MICROSOFT RESEARCH INSTITUTE

MACQUARIE UNIVERSITY~SYDNEY

Flexible Alias Protection

James Noble, Jan Vitek*, John Potter
Microsoft Research Institute,
Macquarie University, Sydney,
and *Université de Genève, Geneva

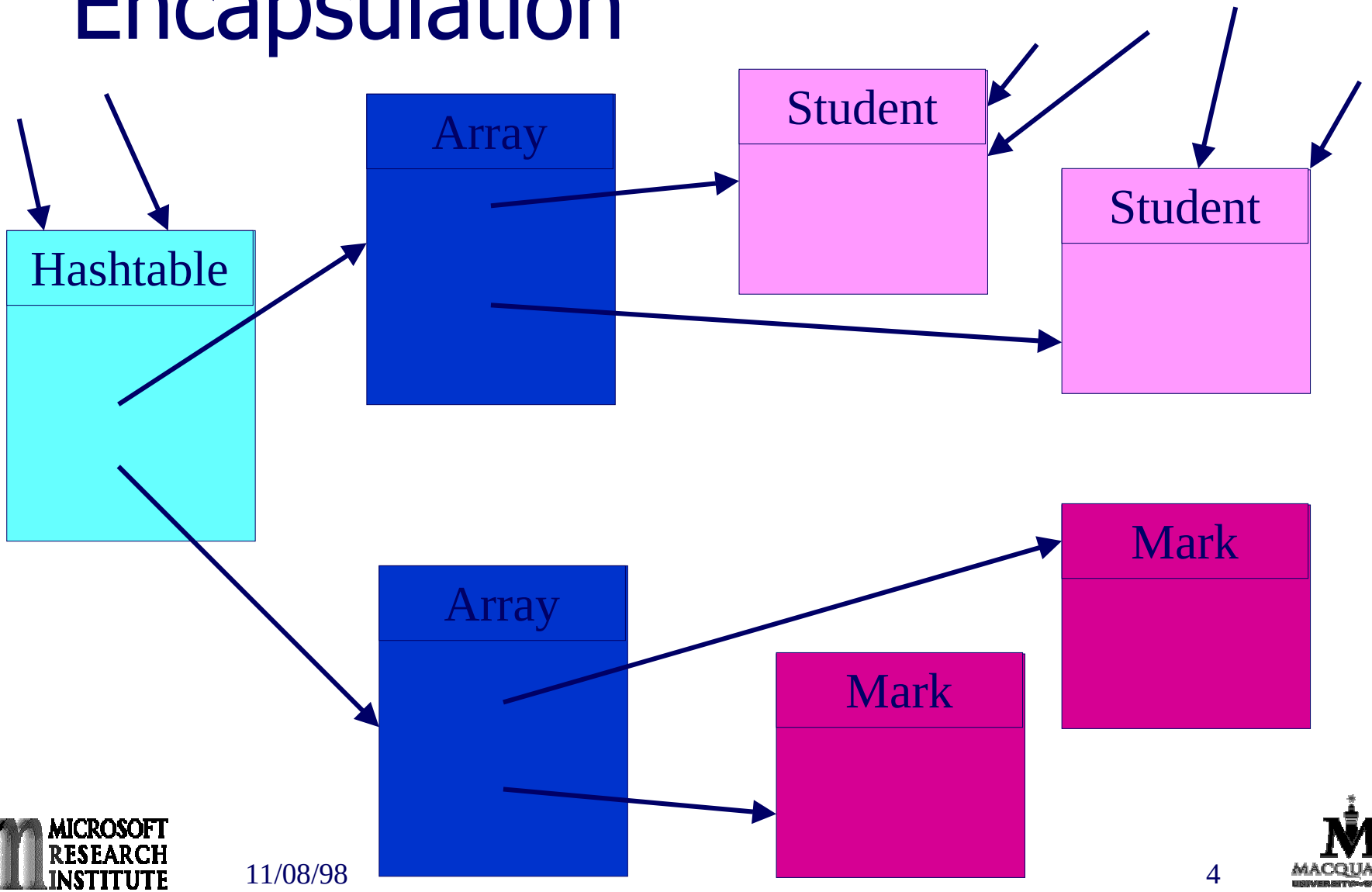
Flexible Alias Protection

- Objects, Aliasing, and Encapsulation
- Flexible Aliasing Protection
- A Static Aliasing Mode System

Objects, Aliasing, Encapsulation

- Object Identity
  - The foundation of OO programming
- Objects have mutable state
- Uniform extent of object reference
- Result: **Endemic Aliasing**
  - **Aggregate objects are composed from other objects**
  - **Object invariants can be breached**

Objects, Aliasing, Encapsulation



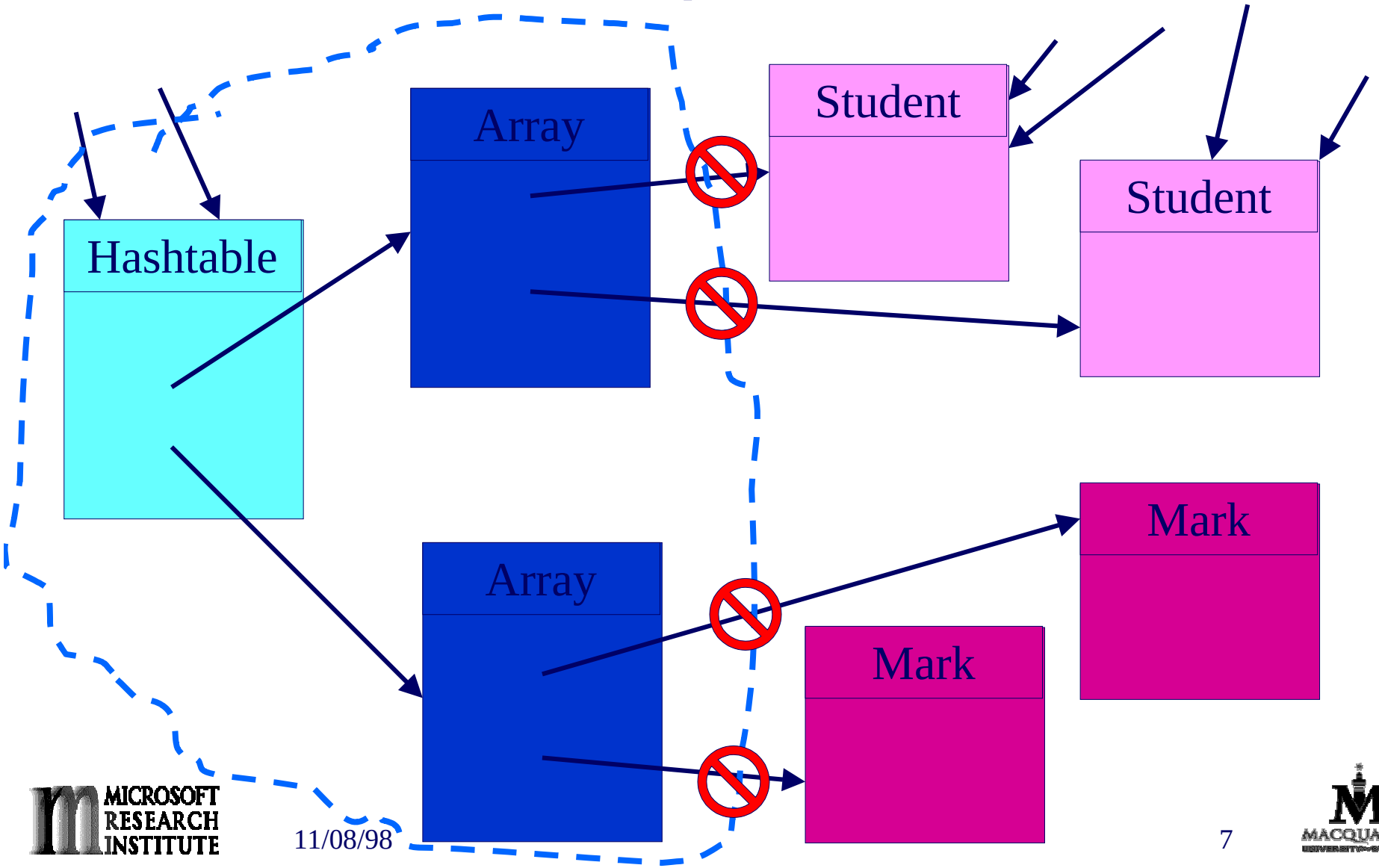
Controlling Aliasing

- Access specifiers — *private, expanded*
  - protect names not objects
  - per class not per object
  - restrict dynamic structures and subtyping
- Research
  - compiler analysis (since FORTRAN)
  - access objects via paths, not pointers
  - demesnes, linearity, swapping, copying…

Full Alias Encapsulation

- A number of mechanisms
  - Islands — Hogg OOPSLA'91
  - Balloons — Almeida ECOOP'97
- Complete encapsulation of aliasing
  - Prevent external references into aggregate
  - Prevent aggregates referring to other objects
    - one object cannot belong to two collections
- Loopholes via dynamic aliases

Full Alias Encapsulation



Flexible Alias Protection

- Problem:
  - Language protection is too weak
  - Full Alias Encapsulation is too strong

- Permit benign forms of aliasing
- Restrict problematic aliasing

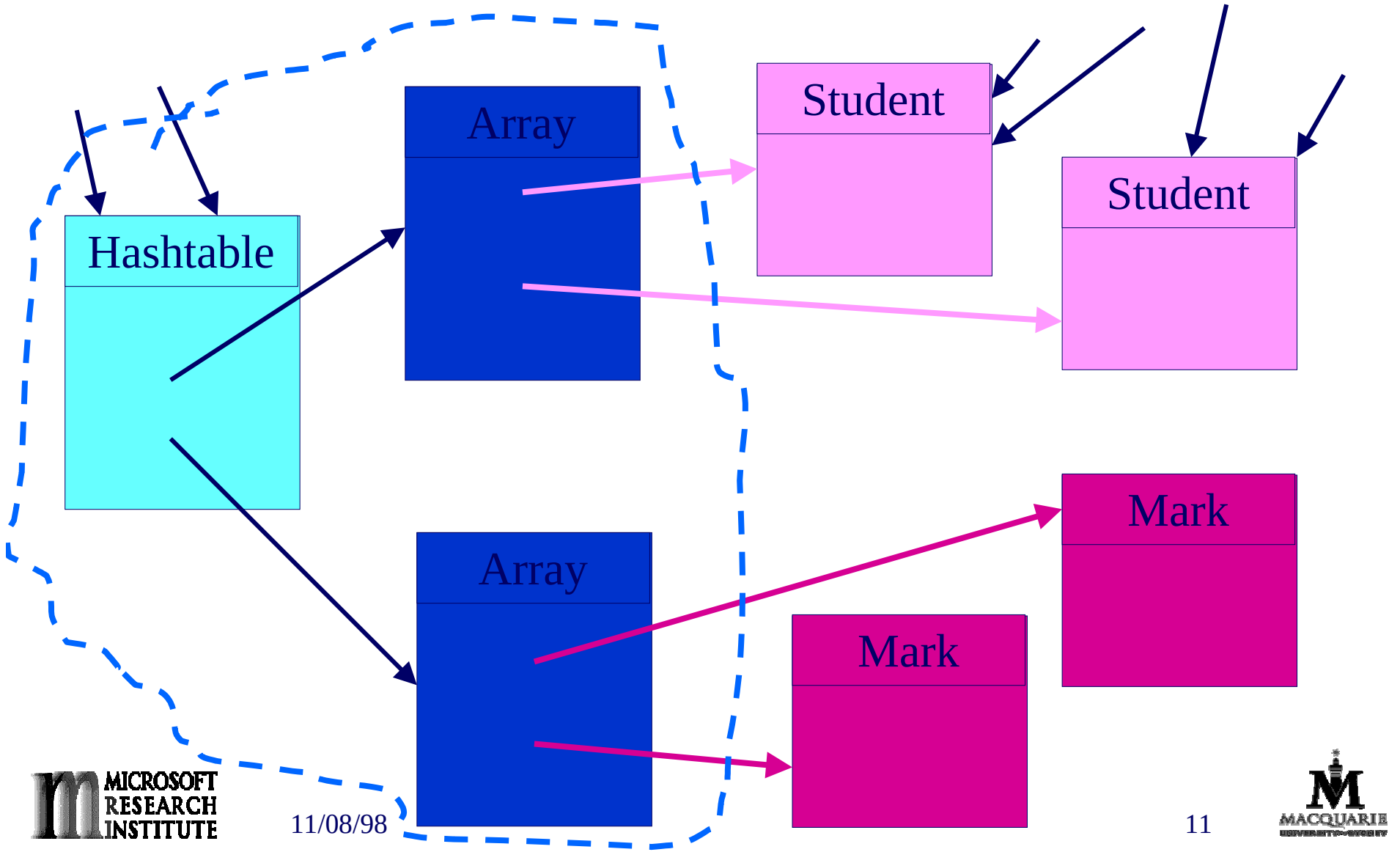
An Aside: Immutable Objects

- Consider immutable objects
  - Integers, Booleans, Symbols, Java Strings
  - their state never changes
  - they can be safely aliased anywhere
- Therefore:
  - treat aliased objects *as immutable*
  - allow references out of containers
  - restrict visibility of state via these aliases

Flexible Alias Protection

- ***Alias-protected container***
- **Representation Objects**
  - private, should be protected against access from outside a container
  - may be used (aliased) freely inside
- **Arguments Objects**
  - public, can be aliased and modified anywhere
  - container may not access their mutable state
- Protection is *per-object*

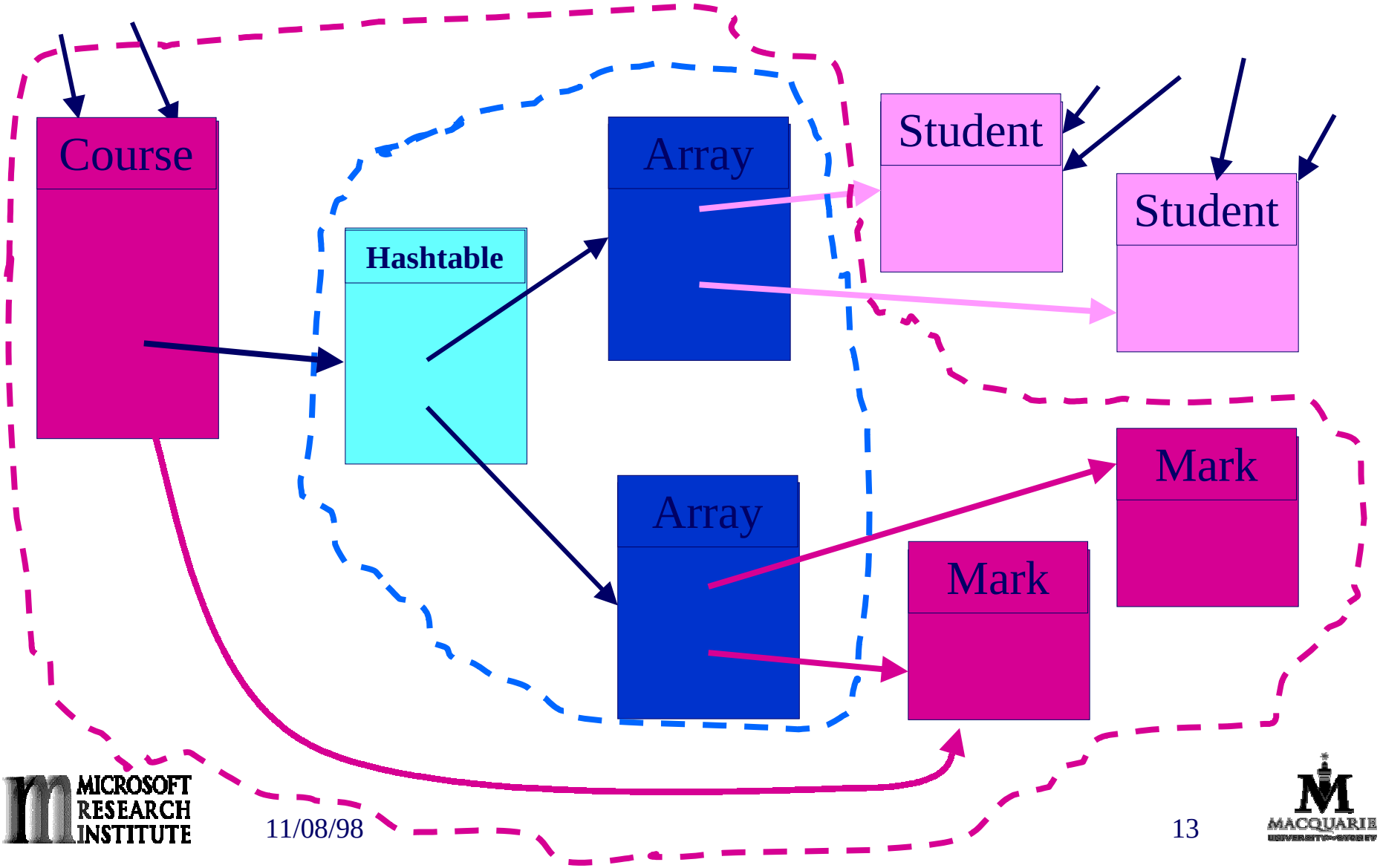
Flexible Alias Protection



Flexible Alias Protection

- **No Representation Exposure**
  - representation should only be accessible via the container's interface
  - static and dynamic references
- **No Argument Dependence**
  - container should not depend on mutable state
  - depend on arguments *immutable interface*
- **No Role Confusion**
  - keep separate argument roles separate

Composing Containers



Aliasing Modes

- Static checking to enforce invariants
- *Modes* decorate local names
  - and control access through that name only
- Modes are checked similarly to types
  - enforced by simple, local rules
  - mode checking is conservative
  - mode checking is modular
- Modes are *per-object*

Aliasing Modes

- **rep —** part of a container's representation
- **arg R —** container's argument, no access to mutable state.
- **free —** the only reference to a new object
- **val —** a reference to an immutable object
- **var R —** a reference to an mutable object

Example of Aliasing Modes

```
class Hashtable<arg k Hashable, arg i Item> {
  private rep Array<arg k Hashable> keys;
  private rep Array<arg i Item> items;
  private val int size;

   public void put(arg k Hashable key, arg i Item value);
  public arg i Item get(arg k Hashable key);
}
```

Mode Invariants

- **No Representation Exposure**
  - mode **rep** may not appear in an alias-protected container's interface.
- **No Argument Dependence**
  - mode **arg** may not change or depend upon mutable state.
- **No Role Confusion**
  - modes are not assignment compatible
    - except mode **free**

Usability

- An engineering compromise
  - how much safety is provided?
  - at what cost to style and efficiency?
- Syntactic overhead
- Simplicity
- Modularity
- Modes provide a conceptual language for programs and designs.

Status

- *mmmPizza*
  - Pizza with aliasing modes
  - Pizza collection library is mode-safe
- formal descriptions
  - object ownership
  - soundness of mode type system
- dynamic monitoring, software visualisation, design patterns, etc.

Conclusion

- Aliasing is endemic in Object-Oriented programming.
- Flexible Aliasing Protection
  - No Representation Exposure
  - No Argument Dependence
  - No Role Confusion
- Aliasing modes provide static guarantees

Credits

- *Islands* John Hogg, OOPSLA'91.
- *Geneva Convention on Object Aliasing,* Hogg et. al., OOPS Messenger v3n2, 1992.
- *Balloons* Paolo Sérgio Almeida, ECOOP'97
- David Clarke
- David Holmes, Eydun Eli Jacobsen, Doug Lea
- Martin Odersky (for Pizza)
- Microsoft Pty. Ltd., Australia.

THE END

MICROSOFT RESEARCH INSTITUTE

MACQUARIE UNIVERSITY