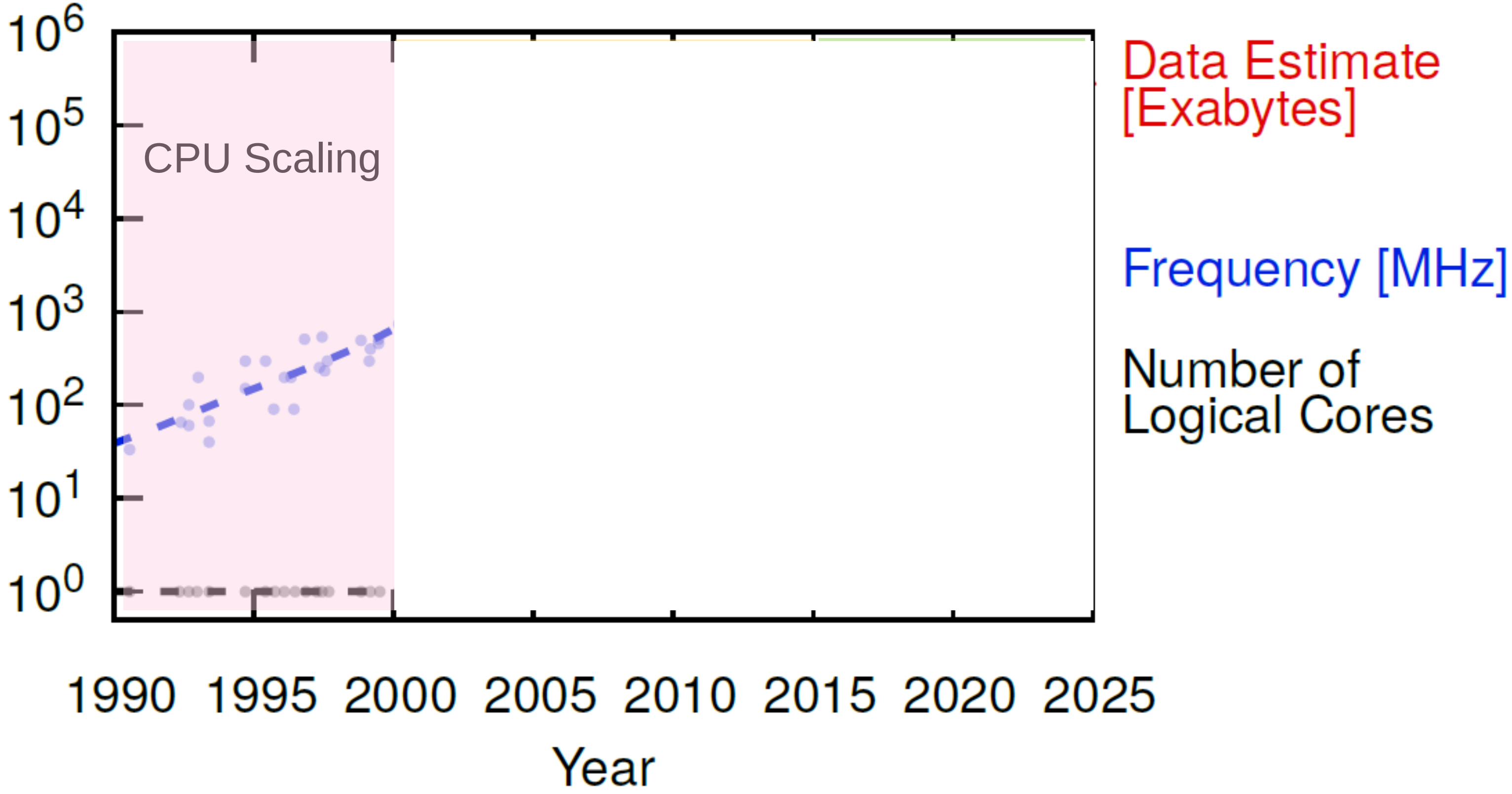
Offloading Compute-intensive Tasks to FPGAs in the Datacenter

Zsolt István

IMDEA Software Institute, Madrid

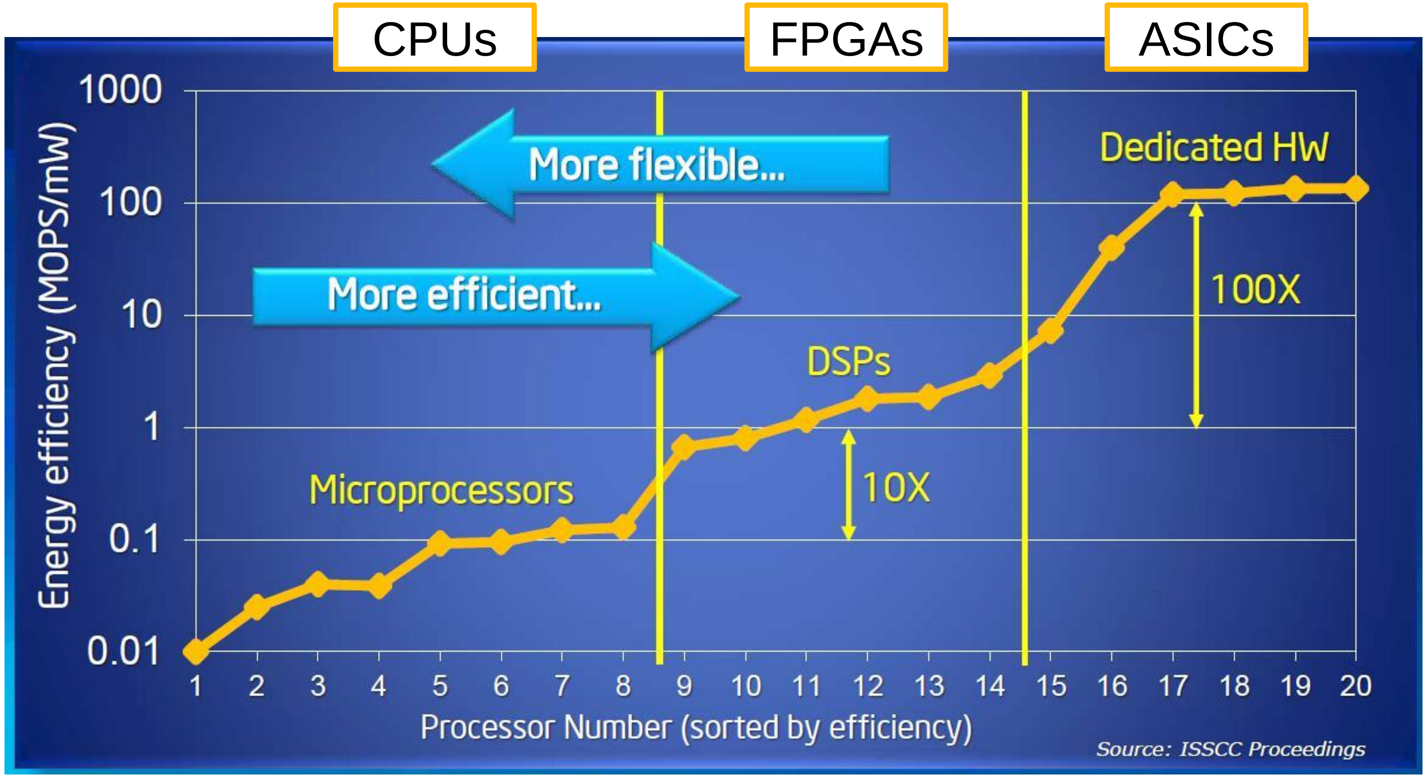
zsolt.istvan@imdea.org

History Lesson



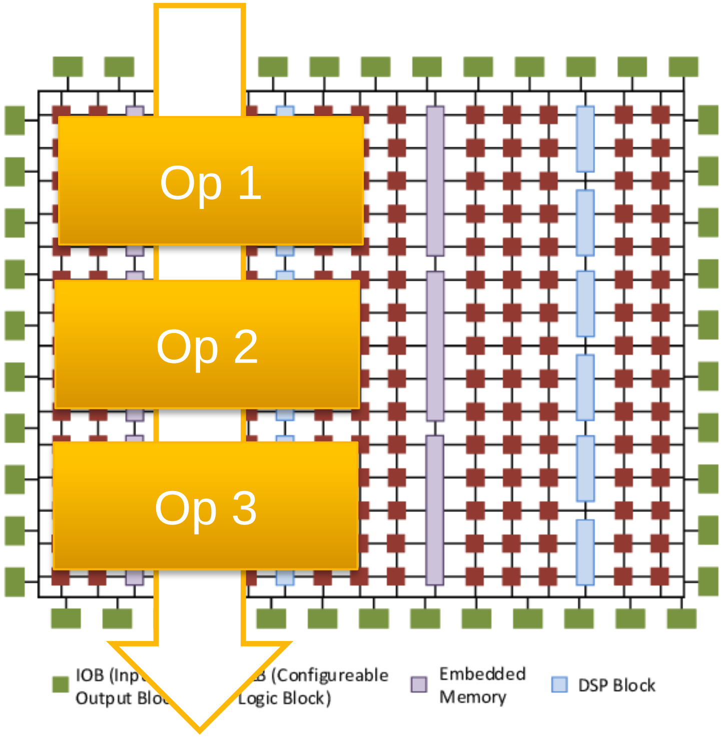
Based on a plot layout by K. Rupp. Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten; 2010-2015 by K. Rupp. Data growth estimate by C. Maxfield.

FPGAs as a middle ground



Source: ISSCC Proceedings

FPGAs in Research and Datacenter

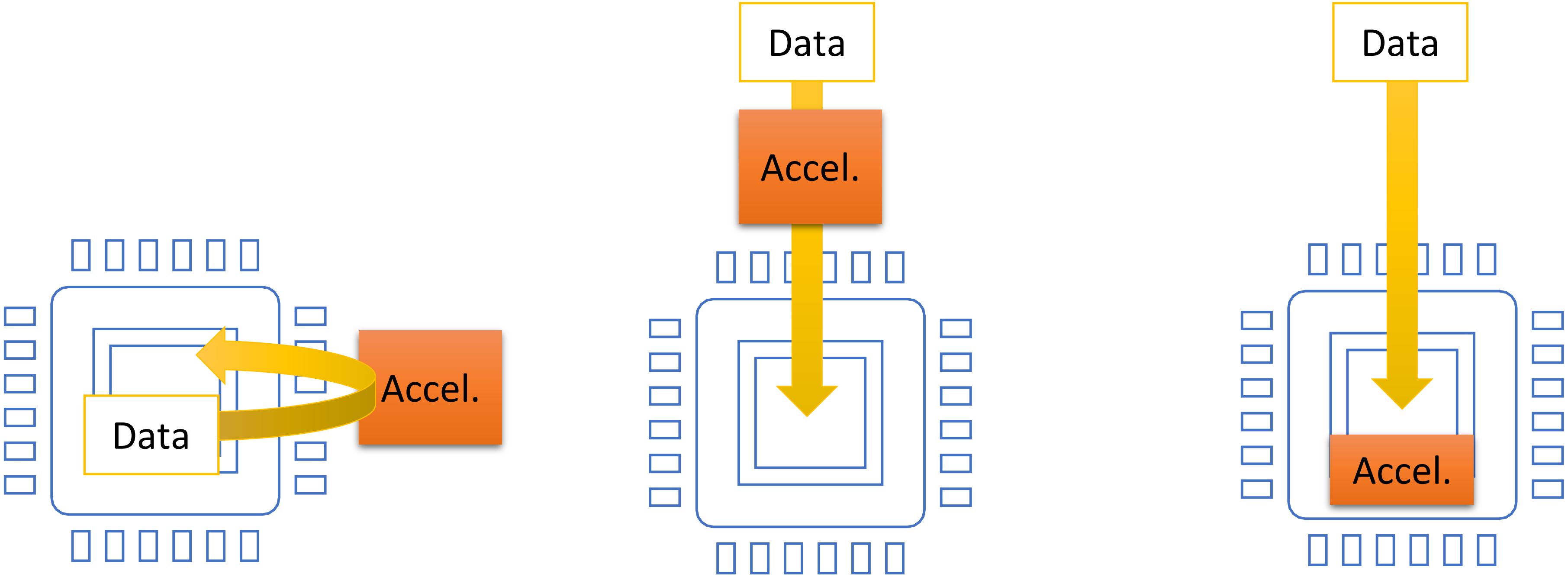


Field **P**rogrammable **G**ate **A**rray (FPGA)

- Free choice of architecture
- Fine-grained pipelining, communication, distributed memory
- Tradeoff: all “code” occupies chip space

- Running in the 100-600MHz range
- <25W power consumption

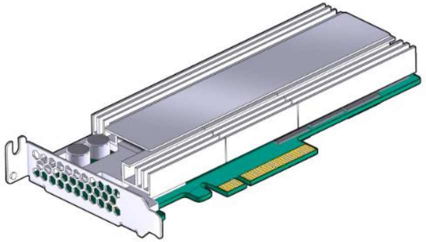
Integration Options

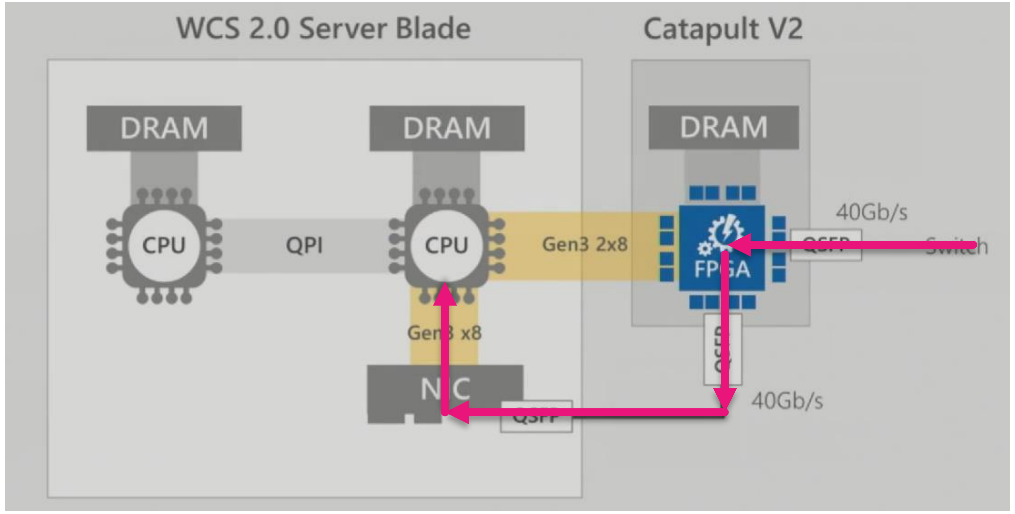


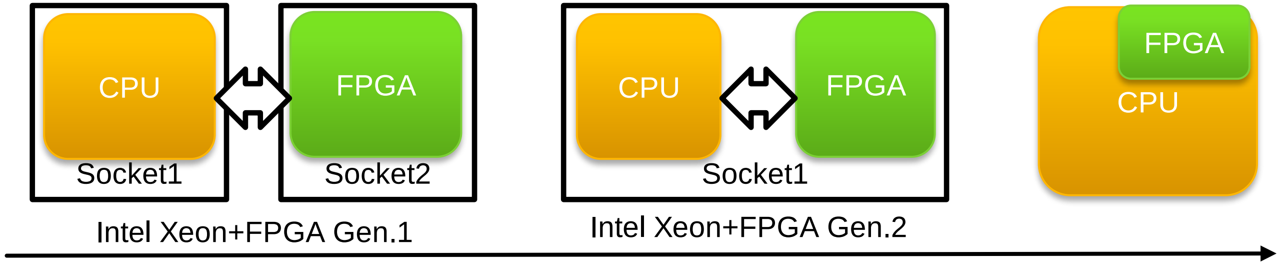
1) On the side **2) In data-path** **3) Co-processor**

In the Cloud Today

- Accelerator
  - Amazon F1
  - For compute-intensive tasks
- In data path
  - Microsoft Catapult
  - For reducing data movement
- Co-processor
  - Intel Xeon+FPGA
  - For compute tasks

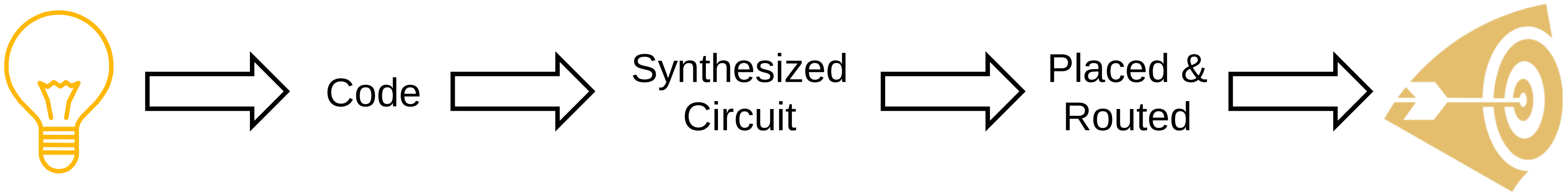






Programming FPGAs

- Challenge: adapting algorithms to the parallelism of the FPGA



Code

Synthesized Circuit

Placed & Routed

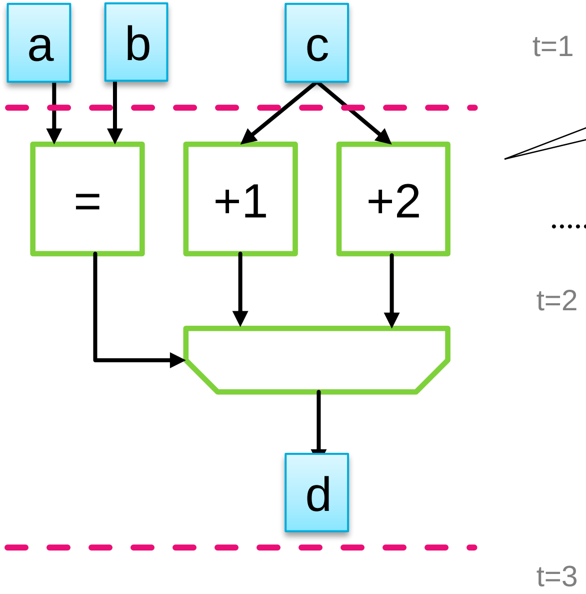
- Coding: Hardware definition languages, high level languages
- Synthesis: Produce a logic-gate level representation (any FPGA)
- Place & route: Circuit that gets mapped onto specific FPGA

From code to circuit

- All code is turned into registers and gates mapped to logic blocks
- Organize into *modules* – f_{module}*(inputs)* → *outputs*
- Execution *synchronous* to a clock

Programming in HDL (Verilog, VHDL): low level of abstraction, but full control

```
Loop forever:
if (a==b) then
  d <= c+1
else
  d <= c+2
end if
```



Fmax depends on "most expensive" step

Fmax = 100MHz
Latency = 1 cycle

From code to circuit

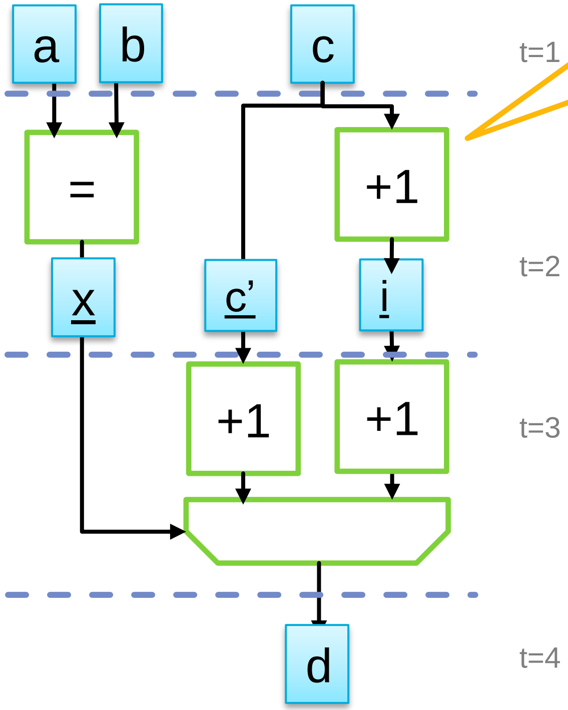
- All code is turned into registers and gates mapped to logic blocks
- Organize into *modules* – f_{module}*(inputs)* → *outputs*
- Execution *synchronous* to a clock

Modified HDL

```
Loop forever:
if (a==b) then
  x <= 0
else
  x <= 1
end if

c' <= c
i <= c+1

if (x==0) then
  d <= c'+1
else
  d <= i+1
end if
```



If programming the FPGA in HLS, the compiler can (sometimes) help

Fmax = 150MHz
Latency = 2 cycles

Two examples

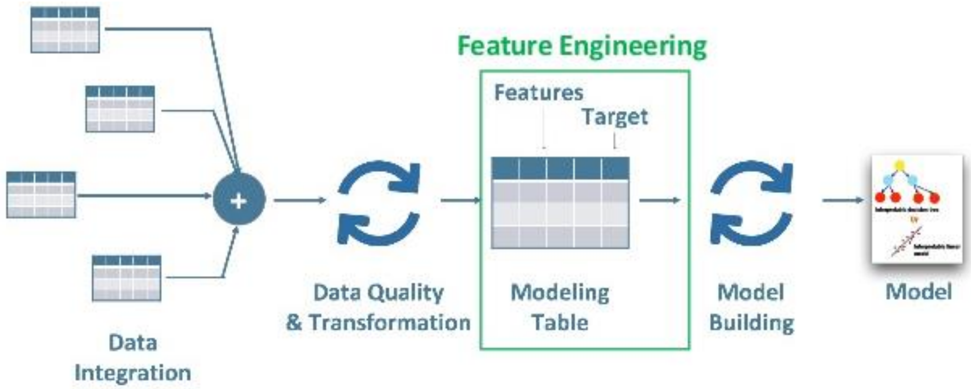
- Regular expression matching:
  - Constant throughput regardless of the expression
- K-means:
  - Flexible use of resources (throughput / exploration)

Regular Expressions in Data Analytics

- Regular expression → search pattern in text
  - Address in Cluj: `Cluj(( |-)Napoca)?`
- Filter rows in databases (map to regex)

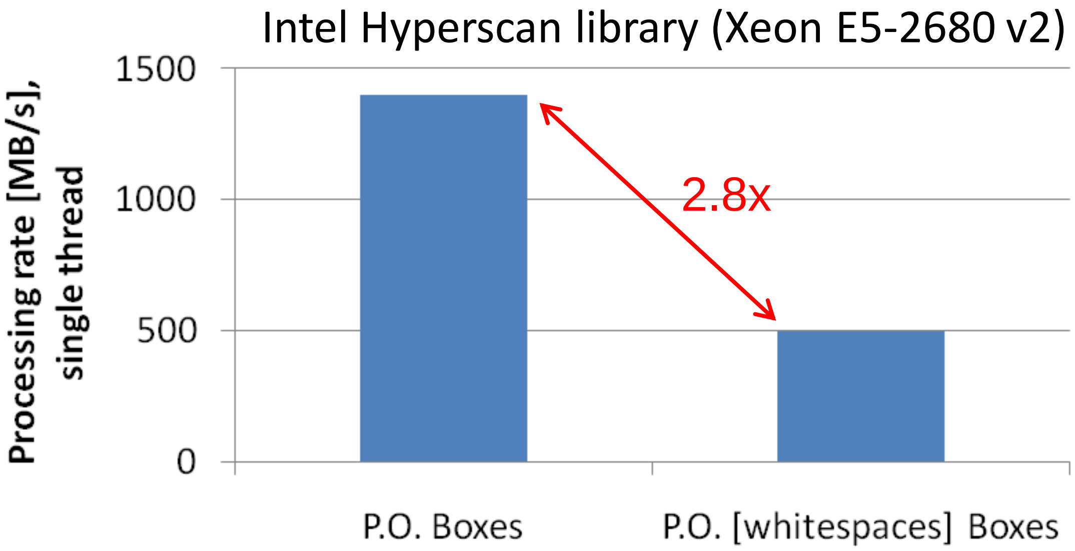
```
SELECT … FROM customer
WHERE  age<35 AND purchases>2
AND address LIKE “%Cluj%Napoca%”
```

- Feature engineering in machine learning pipelines



Regular Expression Matching – Challenges

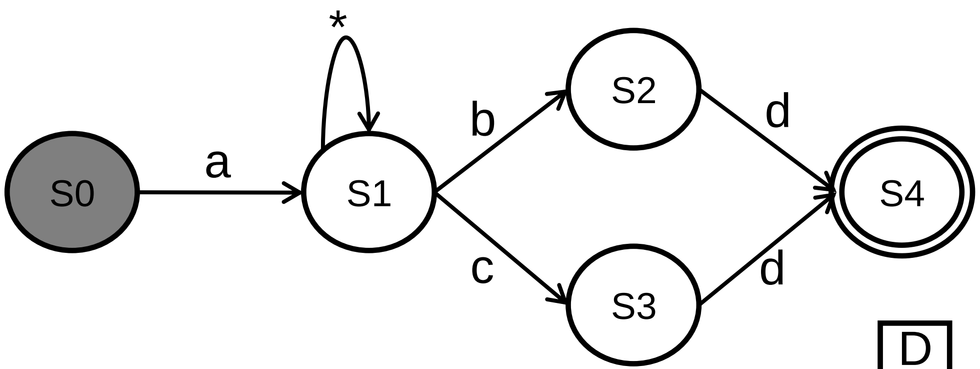
- Compute intensive & Performance depends on complexity



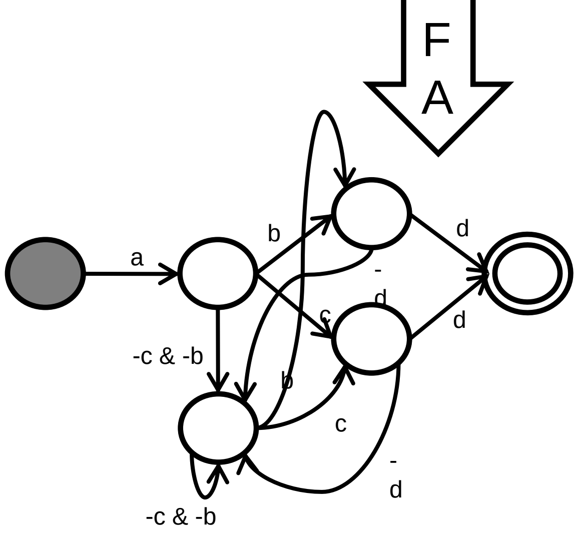
Finite State Automaton

Input: acbd

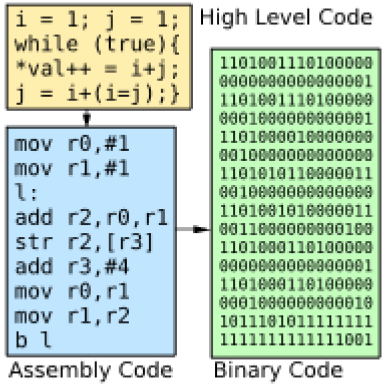
- a.*(b|c)d



- NFA vs. DFA
  - **N**ondeterministic – Multiple states active at the same time (more compute per input character)
  - **D**eterministic – State “explosion” (large footprint of state machine in memory)



Implementing NFAs


High Level Code

```
i = 1; j = 1;
while (true){
*val++ = i+j;
j = i+(i=j);}
```

```
mov r0,#1
mov r1,#1
l:
add r2,r0,r1
str r2,[r3]
add r3,#4
mov r0,r1
mov r1,r2
b l
```

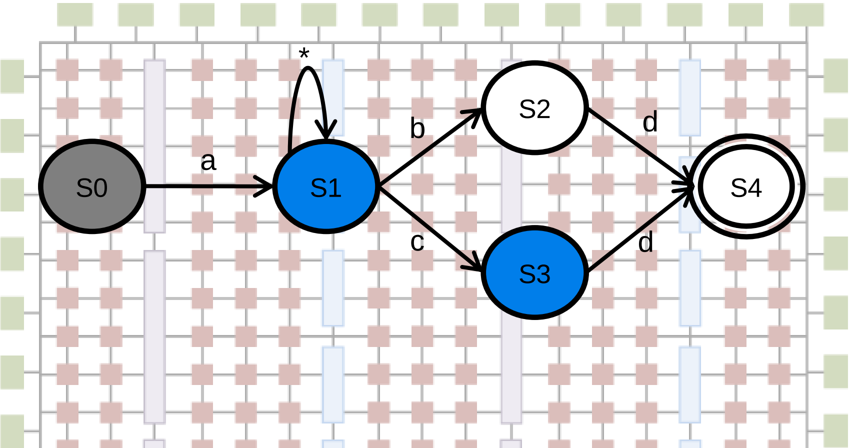
Assembly Code

```
1101001110100000
0000000000000001
1101001110100000
0001000000000001
1101000010000000
0010000000000000
1101010110000011
0010000000000000
1101001010000011
0011000000000100
1101000110100000
0000000000000001
1101000110100000
0001000000000010
1011101011111111
1111111111111001
```

Binary Code

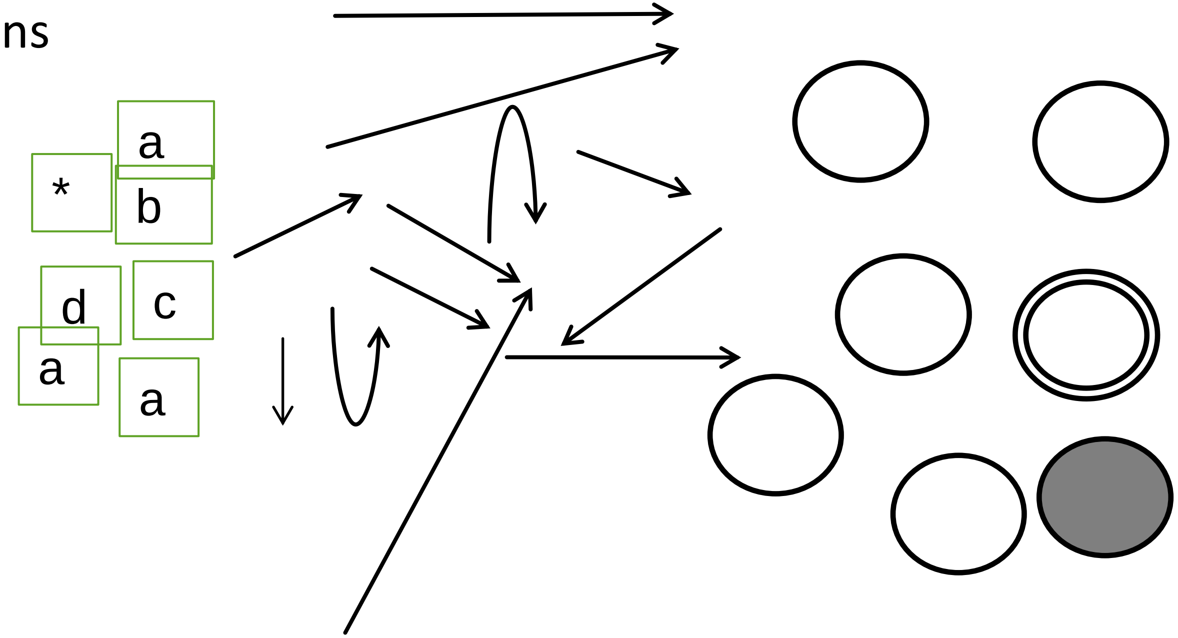
- CPUs turn NFAs into iterative instructions

- FPGAs good with NFAs
  - Typically in networking scenarios (SNORT rules, etc.)
  - The automaton can be compiled to actual circuitry → **no flexibility!**

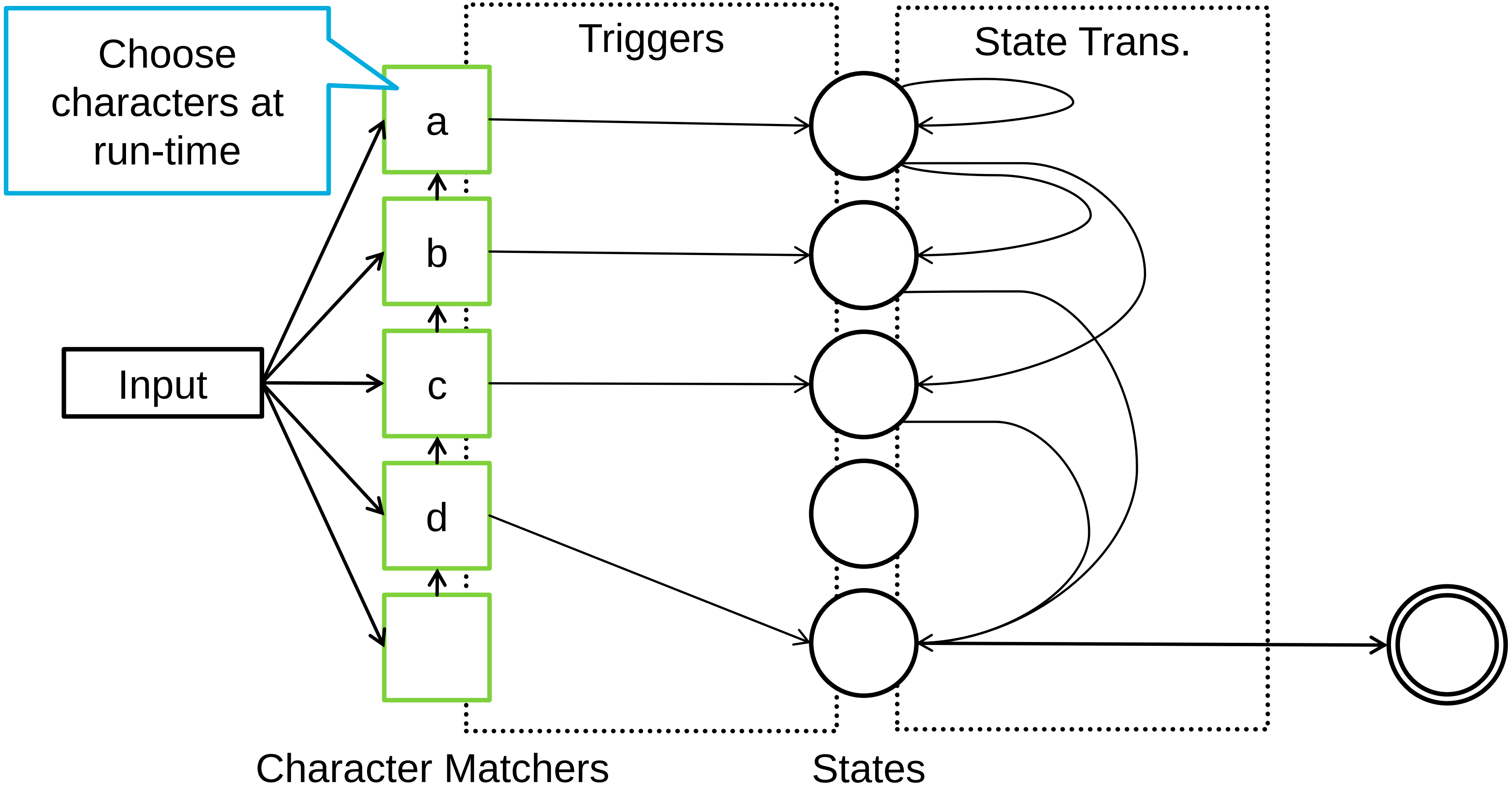


Towards a Parameterizable Design

- Deconstruct the NFA:
  - Characters
  - Transitions
  - States

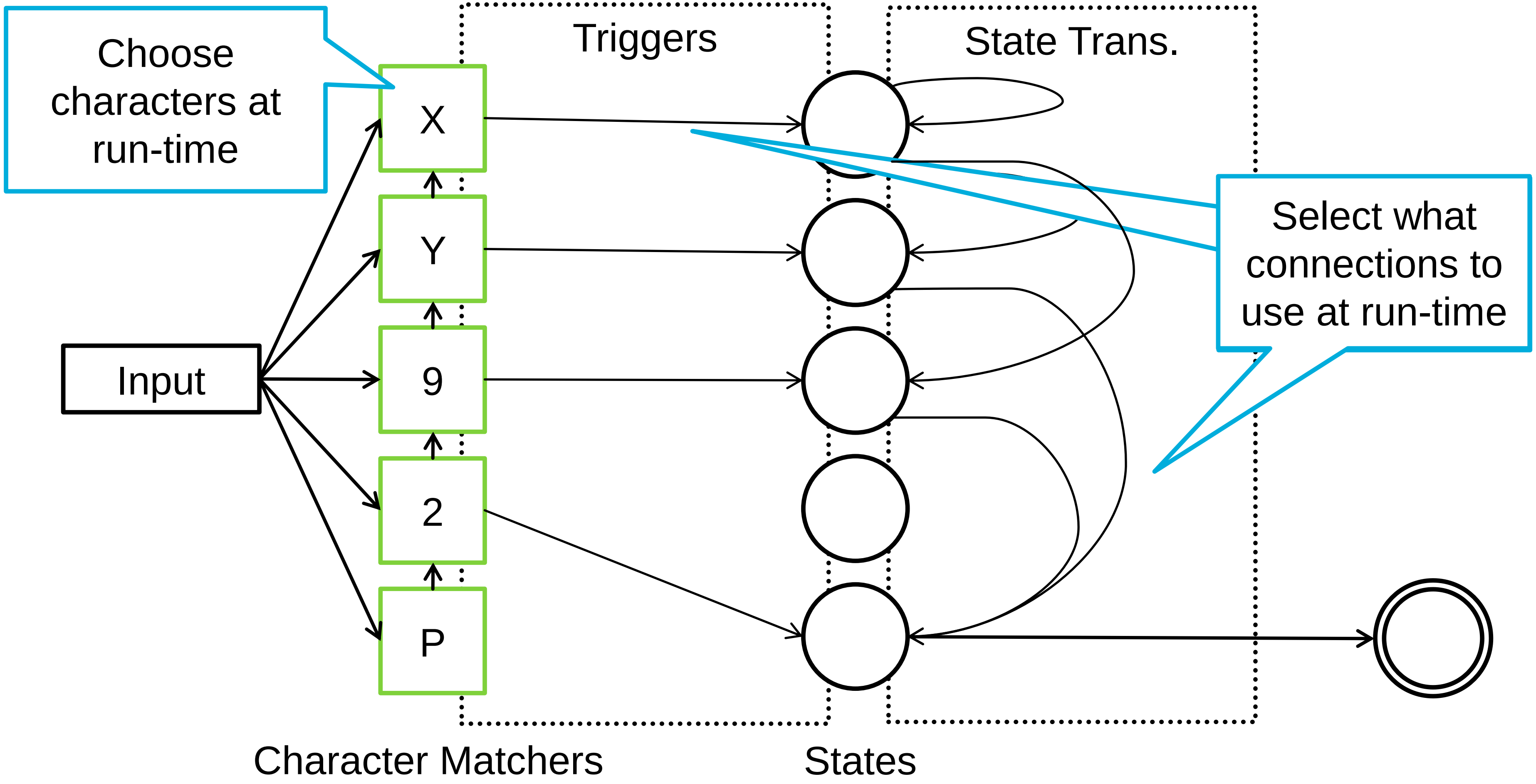


Pipelined and parameterized design

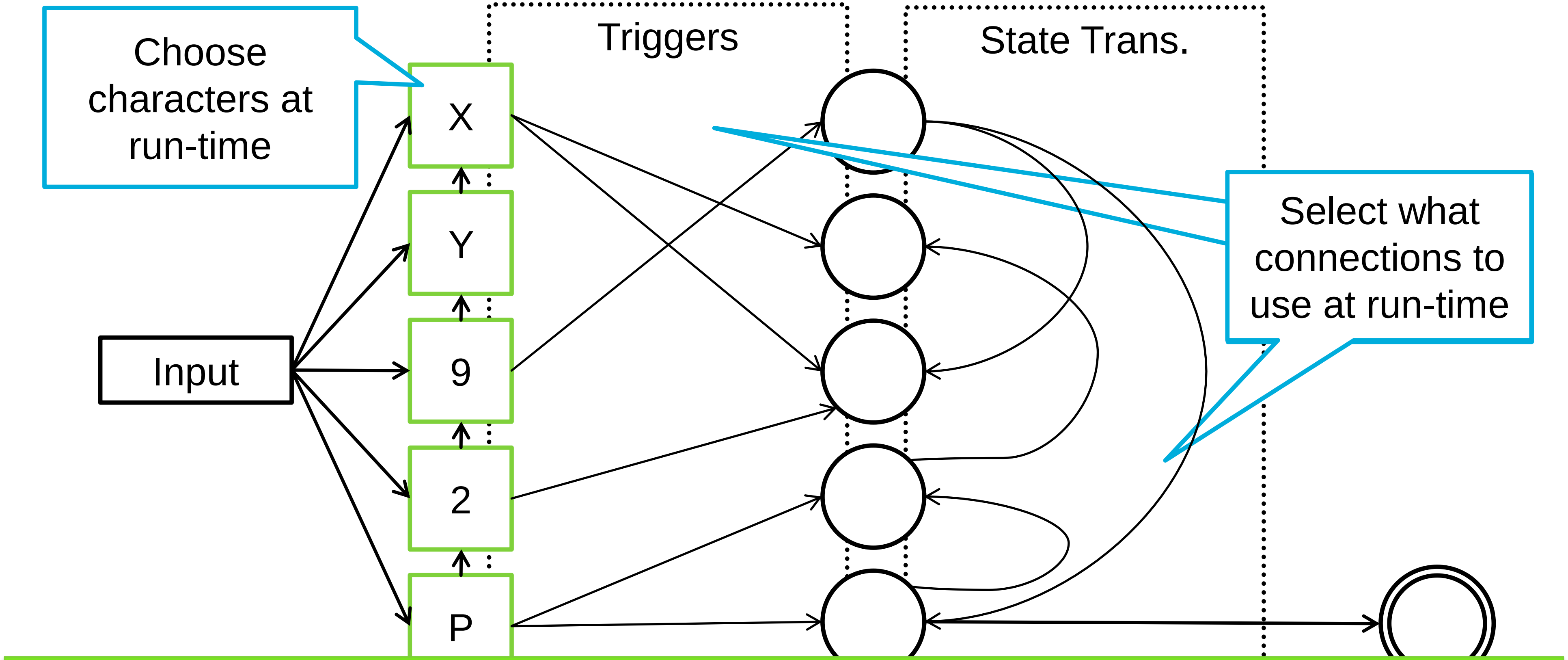


Choose characters at run-time

Pipelined and parameterized design


Choose characters at run-time
Triggers
State Trans.
X
Y
9
2
P
Input
Select what connections to use at run-time
Character Matchers
States

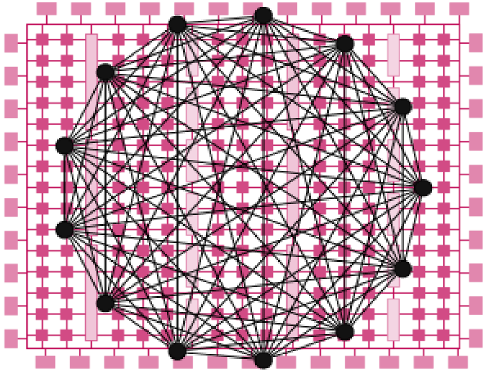
Pipelined and parameterized design


Choose characters at run-time
Triggers
State Trans.
X
Y
9
2
P
Input
Select what connections to use at run-time

Pipelined design turns the problem into bandwidth-bound regardless of complexity

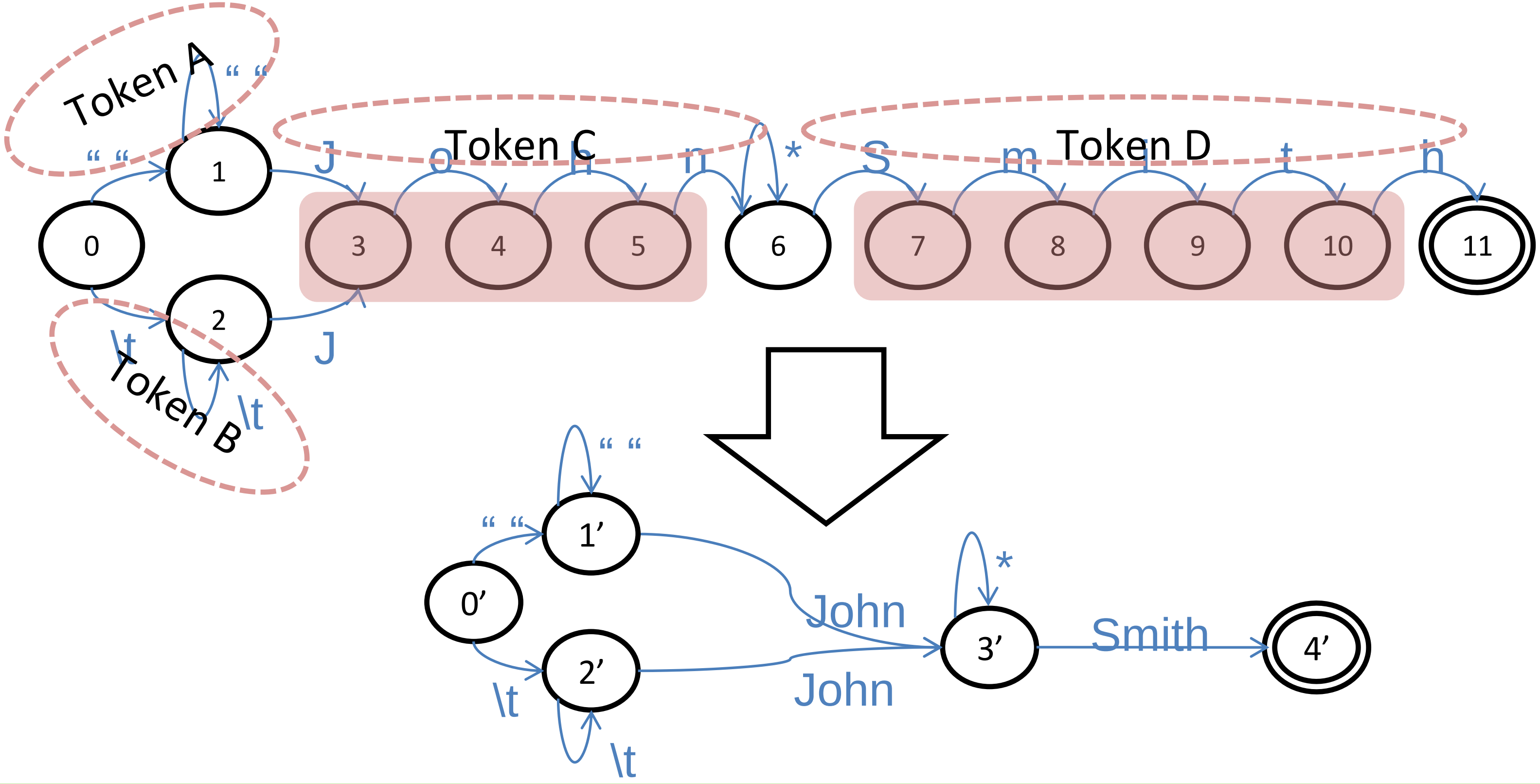
Skeleton of an NFA

- To be able to implement any expression we need the equivalent of a fully connected graph



- Resources limited on the FPGA
  - All “code” occupies space.
- Find a way to compress NFAs
  - Less states & Less character matchers?

Decoupling Characters from States

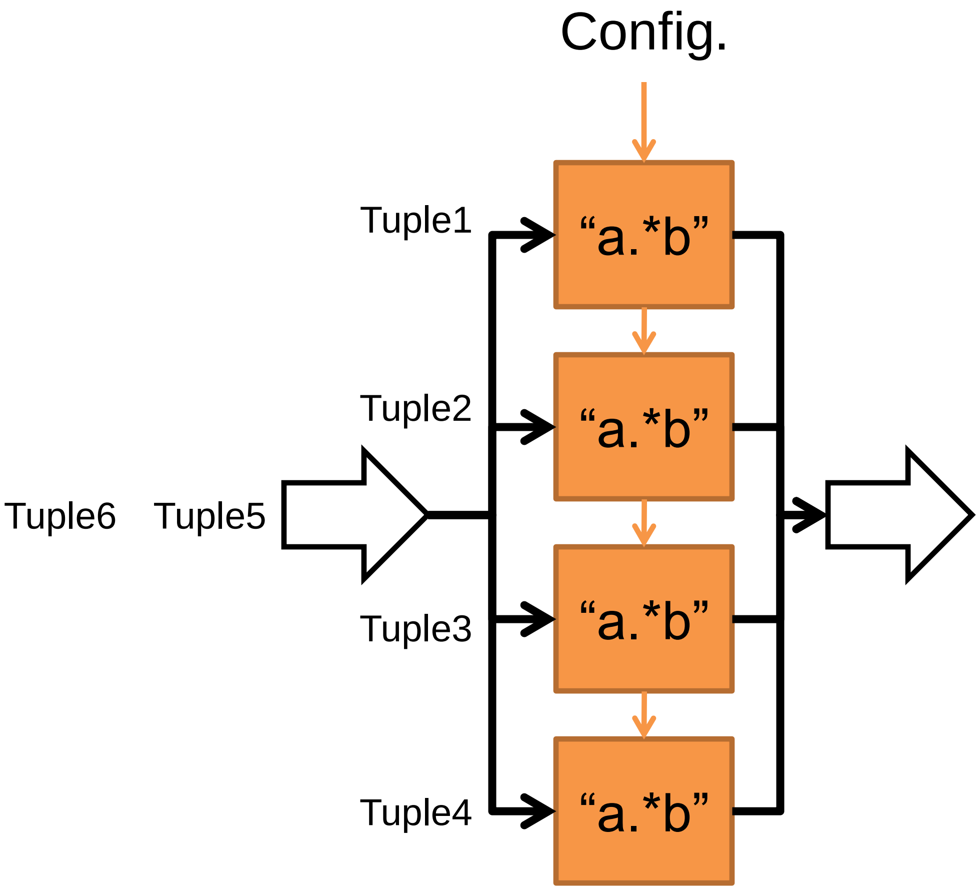


•Linear resource cost for "Tokens"

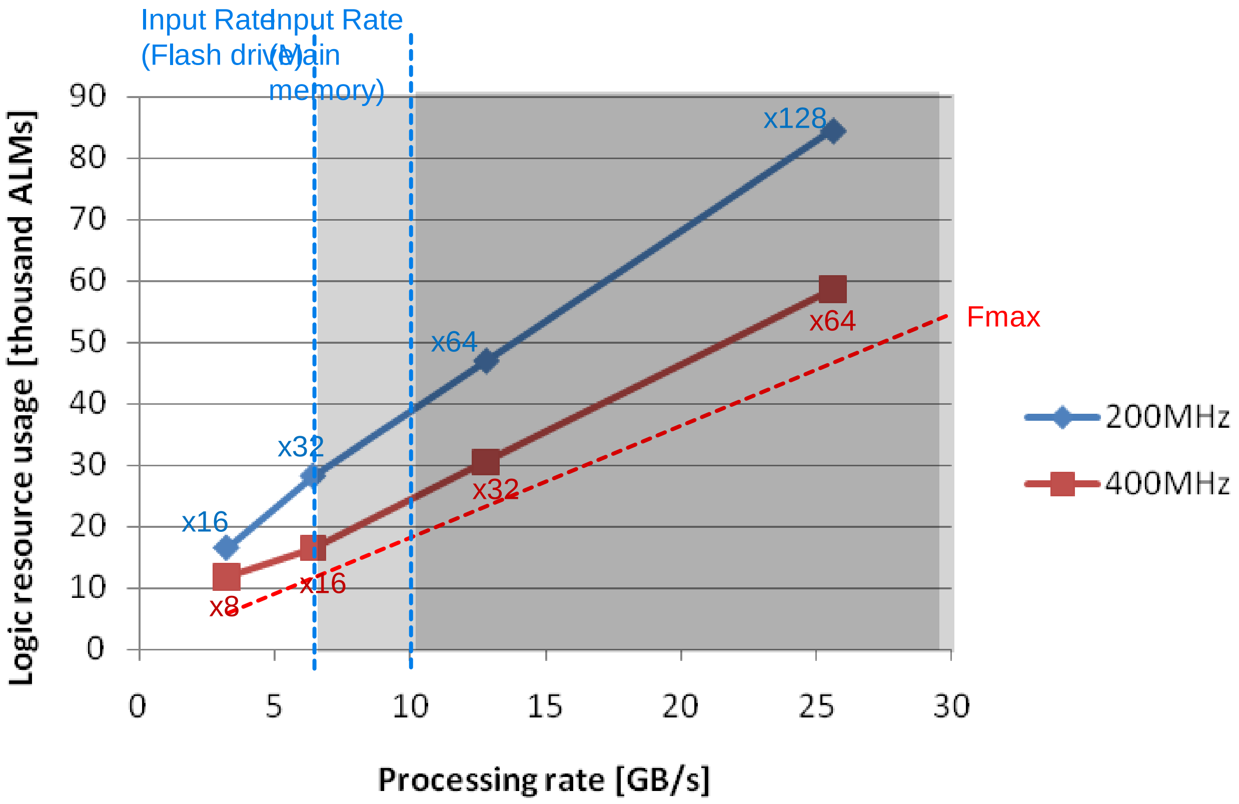
•Much smaller fully connected graph for "States"

Data parallel execution –a “Regex Processor”

- One Regex Engine can process 1B/cycle
- Split input across multiple units in parallel
  - No overhead in on-chip communication

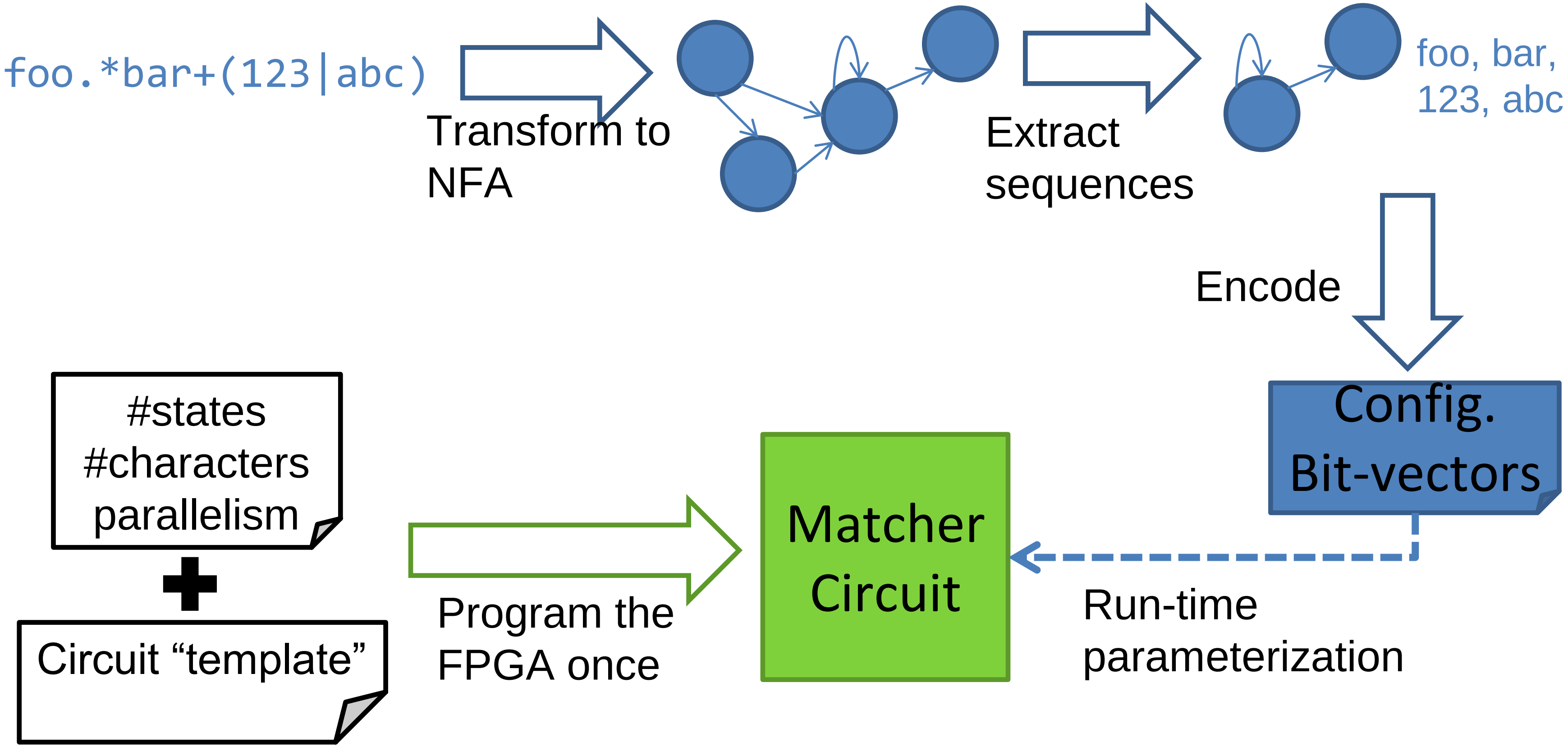


Scale the design to desired bandwidth



Can use chip-space to add other types of computation

From expression to execution



`foo.*bar+(123|abc)`

Transform to NFA

Extract sequences

foo, bar, 123, abc

Encode

Config. Bit-vectors

#states
#characters
parallelism

✚

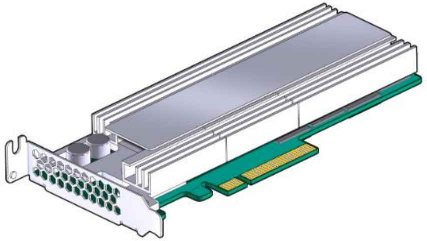
Circuit “template”

Program the FPGA once

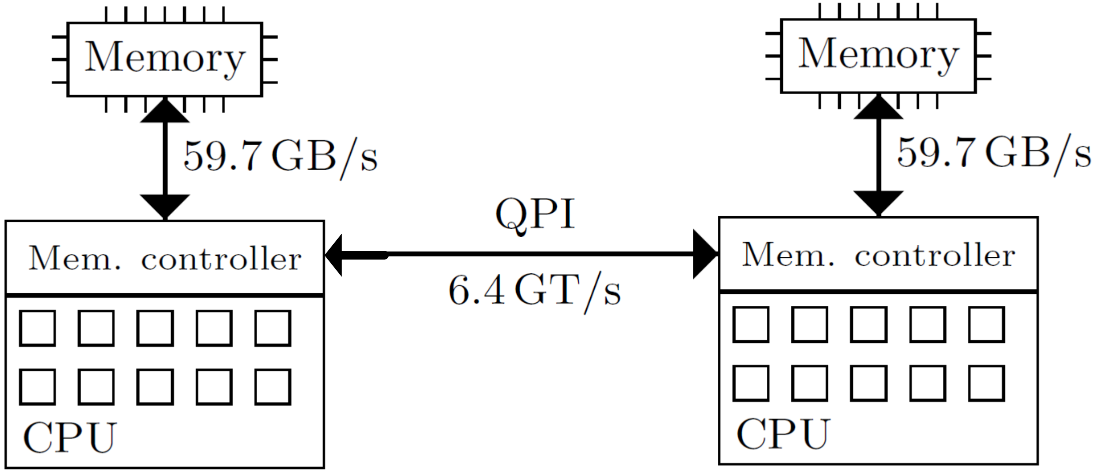
Matcher Circuit

Run-time parameterization

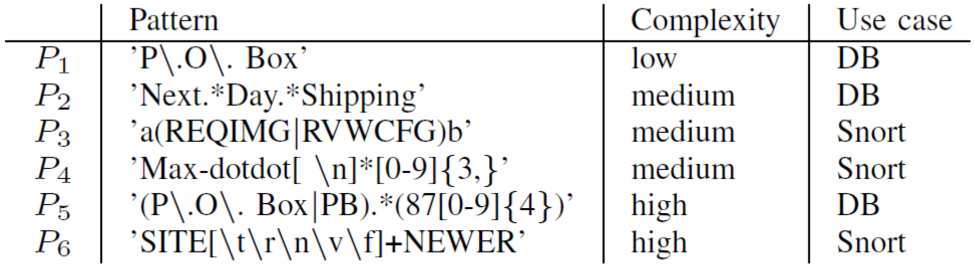
Deployment of FPGAs



- As an accelerator card – high latency, far from main memory
  - Amazon EC2, Microsoft Catapult (>1M devices deployed)
- As a co-processor – low latency, high bandwidth
  - Intel Xeon+FPGA machines



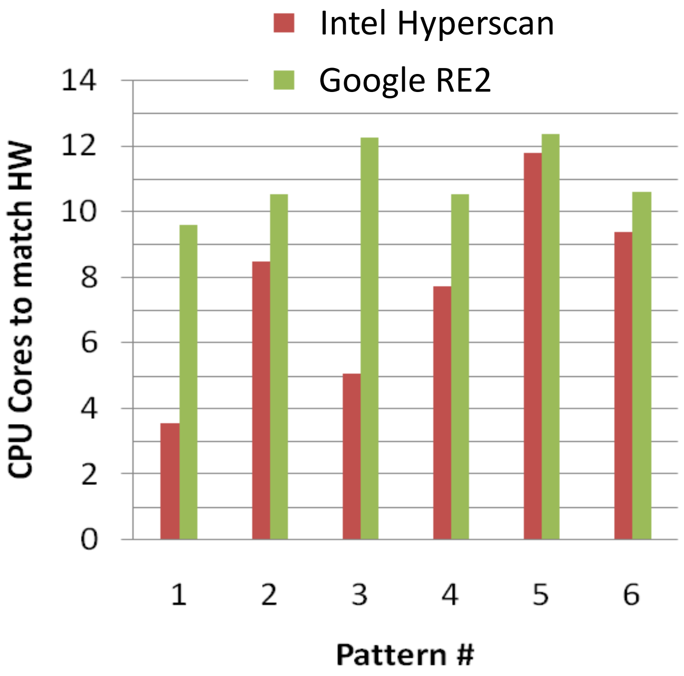
Changing the execution model pays off



	Pattern	Complexity	Use case
P_1	’P\.O\. Box’	low	DB
P_2	’Next.*Day.*Shipping’	medium	DB
P_3	’a(REQIMG|RVWCFG)b’	medium	Snort
P_4	’Max-dotdot[\n]*[0-9]{3,}’	medium	Snort
P_5	’(P\.O\. Box|PB).*(87[0-9]{4})’	high	DB
P_6	’SITE[\t\r\n\v\f]+NEWER’	high	Snort

HW throughput: 5GB/s for all patterns
(limited by communication to memory)

CPU used: Intel Xeon E5-2680v2
HW used: Intel Xeon+FPGA 1st Gen



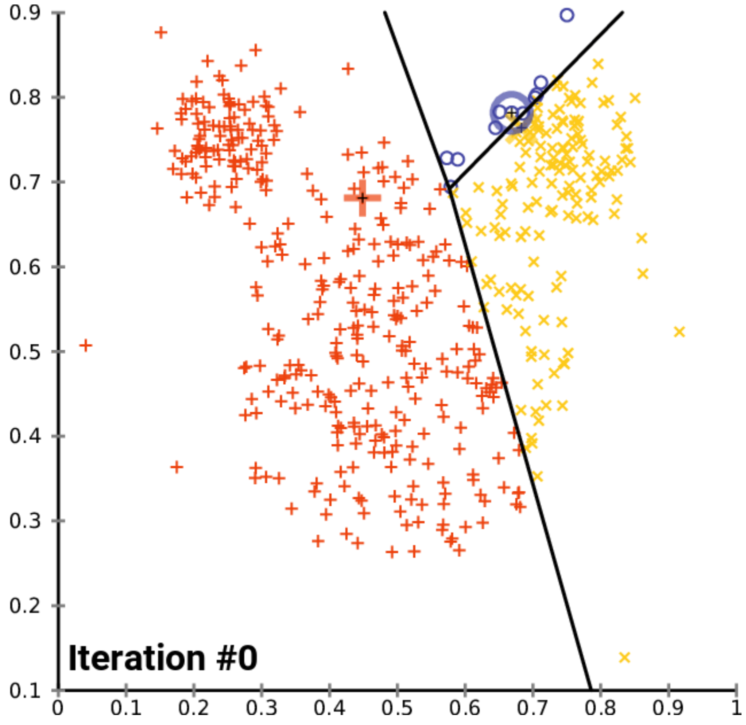
- Software can be competitive, but needs many cores
- Throughput not dependent on complexity
- Matching can be combined with other processing on FPGA

K-means – Algorithm

- Goal: partition unlabeled data into several clusters, where the number of clusters is the “k” in the k-means.

- Two steps in each iteration:
  - **Assignment**: assign data points to closet centroid according to distance metric
  - **Centroid update**: the centroids are re-calculated by averaging all the data points within each cluster

- Long process if the data set and number of iterations are large



Design – Execution Walk-Through

1 Receives K-Means parameters

2 Fetch the initial centroids and the data

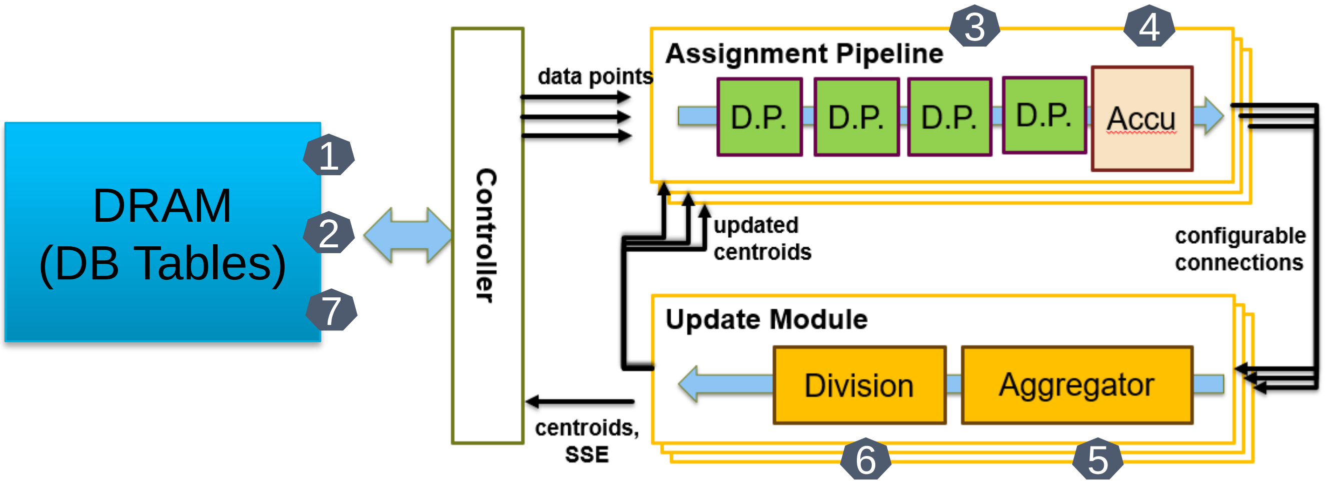
3 Calculates the distance between a data point and all the centroids and assign it to closest centroid

4 Accumulates data points per cluster and counts how many data points are assigned to each cluster

5 Collect partial results from each pipeline

6 Division for updating new centroid

7 Writes back the final results

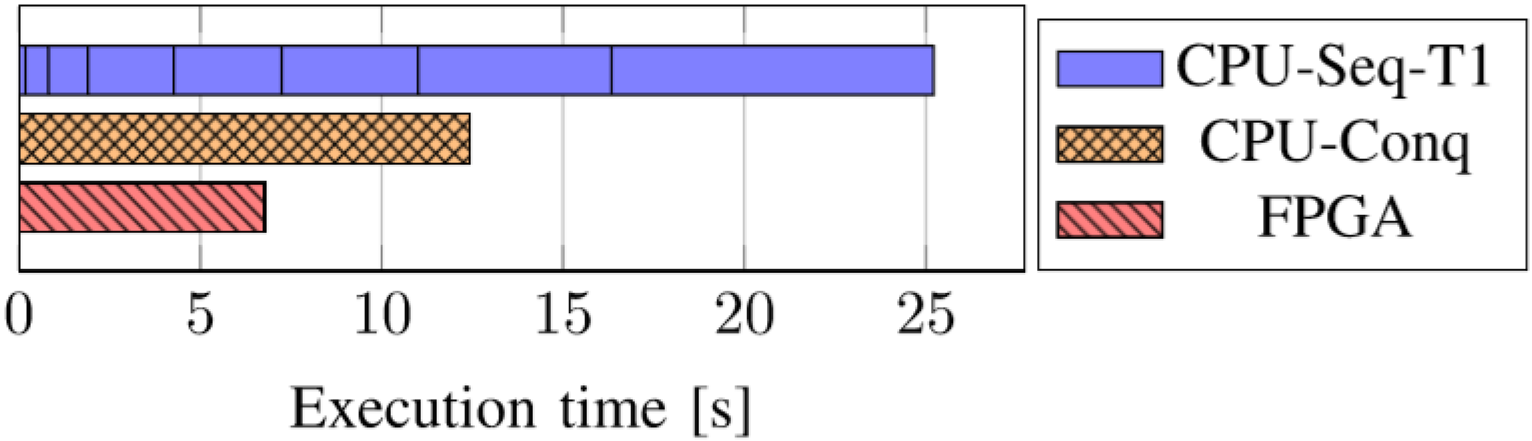


Zhenhao He, David Sidler, Zsolt István, Gustavo Alonso: *A Flexible K-Means Operator for Hybrid Databases*. FPL 2018

Uses of Parallelism

- K-Means algorithm
  - FPGA outperforms several cores of the CPU
  - Can use parallelism in two ways – cover more queries

Need to determine K
(Elbow method)



K is kr
Cent
knc

Fig. 5: Evaluation of multiple k, both sequentially and concurrently on software and concurrently in hardware

Closing Remarks

- Specialized hardware allows breaking traditional tradeoffs
  - Convert from compute-bound to bandwidth-bound
  - Adding a “spatial element” to the design tradeoffs

- Looking ahead: Datacenters are becoming more heterogenous
  - Need to think about how we split functionality across processor types
  - Programming non-CPU devices

We’re hiring at IMDEA Software! If you are looking for an internship or PhD position, contact me!

`zsolt.istvan@imdea.org`