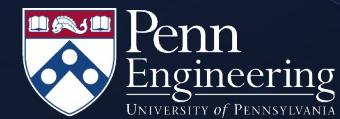




# Tracking how dependently-typed functions use their arguments

Stephanie Weirich  
University of Pennsylvania  
Philadelphia, USA



# Collaborators

## Yiyun Liu

Jonathan Chan  
Jessica Shi  
Pritam Choudhury



Yiyun: DCOI: *Dependent Calculus of Indistinguishability*

- POPL 24 – DCOI the language
  - POPL 25 – DCOI the logic
  - POPL 26 – Equality w/ surjective pairing
- Forthcoming dissertation *Dependency tracking and Dependent types*



# Let's talk about constant functions

---

$\text{id} : \forall (A : \text{Type}) \rightarrow A \rightarrow A$

$\text{id} = \lambda A x. x$

$$\text{id} = \lambda \_ \ x. \ x$$

Erasure semantics for  
type polymorphism

# Erasure semantics for type polymorphism

---

```
data List (A : Type) : Type where
  Nil : List A
  Cons : A → List A → List A

map : ∀ (A B : Type) → (A → B) → List A → List B
map = λ A B f xs.
  case xs of
    Nil ⇒ Nil
    Cons y ys ⇒ Cons (f y) (map A B f ys)
```

# Erasure semantics for type polymorphism

---

data

Nil

Cons

```
map = λ _ _ f xs.  
    case xs of  
        Nil ⇒ Nil  
        Cons y ys ⇒ Cons (f y) (map _ _ f ys)
```

# Erasure in **dependently-typed** languages

---

```
data Vec (n:Nat) (A:Type) : Type where
  Nil  : Vec Zero A
  Cons :  $\Pi(m:\text{Nat}) \rightarrow A \rightarrow (\text{Vec } m A) \rightarrow \text{Vec } (\text{Succ } m) A$ 

map :  $\forall(A\ B : \text{Type}) \rightarrow \Pi(n:\text{Nat}) \rightarrow (A \rightarrow B)$ 
       $\rightarrow \text{Vec } n A \rightarrow \text{Vec } n B$ 
map =  $\lambda\ A\ B\ n\ f\ xs.$ 
      case xs of
        Nil  $\Rightarrow$  Nil
        Cons m y ys  $\Rightarrow$ 
          Cons m (f y) (map A B m f ys)
```

# Erasure in **dependently-typed** languages

---

```
data Vec (n:Nat) (A:Type) : Type where
  Nil  : Vec Zero A
  Cons : ∀(m:Nat) → A → (Vec m A) → Vec (Succ m) A

map : ∀(A B : Type) → ∀(n:Nat) → (A → B)
      → Vec n A → Vec n B
map = λ A B n f xs.
  case xs of
    Nil ⇒ Nil
    Cons m y ys ⇒
      Cons m (f y) (map A B m f ys)
```

# Erasure in **dependently-typed** languages

---

data

Nil

Cons

```
map = λ _ _ _ f xs.  
  case xs of  
    Nil ⇒ Nil  
    Cons _ y ys ⇒  
      Cons _ (f y) (map _ _ _ f ys)
```

# Refinement/Subset types

---

```
type EvenNat = { n : Nat | isEven n } Erasable proof
```

```
plusIsEven : Π(m n : Nat) → isEven m → isEven n  
          → isEven (m + n)
```

```
plusIsEven = λ m n p1 p2. ...
```

```
plus : EvenNat → EvenNat → EvenNat
```

```
plus = λ en em. case en, em of  
  (n, np), (m, mp) ⇒  
    (n + m, plusIsEven n m np mp)
```

# Refinement/Subset types

---

```
plus = λ en em. case en, em of
  (n, _), (m, _) ⇒
    (n + m, _)
```

# Erasable code is irrelevant

---

- Not all terms are needed for computation: some function arguments and data structure components are there only for type checking
- Especially common in dependently-typed programming and proving
- Can call such code *irrelevant*

# Why care about irrelevance?

---

1. The compiler can produce faster code
  - No need to compute erased arguments
2. The type checker can run more quickly
  - Comparing types for equality requires reduction, which can be sped up by erasure
3. Verification is less work for programmers
  - Proving that terms are equal is easier when you can ignore the irrelevant parts
4. More programs type check
  - May not be able to prove the irrelevant parts equal

Compile-time irrelevance

## Less work for verification: proof irrelevance

---

```
type EvenNat = { n : Nat | isEven n }

-- prove equality of two EvenNats
congEvenNat : (n m : Nat)
             → (np : isEven n)
             → (mp : isEven m)
             → (n = m)
             -- no need for proof of np = mp
             → ((n, np) = (m, mp) : EvenNat)
congEvenNat = λ n m en em p. ...
```

# More programs type check

---

...when more terms are equal, *by definition*

type tells us that  $f$  is a  
constant function

example :  $\forall(f : \forall(x : \text{Bool}) \rightarrow \text{Bool})$   
 $\rightarrow (f \text{ True} = f \text{ False})$

example =  $\lambda f . \text{Refl}$

Sound for the type checker to  
decide that these terms are equal

Proof of equality comes  
directly from type checker

# Type checkers identify irrelevant code

---

But how?

How should type checkers identify and take advantage of irrelevant code?

---

- 1. Modes**
- 2. Dependency**
- 3. Other ways  
(erasure, grades,  
sorts, truncation,  
etc)**

# Core dependent type system

---

$$\boxed{\Gamma \vdash a : A}$$

$$\begin{array}{c} \text{VAR} \\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \end{array}$$

$$\begin{array}{c} \text{PI} \\ \frac{\begin{array}{c} \Gamma \vdash A : \star \\ \Gamma, x : A \vdash B : \star \end{array}}{\Gamma \vdash \Pi x : A. B : \star} \end{array}$$

$$\begin{array}{c} \text{ABS} \\ \frac{\begin{array}{c} \Gamma, x : A \vdash b : B \\ \Gamma \vdash \Pi x : A. B : \star \end{array}}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B} \end{array}$$

$$\begin{array}{c} \text{APP} \\ \frac{\begin{array}{c} \Gamma \vdash b : \Pi x : A. B \\ \Gamma \vdash a : A \end{array}}{\Gamma \vdash b a : B[a/x]} \end{array}$$

$$\begin{array}{c} \text{CONV} \\ \frac{\Gamma \vdash a : A \quad \Gamma \vdash B : \star \quad \vdash A \equiv B}{\Gamma \vdash a : B} \end{array}$$

# Modes

---

Distinguish relevant and irrelevant abstractions through  
*modes*

Pfenning, LICS 01

**Mishra-Linger and Sheard, FoSSaCS 08** DDC, Choudhury and Weirich, ESOP 22  
Abel and Scherer, LMCS 12 DE, Liu and Weirich, ICFP 23

# Modal types and modes

---

- Modal type marks irrelevant code:  $\square A$
- Type system controlled by modes:  $m ::= R \mid I$ 
  - Variables have modes, must be  $R$  when used
$$\Gamma ::= \varepsilon \mid \Gamma, x :^m A$$
  - Resurrection ( $\Gamma^m$ ): replaces all  $m$  modes with  $R$
  - Mode-annotated quantification:  
 $\Pi x :^m A . B$  unifies  $\Pi x : A . B$  and  $\forall x : A . B$

# Modal types for irrelevance

---

Only relevant variables  
can be used

$$\text{M-VAR} \quad \frac{x : ^{\mathbf{R}} A \in \Gamma}{\Gamma \vdash x : A}$$

$$\text{M-Box} \quad \frac{\Gamma^{\mathbf{I}} \vdash a : A}{\Gamma \vdash \mathbf{box} \, a : \square A}$$

$$\text{M-LETBox} \quad \frac{\Gamma \vdash a : \square A \quad \Gamma, x : ^{\mathbf{I}} A \vdash b : B \quad \Gamma \vdash B : \star}{\Gamma \vdash \mathbf{unbox} \, x = a \, \mathbf{in} \, b : B}$$

Modal types mark irrelevant  
subterms.  
*Resurrection* means that any  
variable can be used inside a box.

The contents of a box are  
accessible only in other boxes.

# Mode-annotated functions

Only relevant variables can be used

$$\frac{x :^{\mathbf{R}} A \in \Gamma}{\Gamma \vdash x : A}$$

M-APP

$$\frac{\Gamma \vdash b : \Pi x :^m A. B \quad \Gamma^m \vdash a : A}{\Gamma \vdash b a^m : B[a/x]}$$

Irrelevant arguments checked with resurrected context

$\Pi$ -bound variables always relevant in the type

M-PI

$$\frac{\Gamma \vdash A : \star \quad \Gamma, x :^{\mathbf{R}} A \vdash B : \star}{\Gamma \vdash \Pi x :^m A. B : \star}$$

M-ABS

$$\frac{\begin{array}{c} \Gamma^{\mathbf{I}} \vdash \Pi x :^m A. B : \star \\ \Gamma, x :^m A \vdash b : B \end{array}}{\Gamma \vdash \lambda x :^m A. a : \Pi x :^m A. B}$$

Mode on  $\Pi$ -type determines mode in the context

M-CONV

$$\frac{\Gamma \vdash a : A \quad \vdash A \equiv B}{\Gamma \vdash a : B}$$

Conversion ignores irrelevant arguments

Types checked with "resurrected" context

# Conversion ignores irrelevant arguments

---

- Usual rules, plus
  - compare arguments marked R
  - ignore arguments marked I or inside a box

EQ-REL

$$\frac{\vdash b_1 \equiv b_2 \quad \vdash a_1 \equiv a_2}{\vdash b_1 a_1^R \equiv b_2 a_2^R}$$

EQ-IRR

$$\frac{\vdash b_1 \equiv b_2}{\vdash b_1 a_1^I \equiv b_2 a_2^I}$$

# Modes for irrelevance

---

- Benefits
  - Modes generalize  $\Pi$  and  $\forall$
  - Easy implementation: mark variables when introduced in the context, mark the context for resurrection
- Drawbacks
  - No irrelevant projections
  - Formation rule for  $\Pi$ -types looks a bit strange

# Drawback: no irrelevant projections

```
filter : ∀(A:Type) → ∀(n:Nat)  
        → (A → Bool) → (Vec n A) → ∃(m:Nat) × (Vec m A)
```

```
filter = λ A n f v.
```

```
  case v of
```

```
    Nil ⇒
```

```
      (0, Nil)
```

```
    Cons m x xs ⇒
```

```
      let p = filter A m f
```

```
      if f x
```

```
        then (Succ p.1, Cons p.1 x p.2)
```

```
        else p
```

Length is not statically known and irrelevant.

UNSAFE: Irrelevant first projection means that it is ignored during conversion  
 $\text{Vec } p.1 \text{ A} = \text{Vec } ?? \text{ A}$

## Alternative rule for $\Pi$ -types

---

- From [Pfenning 99] [Abel and Scherer 2012]
- Irrelevant arguments must be irrelevant *everywhere* including in types. No parametric polymorphism!

M-PI

$$\frac{\Gamma \vdash A : \star}{\Gamma, x :^R A \vdash B : \star}$$
$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash \Pi x :^m A.B : \star}$$

M-PI-ALT

$$\frac{\Gamma \vdash A : \star}{\Gamma, x :^m A \vdash B : \star}$$
$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash \Pi x :^m A.B : \star}$$

# Dependency

---

Track when outputs depend on inputs

DCC, Abadi et al., POPL 99

DDC, Choudhury and Weirich, ESOP 22  
**DCOI, Liu, Chan, Shi, Weirich, POPL 24, 25**

# Dependency tracking

---

- Typing judgment ensures that low-level outputs do not depend on high-level inputs

$$x :^H \text{Bool} \vdash a :^L \text{Int}$$

**Input level**

$x$  can only be used when  
observer level is  $\geq H$

**Observer level**

$a$  can only use variables whose  
levels are  $\leq L$

- Type system parameterized by ordered set of levels
  - Relevance ( $R < I$ )
  - *Other examples:* Security levels ( $\text{Low} < \text{Med} < \text{High}$ )  
Staged computation ( $0 < 1 < 2 \dots$ )

# Typing rules with dependency levels

$$\boxed{\Gamma \vdash a :^\ell A}$$

D-VAR

$$\frac{x :^m A \in \Gamma \quad m \leq \ell}{\Gamma \vdash x :^\ell A}$$

Variable usage  
restricted by  
observer level

D-ABS

$$\frac{\Gamma, x :^m A \vdash b :^\ell B \quad \Gamma \vdash \Pi x :^m A.B :^{\ell_1} \star}{\Gamma \vdash \lambda x :^m A. b :^\ell \Pi x :^m A.B}$$

Terms do not  
observe types,  
so level  
unimportant

$\Pi$ -types record the dependency  
levels of their arguments

D-PI

$$\frac{\Gamma \vdash A :^\ell \star \quad \Gamma, x :^m A \vdash B :^\ell \star}{\Gamma \vdash \Pi x :^m A.B :^\ell \star}$$

Vars have same  
level in terms  
and types

D-APP

$$\frac{\Gamma \vdash b :^\ell \Pi x :^m A.B \quad \Gamma \vdash a :^m A}{\Gamma \vdash b a^m :^\ell B[a/x]}$$

Application requires compatible  
dependency levels

# Indistinguishability: indexed definitional equality

---

$$\vdash a \equiv^\ell b$$

Observer at level  $\ell$  cannot distinguish between terms

If observer has a higher level than the argument, arguments must agree

EQ-D-APP-DIST

$$\frac{\vdash b_0 \equiv^\ell b_1 \quad \vdash a_0 \equiv^\ell a_1 \quad \ell_0 \leq \ell}{\vdash b_0 {a_0}^{\ell_0} \equiv^\ell b_1 {a_1}^{\ell_0}}$$

If observer does not have a higher level, arguments are ignored

EQ-D-APP-INDIST

$$\frac{\vdash b_0 \equiv^\ell b_1 \quad \ell_0 \not\leq \ell}{\vdash b_0 {a_0}^{\ell_0} \equiv^\ell b_1 {a_1}^{\ell_0}}$$

## Conversion can be used at **any** observer level

---

$$\frac{\text{D-Conv} \quad \Gamma \vdash a :^\ell A \quad \Gamma \vdash B :^{\ell_0} \star \quad \vdash A \equiv^{\ell_0} B}{\Gamma \vdash a :^\ell B}$$

Type system is **sound** because we cannot equate types with different head forms at *any* dependency level

# DCOI: irrelevant projections

```
vfilter : (A:I Type) → (n:I Nat)           Type is checked with I-observer  
          → (A → Bool) → (Vec n A) → (m:I Nat) × (Vec m A)  
vfilter = λ A n f v.  
  case v of  
    Nil ⇒ (0I, Nil)           Definition checks with R observer, but  
    Cons mI x xs ⇒           contains I-marked subterms  
      let p = vfilter AI mI f xs in  
      if f x  
        then ((Succ p.1)I, Cons p.1I x p.2)  
        else p                   First projection allowed in  
                                I-marked subterms only
```

# DCOI: Dependent Calculus of Indistinguishability

---

- **Yiu, Chan, Shi and Weirich.** *Internalizing Indistinguishability with Dependent Types.* POPL 2024
  - Based on Pure Type System (PTS)
  - Key results: Syntactic type soundness, noninterference
- **Yiu, Chan and Weirich.** *Consistency of a Dependent Calculus of Indistinguishability.* POPL 2025
  - Predicative universe hierarchy
  - Observer-indexed propositional equality, J-eliminator
  - Key results: Consistency, normalization, and decidable (observer-indexed) equality
- All results mechanized using Rocq proof assistant

# DCOI: Proof-specific features

False-elimination

$$\text{D-ABSURD} \quad \frac{\Gamma \vdash a :^m \perp \quad \Gamma \vdash A :^{\ell_1} \star}{\Gamma \vdash \mathbf{absurd} \, a :^{\ell} \, A}$$

Proof of false can be eliminated at *any* observer level.

Propositional indistinguishability

D-EQ

$$\frac{\Gamma \vdash a :^m A \quad \Gamma \vdash b :^m A \quad m \leq \ell}{\Gamma \vdash a =^m b :^{\ell} \star}$$

Equality must be observable to the type

D-REFL

$$\frac{\Gamma \vdash a :^m A}{\Gamma \vdash \mathbf{refl} :^{\ell} a =^m a}$$

Can create reflexivity proofs about any level

D-J

$$\frac{\begin{array}{c} \Gamma \vdash a :^{\ell_0} \\ \Gamma, x :^m A, y :^{\ell_0} a_1 =^m a_2 \\ a_1 =^m x \vdash B :^m \star \\ \Gamma \vdash b :^{\ell} B[a_1/x][\mathbf{refl}/y] \end{array}}{\Gamma \vdash \mathbf{J} a b :^{\ell} B[a_2/x][a/y]}$$

Level of proof must be less than current level  
 Equality between terms at level  $m$  witnessed via substitution at level  $m$   
 Equality between proofs at level  $\ell_0$  witnessed via substitution at level  $\ell_0$

# DCOI: Dependent Calculus of Indistinguishability

---

- Benefits
  - Irrelevant projection is sound!
  - Irrelevant absurdity is consistent!
  - Can reason about *indistinguishability* as a proposition
- Open questions
  - Compatibility with type-directed equality?
  - Relationship with sProp?
  - Irrelevance for single-constructor eliminators?
  - Level quantification? Is that compatible with syntax-directed equality?
  - Applications of dependency tracking besides irrelevance?

# Related work on Irrelevance

---

- Mode-based
  - Pfenning 2001, Mishra-Linger & Sheard 2008, Abel & Scherer 2012
- Dependency tracking
  - Type theory in color: Bernardy & Moulin 2013
  - Two level type theories: Kovács 2022, Annenkov et al. 2023
- Erasure-based
  - Miquel 2001, Barras & Bernardo 2008
- Proof irrelevance: Prop/sProp (Rocq), Prop (Agda)
  - Hofmann& Streicher 1988, Gilbert et al. 2019
- Quantitative Type Theory (Run-time irrelevance / erasure)
  - McBride 2016, Atkey 2018, Abel & Bernardy 2020, Choudhury et al. 2021, Moon et al. 2021, Abel et al. 2023

# Conclusions

---

- In dependent type systems, identifying irrelevant computations is important for *efficiency* and *expressivity*
- Type systems can track more than "types", they can also tell us what happens during computation
- Dependency analysis is a powerful hammer in type system design



# Erasure

---

You can't use something  
that is not there

Miquel, TLCA 01  
**Barras and Bernardo, FoSSaCS 2008**

Zombie/Trellys [Kimmel et al. MSFP 2012]  
Dependent Haskell [Weirich et al. ICFP 2018]

# ICC: Implicit Calculus of Constructions

---

- Extend core language with irrelevant (implicit) abstractions

E-ABS

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash \forall x : A. B : \star \quad x \notin \text{fv}|b|}{\Gamma \vdash \lambda x :^{\mathbf{I}} A. b : \forall x : A. B}$$

E-APP

$$\frac{\Gamma \vdash b : \forall x : A. B \quad \Gamma \vdash a : A}{\Gamma \vdash b\ a^{\mathbf{I}} : B[a/x]}$$

# ICC: Implicit Calculus of Constructions

---

- Extend core language with irrelevant (implicit) abstractions
- Annotations enable decidable type checking

E-ABS

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash \forall x : A. B : \star \quad x \notin \text{fv}|b|}{\Gamma \vdash \lambda x :^{\mathbf{I}} A. b : \forall x : A. B}$$

E-APP

$$\frac{\Gamma \vdash b : \forall x : A. B \quad \Gamma \vdash a : A}{\Gamma \vdash b [a]^{\mathbf{I}} : B[a/x]}$$

# ICC: Implicit Calculus of Constructions

---

- Extend core language with irrelevant (implicit) abstractions
- Annotations enable decidable type checking
- **Irrelevant parameters must not appear *relevantly***
- Erasure operation  $|a|$  removes irrelevant terms

E-ABS

$$\frac{\begin{array}{c} \Gamma, x : A \vdash b : B \\ \Gamma \vdash \forall x : A. B : \star \\ x \notin \text{fv}|b| \end{array}}{\Gamma \vdash \lambda x :^{\mathbf{I}} A. b : \forall x : A. B}$$

E-APP

$$\frac{\Gamma \vdash b : \forall x : A. B \quad \Gamma \vdash a : A}{\Gamma \vdash b\ a^{\mathbf{I}} : B[a/x]}$$

# Erasure during conversion

---

- Conversion between *erased* types
- Compile-time irrelevance: erased parts ignored when comparing types for equality

**E-CONV-ANN**

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : \star \quad \vdash |A| \equiv |B|}{\Gamma \vdash a : B}$$

# Erasure: Implicit Calculus of Constructions

---

- Benefits
  - Simple!
  - Directly connects to erasure in compilation
  - Orthogonal: features independent from the rest of the system
- Drawbacks
  - No irrelevant projections



# Internalized indistinguishability

---



Penn  
Engineering  
UNIVERSITY OF PENNSYLVANIA

## Results about DCOI

---

- [POPL 24] A PTS based on these rules is type sound

Results mechanized using Coq/Rocq theorem prover

- [POPL 25] This type system with a predicative universe hierarchy
  - Is consistent (cannot prove false)
  - Is normalizing (reduction terminates)
  - The indistinguishability relation is decidable

Results mechanized using Coq/Rocq theorem prover



# PRESENTATION TITLE

Name, Title  
Date



# Section Title Slide on white

---

Subtitle Goes Here

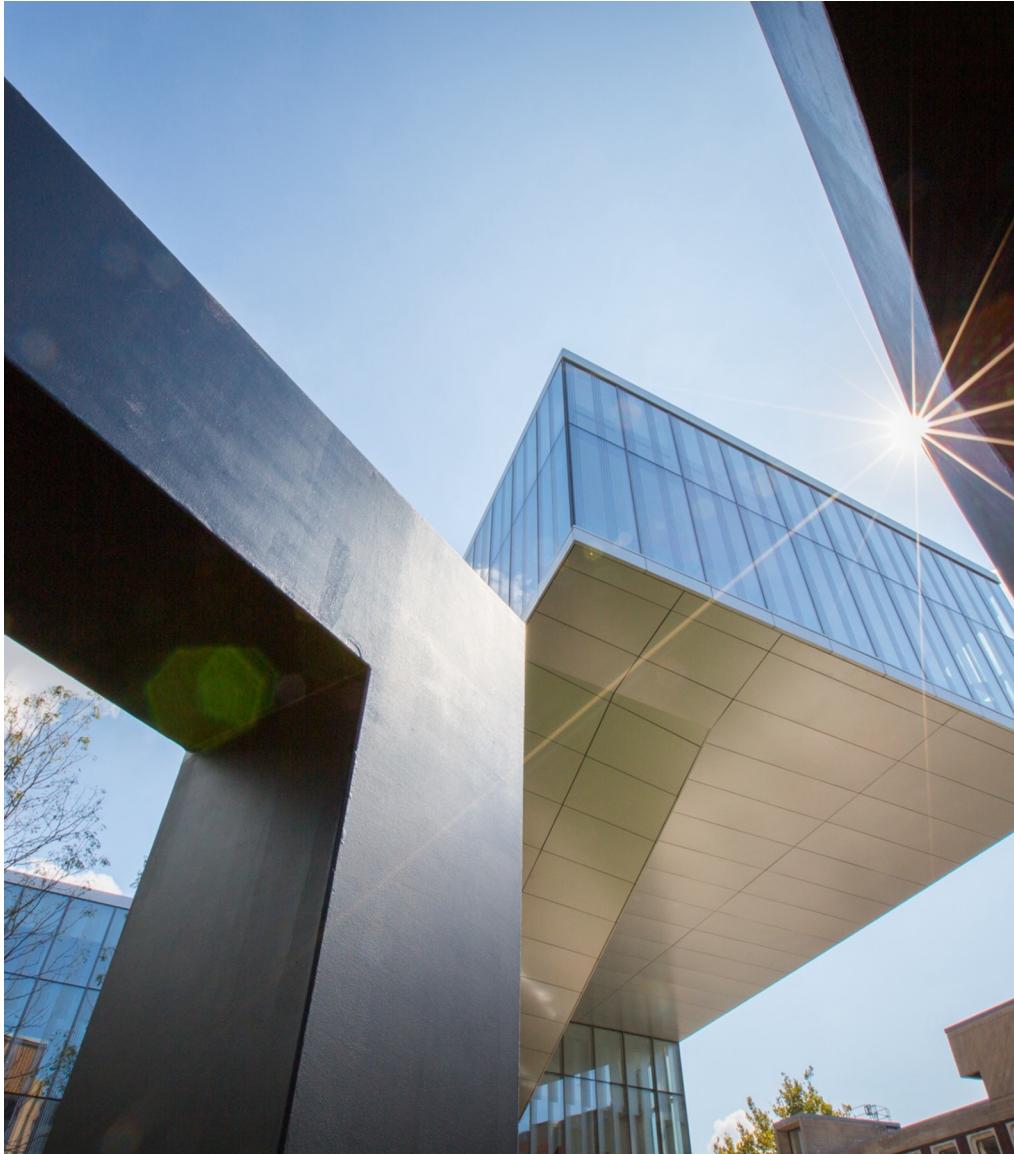
# Content Slide

---

# HEADLINE FOR SLIDE

# HEADLINE OF SLIDE

- Content for slide
- Content for slide
- Content for slide



- Add Text here
- Add Text here

Full Image Slide: Bottom Title (Blue)

# Full Image Slide: Top Title

---

