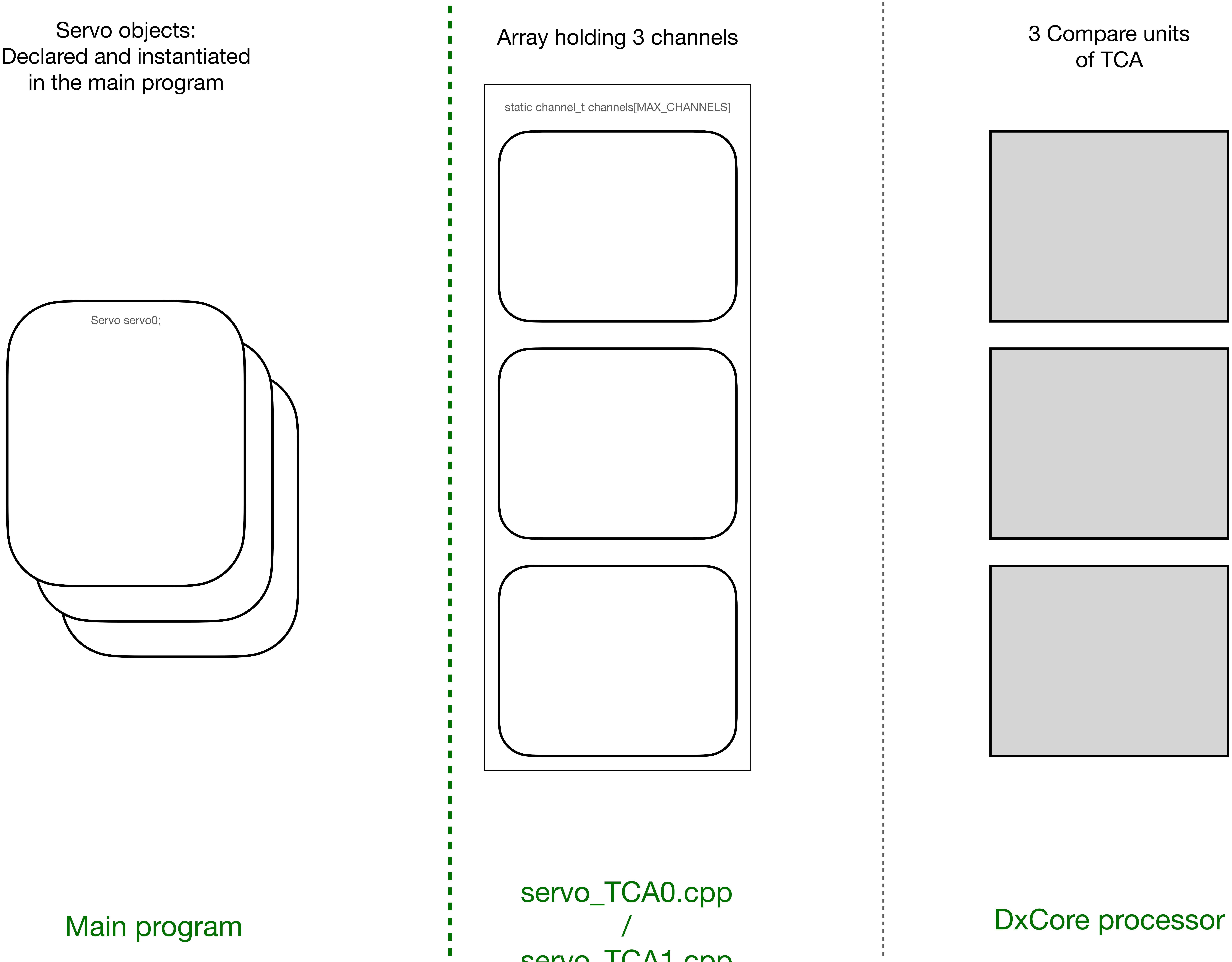
Internal Structure

Servo objects,
the channels array,
the three Compare Units



Servo objects:
Declared and instantiated
in the main program

Servo servo0;

Main program

Array holding 3 channels

static channel_t channels[MAX_CHANNELS]

servo_TCA0.cpp
/
servo_TCA1.cpp

3 Compare units
of TCA

DxCore processor

The Servo class acts as an interface to the channel array.

The class is upwards compatible to existing servo libraries.

The *acceptsNewValue* and *constantOutput* methods
are added to this library
and are not available in other servo libraries.

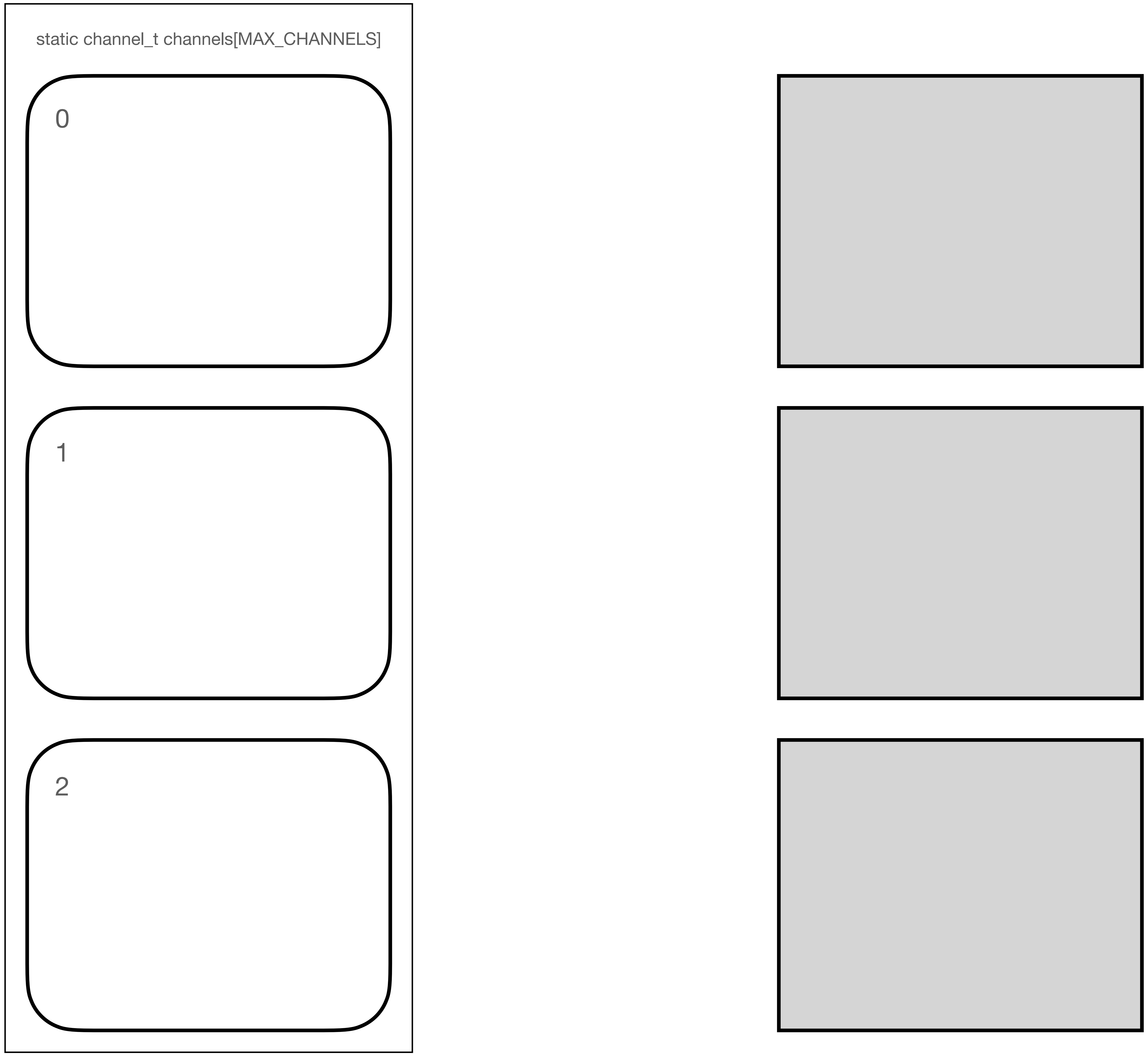
```
Servo()
uint8_t attach(uint8_t pin)
uint8_t attach(uint8_t pin, int min, int max)
detach()
write(uint16_t value);
writeMicroseconds(uint16_t value);
int read();
uint16_t readMicroseconds();
bool attached();
bool acceptsNewValue();
constantOutput(uint8_t on_off)

uint8_t servoIndex;
int8_t min;
int8_t max;
```

After an object of the Servo class is instantiated,
the attributes are initialized with the following values:

servoIndex: 0, 1 or 2
In case of failure: INVALID_SERVO

min: 0
max: 0



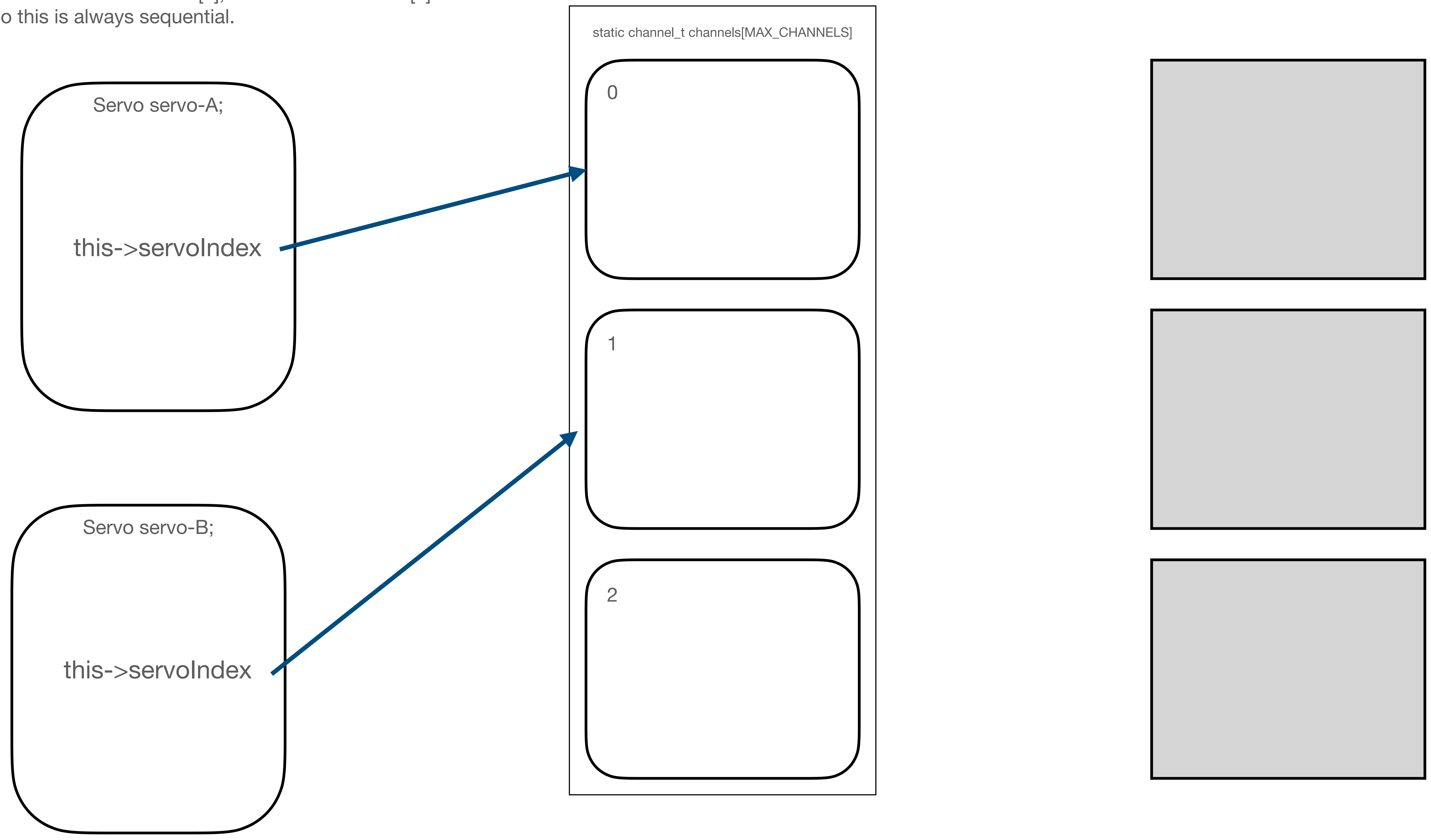
static channel_t channels[MAX_CHANNELS]

0

1

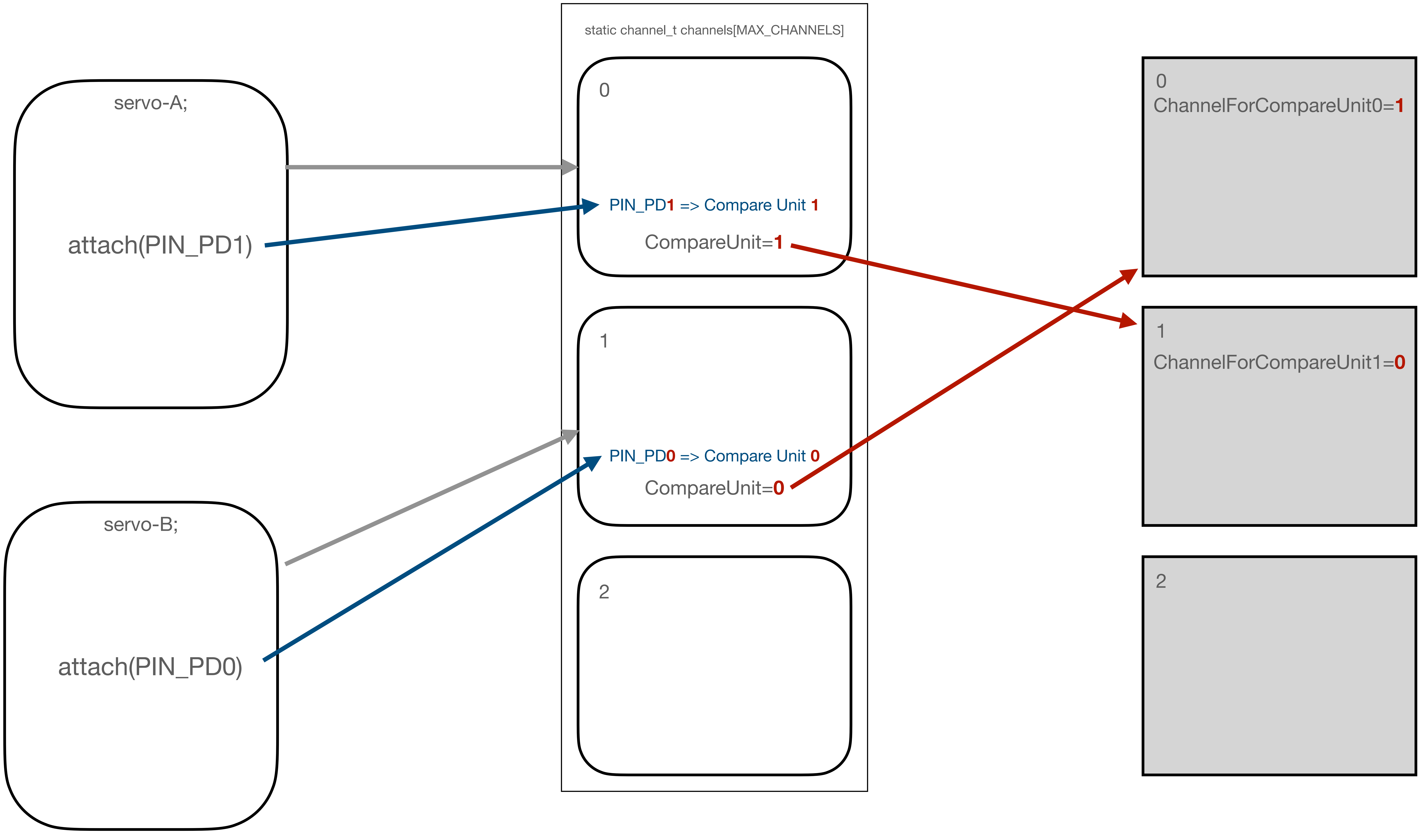
2

Instantiation of servo objects
The first instantiated object gets an index to channels[0]
The second to channels[1], the third to channels[3]
So this is always sequential.



If an attempt is made to instantiate more than 3 servos,
their channelIndex becomes INVALID_SERVO

Attach()



servo-A;

attach(PIN_PD1)

servo-B;

attach(PIN_PD0)

static channel_t channels[MAX_CHANNELS]

0

PIN_PD1 => Compare Unit 1

CompareUnit=1

1

PIN_PD0 => Compare Unit 0

CompareUnit=0

2

0
ChannelForCompareUnit0=1

1
ChannelForCompareUnit1=0

2

Internal Structure

Interrupt Service Routine

ISR: switch (CurrentCompareUnit)
Which CMPBUF gets what value?

+2 ms
+4 ms
+6 ms
+8 ms
+10 ms
+12 ms
+14 ms
+16 ms
+18 ms
+20 ms
WO0
WO1
WO2
ISR
CU0

CMP=CMPBUF
CMP=0
CMP=0

CMP=0
CMP=CMPBUF
CMP=0

CMP=0
CMP=0
CMP=CMPBUF

0

CMPBUF=0
CMPBUF=TICKS
CMPBUF=0

1

CMPBUF=0
CMPBUF=0
CMPBUF=TICKS

2

CMPBUF=TICKS
CMPBUF=0
CMPBUF=0

CMP receives its value around 6,6ms after it was entered into CMPBUF

CMPBUF=0
is actually
if (_TIMER.CMPx != OUT_HIGH) _TIMER.CMPxBUF = 0;
The check
if (_TIMER.CMPx != OUT_HIGH)
is needed to facilitate the startup of servos with a HIGH signal
(see later)

Time between specifying a new value
for a servo’s pulse width, and the actual
occurrence of the pulse.
That delay is between 6,6 and 26,6 ms

+5 ms
+10 ms
+15 ms
+20 ms
+25 ms
+30 ms
+35 ms
+40 ms
WO0
WO1
WO2
ISR
CU0
0
1
2

CMP=CMPBUF

interval in which main sketch
can enter the new value
for the servo’s pulse width
for Compare Unit 0

CMPBUF=TICKS
CMPBUF=0
CMPBUF=0

```
if servox.acceptsNewValue() {
  servox.write(value);
}
```

channels[myServo].ticks = usToTicks(value);

Start and Stop of servos

Startup

(temporarily) stop output pulses
and switch off servo’s power

Startup behaviour: Method 1

At startup, signal stays low for a certain period
After that period, the pulse train starts

0
+100 ms
+200 ms
+300 ms
+400 ms
+500 ms
+600 ms
+700 ms
+800 ms
+900 ms
+1000 ms
WO0

code:

```
servo0.constantOutput(0);
servo0.attach(PIN_PD0);
delay(400);
servo0.write(1500);
```

Call constantOutput(0), before calling the first write().
This can be done just before or after the pin gets attached.
Start the desired waiting period, after which the first
write is called.

Ansteuerpuls zu Beginn blockieren und verzögert sowie synchron freigeben

See: https://www.opendcc.de/elektronik/opendecoder/servo_erfahrungen.html

Startup behaviour: Method 2

At startup, signal stays high for a certain period
After that period, the pulse train starts

+100 ms
+200 ms
+300 ms
+400 ms
+500 ms
+600 ms
+700 ms
+800 ms
+900 ms
+1000 ms
+1100 ms
WO0

code:

```
servo0.constantOutput(1);
servo0.attach(PIN_PD0);
delay(600);
servo0.write(1500);
```

Call constantOutput(1), before calling the first write().
This can be done just before or after the pin gets attached.
Start the desired waiting period, after which the first
write is called.

Ansteuerpuls zu Beginn auf Dauer-Ein setzen und verzögert in den Pulsbetrieb umschalten

See: https://www.opendcc.de/elektronik/opendecoder/servo_erfahrungen.html

To (temporarily) stop the output pulses and switch off the servo’s power,
1) constantOutput(0) (or 1)
2) wait for isReady()
3) Switch the servo’s power off

+20 ms
+40 ms
+60 ms
+80 ms
+100 ms
+120 ms
+140 ms
+160 ms
WO0
ISR

`servox.constantOutput(0)`

`servox.acceptsNewValue()`

```
if servox.acceptsNewValue() {
  switch_power_off;
}
```

If we would just switch off the servo power, without
1) issuing a command to stop the pulse train (`constantOutput(0)`), and
2) waiting until this has become effective (`isReady()`),
we run the risk that the power gets switched off somewhere during the pulse, leaving the servo in an undefined state.

Some details

Interrupt Service Routine

Internal: isActive

```
void attach() {
  isActive = true
  …
}
```

```
void detach() {
  isActive = false
  …
}
```

```
bool attached() {
  return isActive
  …
}
```

```
ISR {
  …
  if (channels[compareUnit0].isActive) …
  …
}
```