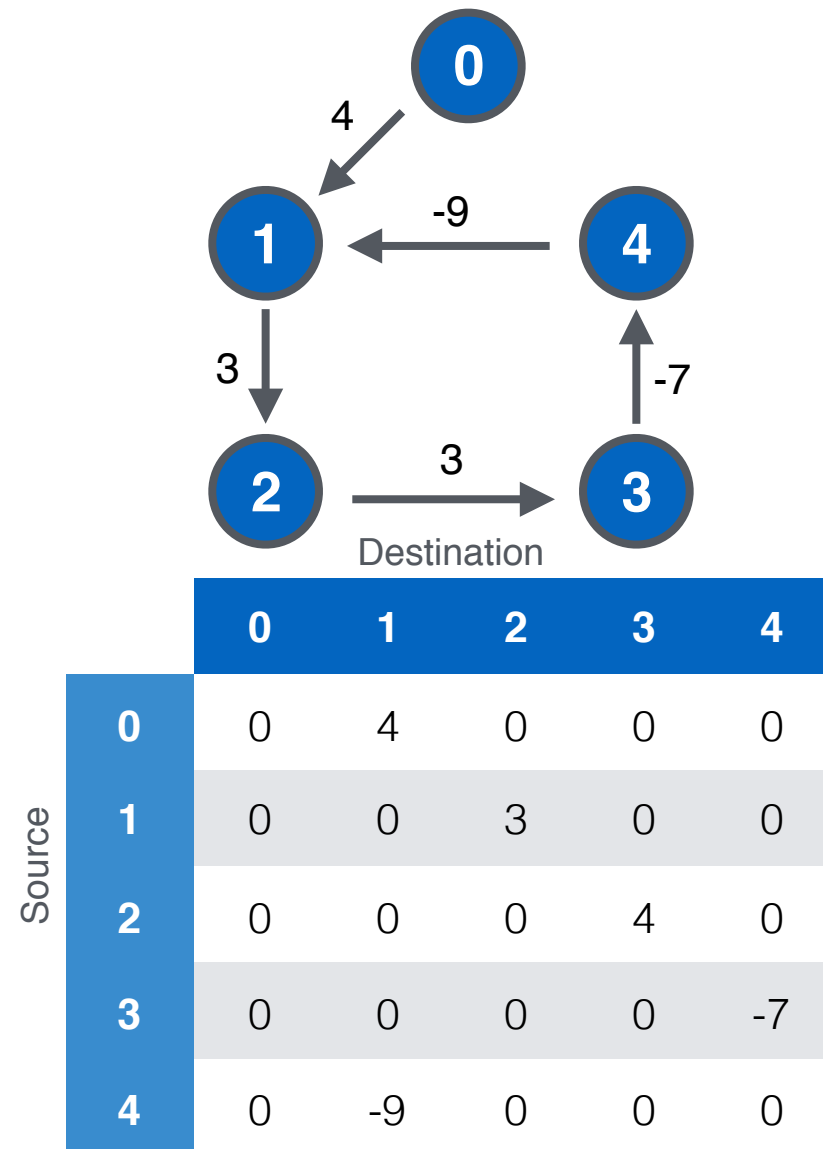# High-Frequency FOREX Trading:

Identification of Triangular Arbitrage Opportunities
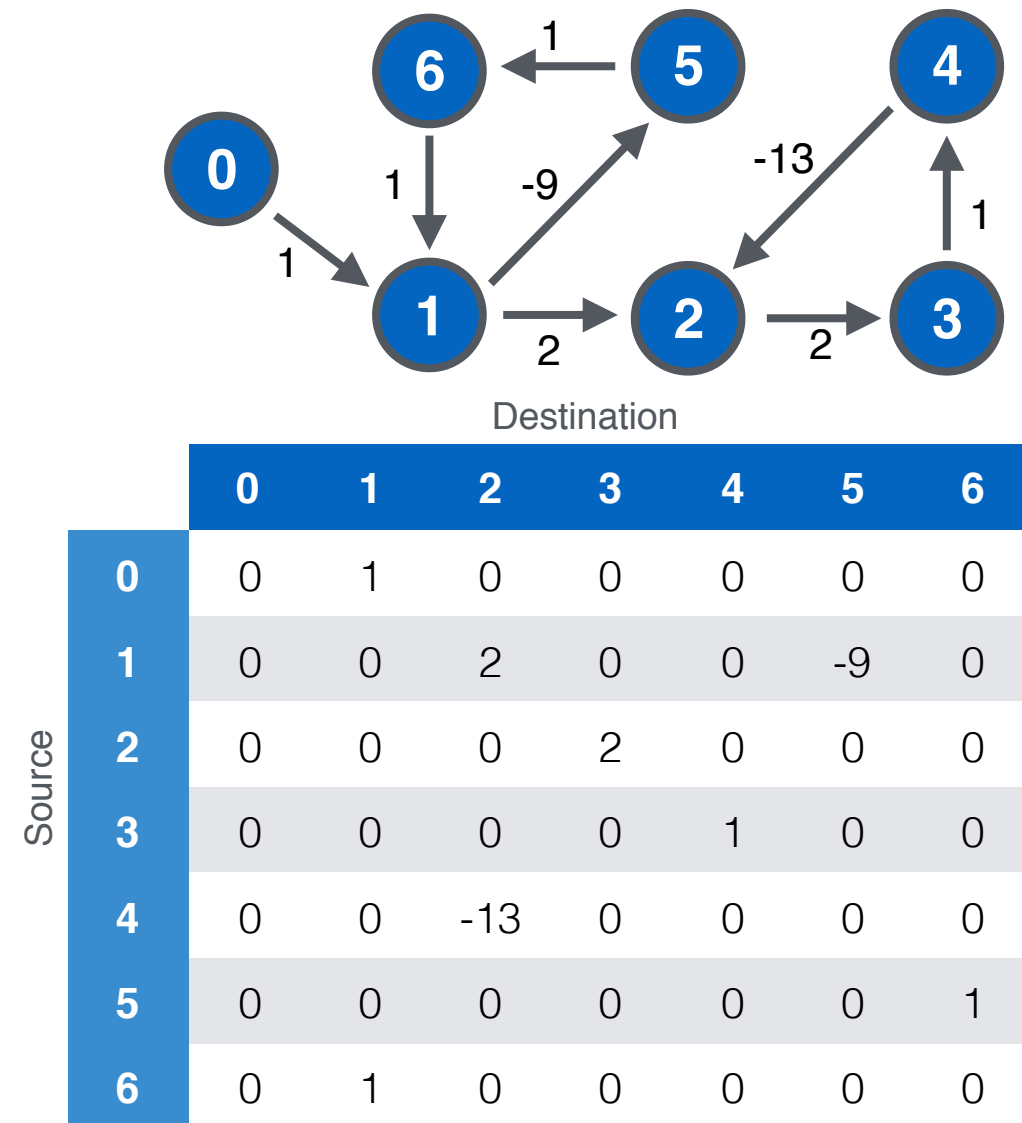
Graham Gobieski, Kevin Kwan, Ziyi Zhu, Shang Liu

Demos

Demo 1

	0	1	2	3	4
0	0	4	0	0	0
1	0	0	3	0	0
2	0	0	0	4	0
3	0	0	0	0	-7
4	0	-9	0	0	0

Demo 2

|	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	0	0	2	0	0	-9	0
2	0	0	0	2	0	0	0
3	0	0	0	0	1	0	0
4	0	0	-13	0	0	0	0
5	0	0	0	0	0	0	1
6	0	1	0	0	0	0	0

Demo 3: Live Cycles

Motivation

- High-Frequency Trading: taking advantage of opportunities (inefficiencies, etc.) on very short timescales

- Triangular Arbitrage: due to market inefficiencies, exchanging a currency between three or more currencies and arriving back at the original currency might be profitable

- Timescale: ~5-20ms, data streams over network

1.02 x 1.06 x .978 > 1
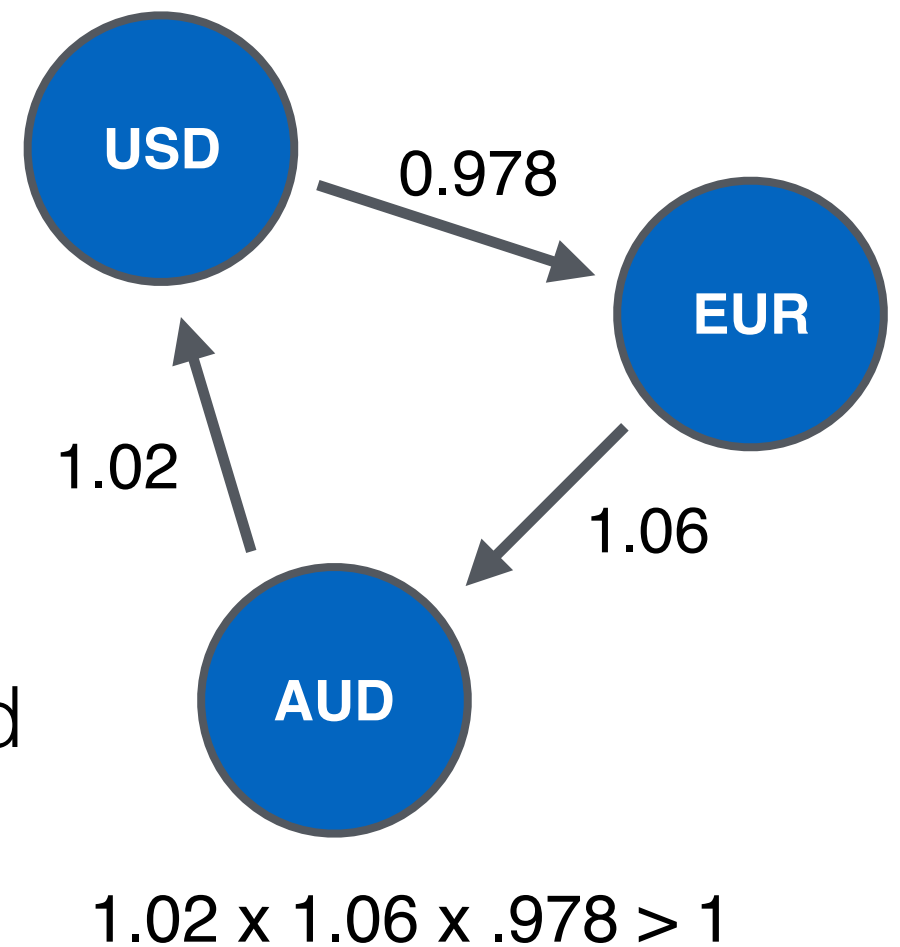
Bellman-Ford

```
for each vertex x in V do
    if x is source then
        w(x) = 0
    else
        w(x) = INFINITY
        p(x) = NULL
    end if
end for

for i = 1 to v - 1 do
    for each edge(i, j) in E do
        if w(i) + w(i, j) < w(j) then //Relaxation
            w(j) = w(i) + w(i, j)
            p(j) = i
        end if
    end for
end for

for each edge(i, j) in E do
    if w(j) > w(i) + w(i, j) then
        //Found Negative-Weight Cycle
    end if
end for
```
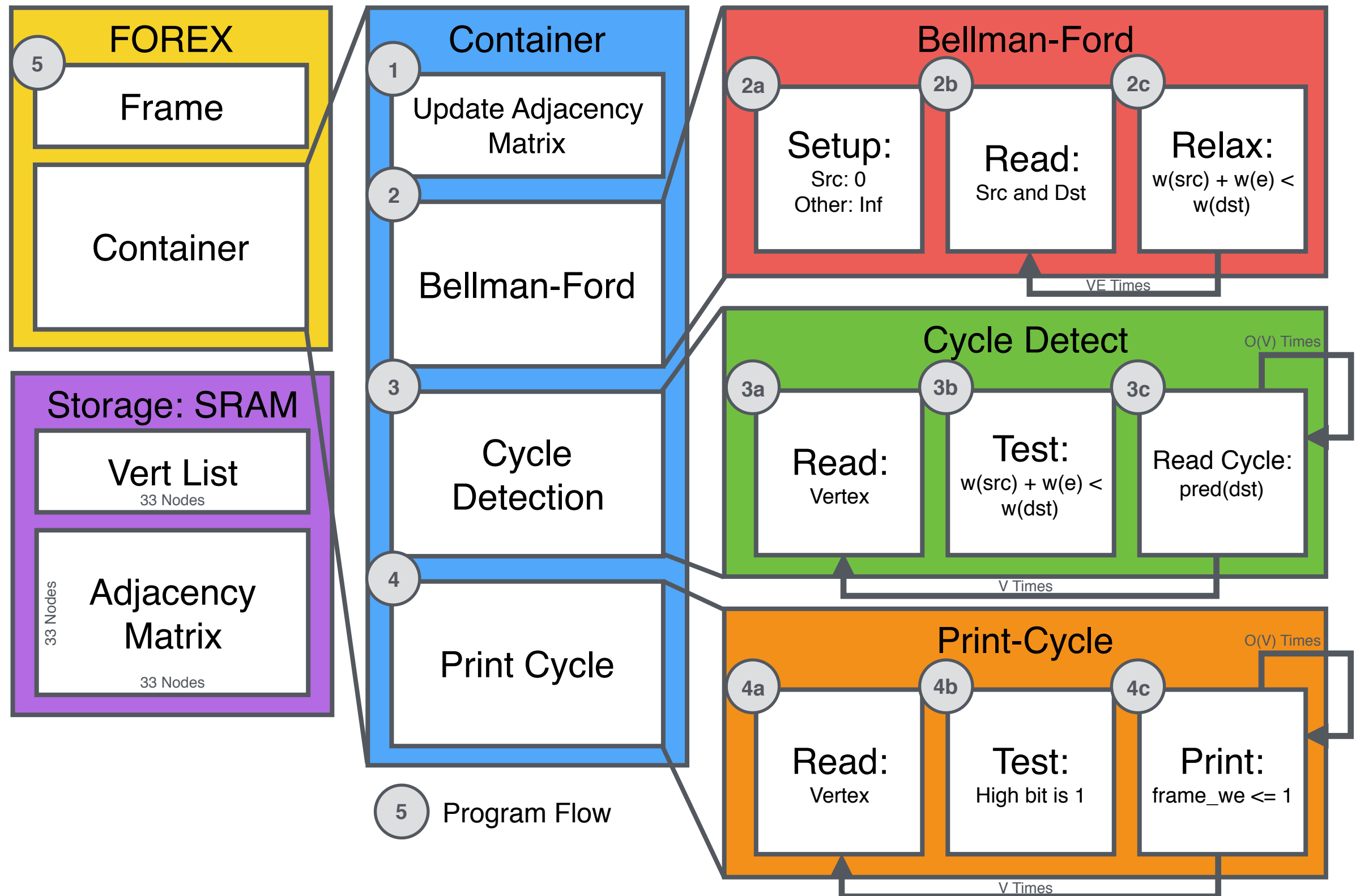
Transformation
1. w1 * w2 * w3 * ... * wn > 1
2. log(w1) + log(w2) + log(w3) + ... + log(wn)< 0
3. -(log(w1) + log(w2) + log(w3) + ... + log(wn))< 0

Hardware Design

FOREX
5

Frame

Container

Storage: SRAM
Vert List
33 Nodes

Adjacency Matrix
33 Nodes

33 Nodes

Container
1 Update Adjacency Matrix

2 Bellman-Ford

3 Cycle Detection

4 Print Cycle

5 Program Flow

Bellman-Ford
2a Setup:
Src: 0
Other: Inf

2b Read:
Src and Dst

2c Relax:
$w(src) + w(e) < w(dst)$

VE Times

Cycle Detect
O(V) Times

3a Read:
Vertex

3b Test:
$w(src) + w(e) < w(dst)$

3c Read Cycle:
pred(dst)

V Times

Print-Cycle
O(V) Times

4a Read:
Vertex

4b Test:
High bit is 1

4c Print:
frame_we <= 1

V Times

Overview

Bellman

Print Cycle

Software Design

Option 1

Python Front-End
1. Load Data
2. Preprocess Data
3. Write Data via custom Ioctl call

Kernel Module
1. Setup Memory-mapped I/O
2. Write Data to Bus

Amba Bus

Option 2

C Front-End
1. Load Data
2. Preprocess Data
3. Write Data via custom Ioctl call
4. Write keyboard events

Kernel Module 2
1. Setup Memory-mapped I/O
2. Write Key Data to Bus

Amba Bus

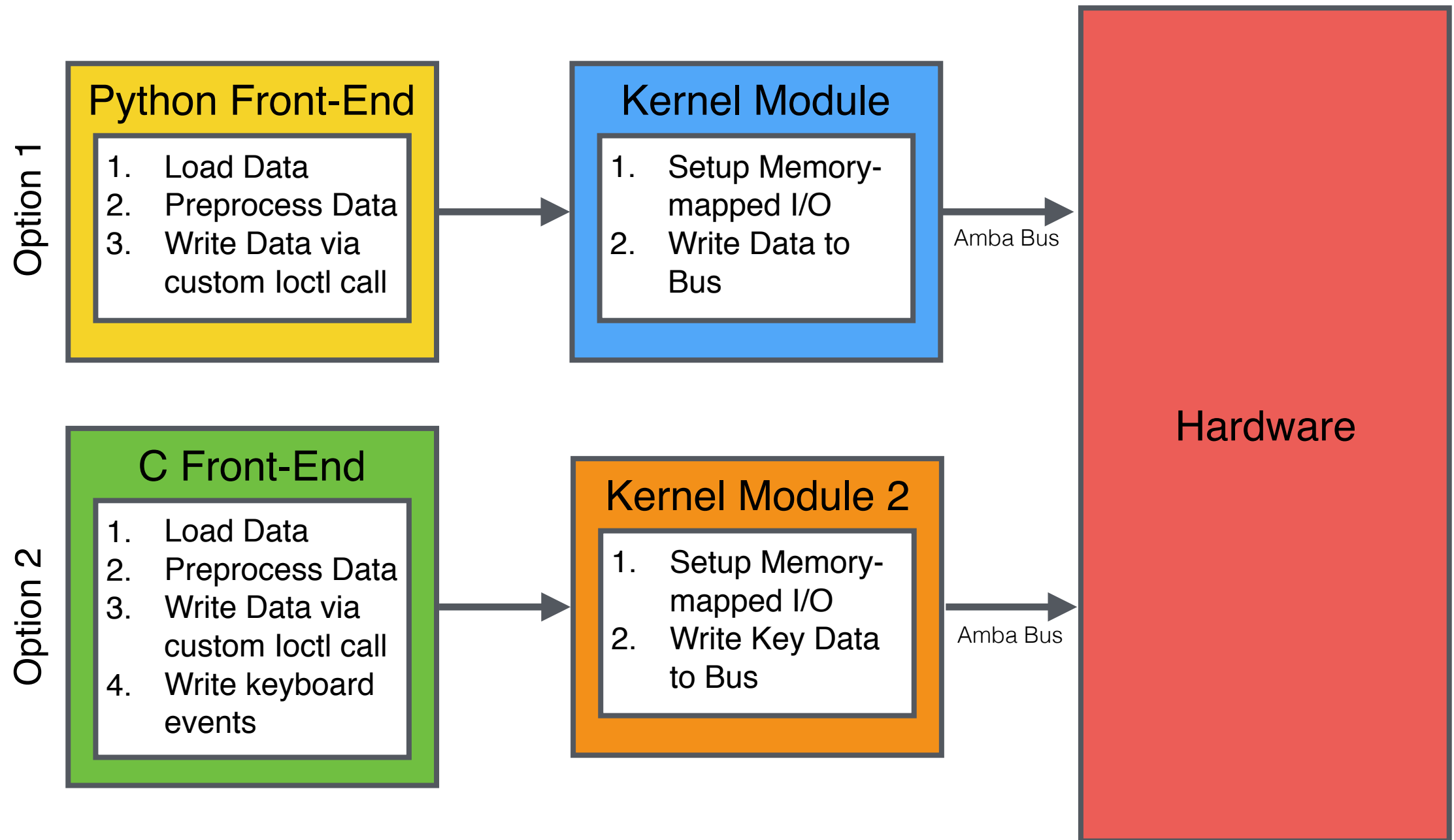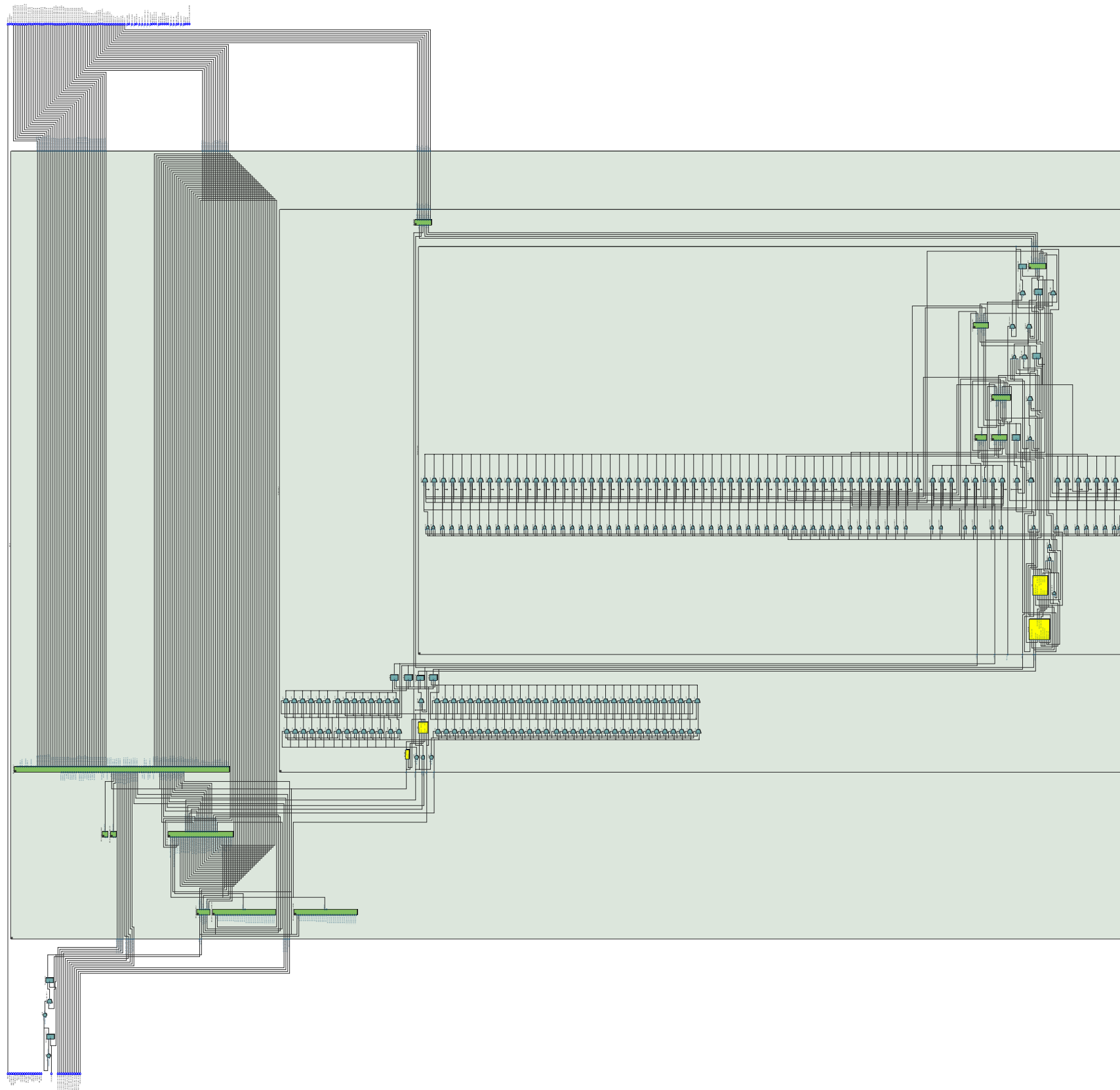**Hardware**
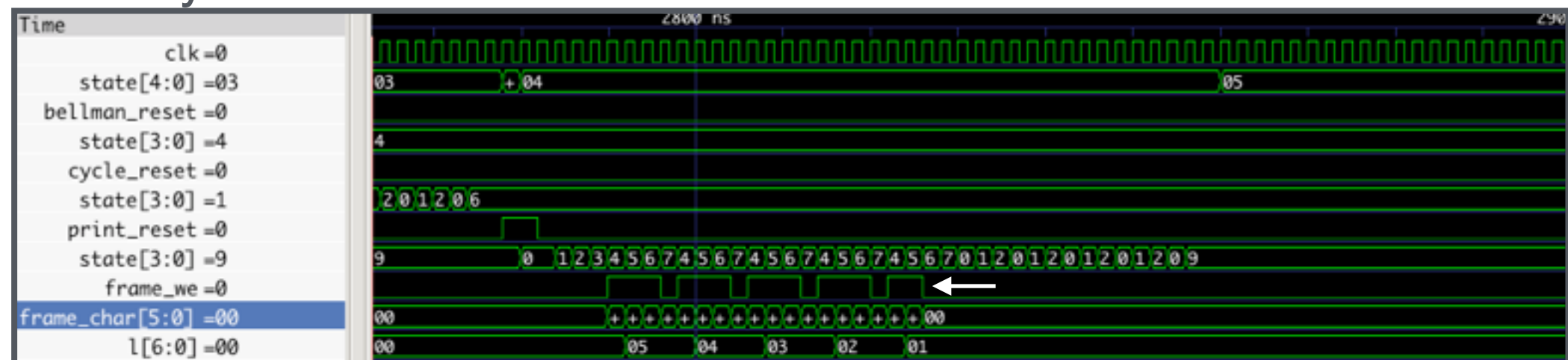
RTL Synthesis

Challenges

- Obviously Timing…

- Memory Accesses

- Nested Non-blocking Assignments

- Combinational Logic and Sequential Logic Working Together

- Compile Time