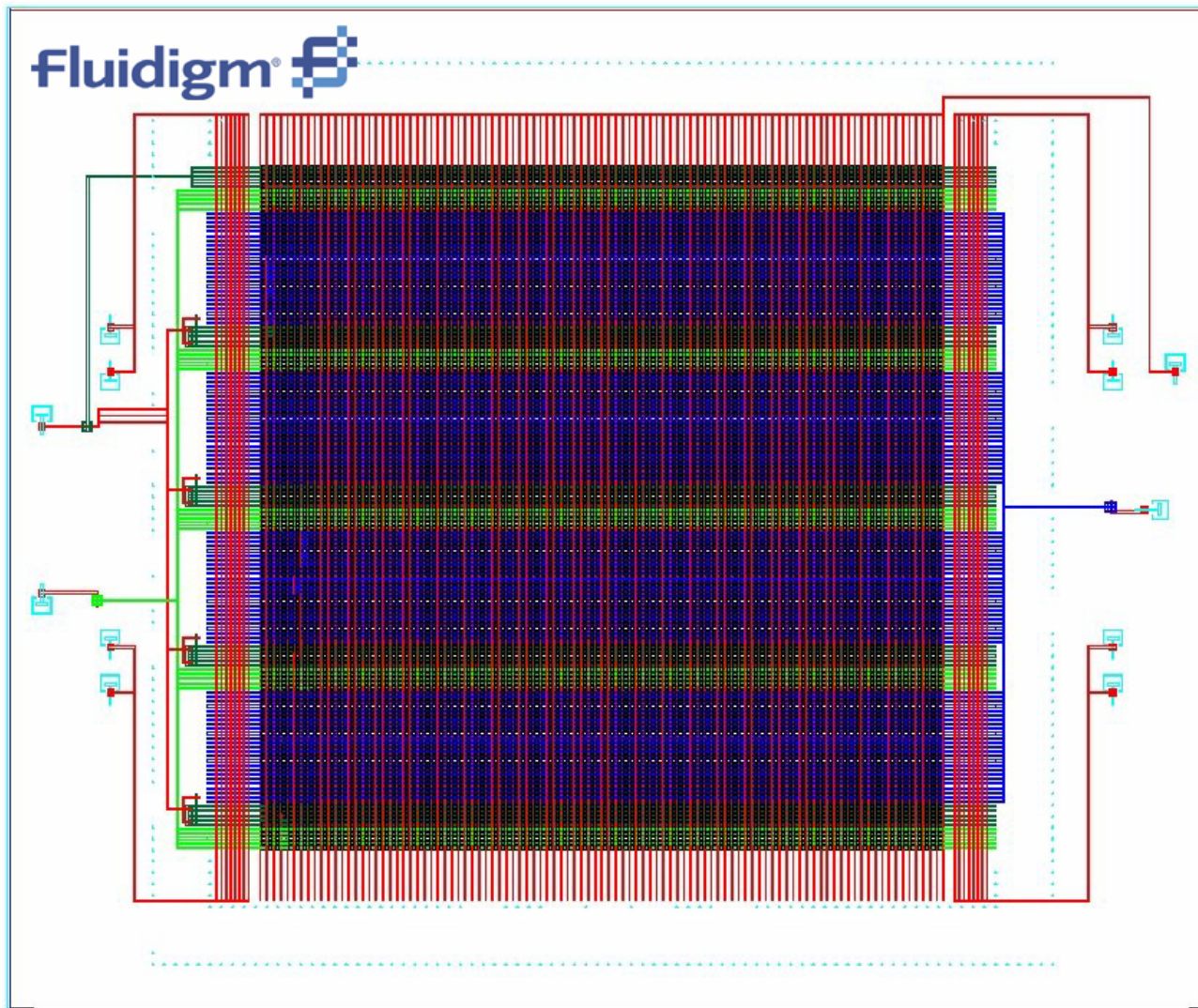
1 mm 10x real-time

# Moore's Law of Microfluidics: Valve Density Doubles Every 4 Months



Source: Fluidigm Corporation ([http://www.fluidigm.com/images/mlaw\\_lg.jpg](http://www.fluidigm.com/images/mlaw_lg.jpg))

# Moore's Law of Microfluidics: Valve Density Doubles Every 4 Months



Source: Fluidigm Corporation (<http://www.fluidigm.com/didIFC.htm>)

# How to Conduct Experiments?

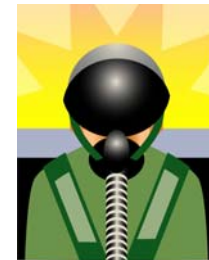


Human director ↔ Lab bench

Lab-on-a-chip

- **Two choices:**

- Design custom microfluidic chip to automatically perform your experiment
- Orchestrate every valve operation by hand (e.g., using Labview)





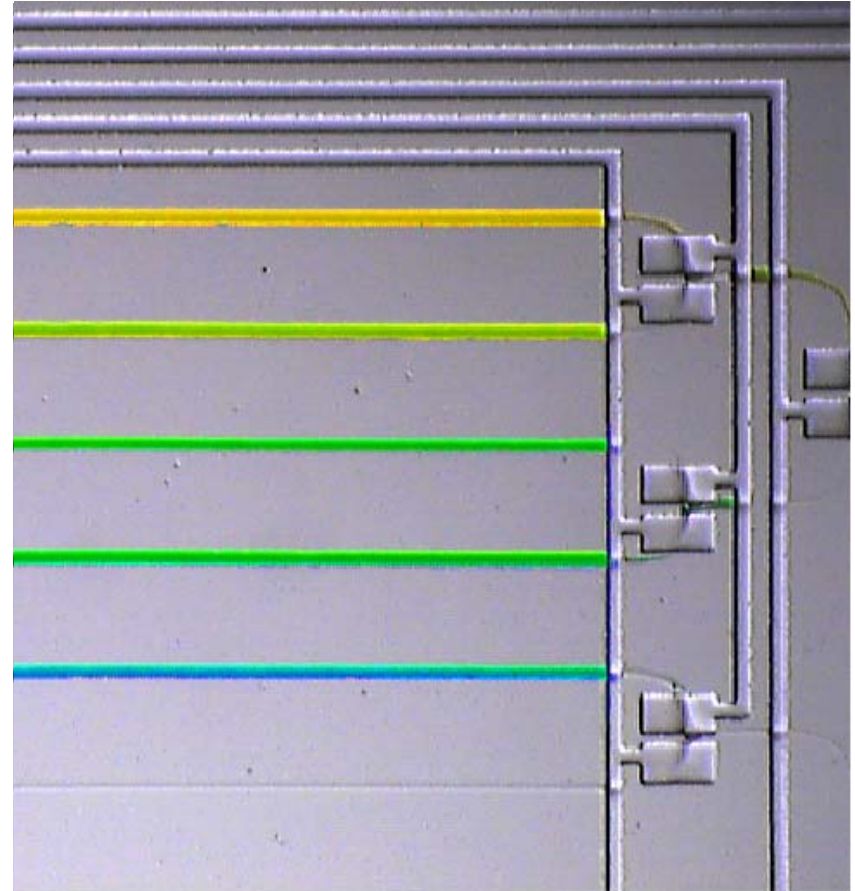
# Programmable Solution

- **Example: Gradient generation**

```
Fluid yellow = input (0);  
Fluid blue = input(1);  
for (int i=0; i<=4; i++) {  
    mix(yellow, 1-i/4, blue, i/4);  
}
```

- **Hidden from programmer:**

- Location of fluids
- Details of mixing, I/O
- Logic of valve control
- Timing of chip operations



*450 Valve Operations*

# Programmable Solution

- **Example: Gradient generation**

```
Fluid yellow = input (0);  
Fluid blue = input(1);  
for (int i=0; i<=4; i++) {  
    mix(yellow, 1-i/4, blue, i/4);  
}
```



```
setValve(0, HIGH); setValve(1, HIGH);  
setValve(2, LOW); setValve(3, HIGH);  
setValve(4, LOW); setValve(5, LOW);  
setValve(6, HIGH); setValve(7, LOW);  
setValve(8, LOW); setValve(9, HIGH);  
setValve(10, LOW); setValve(11, HIGH);  
setValve(12, LOW); setValve(13, HIGH);  
setValve(14, LOW); setValve(15, HIGH);  
setValve(16, LOW); setValve(17, LOW);  
setValve(18, LOW); setValve(19, LOW);  
wait(2000);  
setValve(14, HIGH); setValve(2, LOW);  
wait(1000);  
setValve(4, HIGH); setValve(12, LOW);  
setValve(16, HIGH); setValve(18, HIGH);  
setValve(19, LOW);  
wait(2000);
```

- **Hidden from programmer:**

- Location of fluids
- Details of mixing, I/O
- Logic of valve control
- Timing of chip operations

*450 Valve Operations*



# Programmable Solution

- **Example: Gradient generation**

```
Fluid yellow = input (0);  
Fluid blue = input(1);  
for (int i=0; i<=4; i++) {  
    mix(yellow, 1-i/4, blue, i/4);  
}
```



```
wait(2000);  
setValve(14, HIGH); setValve(2, LOW);  
wait(1000);  
setValve(4, HIGH); setValve(12, LOW);  
setValve(16, HIGH); setValve(18, HIGH);  
setValve(19, LOW);  
wait(2000);  
setValve(0, LOW); setValve(1, LOW);  
setValve(2, LOW); setValve(3, HIGH);  
setValve(4, LOW); setValve(5, HIGH);  
setValve(6, HIGH); setValve(7, LOW);  
setValve(8, LOW); setValve(9, HIGH);  
setValve(10, HIGH); setValve(11, LOW);  
setValve(12, LOW); setValve(13, LOW);  
setValve(14, LOW); setValve(15, HIGH);  
setValve(16, HIGH); setValve(17, LOW);  
setValve(18, HIGH); setValve(19, LOW);
```

- **Hidden from programmer:**

- Location of fluids
- Details of mixing, I/O
- Logic of valve control
- Timing of chip operations

# Fluidic Abstraction Layers

## Abstract Computational Problem

- SAT formula, max-clique graph



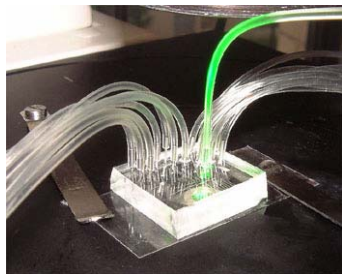
## Protocol Description Language

- readable code with high-level mixing ops

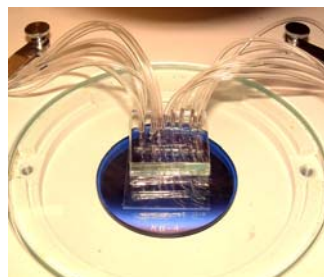


## Fluidic Instruction Set Architecture (ISA)

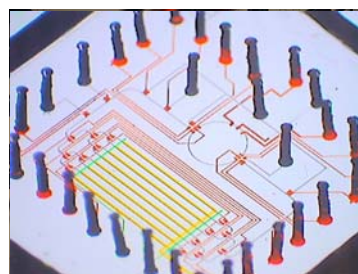
- primitives for I/O, storage, transport, mixing



chip 1



chip 2



chip 3

## Silicon Analog

Mathematica



C



x86



Pentium III,  
Pentium IV

# Fluidic Abstraction Layers

## Abstract Computational Problem

- SAT formula, max-clique graph



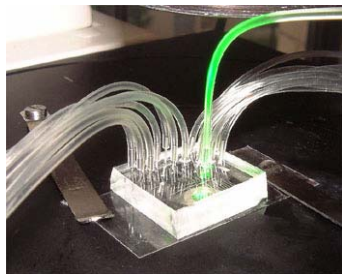
## Protocol Description Language

- readable code with high-level mixing ops

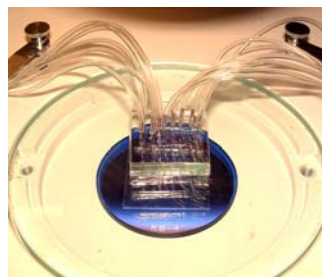


## Fluidic Instruction Set Architecture (ISA)

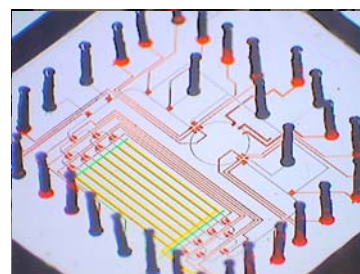
- primitives for I/O, storage, transport, mixing



chip 1



chip 2



chip 3

## Benefits:

- Portability
- Division of labor
- Scalability
- Expressivity

# Fluidic Abstraction Layers

## Abstract Computational Problem

- SAT formula, max-clique graph



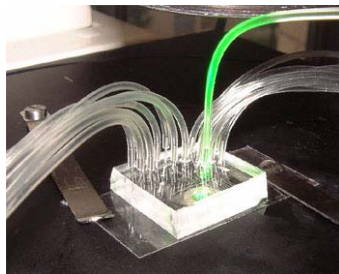
## Protocol Description Language

- readable code with high-level mixing ops

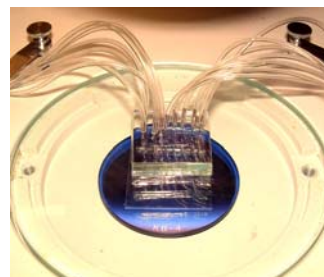


## Fluidic Instruction Set Architecture (ISA)

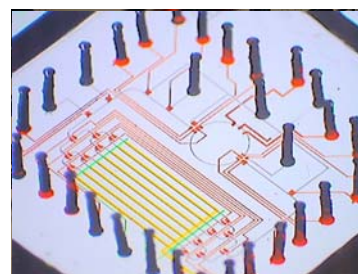
- primitives for I/O, storage, transport, mixing



chip 1



chip 2



chip 3

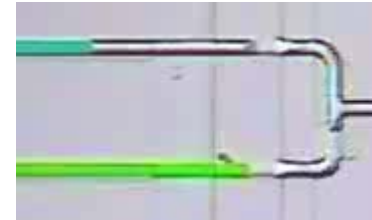
## Benefits:

- Portability
- Division of labor
- Scalability
- Expressivity

# Abstraction 1: Digital Architecture

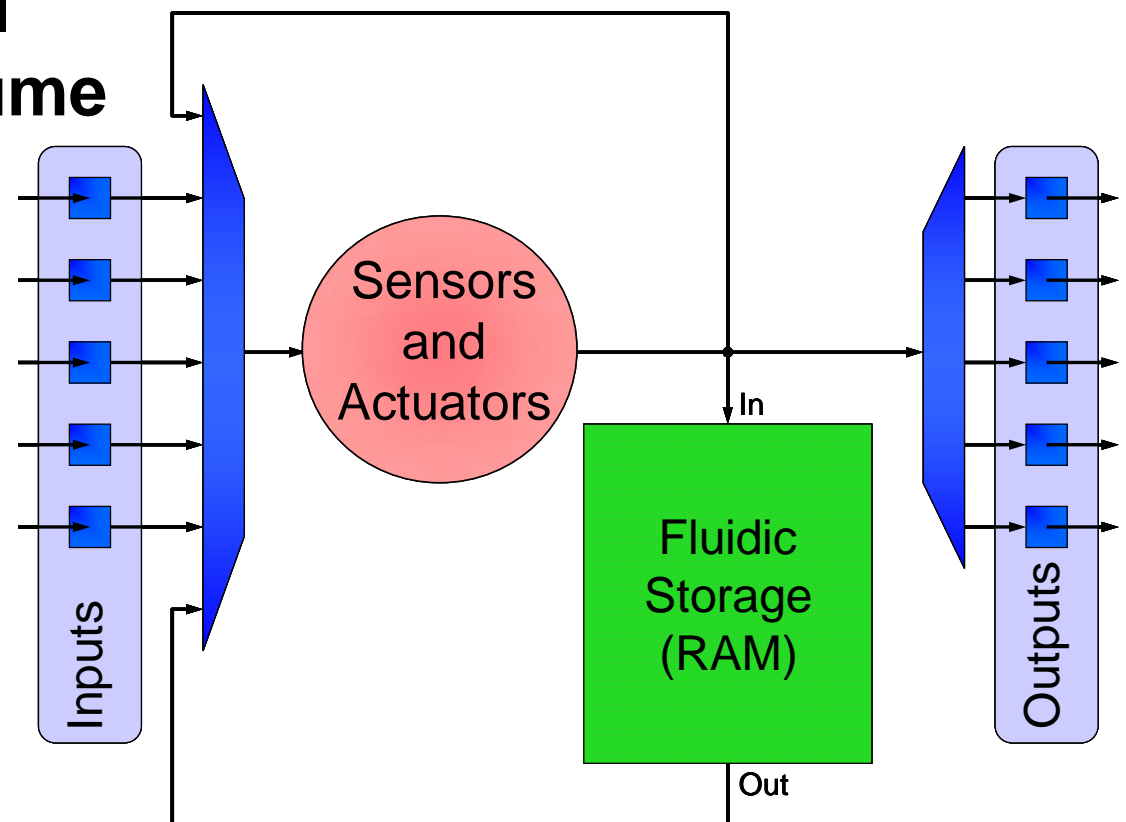
- **Recent chips can control independent fluid samples**

- Droplet-based samples [Fair et al.]
- Continuous-flow samples [Urbanski et al.]
- Microfluidic latches [Urbanski et al.]



- **In abstract machine, all samples have unit volume**

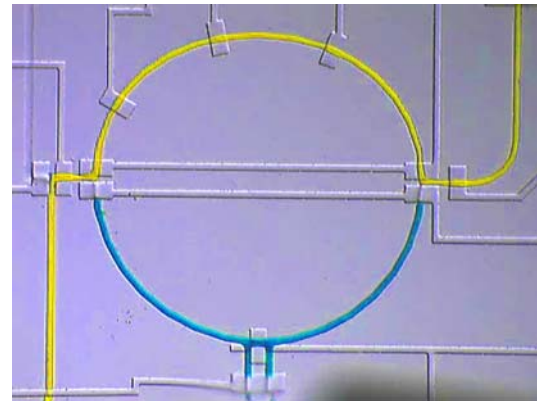
- Input/output a sample
- Store a sample
- Operate on a sample



# Abstraction 2: Mix Instruction

- **Microfluidic chips have various mixing technologies**

- Electrokinetic mixing [Levitan et al.]
- Droplet mixing [Fair et al.]
- Rotary mixing [Quake et al.]



- **Common attributes:**

- Ability to mix two samples in equal proportions, store result

- **Fluidic ISA: `mix`** (int src<sub>1</sub>, int src<sub>2</sub>, int dst)

- Ex: `mix(1, 2, 3)`

Storage Cells	
1	Yellow
2	Blue
3	Yellow
4	Yellow

Mixer
Yellow
Yellow

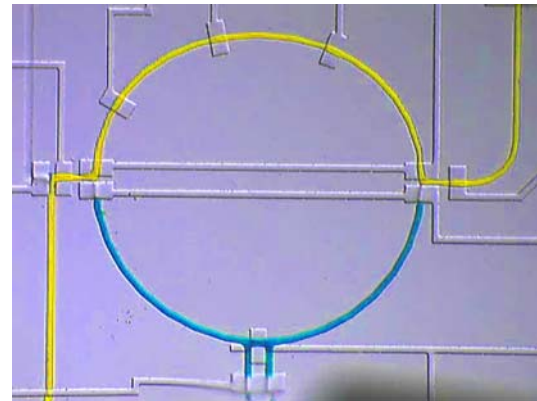
- To allow for lossy transport, only 1 unit of mixture retained



# Abstraction 2: Mix Instruction

- **Microfluidic chips have various mixing technologies**

- Electrokinetic mixing [Levitan et al.]
- Droplet mixing [Fair et al.]
- Rotary mixing [Quake et al.]



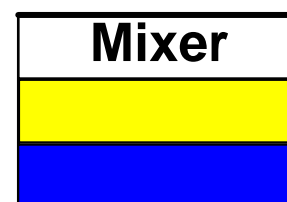
- **Common attributes:**

- Ability to mix two samples in equal proportions, store result

- **Fluidic ISA: `mix`** (int src<sub>1</sub>, int src<sub>2</sub>, int dst)

- Ex: `mix(1, 2, 3)`

Storage Cells	
1	
2	
3	
4	

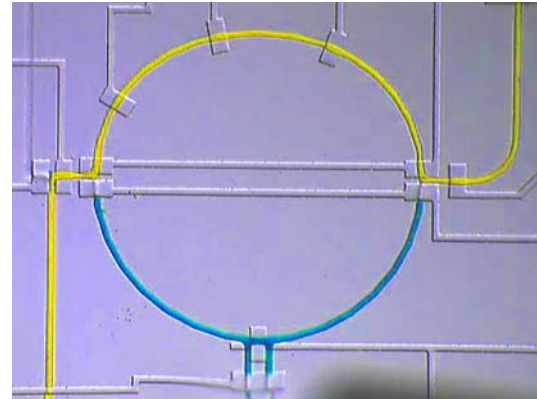


- To allow for lossy transport, only 1 unit of mixture retained

# Abstraction 2: Mix Instruction

- **Microfluidic chips have various mixing technologies**

- Electrokinetic mixing [Levitan et al.]
- Droplet mixing [Fair et al.]
- Rotary mixing [Quake et al.]



- **Common attributes:**

- Ability to mix two samples in equal proportions, store result

- **Fluidic ISA: `mix`** (int src<sub>1</sub>, int src<sub>2</sub>, int dst)

- Ex: `mix(1, 2, 3)`

Storage Cells	
1	
2	
3	
4	

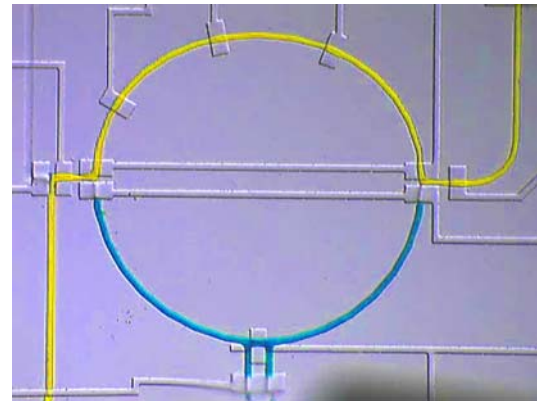


- To allow for lossy transport, only 1 unit of mixture retained

# Abstraction 2: Mix Instruction

- **Microfluidic chips have various mixing technologies**

- Electrokinetic mixing [Levitan et al.]
- Droplet mixing [Fair et al.]
- Rotary mixing [Quake et al.]



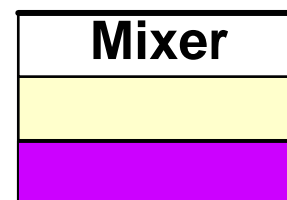
- **Common attributes:**

- Ability to mix two samples in equal proportions, store result

- **Fluidic ISA: `mix`** (int src<sub>1</sub>, int src<sub>2</sub>, int dst)


- Ex: `mix(1, 2, 3)`

Storage Cells	
1	Yellow
2	Yellow
3	Purple
4	Yellow



- To allow for lossy transport, only 1 unit of mixture retained

# Gradient Generation in Fluidic ISA



```
wait(2000);  
setValve(14, HIGH); setValve(2, LOW);  
wait(1000);  
setValve(4, HIGH); setValve(12, LOW);  
setValve(16, HIGH); setValve(18, HIGH);  
setValve(19, LOW);  
wait(2000);  
setValve(0, LOW); setValve(1, LOW);  
setValve(2, LOW); setValve(3, HIGH);  
setValve(4, LOW); setValve(5, HIGH);  
setValve(6, HIGH); setValve(7, LOW);  
setValve(8, LOW); setValve(9, HIGH);  
setValve(10, HIGH); setValve(11, LOW);  
setValve(12, LOW); setValve(13, LOW);  
setValve(14, LOW); setValve(15, HIGH);  
setValve(16, HIGH); setValve(17, LOW);  
setValve(18, HIGH); setValve(19, LOW);
```

## Direct Control

- 450 valve actuations
- only works on 1 chip

*abstraction*

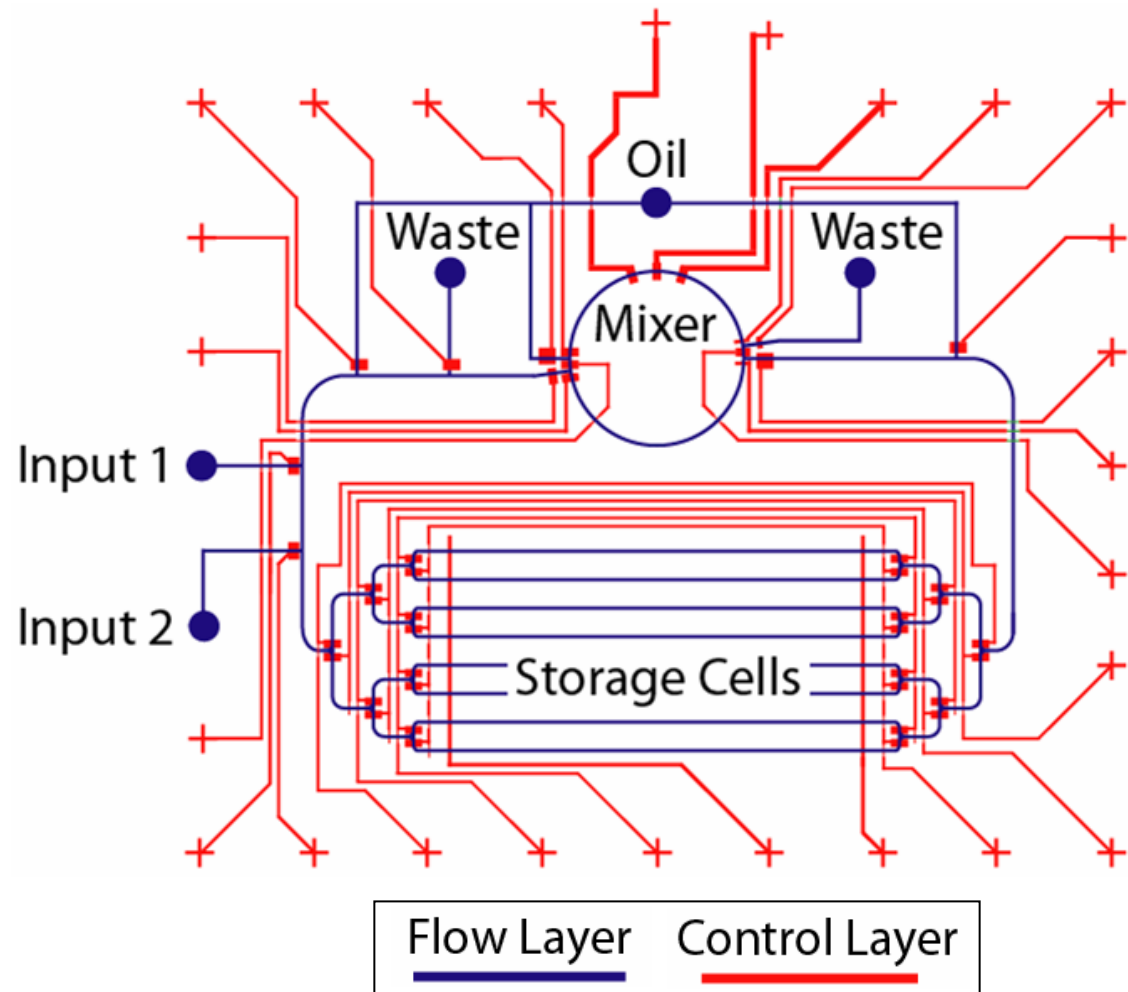


```
input(0, 0);  
input(1, 1);  
input(0, 2);  
mix(1, 2, 3);  
input(0, 2);  
mix(2, 3, 1);  
input(1, 3);  
input(0, 4);  
mix(3, 4, 2);  
input(1, 3);  
input(0, 4);  
mix(3, 4, 5);  
input(1, 4);  
mix(4, 5, 3);  
mix(0, 4);
```

## Fluidic ISA

- 15 instructions
- portable across chips

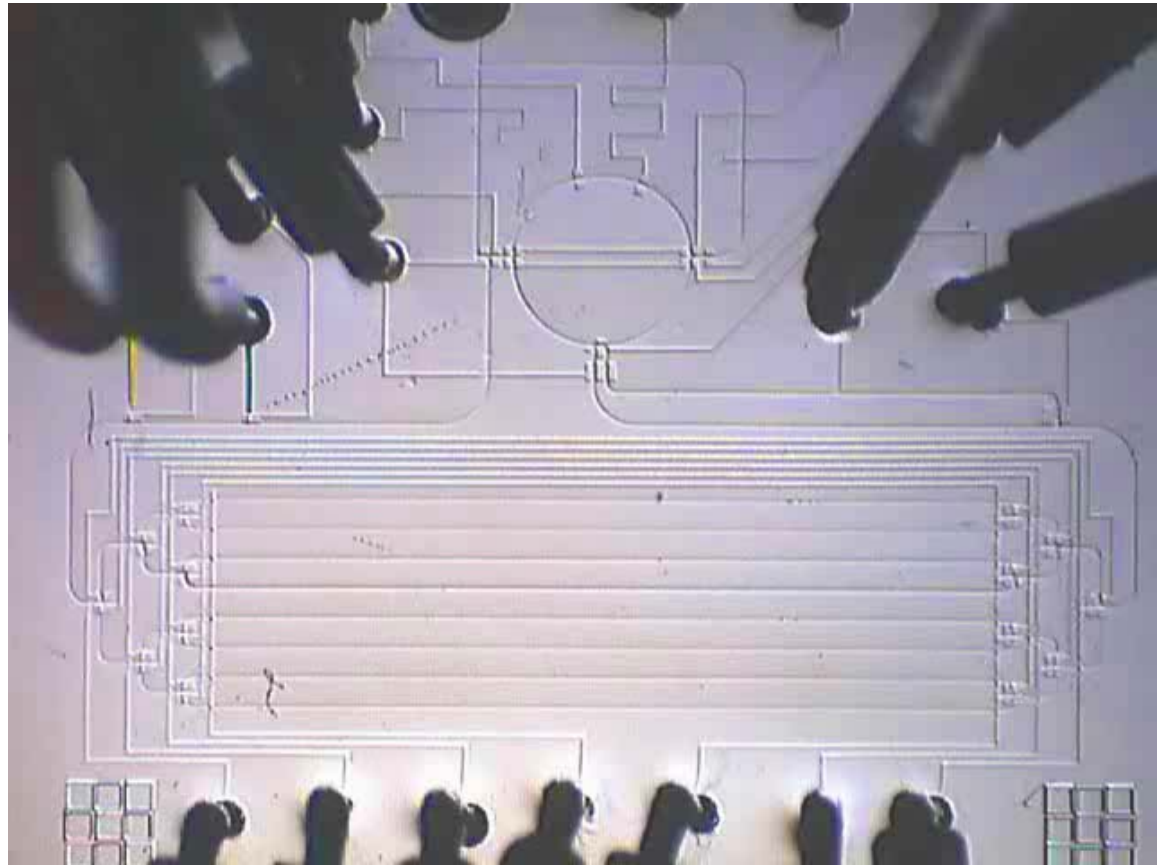
# Implementation: Oil-Driven Chip



	<b>Inputs</b>	<b>Storage Cells</b>	<b>Background Phase</b>	<b>Wash Phase</b>	<b>Mixing</b>
Chip 1	2	8	Oil	—	Rotary

# Implementation: Oil-Driven Chip

```
mix (S1, S2, D) {  
  1. Load S1  
  2. Load S2  
  3. Rotary mixing  
  4. Store into D  
}
```

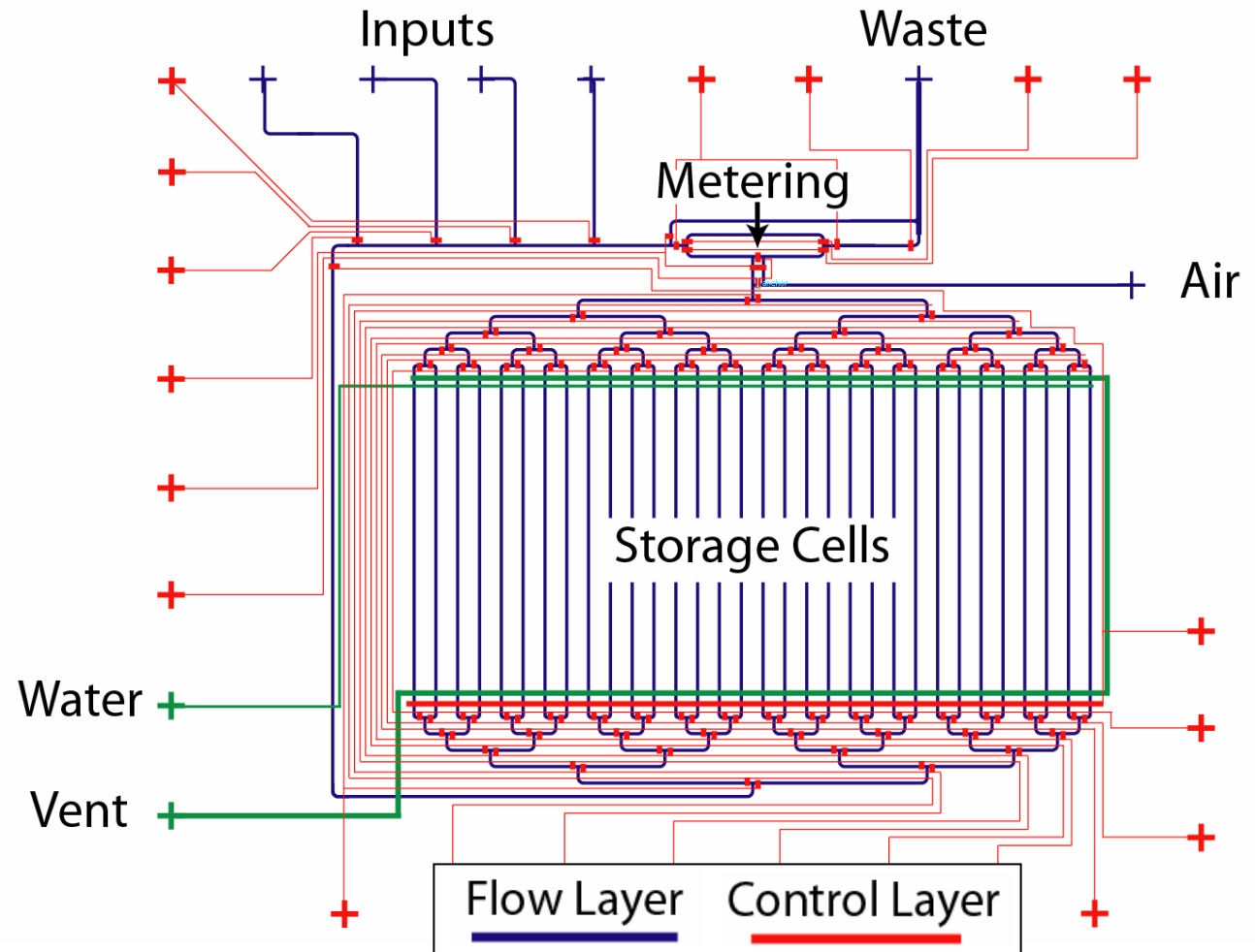


*50x real-time*

	<b>Inputs</b>	<b>Storage Cells</b>	<b>Background Phase</b>	<b>Wash Phase</b>	<b>Mixing</b>
Chip 1	2	8	Oil	—	Rotary



# Implementation 2: Air-Driven Chip

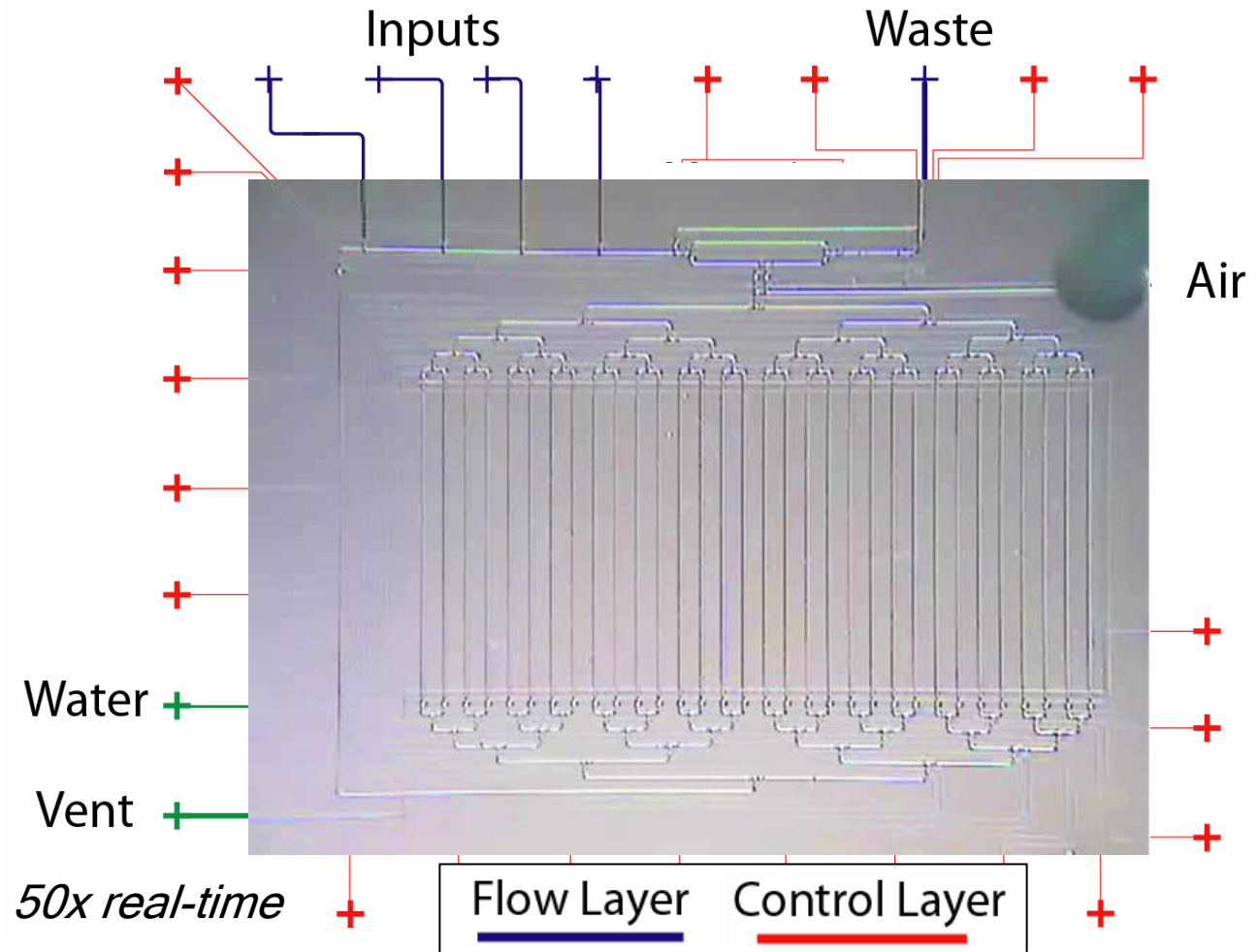


	Inputs	Storage Cells	Background Phase	Wash Phase	Mixing
Chip 1	2	8	Oil	—	Rotary
Chip 2	4	32	Air	Water	In channels

# Implementation 2: Air-Driven Chip

```

mix (S1, S2, D) {
  1. Load S1
  2. Load S2
  3. Mix / Store into D
  4. Wash S1
  5. Wash S2
}
    
```



	Inputs	Storage Cells	Background Phase	Wash Phase	Mixing
Chip 1	2	8	Oil	—	Rotary
Chip 2	4	32	Air	Water	In channels

# Abstraction Layers

## Abstract Computational Problem

- SAT formula, max-clique graph



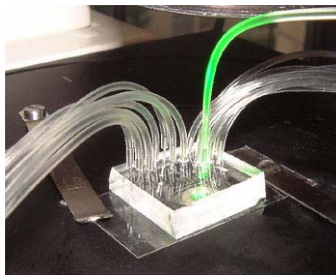
## Protocol Description Language

- readable code with high-level mixing ops

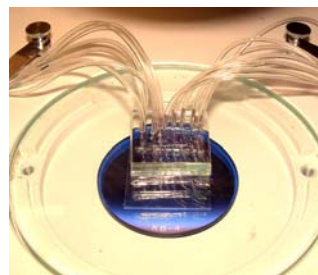


## Fluidic Instruction Set Architecture (ISA)

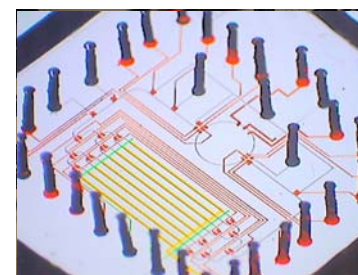
- primitives for I/O, storage, transport, mixing



chip 1



chip 2

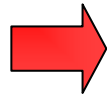


chip 3

# Abstraction 1: Managing Fluid Storage

## Fluidic ISA

```
input(0, 0);
input(1, 1);
input(0, 2);
mix(1, 2, 3);
input(0, 2);
mix(2, 3, 1);
input(1, 3);
input(0, 4);
mix(3, 4, 2);
input(1, 3);
input(0, 4);
mix(3, 4, 5);
input(1, 4);
mix(4, 5, 3);
mix(0, 4);
```



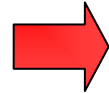
```
Fluid[] out = new Fluid[8];
Fluid yellow, blue, green;
out[0] = input(0);
yellow = input(0);
blue = input(1);
green = mix(yellow, blue);
yellow = input(0);
out[1] = mix(yellow, green);
yellow = input(0);
blue = input(1);
out[2] = mix(yellow, blue);
yellow = input(0);
blue = input(1);
green = mix(yellow, blue);
blue = input(1);
out[3] = mix(blue, green);
out[4] = input(1);
```

## 1. Storage Management

- **Programmer uses location-independent Fluid variables**
  - Runtime system assigns & tracks location of each Fluid
  - Comparable to automatic memory management (e.g., Java)

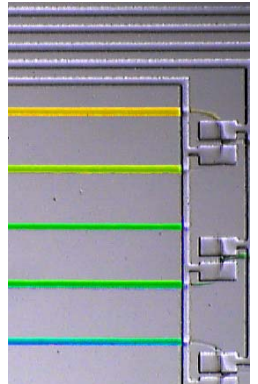
# Abstraction 2: Fluid Re-Generation

```
Fluid[] out = new Fluid[8];  
Fluid yellow, blue, green;  
out[0] = input(0);  
yellow = input(0);  
blue = input(1);  
green = mix(yellow, blue);  
yellow = input(0);  
out[1] = mix(yellow, green);  
yellow = input(0);  
blue = input(1);  
out[2] = mix(yellow, blue);  
yellow = input(0);  
blue = input(1);  
green = mix(yellow, blue);  
blue = input(1);  
out[3] = mix(blue, green);  
out[4] = input(1);
```



```
Fluid[] out = new Fluid[8];  
Fluid yellow = input(0);  
Fluid blue = input(1);  
Fluid green = mix(yellow, blue);  
  
out[0] = yellow;  
out[1] = mix(yellow, green);  
out[2] = green;  
out[3] = mix(blue, green);  
out[4] = blue;
```

## 2. Fluid Re-Generation

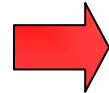


- **Programmer may use a Fluid variable multiple times**
  - Each time, a physical Fluid is consumed on-chip
  - Runtime system re-generates Fluids from computation history

# Abstraction 3: Arbitrary Mixing

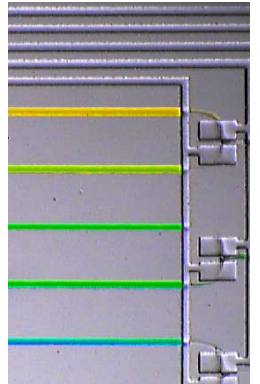
```
Fluid[] out = new Fluid[8];
Fluid yellow = input(0);
Fluid blue = input(1);
Fluid green = mix(yellow, blue);

out[0] = yellow;
out[1] = mix(yellow, green);
out[2] = green;
out[3] = mix(blue, green);
out[4] = blue;
```



```
Fluid[] out = new Fluid[8];
Fluid yellow = input (0);
Fluid blue = input (1);

out[0] = yellow;
out[1] = mix(yellow, 3/4, blue, 1/4);
out[2] = mix(yellow, 1/2, blue, 1/2);
out[3] = mix(yellow, 1/4, blue, 3/4);
out[4] = blue;
```



## 2. Fluid Re-Generation

## 3. Arbitrary Mixing

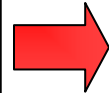
- **Allows mixing fluids in any proportion, not just 50/50**
  - Fluid **mix** (Fluid  $F_1$ , float  $p_1$ , Fluid  $f_2$ , float  $F_2$ )
    - Returns Fluid that is  $p_1$  parts  $F_1$  and  $p_2$  parts  $F_2$
  - Runtime system translates to 50/50 mixes in Fluidic ISA
  - Note: some mixtures only reachable within error tolerance  $\varepsilon$



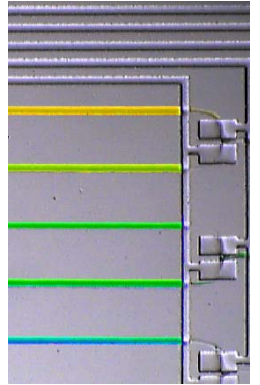
# Abstraction 3: Arbitrary Mixing

```
Fluid[] out = new Fluid[8];  
Fluid yellow = input (0);  
Fluid blue = input (1);
```

```
out[0] = yellow;  
out[1] = mix(yellow, 3/4, blue, 1/4);  
out[2] = mix(yellow, 1/2, blue, 1/2);  
out[3] = mix(yellow, 1/4, blue, 3/4);  
out[4] = blue;
```



```
Fluid[] out = new Fluid[8];  
Fluid yellow = input (0);  
Fluid blue = input (1);  
  
for (int i=0; i<=4; i++) {  
    out[i] = mix(yellow, 1-i/4, blue, i/4);  
}
```



## 3. Arbitrary Mixing

## 4. Parameterized Mixing

- **Allows mixing fluids in any proportion, not just 50/50**
  - Fluid **mix** (Fluid  $F_1$ , float  $p_1$ , Fluid  $f_2$ , float  $F_2$ )
    - Returns Fluid that is  $p_1$  parts  $F_1$  and  $p_2$  parts  $F_2$
  - Runtime system translates to 50/50 mixes in Fluidic ISA
  - Note: some mixtures only reachable within error tolerance  $\epsilon$

# BioStream Protocol Language

- **Supports all the abstractions**

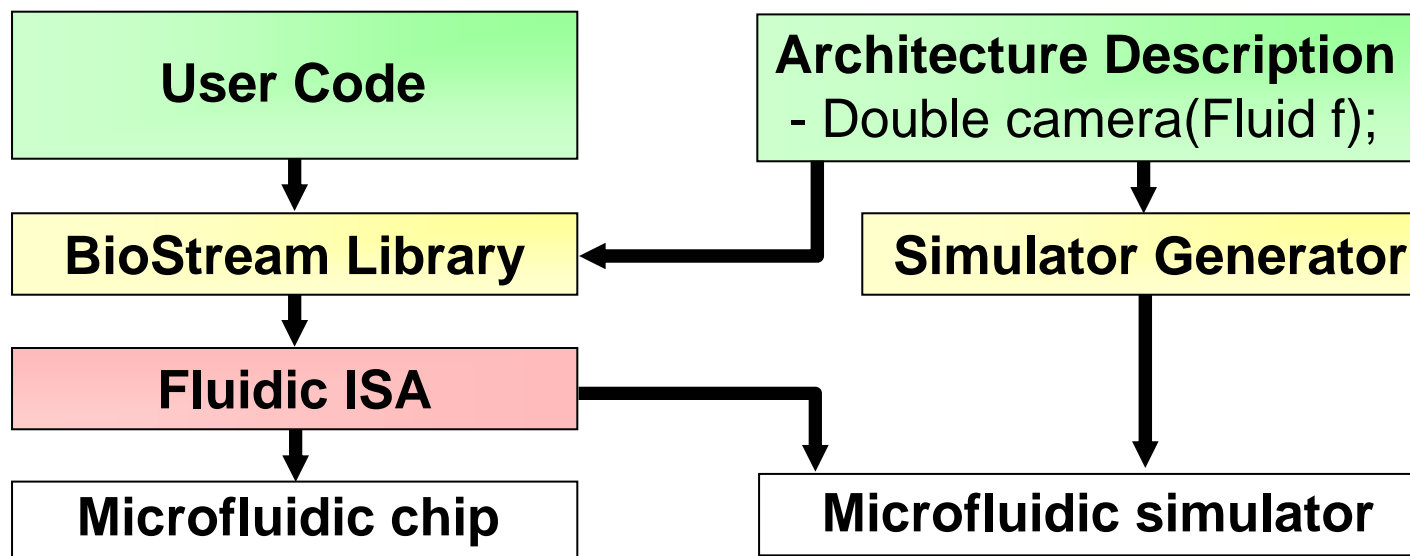
- Automatic storage management
- Re-generation of fluids
- Arbitrary mixing

```
Fluid yellow = input (0);  
Fluid blue = input (1);  
Fluid[] out = new Fluid[8];  
  
for (int i=0; i<=4; i++) {  
    out[i] = mix(yellow, 1-i/4, blue, i/4);  
}
```

- **Implemented as a Java library**

- Allows flexible integration with general-purpose Java code

- **Targets microfluidic chips or auto-generated simulator**



# Example: Fixed-pH Reaction

- **Goal: maintain given pH throughout reaction**
  - For in-vitro modeling of natural environment
- **Method: periodically test, adjust pH if needed**

```
Fluid sample = input (0);
Fluid acid = input(1);
Fluid base = input(2);
do {
    Fluid pH_test = mix(sample, 0.9, indicator, 0.1); // test pH of sample
    double pH = test_luminescence(pH_test);
    if (pH > 7.5) { // if pH too high, add acid
        sample = mix (sample, 0.9, acid, 0.1);
    } else if (pH < 6.5) { // if pH too low, add base
        sample = mix (sample, 0.9, base, 0.1);
    }
    wait(60);
} while (detect_activity(sample));
```

# Example: Fixed-pH Reaction

- **Goal: maintain given pH throughout reaction**
  - For in-vitro modeling of natural environment
- **Method: periodically test, adjust pH if needed**

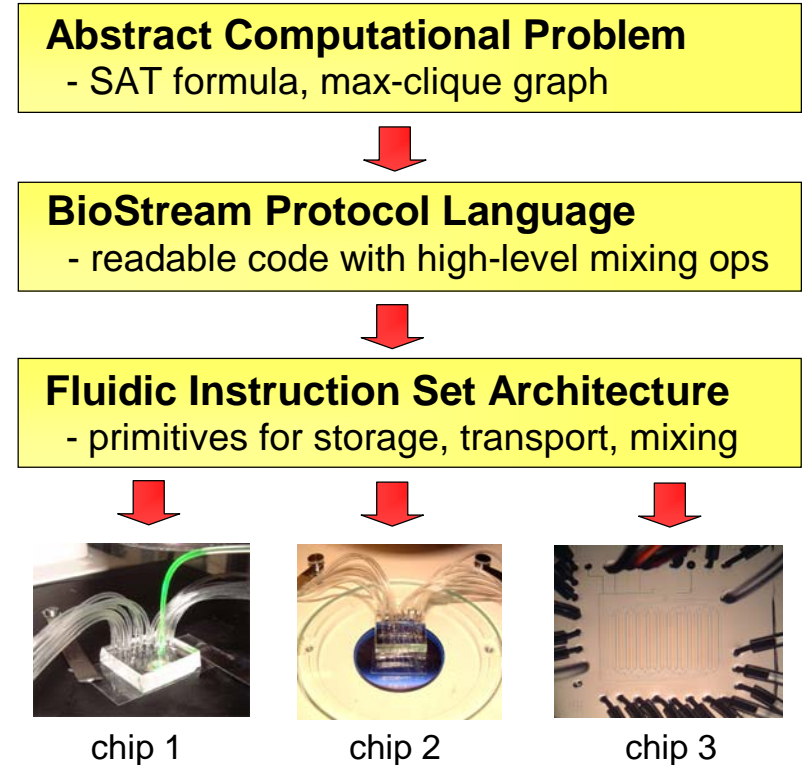
```
Fluid sample = input (0);  
Fluid acid = input(1);  
Fluid base = input(2);  
do {  
    Fluid pH_test = mix(sample, 0.9, indicator, 0.1);  
    double pH = test_luminescence(pH_test);  
    if (pH > 7.5) {  
        sample = mix (sample, 0.9, acid, 0.1);  
    } else if (pH < 6.5) {  
        sample = mix (sample, 0.9, base, 0.1);  
    }  
    wait(60);  
} while (detect_activity(sample));
```

## Feedback-Intensive Applications:

- Recursive descent search
- Directed evolution
- Cell isolation and manipulation
- Dose/response curves
- Long, complex protocols

# Big Picture

- **Abstraction layers enable simple, scalable hardware**
  - Instead of custom chips, build general-purpose devices
- **Vision for microfluidics: everyone uses a standard chip**
  - Thousands of storage cells
  - Dozens of parallel mixers
  - Integrated support for cell manipulation, PCR, readout, etc.
- **Vision for software: a defacto language for experimental science**
  - You can download a colleague's code, run it on your chip



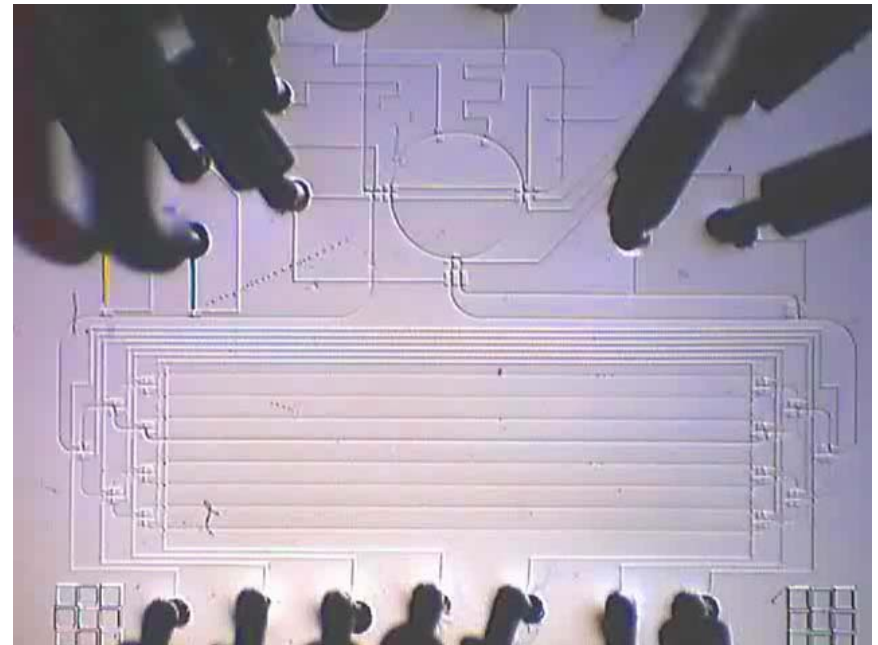
# Related Work

- **Automatic generation / scheduling of biology protocols**
  - EDNAC computer for automatically solving 3-SAT [Johnson]
  - Compile SAT to microfluidic chips [Landweber et al.] [van Noort]
  - Robot scientist: generates/tests genetic hypotheses [King et al.]
  - Mapping sequence graphs to grid-based chips [Su/Chakrabarty]
- **Custom microfluidic chips for biological computation**
  - DNA computing [Grover & Mathies] [van Noort et al.] [McCaskill] [Livstone, Weiss, & Landweber] [Gehani & Reif] [Farfel & Stefanovic]
  - Self-assembly [Somei, Kaneda, Fujii, & Murata] [Whitesides et al.]
- **General-purpose microfluidic chips**
  - Using electrowetting, with flexible mixing [Fair et al.]
  - Using dielectrophoresis, with retargettable GUI [Gascoyne et al.]
  - Using Braille displays as programmable actuators [Gu et al.]



# Conclusions

- **End-to-end system for programmable microfluidics**
  - Usable: high-level code is natural expression of protocol
  - Portable: same code executes on diverse chips
  - Scalable: protocols automatically utilize parallel resources
- **Relies on abstraction layers**
  - Fluidic ISA
  - BioStream language
- **Future work**
  - Incorporate stateful molecules, cells into abstraction layers
  - Looking for killer applications!



<http://cag.csail.mit.edu/biostream>