Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Evaluation of parallelism techniques on embedded multi-core platforms

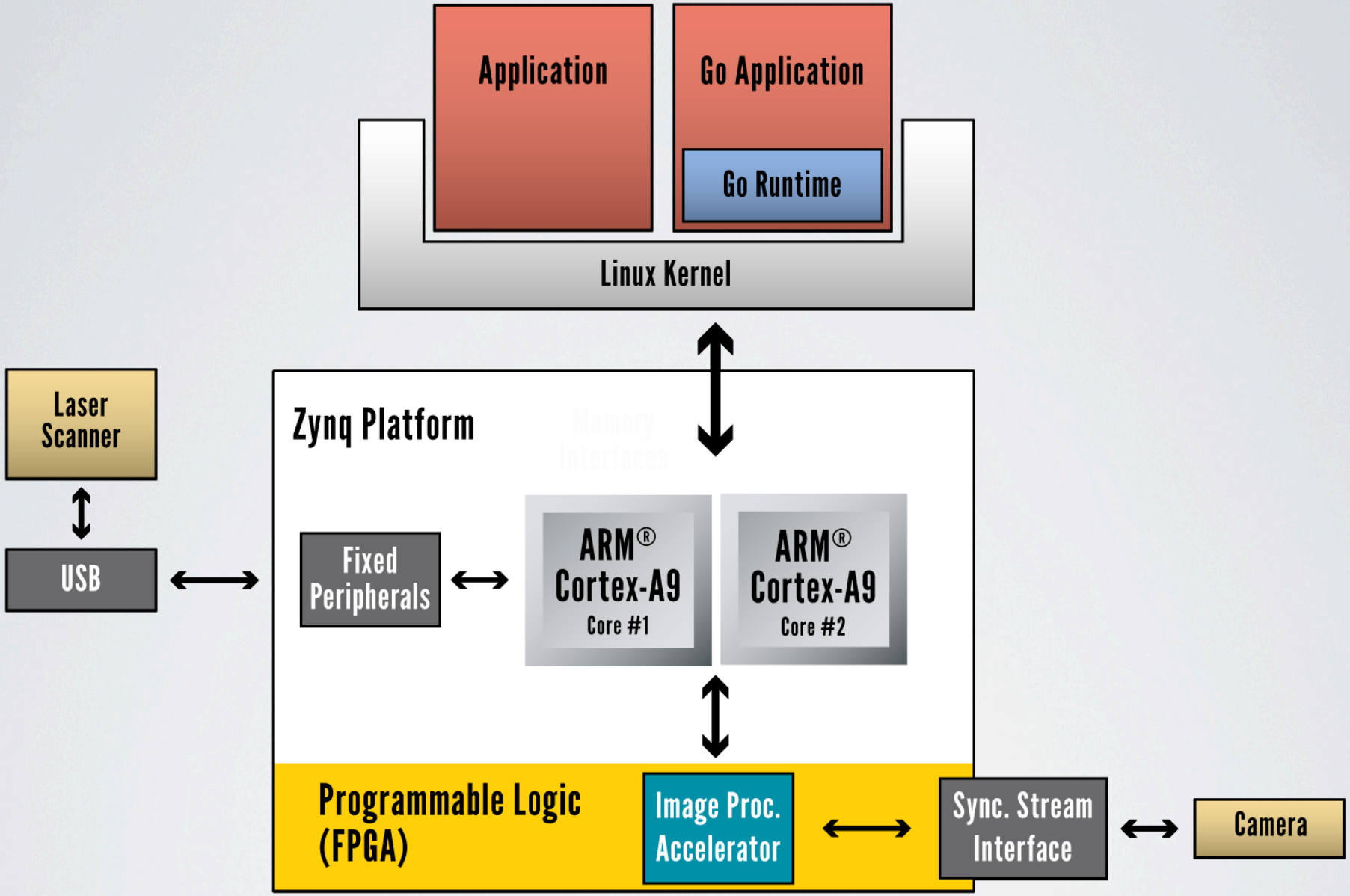
Lucas Jenß

Mentor: Prof. Dr. B. Schwarz

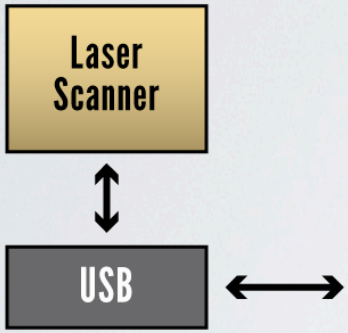
Outline

1. Development Platform
2. Parallelism Techniques
   - Pipeline Pattern
   - Single Instruction – Multiple Data (SIMD)
3. Google Go for parallelism

The Platform


Application
Go Application
Go Runtime
Linux Kernel
Laser Scanner
USB
Zynq Platform
Fixed Peripherals
ARM® Cortex-A9 Core #1
ARM® Cortex-A9 Core #2
Programmable Logic (FPGA)
Image Proc. Accelerator
Sync. Stream Interface
Camera

Obstacle Detection

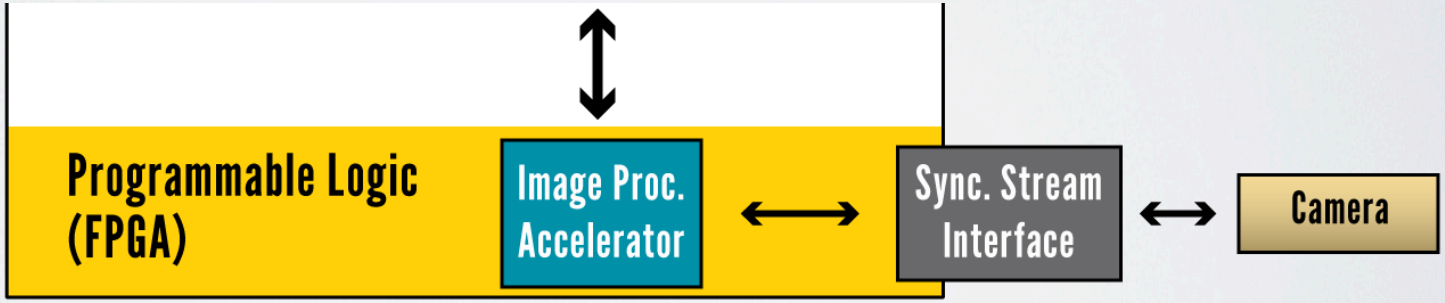


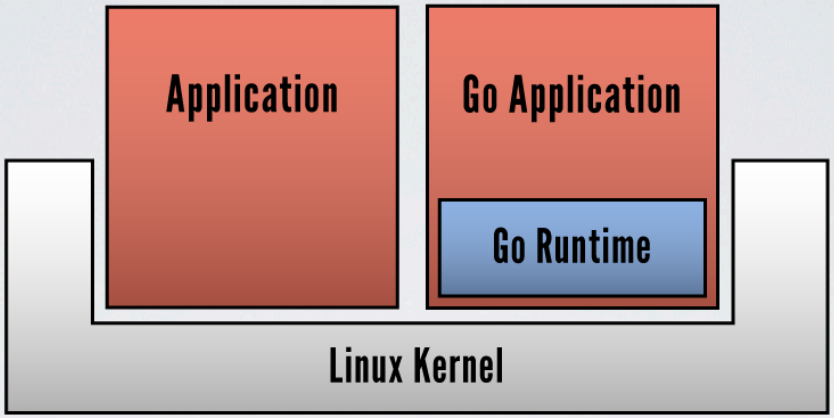
- Using Laser scanner
  - Data rate: ~34KB/s
  - 10Hz scanning freq.
  - 240° in 0.36° steps
- Currently implemented using Java on Android OS

[1]

Lane Guiding Control

- Lane Guiding Control using high FPS camera (60 FPS, 640x480px)
- Data rate: ~17.6MB/s
- Pixel Clock: 27MHz
- Implemented using C and RTL models, on Linux





- Current state: mix of different technologies
  - Java on Android
  - C on Linux parallel to RTL/FPGA
- Goals:
  - technological unification of the platform
  - parallelization of tasks to improve throughput

Outline

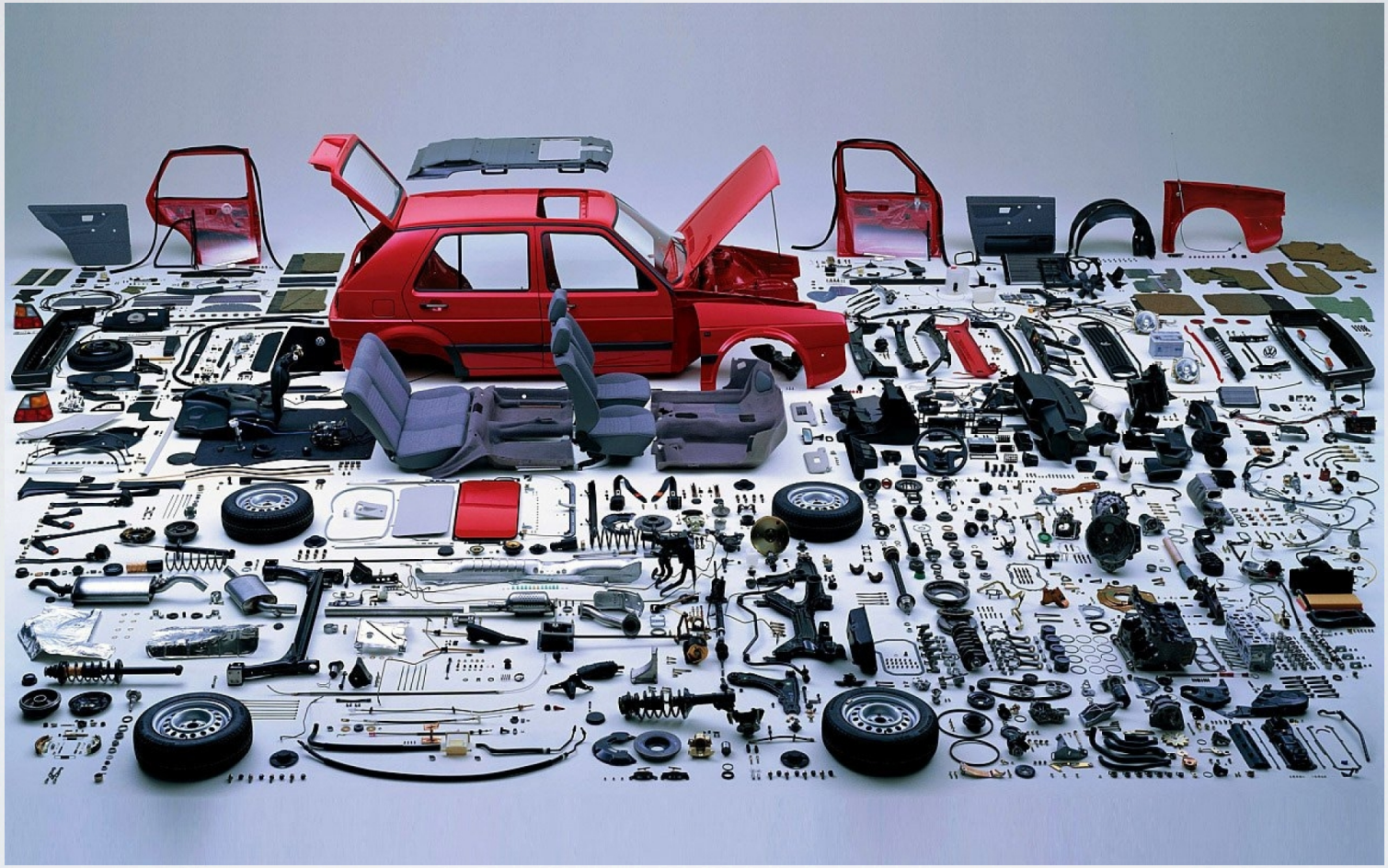
1. Development Platform
2. Parallelism Techniques
   - Pipeline Pattern
   - Single Instruction – Multiple Data (SIMD)
3. Google Go for parallelism

Parallelism Techniques

Pipeline pattern

Pipeline Pattern

”Assembly line” example

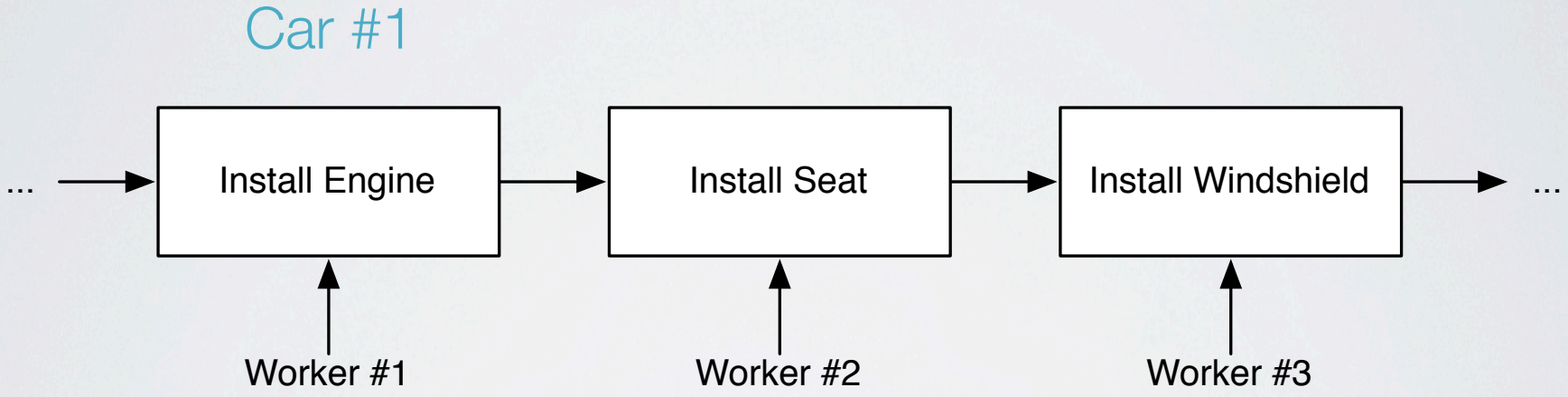


[2]

Pipeline Pattern

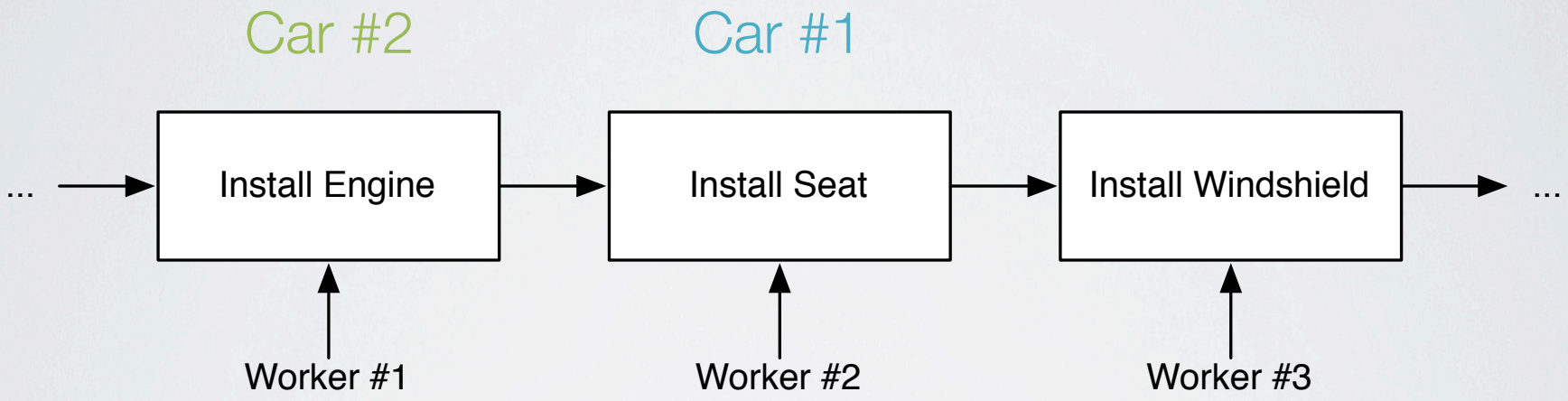
”Assembly line” example

- Analogy: Assembly line (e.g. for a Car)



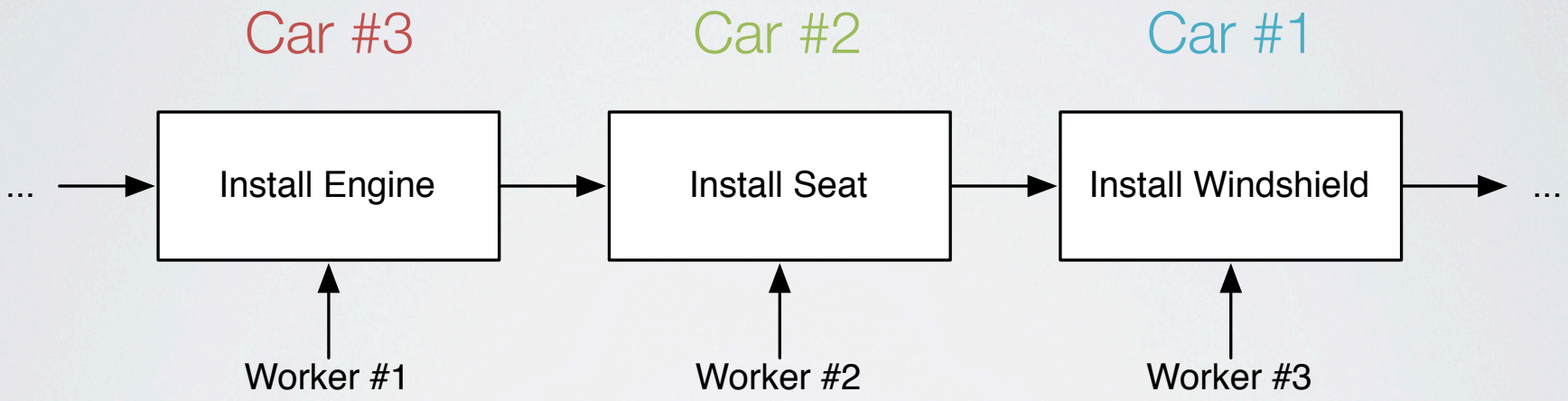
Pipeline Pattern

”Assembly line” example



Pipeline Pattern

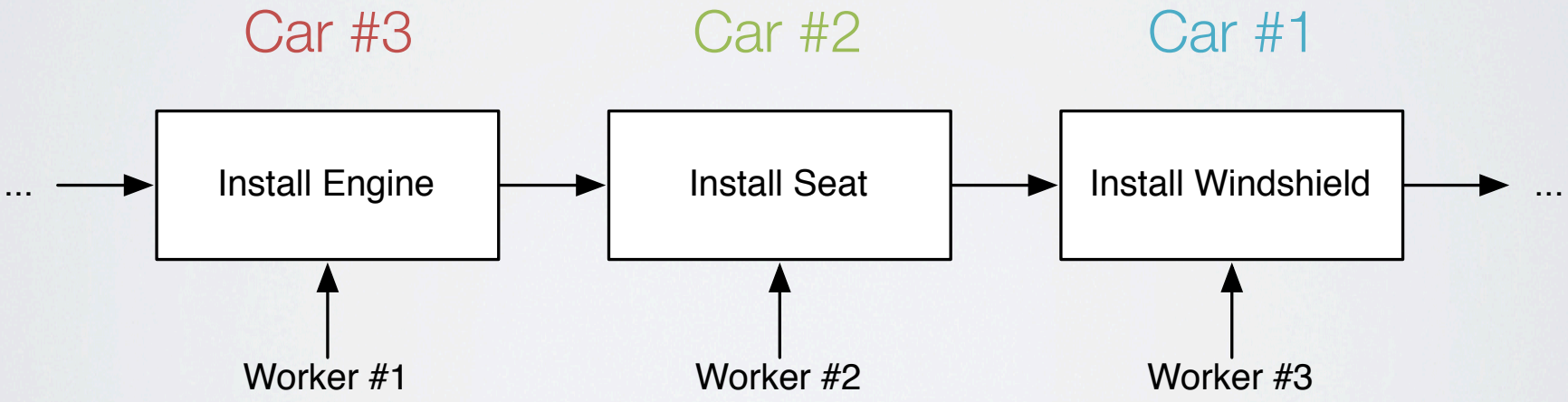
”Assembly line” example



Pipeline Pattern

”Assembly line” example

- All workers
  - are busy at all times
  - work on different tasks on different cars
  - yet all tasks are directly related



Pipeline Pattern

”Assembly line” example

- Assembly line has
  - Workers
  - Cars, or some other items
  - Transport mechanism for the items

Pipeline Pattern

”Assembly line” example

- For the pipeline pattern, this translates to
  - Threads (instead of Workers)
  - Data (instead of Cars)
  - Communication channel
    (instead of the transport mechanism)

Pipeline Stage

Pseudo code

```
while(more elements) {
    receive element from previous stage
    perform operation on element
    send element to the next stage
}
```

Pipeline Stage

Pseudo code

```
while(more elements) {
    receive element from previous stage
    perform operation on element
    send element to the next stage
}
```

Pipeline Stage

Pseudo code

```
while(more elements) {
    receive element from previous stage
    perform operation on element
    send element to the next stage
}
```

Pipeline Pattern

- Theoretically:
  - Parallelization of any computing problem
  - Because the pipeline “transforms” linearity into parallelism

Pipeline Pattern

- In practice:
  - Communication between stages is the bottleneck
  - In order to achieve good performance:
    - Communication time $\ll$ Stage operation time
    - Small difference between stage operation times

Outline

1. Development Platform
2. Parallelism Techniques
   - Pipeline Pattern
   - Single Instruction – Multiple Data (SIMD)
3. Google Go for parallelism

Single Instruction – Multiple Data

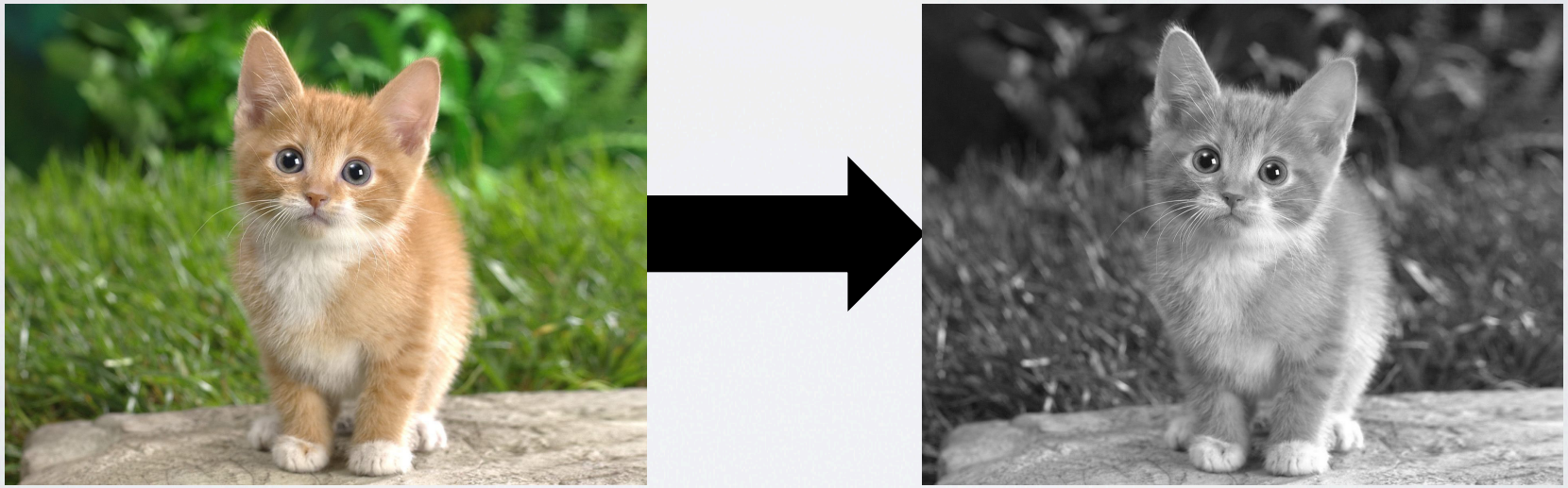
Using the ARM NEON engine

Single Instruction - Multiple Data

- Exploits data-level parallelism
  - Operations repeatedly applied to independent data

Single Instruction - Multiple Data

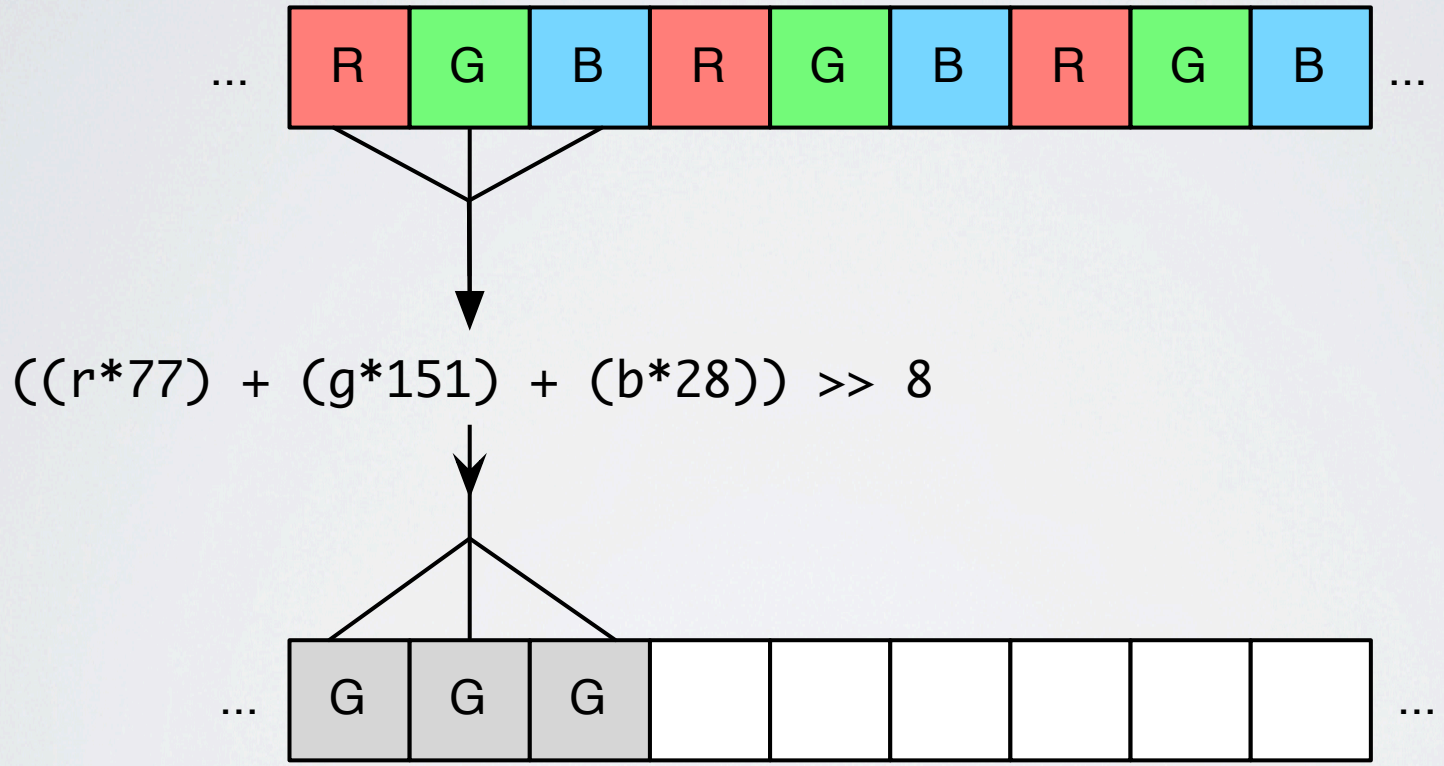
- Example: Converting an image to gray scale
  - Weighted average of each pixel's RGB values
    - Sum of products



[3]

Single Instruction – Single data

Converting an image to gray scale



Single Instruction – Single data

Image to gray scale – Sum of products

```
void rgb2grey(uint8_t *dest, uint8_t *src, int pixels)
{
    for(int i=0; i < pixels; i++) {
        uint8_t r = *src++;
        uint8_t g = *src++;
        uint8_t b = *src++;

        uint16_t y =
            r * 77 +
            g * 151 +
            b * 28;

        *dest++ = (y >> 8);
    }
}
```

Single Instruction – Single data

Converting an image to gray scale

```
void rgb2grey(uint8_t *dest, uint8_t *src, int pixels)
{
    for(int i=0; i < pixels; i++) {
        uint8_t r = *src++;
        uint8_t g = *src++;
        uint8_t b = *src++;

        uint16_t y =
            r * 77 +
            g * 151 +
            b * 28;

        *dest++ = (y >> 8);
    }
}
```

Single Instruction – Single data

Converting an image to gray scale

```
void rgb2grey(uint8_t *dest, uint8_t *src, int pixels)
{
    for(int i=0; i < pixels; i++) {
        uint8_t r = *src++;
        uint8_t g = *src++;
        uint8_t b = *src++;

        uint16_t y =
            r * 77 +
            g * 151 +
            b * 28;

        *dest++ = (y >> 8);
    }
}
```

Single Instruction – Single data

Converting an image to gray scale

```
void rgb2grey(uint8_t *dest, uint8_t *src, int pixels)
{
    for(int i=0; i < pixels; i++) {
        uint8_t r = *src++;
        uint8_t g = *src++;
        uint8_t b = *src++;

        uint16_t y =
            r * 77 +
            g * 151 +
            b * 28;

        *dest++ = (y >> 8);
    }
}
```

Single Instruction – Multiple data

Converting an image to gray scale

- ARM NEON uses special registers
  - Separate from CPU registers
  - “Q”(uad) and “D”(ouble), 128-bit and 64-bit wide
  - Multiple elements per register
    - e.g. 8 * 8 bit in “D” register

Single Instruction – Multiple data

Converting an image to gray scale

- Steps to use SIMD:
  1. Load data into NEON registers
  2. Apply operations
  3. Write data back to memory

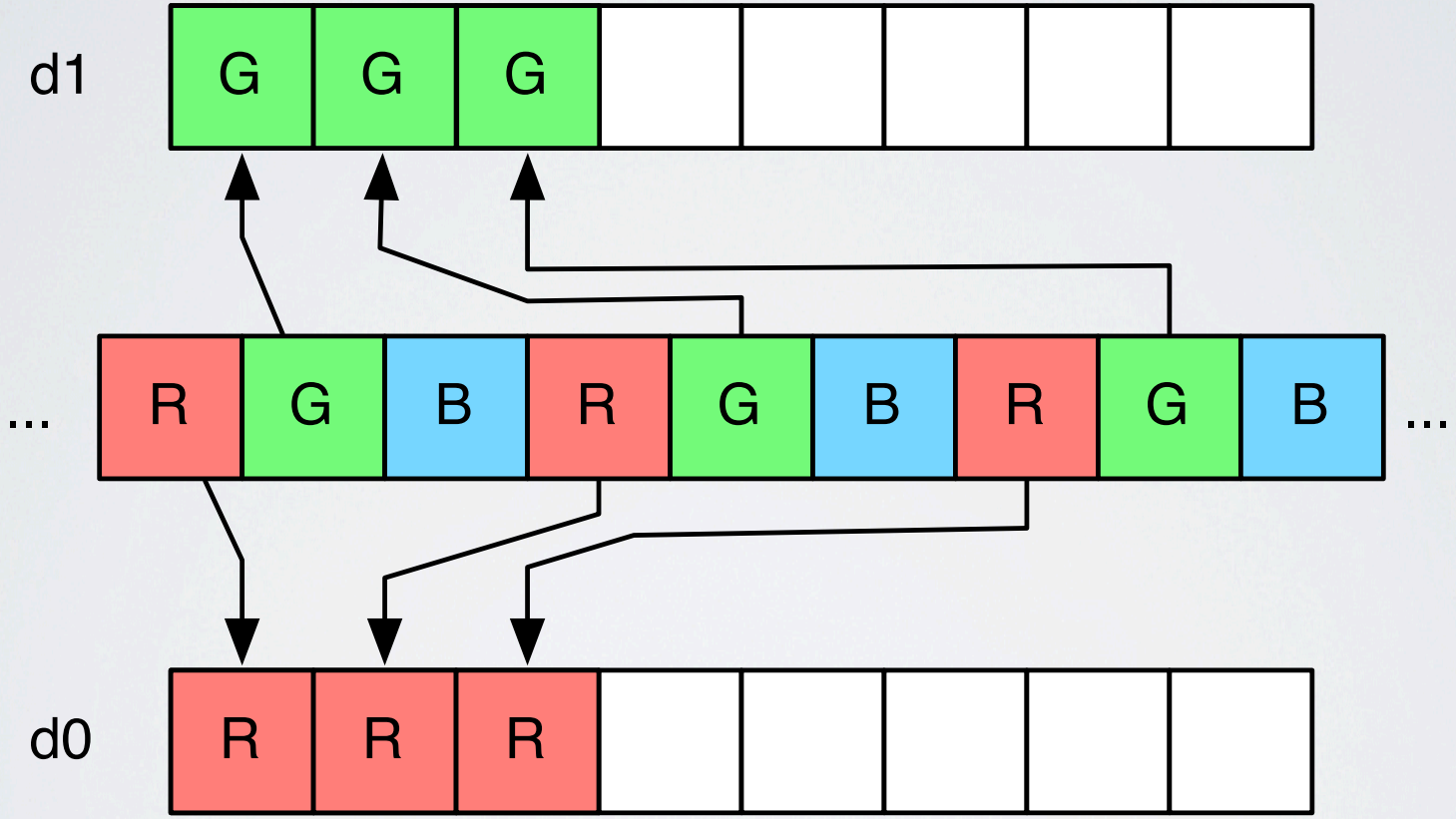
Single Instruction – Multiple data

Converting an image to gray scale

- Steps to use SIMD:
  1. Load data into NEON registers
  2. Apply operations
  3. Write data back to memory

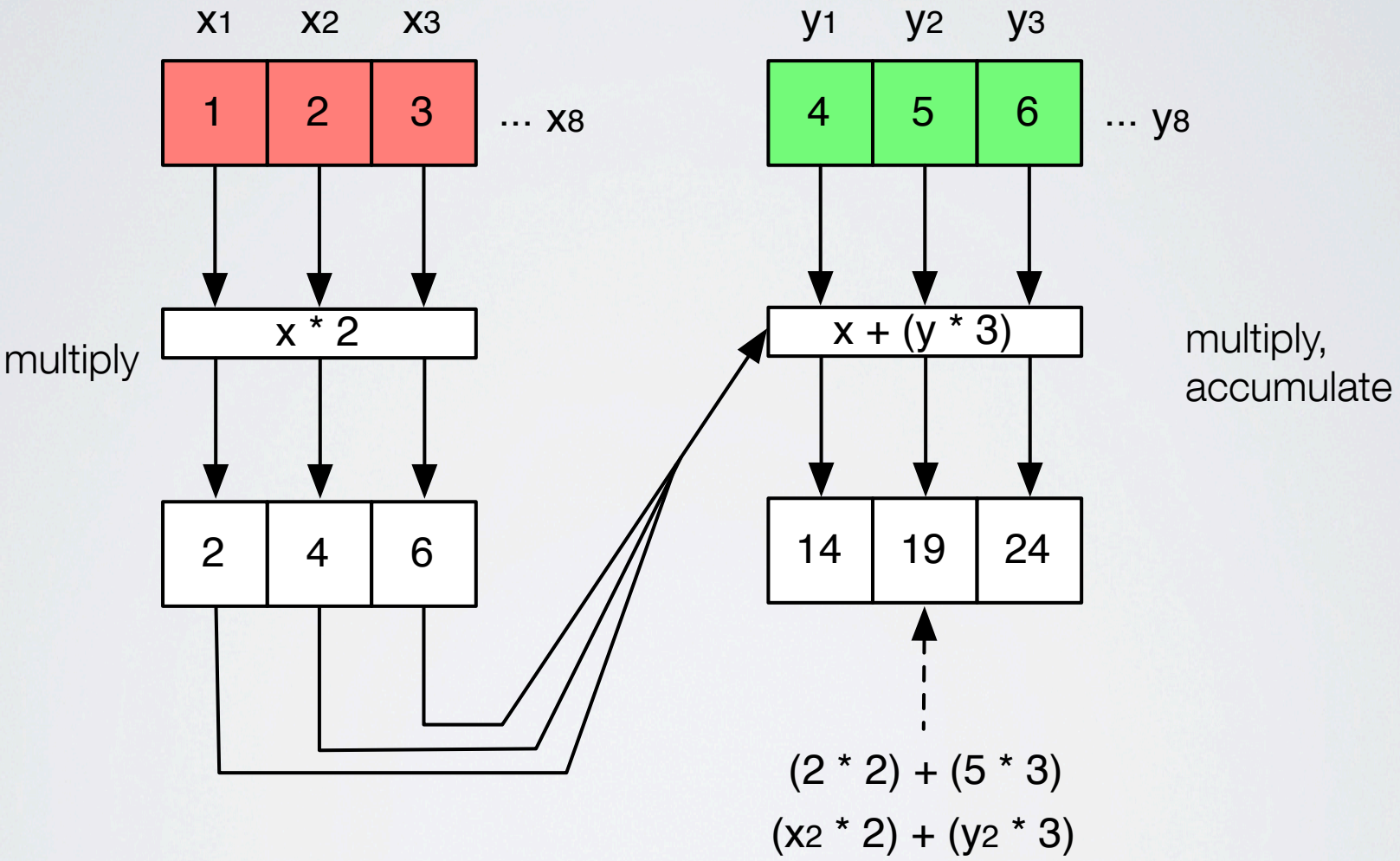
Single Instruction – Multiple data

Load data into NEON “D” registers



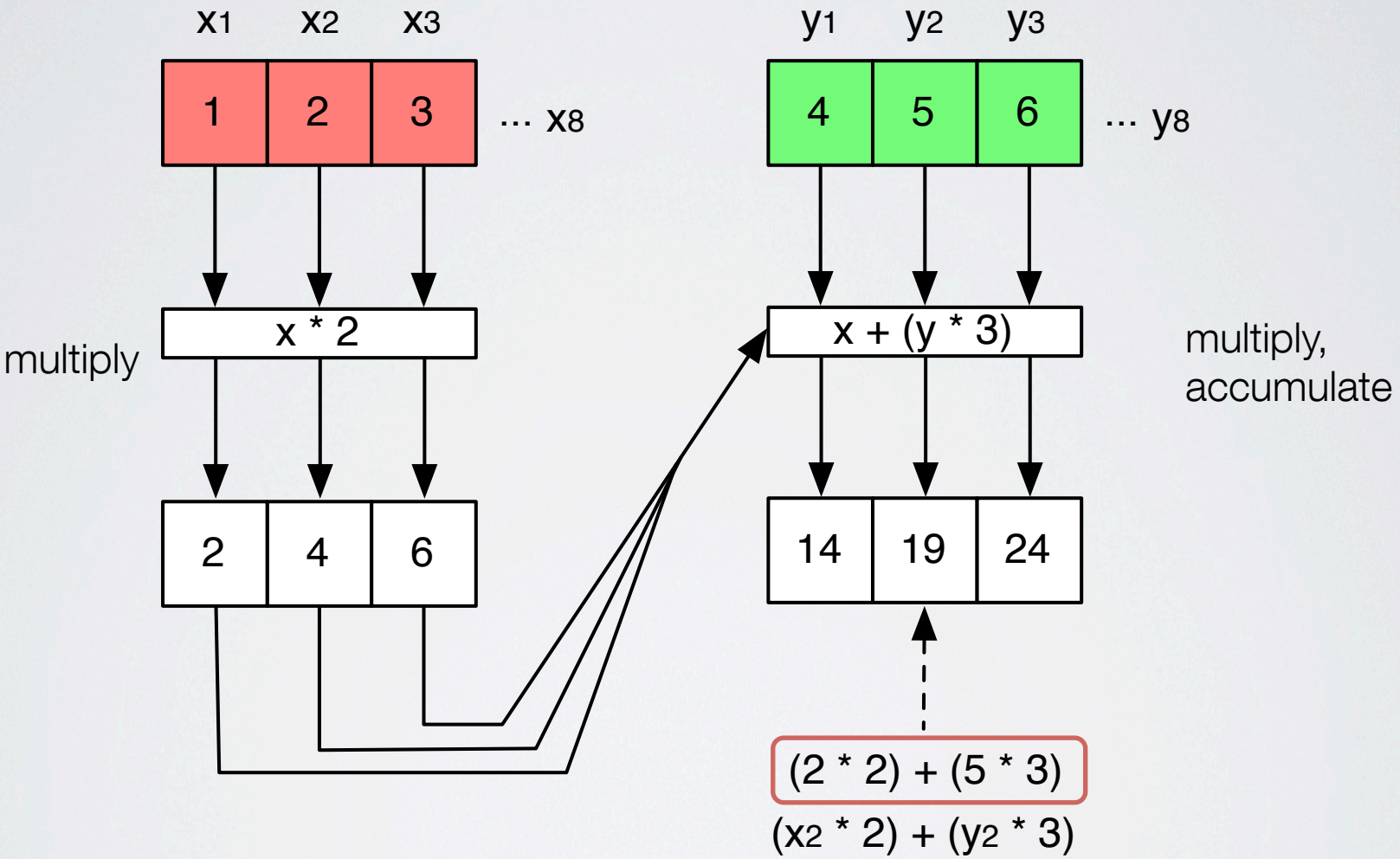
Single Instruction – Multiple data

Apply operations



Single Instruction – Multiple data

Apply operations



Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Load (3 Interleaved)

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;
```

uint8x8x3_t

Type

Elements per Register

# of Registers

```
}
```

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;



}
```

Loads 8bit * 8 * 3 = 192 bit = 24 byte!

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Apply operations

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Multiply

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Multiply + Accumulate

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Shift Right

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

Write back to memory

Single Instruction – Multiple data

Converting to gray scale

```
for(int i=0; i < pixels; i += FACTOR) {
    uint8x8x3_t rgb = vld3_u8(src);
    uint16x8_t acc;

    acc = vmull_u8(rgb.val[0], rcoeff);
    acc = vmlal_u8(acc, rgb.val[1], gcoeff);
    acc = vmlal_u8(acc, rgb.val[2], bcoeff);

    uint8x8_t result = vshrn_n_u16(acc, 8);
    vst1_u8(dest, result);

    src += FACTOR;
    dest += FACTOR;
}
```

FACTOR = # of elements processed per iteration (24)

Single Instruction – Multiple data

Performance

- Native C-Version
  - 16 Instructions / Pixel
- SIMD Version:
  - 15 Instructions / 8 Pixels
  - ~2 Instructions / Pixel

Outline

1. Development Platform
2. Parallelism Techniques
   - Pipeline Pattern
   - Single Instruction – Multiple Data (SIMD)
3. Google Go for parallelism

Google Go

for parallel applications

Google Go

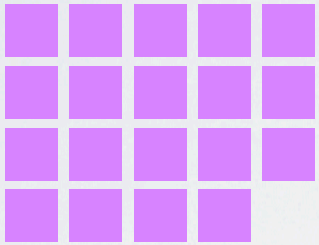
- A programming language that is
  - simple
  - compiled
  - built with concurrency in mind

Language Specification

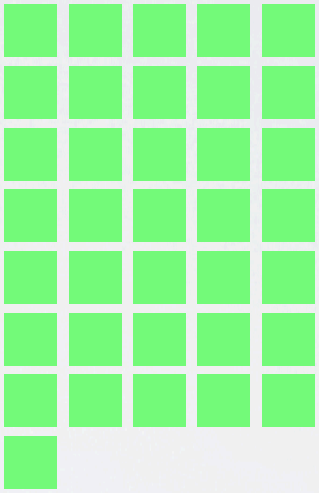
Who's keeping it simple?

Go

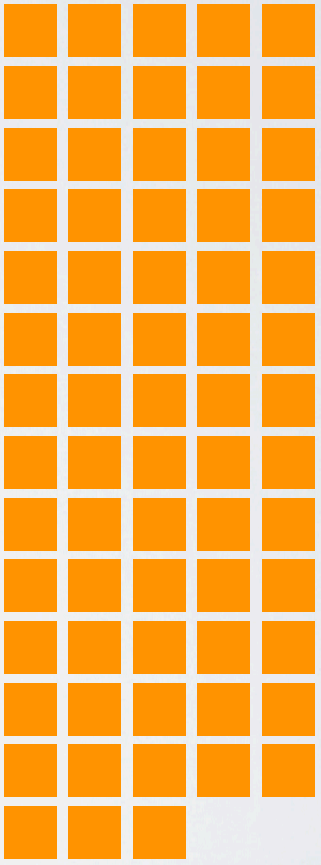
Scala



C++



Java



■ = 10 pages (DIN A4)

Image courtesy of Marian Tietz
"Google Go – An Introduction"

Google Go

Concurrency as a language feature

- Go-Routines (`go` keyword)
  - lightweight threads scheduled by the Go runtime
  - concurrently execute ***any*** function

  - sequential execution:

```
function(parameter1, parameterN…)
```

Google Go

Concurrency as a language feature

- Go-Routines (`go` keyword)
  - lightweight threads scheduled by the Go runtime
  - concurrently execute ***any*** function

  - concurrent execution:

```
go function(parameter1, parameterN…)
```

Google Go

Concurrency as a language feature

- Channels
  - first-class value (i.e. handled as a primitive)
  - thread-safe communication mechanism
  - buffered / unbuffered

Google Go

Concurrency as a language feature

- Channels
  - create channel

```
chan := make(chan int)
```

Google Go

Concurrency as a language feature

- Channels
  - send data to channel

```
chan <- 5
```

Google Go

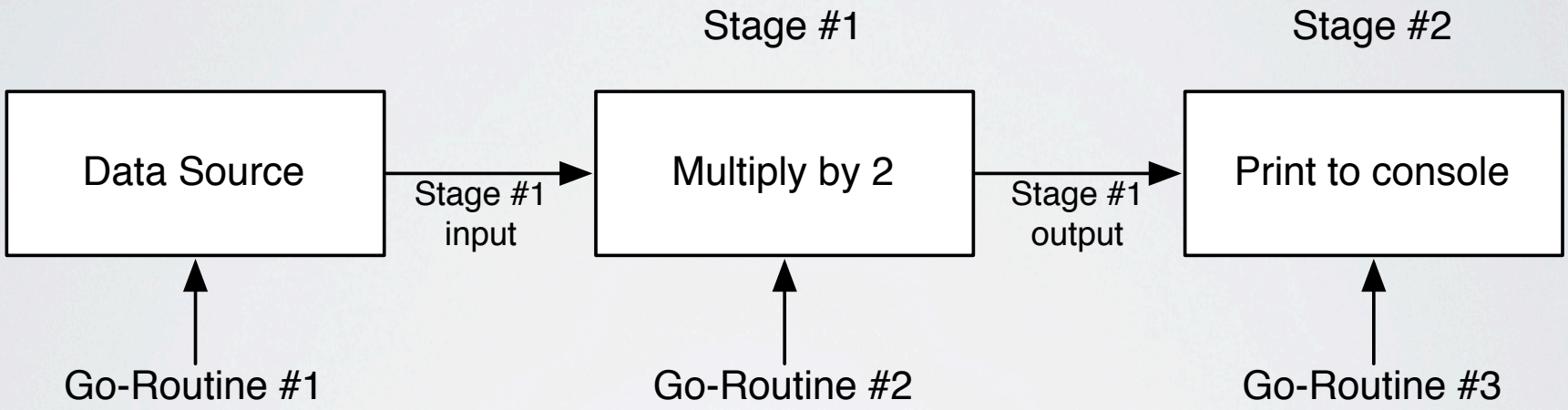
Concurrency as a language feature

- Channels
  - retrieve data from channel

```
value := <- chan
```

Code example

Pipeline pattern in Go!



Code example

Pipeline pattern in Go!

```
func main() {
    stage1_in := make(chan int)
    stage1_out := make(chan int)

    go func() {
        for _, element := range [4]int{1, 2, 3, 4} {
            stage1_in <- element
        }
    }()

    go stage1(stage1_in, stage1_out)
    go stage2(stage1_out, nil)
}
```

Code example

Pipeline pattern in Go!

```
func main() {
    stage1_in := make(chan int)
    stage1_out := make(chan int)

    go func() {
        for _, element := range [4]int{1, 2, 3, 4} {
            stage1_in <- element
        }
    }()

    go stage1(stage1_in, stage1_out)
    go stage2(stage1_out, nil)
}
```

Code example

Pipeline pattern in Go!

```
func main() {
    stage1_in := make(chan int)
    stage1_out := make(chan int)

    go func() {
        for _, element := range [4]int{1, 2, 3, 4} {
            stage1_in <- element
        }
    }()

    go stage1(stage1_in, stage1_out)
    go stage2(stage1_out, nil)
}
```

Code example

Pipeline pattern in Go!

```
func main() {
    stage1_in := make(chan int)
    stage1_out := make(chan int)

    go func() {
        for _, element := range [4]int{1, 2, 3, 4} {
            stage1_in <- element
        }
    }()

    go stage1(stage1_in, stage1_out)
    go stage2(stage1_out, nil)
}
```

Code example

Pipeline pattern in Go!

```
func stage1(in chan int, out chan int) {
    for {
        current := <- in
        op := current * 2
        out <- op
    }
}

func stage2(in chan int, out chan int) {
    for {
        current := <- in
        fmt.Println("Value: ", current)
    }
}
```

Code example

Pipeline pattern in Go!

```
func stage1(in chan int, out chan int) {
    for {
        current := <- in
        op := current * 2
        out <- op
    }
}

func stage2(in chan int, out chan int) {
    for {
        current := <- in
        fmt.Println("Value: ", current)
    }
}
```

Read data from channel

Code example

Pipeline pattern in Go!

```
func stage1(in chan int, out chan int) {
    for {
        current := <- in
        op := current * 2
        out <- op
    }
}

func stage2(in chan int, out chan int) {
    for {
        current := <- in
        fmt.Println("Value: ", current)
    }
}
```

Perform operation

Code example

Pipeline pattern in Go!

```
func stage1(in chan int, out chan int) {
    for {
        current := <- in
        op := current * 2
        out <- op
    }
}

func stage2(in chan int, out chan int) {
    for {
        current := <- in
        fmt.Println("Value: ", current)
    }
}
```

Send data to next stage's channel

Code example

Pipeline pattern in Java…

Code example

Pipeline pattern in Java…

```
import java.util.ArrayList;
import java.util.List;



public interface Pipeline extends Stage {
         public void addStage ( Stage stage);

         public void addErrorStage (Stage stage);

         public void addFinalStage (Stage stage);
}

public interface PipelineContext {
        public List<Error> getErrors ();
}

public class PipelineContextAdaptor implements PipelineContext {

        private List<Error> m_errors = new ArrayList<Error> ();
        public List<Error> getErrors() {
                return m_errors;
        }

        public void addError (Error e){
                m_errors.add(e);
        }

}

public interface Stage {
        public void execute (PipelineContext context);
}
```

```
public class SequentialPipeline implements Pipeline {

        private List<Stage> m_stages = new ArrayList<Stage> ();
        private List<Stage> m_errorStages= new ArrayList<Stage> ();
        private List<Stage> m_finalStages= new ArrayList<Stage> ();

        public void addStage(Stage stage) {
                m_stages.add(stage);

        }

        public void addErrorStage(Stage stage) {
                m_errorStages.add(stage);
        }

        public void addFinalStage(Stage stage) {
                m_finalStages.add(stage);

        }

        public void execute(PipelineContext context) {

                for (Stage stage:m_stages){

                        stage.execute(context);

                        if (context.getErrors()!= null && !context.getErrors().isEmpty(){
                                break;
                        }

                }

                if (context.getErrors()!= null && !context.getErrors().isEmpty()){
                        for (Stage errorStage: m_errorStages){
                                errorStage.execute(context);
                        }
                }

                for (Stage finalStage: m_finalStages){
                        finalStage.execute(context);
                }
        }

}
```

Google Go

Can we use it on embedded platforms?

- We’ve seen that Go
  - allows for quicker development due to less boilerplate
    - simple syntax – few corner cases

Google Go

Can we use it on embedded platforms?

- We've seen that Go
  - is great for building parallel applications
    - channels as a language feature
    - go-routines (lightweight threads) as a language feature
    - compiler/runtime knows these features, **can find errors**

Google Go

Can we use it on embedded platforms?

- Go Concurrency
  - Straightforward implementation of the pipeline pattern
  - But other concurrency patterns work great too
    - Actors
    - Fork/Join
    - Master/Worker

Google Go

Can we use it on embedded platforms?

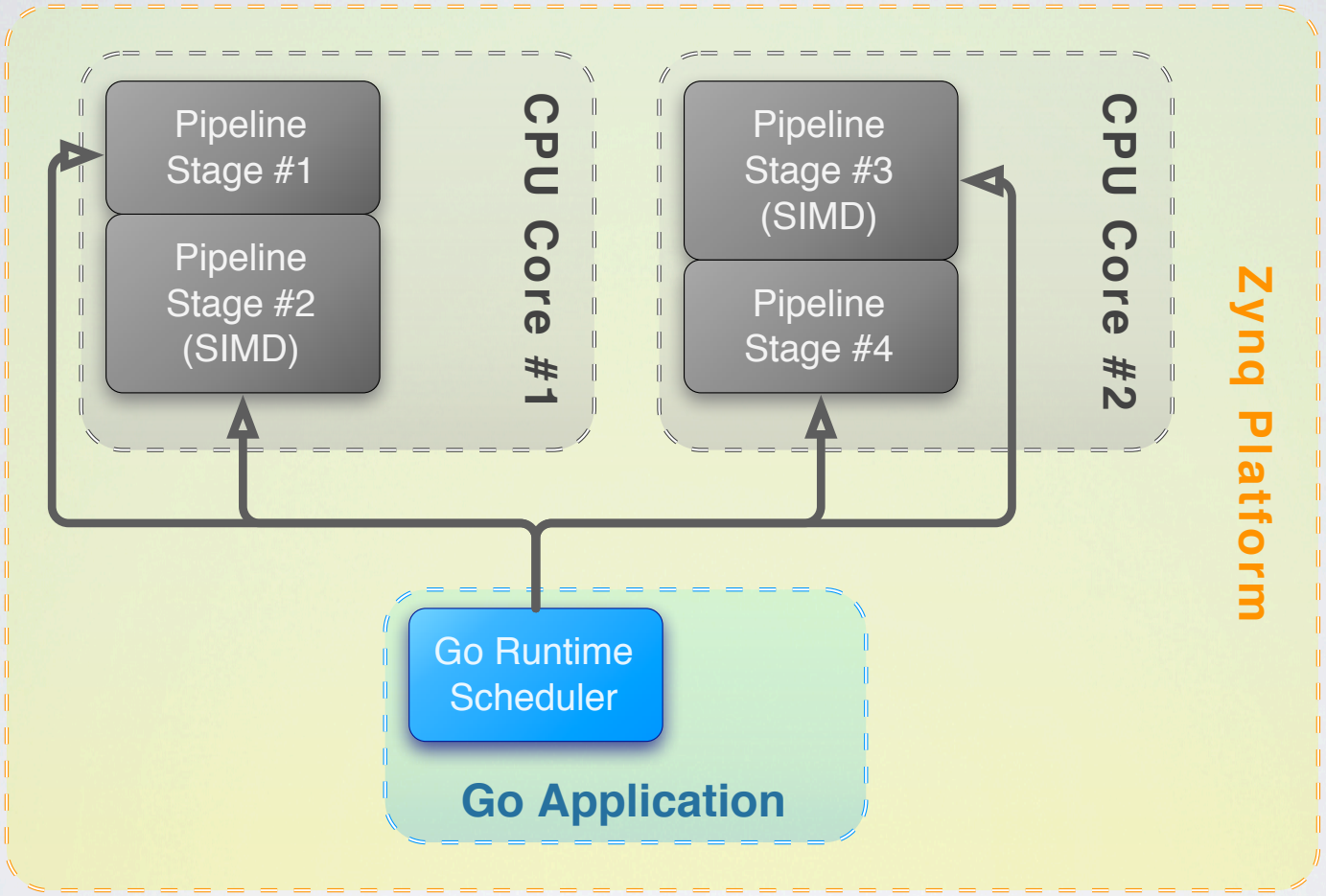
- But: simplicity comes at a cost
- Therefore, the question is
  - How big is the performance impact of using Go on embedded system?
  - And ultimately: Do the advantages of Go outweigh said performance impact?

Next Milestones

Next Milestones

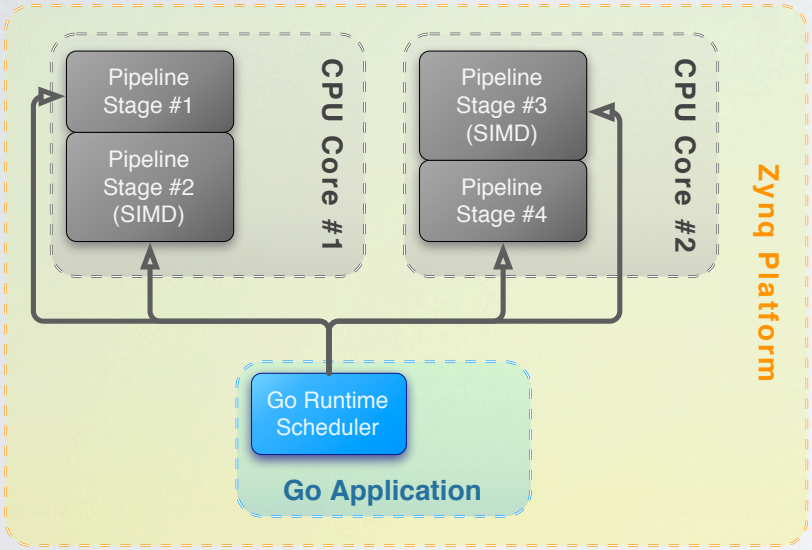
- Technological unification on Linux
  - Determine if Go is a viable alternative to C
- Re-implementation of obstacle detection in Go or C
    - Pipeline Pattern and SIMD for throughput optimization

The Big Picture



Legend: Go-Routine

The Big Picture



Legend: Go-Routine

- Processing pipeline distributed to multiple cores
- Some stages use SIMD
- Go Runtime schedules the go-routines

The End

Any questions?

References

FAUST Project (2013). FAUST - Fahrerassistenz und autonome Systeme.
URL: http://faust.informatik.haw-hamburg.de (last accessed: 2013-05-07)

Kejariwal, A., Veidenbaum, A. V., Nicolau, A., Girkar, M., Tian, X. and Saito, H. (2009). On the exploitation of loop-level parallelism in embedded applications, *ACM Transac- tions on Embedded Computing Systems* **8**(2): 1–34. ISSN: 15399087.
URL: http://portal.acm.org/citation.cfm?doid=1457255.1457257

Mattson, T. G., Sanders, B. A. and Massingill, B. L. (2010). *Patterns for Parallel Pro- gramming*, 6th edn, Addison-Wesley, Westford, Massachusetts. ISBN: 0-321-22811- 1.

Pike, R. (2012). Go at Google, *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity - SPLASH '12* p. 5.
URL: http://dl.acm.org/citation.cfm?doid=2384716.2384720

Preud'Homme, T., Sopena, J., Thomas, G. and Folliot, B. (2012). An Improvement of OpenMP Pipeline Parallelism with the BatchQueue Algorithm, *2012 IEEE 18th International Conference on Parallel and Distributed Systems* (i): 348–355.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber= 6413677

Images

[1] URL:
http://de.manu-systems.com/URG-04LX-UG01.shtml (last accessed on 2013-06-02)

[2] URL:
http://www.wall321.com/Cars/Volkswagen/cars_vehicles_volkswagen_1280x800_wallpaper_44233/download_1920x1200 (last accessed on 2013-05-21)

[3] URL:
http://www.wallpaperdev.com/wallpaper/1600x1200/cute-cat-by-ashish-8511.html (last accessed on 2013-06-03)