

On The Move

Migrating from One RTOS to Another

Reasons for Migration

- RTOS availability for new chip architecture
- Customer demands
- Adherence to standards
- Change in technical requirements
- Change in commercial business model

Migration Issues

- How can migration of code be managed?
- What about migration of skills?
- What preparations can be made for this inevitable migration?

Code Migration

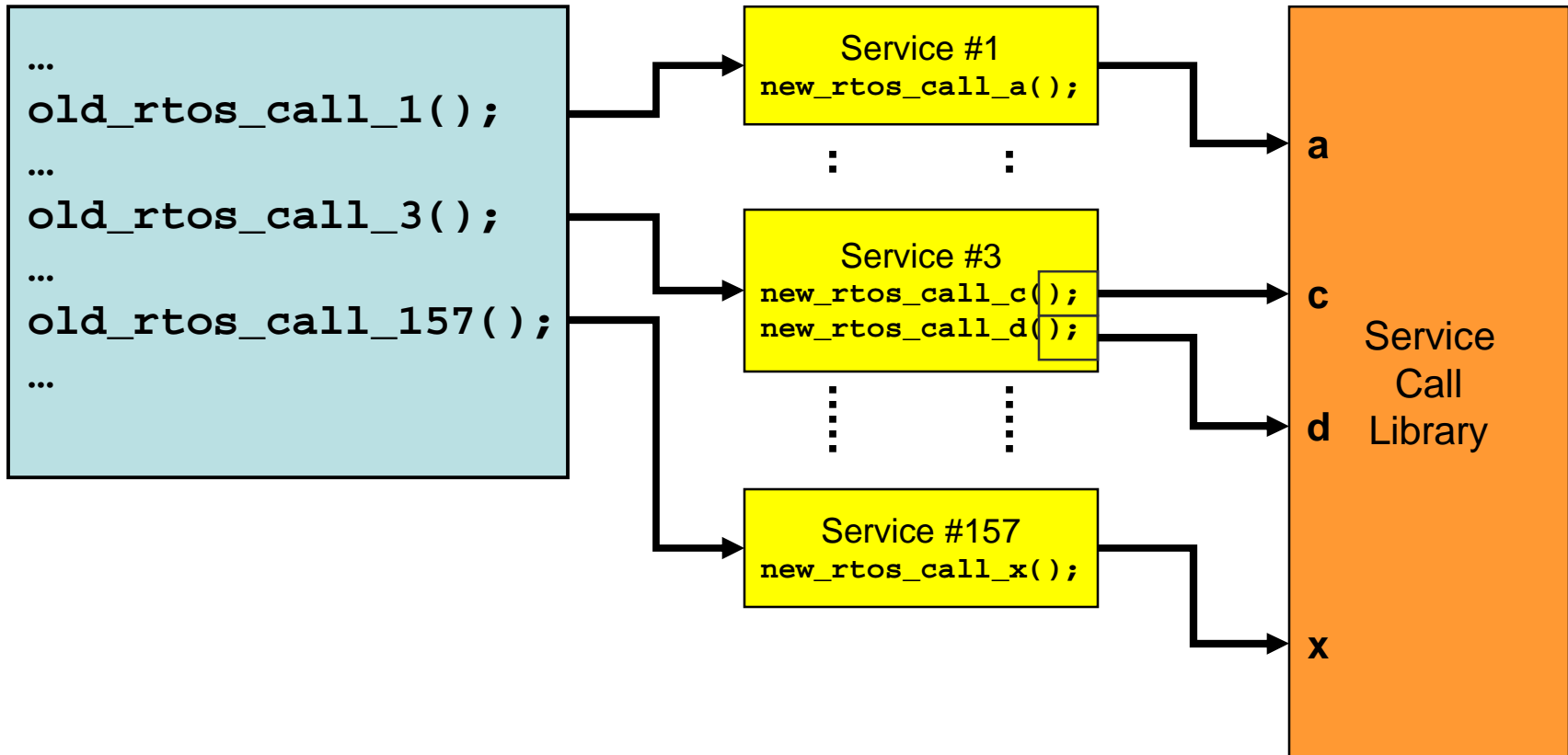
- Moving code from using one API to another
- May be done by hand
- Difficult will vary
 - similar APIs make it easier
 - richer target API is a benefit

Wrappers

Application

Wrapper

RTOS



Wrapper Strategies

- Map an old RTOS API onto a new one
- Design a “neutral” API and implement a wrapper for each selected RTOS
- Use a wrapper to implement a standard API on a proprietary RTOS

Wrapper Implementation

- Several approaches:
 - calls to underlying RTOS
 - may be optimized to jumps
 - C language `#define` macros
 - C++ classes
 - classes for RTOS objects
 - application-oriented classes which hide RTOS usage

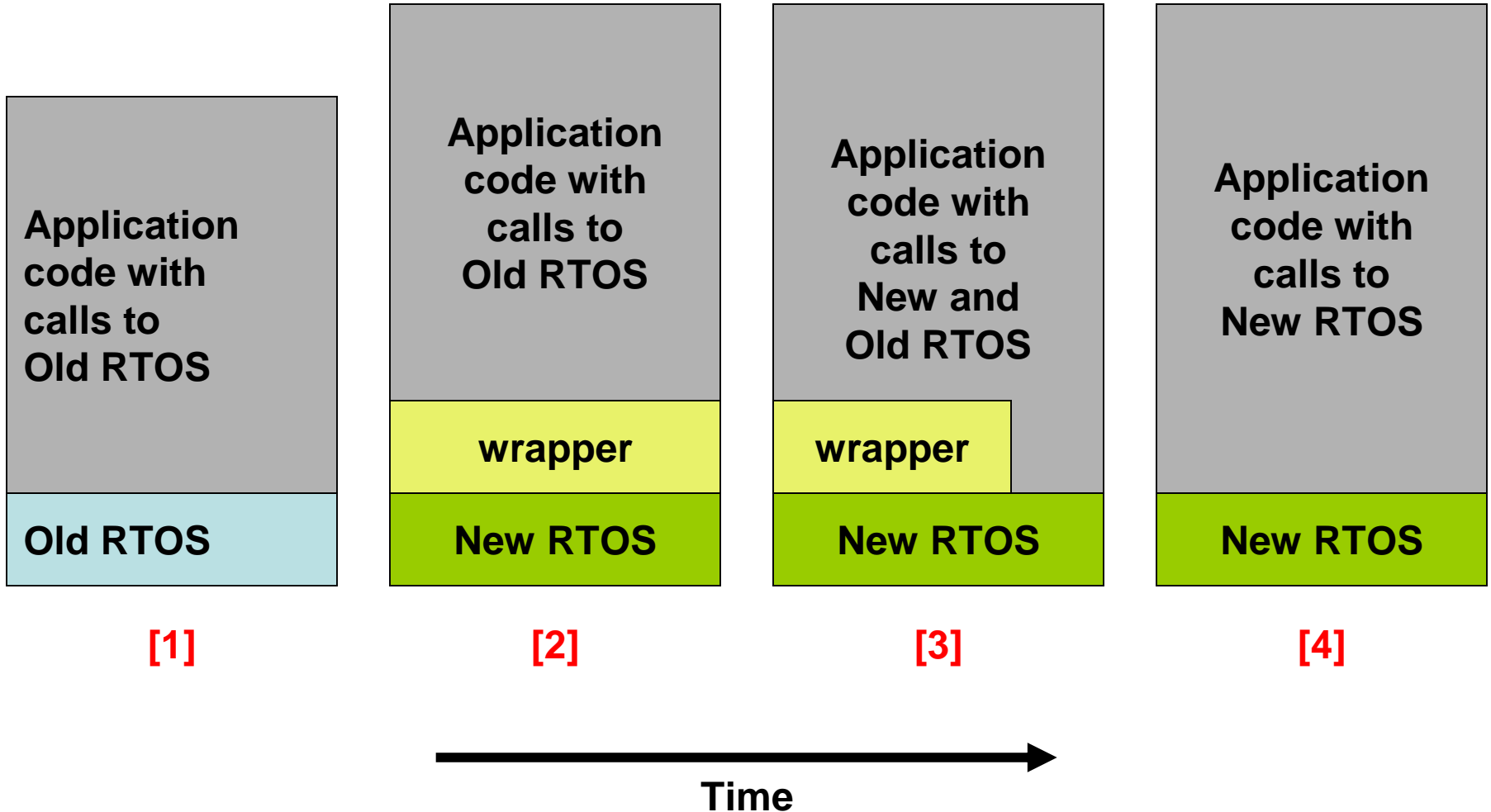
Wrapper Overheads

- Some time/memory overhead
 - except for pure macro approach
- Call-based approach carries call/return sequence time overhead
 - may be optimized by using jumps
 - memory minimized by using a library
- New RTOS may be more efficient
 - so overheads do not have an impact

Wrapper Challenges

- What kind of task identifier is used?
 - name, number, pointer ...
- What about priorities?
 - how many?
 - what mapping?
 - what direction?
 - start at 0 or 1?
- How are other facilities implemented?

API Availability



RTOS Standards

- POSIX
 - UNIX standard
- micro-ITRON
 - very strong in Japan
- OSEK/VDX
 - initially automotive/transportation
- Java
 - language or API?