Callbacks

a cost-benefit comparison

Paweł Wiśniewski
embeff GmbH

Agenda

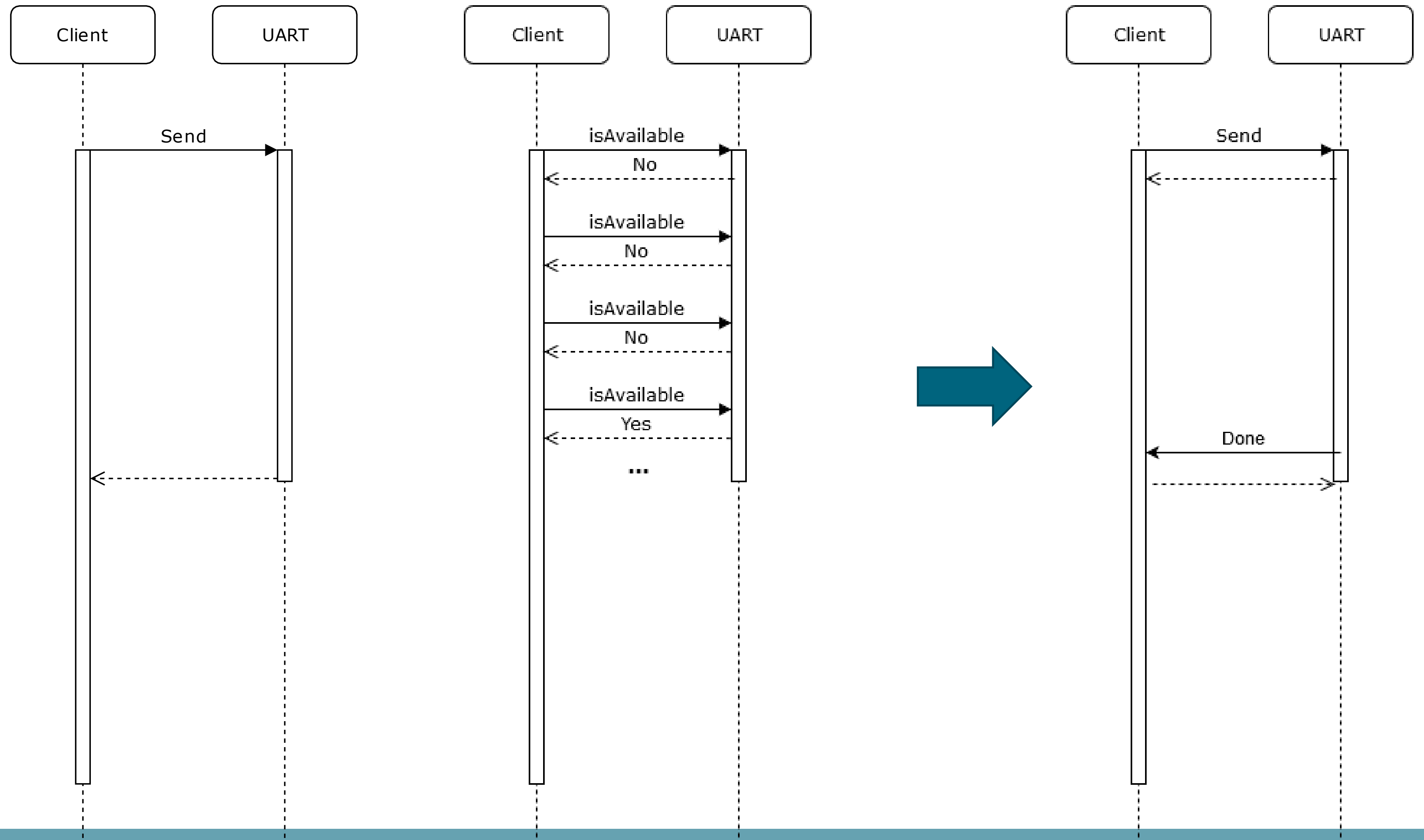
- What is a callback?
- Motivation
- Benchmarks of different implementations
- Conclusion

What is a callback?

- any executable code that is passed as an argument to other code that is expected to *call back* (execute) the argument at a given time.

https://en.wikipedia.org/wiki/Callback_(computer_programming)

Use case



- Sending Data
- Receiving Data
- Waiting
- …

What do we need?

- A possibility to configure behavior
  - Who should be notified
  - How to execute

Implementation possibilities

- Function pointer
- Interface
- std::function
- Template
- …

Motivation

- Abstract classes
- Others saying „this new stuff is to expensive (execution/RAM)” without any evidence
- Feeling „there must be another way“

Comparison

Potential cost

- Execution time
  - call & return
  - assignment
- Ram
- Flash

Potential Benefits

- Flexibility
- Maintainability
- Testability

Benchmarks

Function pointer - implementation

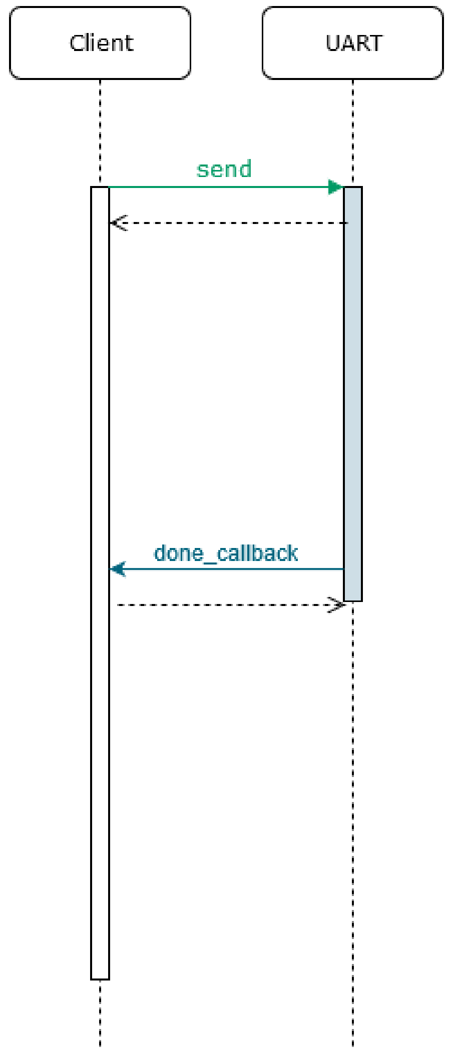
```
struct UART {
  callback_t done_callback;
  void Working() {
    // do the task....
    if(done_callback)
      done_callback();
  }

  void send(callback_t _callback) {
    done_callback = _callback;
    startWork();
  }
};

int main() {
  UART uart;
  //...
  uart.send(done_notification);
}
```

```
using callback_t = void(*)();
```

```
void done_notification() {
   //...
}
```



Function pointer – Resources
ARM Cortex-M4 32bit

- RAM
  - One pointer – 4 Bytes
- Flash
  - No extra cost
- Execution (CPU cycles)

	No Parameters	int (4 bytes)	struct (3 * int)
call	20	21	35
modification	3	3	3

https://b.barebench.com/z/6M2i3y

Interface - implementation

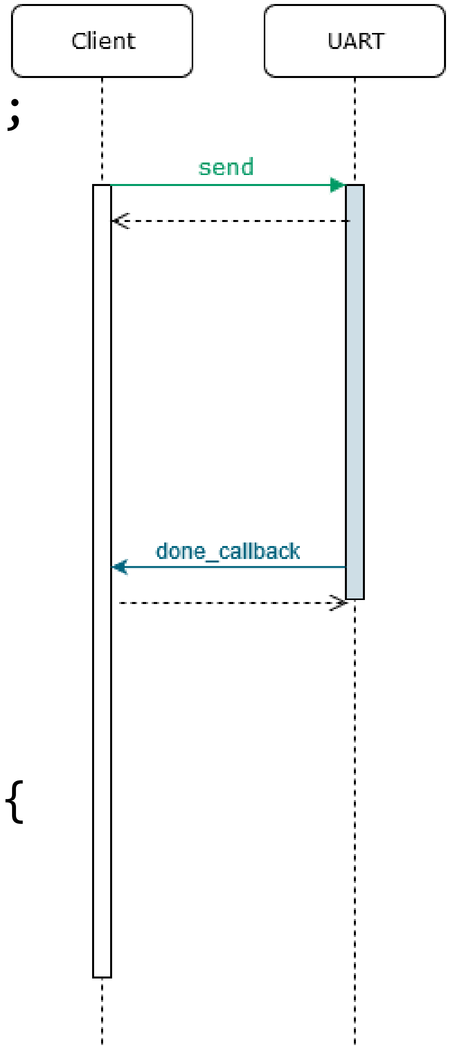
```
struct UART {
  ICallback *callback;
  void Working() {
    // do the task....
    if(callback)
      callback->done_callback();
  }

  void send(ICallback* _callback) {
    callback = _callback;
    startWork();
  }
};

int main() {
  Client client;
  UART uart;
  //...
  uart.send(&client);
}
```

```
struct ICallback {
  virtual void done_callback()=0;
  virtual ~ICallback() {};
};
```

```
struct Client: ICallback {
  //...
  void done_callback() override {
  //...
  }
  //...
};
```



Interface – Resources

ARM Cortex-M4 32bit

- RAM
  - One pointer – 4 Bytes
  - vtable pointer per Instance – 4 Bytes
- Flash
  - vtable
- Execution (CPU cycles)

	No Parameters	int (4 bytes)	struct (3 * int)
call	25	26	44
modification	3	3	3

https://b.barebench.com/z/q6_zBl

std::function

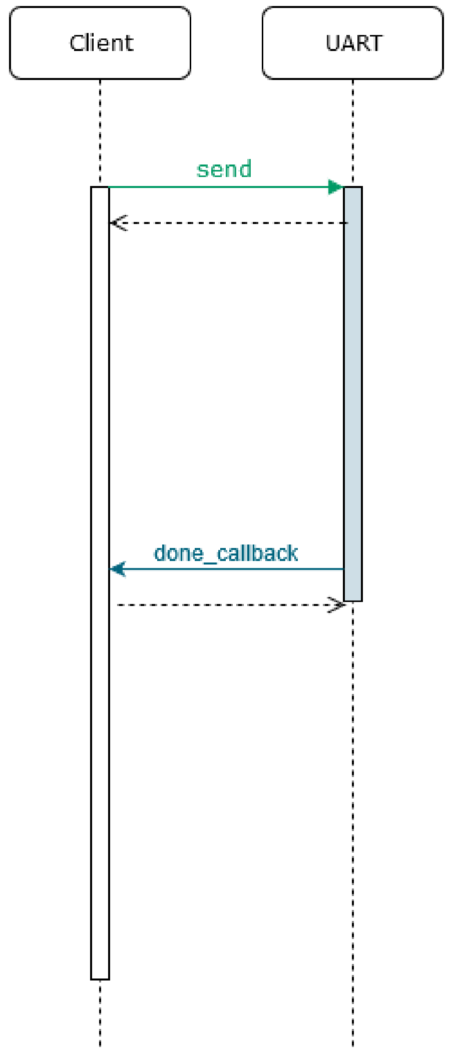
*Class template std::function is a general-purpose polymorphic function wrapper. Instances of std::function can store, copy, and **invoke any Callable** target -- **functions**, **lambda expressions**, **bind expressions**, or other **function objects**, as well as **pointers to member functions** and **pointers to data members**.*

https://en.cppreference.com/w/cpp/utility/functional/function

std::function - implementation

```cpp
struct UART {
  std::function<void()> done_callback;
  void Working() {
    // do the task....
    if(done_callback)
      done_callback();
  }

  void send(std::function<void()>& _callback) {
    done_callback = _callback;
    startWork();
  }
};
int main() {
  UART uart;
  //...
  auto callback = [](){/* do things */};
  uart.send(callback);
}
```



std::function with lambda – Resources

ARM Cortex-M4 32bit

- RAM
  - sizeof(std::function<>) – 16 bytes (gcc 7) + lambda Capture
- Flash
  - std::function
- Execution (CPU cycles)

	No Parameters	int (4 bytes)	struct (3 * int)
call	22	27	44
	No Capture	**Capture int**	**Capture struct**
modification	59	61	236

https://b.barebench.com/z/nrtyl3
https://b.barebench.com/z/Y9MQwp

I don’t need to modify at runtime, what can I do?

- Function pointer
- Interface
- std::function
- **Template parameter**
- …

Callback as Template parameter

```
struct Callback_t {
    void done_callback() {
    // do things...
    }
};


int main(){
    UART<Callback_t> uart;
    // ...
    uart.send();
}
```

Callback as Template parameter

```
template<typename Callback>
struct UART {
    void send() {
        startWorking();
    }

    void Working() {
        // doing the task
        // ....
        Callback::done_callback();
    }
};
```

```
template<typename Callback>
struct UART {
    void send() {
        startWorking();
    }

    void Working() {
        // doing the task
        // ....
        done_callback.done_callback();
    }
private:
    Callback &done_callback;
};
```

Template parameter – Resources

ARM Cortex-M4 32bit

- RAM
  - none
- Flash
  - none
- Execution (CPU cycles)

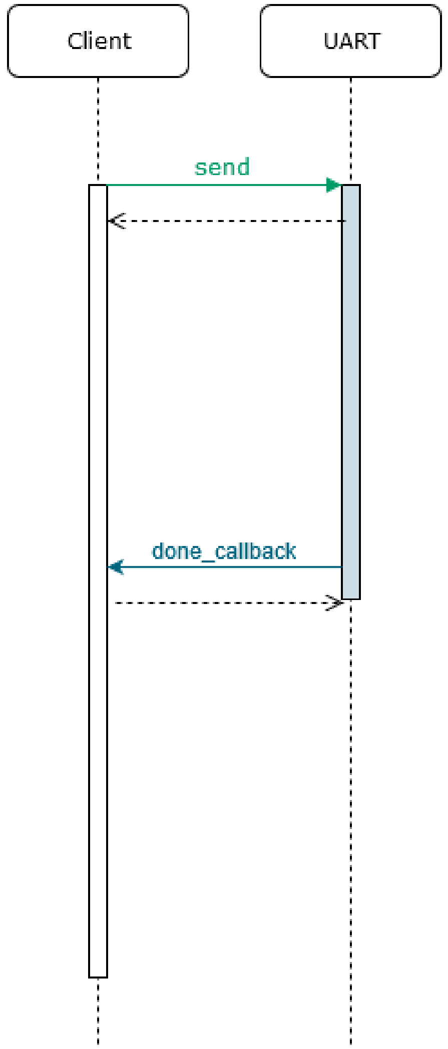
	No Parameters	int (4 bytes)	struct (3 * int)
call	0	0	0

function_ref – Proposal (P0792)

```
struct UART {
  function_ref<void()> done_callback = default_callback;
  void Working() {
    // do the task....
    done_callback();
  }

  void send(function_ref<void()> _callback) {
    done_callback = _callback;
    startWork();
  }
};

int main() {
  UART uart;
  //...
  auto callback = [](){/* do things */};
  uart.send(callback);
}
```



(std::)function_ref with lambda – Resources

ARM Cortex-M4 32bit

- RAM
  - sizeof(function_ref<>) - 8 bytes
- Flash
  - function_ref/none
- Execution (CPU cycles)

	No Parameters	int (4 bytes)	struct (3 * int)
call	20	21	51
	No Capture	**Capture int**	**Capture struct**
modification	11	11	11

https://b.barebench.com/z/PtCZFx
https://b.barebench.com/z/Z1EQm1

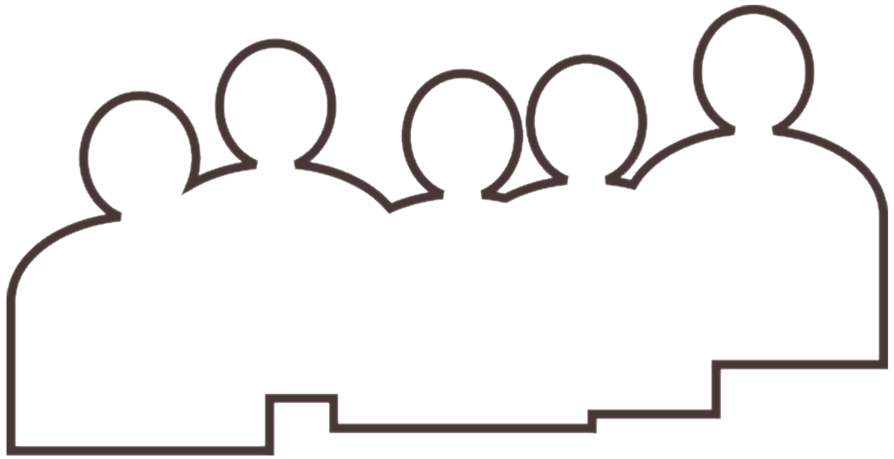
Conclusion

ARM Cortex-M4 32bit (GCC7)

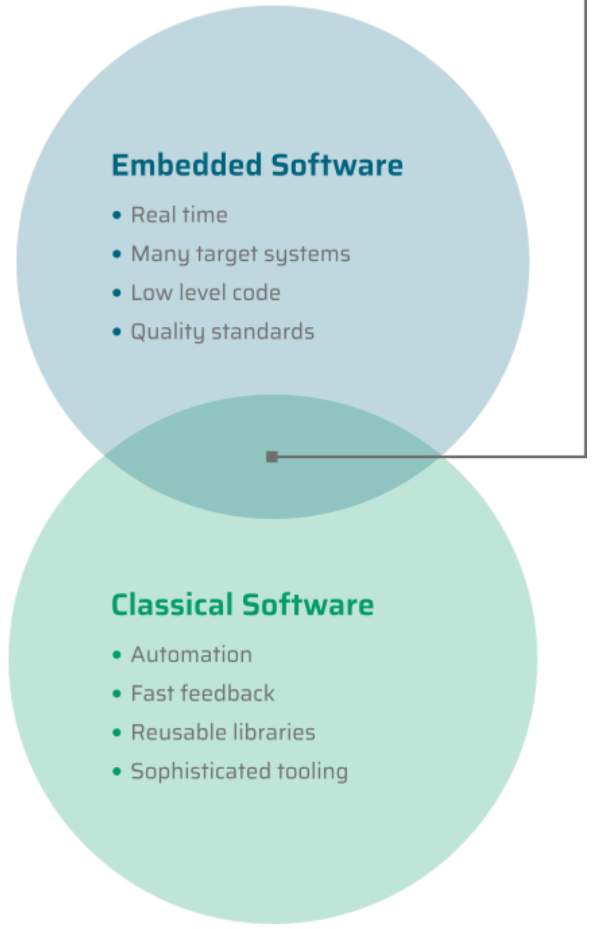
Depending on your use case,
you must choose the right solution for you.

Memory	Ram	Flash	Call without parameters (Cycles)	Modification at runtime (Cycles)
Function Pointer	4 Bytes	None	20	3
Interface	4 Bytes	vtable	25	3
std::function (fptr)	16 Bytes	std::function	27	62
std::function (lambda)	16 Bytes	std::function	22	59
std::function (lambda) + capture	16 Bytes + sizeof(capture)	std::function	44	236
function_ref	8 Bytes	none	20	11
Template Parameter	none	none	0 (15)	X

Thanks for your time!



Making high quality embedded software commonplace.


Embedded Software
• Real time
• Many target systems
• Low level code
• Quality standards
Classical Software
• Automation
• Fast feedback
• Reusable libraries
• Sophisticated tooling