



# Automata Processing

## AP Programming Overview

Ke Wang<sup>1</sup> (Research Scientist of CAP)

Matt Tanner<sup>2</sup>

Kevin Skadron<sup>1</sup> (Center Director of CAP)

Mircea Stan<sup>1</sup> (Assoc. Director of CAP)

<sup>1</sup> University of Virginia

<sup>2</sup> Micron Technology Inc.

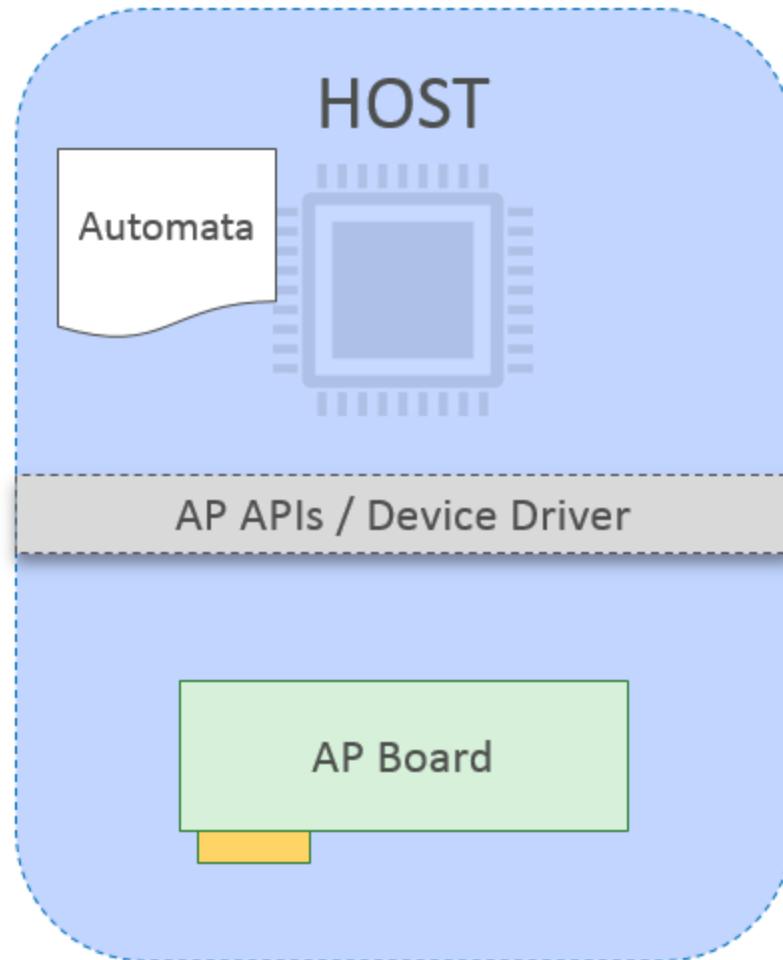
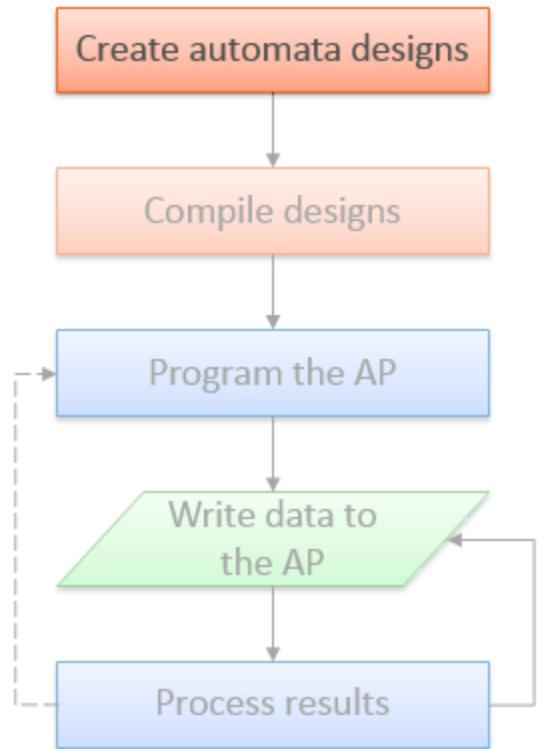
# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

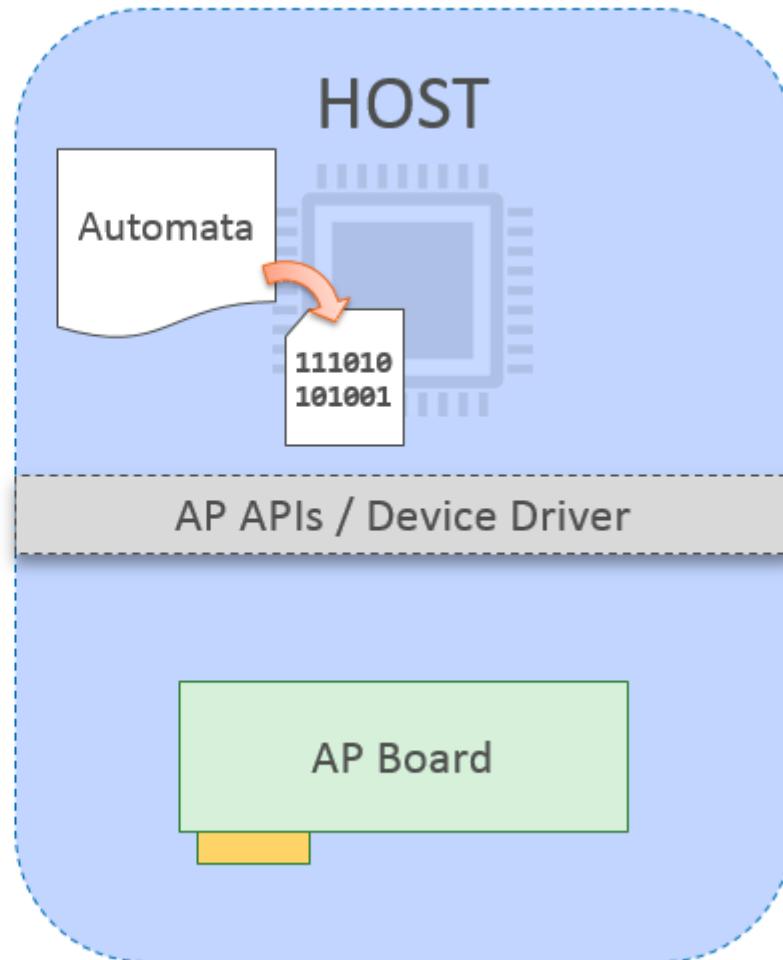
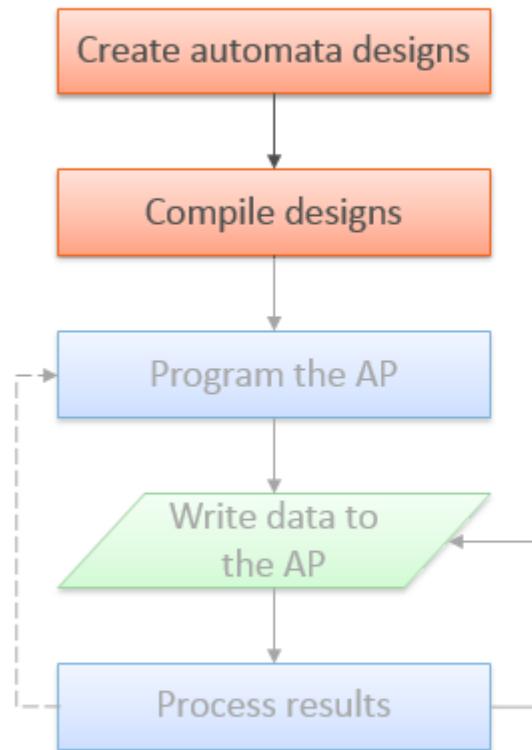
# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

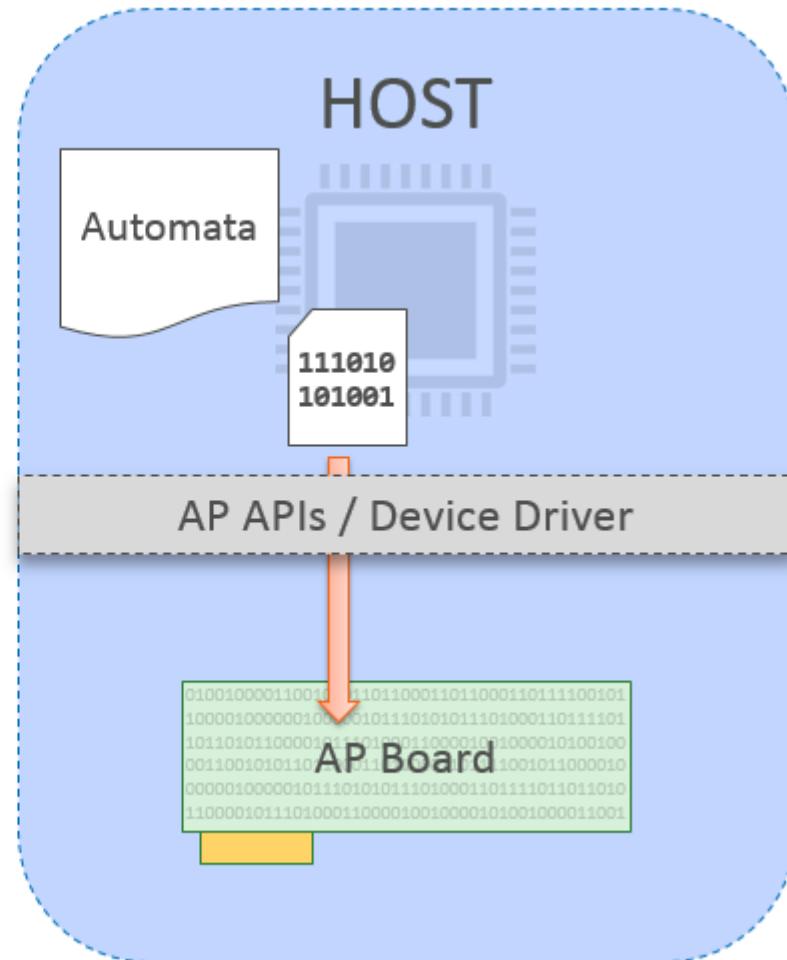
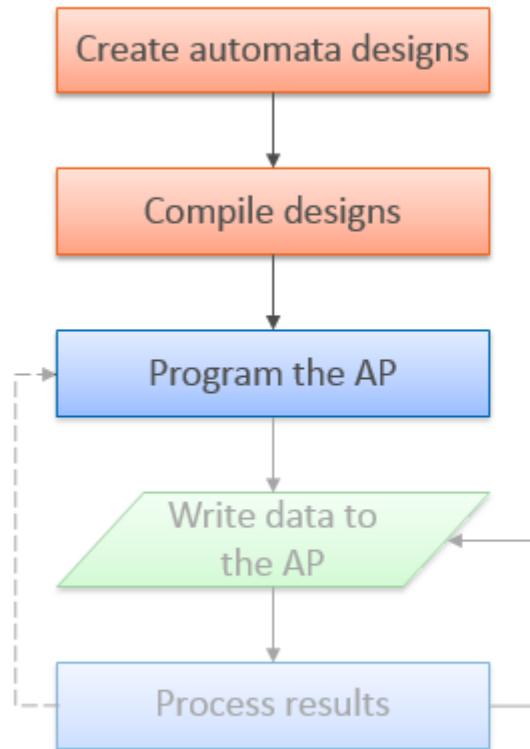
# System Overview



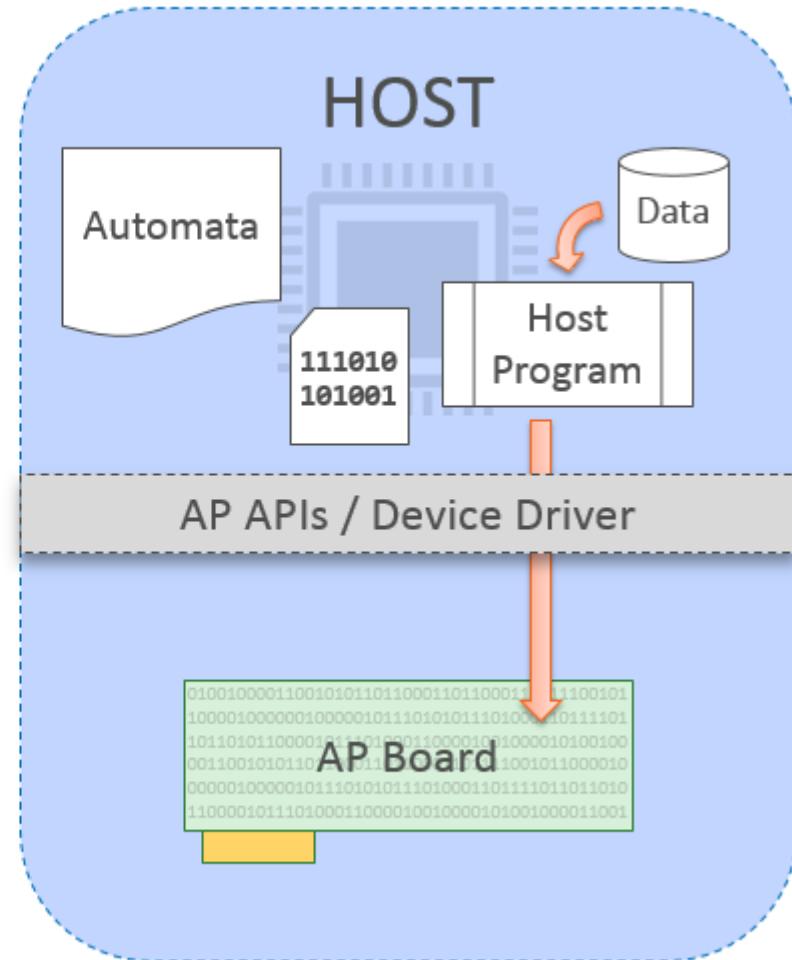
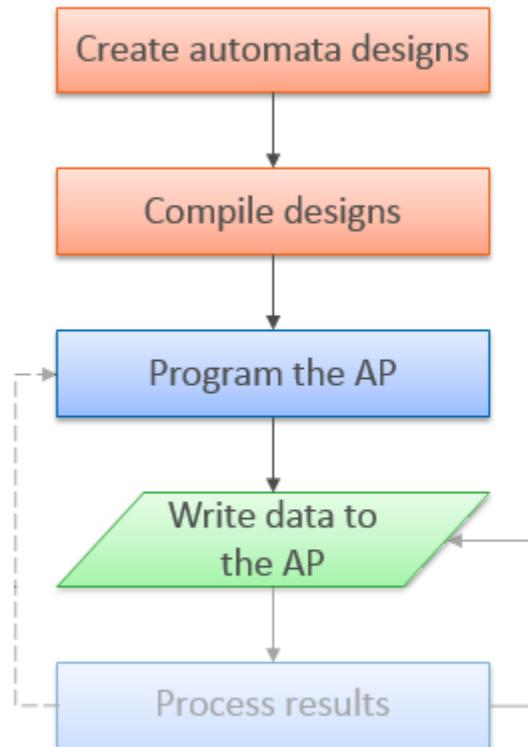
# System Overview



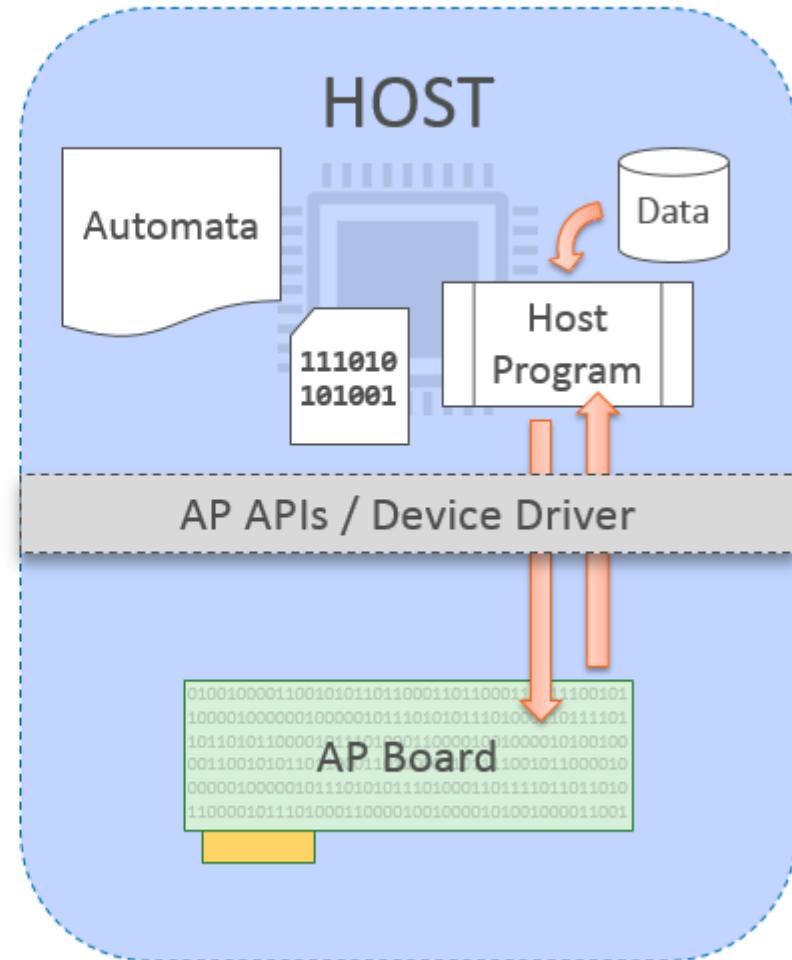
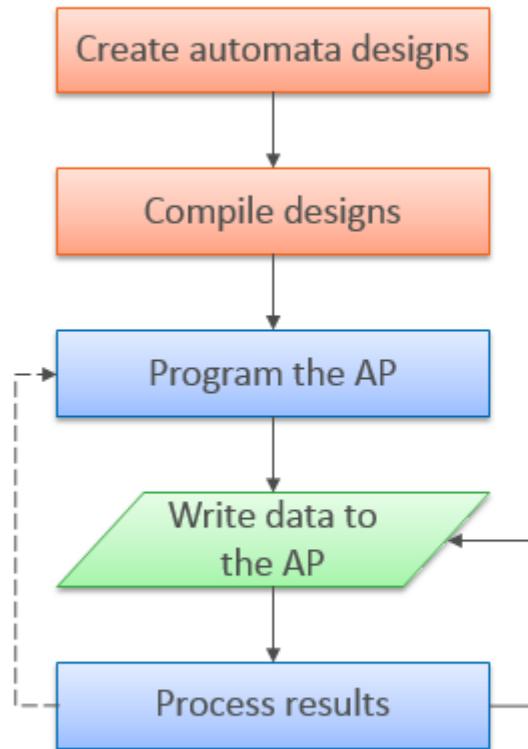
# System Overview



# System Overview

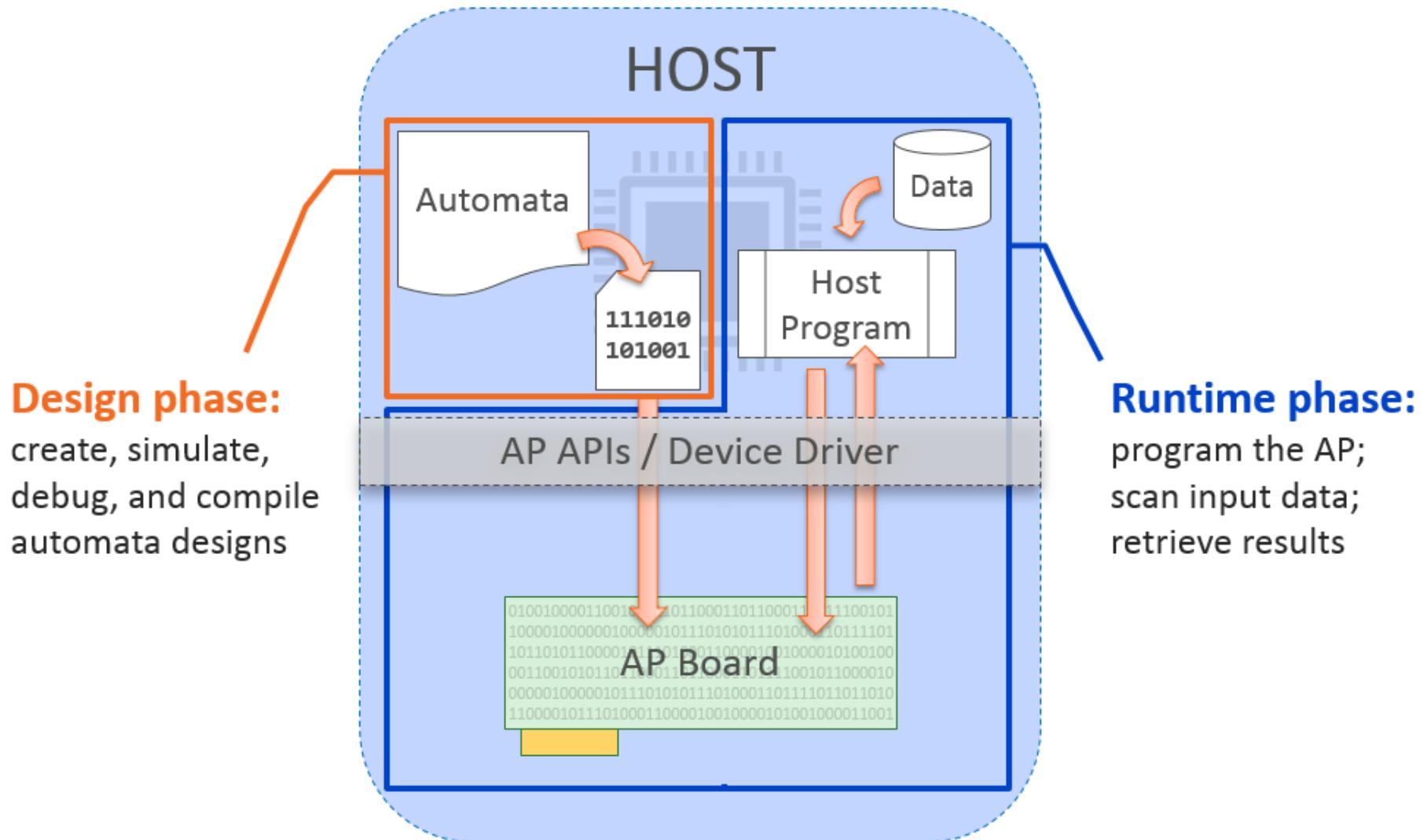


# System Overview



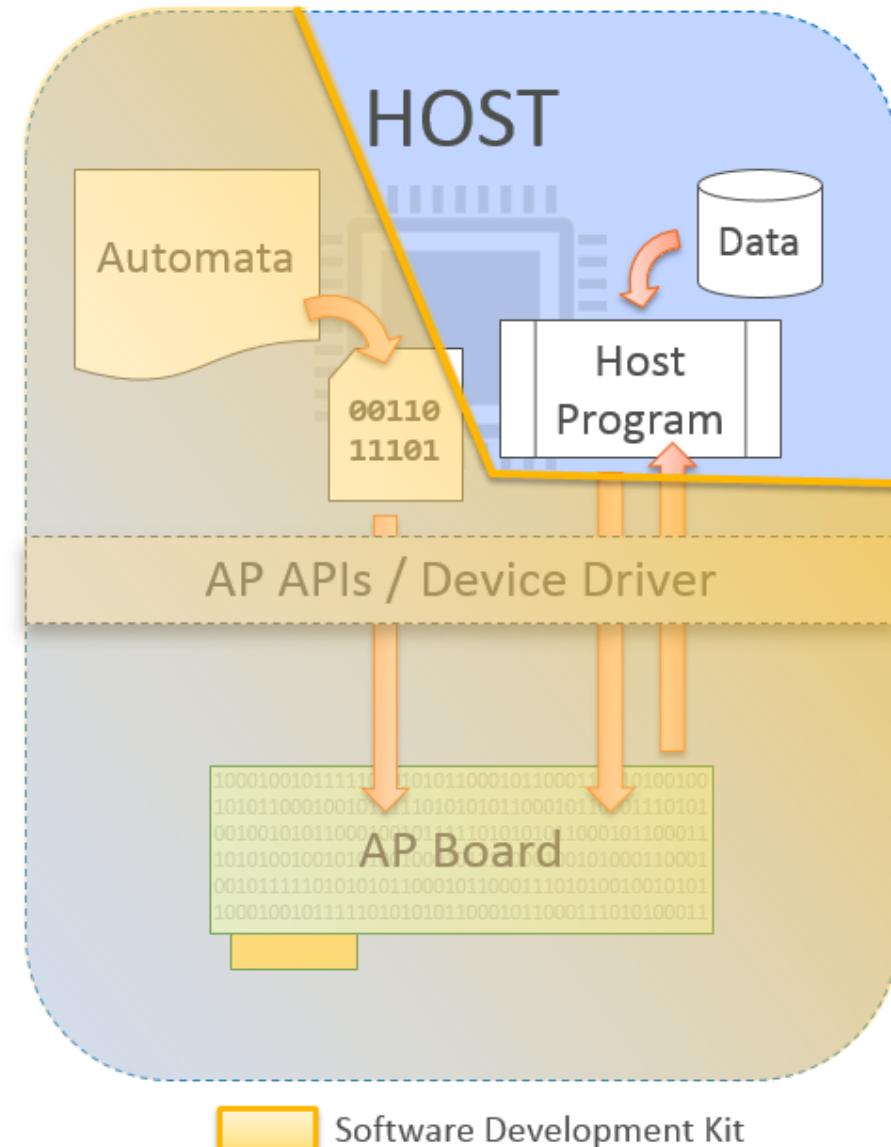
# System Overview

Two distinct phases of AP usage: **design phase** and **runtime phase**



# Software Development Kit (SDK)

- ▶ APIs
  - Design API
  - Compiler API
  - Runtime API
  - Application-specific APIs
- ▶ Workbench
- ▶ Command-line tools
- ▶ Device driver



# Sample Program Development

APIs

```
1 import micronap.sdk as ap
2
3 a = ap.Anml()
4 an = a.CreateAutomataNetwork(anmlId='hello_automata')
5
6 h = an.AddSTE('h', startType=ap.AnmlDefs.ALL_INPUT)
7 i = an.AddSTE('i', match=True)
8 an.AddAnmlEdge(h, i, 0)
9
10 an.ExportAnml('hello_automata.anml')
```



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <automata-network id="hello_automata" name="hello_automata">
3   <description></description>
4   <state-transition-element id="h" symbol-set="h" start="all-input">
5     <activate-on-match element="i"/>
6   </state-transition-element>
7   <state-transition-element id="i" symbol-set="i">
8     <report-on-match/>
9   </state-transition-element>
10 </automata-network>
```

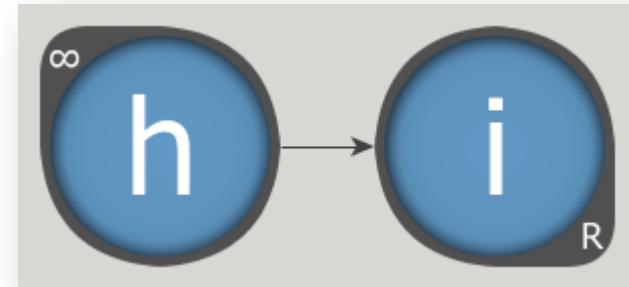


```
apcompile -f hello_automata.fsm hello_automata.anml
```



```
111010
101001
```

Workbench



ANML

# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

# Composition of Automata

- Automata Network Markup Language (ANML)
- AP Workbench
- Macros
- Programming APIs

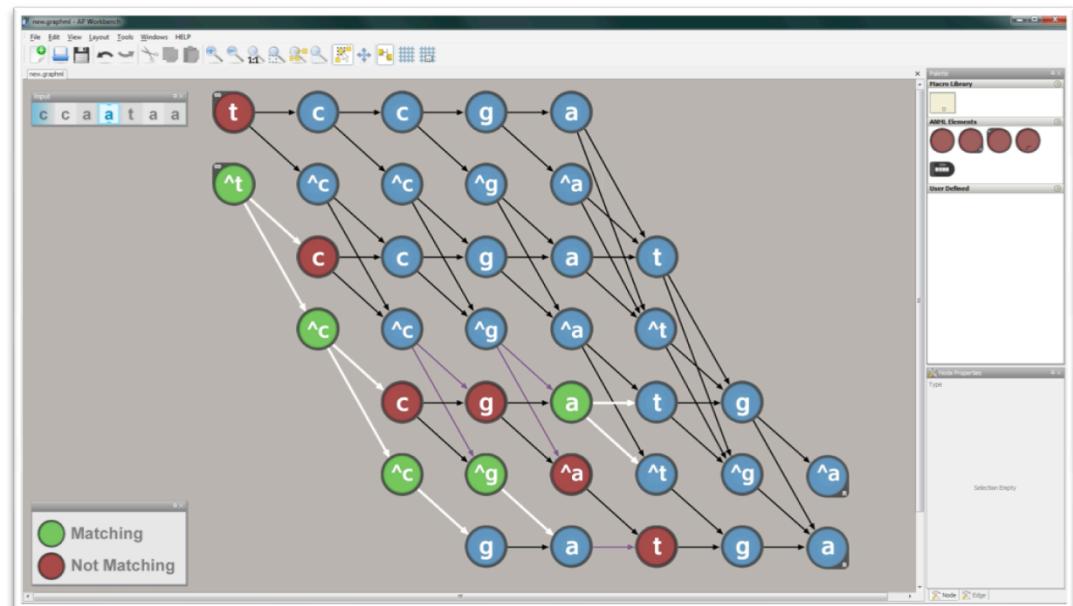
# ANML

- XML-based language for tool-to-tool communication
- Direct creation/manipulation discouraged
- Import into Workbench, export from Workbench
- Import/Export from a program using APIs
- An input language for the compiler

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <automata-network name="motif2" id="motif2">
3       <state-transition-element id="1m1" symbol-set="g" start="all-input">
4           <activate-on-match element="1m2"/>
5           <activate-on-match element="1mm2"/>
6       </state-transition-element>
7       <state-transition-element id="1m2" symbol-set="c">
8           <activate-on-match element="1m3"/>
9           <activate-on-match element="1mm3"/>
10      </state-transition-element>
11      <state-transition-element id="1m3" symbol-set="g">
12          <activate-on-match element="1m4"/>
13          <activate-on-match element="1mm4"/>
14      </state-transition-element>
15      <state-transition-element id="1m4" symbol-set="a">
16          <activate-on-match element="1m5"/>
17          <activate-on-match element="1mm5"/>
18      </state-transition-element>
19      <state-transition-element id="1mm1" symbol-set="[^g]" start="all-input">
20          <activate-on-match element="2mm2"/>
21          <activate-on-match element="2m2"/>
22      </state-transition-element>
23      <state-transition-element id="1mm2" symbol-set="[^c]">
24          <activate-on-match element="2m3"/>
25          <activate-on-match element="2mm3"/>
26      </state-transition-element>
27      <state-transition-element id="1mm3" symbol-set="[^g]">
28          <activate-on-match element="2m4"/>
```

# AP Workbench

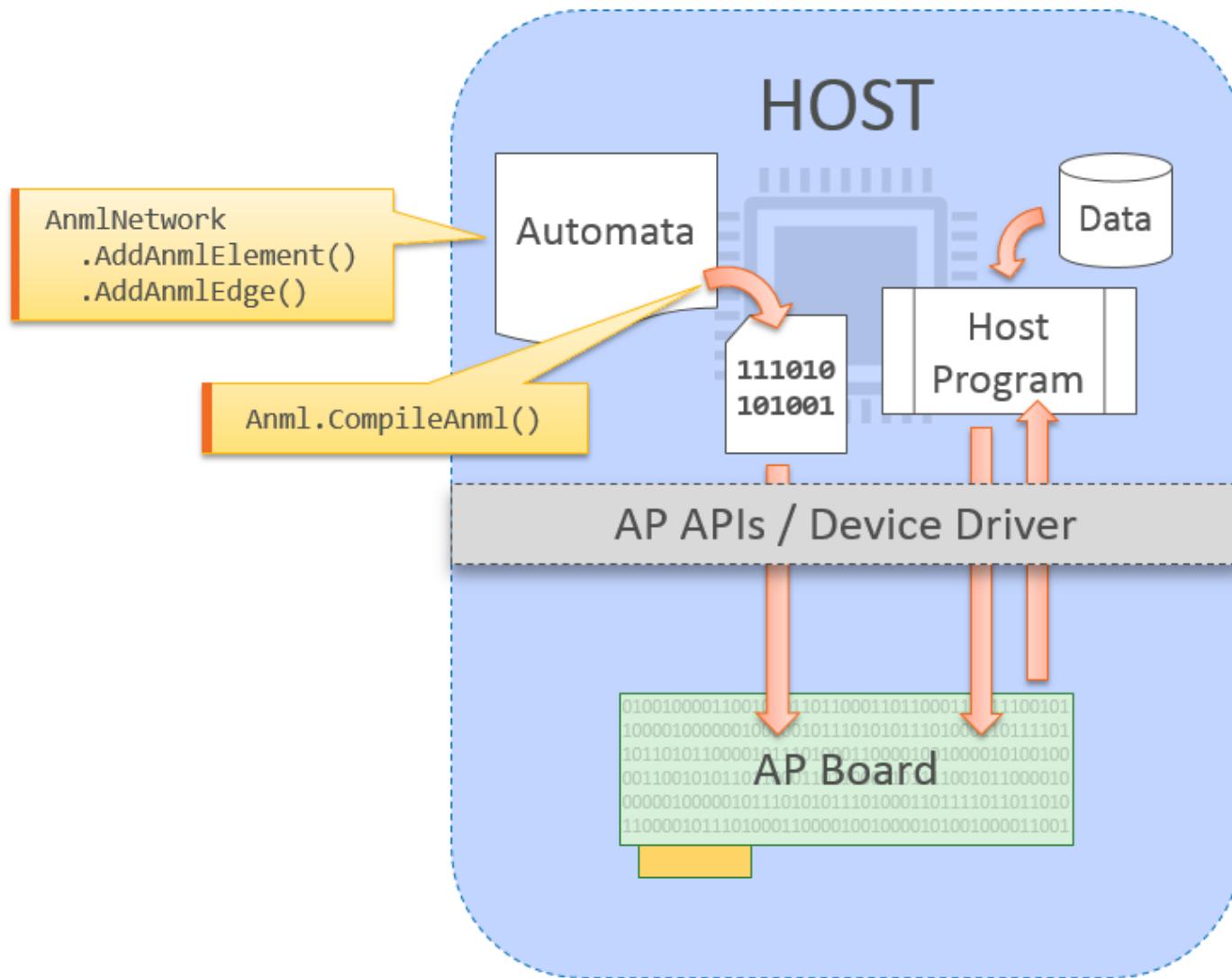
- ▶ AP Workbench tool enables graphical automata design
- ▶ Exposes all core elements found in the AP
  - STEs
  - Counters
  - Booleans
- ▶ Supports visual design simulation and debug



# Macros

- ▶ Reusable networks of AP elements
- ▶ Facilitate modularity and code reuse
- ▶ Allows designers to create libraries of user-defined functionality
- ▶ Enable partial reconfiguration
- ▶ Can be created in AP Workbench or programming APIs

# Design API

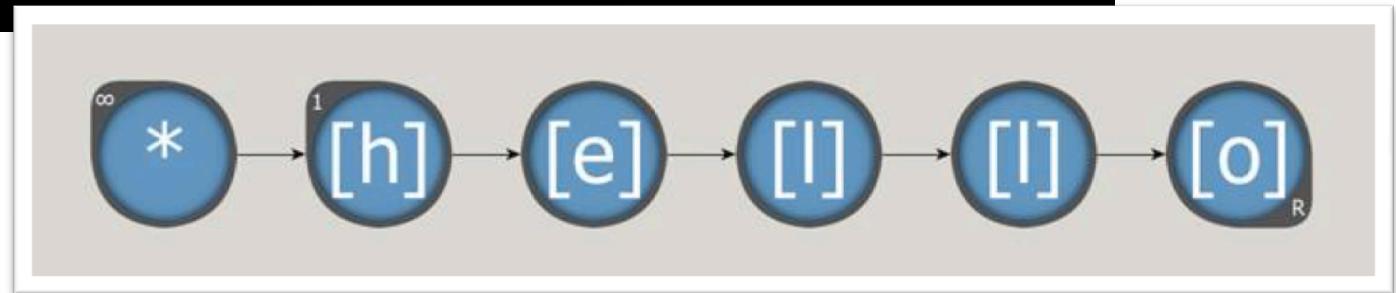


# Design API

- **Anml.CreateAnml()**
  - Creates ANML workspace, a container for automata networks & macros
- **Anml.CreateAutomataNetwork()**
  - Creates new empty automata network
- **AnmlNetwork.AddAnmlElement()**
  - Creates an STE, counter, or Boolean element in the automata network
- **AnmlNetwork.AddAnmlEdge()**
  - Creates a directed edge between two elements
- **Anml.CompileAnml()**
  - Compiles all elements in the ANML workspace into a binary automaton

# Design API Example

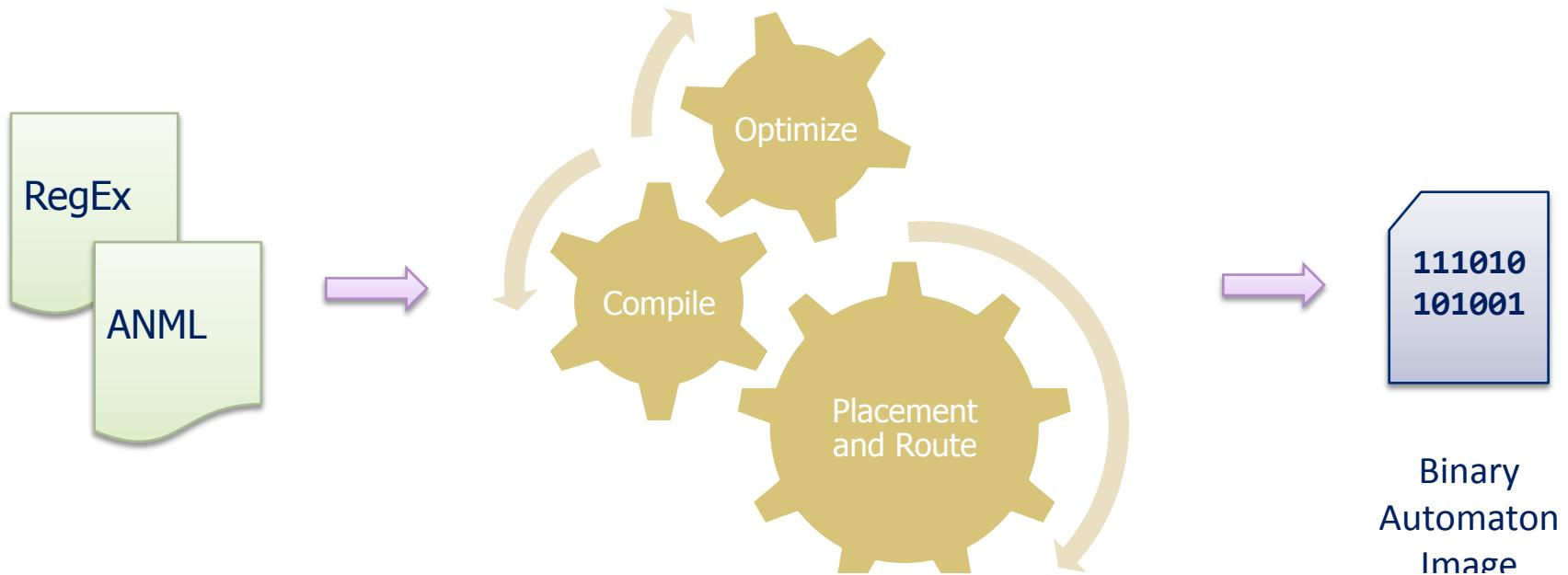
```
1 from micronap.sdk import *
2
3 A = Anml()
4 AN = A.CreateAutomataNetwork()
5
6 start_node = AN.AddSTE("*", AnmlDefs.ALL_INPUT, anmlId="start")
7 h_node     = AN.AddSTE("h", AnmlDefs.START_OF_DATA, anmlId="h_node")
8 e_node     = AN.AddSTE("e", anmlId="e_node")
9 l_node_1   = AN.AddSTE("l", anmlId="l_node_1")
10 l_node_2  = AN.AddSTE("l", anmlId="l_node_2")
11 o_node    = AN.AddSTE("o", anmlId="o_node", match=True)
12
13 AN.AddAnmlEdge(start_node, h_node, 0)
14 AN.AddAnmlEdge(h_node,      e_node, 0)
15 AN.AddAnmlEdge(e_node,      l_node_1, 0)
16 AN.AddAnmlEdge(l_node_1,    l_node_2, 0)
17 AN.AddAnmlEdge(l_node_2,    o_node, 0)
18
19 AN.ExportAnml('hello_world.anml')
20
```



# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

# Compilation



- Compiler assigns specific chip resources and routing
- Compiler performs network optimization, ensures feasibility
- Output from compiler is a programming object that can be loaded into the hardware or saved as a file
- Saved FSM files can be read from the file system and loaded into the hardware at a later time.

# Ways to invoke the Compiler

## 1) Using the command line apcompile tool

1) A file containing regular expressions

2) An ANML file

- Created via AP Workbench
- Created via C, Python, or Java program then exported to ANML
- Hand-generated ANML (discouraged)

▪ `apcompile -A -f <output.fsm> <input_anml_file.anml>`

## 2) Using an API call within a program (C, Python, or Java)

# Compile API Example

```
import micronap.sdk as ap

a = ap.Anml()
an = a.CreateAutomataNetwork(anmlId='hello_automata')

h = an.AddSTE('h', startType=ap.AnmlDefs.ALL_INPUT)
i = an.AddSTE('i', match=True)
an.AddAnmlEdge(h, i, 0)

fsm, emap = a.CompileAnml()
fsm.Save('hello_automata.fsm')
```

# Compiler Options

- ▶ -MT
  - Use multi-threading when possible
- ▶ -O1|--best-quality
  - Trade better utilization for longer compile time
- ▶ -Od
  - Disable automaton minimization
- ▶ --disable-counters
  - Disable the use of counters for quantifications

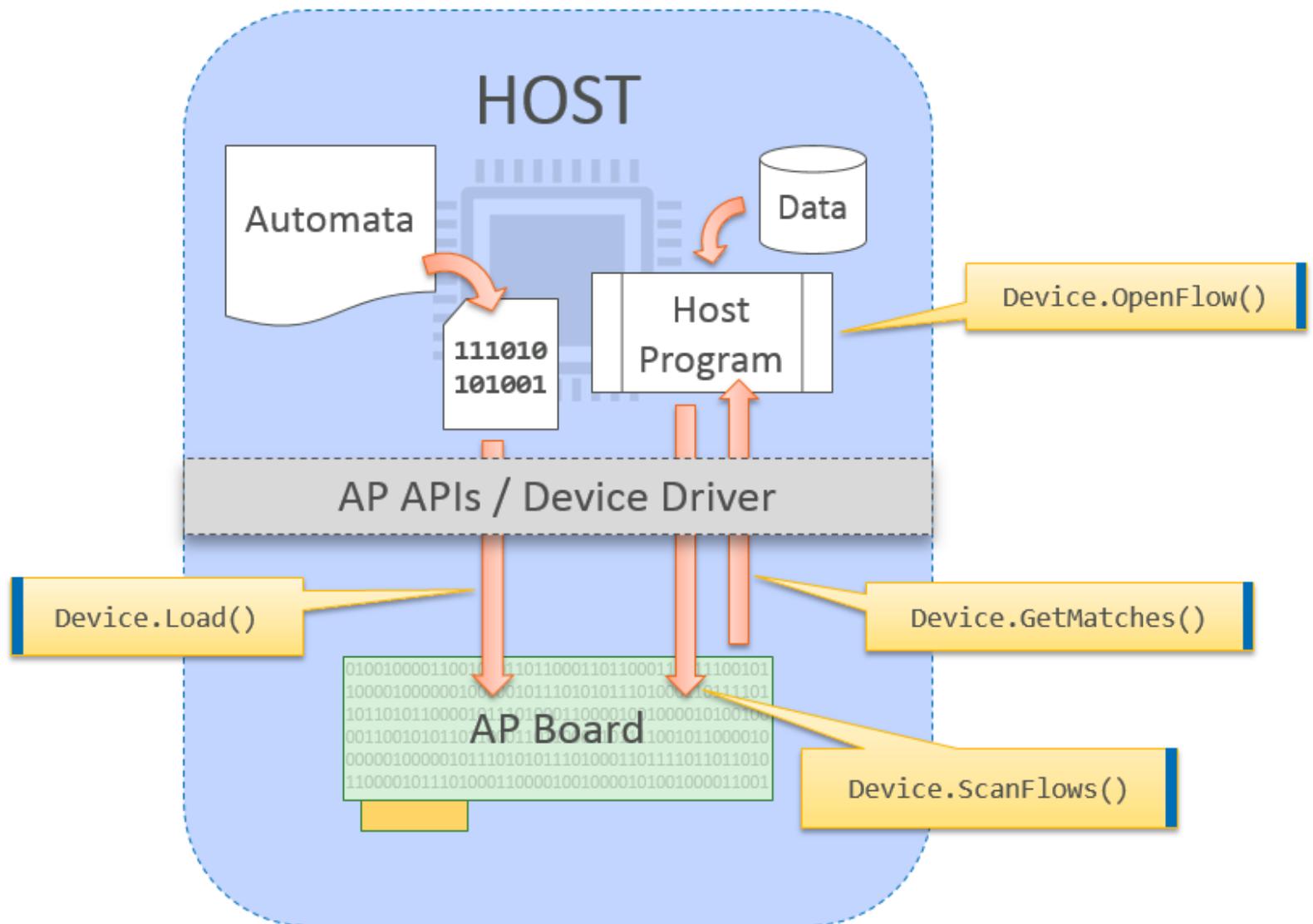
# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

# Runtime (Task Management)

- Input Data Preparation
- Output Data Processing
- Device Management
- Loading and Invoking Automata

# Runtime API



# Input Data Preparation

- ▶ Concept of a flow
  - Independent search task on an automaton
  - Software abstraction of state vector
- ▶ Flow data segmentation
  - Input data is segmented into chunks
  - Size defined by user; 32KB or larger is optimal

```
flow = dev.OpenFlow(rto)
with open('huge_file.dat') as f:
    while True:
        data = f.read(32 * 1024)
        if not data:
            break
        dev.ScanFlows([ (flow, data) ])
    ...
```

# Output Data Processing

- ▶ Report Event Vectors
  - Pattern matches (reports) can occur on any input symbol
  - Any element may potentially report on a given symbol
  - Reports are grouped into a vector
  - Most efficient to align multiple reports to occur in same cycle
  - Match results contain data offset (when) and ID of reporting element (who)

# Runtime API

- Device.ConfigureDevice()
  - Defines partitions of the physical device's AP chips
- Device.OpenDevice()
  - Connects the user application to the physical AP device
  - Returns a device handle
- Device.Load()
  - Loads a compiled binary automaton onto the AP device
  - Returns an automaton handle

# Runtime API

- Device.OpenFlow()
  - Creates a new search task on an automaton
- Device.ScanFlows()
  - Writes data from the user application to the device
  - Supports multiple data writes to multiple flows at once
- Device.GetMatches()
  - Retrieves matches resulting from a call to Device.ScanFlows()
  - Synchronous or asynchronous (between CPU and AP)

# Runtime API

```
import micronap.sdk as ap

fsm = ap.Automaton()
fsm.Restore('hello_automata.fsm')

ap.ConfigureDevice('/dev/frio0')
dev = ap.Device()
dev.OpenDevice('/dev/frio0')
rto = dev.Load(0, fsm)

...
```

# Runtime API

...

```
flow = dev.OpenFlow(dev, rto)
wait = dev.ScanFlows([ (flow, "test data") ])
dev.Wait(wait)

for match in iter(dev.GetMatches, None):
    print 'Match at offset %lu' % match.byte_offset
```

# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- Simulation and debug
- Live Workbench demo

# Libraries

- ▶ Common methods and applications can be standardized into libraries in the SDK
- ▶ Allows for simple reuse of highly optimized methods
- ▶ Shifts burden of automata design to the SDK

# Libraries

- ▶ Currently Available
  - Regular Expression tools
- ▶ In Development
  - Approximate String Match library
- ▶ Future Directions
  - Higher-level design languages
  - Numeric Comparison library
  - Application-based libraries

# Regular Expression Tools

- ▶ Regular expressions convert directly into automata
  - Use **apcompile** with a file of regular expressions

regxes.txt

```
id1: /((su|mo)n|(tues))day/
id2: /(book|dead)(end){1,3}/
id3: /w.{0,5}ed/
```

```
apcompile -f expressions.fsm regxes.txt
```

- Use the **programming APIs** to load and compile the expressions
- Use **Workbench** GUI tool to convert a regular expression into visual form

# AP Programming

- An overview of AP software ecosystem
- Composition of automata
  - ANML; AP Workbench; APIs (C, Python, Java)
- Compilation
  - Compile from API and command line; Compilation options
- Runtime (Task Management)
  - Input data preparation; Output data processing; Device management; Loading and invoking automata
- Libraries
- **Simulation and debug**
- Live Workbench demo

# Simulation and Debug

- ▶ AP Workbench
  - Visual design simulator (using batchSim engine)
- ▶ batchSim
  - Command-line simulator
  - Executes automaton without regard to hardware constraints
- ▶ apemulate
  - Command-line simulator
  - Approximates real device constraints and execution

# Live Workbench Demo

- ▶ <https://www.youtube.com/watch?t=40&v=gw9wmcA9LIQ>

# Input Data Preparation

- ▶ Working with multiple flows
  - Can open multiple flows on a single automaton
  - API search function is vectored
    - Accepts one or more flow/data pairs in a single call
    - Each flow/data pair can be associated with independent automata

```
Device.ScanFlows([ (flow1, data1), (flow2, data2), ... ])
```

```
Device.ScanFlows([ (flow1, data1), (flow2, data1), ... ])
```