



# CS 3110

## Abstraction and Specification

Prof. Clarkson

Fall 2019

Today's music: Never Change by JAY-Z

# **CLICKER QUESTION 1**



**OCaml**



# Review

Previously in 3110:

- OCaml module system

Today:

- Abstract and specification

# Abstraction

(verb)

Forgetting information, so that different things can be treated as the same

(noun)

Artifacts that result from that process

# Specification

(noun)

Intended behavior of abstraction

(verb)

The act of creating such an artifact

# Audience of specification

- Clients

- What they must guarantee (preconditions)
- What they can assume (postconditions)

- Implementers

- What they can assume (preconditions)
- What they must guarantee (postconditions)

# Benefits of specifications

- **Locality:** understand abstraction without needing to read implementation
- **Modifiability:** change implementation without breaking client code
- **Accountability:** clarify who is to blame



# Specifications are contracts



# Satisfaction

An implementation **satisfies** a specification if it provides the described behavior

Many implementations can satisfy the same specification

- **Client** has to assume it could be any of them
- **Implementer** gets to pick one

# What if spec is ambiguous?

Ambiguity is a fact of life.

Do the most reasonable thing you can.

Probably not 

Who wrote it?

- **You:** improve it
- **Client:** seek clarification

*but if you make 500 requests they probably won't hire you again*

# **SPECIFYING FUNCTIONS**

# Template

```
(** [f x] is ...  
    Example: ...  
    Requires: ...  
    Raises: ... *)  
  
val f : t -> u
```

Based on *Abstraction and Specification in Program Development*

(Now *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*)

By Barbara Liskov and John Guttag

# Barbara Liskov



b. 1939

Turing Award Winner 2008

*For contributions to practical and theoretical foundations of programming language and system design, especially related to **data abstraction**, fault tolerance, and distributed computing.*

# **SPECIFYING DATA ABSTRACTIONS**

## **CLICKER QUESTION 2**

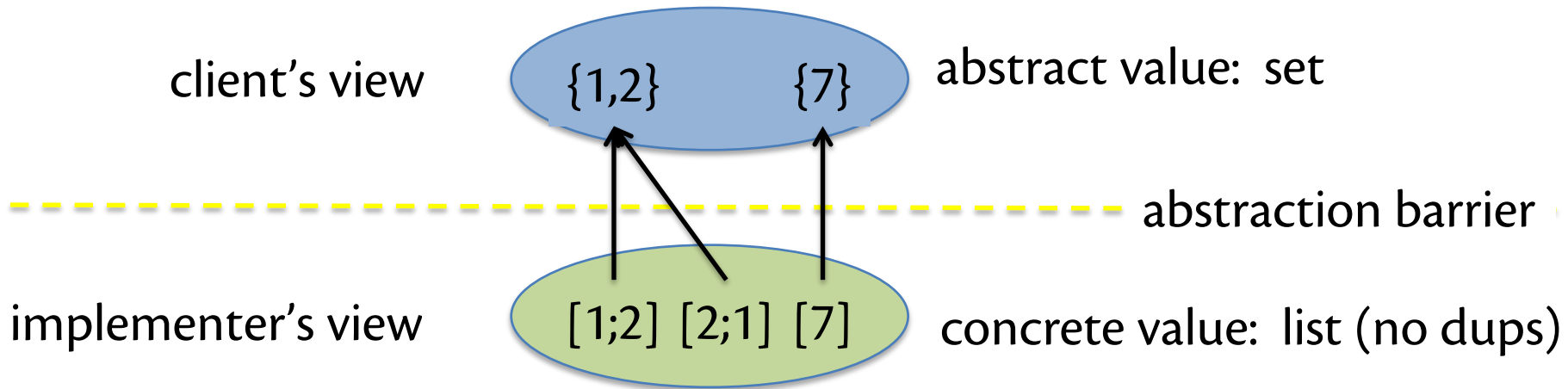


# Representation types

- Q: How to **interpret** the representation type as the data abstraction?
- A: Abstraction function
- Q: How to determine which values of representation type are **meaningful**?
- A: Representation invariant

# **ABSTRACTION FUNCTIONS**

# Abstraction function



the black arrows are the abstraction function

# Abstraction function

maps

valid concrete values

to

abstract values

# Documenting the AF

- Above rep type in implementation you write:  
`(* AF: comment *)`
- Write it **first** before implementing operations

# Implementing the AF

- You don't
  - Abstract values are a mathematical idea not a programming reality
  - Would need to have an OCaml type for abstract values
  - If you had that type, you wouldn't need whatever data abstraction you're working on
- But conversion to strings comes close

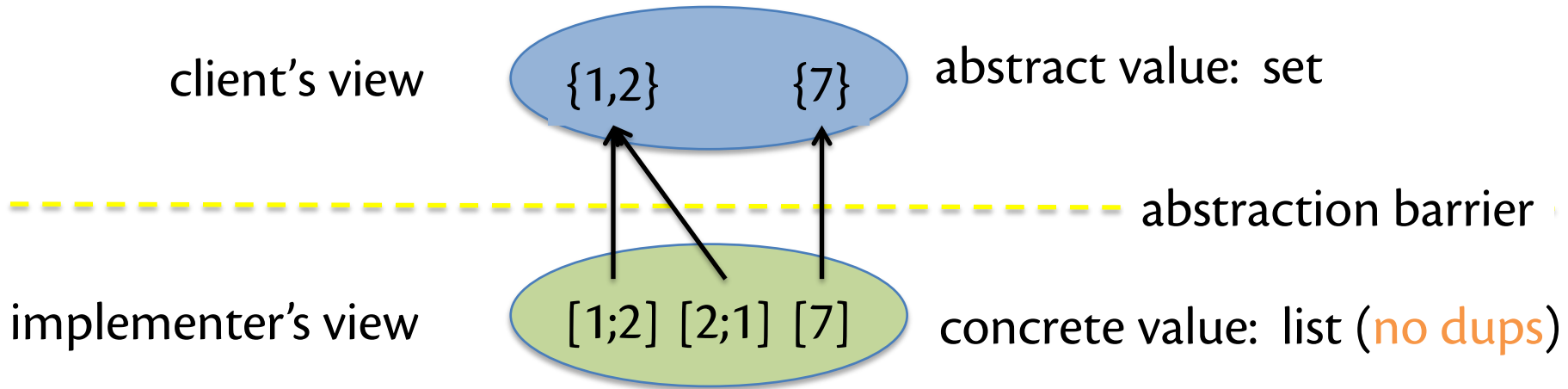
# Representation types

- Q: How to **interpret** the representation type as the data abstraction?
- A: Abstraction function
- Q: How to determine which values of representation type are **meaningful**?
- A: Representation invariant

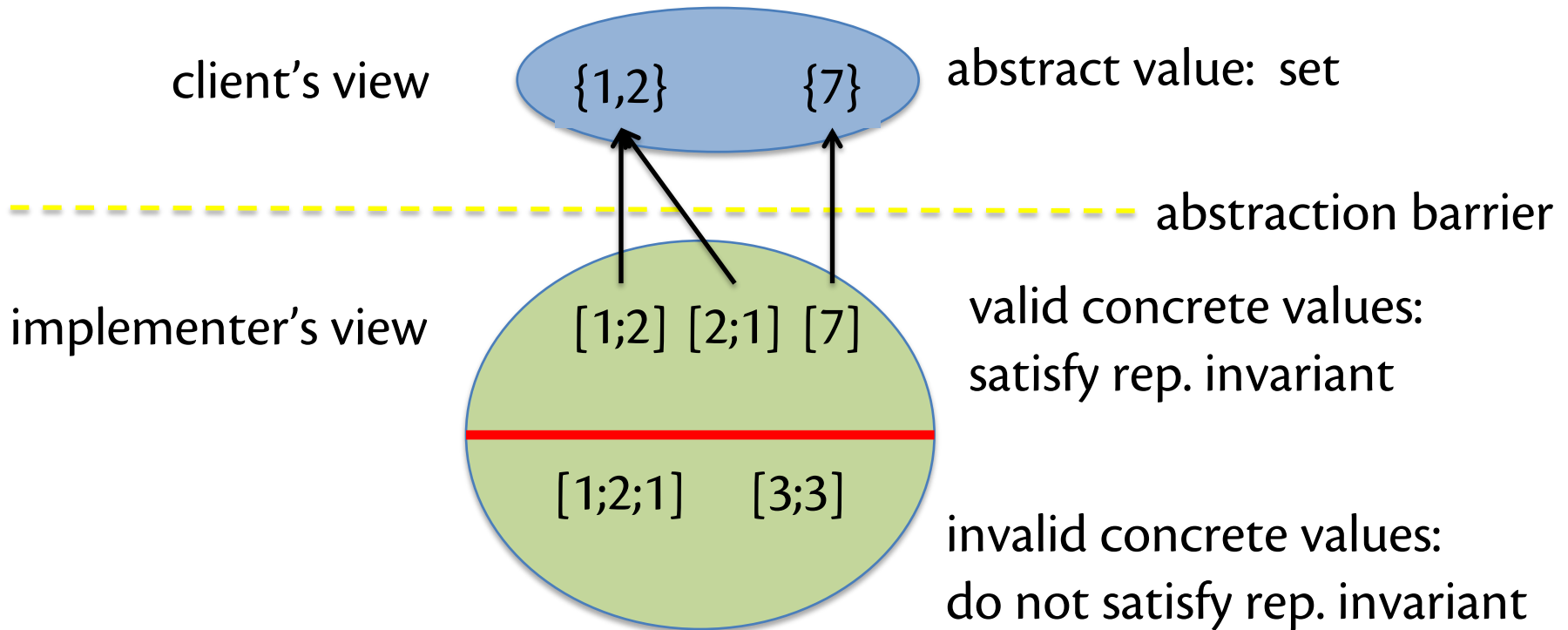
# **REPRESENTATION INVARIANTS**



# Abstraction function



# Representation invariant



the thick red line is the rep. invariant

# Rep. invariant

distinguishes

valid concrete values

from

invalid concrete values

# Documenting the RI

- Above rep type in implementation you write:  
`(* RI: comment *)`
- Write it **first** before implementing operations

# Rep. invariant

implicitly part of  
every precondition and  
every postcondition  
in abstraction

# Invariant may temporarily be violated

concrete  
input



concrete  
operation



concrete  
output



RI holds



RI maybe violated



RI holds



Demo

# Implementing the RI

Idiom:

- write `rep_ok` function
- call function on every input and output

```
let rep_ok (x : t) : t =  
  if (* check RI *) then t  
  else failwith "RI"
```

This saved a 3110 final project game tournament one year!

# Upcoming events

- [last night] A2 due
- [later today] A3 out

*This is invariant.*

**THIS IS 3110**