Michalis Kokologiannakis, Ralf Jung and Peter Müller

RESEARCH TOPICS IN SOFTWARE ENGINEERING

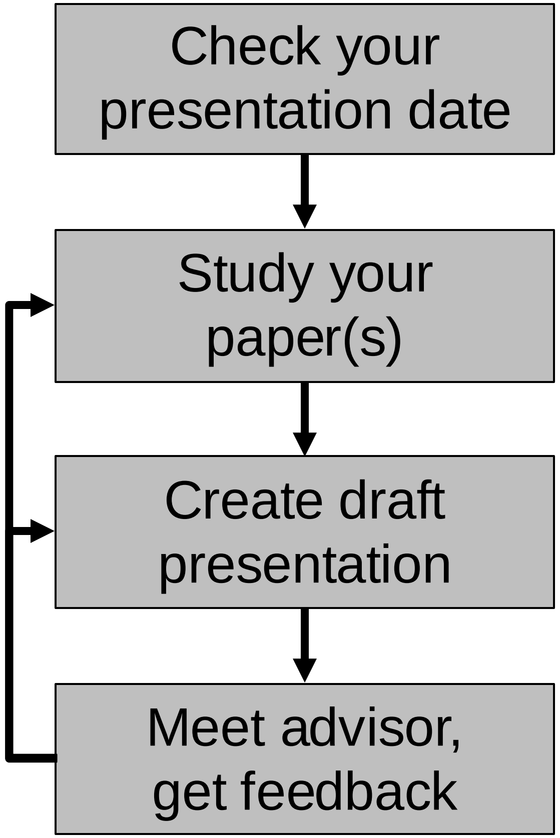
ETH zürich

Autumn 2024

Objectives

- Learn how to present technical work
- Learn how to understand and evaluate research papers
- Learn about key research directions in the area

Preparing a Talk



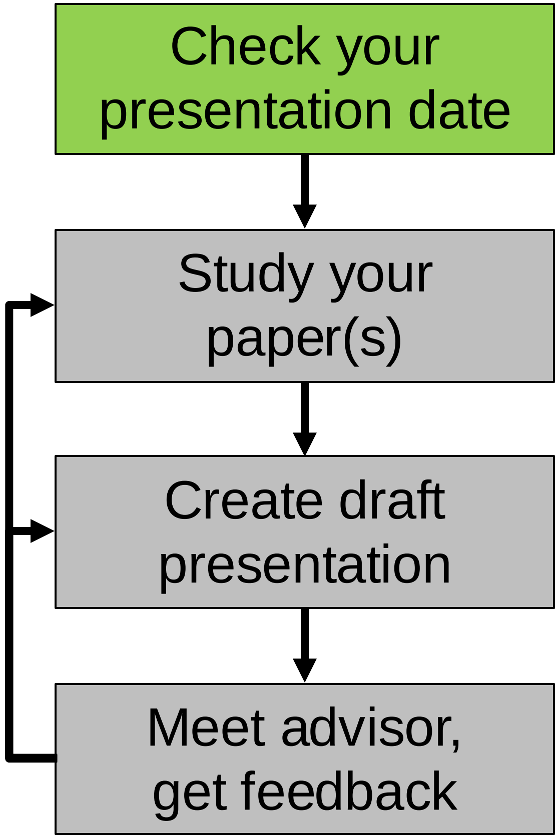
Check your presentation date

Study your paper(s)

Create draft presentation

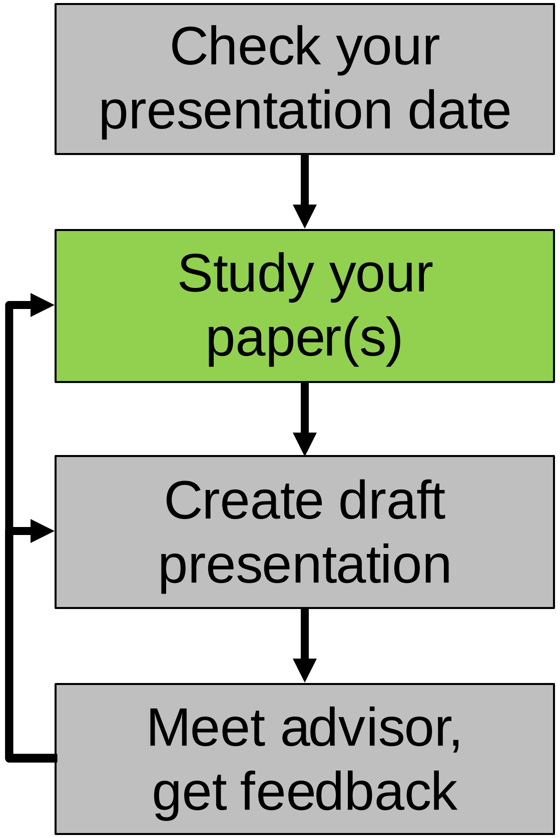
Meet advisor, get feedback

Preparing a Talk: Start Early


Check your presentation date
Study your paper(s)
Create draft presentation
Meet advisor, get feedback

- Preparing a good presentation takes time
- Start early!

Preparing a Talk: Study Paper


Check your presentation date
Study your paper(s)
Create draft presentation
Meet advisor, get feedback

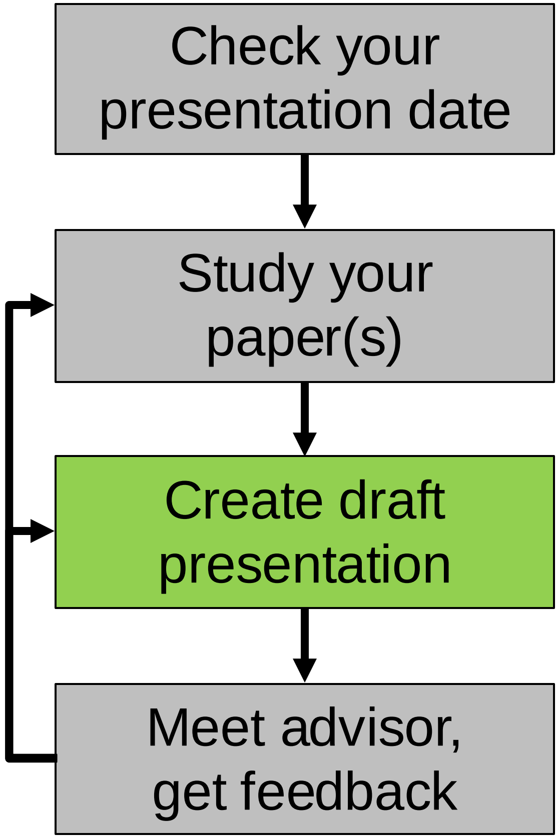
- 3 ‘C’s of reading
  - Carefully: look up terms, possibly read cited papers
  - Critically: find limitations, flaws
  - Creatively: think of improvements

- Try examples by hand

- Try tools if available

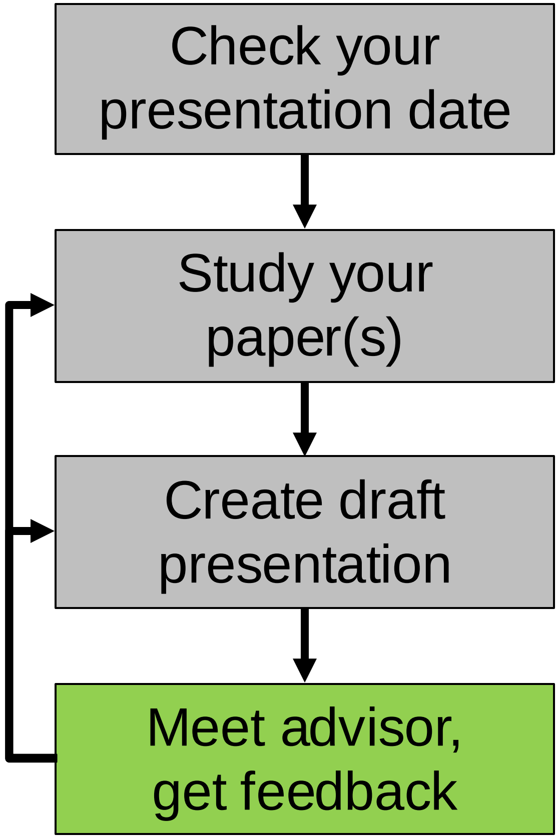
- Consult with TA if questions

Preparing a Talk: Create Draft


Check your presentation date
Study your paper(s)
Create draft presentation
Meet advisor, get feedback

- Explain the motivation for the work
- Clearly present the technical solution and results
  - Use your own example, not the one in the paper
  - Include a demo if appropriate
- Outline limitations or improvements
- Focus on the key concepts
  - Do not present all of the details

Preparing a Talk: Get Feedback


Check your presentation date
Study your paper(s)
Create draft presentation
Meet advisor, get feedback

- Prepare for the meeting
  - Schedule early
  - Send slides in advance
  - Write down questions
- Make sure you address feedback
  - Take notes
- Meeting is mandatory!
  - At least one week before the talk

Grading

- Presentation
  - Understanding of the paper and its context
  - Structure and content
  - Presentation style (speech, slides, visualization, own examples)
  - Discussion

- Participation
  - Did you ask good questions?
  - Did you attend all sessions?

- We will also take into account:
  - the difficulty of the paper
  - suggestions you received from your TA
  - time you had to prepare

Feedback

- We will discuss strengths and weaknesses of your talk in class
  - Let us know upfront if you’d prefer not to

- Arrange a meeting with your TA to get detailed feedback

Schedule

- We will meet once a week, with two presentations per session
  - Next meeting on October 8 (no session the next two weeks!)
  - 22 presentations in total

- Detailed schedule will be published online shortly
  - https://pls.inf.ethz.ch/education/Research_Topics_in_Software_Engineering.html
  - Including names of teaching assistants

Your Talk: Timing

- Your talk should be 30 minutes plus discussion
- 1.5 – 2 minutes per slide
- The pace of your talk is important
  - If you are too fast, the audience cannot follow
  - If you are too slow, people get bored
- Practice your talk
  - Track a checkpoint after circa 10 minutes

Your Talk: Structure

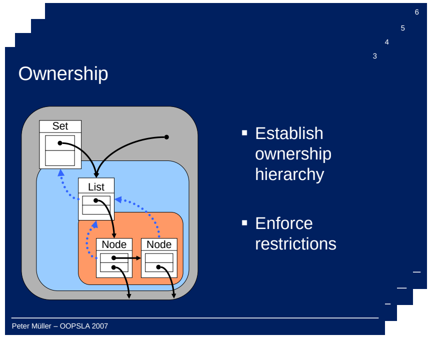
Ownership Transfer in Universe Types

Peter Müller
Microsoft Research
USA

Arsenii Rudich
ETH Zurich
Switzerland

Title slide

Splash



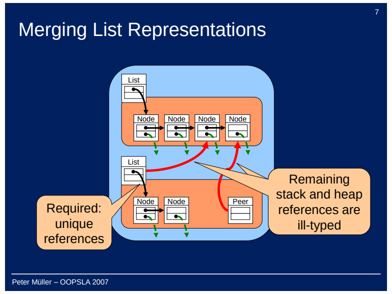
6
5
4
3

Ownership

- Establish ownership hierarchy
- Enforce restrictions

Peter Müller – OOPSLA 2007

Motivation, background



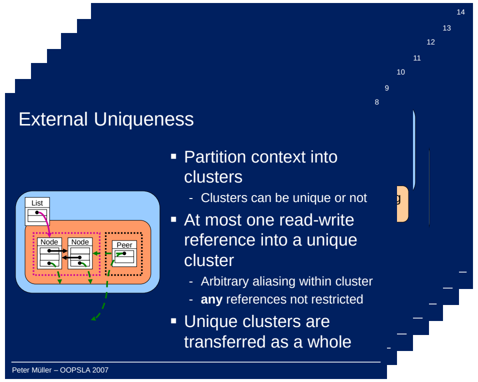
7

Merging List Representations

List
Node
Node
Node
Node
List
Node
Node
Peer
Required: unique references
Remaining stack and heap references are ill-typed

Peter Müller – OOPSLA 2007

Problem



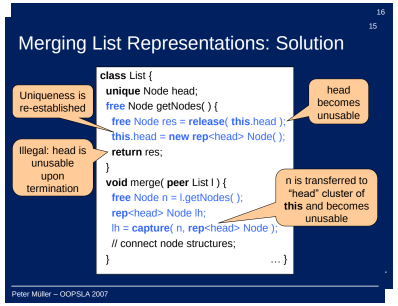
14
13
12
11
10
9
8

External Uniqueness

- Partition context into clusters
  - Clusters can be unique or not
- At most one read-write reference into a unique cluster
  - Arbitrary aliasing within cluster
  - **any** references not restricted
- Unique clusters are transferred as a whole

Peter Müller – OOPSLA 2007

Solution



16
15

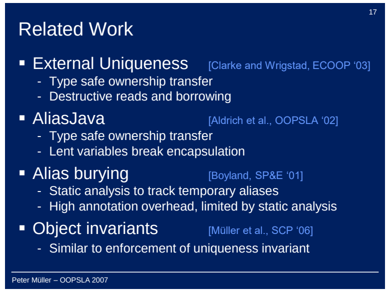
Merging List Representations: Solution

```
class List {
  unique Node head;
  free Node getNodes( ) {
    free Node res = release( this.head );
    this.head = new rep<head> Node( );
    return res;
  }
  void merge( peer List l ) {
    free Node n = l.getNodes( );
    rep<head> Node lh;
    lh = capture( n, rep<head> Node );
    // connect node structures;
  }                                  … }
```

Uniqueness is re-established
Illegal: head is unusable upon termination
head becomes unusable
n is transferred to "head" cluster of this and becomes unusable

Peter Müller – OOPSLA 2007

Evaluation, experiments, demo



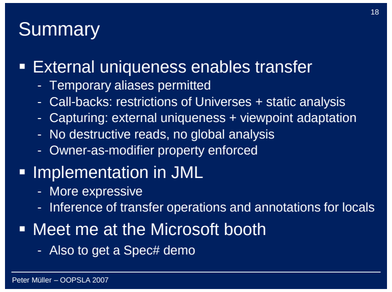
17

Related Work

- External Uniqueness [Clarke and Wrigstad, ECOOP '03]
  - Type safe ownership transfer
  - Destructive reads and borrowing
- AliasJava [Aldrich et al., OOPSLA '02]
  - Type safe ownership transfer
  - Lent variables break encapsulation
- Alias burying [Boyland, SP&E '01]
  - Static analysis to track temporary aliases
  - High annotation overhead, limited by static analysis
- Object invariants [Müller et al., SCP '06]
  - Similar to enforcement of uniqueness invariant

Peter Müller – OOPSLA 2007

Related work



18

Summary

- External uniqueness enables transfer
  - Temporary aliases permitted
  - Call-backs: restrictions of Universes + static analysis
  - Capturing: external uniqueness + viewpoint adaptation
  - No destructive reads, no global analysis
  - Owner-as-modifier property enforced
- Implementation in JML
  - More expressive
  - Inference of transfer operations and annotations for locals
- Meet me at the Microsoft booth
  - Also to get a Spec# demo

Peter Müller – OOPSLA 2007

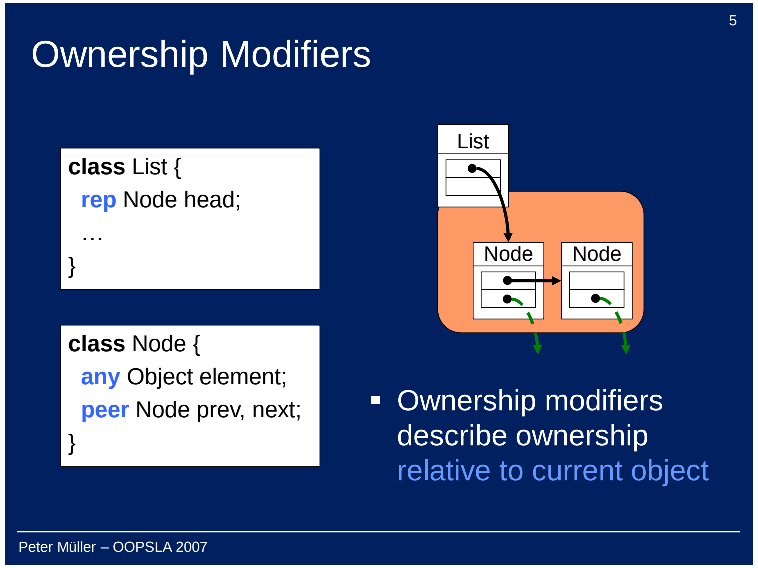
Summary, conclusions

Your Talk: Examples

- Examples are crucial for the understanding
  - Yours and the audience’s
  - Prepare your own example!

- Try to find a running example
  - For motivation, problem, and solution
  - Explain in detail (takes time)

- Reduce code example to the absolute necessary
  - Most people hate reading code
  - Use visualizations



5

Ownership Modifiers

```
class List {
  rep Node head;
  …
}
```

```
class Node {
  any Object element;
  peer Node prev, next;
}
```

- Ownership modifiers describe ownership relative to current object

Peter Müller – OOPSLA 2007

Your Talk: Design

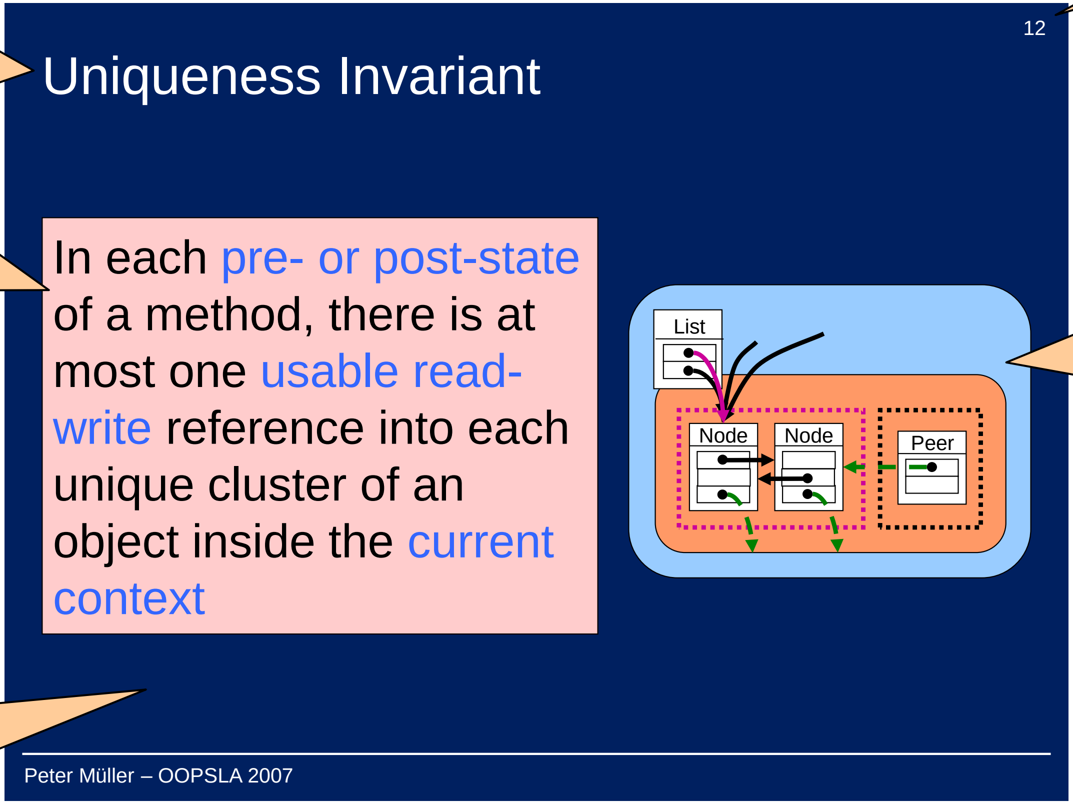
Include slide numbers

Use a descriptive title

Use a large font (at least 18pt)

Use visualizations

Do not overload slide

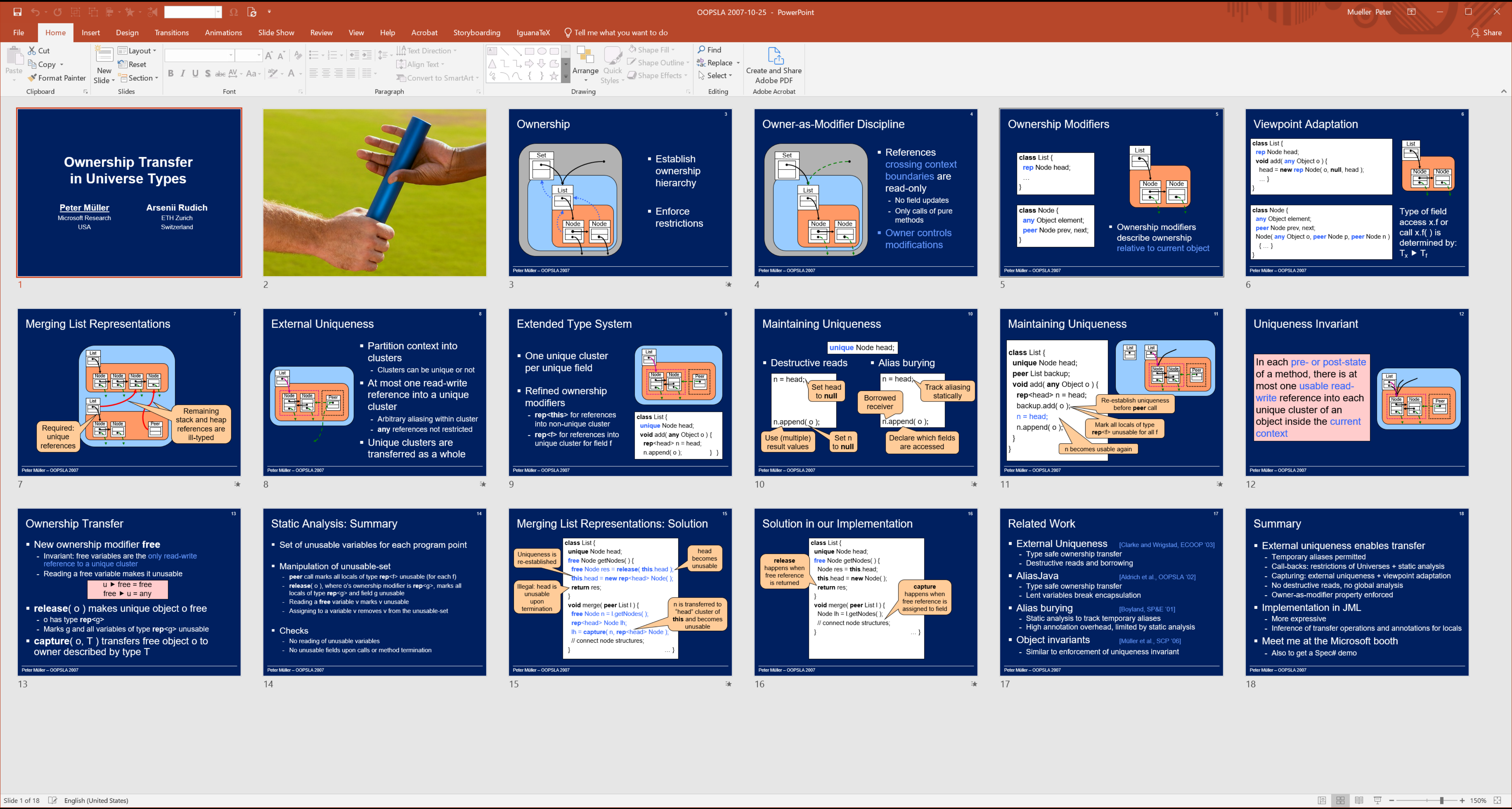


12

Uniqueness Invariant

In each pre- or post-state of a method, there is at most one usable read-write reference into each unique cluster of an object inside the current context

Peter Müller – OOPSLA 2007



1

Ownership Transfer in Universe Types

Peter Müller
Microsoft Research
USA

Arsenii Rudich
ETH Zurich
Switzerland

2

3

Ownership

- Establish ownership hierarchy
- Enforce restrictions

Peter Müller – OOPSLA 2007

4

Owner-as-Modifier Discipline

- References crossing context boundaries are read-only
  - No field updates
  - Only calls of pure methods
- Owner controls modifications

Peter Müller – OOPSLA 2007

5

Ownership Modifiers

```
class List {
  rep Node head;
  …
}
```

```
class Node {
  any Object element;
  peer Node prev, next;
}
```

- Ownership modifiers describe ownership relative to current object

Peter Müller – OOPSLA 2007

6

Viewpoint Adaptation

```
class List {
  rep Node head;
  void add( any Object o ) {
   head = new rep Node( o, null, head );
   … }
}
```

```
class Node {
  any Object element;
  peer Node prev, next;
  Node( any Object o, peer Node p, peer Node n )
   { … }
}
```

Type of field access x.f or call x.f( ) is determined by: $T_x \blacktriangleright T_f$

Peter Müller – OOPSLA 2007

7

Merging List Representations

Required: unique references

Remaining stack and heap references are ill-typed

Peter Müller – OOPSLA 2007

8

External Uniqueness

- Partition context into clusters
  - Clusters can be unique or not
- At most one read-write reference into a unique cluster
  - Arbitrary aliasing within cluster
  - any references not restricted
- Unique clusters are transferred as a whole

Peter Müller – OOPSLA 2007

9

Extended Type System

- One unique cluster per unique field
- Refined ownership modifiers
  - **rep<this>** for references into non-unique cluster
  - **rep<f>** for references into unique cluster for field f

```
class List {
  unique Node head;
  void add( any Object o ) {
   rep<head> n = head;
   n.append( o );           } }
```

Peter Müller – OOPSLA 2007

10

Maintaining Uniqueness

```
unique Node head;
```

- Destructive reads
- Alias burying

```
n = head;

n.append( o );
```

Set head to **null**

Use (multiple) result values

Set n to **null**

```
n = head;

n.append( o );
```

Track aliasing statically

Borrowed receiver

Declare which fields are accessed

Peter Müller – OOPSLA 2007

11

Maintaining Uniqueness

```
class List {
 unique Node head;
 peer List backup;
 void add( any Object o ) {
  rep<head> n = head;
  backup.add( o );
  n = head;
  n.append( o );
 }
}
```

Re-establish uniqueness before **peer** call

Mark all locals of type **rep**<f> unusable for all f

n becomes usable again

Peter Müller – OOPSLA 2007

12

Uniqueness Invariant

In each pre- or post-state of a method, there is at most one usable read-write reference into each unique cluster of an object inside the current context

Peter Müller – OOPSLA 2007

13

Ownership Transfer

- New ownership modifier **free**
  - Invariant: free variables are the only read-write reference to a unique cluster
  - Reading a free variable makes it unusable

$u \blacktriangleright free = free$
$free \blacktriangleright u = any$

- **release**(o) makes unique object o free
  - o has type **rep**<g>
  - Marks g and all variables of type **rep**<g> unusable
- **capture**(o, T) transfers free object o to owner described by type T

Peter Müller – OOPSLA 2007

14

Static Analysis: Summary

- Set of unusable variables for each program point
- Manipulation of unusable-set
  - **peer** call marks all locals of type **rep**<f> unusable (for each f)
  - **release**(o), where o's ownership modifier is **rep**<g>, marks all locals of type **rep**<g> and field g unusable
  - Reading a **free** variable v marks v unusable
  - Assigning to a variable v removes v from the unusable-set
- Checks
  - No reading of unusable variables
  - No unusable fields upon calls or method termination

Peter Müller – OOPSLA 2007

15

Merging List Representations: Solution

```
class List {
 unique Node head;
 free Node getNodes( ) {
  free Node res = release( this.head );
  this.head = new rep<head> Node( );
  return res;
 }
 void merge( peer List l ) {
  free Node n = l.getNodes( );
  rep<head> Node lh;
  lh = capture( n, rep<head> Node );
  // connect node structures;
 }                                    … }
```

Uniqueness is re-established

Illegal: head is unusable upon termination

head becomes unusable

n is transferred to "head" cluster of **this** and becomes unusable

Peter Müller – OOPSLA 2007

16

Solution in our Implementation

```
class List {
 unique Node head;
 free Node getNodes( ) {
  Node res = this.head;
  this.head = new Node( );
  return res;
 }
 void merge( peer List l ) {
  Node lh = l.getNodes( );
  // connect node structures;
 }                                    … }
```

release happens when free reference is returned

capture happens when free reference is assigned to field

Peter Müller – OOPSLA 2007

17

Related Work

- External Uniqueness [Clarke and Wrigstad, ECOOP '03]
  - Type safe ownership transfer
  - Destructive reads and borrowing
- AliasJava [Aldrich et al., OOPSLA '02]
  - Type safe ownership transfer
  - Lent variables break encapsulation
- Alias burying [Boyland, SP&E '01]
  - Static analysis to track temporary aliases
  - High annotation overhead, limited by static analysis
- Object invariants [Müller et al., SCP '06]
  - Similar to enforcement of uniqueness invariant

Peter Müller – OOPSLA 2007

18

Summary

- External uniqueness enables transfer
  - Temporary aliases permitted
  - Call-backs: restrictions of Universes + static analysis
  - Capturing: external uniqueness + viewpoint adaptation
  - No destructive reads, no global analysis
  - Owner-as-modifier property enforced
- Implementation in JML
  - More expressive
  - Inference of transfer operations and annotations for locals
- Meet me at the Microsoft booth
  - Also to get a Spec# demo

Peter Müller – OOPSLA 2007

Powerpoint vs. Latex

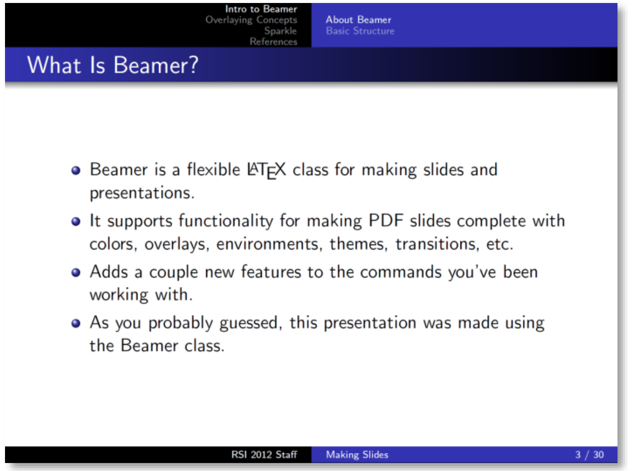
Powerpoint

- Visualizations and animations are easy
- Don’t over-do it!

Creating Animations is Fun

Latex

- Visualizations and animations are painful
- Don’t under-do it!



Intro to Beamer
Overlaying Concepts
Sparkle
References

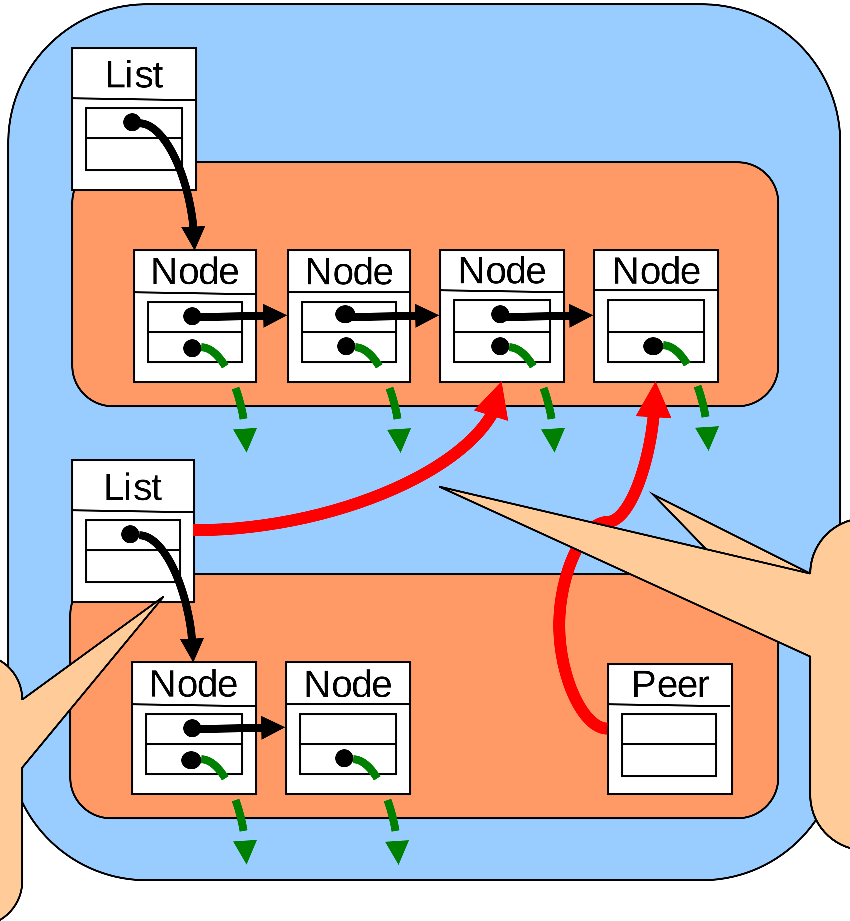
About Beamer
Basic Structure

What Is Beamer?

- Beamer is a flexible LATEX class for making slides and presentations.
- It supports functionality for making PDF slides complete with colors, overlays, environments, themes, transitions, etc.
- Adds a couple new features to the commands you've been working with.
- As you probably guessed, this presentation was made using the Beamer class.

RSI 2012 Staff Making Slides 3 / 30

Merging List Representations



Required: unique references

Remaining stack and heap references are ill-typed

Your Talk: Avoid Frequent Mistakes

- Don’t try to present all details
  - Focus on a few key messages:
    Motivation, problem, main idea, main result

- Don’t stare at the screen or your laptop
  - Look at the audience

- Come prepared
  - Study paper in depth
  - Rehearse your talk (but not too much)

References

- We strongly recommend studying Markus Püschel’s small guide to giving presentations: http://www.inf.ethz.ch/personal/markusp/teaching/guides/guide-presentations.pdf
- We also recommend this presentation by Derek Dreyer on "How to give talks that people can follow": https://www.youtube.com/watch?v=TCytsY8pdsc

What should I do next?

- Look at the list of available papers.

- Send us your top 5 choices via the selection form.

- If you do not submit your selection by Sept. 18 (tomorrow), we will assume that you are no longer interested in taking the seminar!