Parallelizable Algorithm for Hyperspectral Biometrics

Team #1:

Christopher Baker, Timothée Bouhour, and Akshay Malik

ECE 445 Senior Design
December 10th, 2012

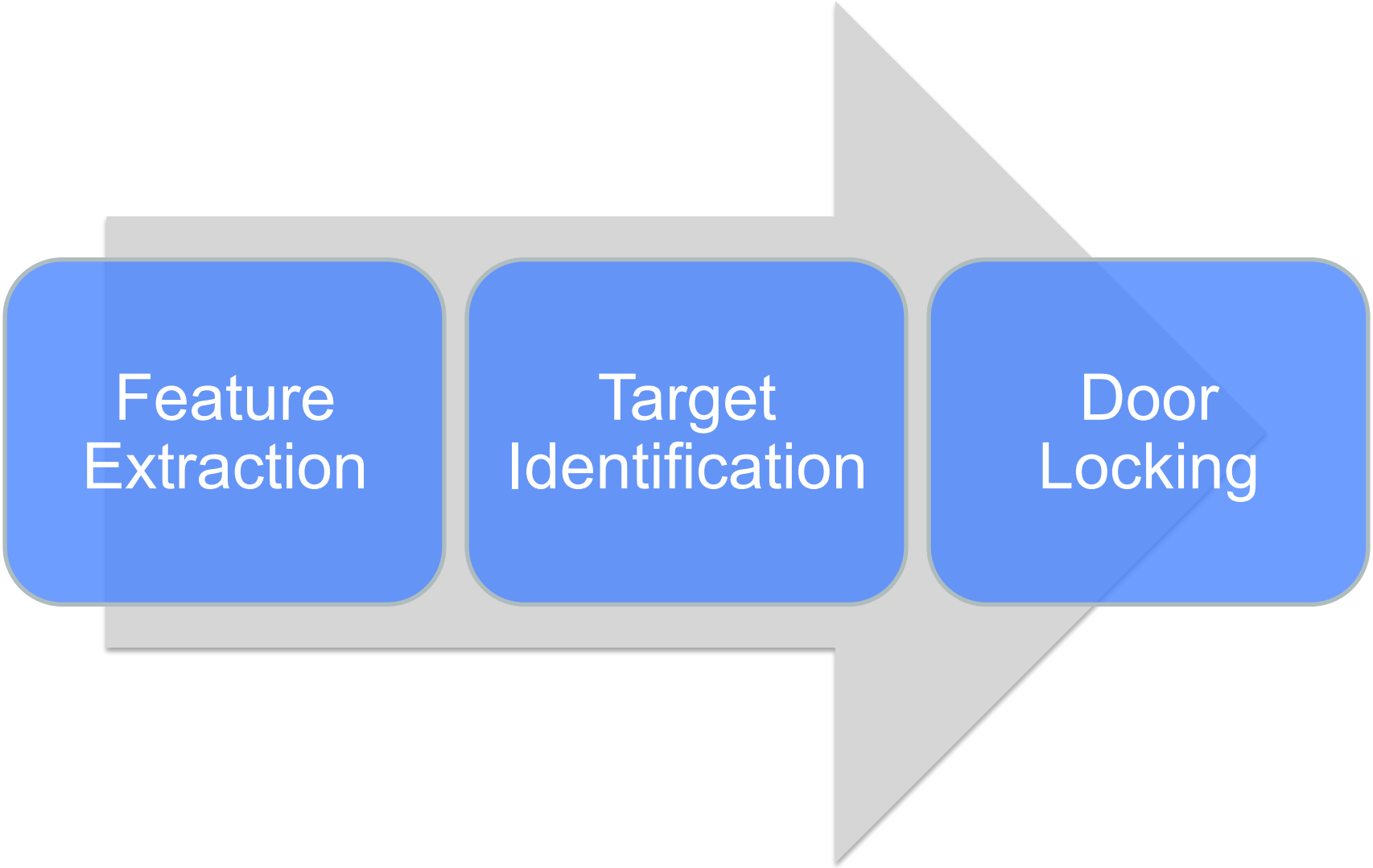
Introduction

- Hyperspectral imaging provides useful biometric information that can be used to uniquely identify a person
- This is useful in applications such as facial recognition for security systems or any other application where a person needs to be identified

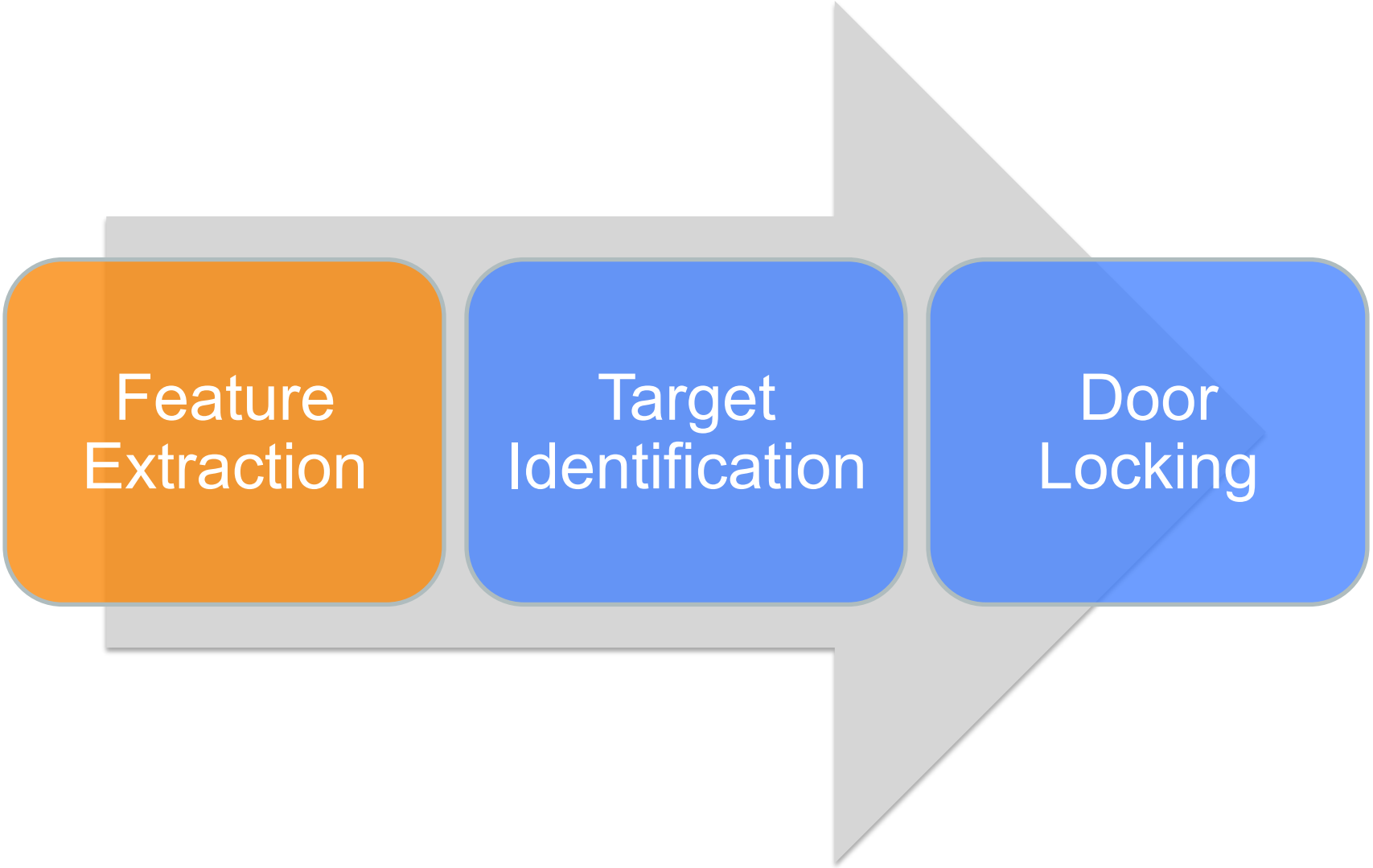
Objectives

- Demonstrate the validity of hyperspectral biometrics as part of an improved security system
  - Motivated by prevention of global terrorism and crime
- Show the benefits of GPU processing in dealing with hyperspectral data

Project Features



Feature Extraction Algorithm

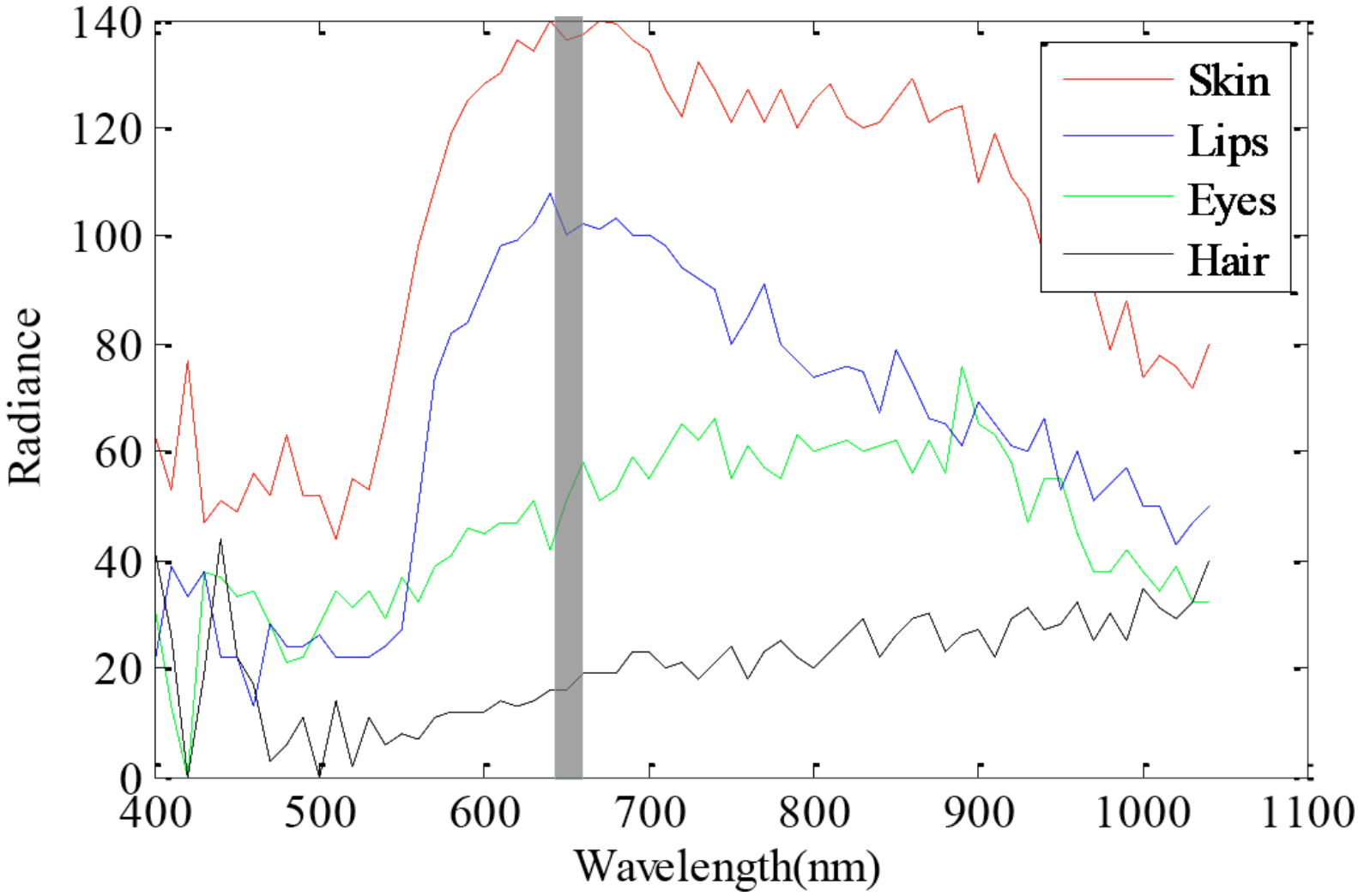


Requirements

- Data must be hyperspectral
- Features are extracted correctly
  - Within a 10% image dimension range
- Databases are sufficiently and correctly populated

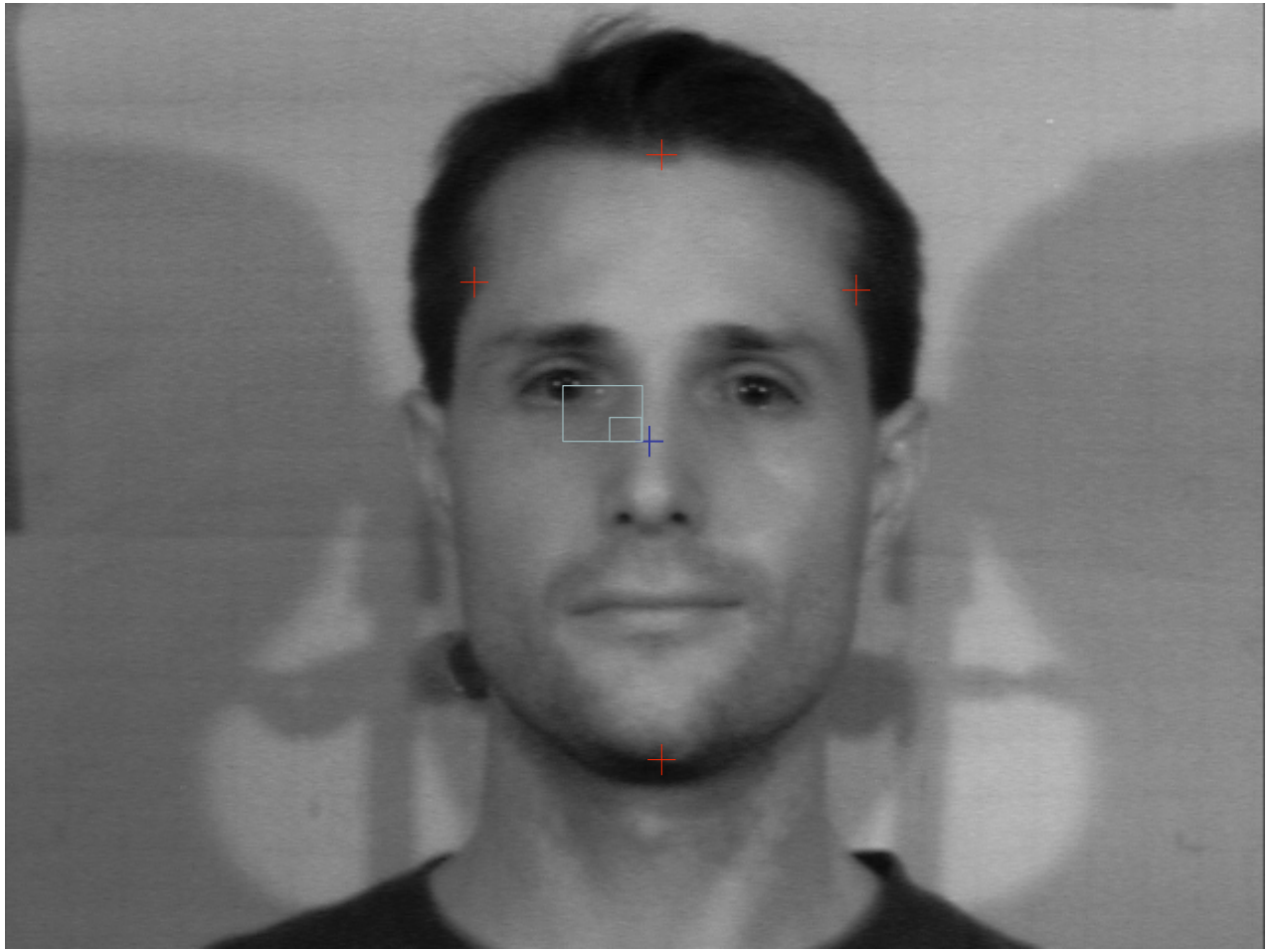
Design decisions

Choice of bands for features



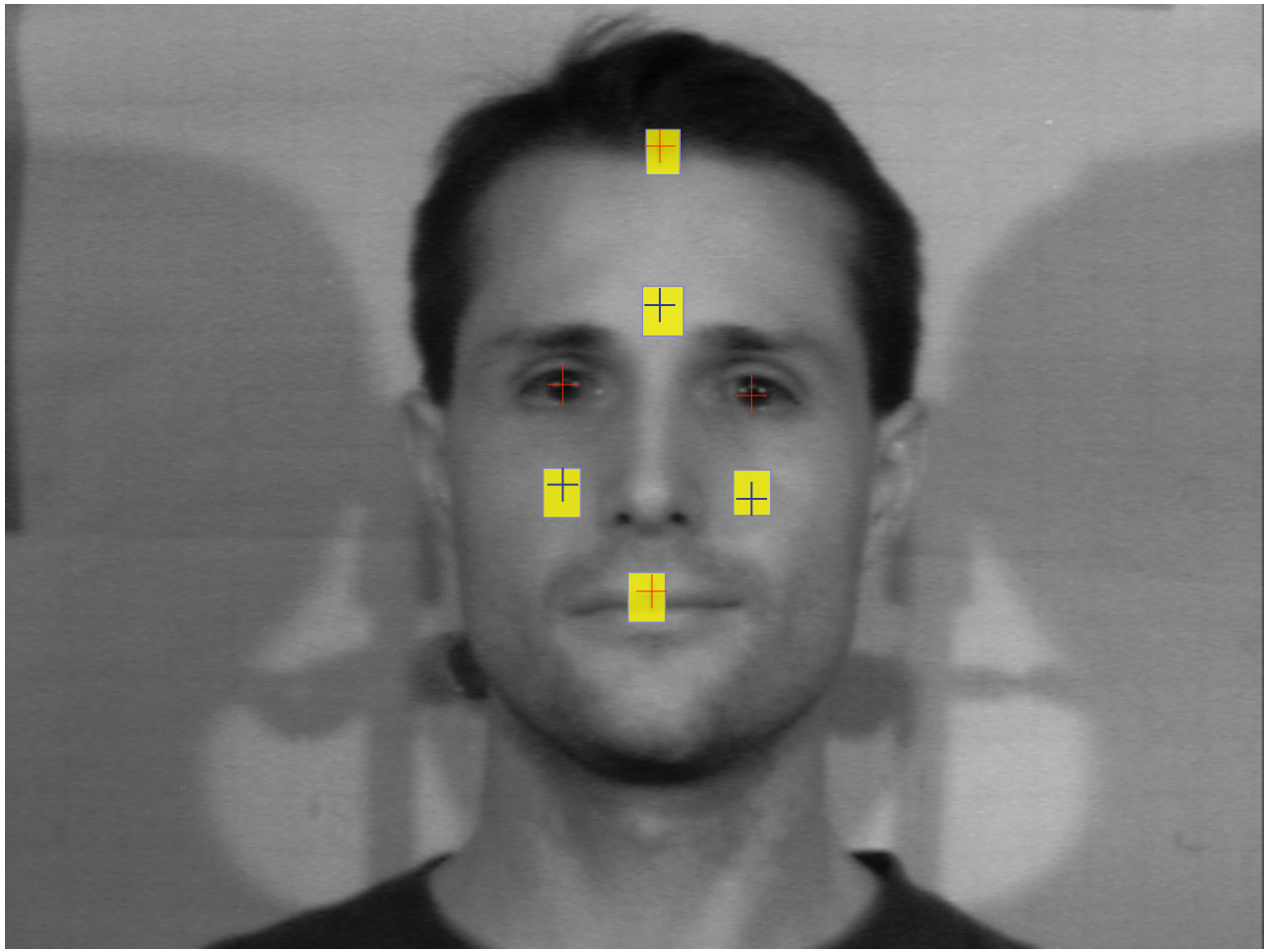
Algorithm Flowchart

- Identify features
- Extract positions
- Pick N*N pixel area
- Store M vectors per pixel



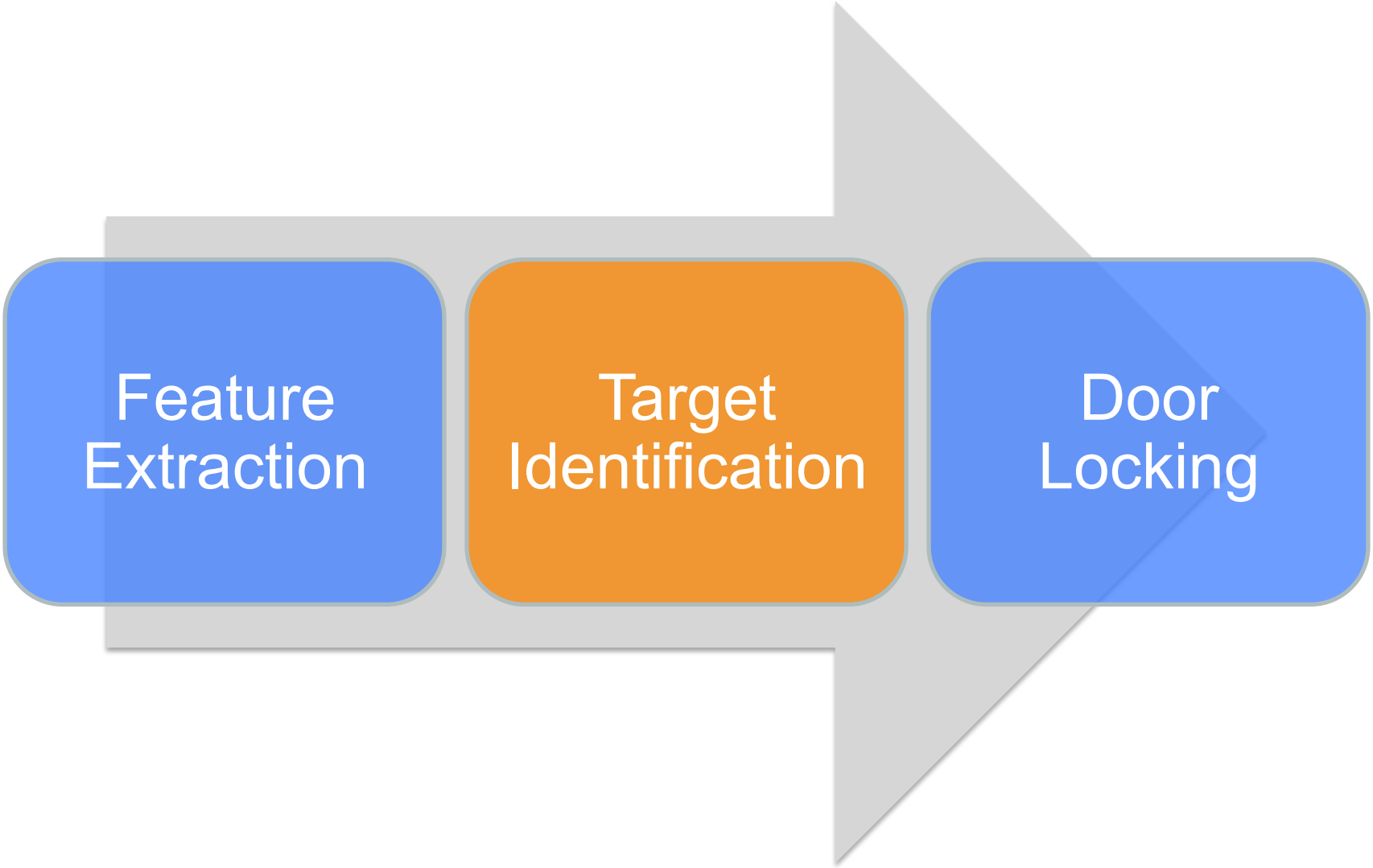
Verification

Left cheek	x	285
	y	248
Right Cheek	x	382
	y	249
Lips	x	331
	y	300
Forehead	x	332
	y	248



- Within 2% of actual position in 34/36 subjects
- Databases populated with 9*9 pixel regions around the cheeks, lips, forehead, and hair

Identification Algorithm

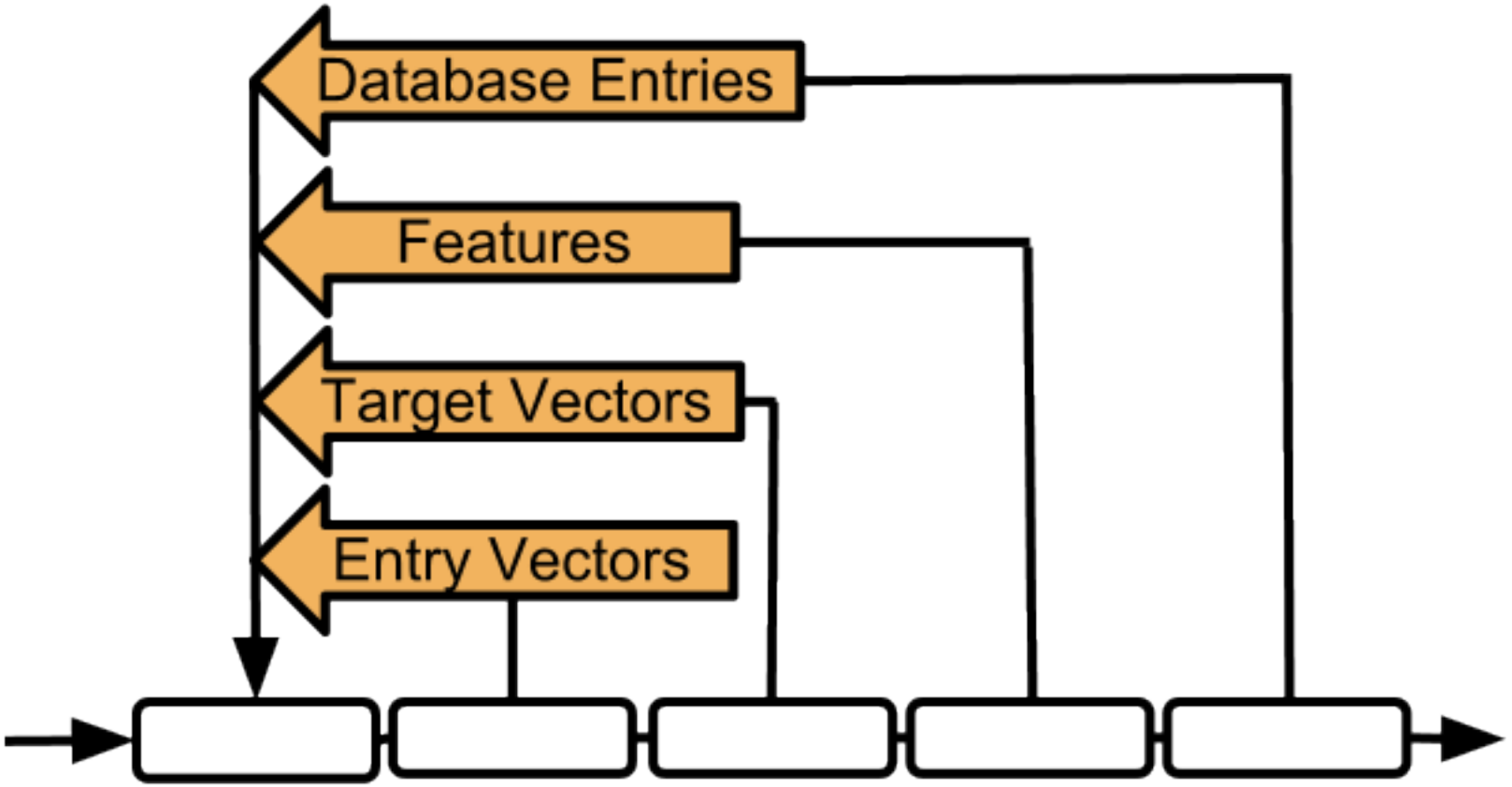


Requirements

- Algorithm is capable of identifying a person in the database
- Algorithm should output top 5 matches in order
- Target should be in 5 top matches 50% of the time

ID Algorithm Development

Initially designed & implemented serial algorithm for benchmarking



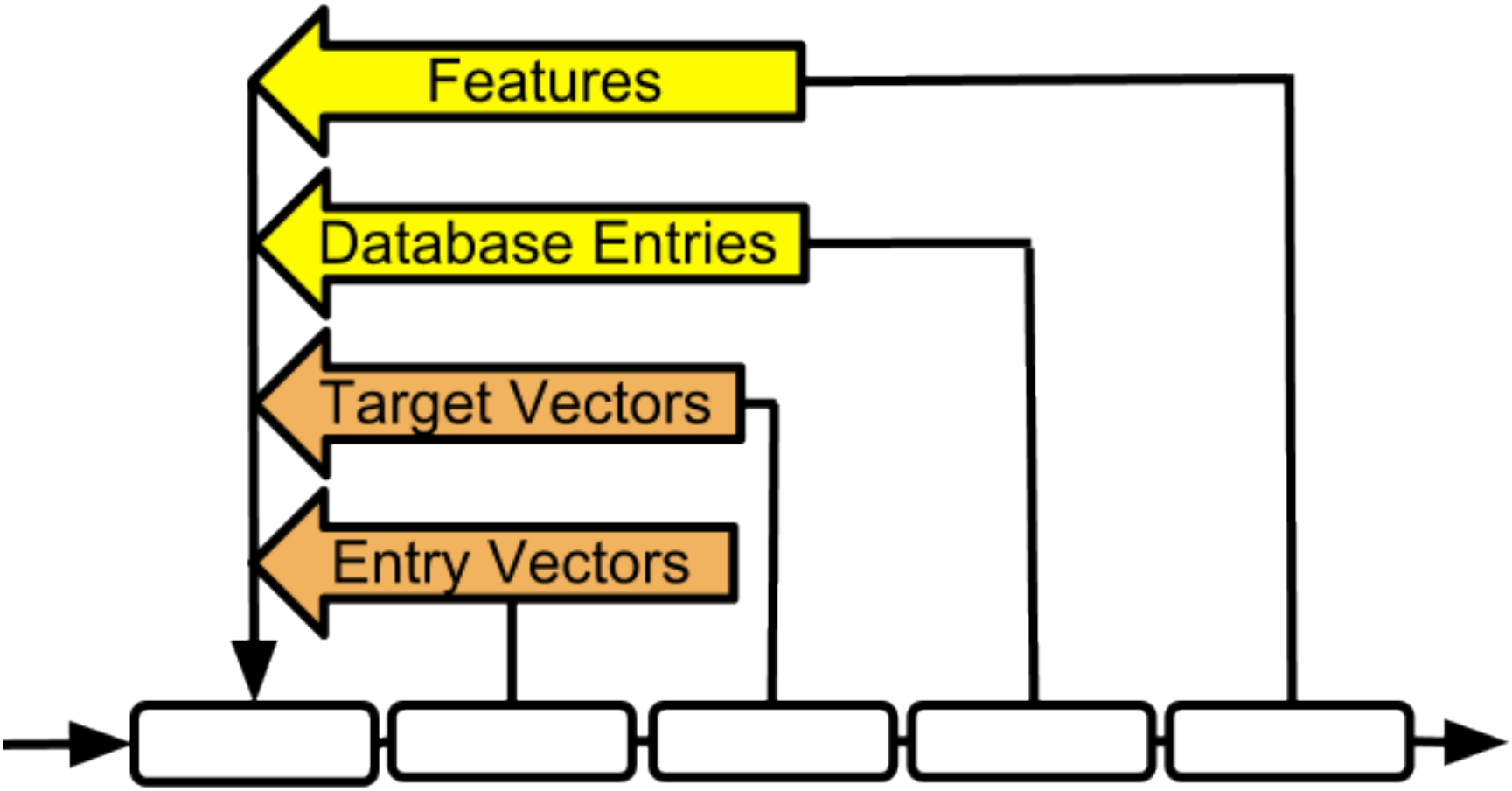
ID Algorithm Development

- Complete algorithm redesign from serial to interface with GPU hardware
  - Not originally planned
  - Exploit thread parallelism
  - Optimization for hardware memory constraints
  - Exploit hardware operation concurrency

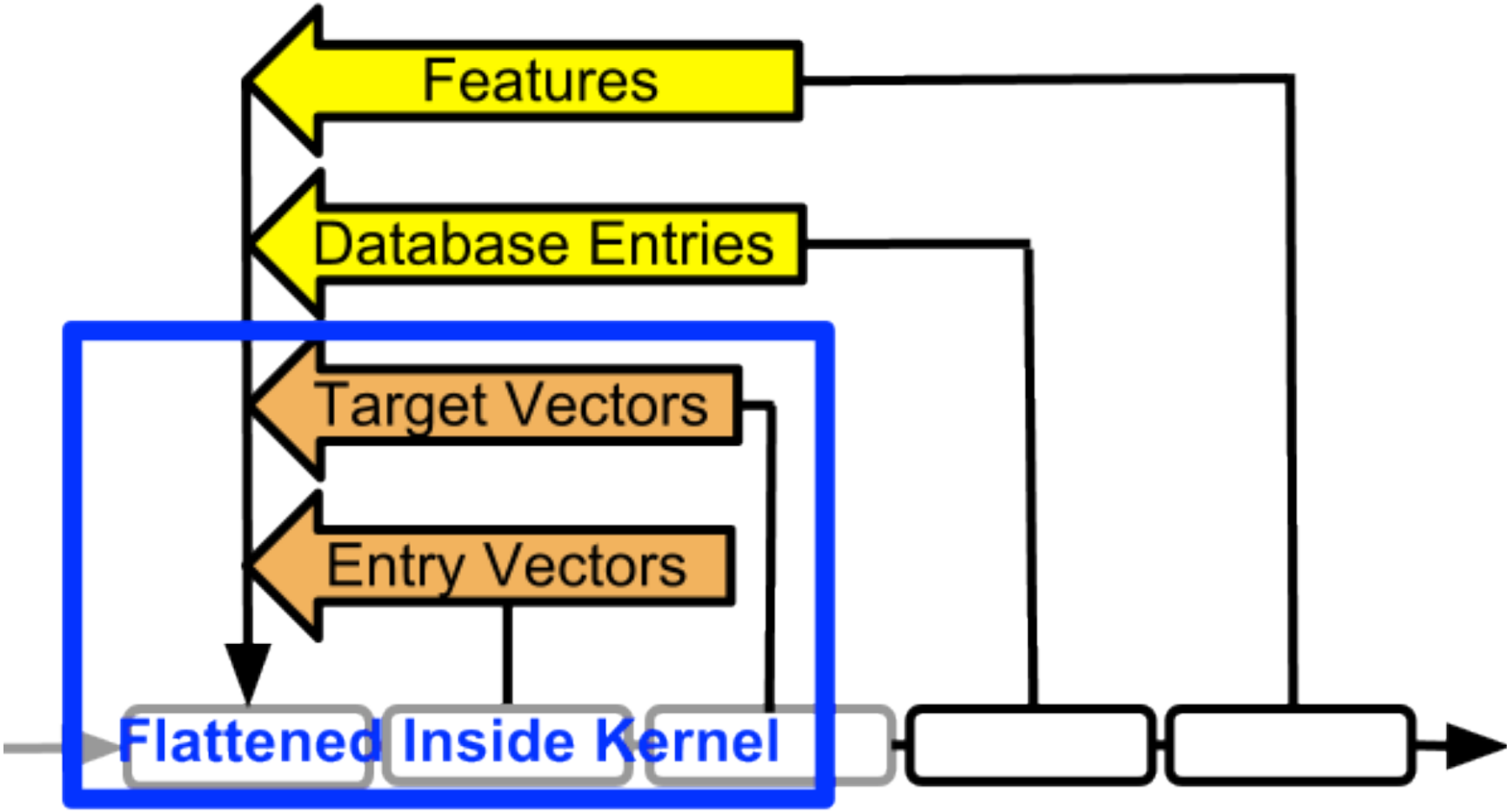
Final Algorithm Design

- Flip the constant features loop to the outside
  - Achieve better data granularity
  - Serialization is now of constant magnitude
- Processing now done in multiple launches of CUDA kernels
  - Each kernel processes a single feature for a group of database entries of optimized size

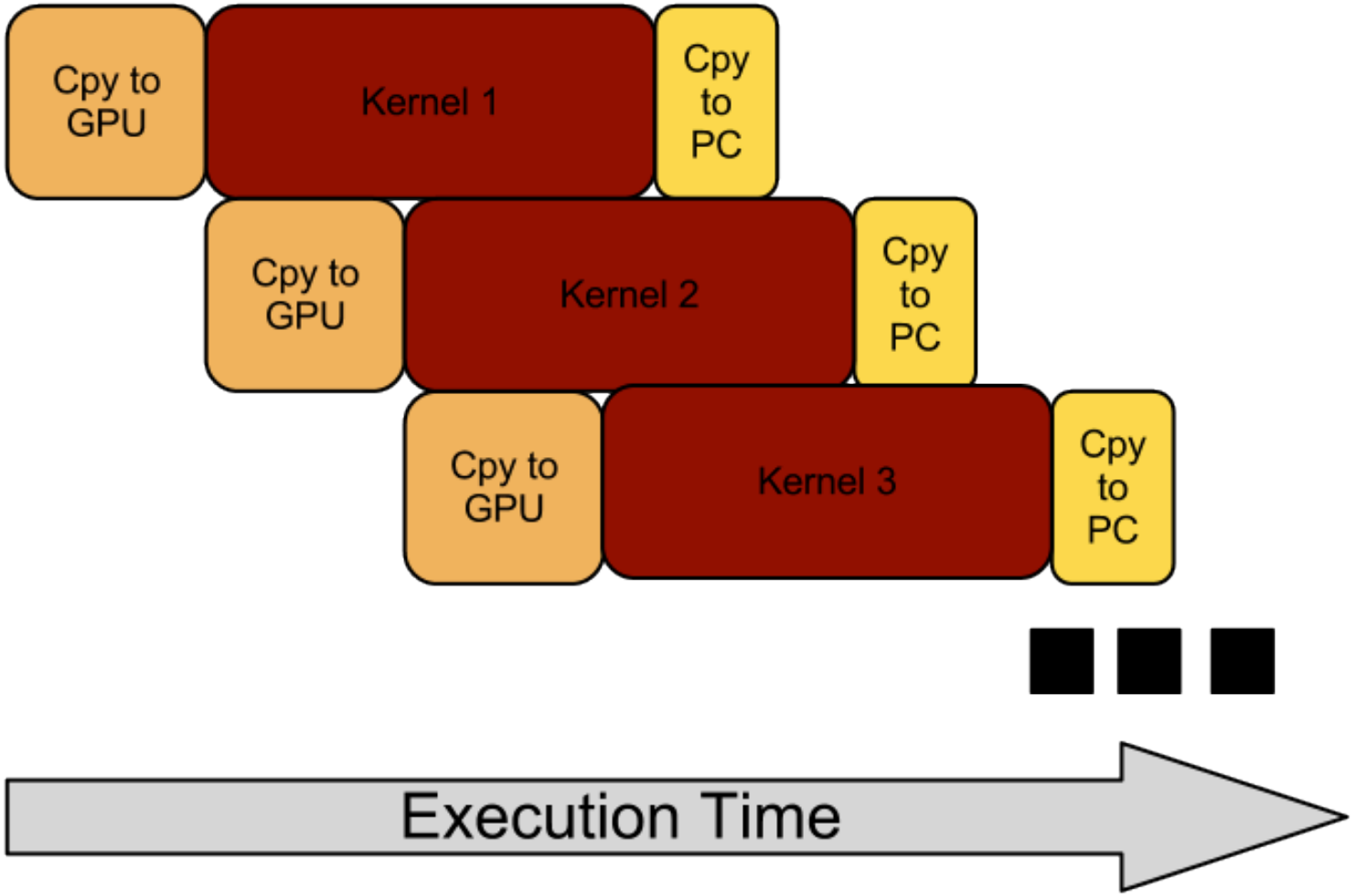
Final Algorithm Design



Final Algorithm Design



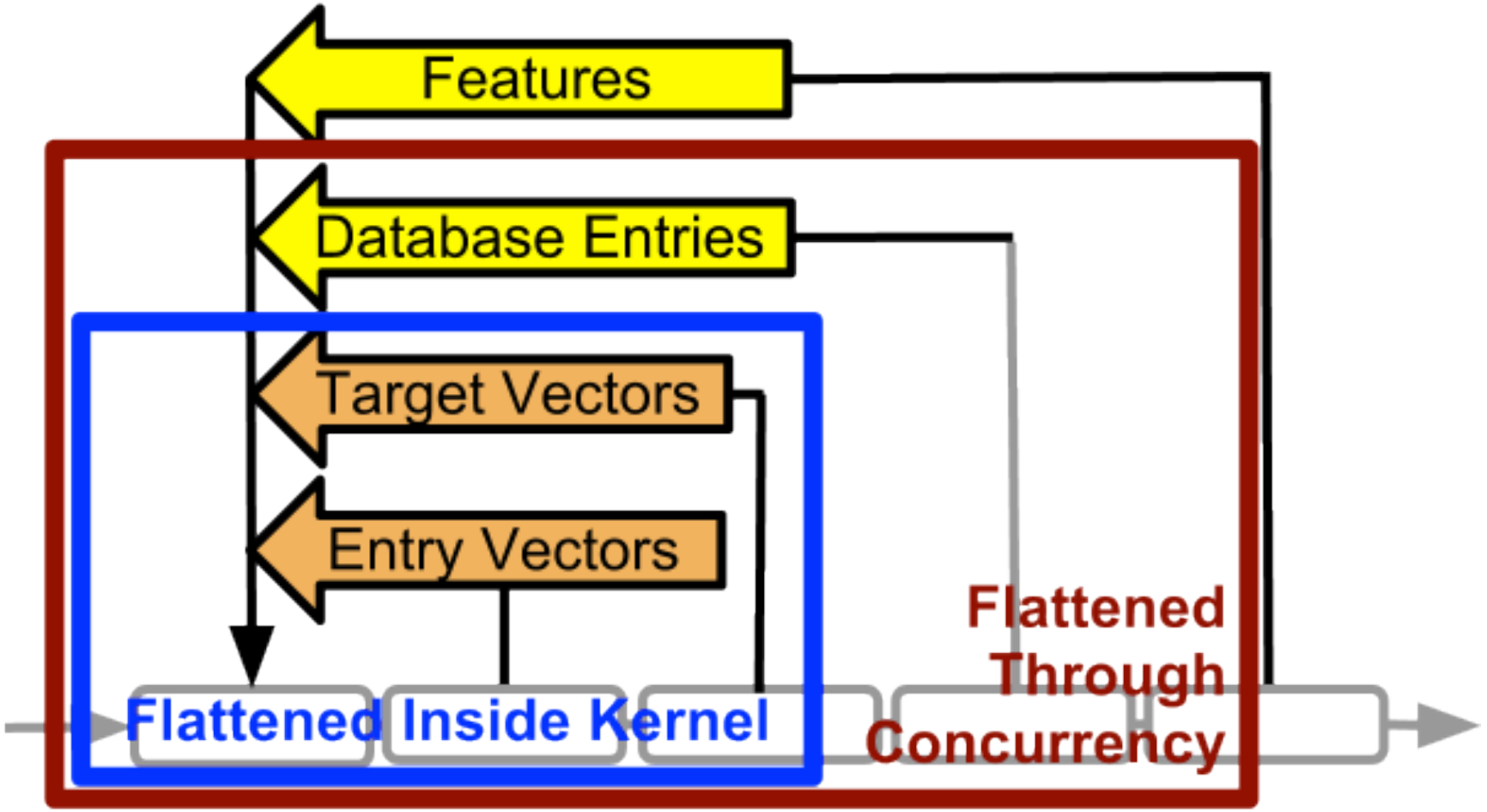
Final Algorithm Design - Concurrency



Final Algorithm Design – Memory Optimizations

- 5 buffers from host manually mapped into device memory space for input speed
- 5 buffers from host manually mapped into device memory space for outputs
- Device constant memory sharing for target

Final Algorithm Design – Fully Parallelized



Verifications

Sample of Program output

```
**************************RESULTS**************************
1: 00100000000000000000000000000000000000000000000000000000
dist: 8.711265
access level : 3
2: 13600000000000000000000000000000000000000000000000000000
dist: 9.673415
access level : 1
3: 07300000000000000000000000000000000000000000000000000000
dist: 9.974702
access level : 1
4: 22600000000000000000000000000000000000000000000000000000
dist: 10.944902
access level : 2
5: 12700000000000000000000000000000000000000000000000000000
dist: 11.414450
access level : 0
************************************************************
```

Verifications

Performance on database of 234 subjects

- 26 entries were searched for
- Other entries were randomly generated

Number of searches:	26
Number in top 10	22
Number in top 5	17
Accuracy (top 10)	**84.62%**
Accuracy (top 5)	**65.38%**
Accuracy (top 1)	**19.23%**

Performance Benchmarking

- Took timing data to find our speed-up
  - Using "time" command from command line
  - Generated "junk" databases of larger size

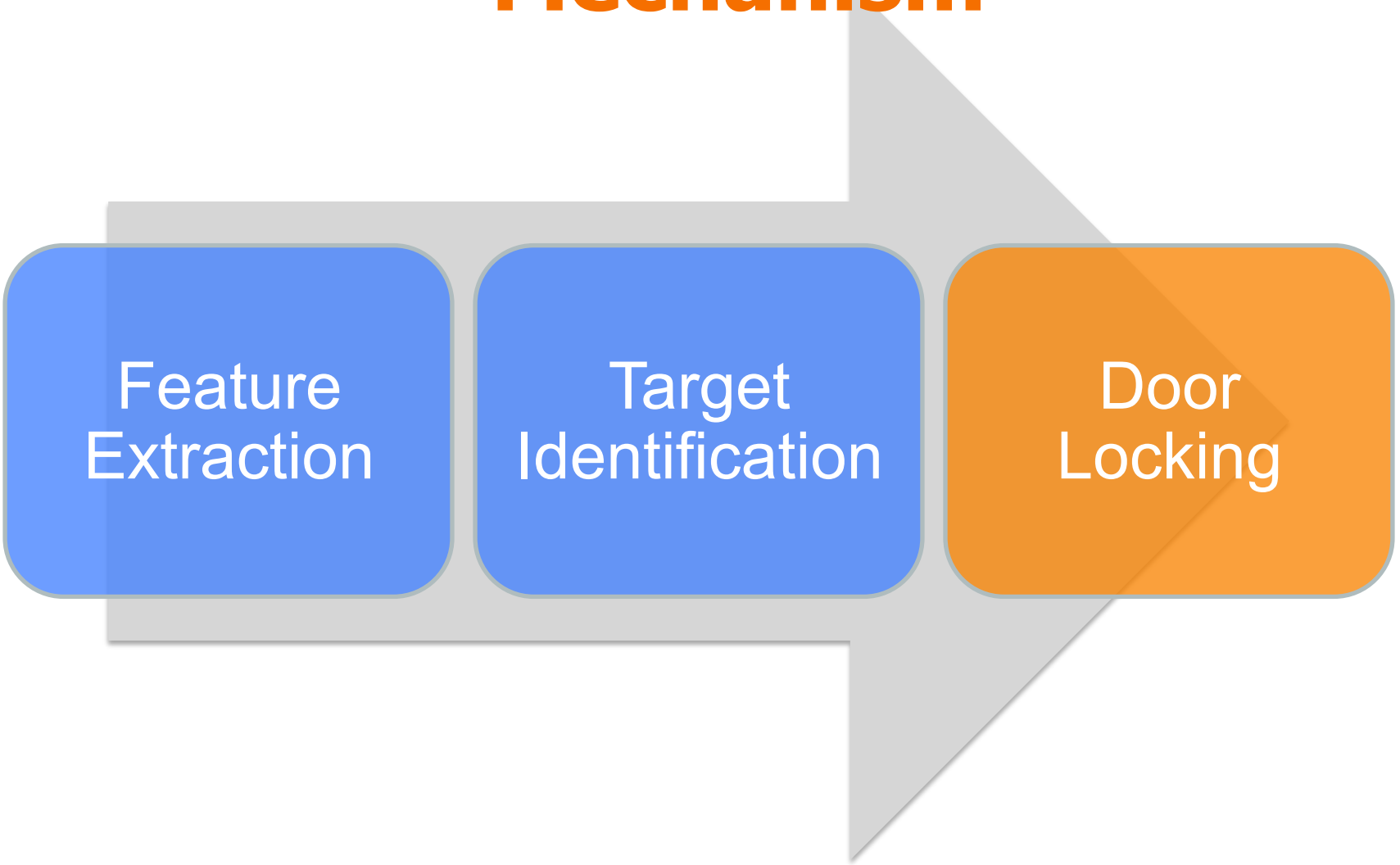
```
real    0m5.585s
user    0m5.521s
sys     0m0.059s
```

Performance Benchmarking

Results of our speed tests:

Database Size	Avg. Serial Runtime (ms)	Avg. CUDA Runtime (ms)	Avg. Speedup
26	325.2	124.34	2.62
234	1066.2	321.80	3.31
1000	5588	1197.46	4.67

Wireless Door Locking Mechanism



Requirements

- Receive and act on signals from the computer
- Engage motors when signals are received
- Rotate the motors the correct amount

Design decisions

- Arduino hosts a web server to receive data
- Interface wirelessly from a computer
- Output pins of the Arduino provide digital highs and lows
- Use 4 H-bridges to control the motors

Design

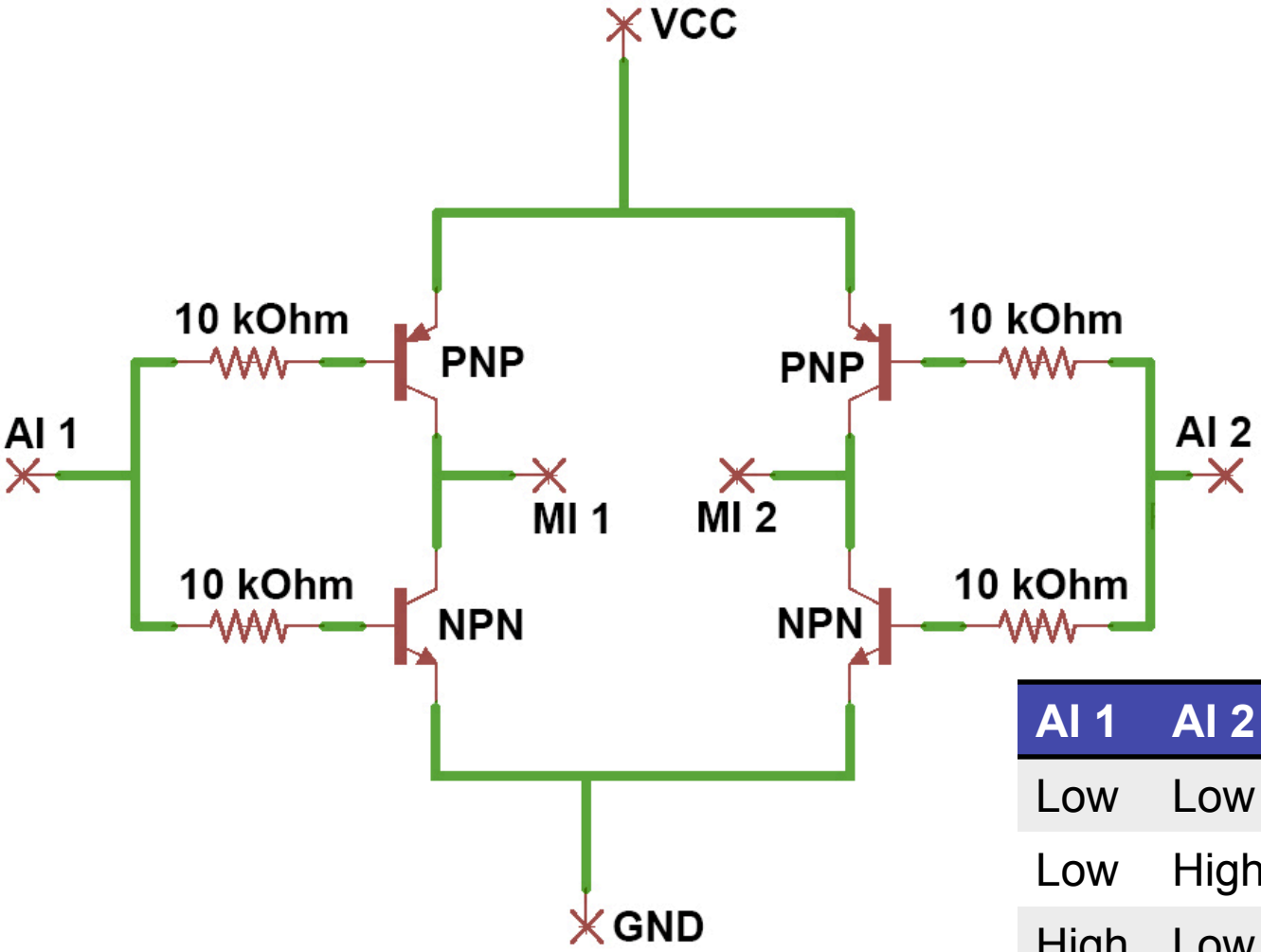
Wireless

Wireless

Digital Outputs (5V, 30mA)

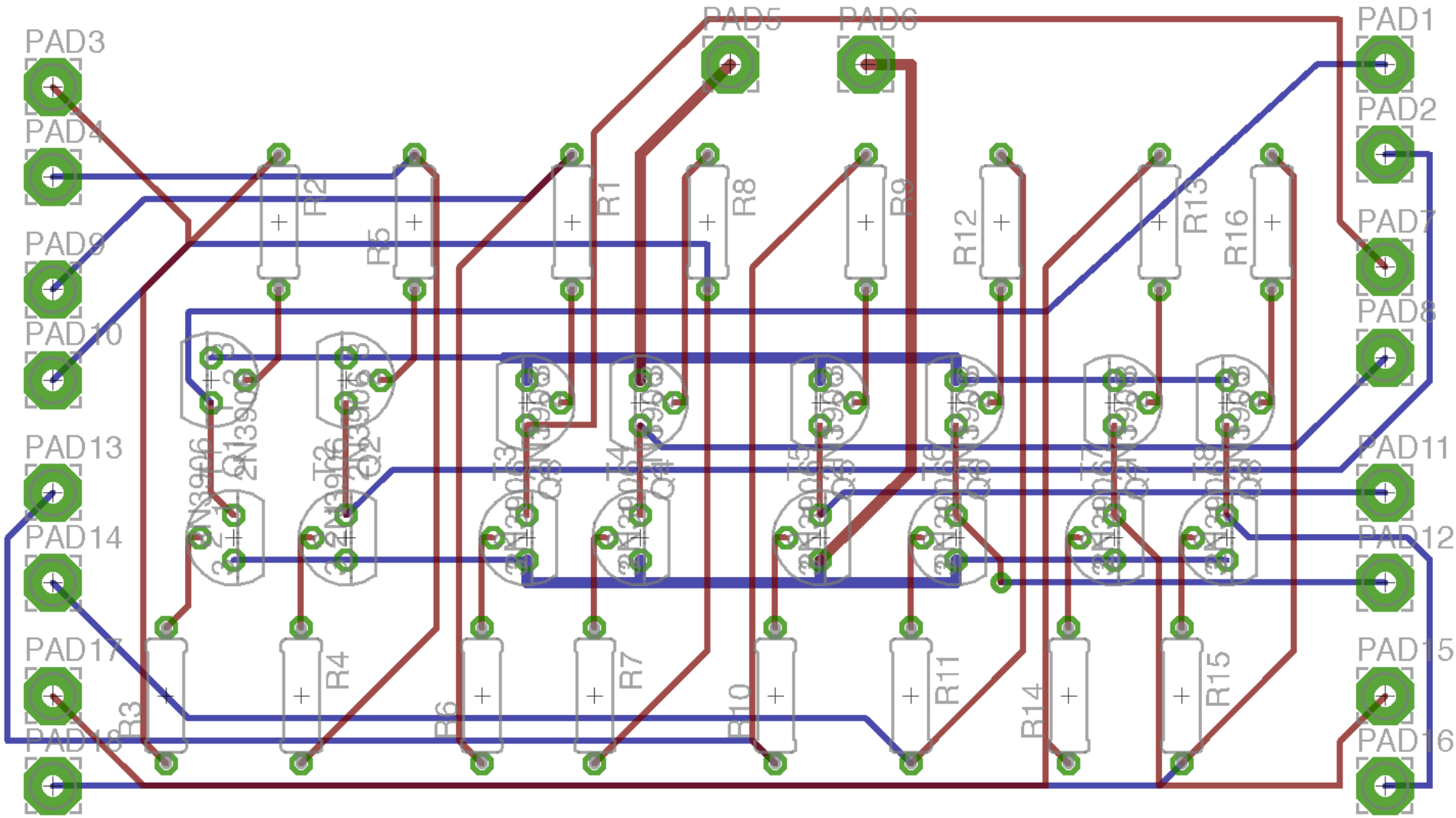
Motor Outputs (5V, 200mA)

H-bridge Design



AI 1	AI 2	Motor control
Low	Low	None
Low	High	Reverse
High	Low	Forward
High	High	None

PCB Design


PAD1
PAD2
PAD3
PAD4
PAD5
PAD6
PAD7
PAD8
PAD9
PAD10
PAD11
PAD12
PAD13
PAD14
PAD15
PAD16
PAD17
PAD18
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
2N3906

Verifications

Correct packet sent without corruption	70-80% of the time
Correct packet interpretation	100% of the time
Arduino pin outputs	4.8V-5.1V at 30mA
H-bridge operation	All motors run with 5V input
Motor rotation	3 second forward, 1 second pause, 3 second reverse 100% of the time

Summary

- Feature Extraction Algorithm
  - Fully meets functionality requirements
  - Could move to more real world data
  - Could improve feature region selection

Summary

- Identification Algorithm
  - Fully met accuracy requirements
  - Would like to improve accuracy still
  - Achieved good speed up

Summary

- Wireless Door Locking Mechanism
  - Meets most requirements
  - Data packet corruption
  - Would like to add automatic packet correction

Recommendations for Further Work

- Improve feature extraction algorithm
- Complimentary sensor development
- Do more in depth analysis on feature choice
- Develop more robust and secure locking system communications

Questions