

# *C++ Allocators for the Working Programmer*

This is simply a placeholder. Your production team will replace this page with the real series page.

# *C++ Allocators for the Working Programmer*

0.0. Identify if we do want a subtitle

*John Lakos*  
*Joshua Berne*

◆◆ **Addison-Wesley**

0.0. PH or AW?

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town  
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City  
Sao Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the United States, please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

*Library of Congress Cataloging-in-Publication Data*

**LIBRARY OF CONGRESS CIP DATA WILL GO HERE; MUST BE ALIGNED AS INDICATED BY LOC**

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearsoned.com/permissions/](http://www.pearsoned.com/permissions/).

ISBN-13: NUMBER HERE

ISBN-10: NUMBER HERE

Text printed in the United States on recycled paper at PRINTER INFO HERE.

First printing, MONTH YEAR

This is John’s dedication to Josh for being so great and  
writing this book so well.

JL

This is Josh’s dedication to his wife, child,  
and mother-in-law for being all supportive and  
wonderful. And to steak. Steak is great.

JMB



# Contents

<b>Foreword</b>	<b>ix</b>
<b>Preface</b>	<b>xi</b>
<b>Acknowledgements</b>	<b>xiii</b>
<b>About the Authors</b>	<b>xv</b>
<b>Chapter 1 Foundations</b>	<b>1</b>
1.1 Motivation	1
1.1.1 What is an Allocator	1
1.1.2 The History of C++ Allocators	1
1.1.3 What we’ll teach you about allocators	2
1.1.4 Making money with allocators	2
1.2 Technical Basics	2
1.2.1 C++ Allocators	2
1.2.2 The <code>std::pmr</code> Interface	2
<b>Chapter 2 Application Developers</b>	<b>3</b>
2.1 What is an Allocator-Aware Type?	3
2.1.1 Defining a PMR Allocator-Aware Type	3
2.1.2 <code>std::pmr</code> Collections	3
2.2 Using Allocator-Aware Types	3
2.2.1 How to use a Custom Memory Resource	3
2.2.2 How to Choose an Allocator	3
2.2.3 Testing Code that Allocates	3
2.3 Case Study 1: Unique Value Counting	3
<b>Chapter 3 Library Writers</b>	<b>4</b>
3.1 Writing Allocator-Aware Types	4
3.1.1 Aggregating Other Allocator-Aware Types	4
3.1.2 Doing Allocation	4
3.1.3 Testing Allocator-Aware Types	4
3.2 Case Study 3: PMR Optional and Variant	4
	vii

<b>Chapter 4 Writing Allocators</b>	<b>5</b>
4.1 Implementation	5
4.1.1 Learning from Global Allocators	5
4.1.2 Thread-Unsafe Allocators	5
4.1.3 Reuse Free Allocators	5
4.1.4 Wrapping Other Allocators for Utility	5
4.2 Benchmarking Allocators	5
4.3 Case Study 4: A Buffered Sequential Allocator	5
<b>Chapter 5 Making Money</b>	<b>6</b>
5.1 Optimizing existing software	6
5.1.1 Identifying short-lived objects	6
5.1.2 Replacing many allocations with few	6
5.2 Designing for allocator usage	6
5.2.1 Shaping tasks for allocators	6
5.2.2 Keeping allocators with subsystems	6
<b>Chapter 6 Advanced</b>	<b>7</b>
6.1 Modern Hardware	7
6.2 Effective Benchmarking	7
<b>Bibliography</b>	<b>9</b>
<b>Chapter A Other Libraries</b>	<b>11</b>
A.1 BDE	11
A.2 Thrust	11
<b>Chapter B Future Developments</b>	<b>12</b>
B.1 More PMR Types	12
B.2 Automating Allocator Supporter	12
<b>Todo list</b>	<b>13</b>



# Foreword

---

The text of the foreword will go here.



# Preface

---

The text of the preface will go here.



# Acknowledgements

---

The text of the author’s acknowledgements will go here.



## About the Authors

---



Author  
Photo  
here

John Lakos, author of *Large-Scale C++ Software Design* [Pearson, 1996] and *Large-Scale C++ — Volume I: Process and Architecture* [Pearson, 2019], serves at Bloomberg in New York City as a senior architect and mentor for C++ software development worldwide. He is also an active voting member of the C++ Standards Committee’s Evolution Working Group. From 1997 to 2001, Dr. Lakos directed the design and development of infrastructure libraries for proprietary analytic financial applications at Bear Stearns. From 1983 to 1997, Dr. Lakos was employed at Mentor Graphics, where he developed large frameworks and advanced ICCAD applications for which he holds multiple software patents. His academic credentials include a Ph.D.

in Computer Science (1997) and an Sc.D. in Electrical Engineering (1989) from Columbia University. Dr. Lakos received his undergraduate degrees from MIT in Mathematics (1982) and Computer Science (1981).



Author  
Photo  
here

Joshua Berne serves at Bloomberg LP as a senior software engineer on Bloomberg’s core library team. After the difficult choice to pursue a career in software engineering over research mathematics, he has been an active programmer in the financial industry, writing day trading applications in C++ for E\*TRADE Capital Markets and, after that, architecting large distributed trading systems in Java for Instinet and IDC. Since joining Bloomberg in 2017, he has been an active participant in the C++ Standards Committee, seeking to bring the advancements made within Bloomberg to the C++ Standard and thus to the rest of the world. His first WG21 paper was [1]





# Chapter 1

## Foundations

---

### 1.1 Motivation

- Why local allocation can help
- Limits of global allocation
- Reference Emery’s paper?

#### 1.1.1 What is an Allocator

- an allocator allocates and deallocates memory
- what is a “general purpose” allocator
  - same contract and requirements as new/delete malloc/free
  - thread-safe - allocate concurrently, deallocate from any thread
  - objects of any size
  - overhead constant in terms of currently allocated memory
- types of “special purpose” allocators
  - Unsynchronized
  - Monotonic
- Global vs. Local allocators
  - global allocators can be specialized
  - local allocators can be general purpose

#### 1.1.2 The History of C++ Allocators

- describe C++03 allocators
- [2] - Towards a better allocator model
- [3], [4],
- Scoped allocators: [5], [6],
- c++17, c++20 changes to PMR

### **1.1.3 What we’ll teach you about allocators**

- Summary of what each chapter will teach

### **1.1.4 Making money with allocators**

- Summary of how architecture can facilitate leveraging allocators

---

## **1.2 Technical Basics**

### **1.2.1 C++ Allocators**

- Go over the mess of c++03 style allocator types

### **1.2.2 The `std::pmr` Interface**

- Show `std::pmr::polymorphic_allocator`
- Show simplification to `ALLOCATOR` types
- Show the `memory_resource` interface, how to do an allocation

# Chapter 2

## Application Developers

---

---

### 2.1 What is an Allocator-Aware Type?

#### 2.1.1 Defining a PMR Allocator-Aware Type

#### 2.1.2 `std::pmr` Collections

---

### 2.2 Using Allocator-Aware Types

#### 2.2.1 How to use a Custom Memory Resource

#### 2.2.2 How to Choose an Allocator

#### 2.2.3 Testing Code that Allocates

---

### 2.3 Case Study 1: Unique Value Counting

# Chapter 3

## Library Writers

---

---

### 3.1 Writing Allocator-Aware Types

#### 3.1.1 Aggregating Other Allocator-Aware Types

#### 3.1.2 Doing Allocation

#### 3.1.3 Testing Allocator-Aware Types

---

### 3.2 Case Study 3: PMR Optional and Variant

# Chapter 4

## Writing Allocators

---

---

### 4.1 Implementation

- 4.1.1 Learning from Global Allocators
- 4.1.2 Thread-Unsafe Allocators
- 4.1.3 Reuse Free Allocators
- 4.1.4 Wrapping Other Allocators for Utility

---

### 4.2 Benchmarking Allocators

---

### 4.3 Case Study 4: A Buffered Sequential Allocator

# Chapter 5

## Making Money

---

### 5.1 Optimizing existing software

#### 5.1.1 Identifying short-lived objects

- escape analysis
- recursive functions
- Automated tooling to help discover?

#### 5.1.2 Replacing many allocations with few

- identify
- 

### 5.2 Designing for allocator usage

#### 5.2.1 Shaping tasks for allocators

- Differentiating between long and short lived data.
- Message processing in local allocators, updating persistent state in global allocator
- Structuring persistent data for advantageous cache usage

#### 5.2.2 Keeping allocators with subsystems

- Moving allocators with their data - queues of smart pointers,

# Chapter 6

## Advanced

---

---

### 6.1 Modern Hardware

---

### 6.2 Effective Benchmarking

---

Here we would be discussing the approach we have to benchmarking.

```
struct S {  
    void foo();  
};  
  
void S::foo()  
{  
}
```

6.2. Determine a better location for benchmarking section





# Bibliography

---

- [1] Joshua Berne, Nathan Burgers, Hyman Rosen, John Lakos, “Contract checking in c++: A (long-term) road map,” Tech. Rep. P1332R0, WG21 - The C++ Standards Committee, 2018.
- [2] Pablo Halpern, “Towards a better allocator model,” Tech. Rep. N1850, WG21 - The C++ Standards Committee, 2005.
- [3] Pablo Halpern, “Omnibus allocator fix-up proposals,” Tech. Rep. N2387, WG21 - The C++ Standards Committee, 2007.
- [4] Pablo Halpern, “Small allocator fix-ups,” Tech. Rep. N2436, WG21 - The C++ Standards Committee, 2007.
- [5] Pablo Halpern, “The scoped allocator model,” Tech. Rep. N2446, WG21 - The C++ Standards Committee, 2007.
- [6] Pablo Halpern, “The scoped allocator model (rev 1),” Tech. Rep. N2523, WG21 - The C++ Standards Committee, 2008.



# Appendix A

## Other Libraries

---

---

### A.1 BDE

---

### A.2 Thrust

# Appendix B

## Future Developments

---

---

### B.1 More PMR Types

---

### B.2 Automating Allocator Suppoer

---

# Todo list

---

0.0. Identify if we do want a subtitle . . . . .	iii
0.0. PH or AW? . . . . .	iii
6.2. Determine a better location for benchmarking section . . . . .	7