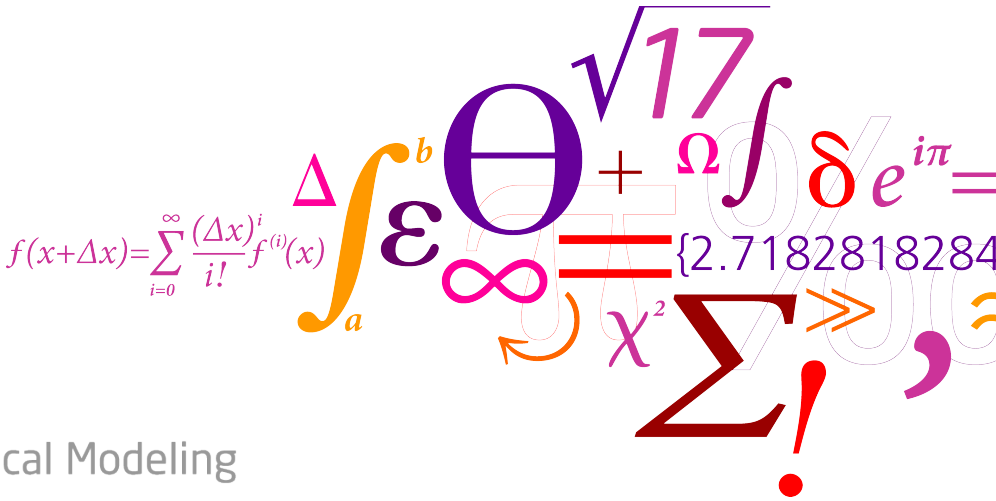
Recent Research and Emerging Challenges in the System-Level Design of Digital Microfluidic Biochips

Paul Pop, Elena Maftei, Jan Madsen
Technical University of Denmark

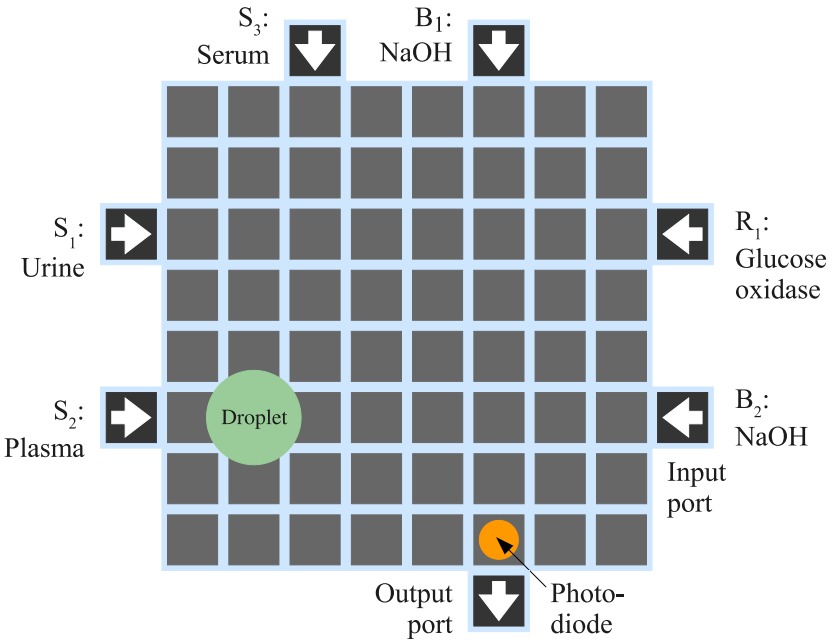


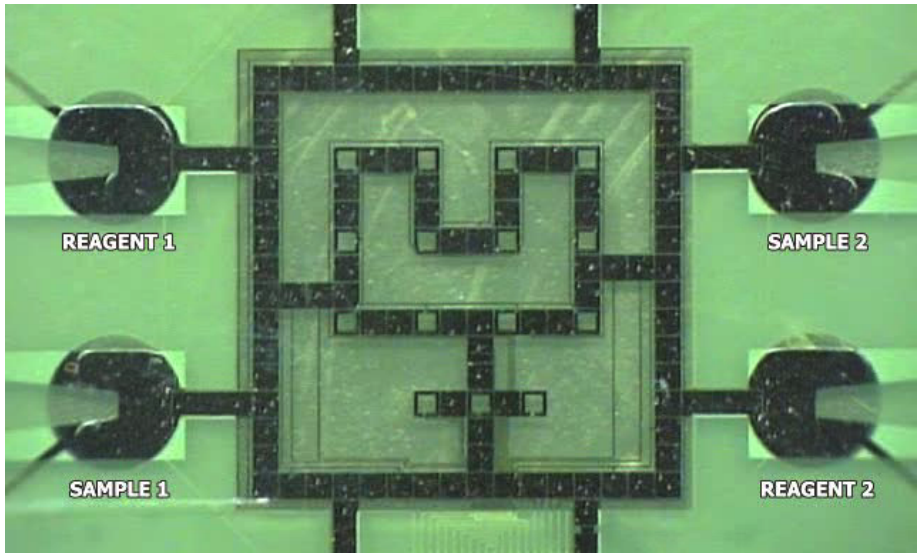
DTU Informatics
Department of Informatics and Mathematical Modeling

Outline

- Digital microfluidic biochips
  - Architecture model: module vs. routing-based
  - Application model
- System level design
  - Module-based synthesis
  - Routing-based synthesis
- Challenges
  - Fault-tolerant design
  - Pin-constrained design
  - Application-specific architectures

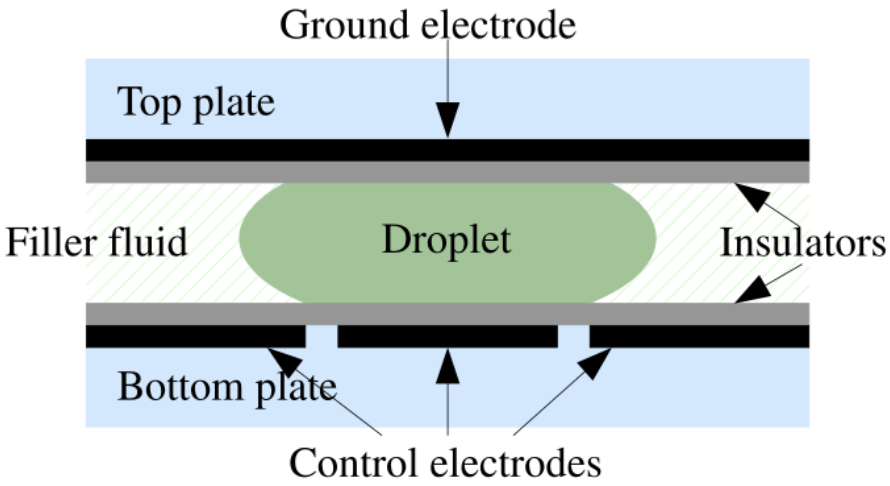
Architecture model





Biochip from Duke University

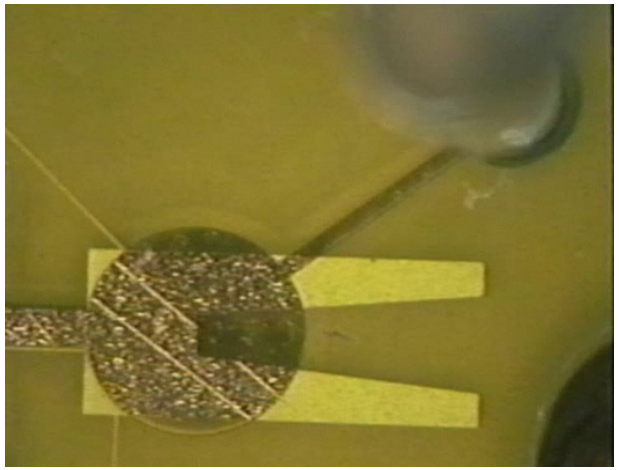
Electrowetting on Dielectric



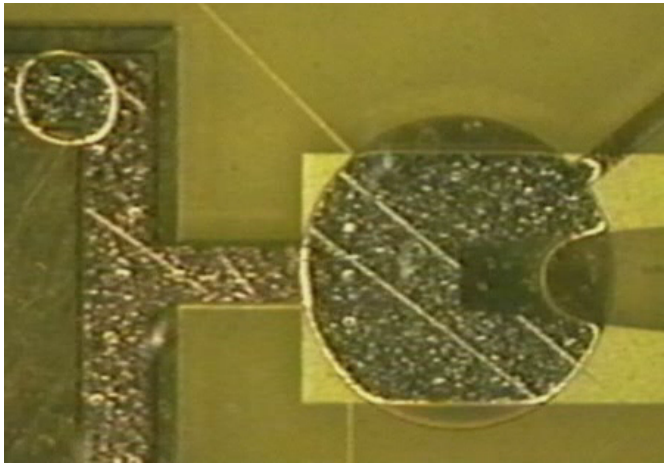
Operations, cont.

Reservoir loading

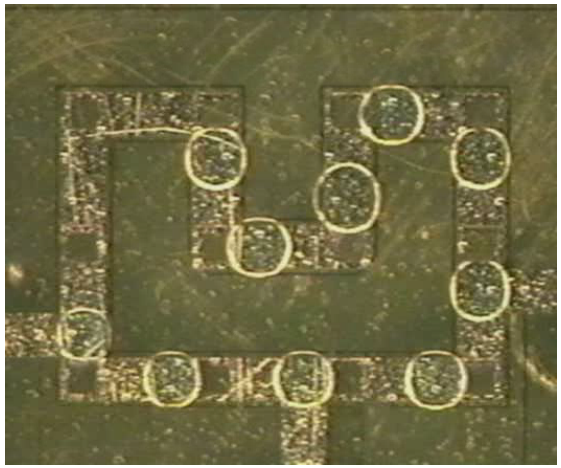
(0.1M KCL with dye)



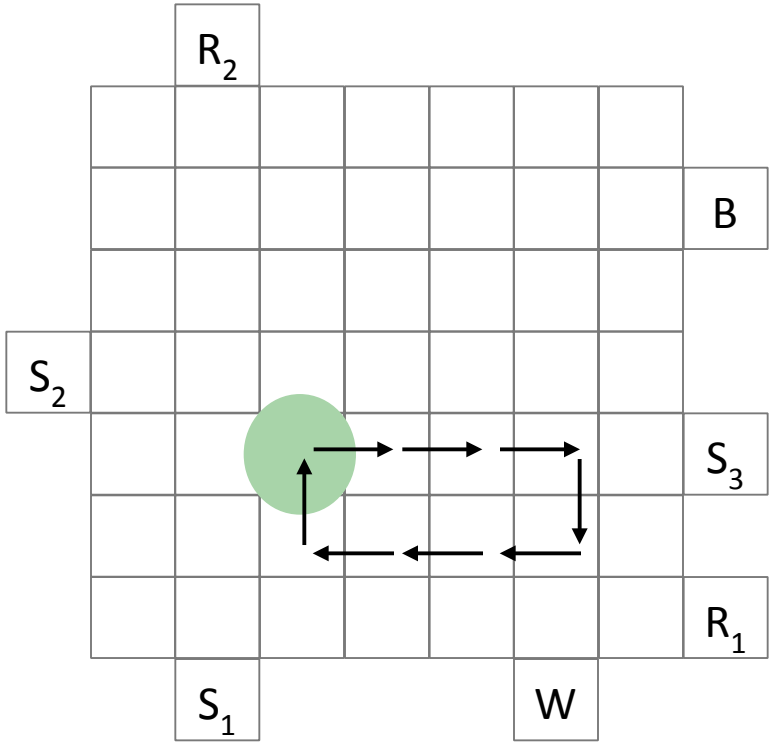
Droplet dispensing



Transport on 3-phase inner bus



Reconfigurability



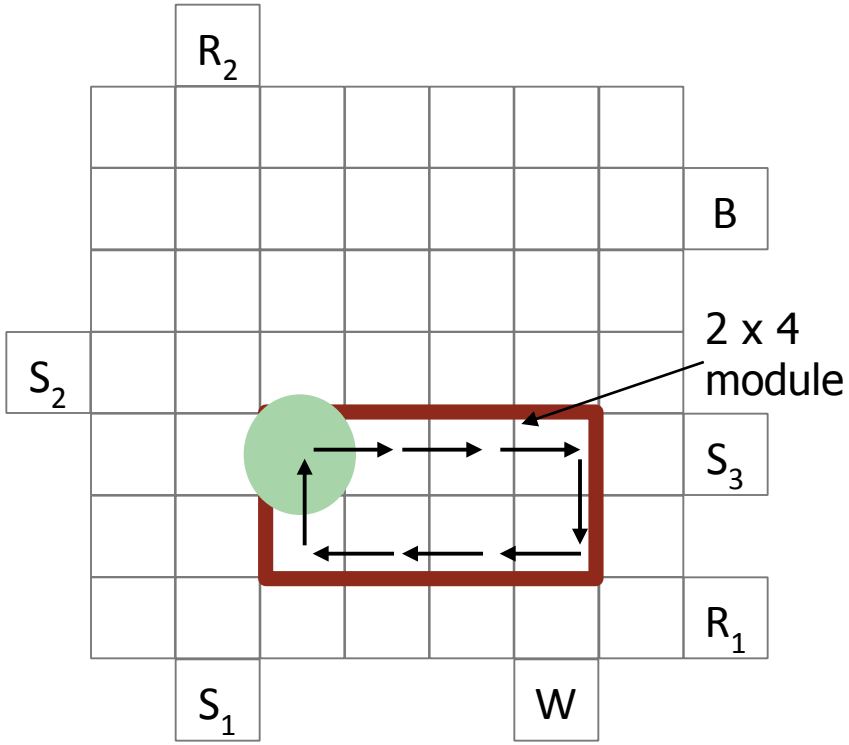
Non-reconfigurable

- Dispensing
- Detection

Reconfigurable

- Splitting/Merging
- Storage
- Mixing/Dilution

Operation execution: Module based



Module library

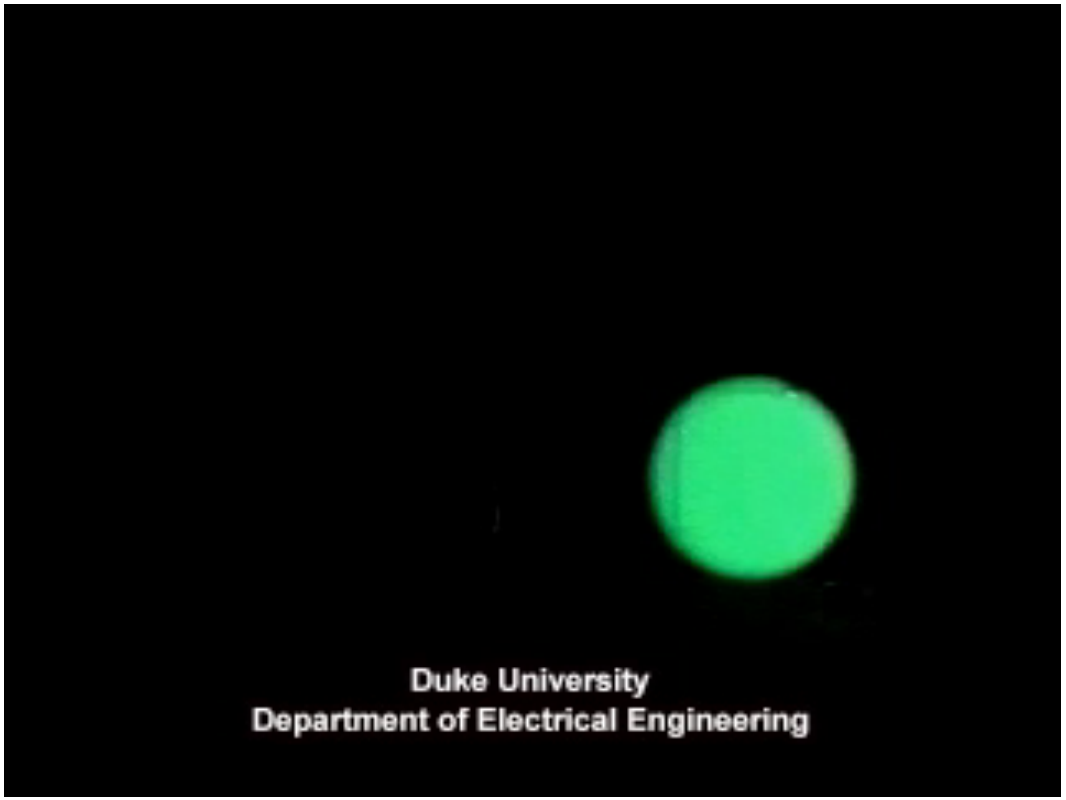
Operation	Area (cells)	Time (s)
Mix	2 x 4	3
Mix	2 x 2	4
Dilution	2 x 4	4
Dilution	2 x 2	5

Operations: Mixing

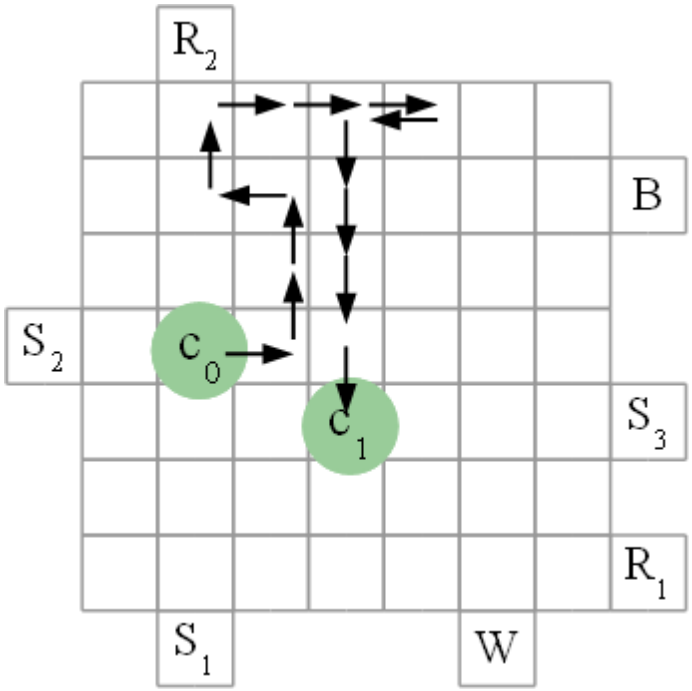
- Droplets can move anywhere

- Fixed area:
  module-based
  operation execution

- Unconstrained:
  routing-based
  operation execution



Operation execution: Routing based



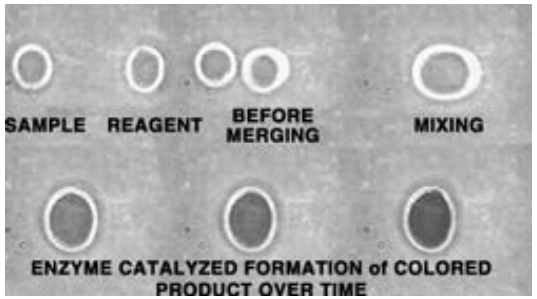
- Droplets can move anywhere
- Constrained to a module
  - We know the completion time from the module library.
- Unconstrained, any route
  - How can we find out the operation completion times?

Application model: from this…

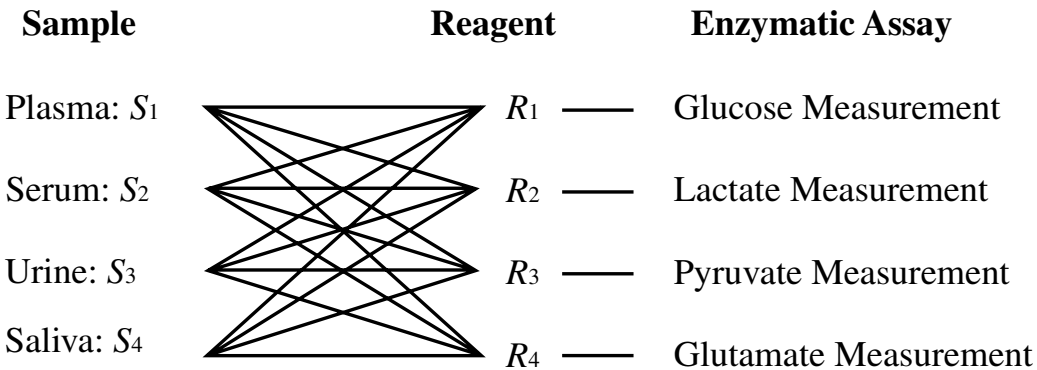
$$\text{Glucose} + H_2O + O_2 \xrightarrow{\text{Glucose Oxidase}} \text{Gluconic Acid} + H_2O_2$$

$$2H_2O_2 + \text{4-AAP} + \text{TOPS} \xrightarrow{\text{Peroxidase}} \text{Quinoneimine} + 4H_2O$$

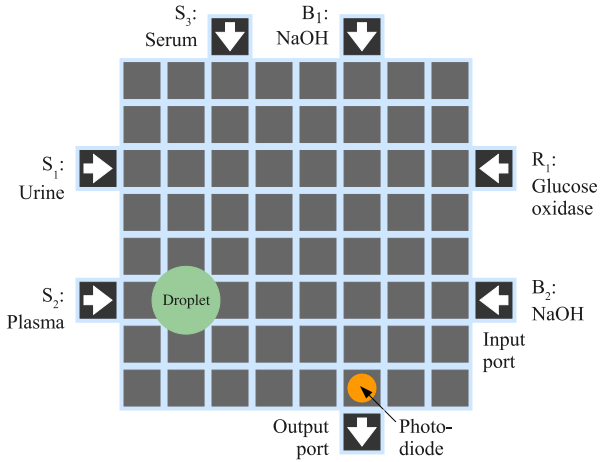
Trinder's reaction, a colorimetric enzyme-based method



Glucose assay steps on the biochip

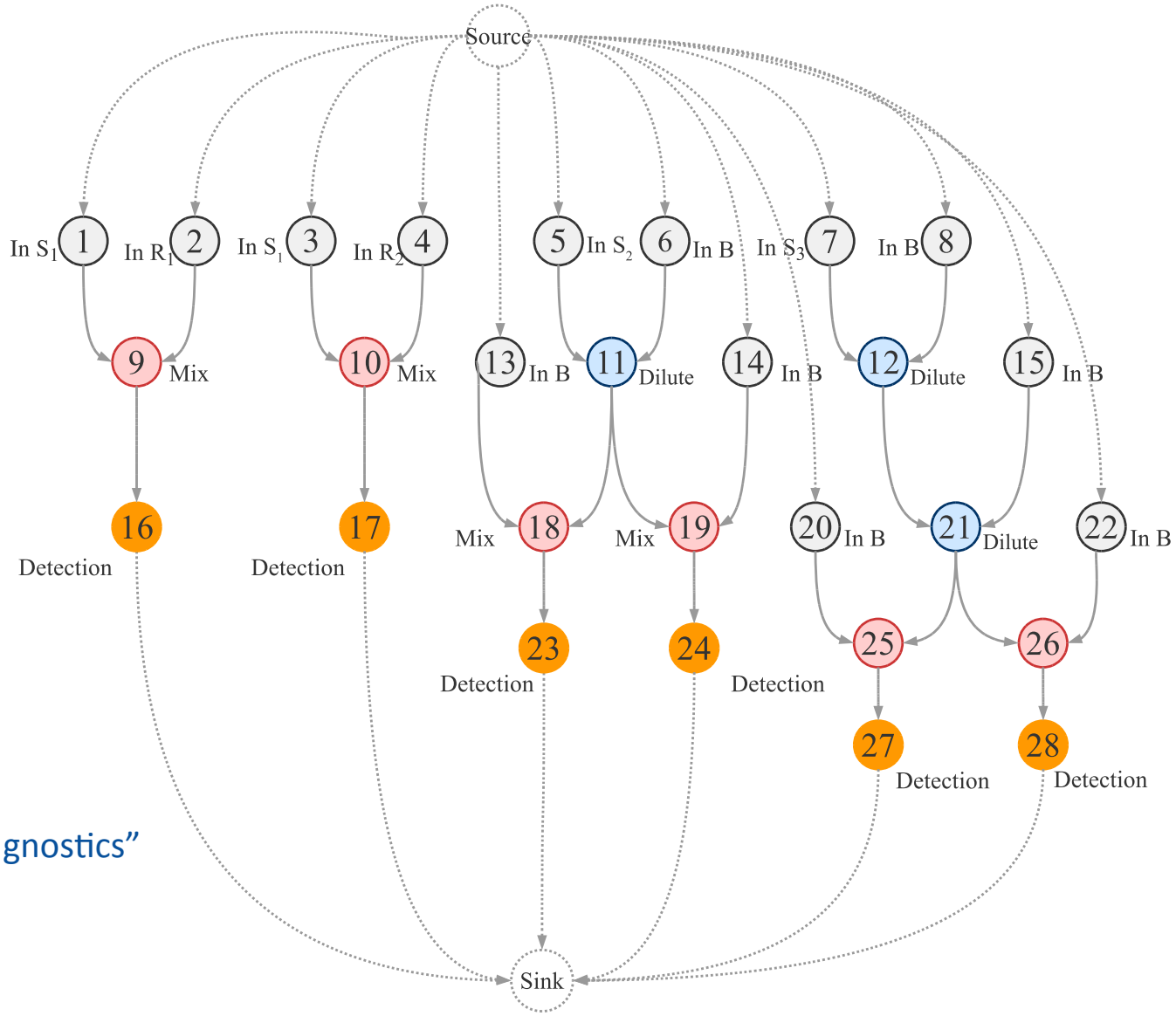


Several such reactions assays in parallel: "in-vitro diagnostics" application



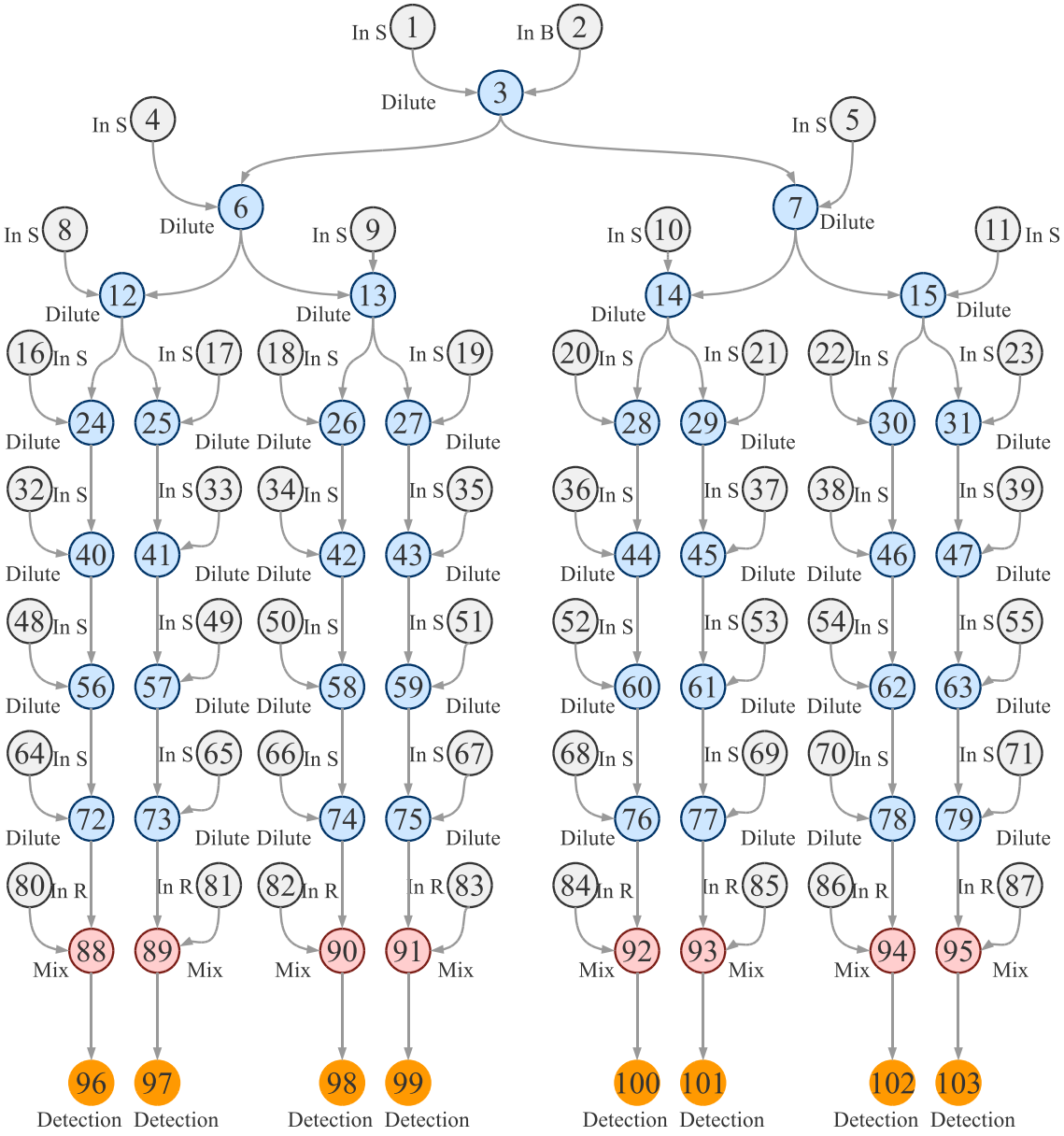
Reconfigurable architecture

Application model: …to this—an acyclic directed graph



"in-vitro diagnostics" application

Another application example: “Colorimetric protein assay”

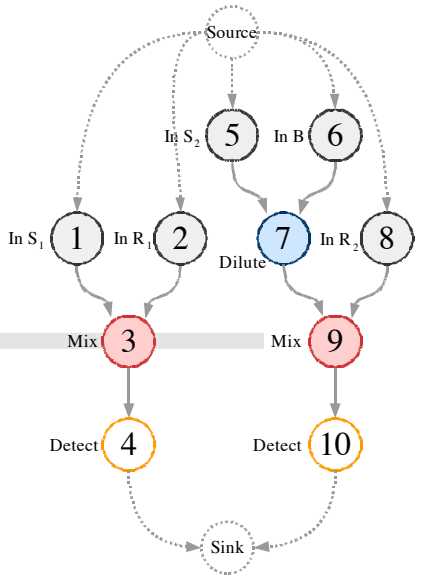


System-level design tasks

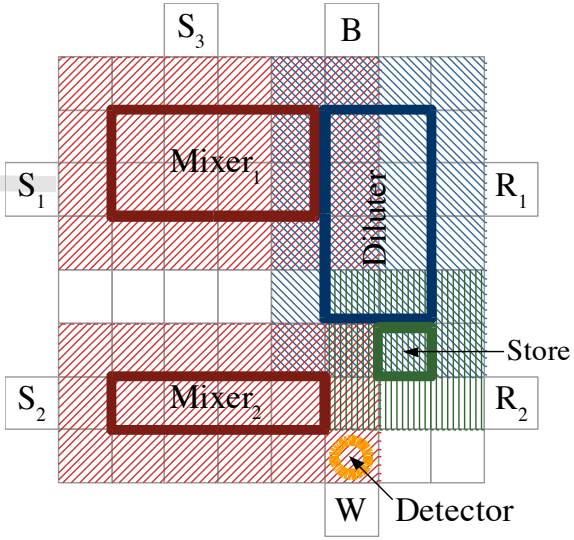
Allocation

Operation	Area (cells)	Time (s)
Mixing	2x2	6
Mixing	2x3	5
Mixing	2x4	4
Dilution	2x2	6
Dilution	2x3	5
Dilution	2x4	3
Storage	1x1	-

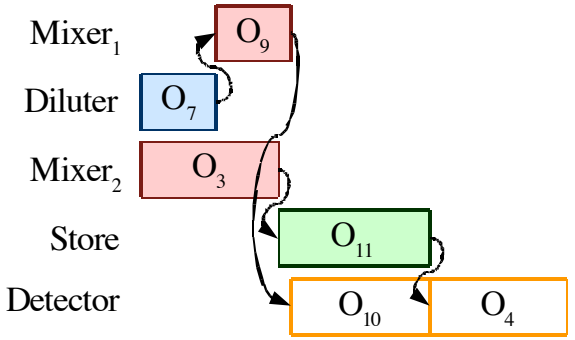
Binding



Placement & routing



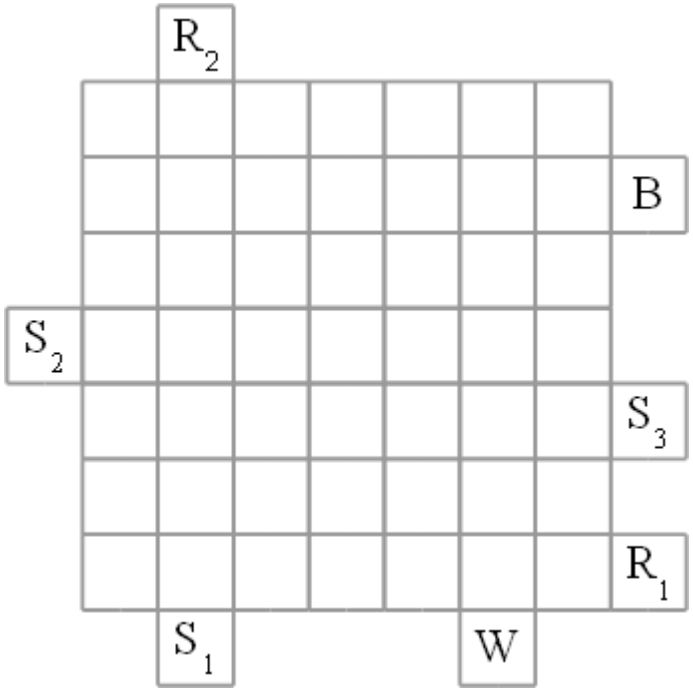
Scheduling

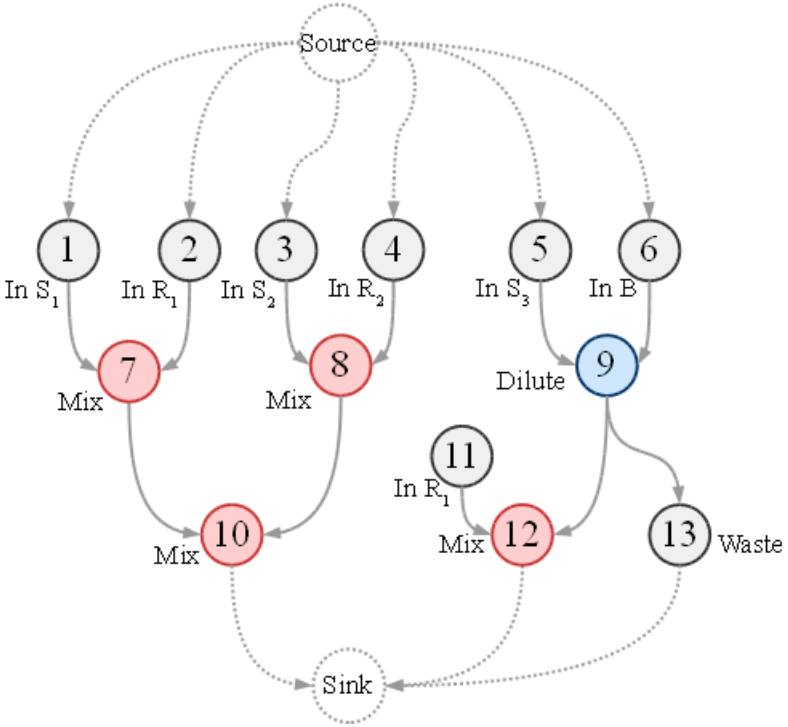


My motivation: adapt familiar design methods to a new area

	FPGA	Digital biochip
Basic Devices	Transistors	Control electrodes
	Net Wires	Reservoirs
	Clock lines	Transparent cells
Tiles	RAM	Mixers
	Multiplexer	Transport bus
	CLBs	Optical detectors
Systems	Configured FPGA	Configured biochip

Module-Based Synthesis



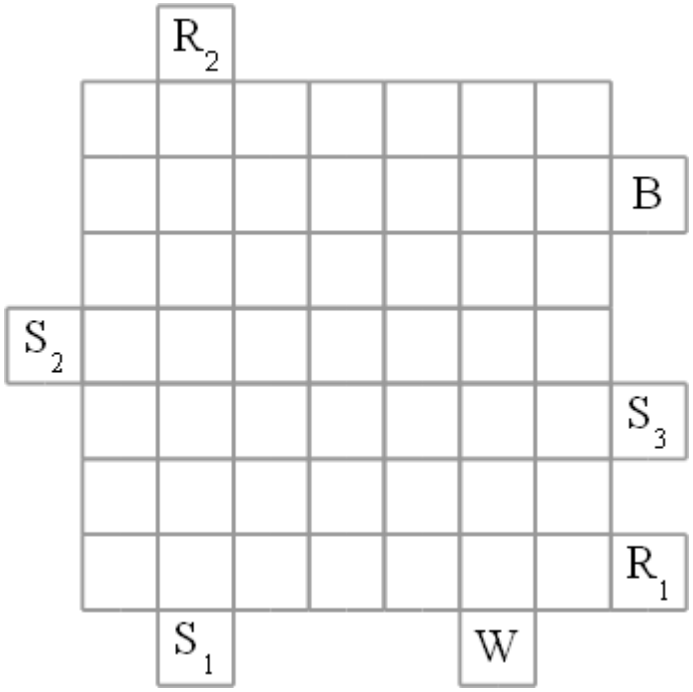


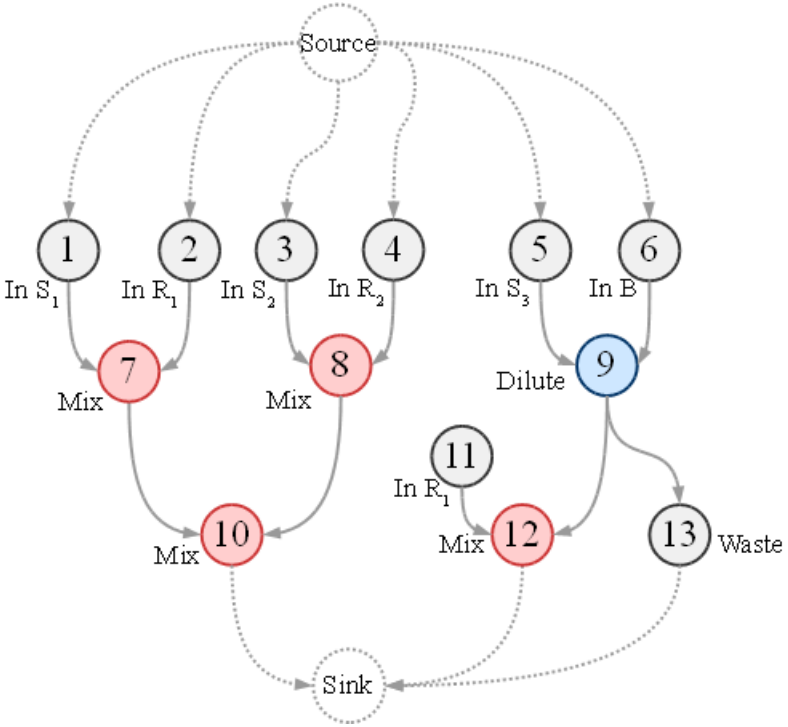
Module-Based Synthesis

O7

O8

O9



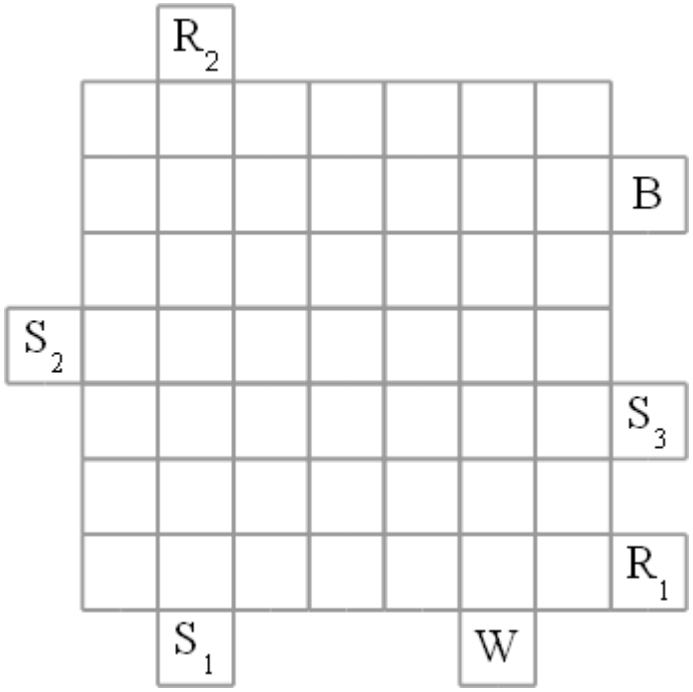


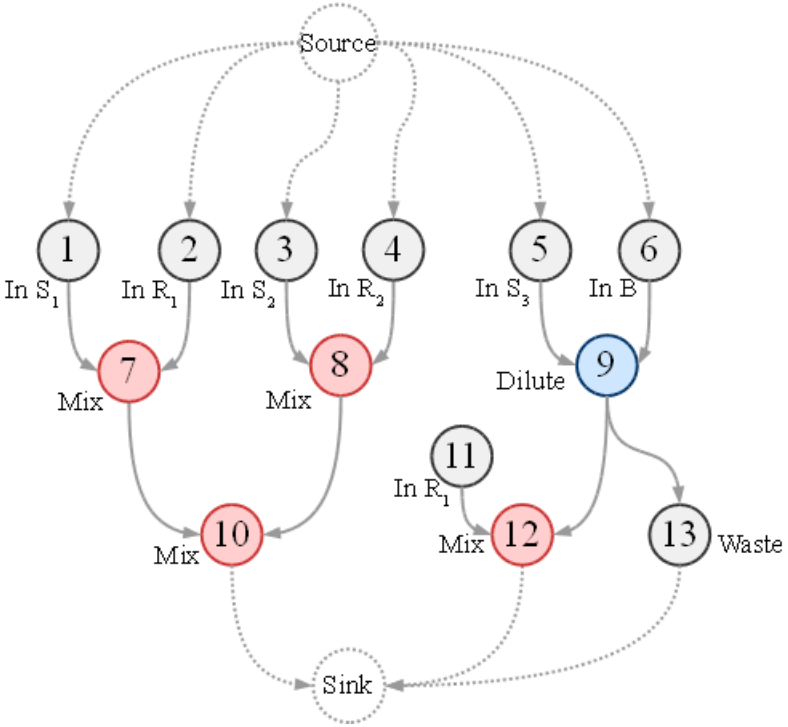
Module-Based Synthesis

O7

O8

O9

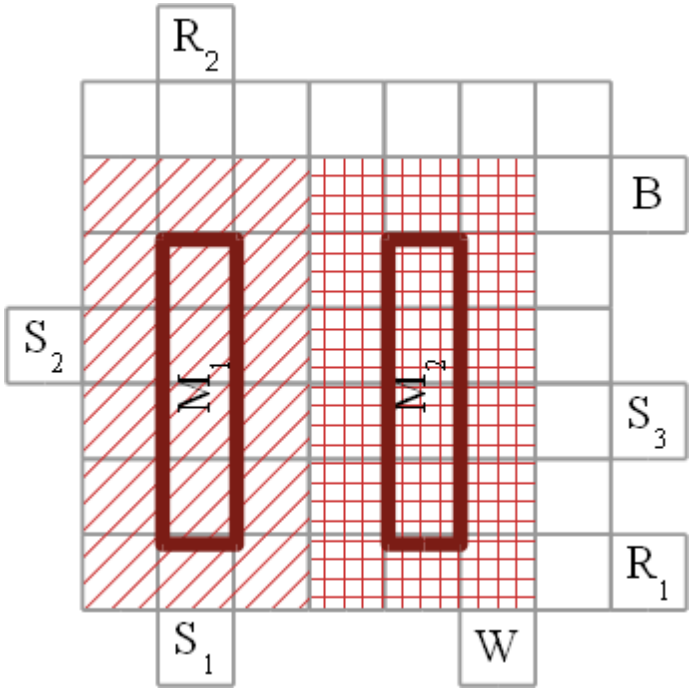


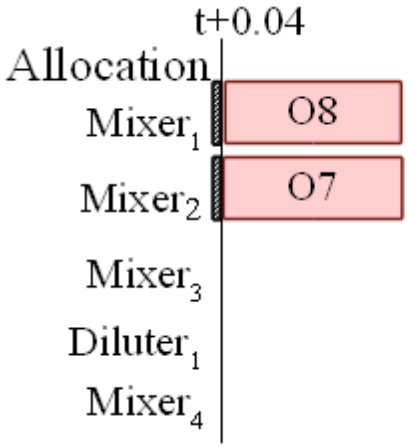


Module-Based Synthesis

O7 1x4

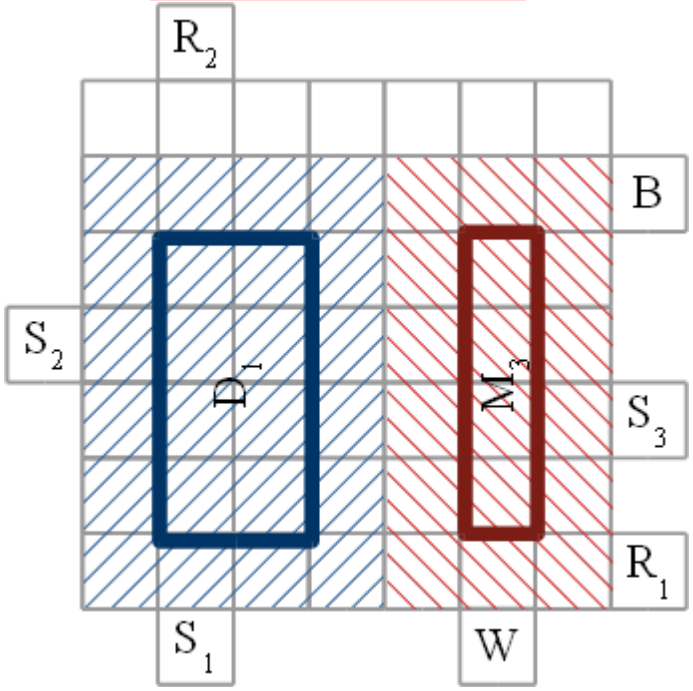
O8 1x4

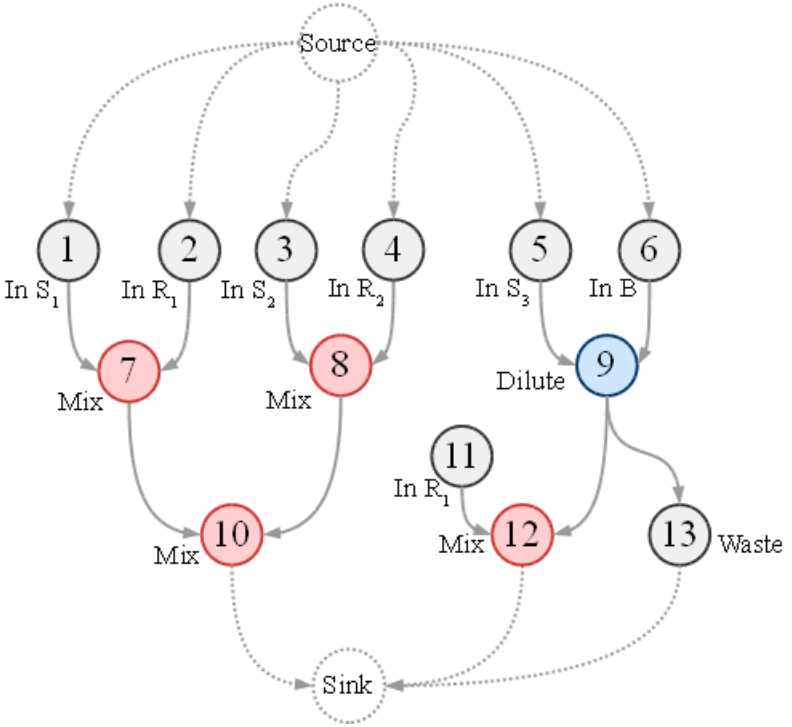




Module-Based Synthesis

O9	2x4
O10	1x4

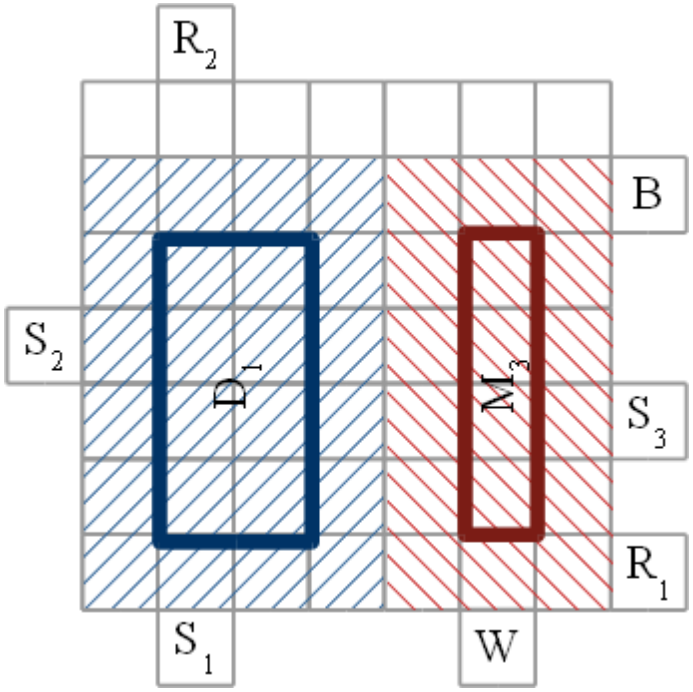


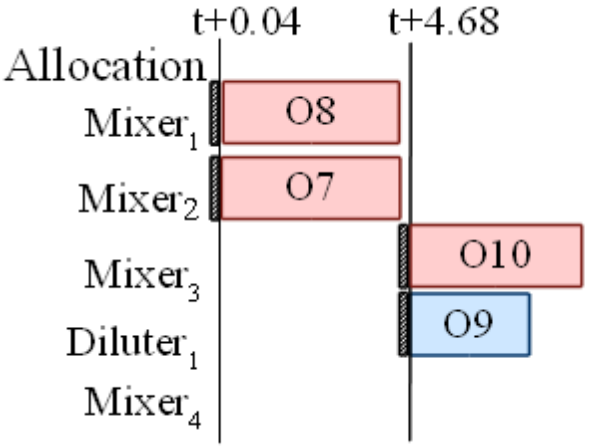


Module-Based Synthesis

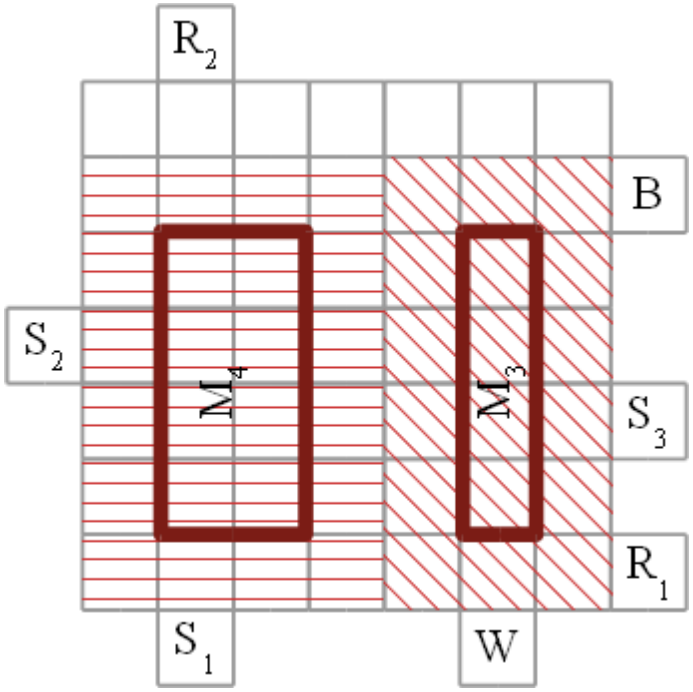
O9 2x4

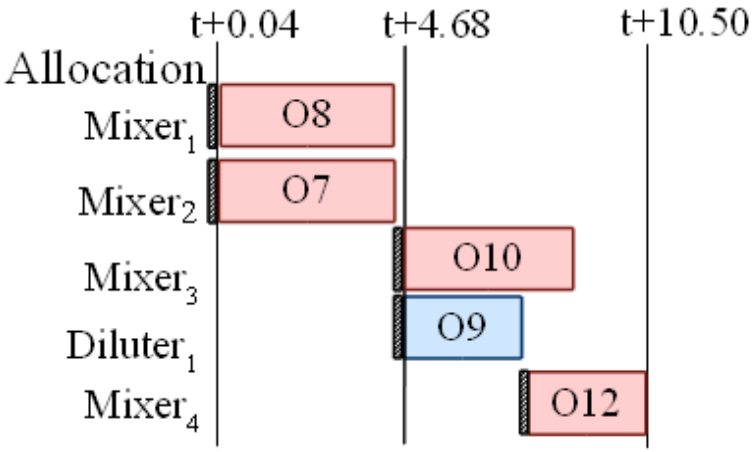
O10 1x4





Module-Based Synthesis





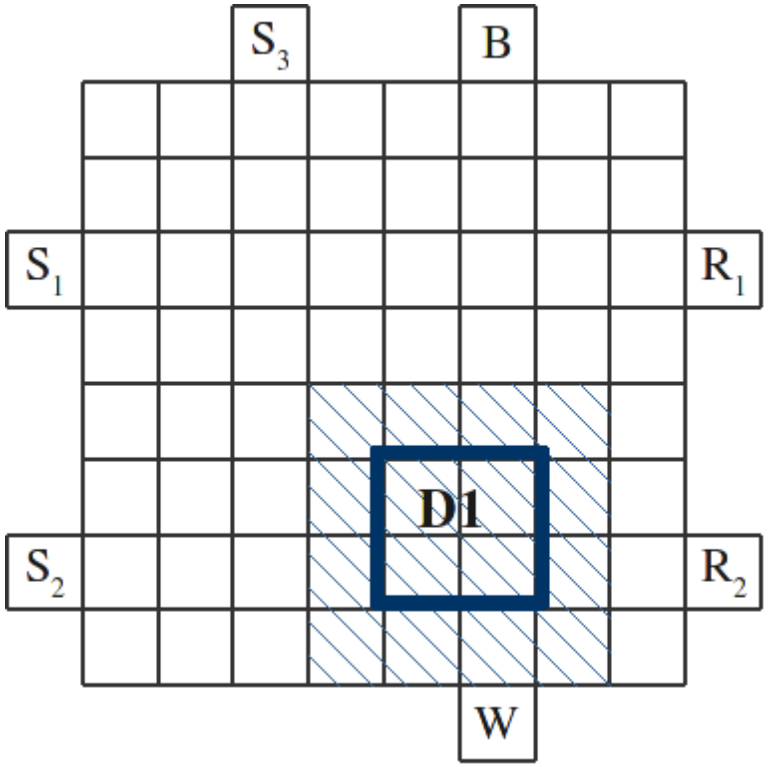
Problem Formulation

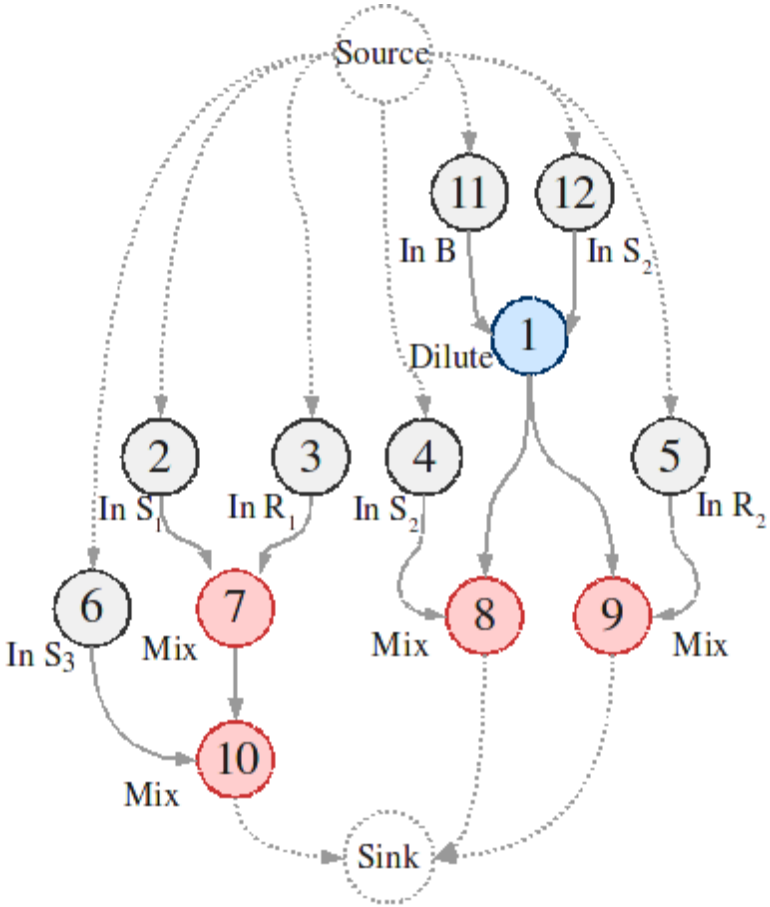
- **Given**
  - Application: graph
  - Biochip: array of electrodes
  - Library of modules

- **Determine**
  - **Allocation** of modules from modules library
  - **Binding** of modules to operations in the graph
  - **Scheduling** of operations
  - **Placement** of modules on the array

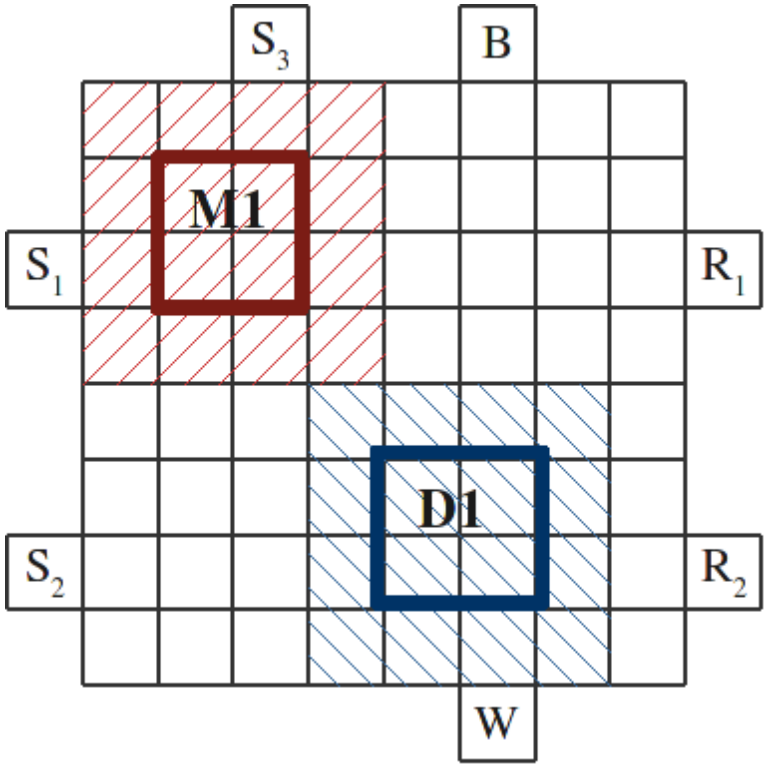
- **Such that**
  - the application execution time is minimized

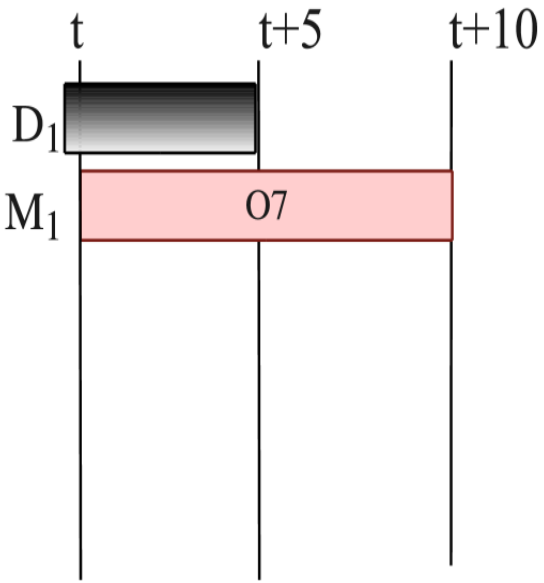
Reconfigurability



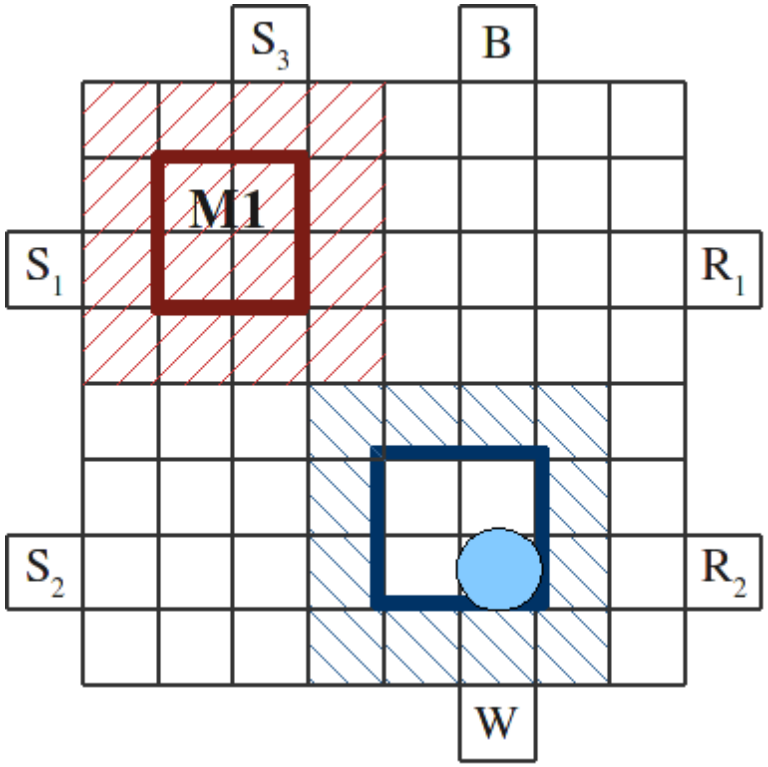


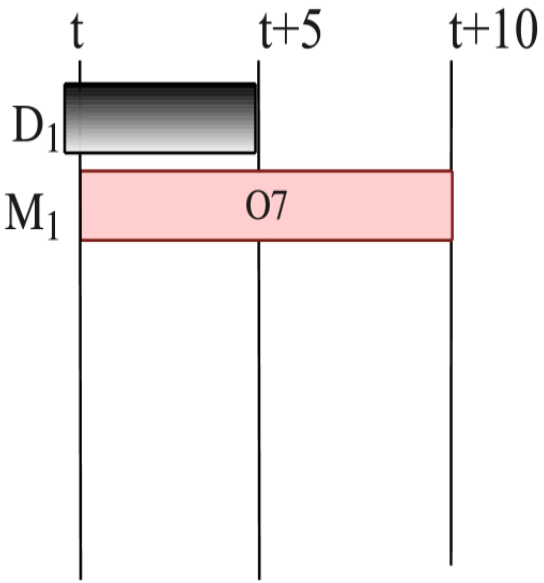
Reconfigurability



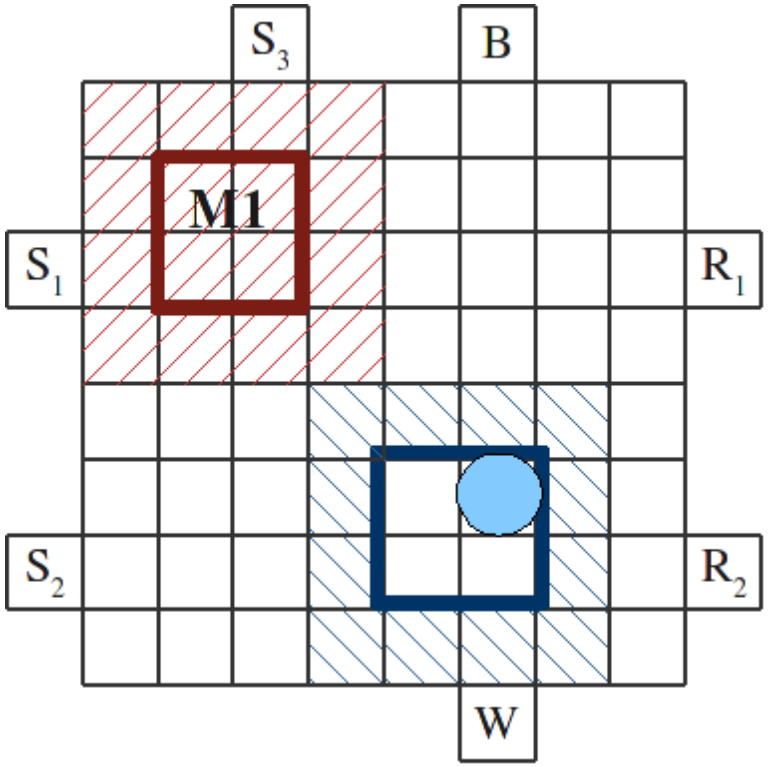


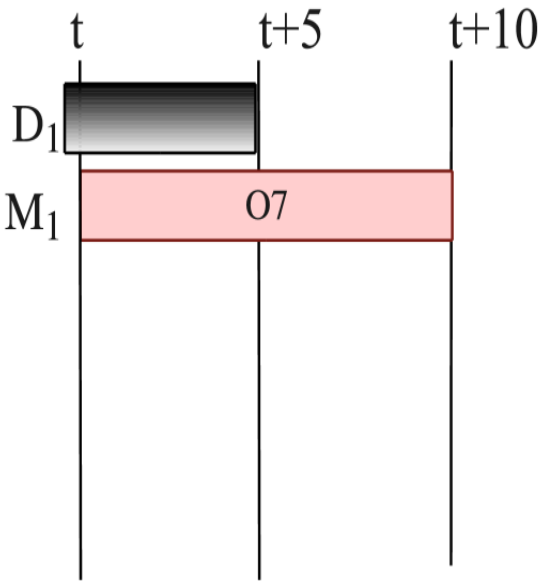
Reconfigurability



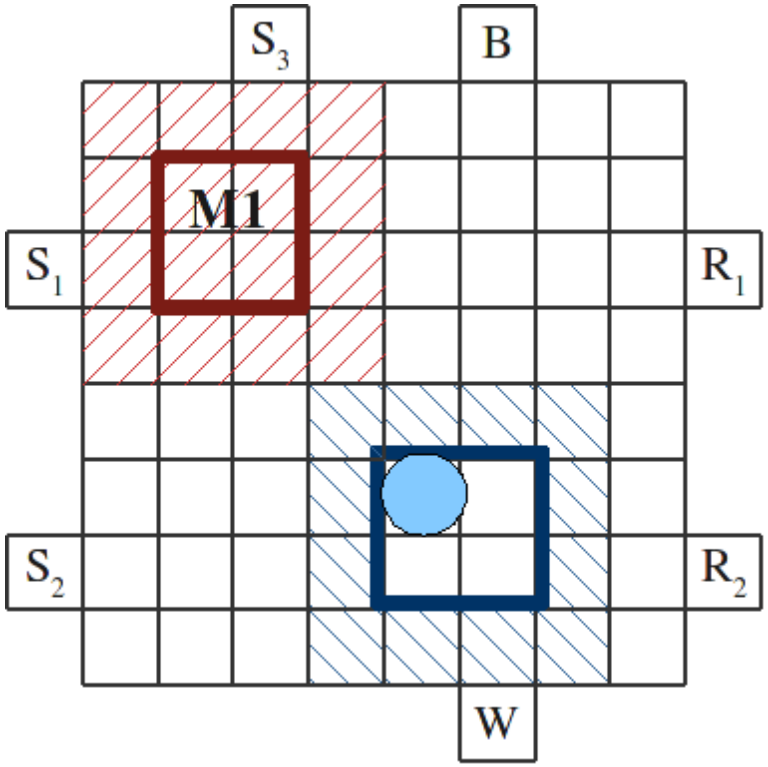


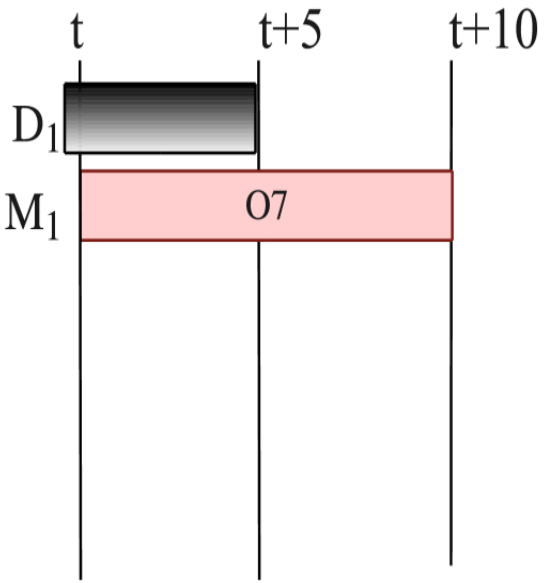
Reconfigurability



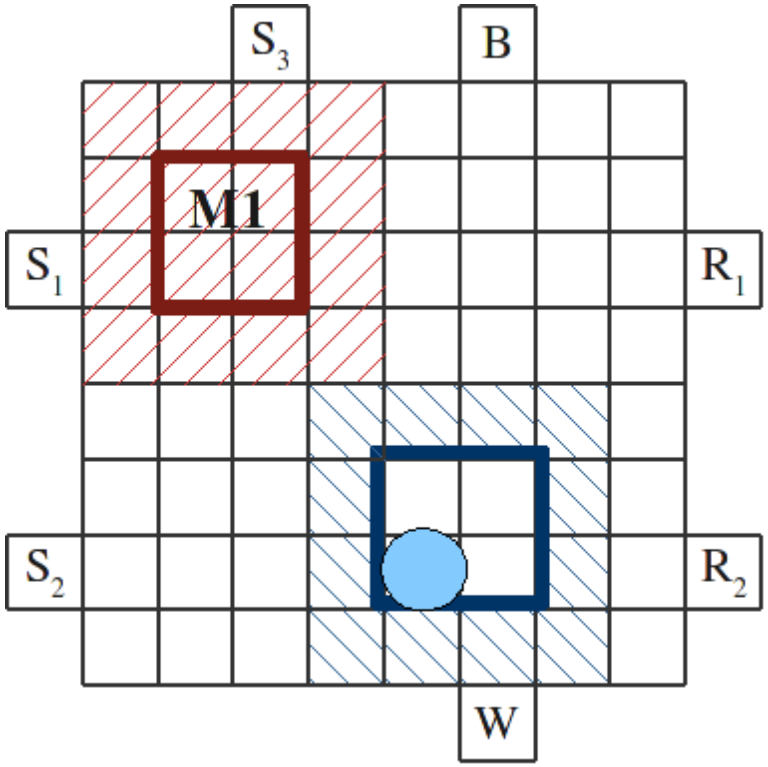


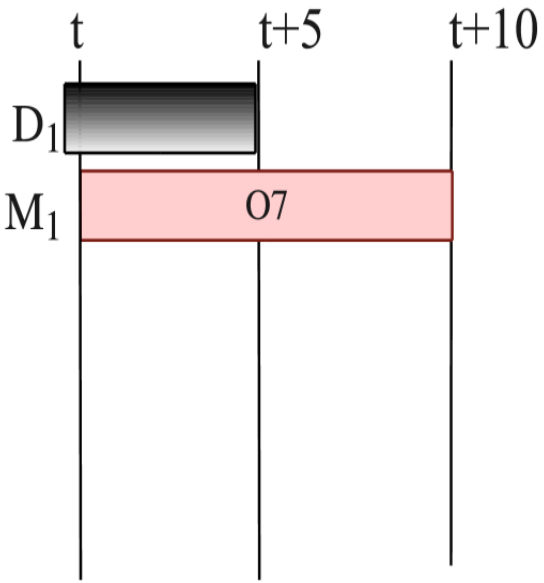
Reconfigurability



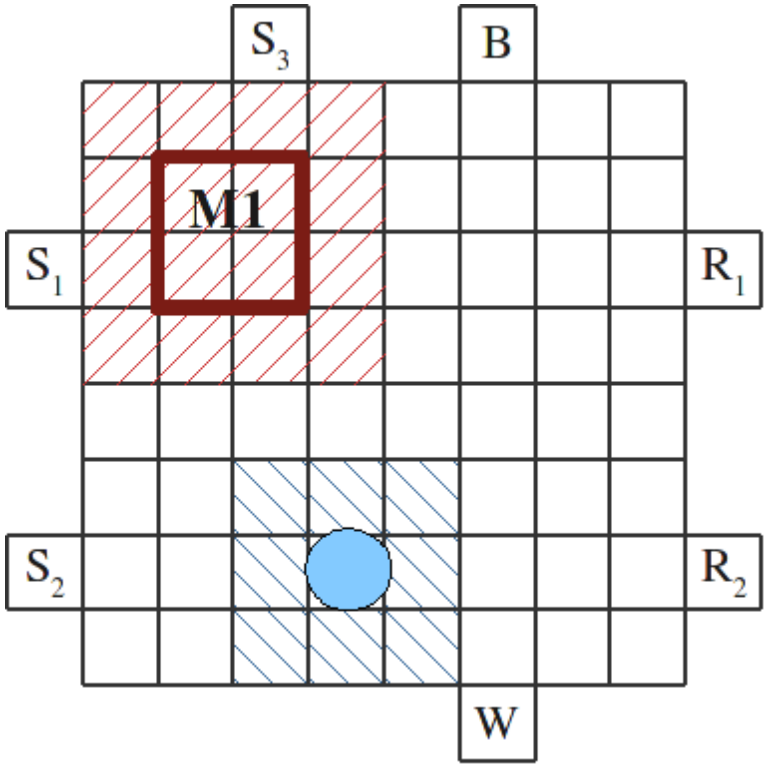


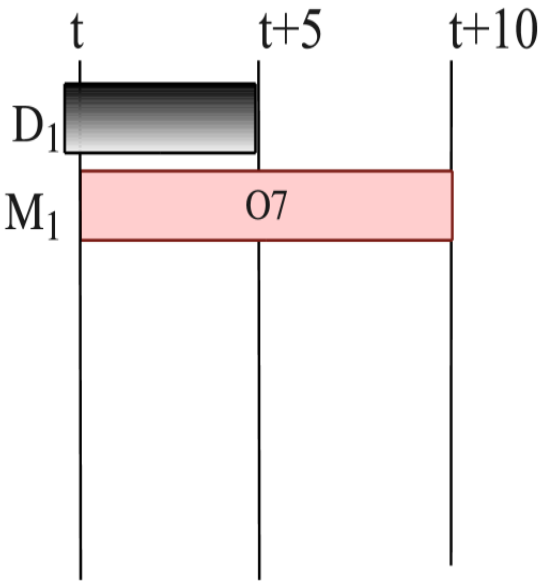
Reconfigurability



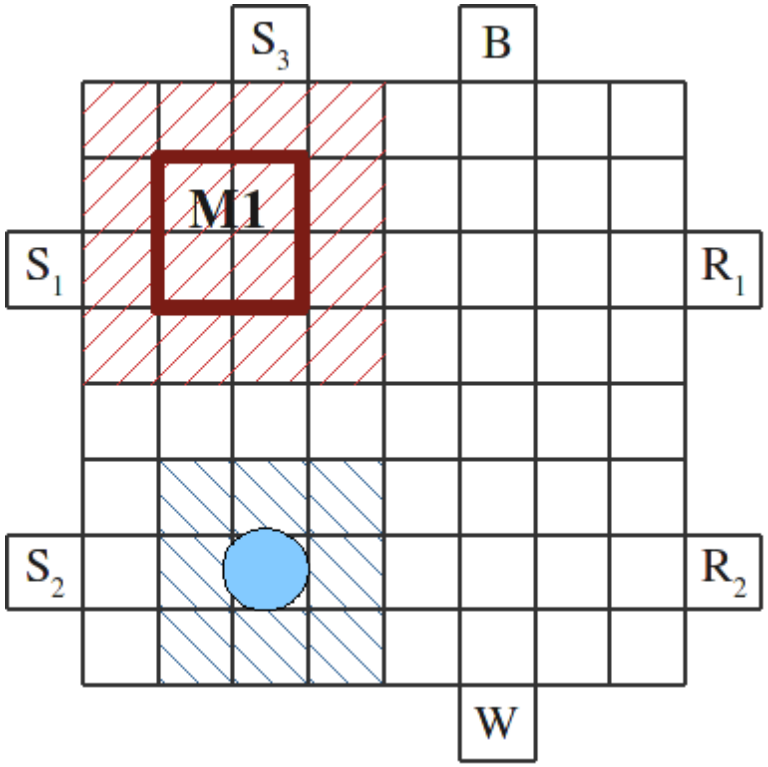


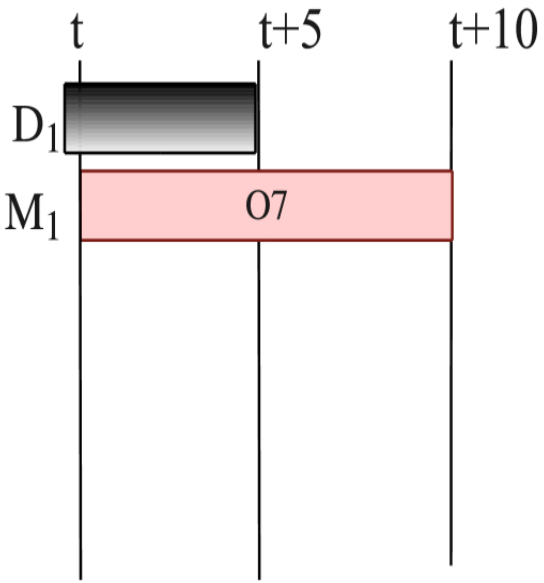
Reconfigurability



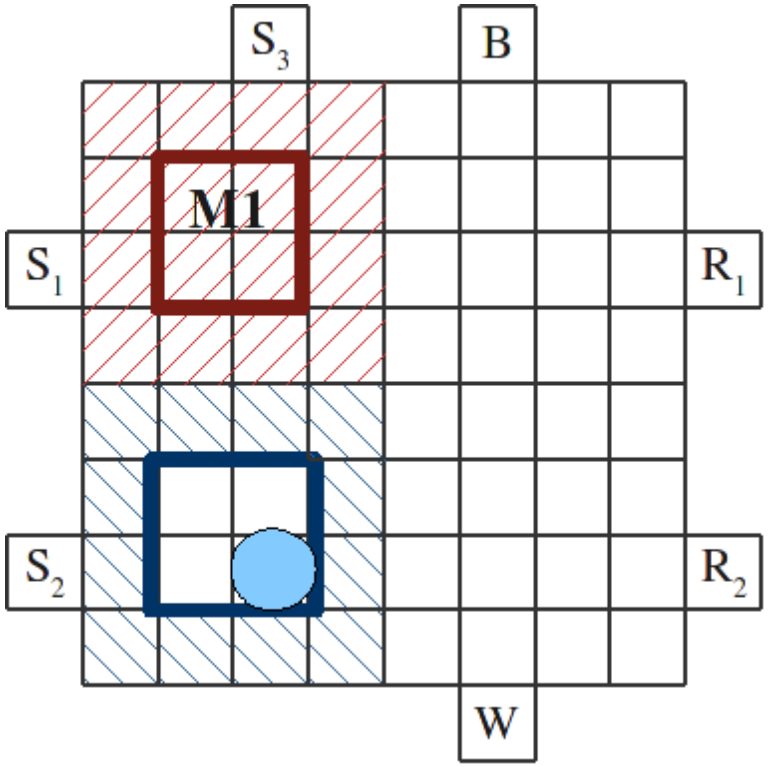


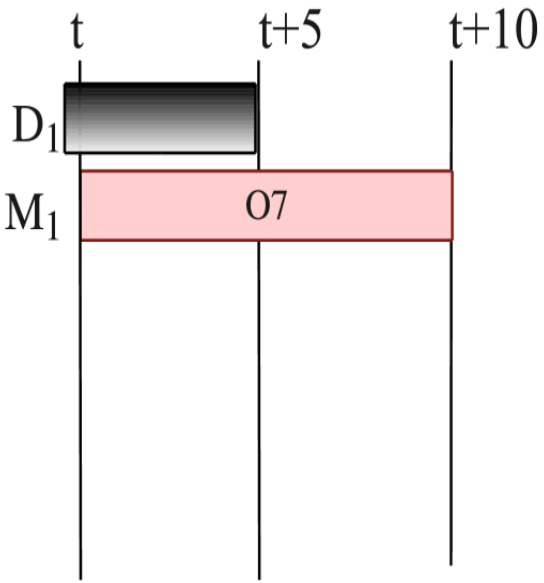
Reconfigurability



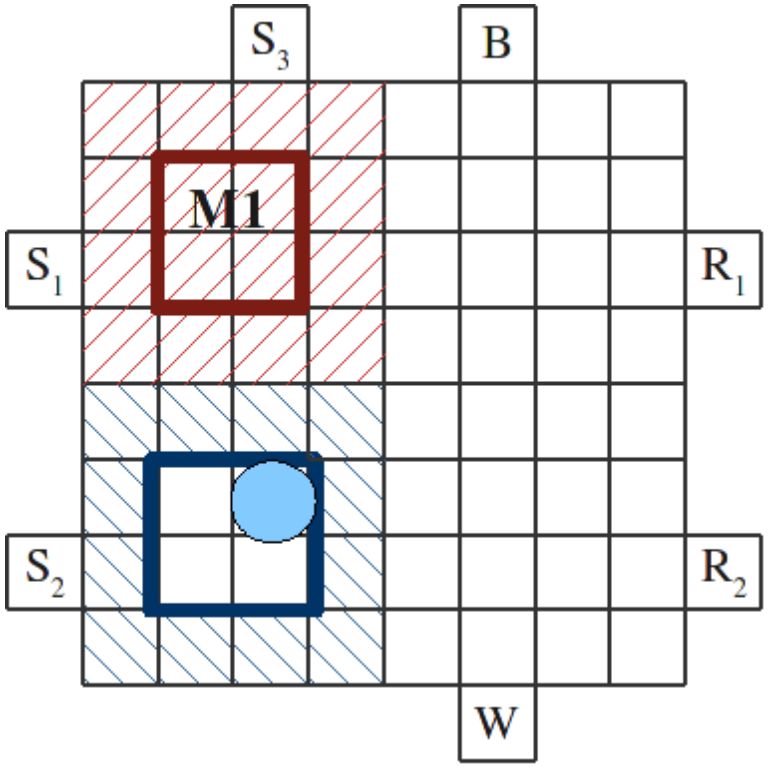


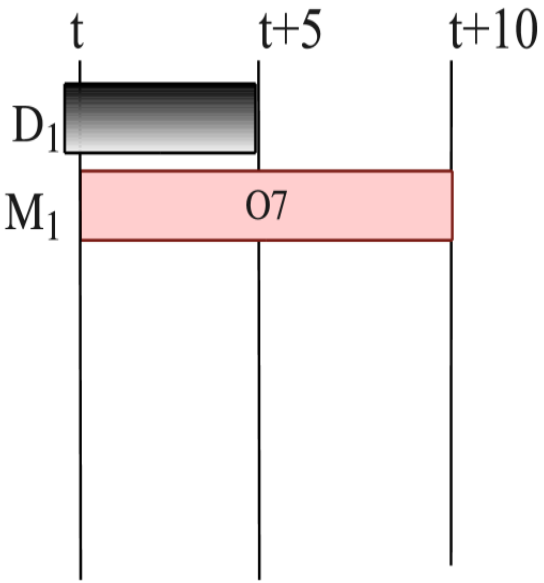
Reconfigurability



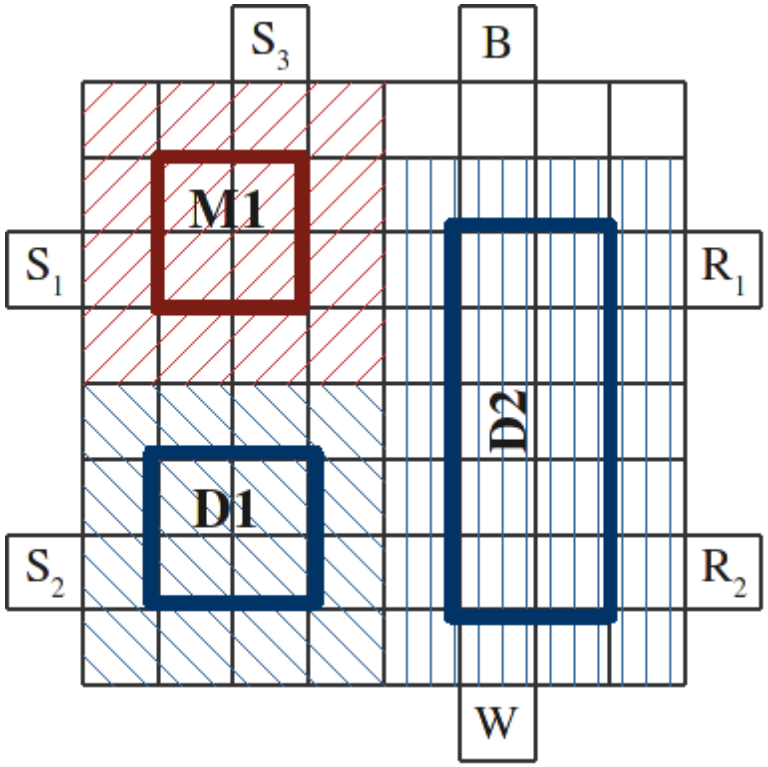


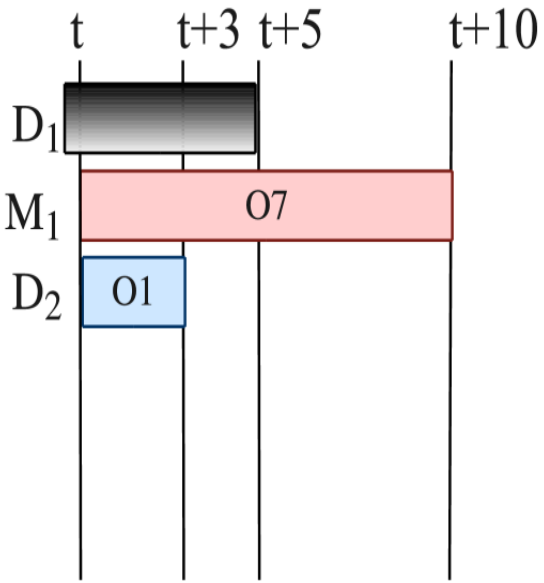
Reconfigurability



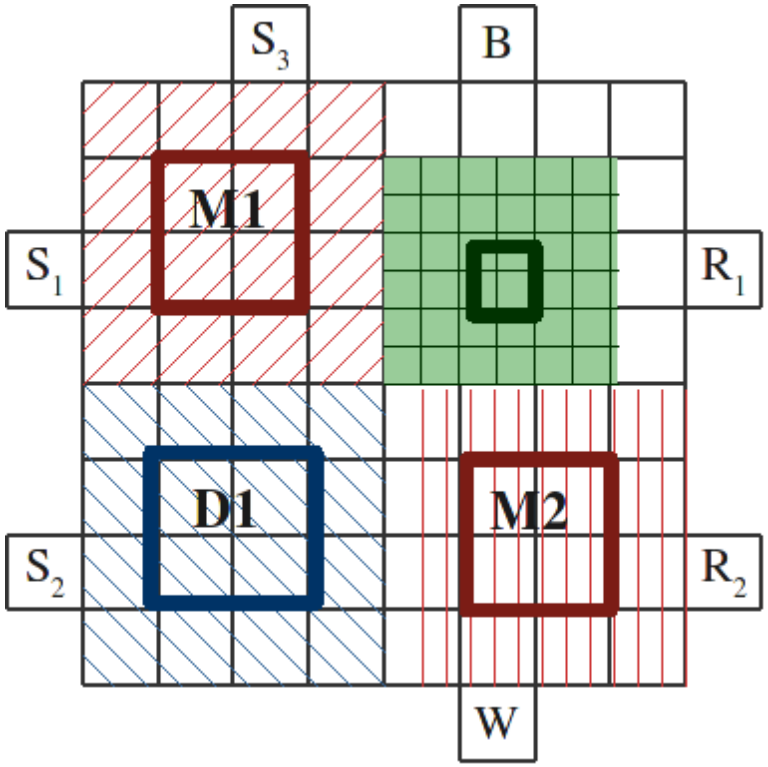


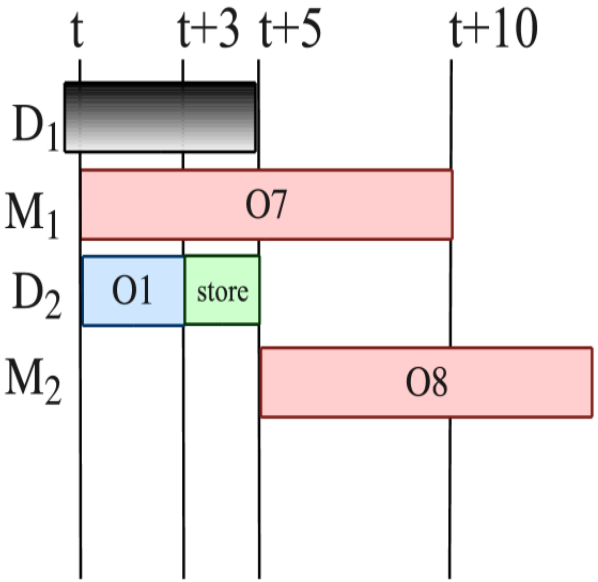
Reconfigurability



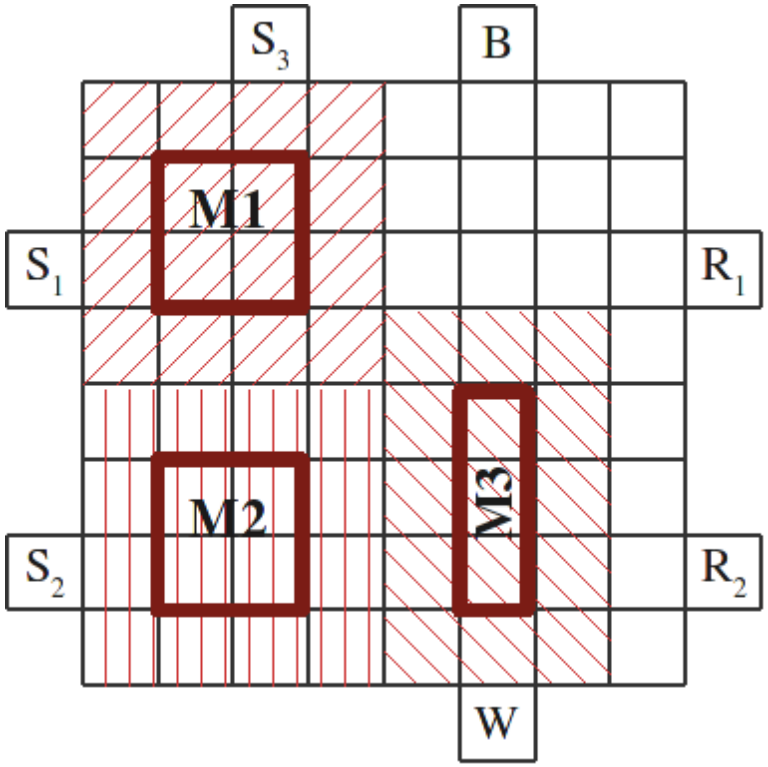


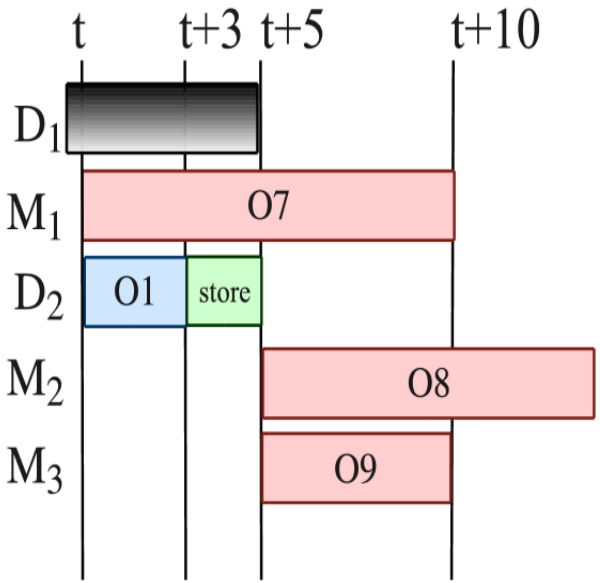
Reconfigurability



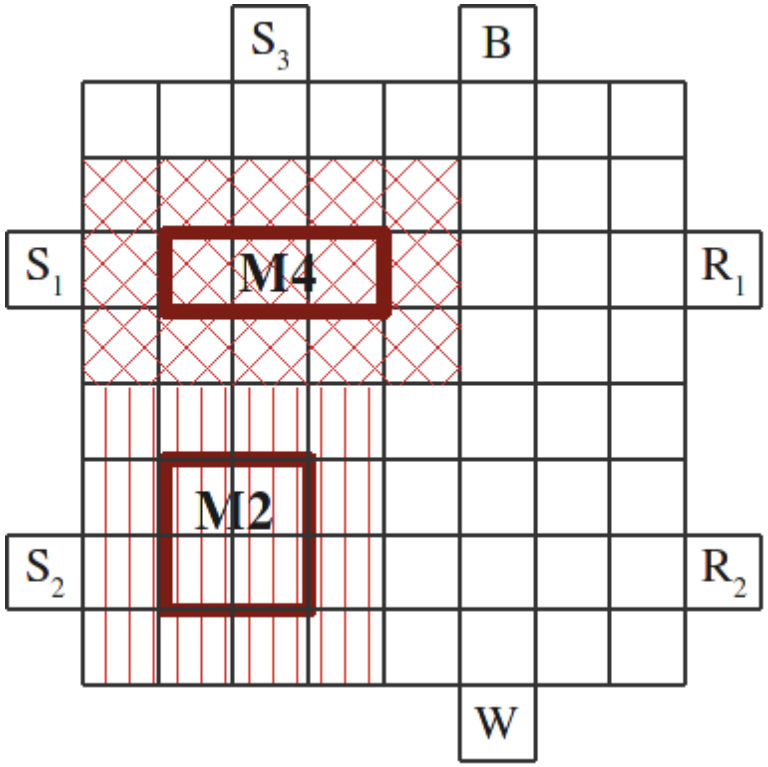


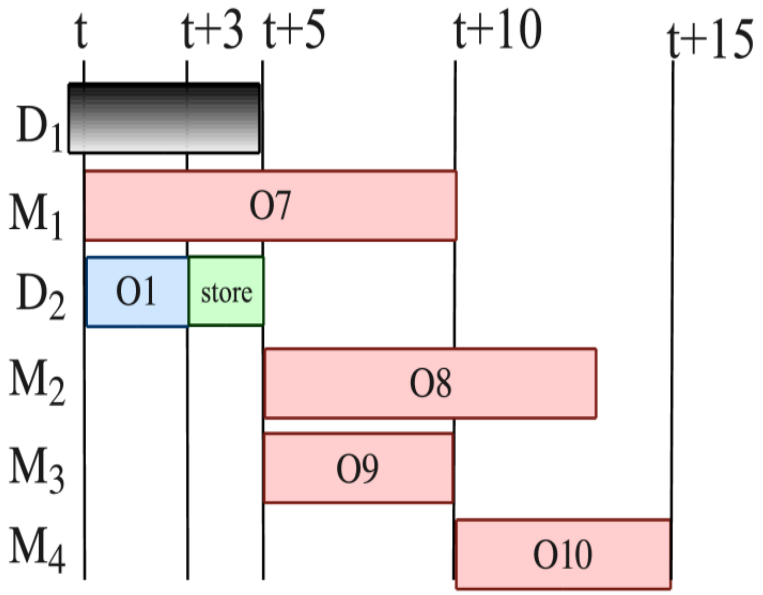
Reconfigurability



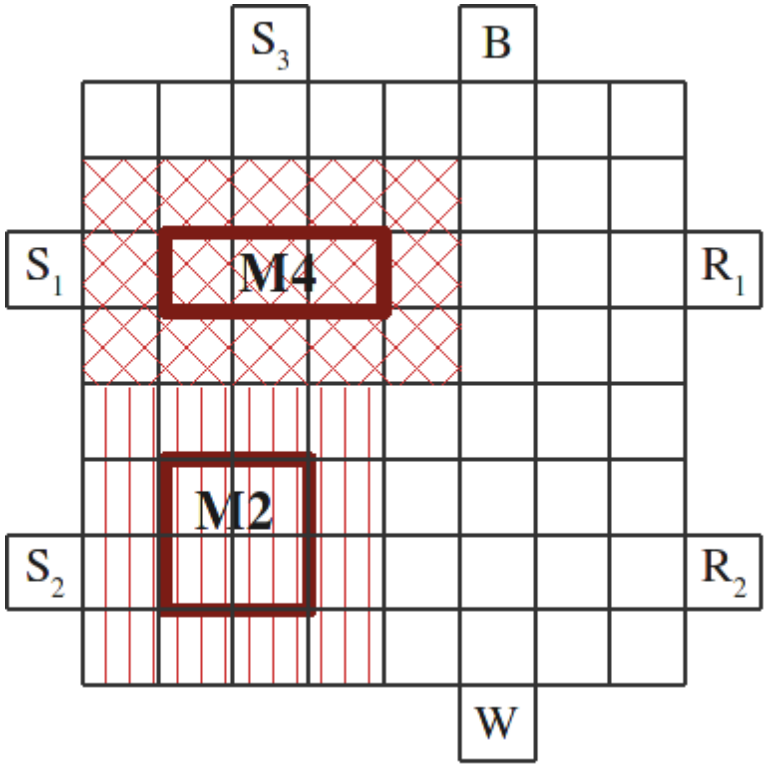


Reconfigurability

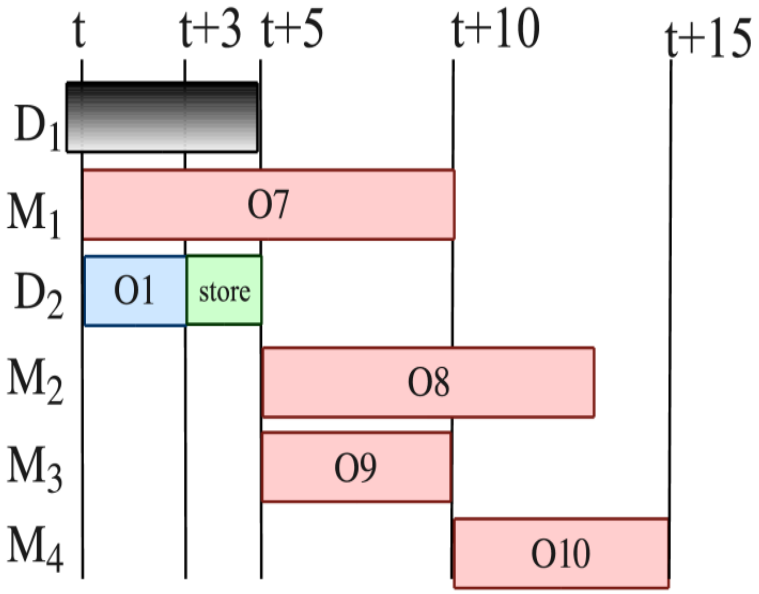




Reconfigurability



Without dynamic reconfiguration: t+18



Solution

- Binding of modules to operations

Tabu Search

- Schedule of the operations
  - Placement of modules performed inside scheduling

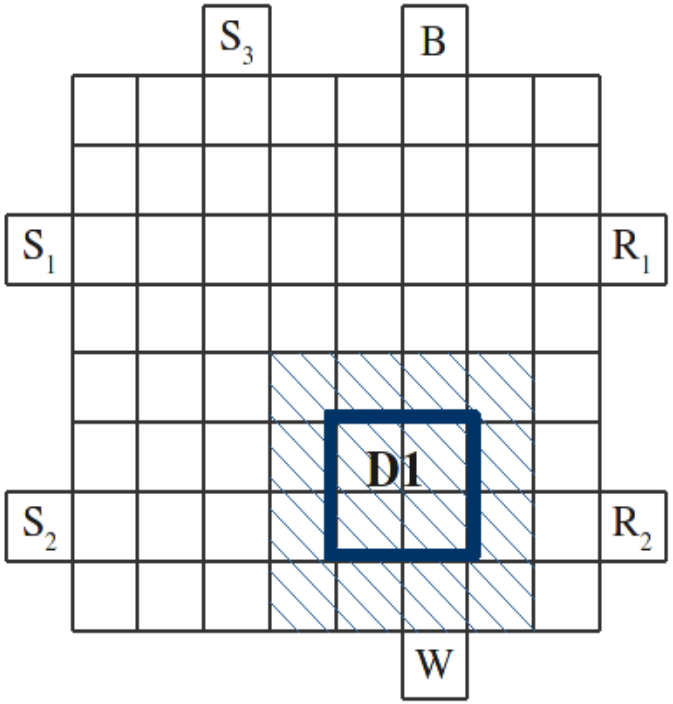
List Scheduling

- Placement of the modules
  - Free space manager based on [Bazargan et al. 2000] that divides free space on the chip into overlapping rectangles

Maximal Empty Rectangles

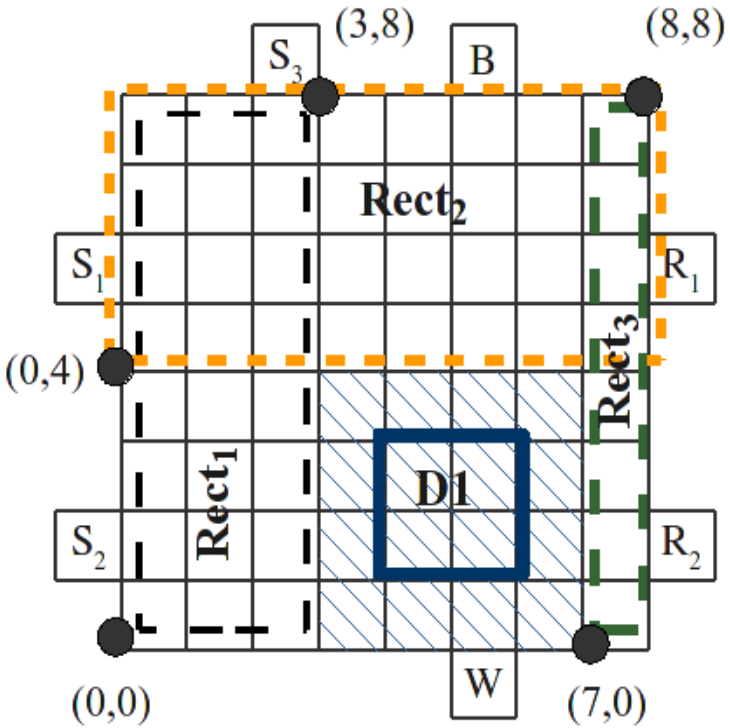
- Other solutions proposed in the literature:
  - Integer Linear Programming
  - Simulated Annealing

Dynamic Placement Algorithm

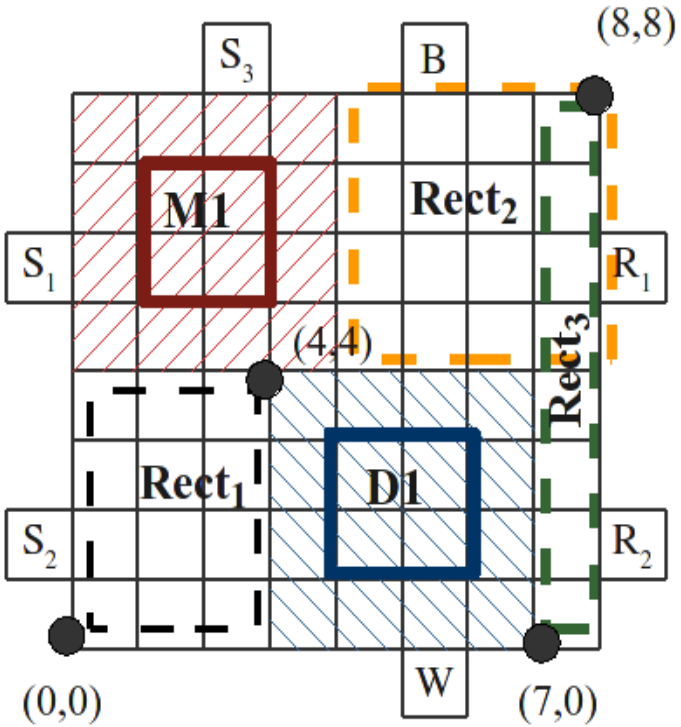


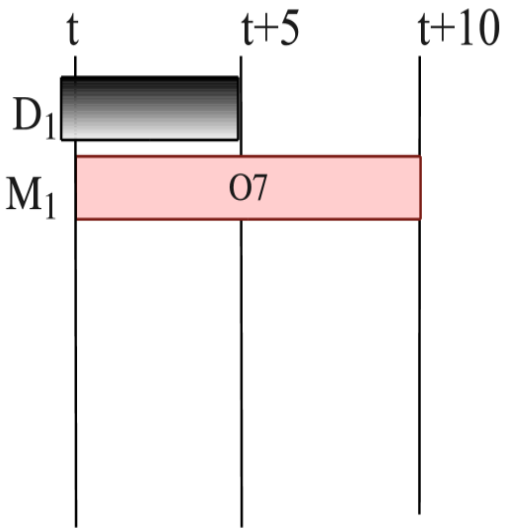
Operation	Module
O_7 (mix)	M_1 (2x2)
O_1 (diluter)	D_2 (2x5)

Dynamic Placement Algorithm

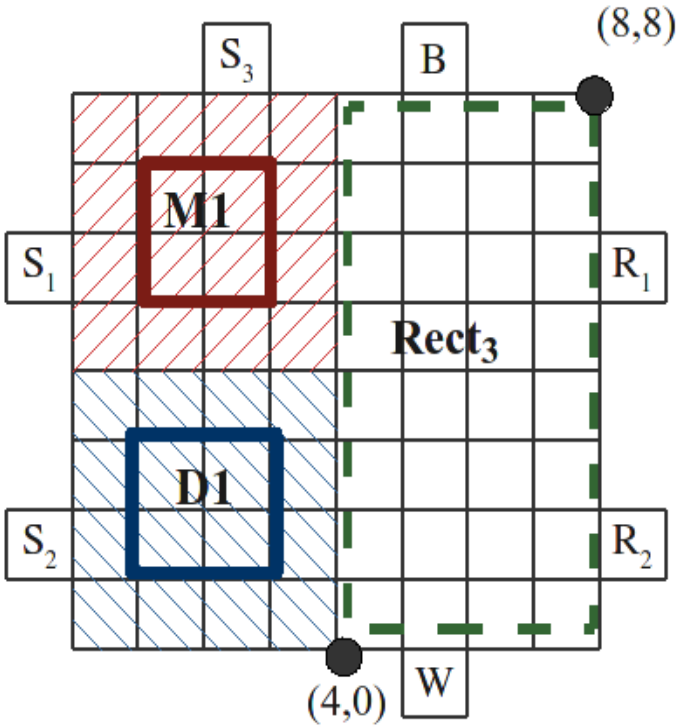


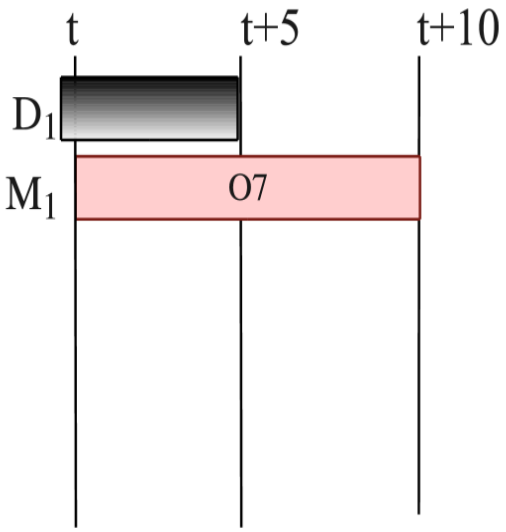
Dynamic Placement Algorithm



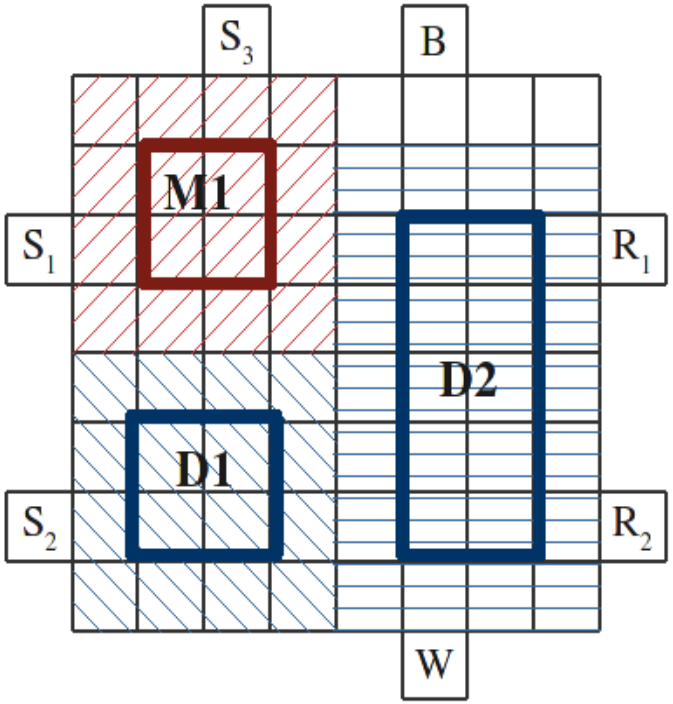


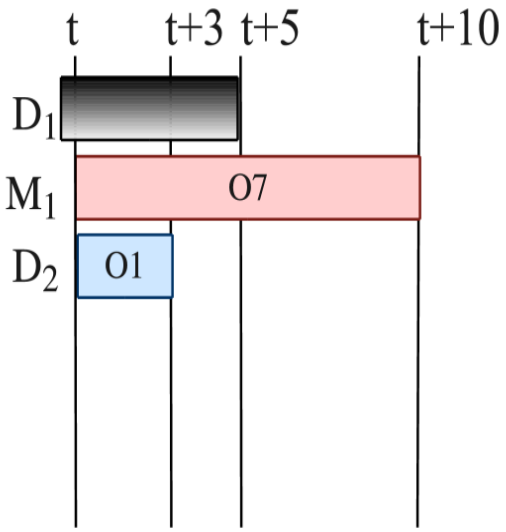
Dynamic Placement Algorithm



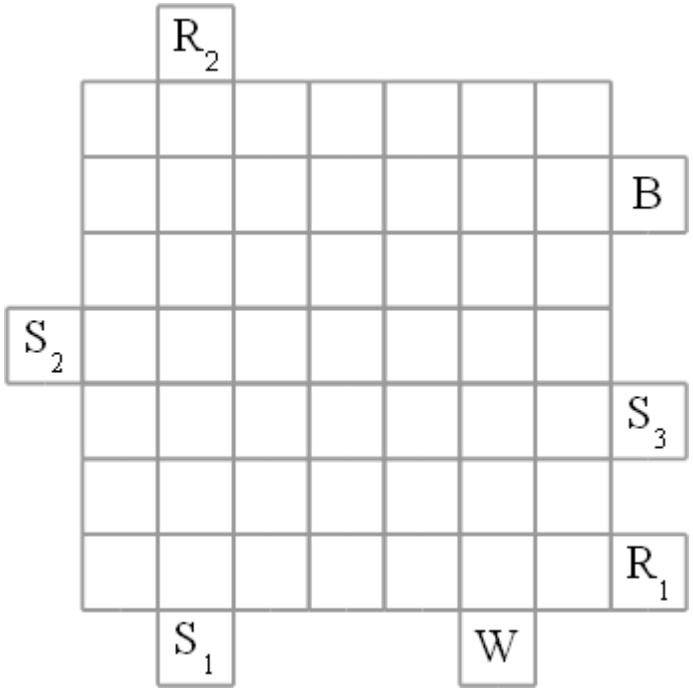


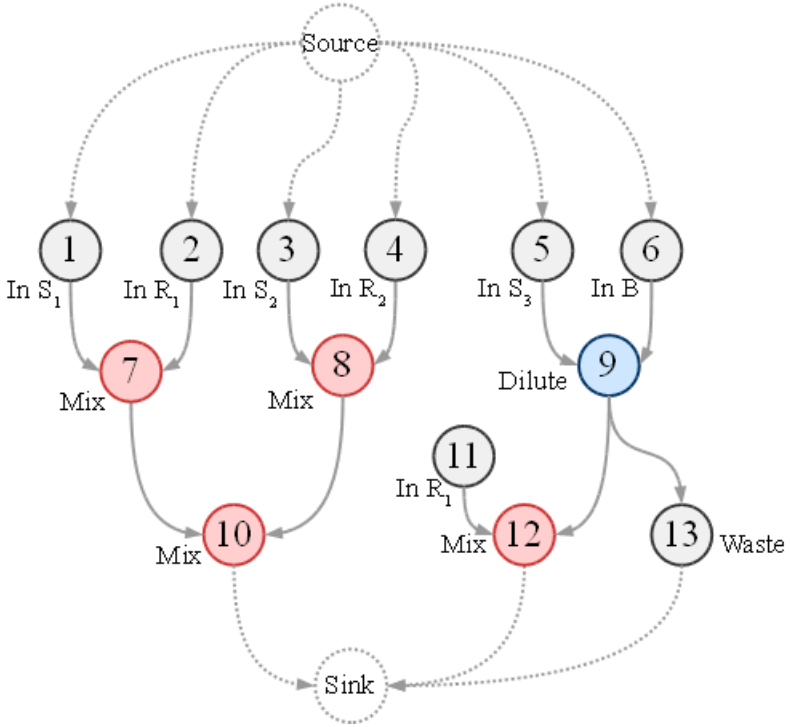
Dynamic Placement Algorithm





Routing-Based Synthesis



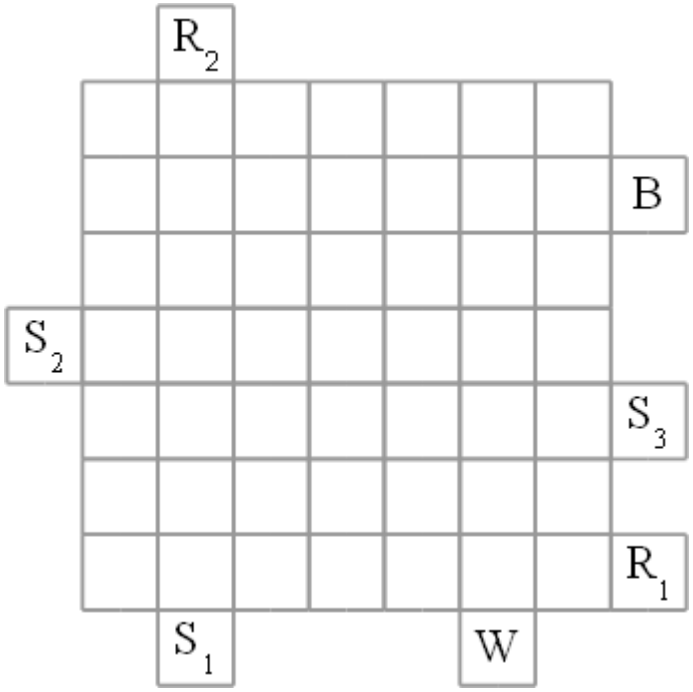


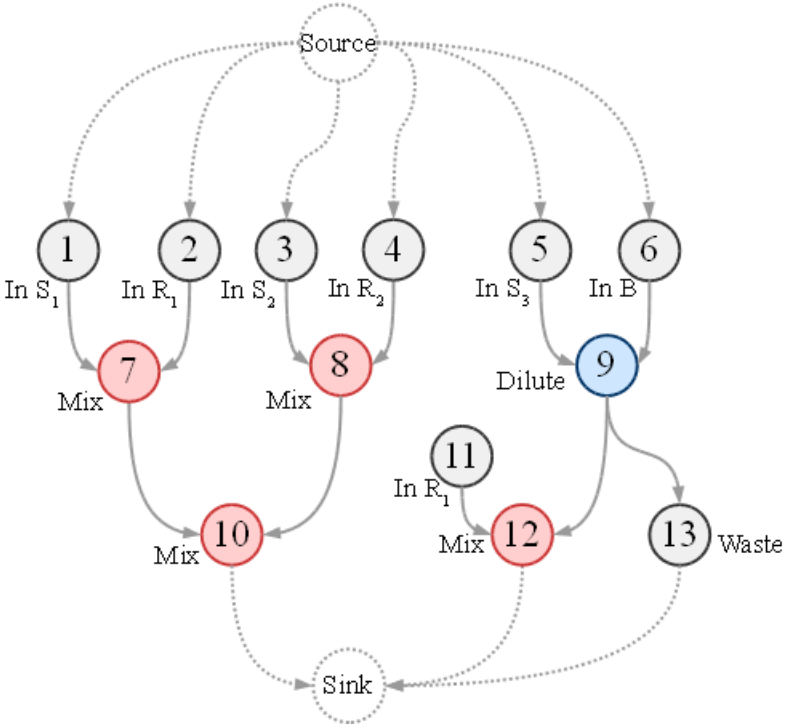
Routing-Based Synthesis

O7

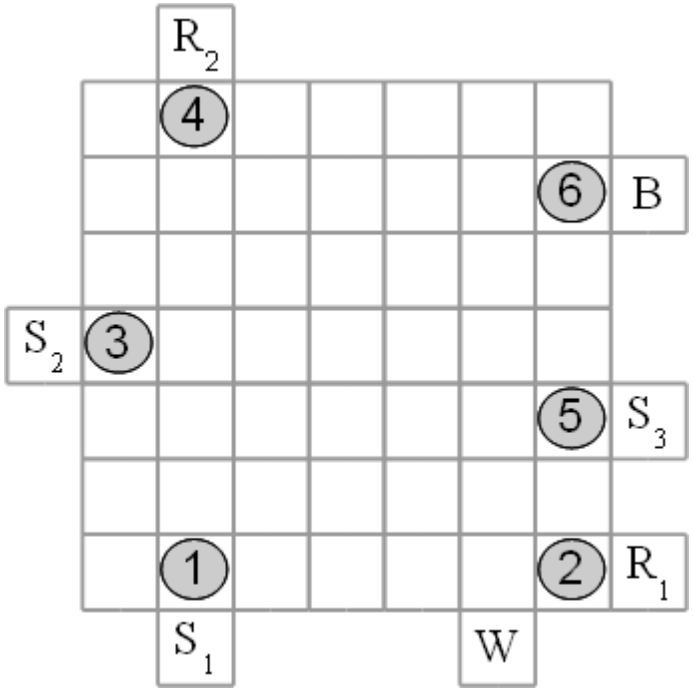
O8

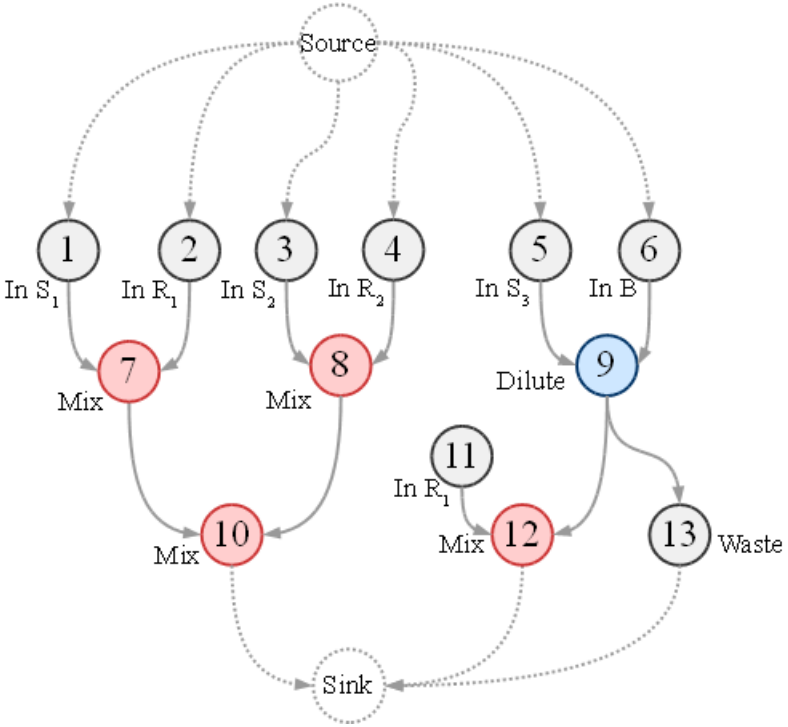
O9



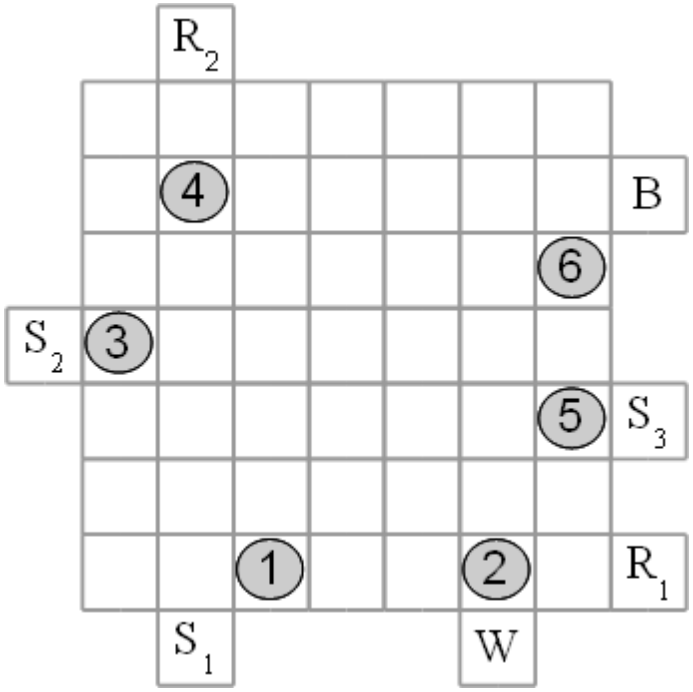


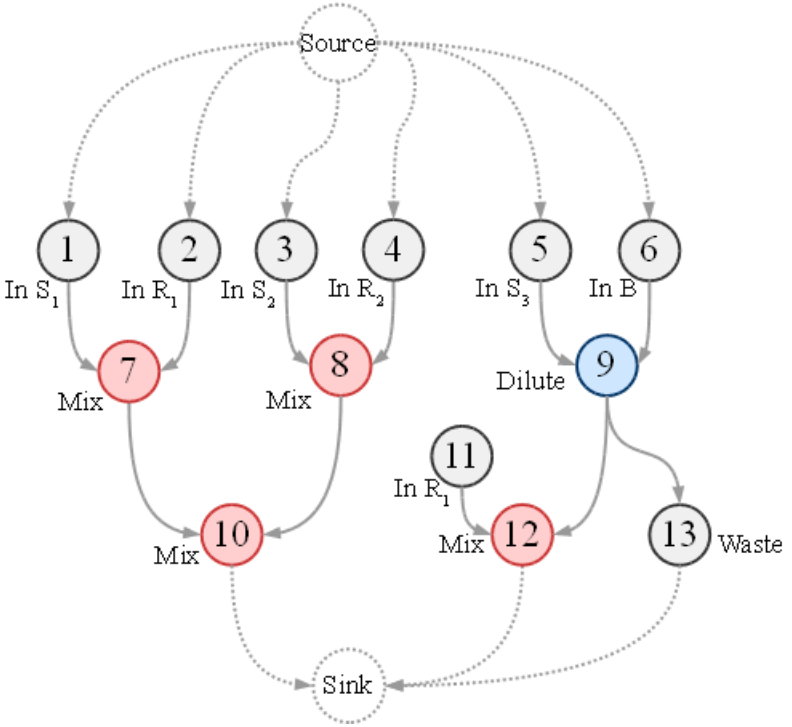
Routing-Based Synthesis



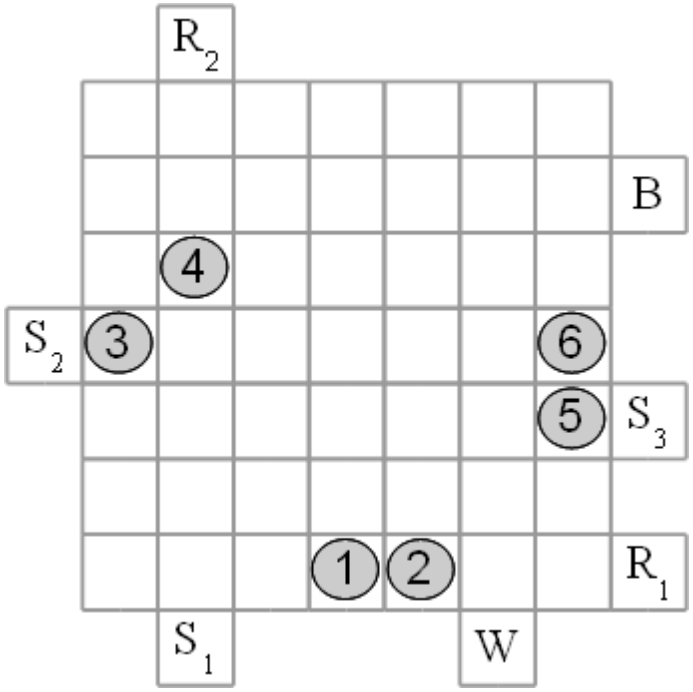


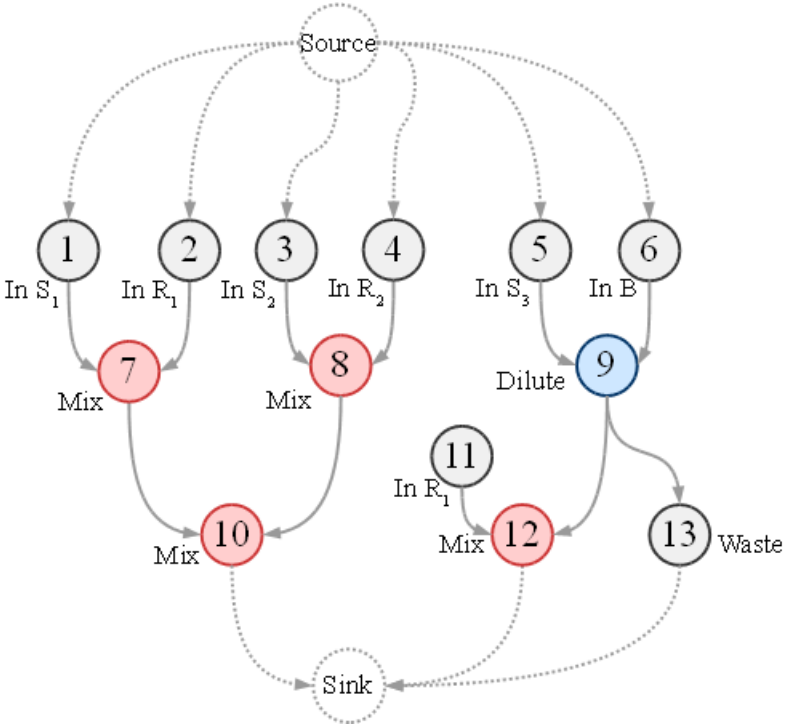
Routing-Based Synthesis



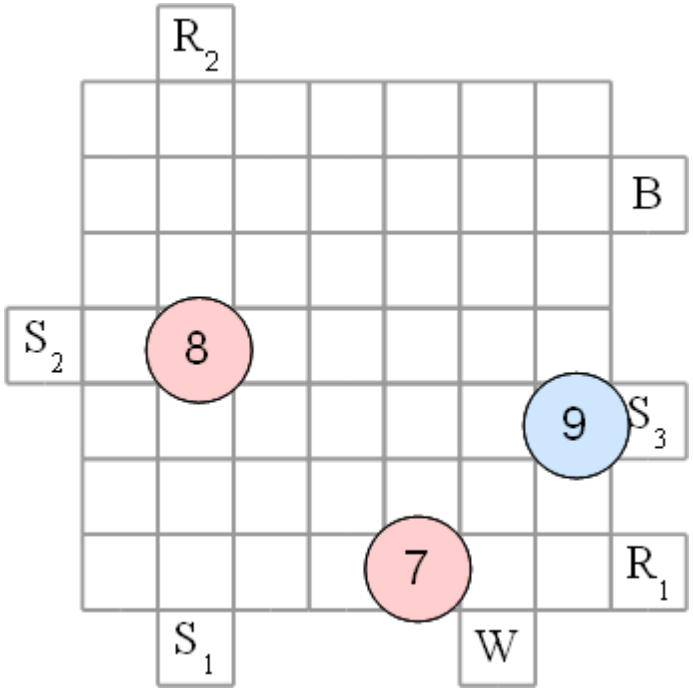


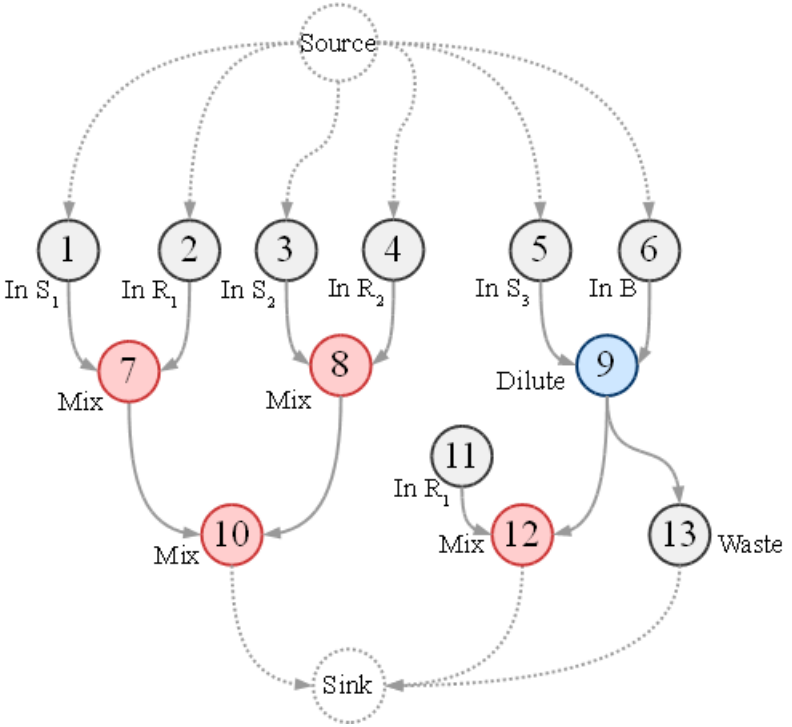
Routing-Based Synthesis



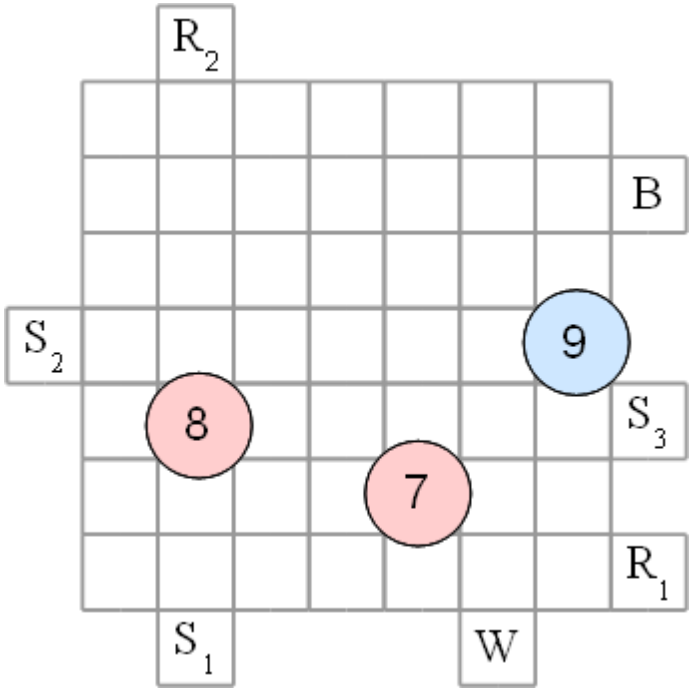


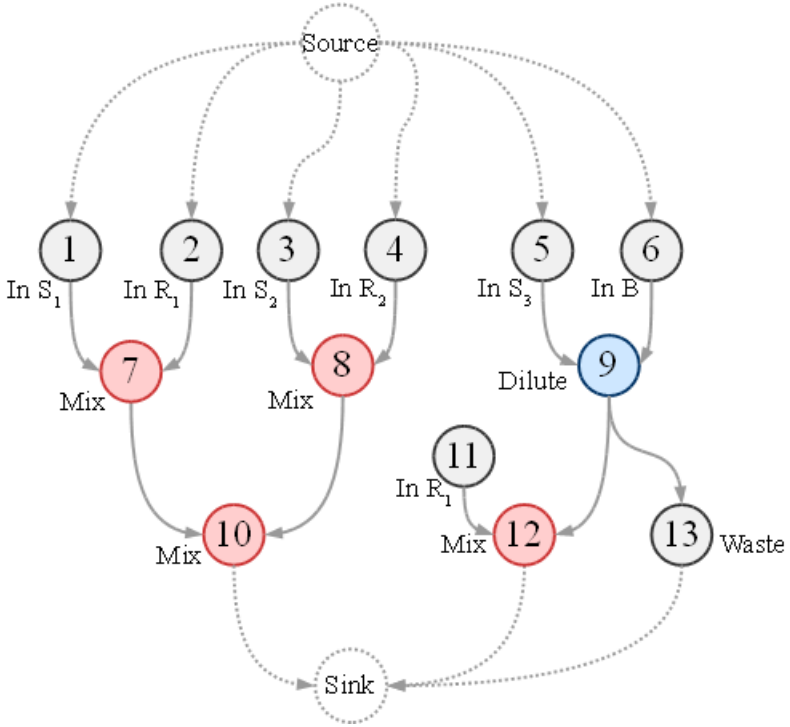
Routing-Based Synthesis



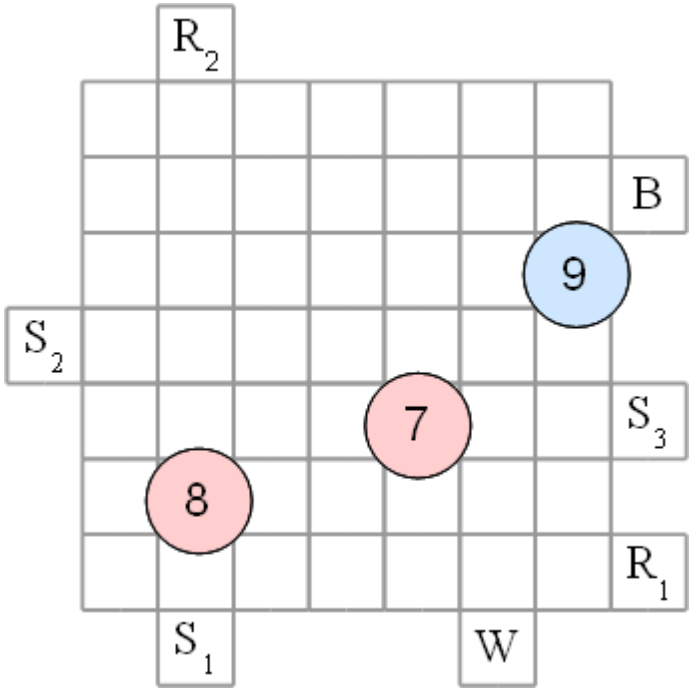


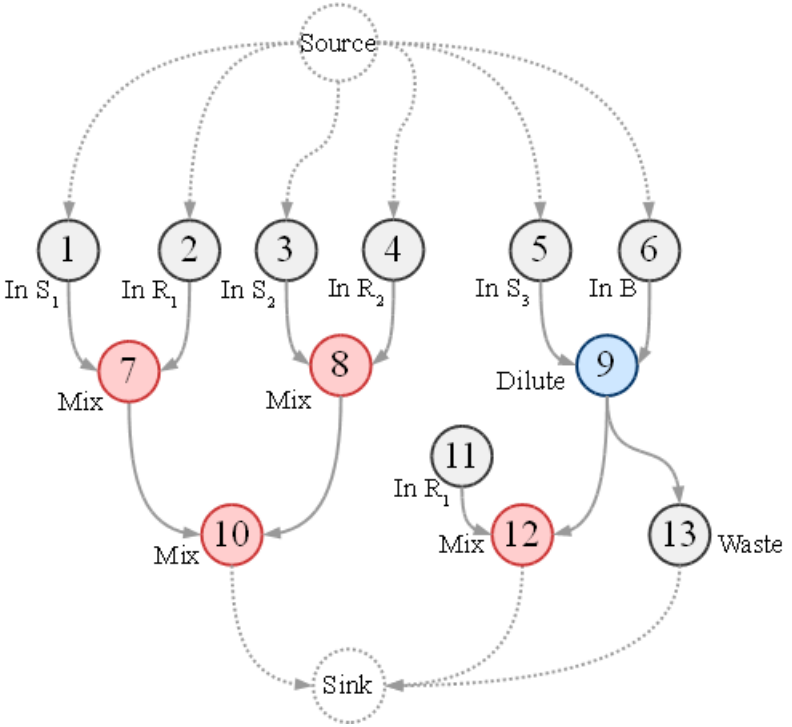
Routing-Based Synthesis



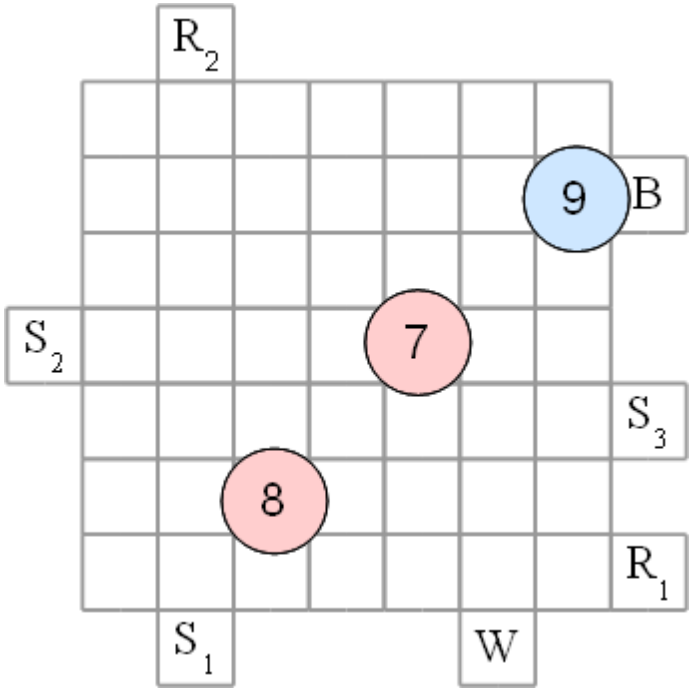


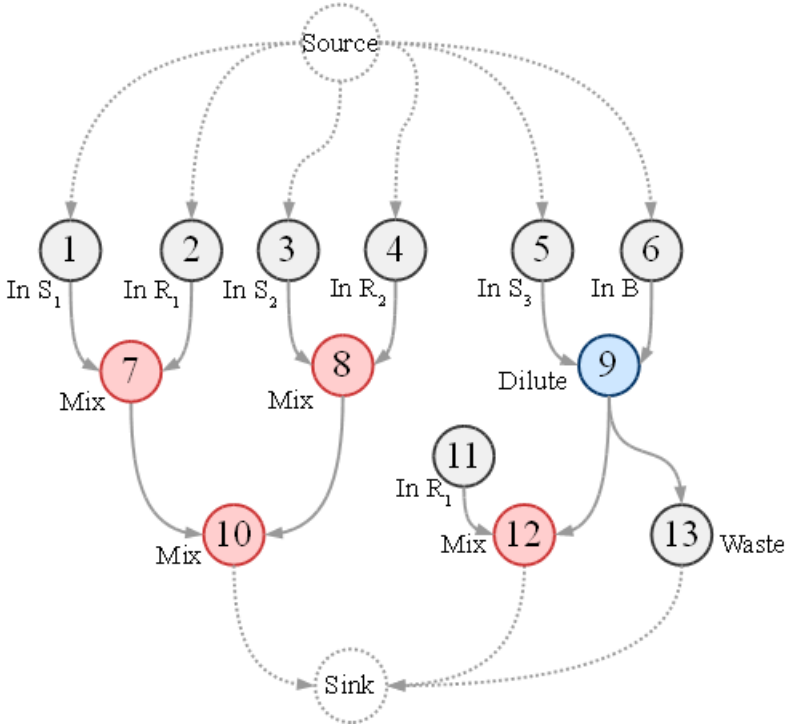
Routing-Based Synthesis



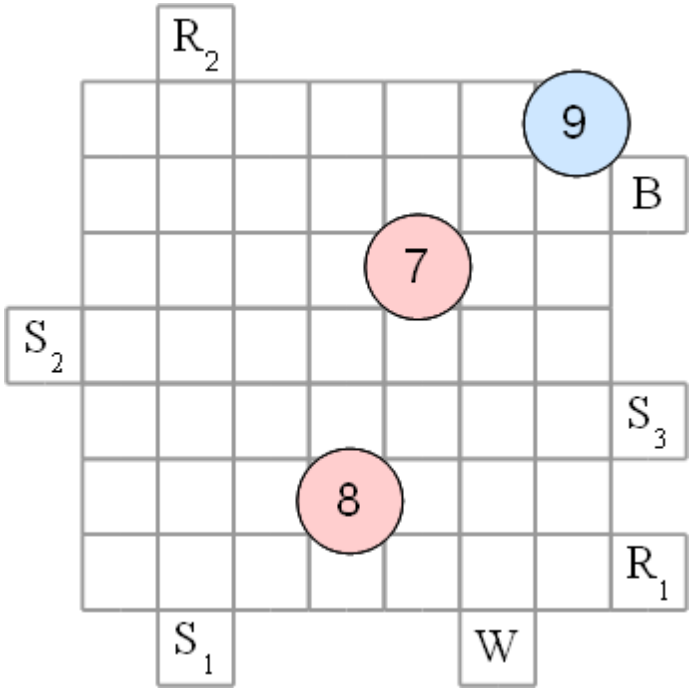


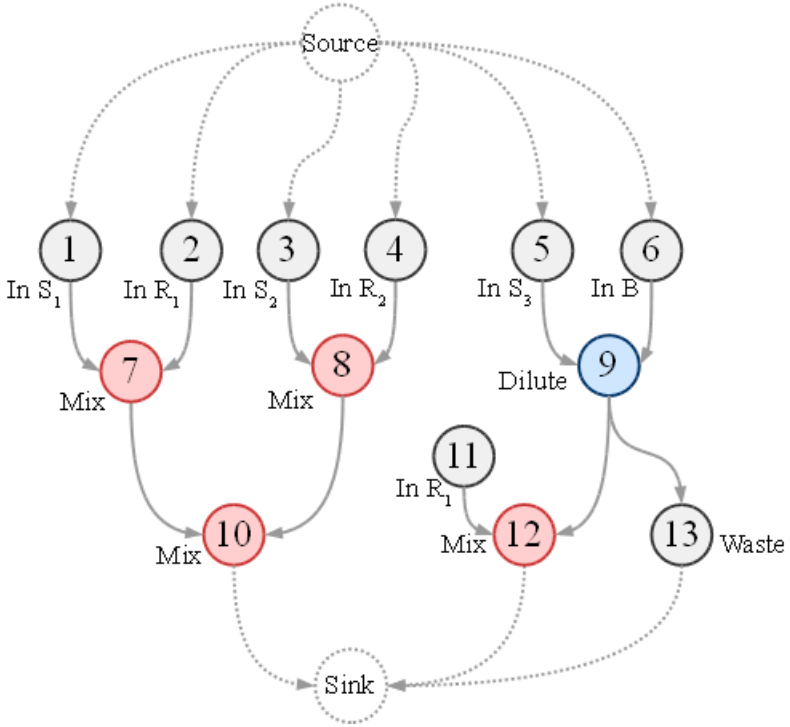
Routing-Based Synthesis



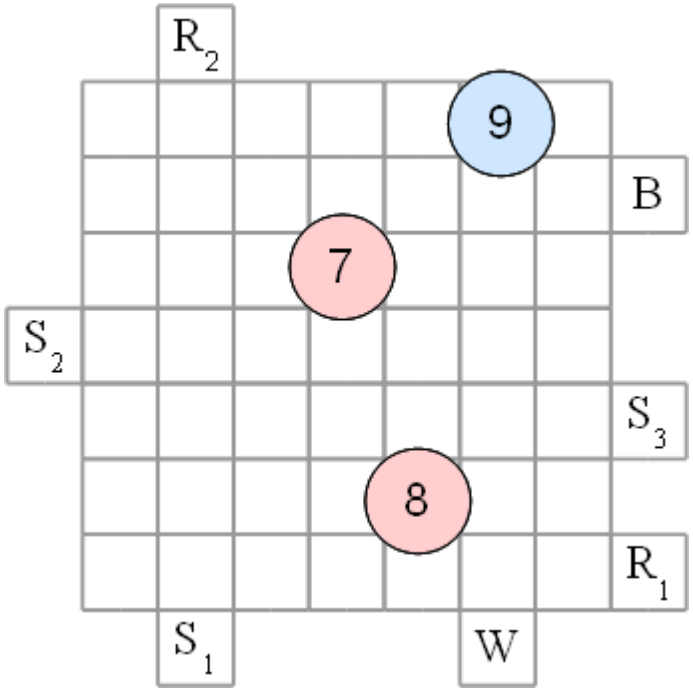


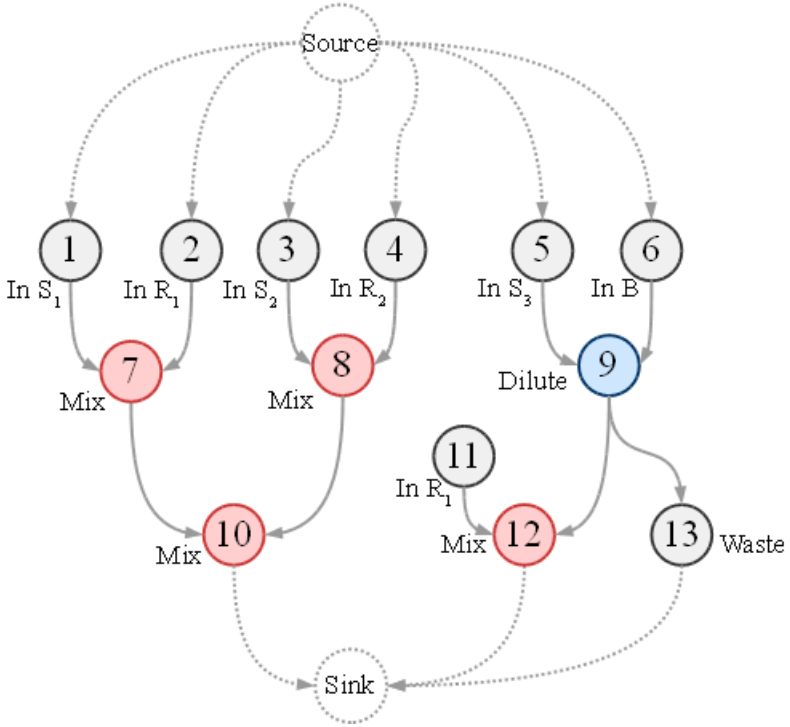
Routing-Based Synthesis



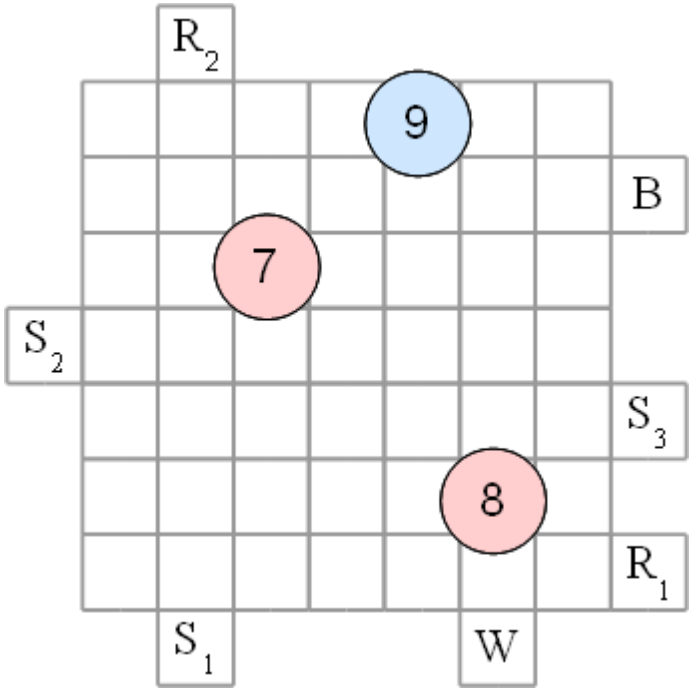


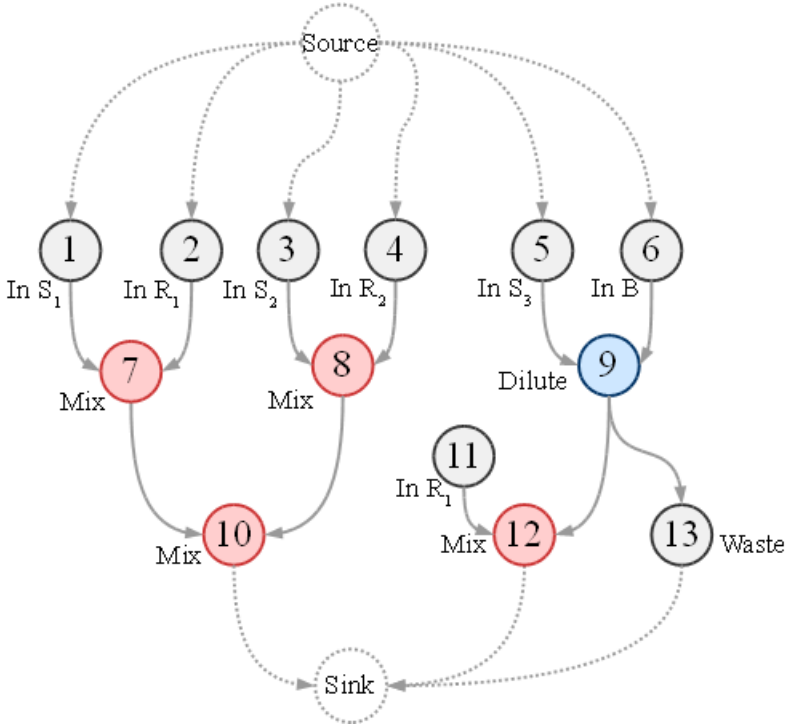
Routing-Based Synthesis



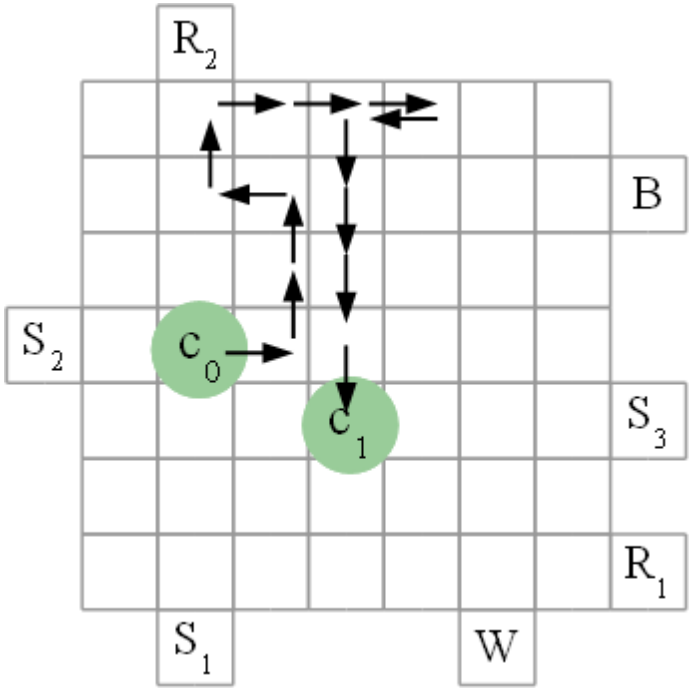


Routing-Based Synthesis



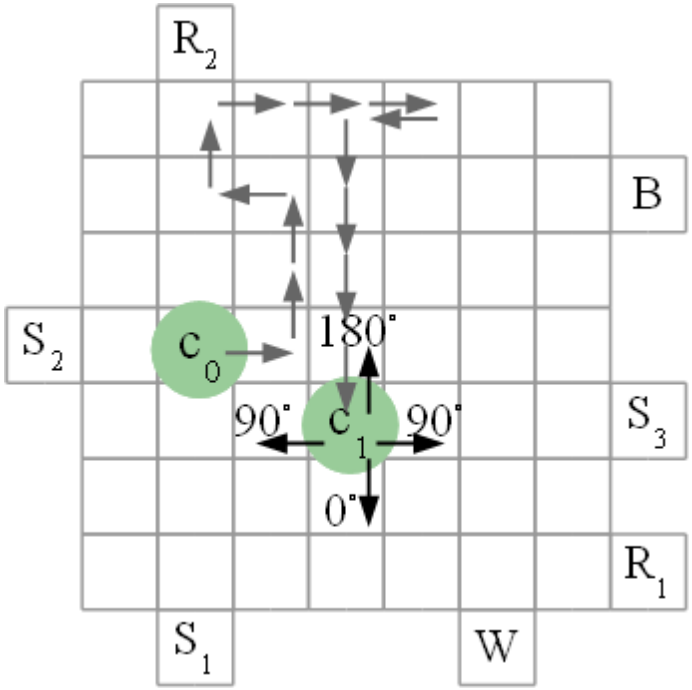


When will the operations complete?



- For module-based synthesis we know the *completion time* from the module library.
- But now there are no modules, the droplets can move anywhere:
  - How can we find out the operation *completion times*?

Characterizing operations



- If the droplet does not move: very slow mixing by diffusion
- If the droplet moves, how long does it take to complete?
- Mixing percentages:

p^0, p^{90}, p^{180} ?

Characterizing operations

Operation	Area(cells)	Time(s)
Mix/Dlt	2x4	2.8
Mix/Dlt	1x4	4.6
Mix/Dlt	2x3	5.6
Mix/Dlt	2x2	9.96

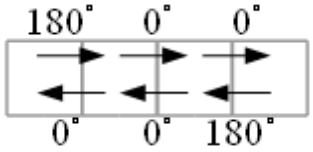
- We know how long an operation takes on modules
- Starting from this, can determine the percentages?

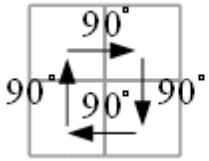
Decomposing modules

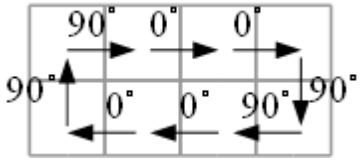
Safe, conservative estimates

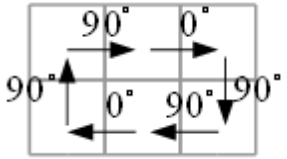
p^{90} = 0.1%, p^{180} = -0.5%,
p^{0} = 0.29% and 0.58%

Operation	Area(cells)	Time(s)
Mix/Dlt	2x4	2.8
Mix/Dlt	1x4	4.6
Mix/Dlt	2x3	5.6
Mix/Dlt	2x2	9.96



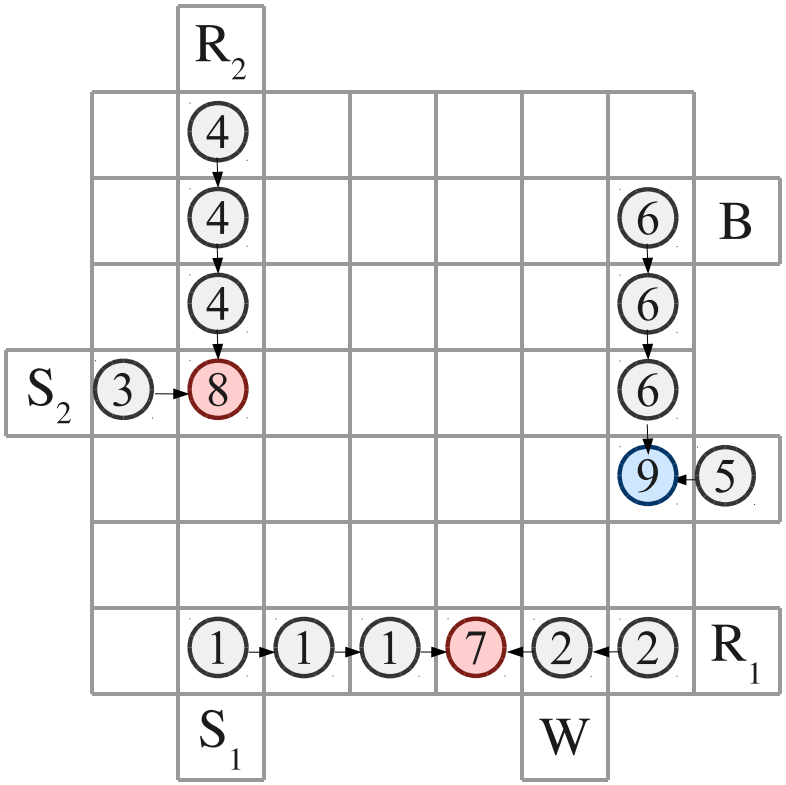


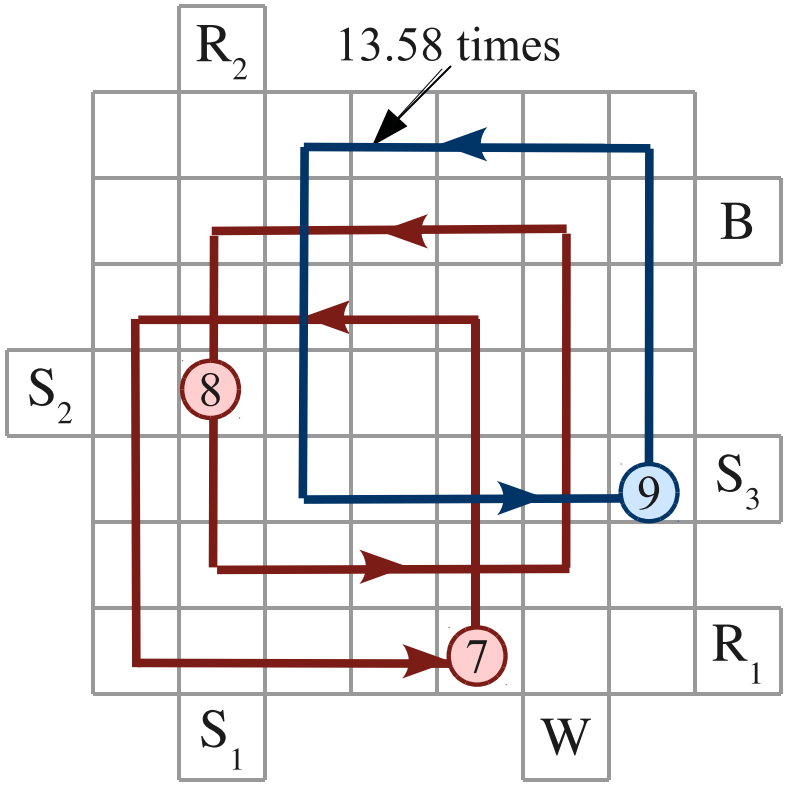




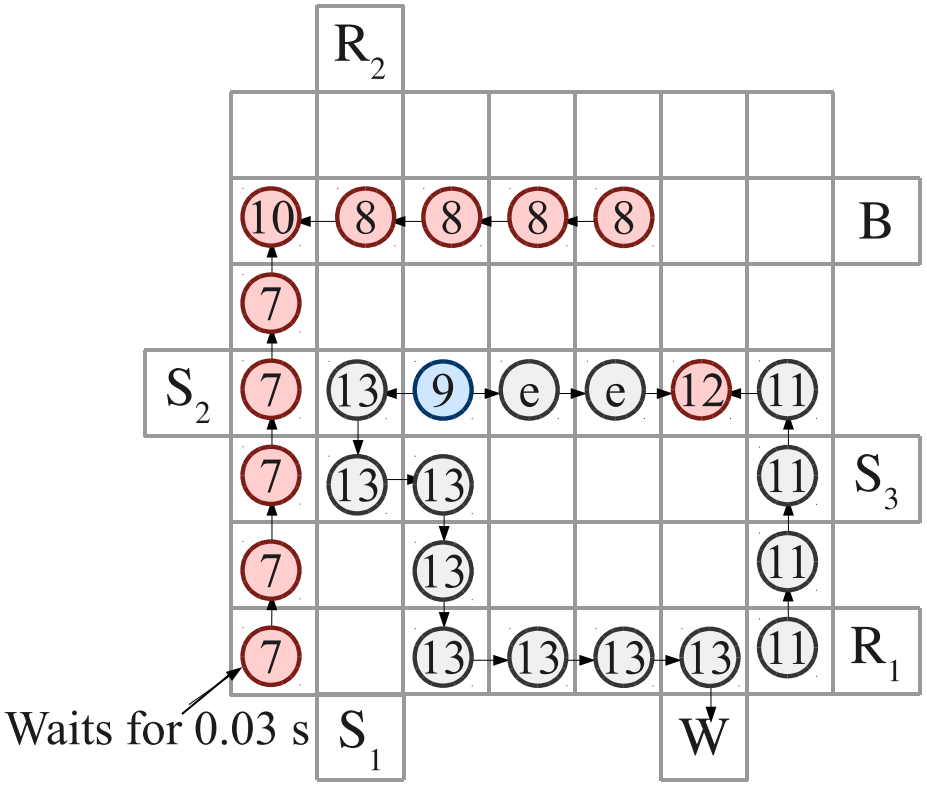
Moving a droplet one cell takes 0.01 s.

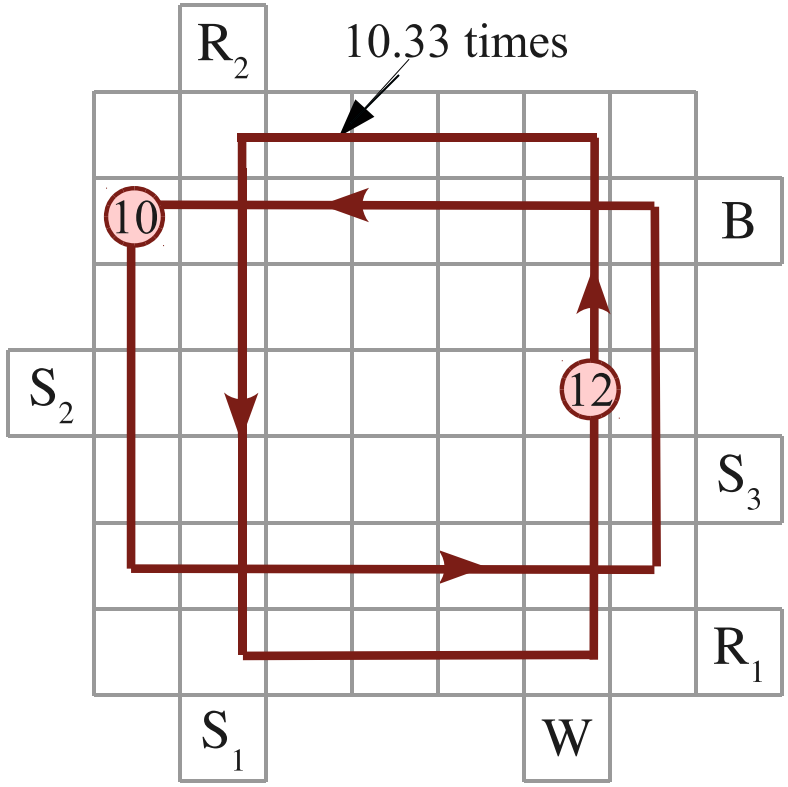
Routing-Based Synthesis



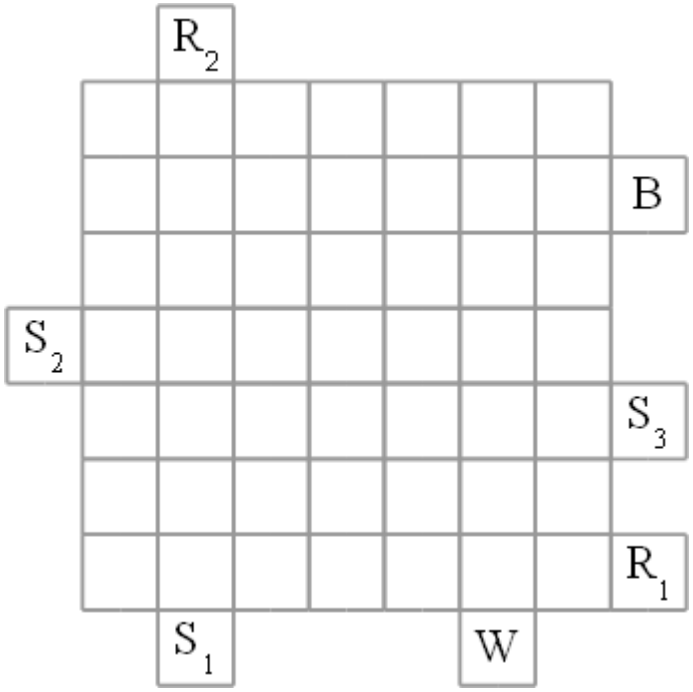


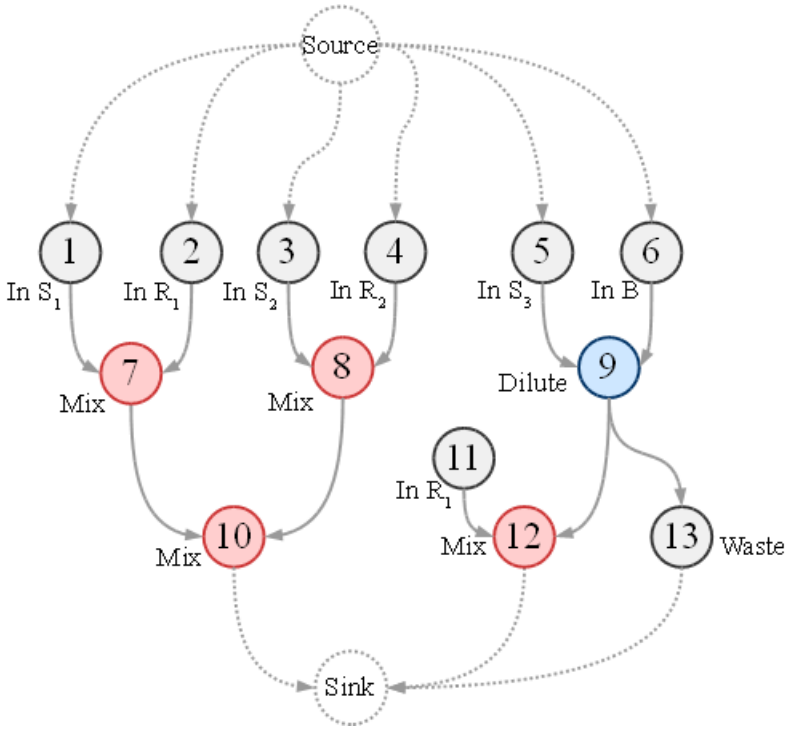
Routing-Based Synthesis





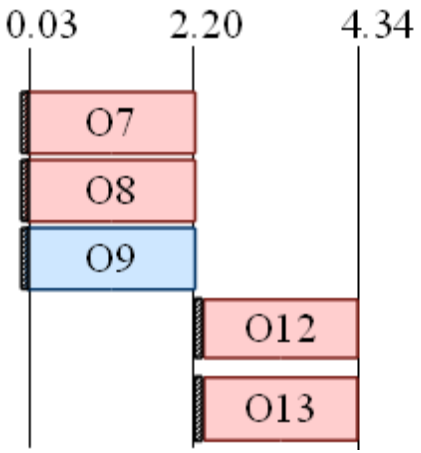
Routing- vs. Module-Based Synthesis



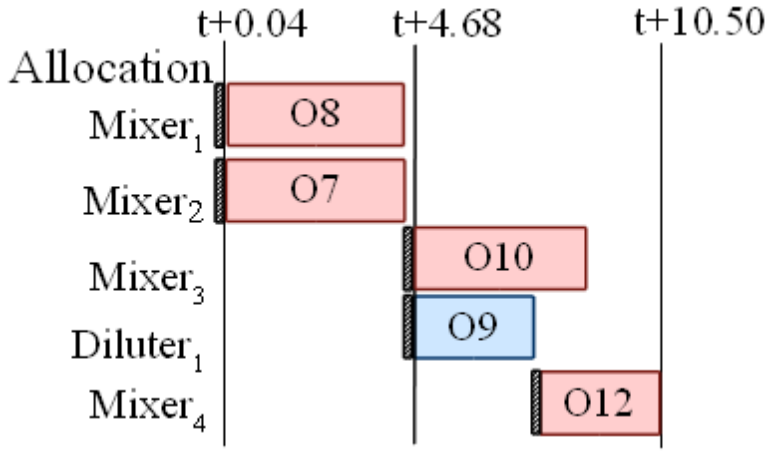


Routing- vs. Module-Based Synthesis

Routing-Based Synthesis



Module-Based Synthesis



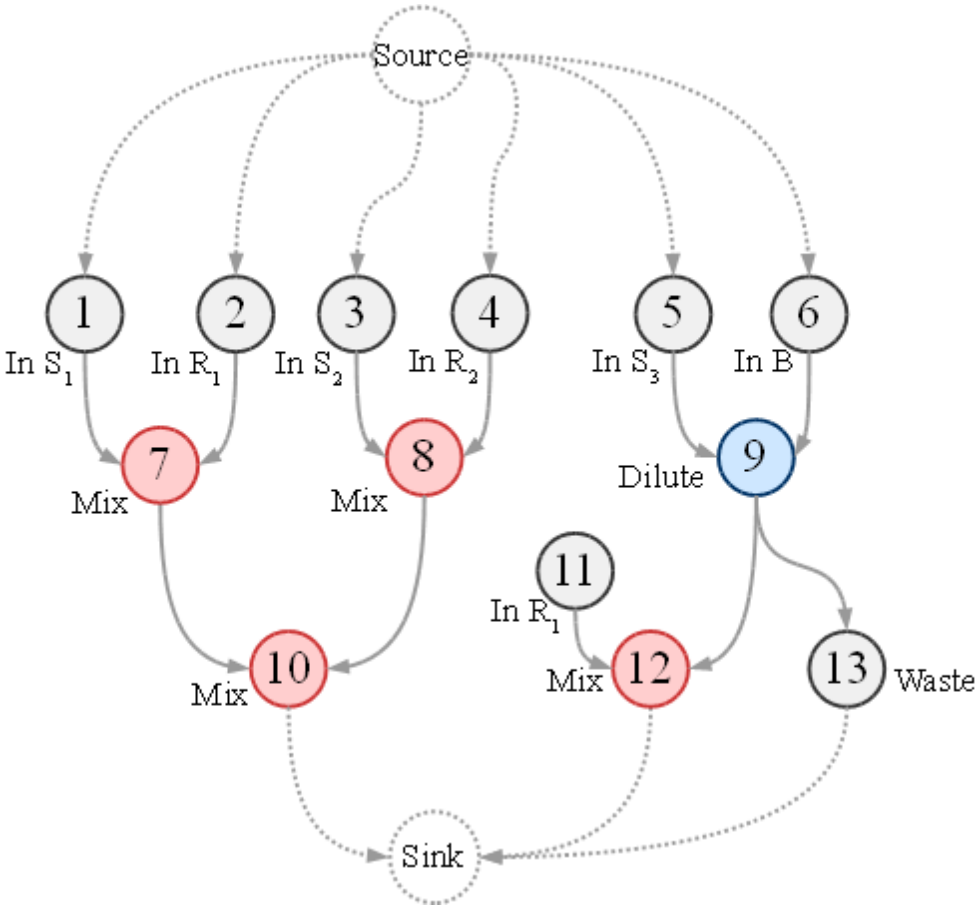
Problem Formulation

- **Given**
  - Application: graph
  - Biochip: array of electrodes
  - Library of non-reconfigurable devices

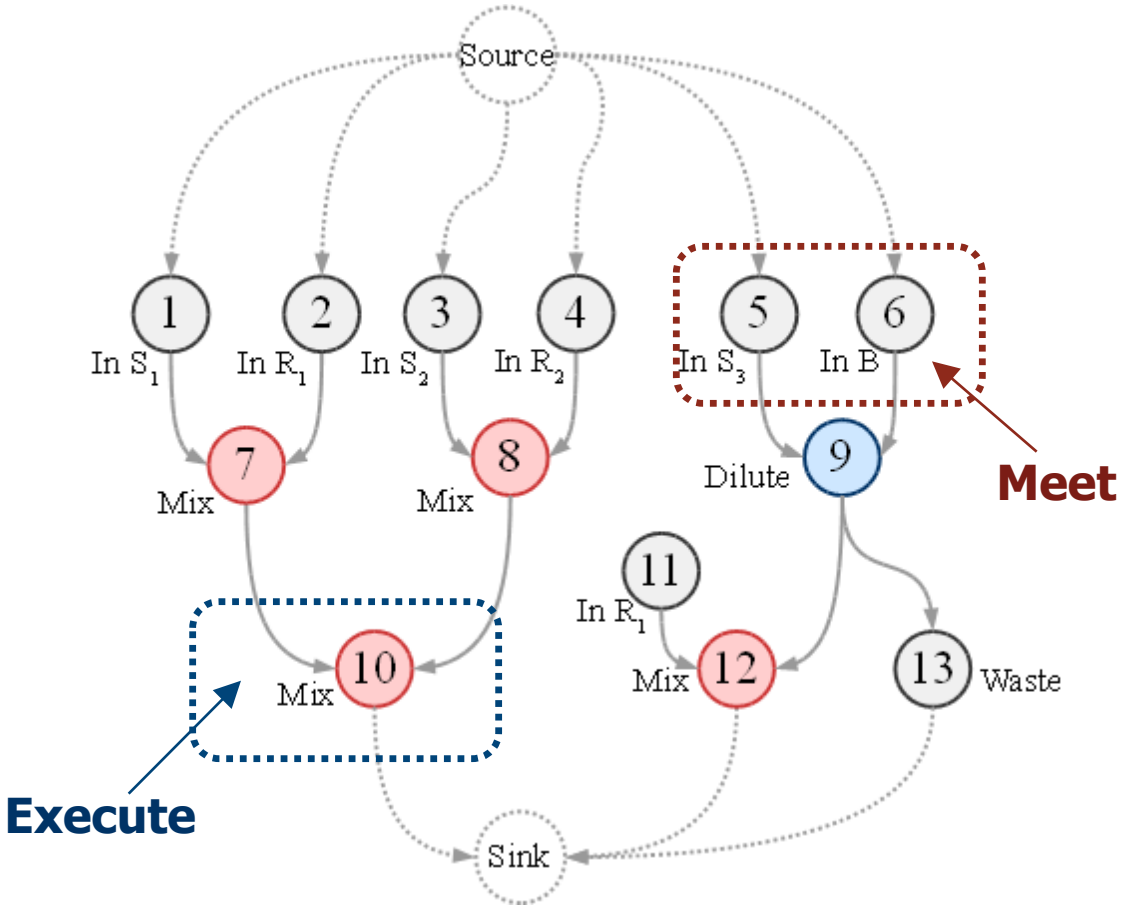
- **Determine**
  - **Droplet routes** for all reconfigurable operations
  - **Allocation** and **binding** of non-reconfigurable modules from a library
  - **Scheduling** of operations

- **Such that**
  - the application completion time is minimized

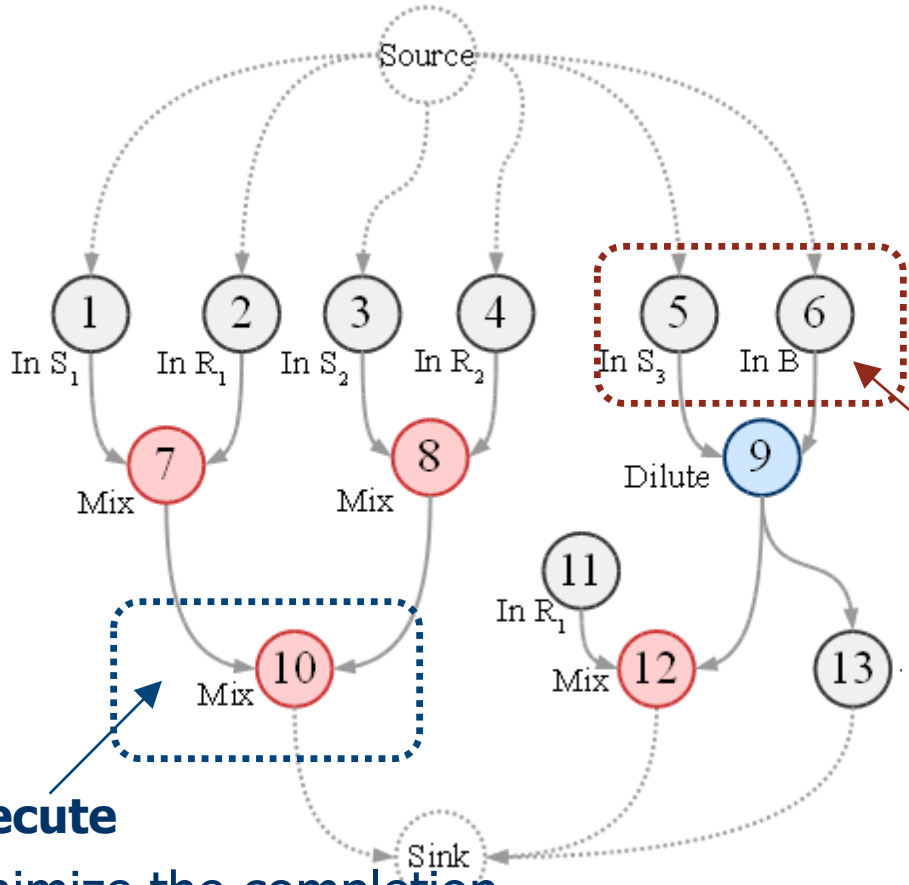
Proposed Solution



Proposed Solution



Proposed Solution



Meet

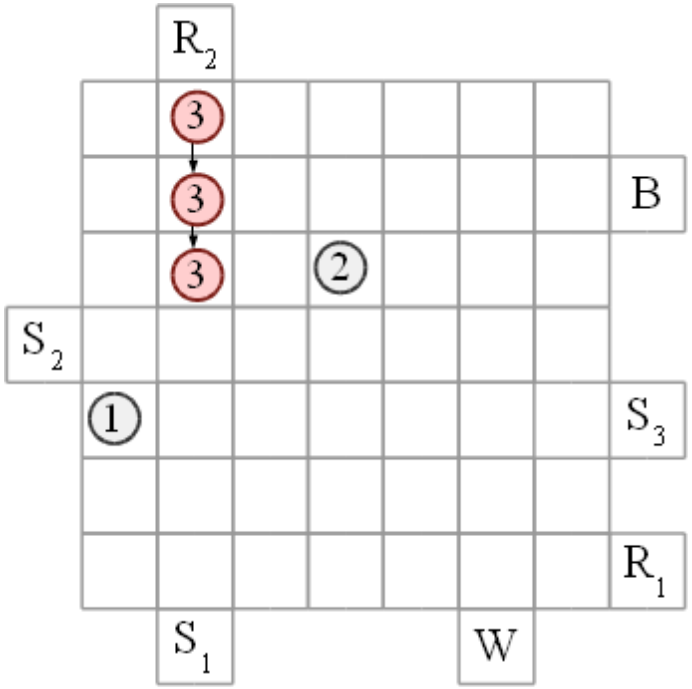
Minimize the time until the droplet(s) arrive at destination

Execute

Minimize the completion time for the operation

GRASP-Based Heuristic

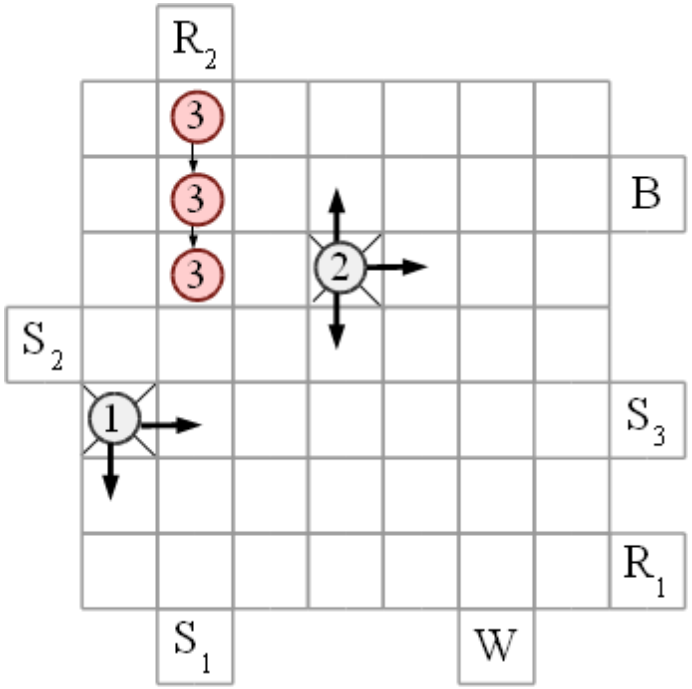
- **G**reedy **R**andomized **A**daptive **S**earch **P**rocedure



- For each droplet:
  - Determine possible moves
  - Evaluate possible moves
  - Make a list of best N possible moves
  - Perform a randomly chosen possible move

GRASP-Based Heuristic

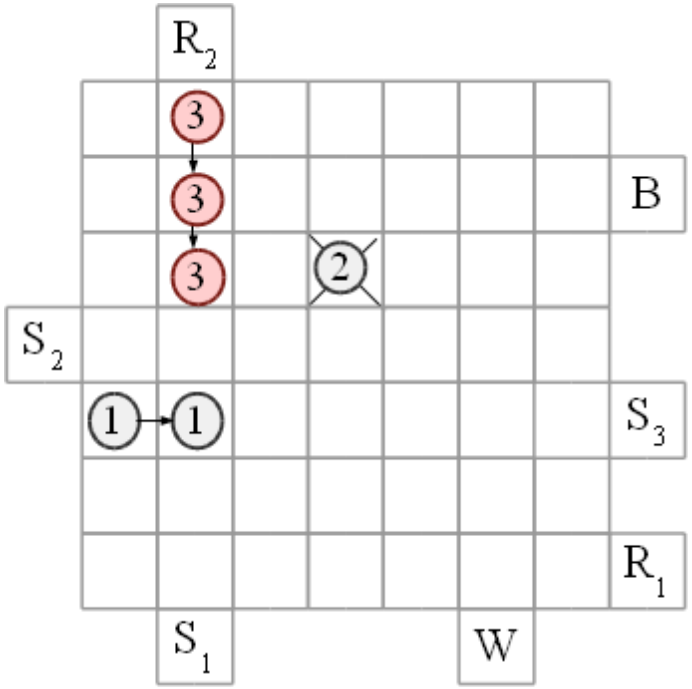
- Greedy Randomized Adaptive Search Procedure



- For each droplet:
  - Determine possible moves
  - Evaluate possible moves
  - Make a list of best N possible moves
  - Perform a randomly chosen possible move

GRASP-Based Heuristic

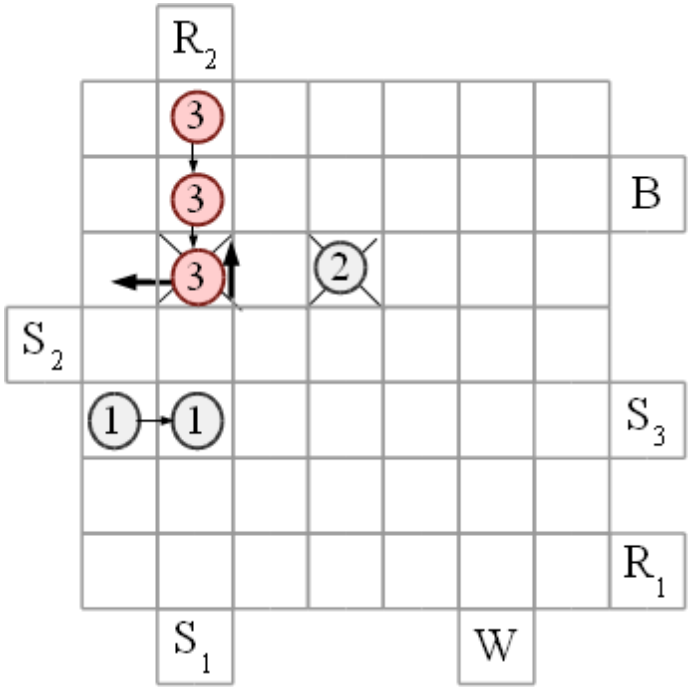
- Greedy Randomized Adaptive Search Procedure



- For each droplet:
  - Determine possible moves
  - Evaluate possible moves
  - Make a list of best N possible moves
  - Perform a randomly chosen possible move

GRASP-Based Heuristic

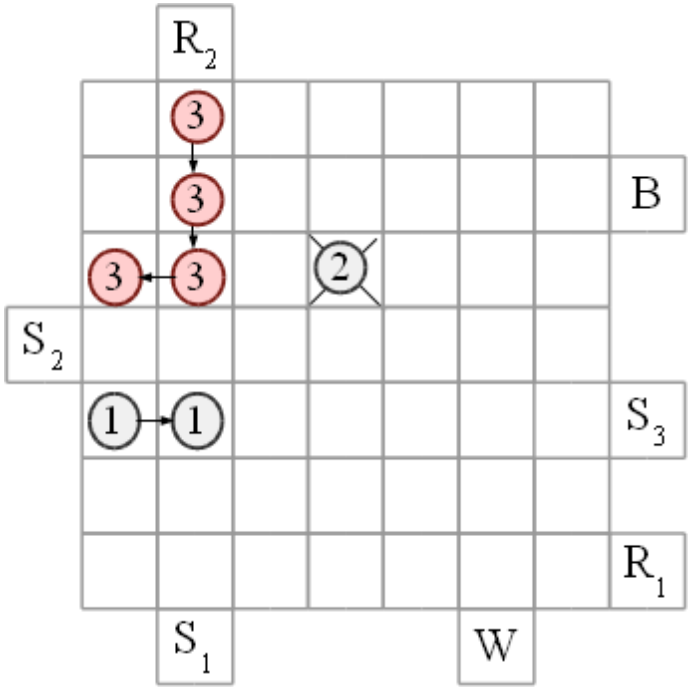
- Greedy Randomized Adaptive Search Procedure



- For each droplet:
  - Determine possible moves
  - Evaluate possible moves
  - Make a list of best N possible moves
  - Perform a randomly chosen possible move

GRASP-Based Heuristic

- Greedy Randomized Adaptive Search Procedure



- For each droplet:
  - Determine possible moves
  - Evaluate possible moves
  - Make a list of best N possible moves
  - Perform a randomly chosen possible move

Experimental Evaluation

Routing-Based Synthesis (**RBS**) vs. to Module-Based Synthesis (**MBS**)

Application	Area	Best	
		RBS	MBS
In-vitro (28 operations)	8×9	68.43	72.94
	8×8	68.87	82.12
	7×8	69.12	87.33
Proteins (103 operations)	11×11	113.63	184.06
	11×10	114.33	185.91
	10×10	115.65	208.90

Conclusions

- **Module**-based vs. **routing**-based
  - Module-based needs an extra routing step between the modules; Routing-based performs unified synthesis and routing
  - Module-based wastes space: only one module-cell is used; Routing-based exploits better the application parallelism
  - Module-based can contain the contamination to a fixed area;
  - We have extended routing-based to address contamination

Droplet Routing

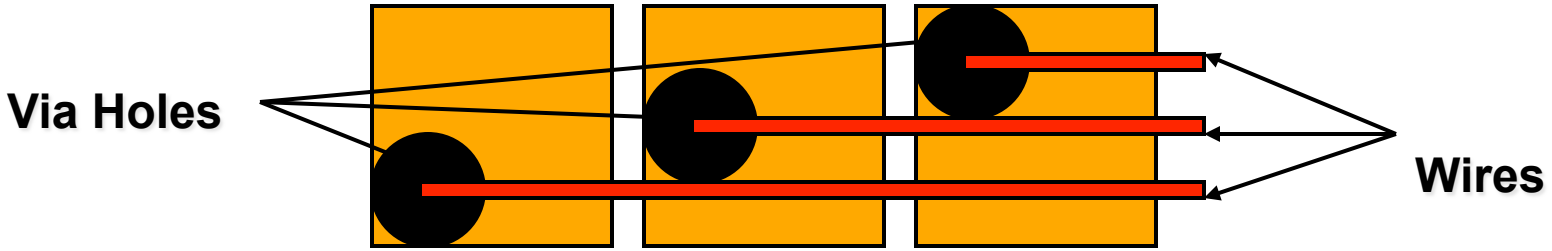
- A key physical design problem for digital microfluidic biochips
- Given the results from architectural-level synthesis and module placement:
  - Determine droplet pathways using the available cells in the microfluidic array; these routes are used to transport droplets between modules, or between modules and fluidic I/O ports (i.e., boundary on-chip reservoirs)
- To find droplet routes with minimum lengths
  - Analogous to the minimization of the total wirelength in VLSI routing
- Need to satisfy critical constraints
  - A set of fluidic constraints
  - Timing constraints: (delay for each droplet route does not exceed some maximum value, e.g., 10% of a time-slot used in scheduling)

Challenge: Design of Pin-Constrained Biochips

Direct Addressing

- Each electrode connected to an independent pin
- For large arrays (e.g., > 100 x 100 electrodes)
  - Too many control pins ⇒ high fabrication cost
  - Wiring plan not available

PCB design: 250 um via hole, 500 um x 500 um electrode

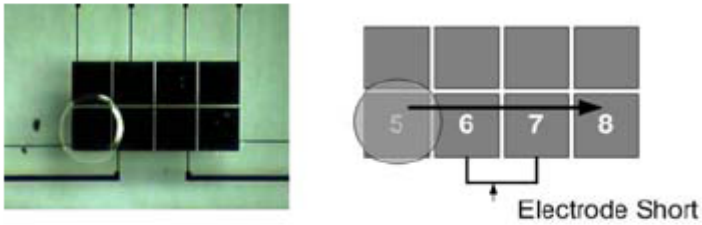


Nevertheless, we need high-throughput *and* low cost:

DNA sequencing (10^6 base pairs), Protein crystallization (10^3 candidate conditions)

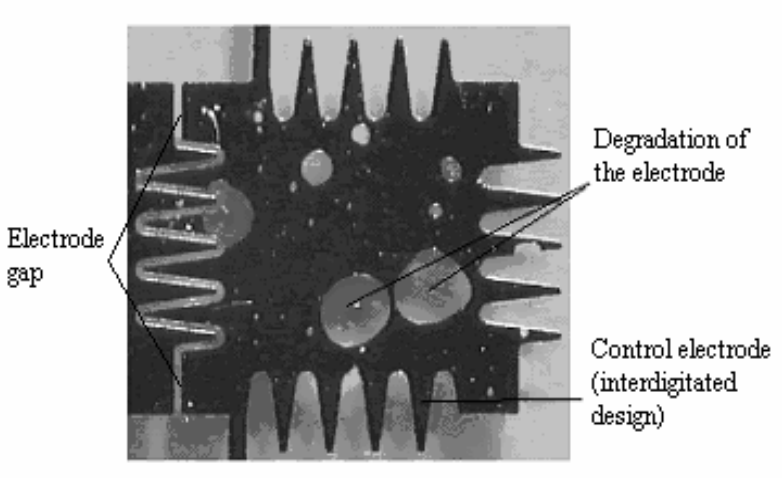
Disposable, marketability, $1 per chip

Challenge: Fault-tolerant design

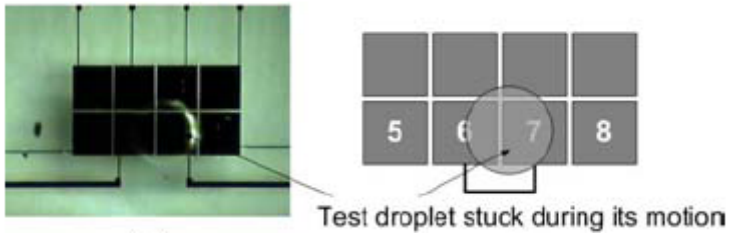


Electrode short

Imperfect splitting



Electrode degradation

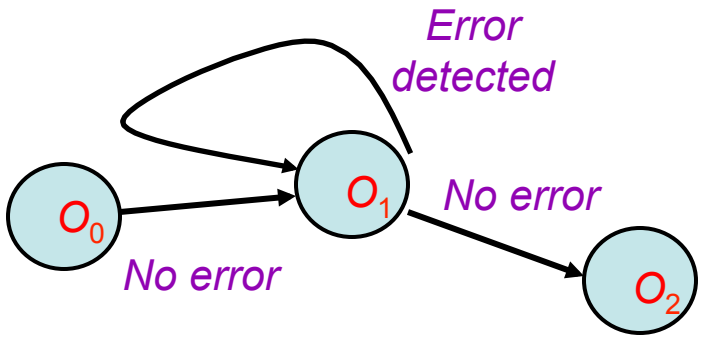


Hindered transportation

Motivation for Error Recovery

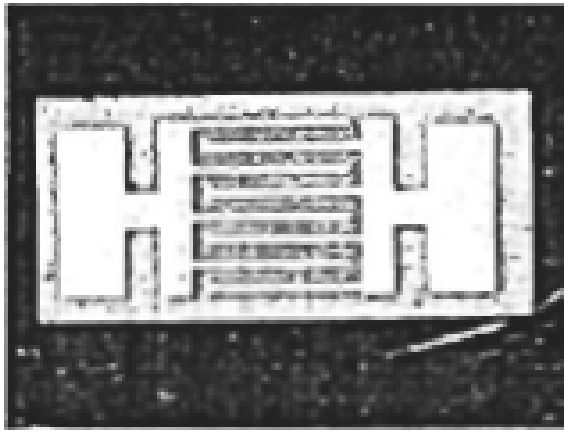
- Verify correctness of fluidic operations in bioassay
  - Monitor bioassay status to find errors
  - Parameters for monitoring: volume of product droplet, sample concentration, *others*?
- Correct errors as soon as possible
  - Re-execute only the erroneous part of bioassay
- Drawback of current synthesis tools
  - Only provide a "data path", no control or feedback mechanism
  - Monitor bioassay result at the end and re-execute the entire assay to correct errors

Need control-path design for error detection and recovery

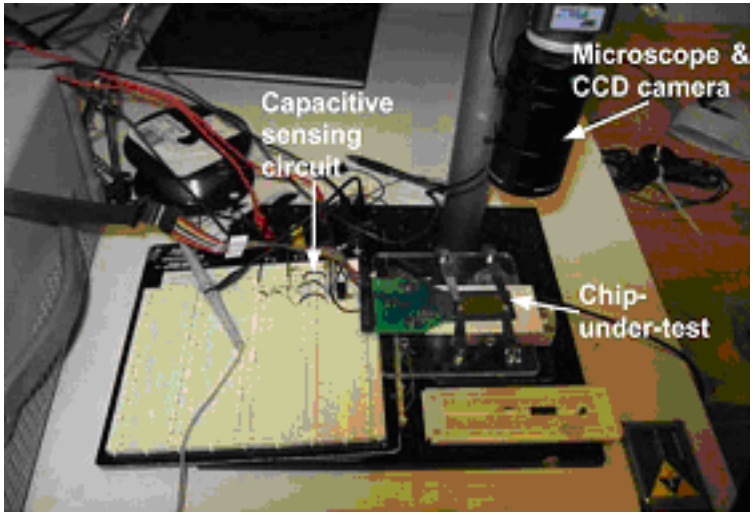


Droplet Detection Mechanisms

- Capacitive-sensing circuit for volumetric test
- Optical detection for concentration test



Thin-film MSM detector (S.-W. Seo, PhD Thesis 2003)



Capacitive-sensing circuit
(M. G. Pollack, PhD Thesis 2001)

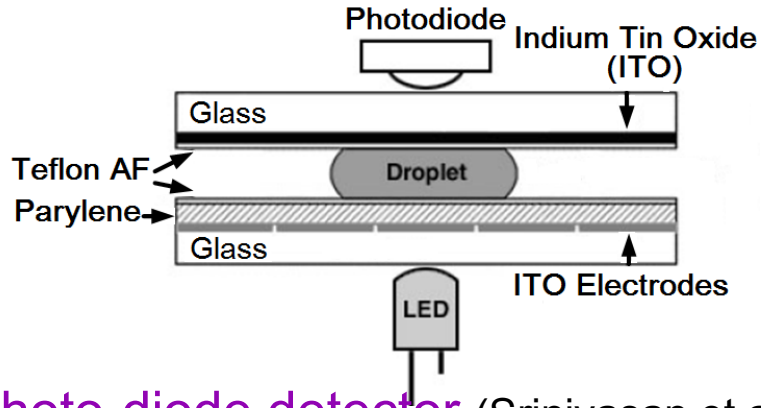
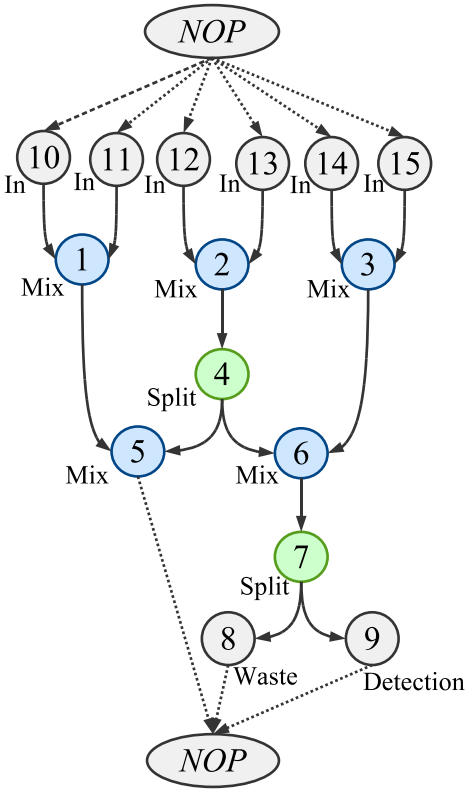
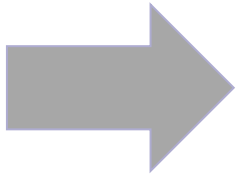
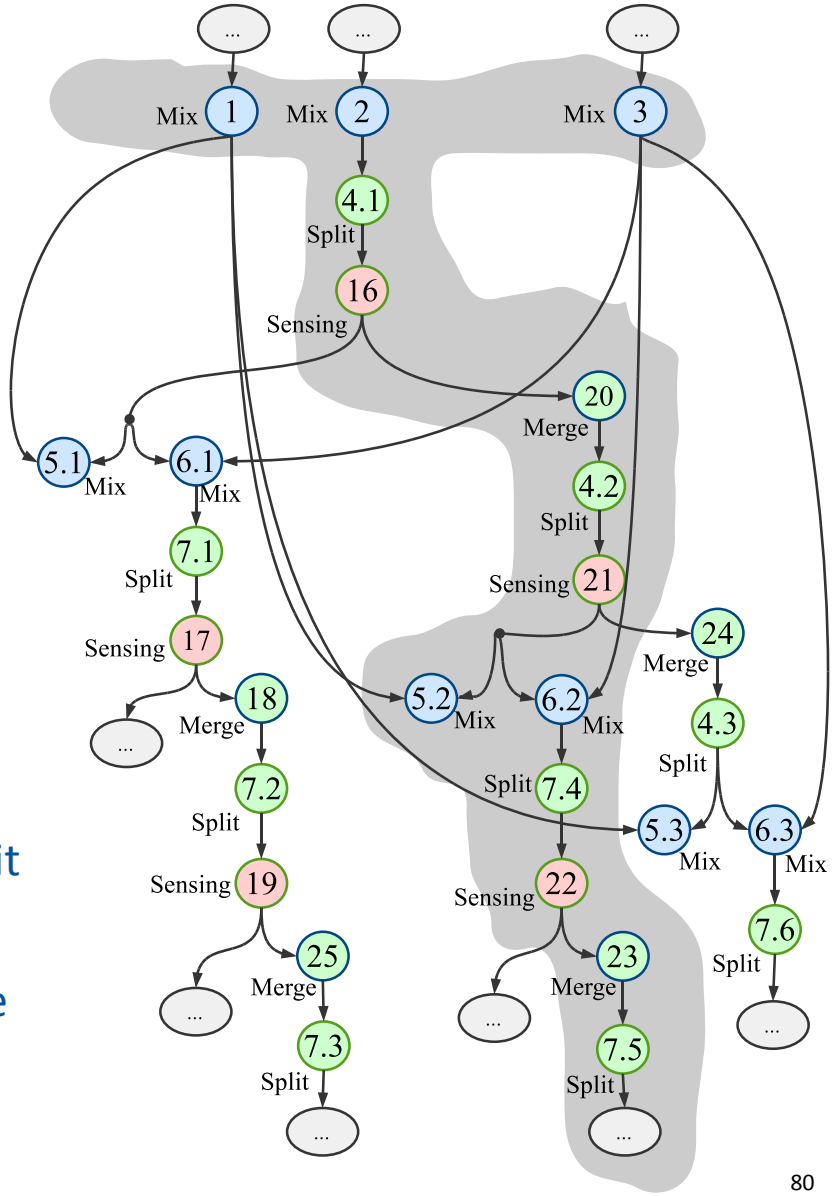


Photo-diode detector (Srinivasan et al., MicroTAS'03)

Fault-tolerant graph: captures fault scenarios due to split operations



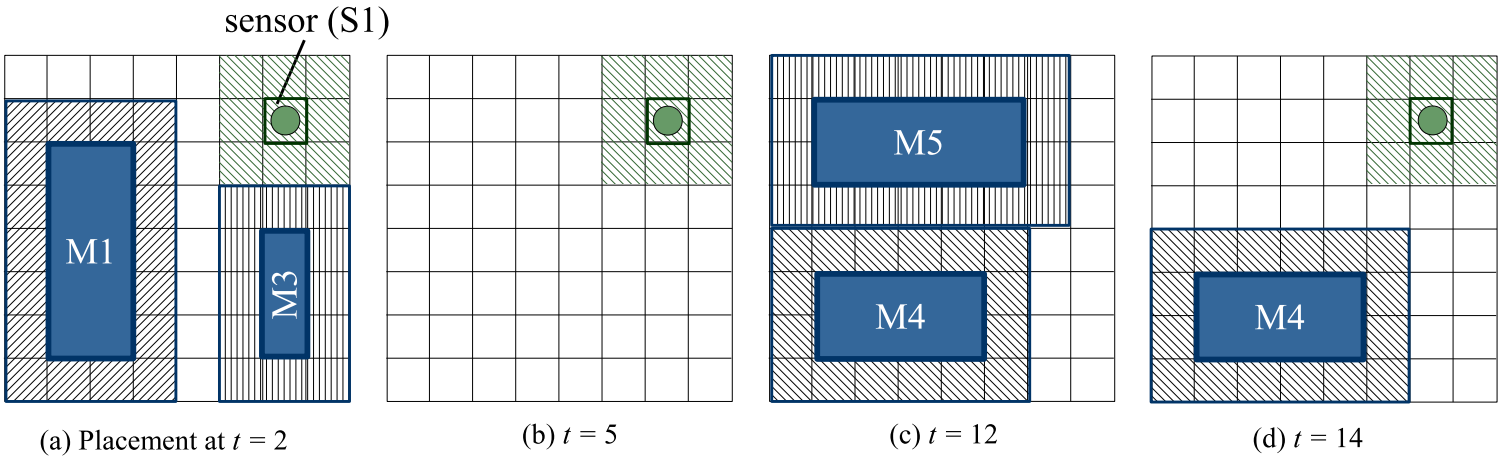




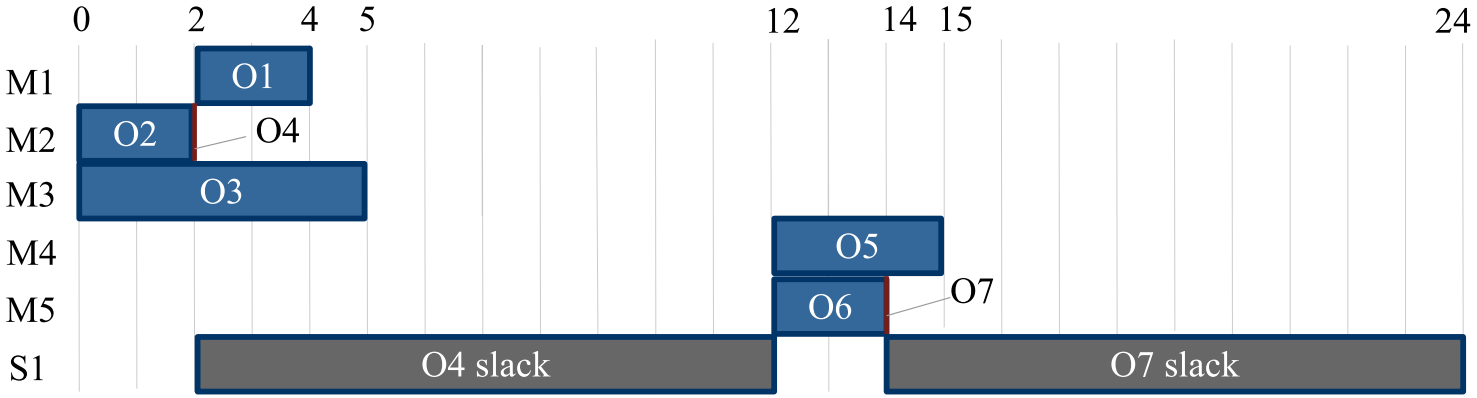
- A sensing operation is introduced after each split
- If the split was OK, the graph continues
- If the split was NOT OK, we retry: insert a merge operation followed by another split

Assumption: at most two consecutive errors

Straightforward scheduling



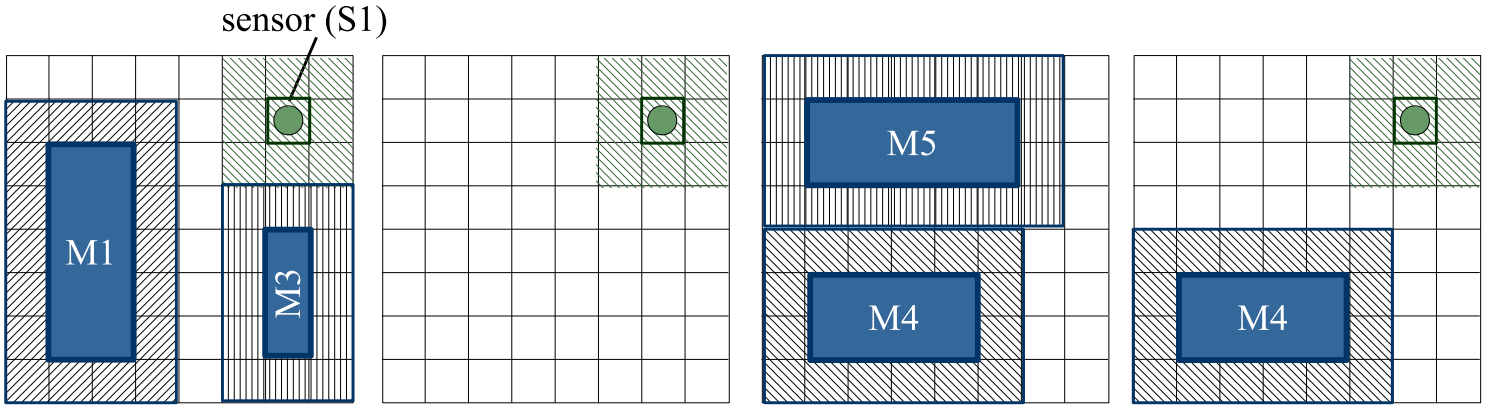
(a) Placement at $t = 2$ (b) $t = 5$ (c) $t = 12$ (d) $t = 14$



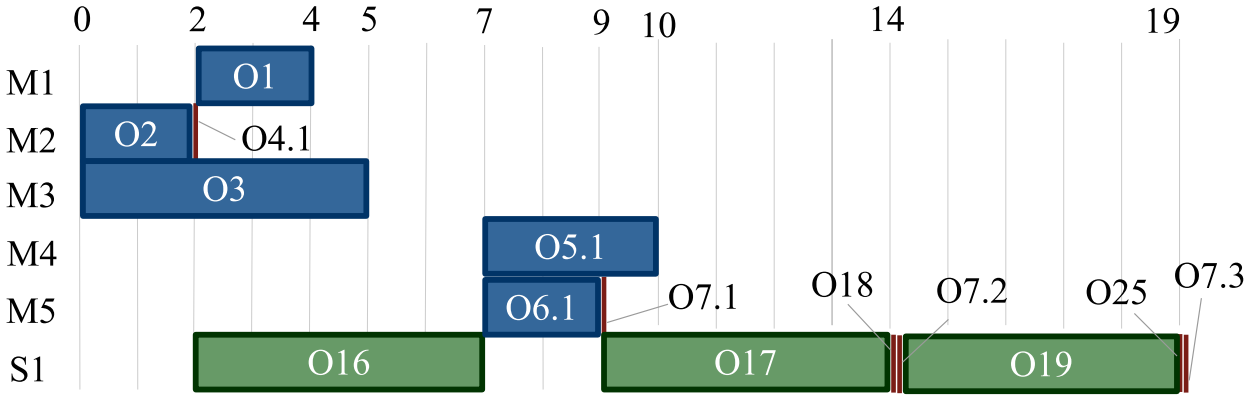
(e) Schedule

Adding worst-case slack after each split to allow for recovery

Scheduling the fault-tolerant graph: backup schedules for fault scenarios



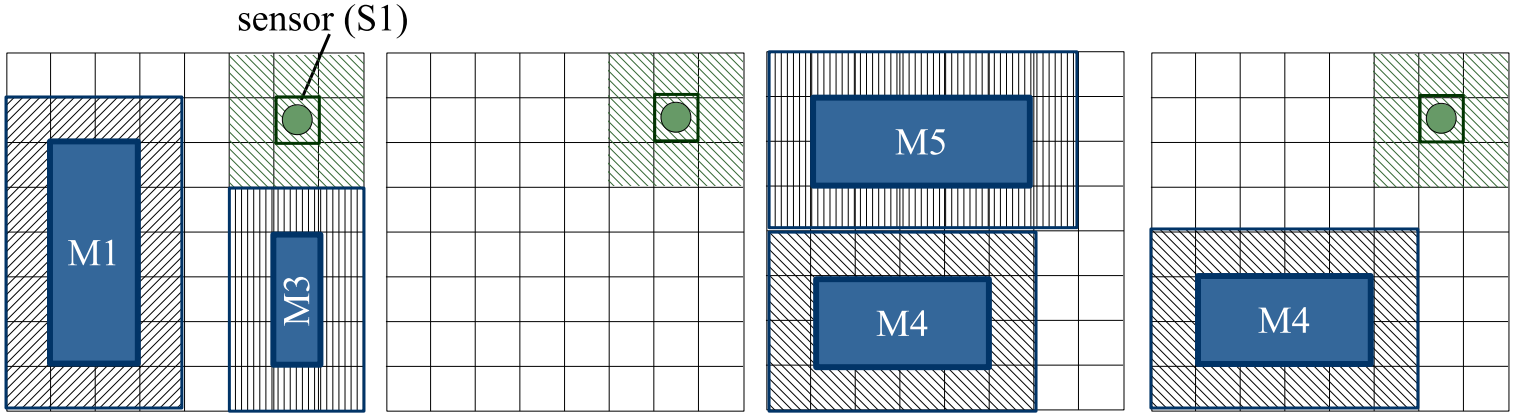
(a) Placement at $t = 2$ (b) $t = 5$ (c) $t = 7$ (d) $t = 9$



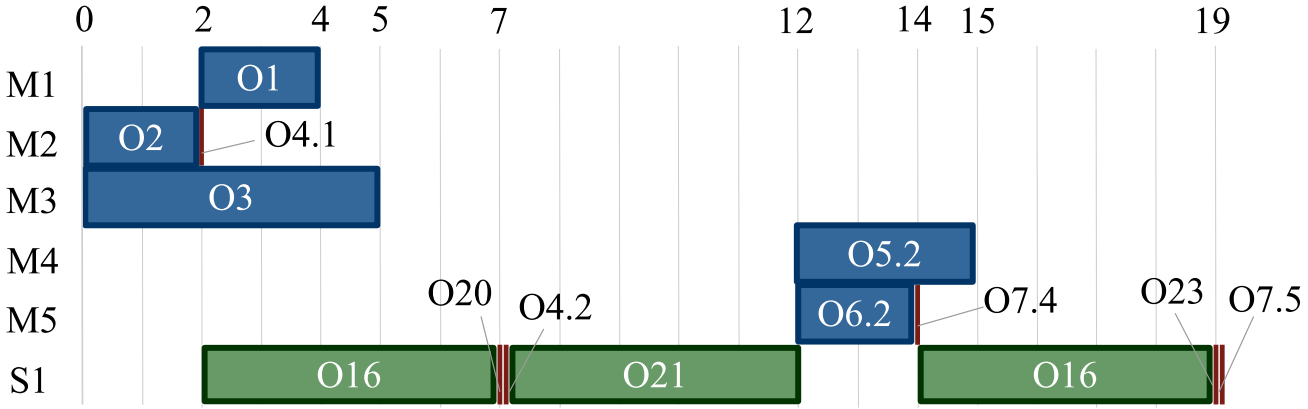
(e) Schedule

Fault-tolerant schedule for two faults in O7

Scheduling the fault-tolerant graph: backup schedules for fault scenarios



(a) Placement at $t = 2$ (b) $t = 5$ (c) $t = 12$ (d) $t = 14$

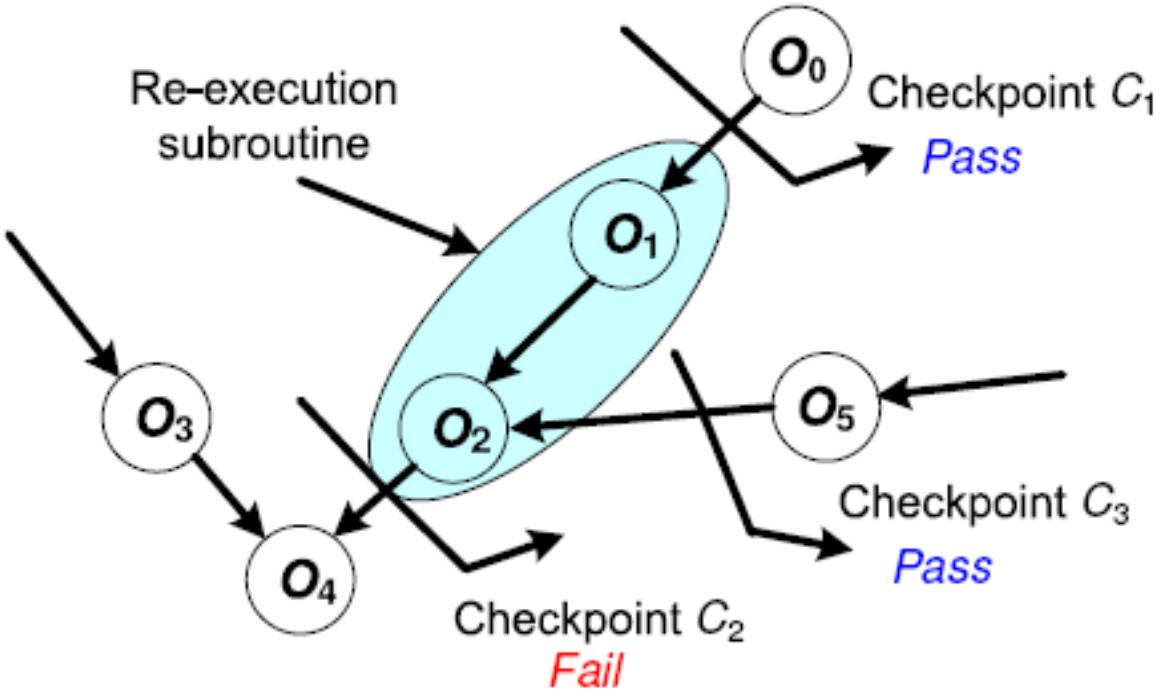


(e) Schedule

Fault-tolerant schedule for faults in O4 and O7

Another approach: Control-Path Design

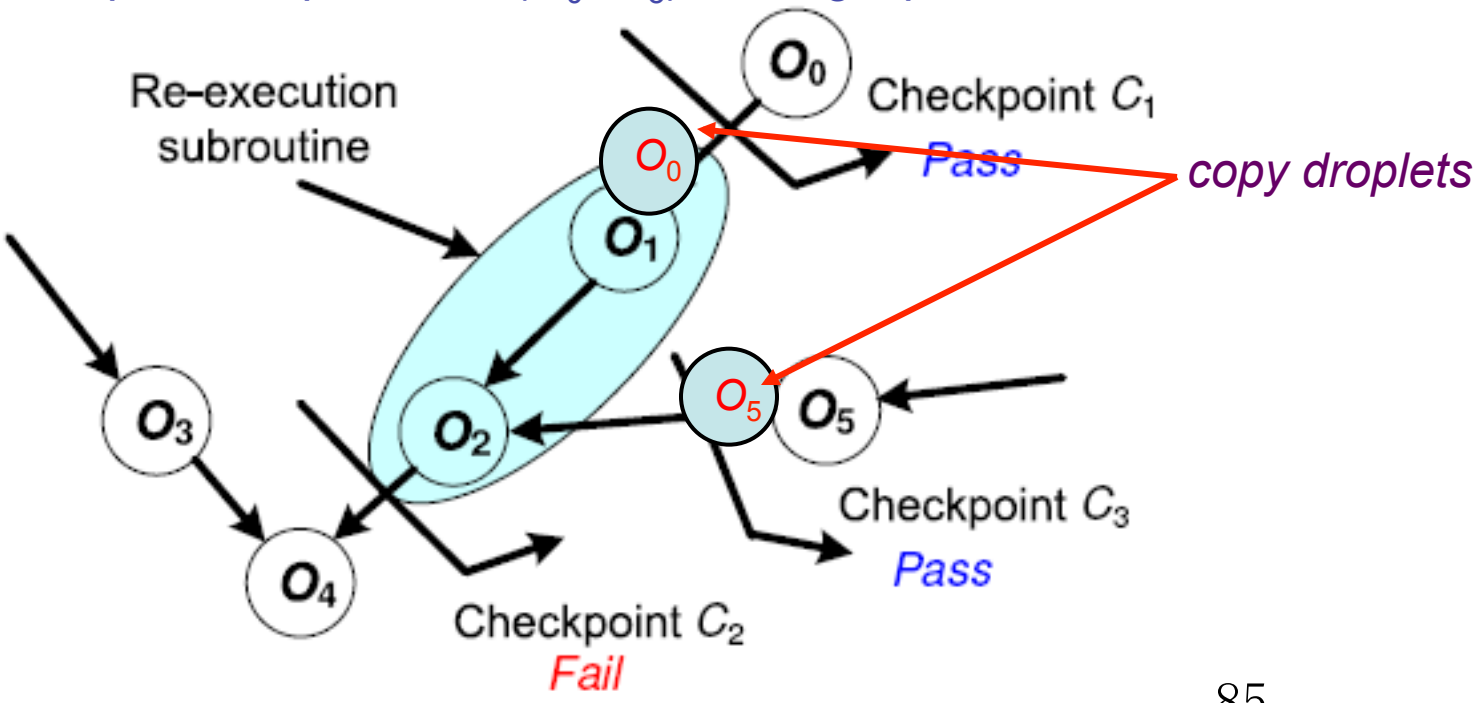
- Add *checkpoints* to monitor outcomes of fluidic operations
  - Checkpoint: storage of the intermediate product droplet
  - Add checkpoints based on error-propagation estimates
- Assign each checkpoint a *re-execution subroutine*
  - Subroutine: fluidic operations between checkpoints
  - Correct the detected error by re-executing the subroutine



- Status at checkpoints
  - C_1: Pass
  - C_2: Fail
  - C_3: Pass
- Re-execution subroutine for C_2
  - Operations O_1 and O_2

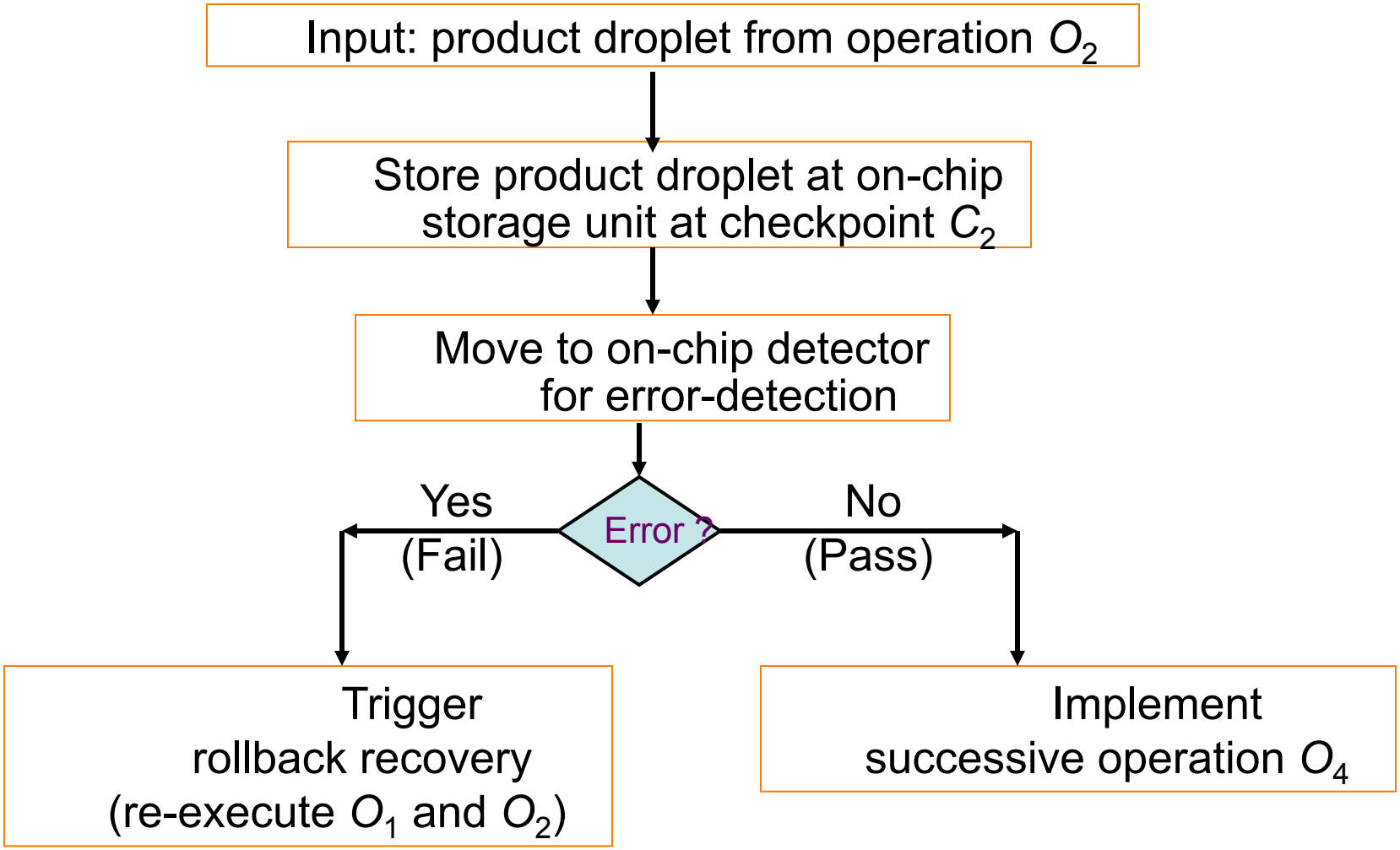
Control-Path Design

- Error detection at the checkpoint
  - Performed for intermediate product droplet at the checkpoint
  - Concentration test (using photo-detector)
  - Volumetric test (using capacitive-sensing circuit)
- Droplet preparation for re-execution subroutine
  - Copy droplets are consumed during re-execution of a subroutine
  - Output droplets of operations (O_0, O_5) feeding inputs of subroutine



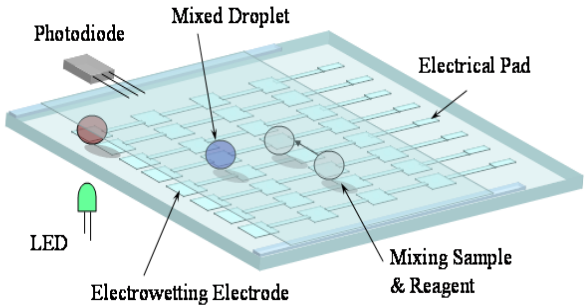
Control-Path Design

- Implementation flow for error recovery at checkpoint C_2



86

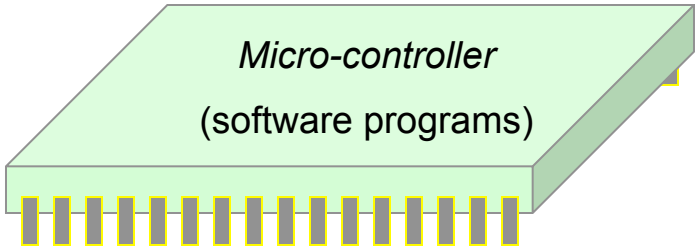
Implementation for Rollback Recovery at Checkpoint C_2



Microfluidic Array

Bioassay Instructions

Bioassay Results



Time: clock cycle 28
Instruction: start 0087 (C_2)

Time: clock cycle 33
Result: error detected at C_2

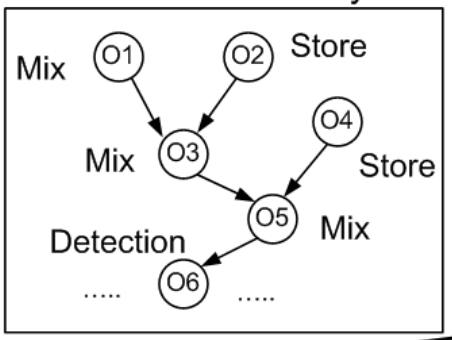
Time: clock cycle 33
Instruction:
(1) stop 0090 (O_3)
(2) stop time counter
(3) start 0085(O_1) to 0087(C_2)

Time: clock cycle 33
Result: no error at C_2

Time: clock cycle 33
Instruction:
(1) resume following bioassay
(2) resume time counter

System-Level Design of Microfluidic Biochips

Input: Sequencing graph of bioassay



Digital microfluidic module library

Mixing components	Area	Time
2x2-array mixer	4 cells	10 s
2x3-array mixer	6 cells	6 s
2x4-array mixer	8 cells	3 s
1x4-array mixer	4 cells	5s
Detectors		
LED+Photodiode	1 cell	30 s

Design specifications

Maximum array area A_{max}: 20x20 array

Maximum number of optical detectors: 4

Number of reservoirs: 3

Maximum bioassay completion time T_{max}: 50 seconds

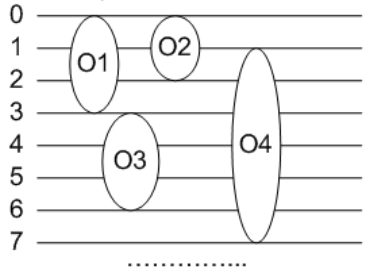
Unified Synthesis of Digital Microfluidic Biochip

Output:

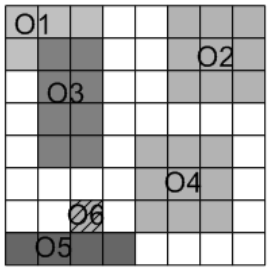
Resource binding

Operation	Resource
O1	2x3-array mixer
O2	Storage unit (1 cell)
O3	2x4-array mixer
O4	Storage unit (1 cell)
O5	1x4-array mixer
O6	LED+Photodiode

Schedule

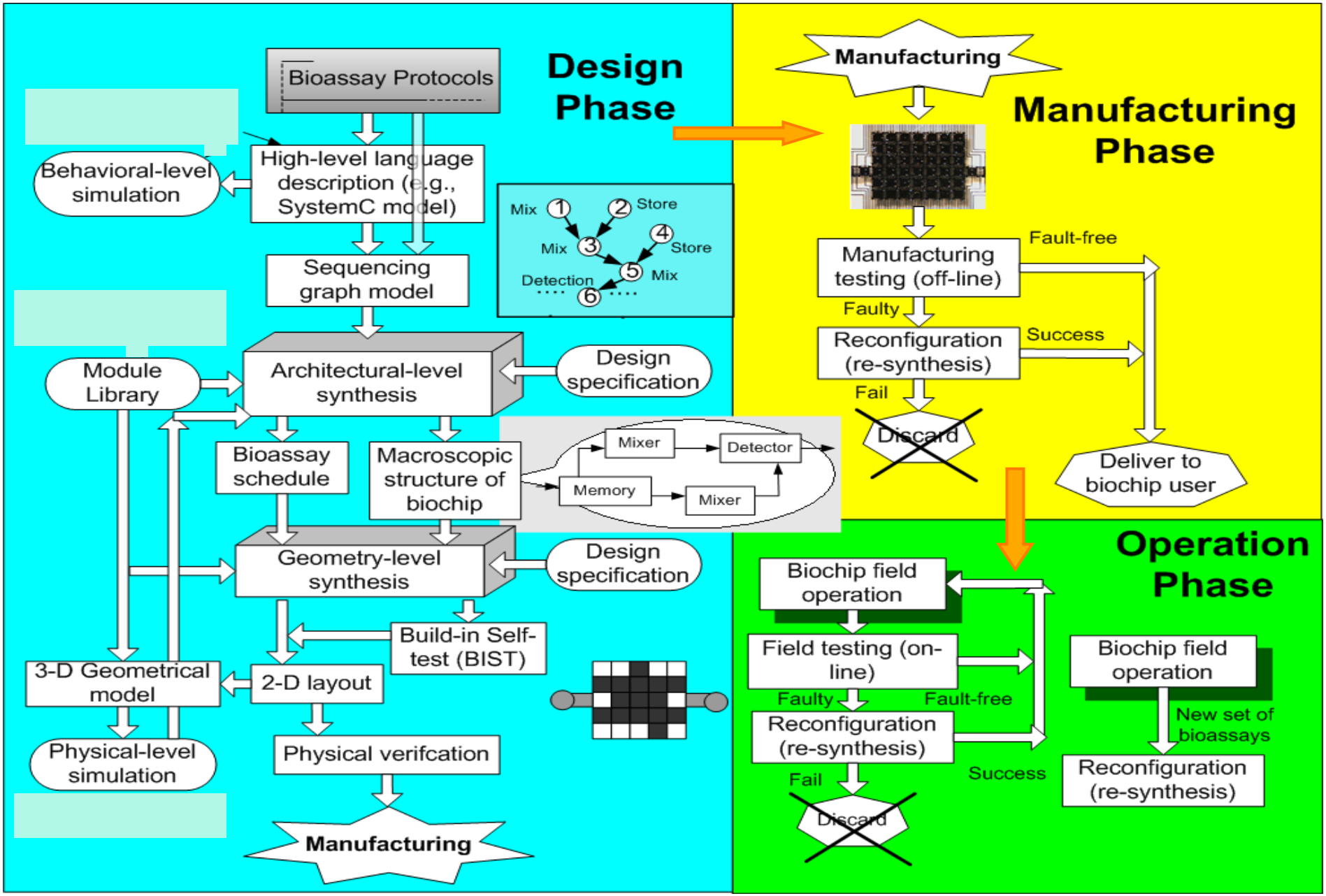


Placement

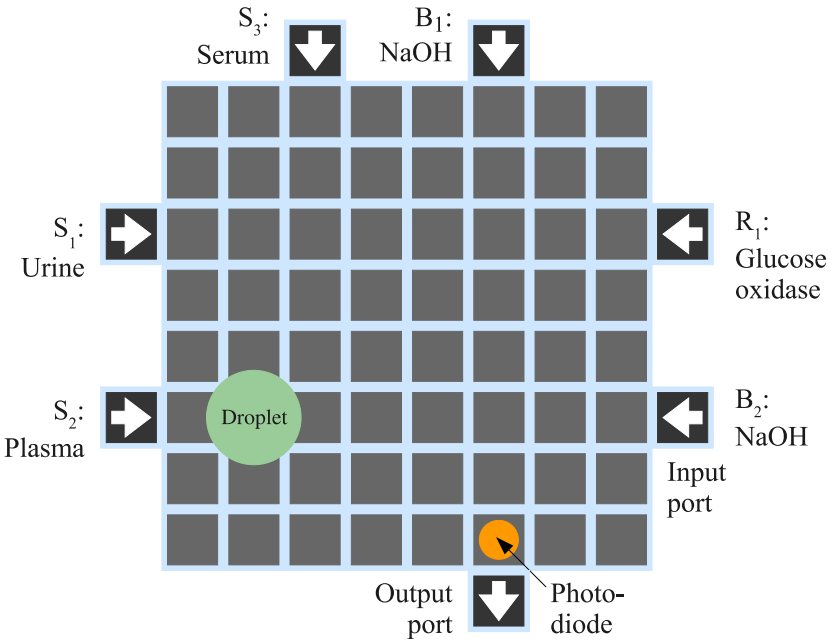


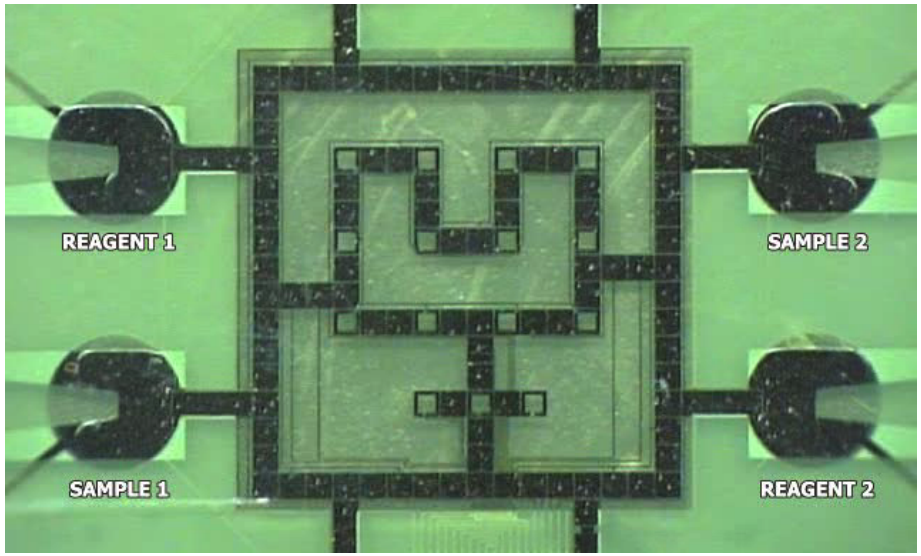
Biochip design results: **Array area:** 8x8 array **Bioassay completion time:** 25 seconds

Biochip Design Automation Overview



Challenge: Architecture-specific biochip design





Biochip from Duke University