Co-Dfns Compiler

Aaron W. Hsu

Indiana University

APL is fast. Who needs a compiler?

Compilers can enhance reliability and performance.

Interpreter Limitations

Scalar fusion leads to poor cache behavior

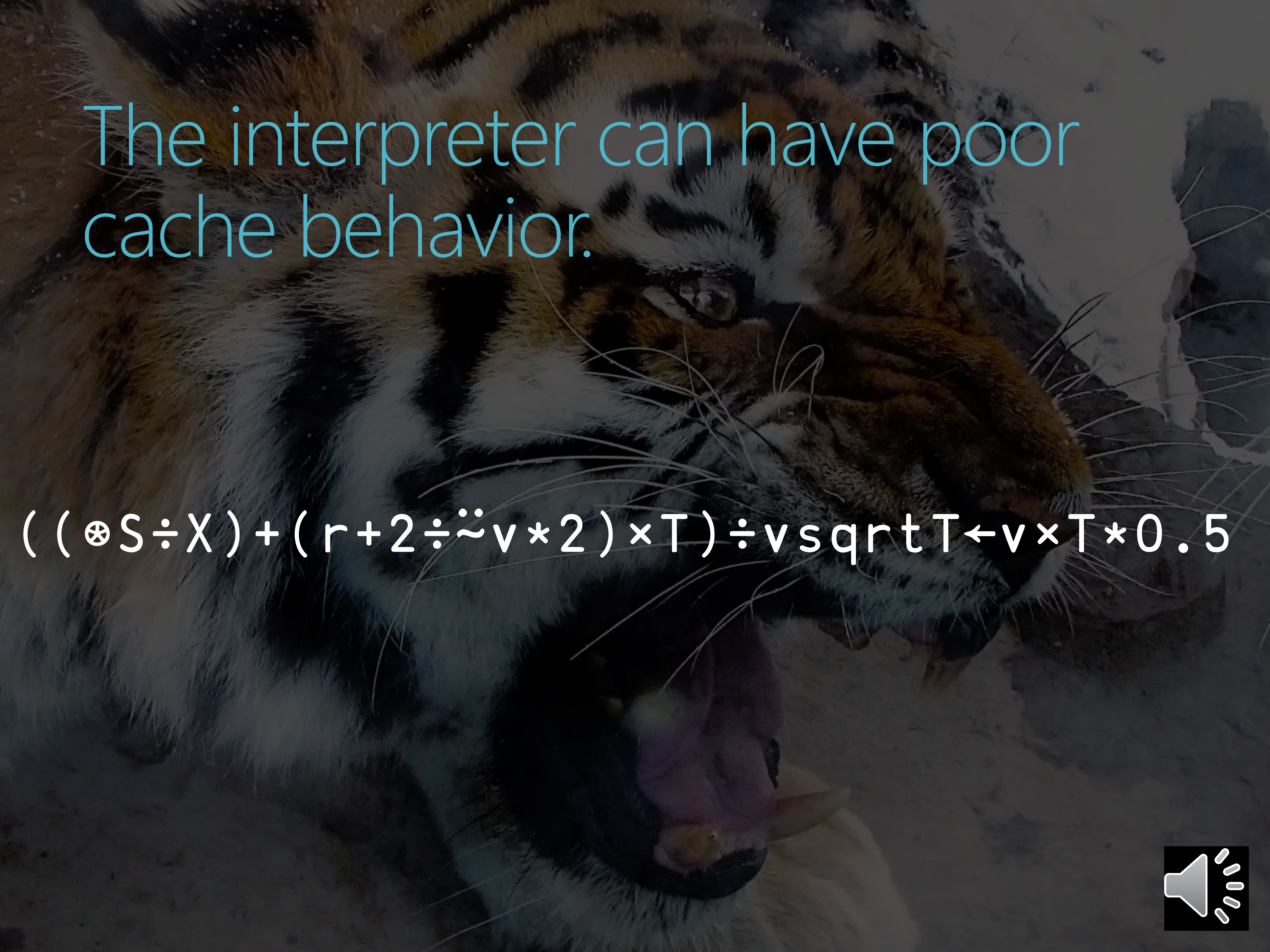
Procedure calls can be expensive

Common optimizations are not possible

No static analysis of code for errors or correctness

Co-Dfns is a headlong jump into APL compilation.

D-fns eliminates serious roadblocks to performance.



The interpreter can have poor cache behavior.

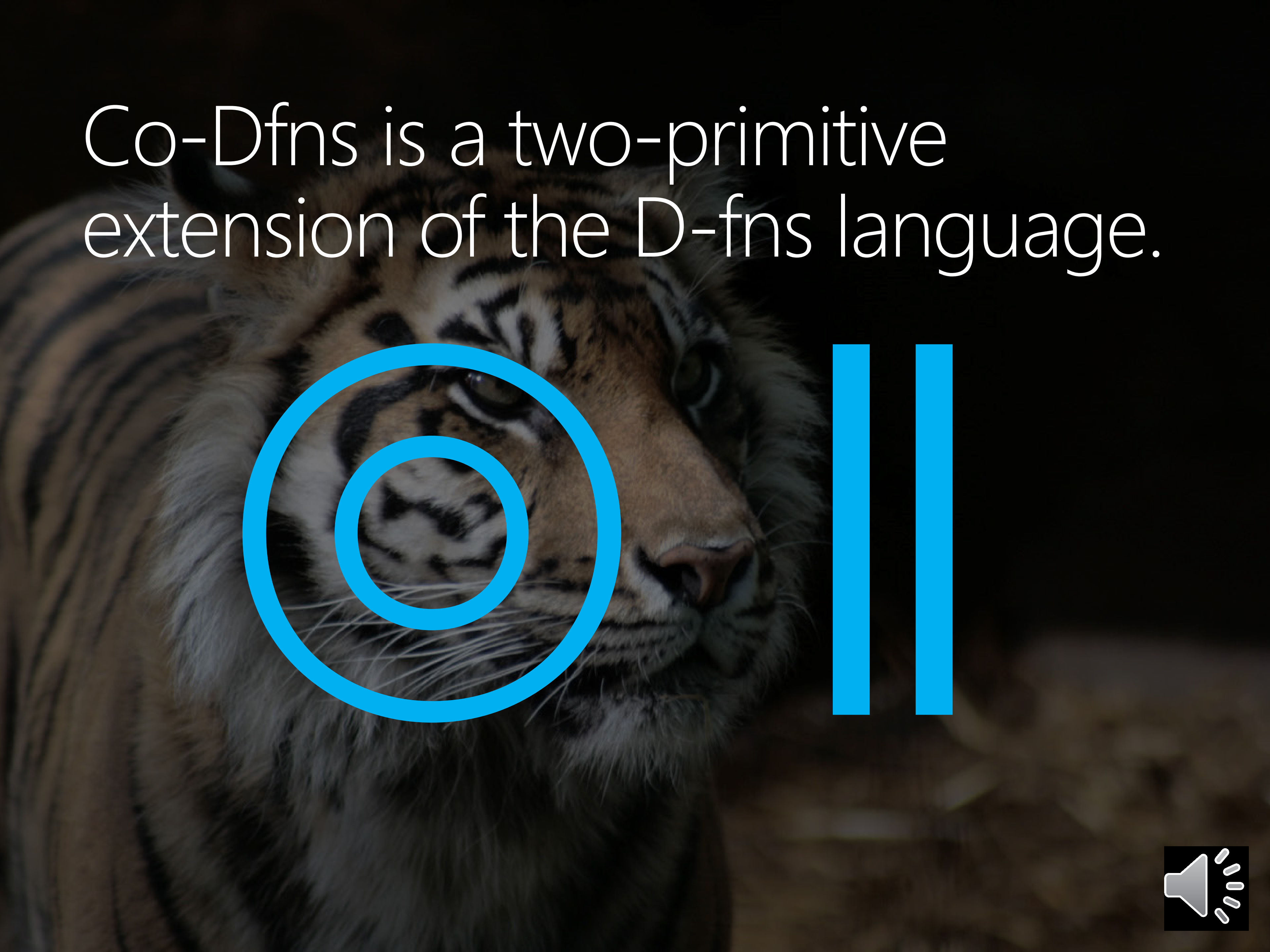
```
((⍟S÷X)+(r+2÷⍨v*2)×T)÷vsqrtT←v×T*0.5
```

Co-Dfns is for predictable, safe, controllable performance.

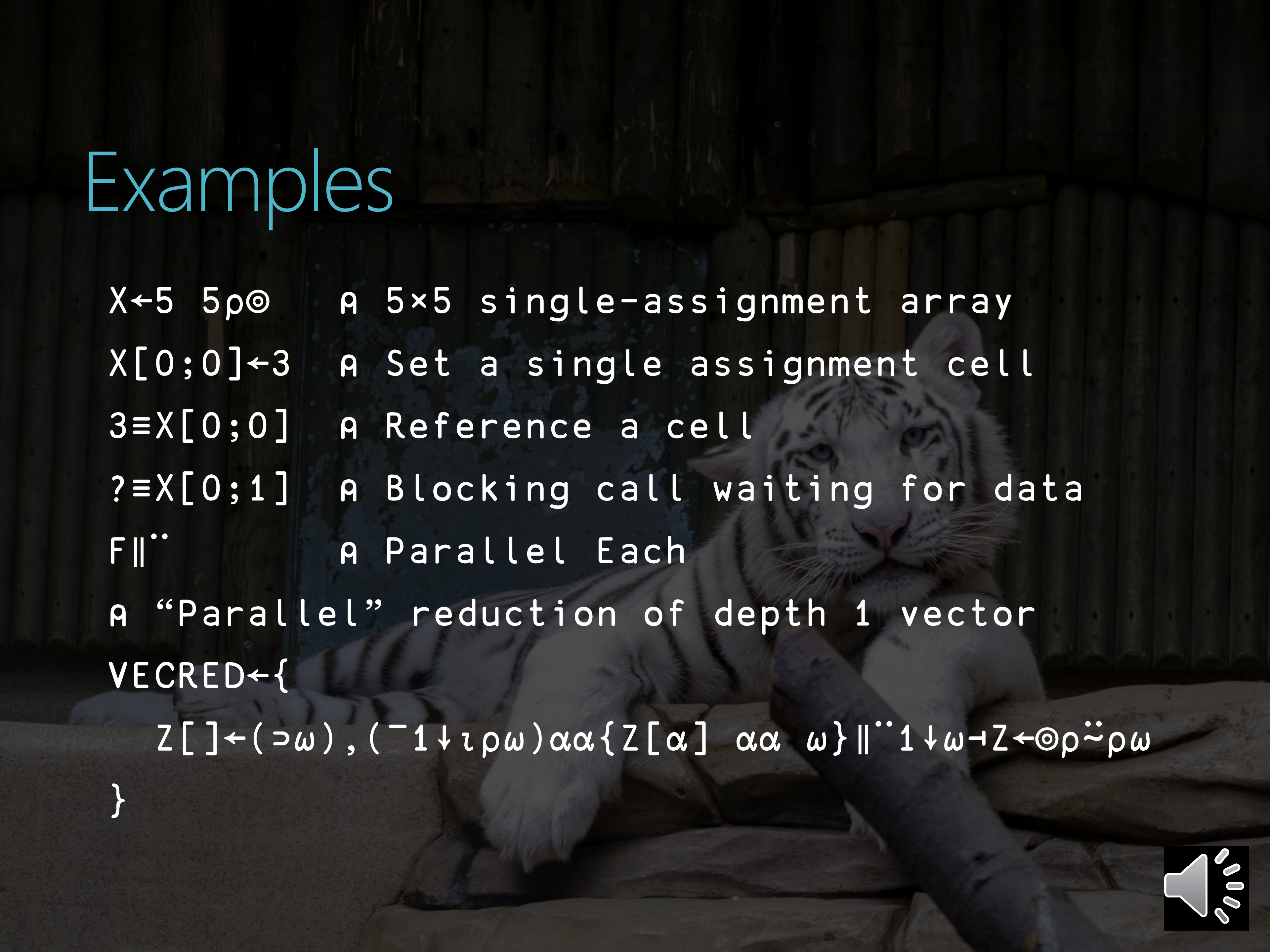
Tuning expertise should not be a black art.

Co-Dfns is a two-primitive extension of the D-fns language.

◎ ‖

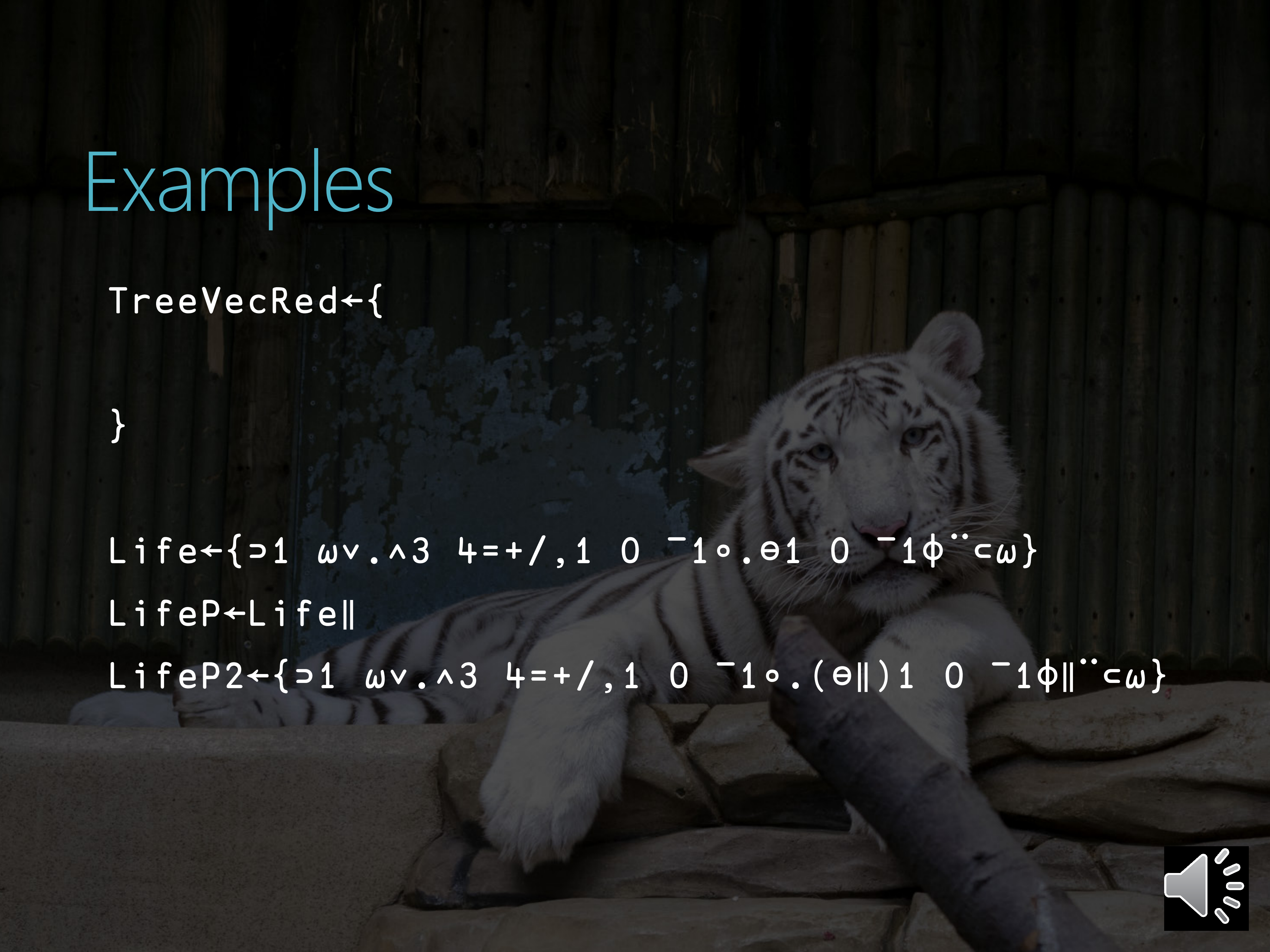


Examples



```
X←5 5⍴◎      ⍝ 5×5 single-assignment array
X[0;0]←3     ⍝ Set a single assignment cell
3≡X[0;0]     ⍝ Reference a cell
?≡X[0;1]     ⍝ Blocking call waiting for data
F‖¨          ⍝ Parallel Each
⍝ “Parallel” reduction of depth 1 vector
VECRED←{
  Z[]←(⊃⍵),(¯1↓⍳⍴⍵)⍺⍺{Z[⍺] ⍺⍺ ⍵}‖¨1↓⍵⊣Z←◎⍴⍨⍴⍵
}
```

Examples



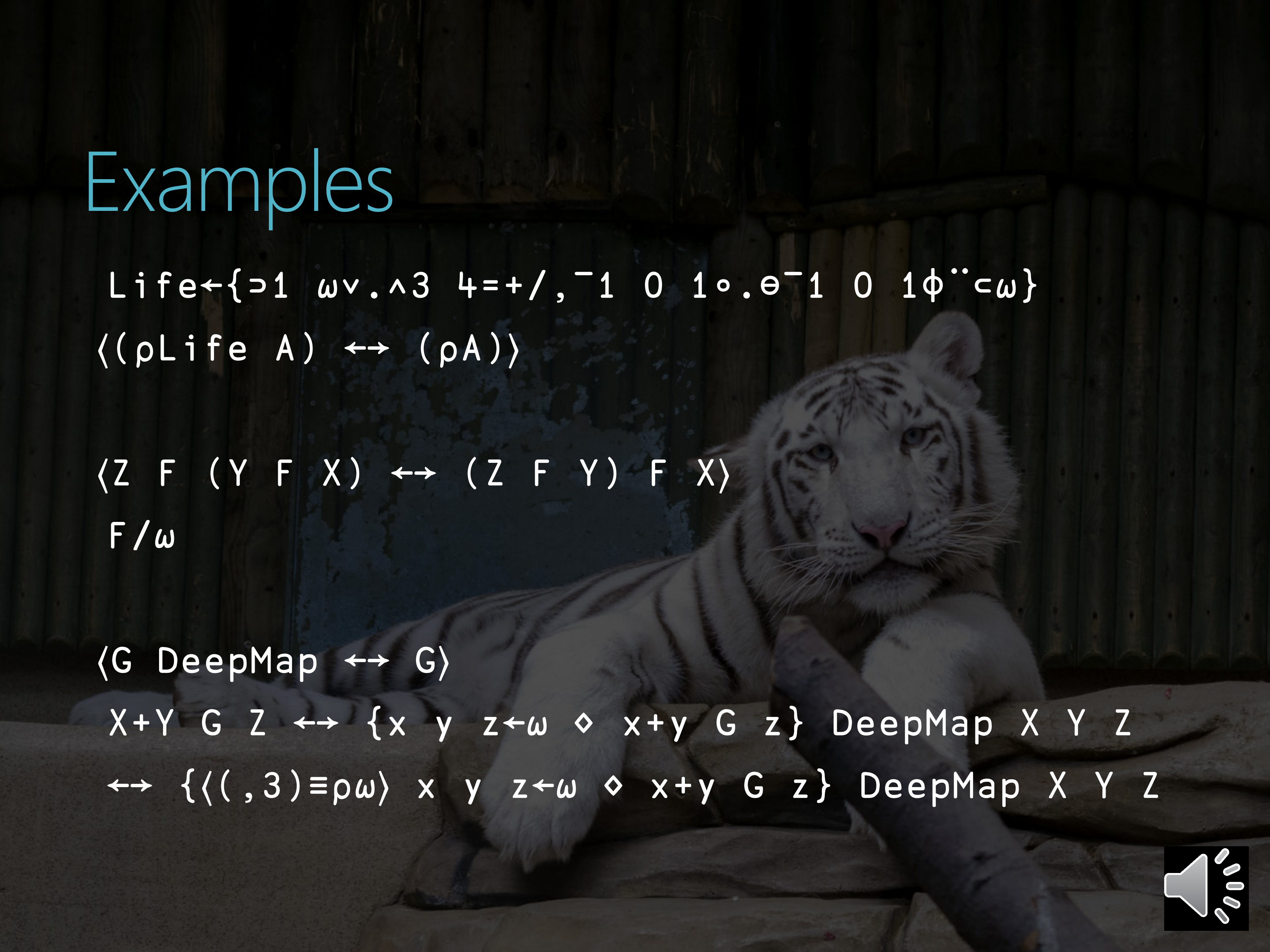
```
TreeVecRed←{


}


Life←{⊃1 ⍵∨.∧3 4=+/,1 0 ¯1∘.⊖1 0 ¯1⌽¨⊂⍵}
LifeP←Life‖
LifeP2←{⊃1 ⍵∨.∧3 4=+/,1 0 ¯1∘.(⊖‖)1 0 ¯1⌽‖¨⊂⍵}
```

Why shouldn't we be able to reason about performance?

ZPL and others make this a critical component.

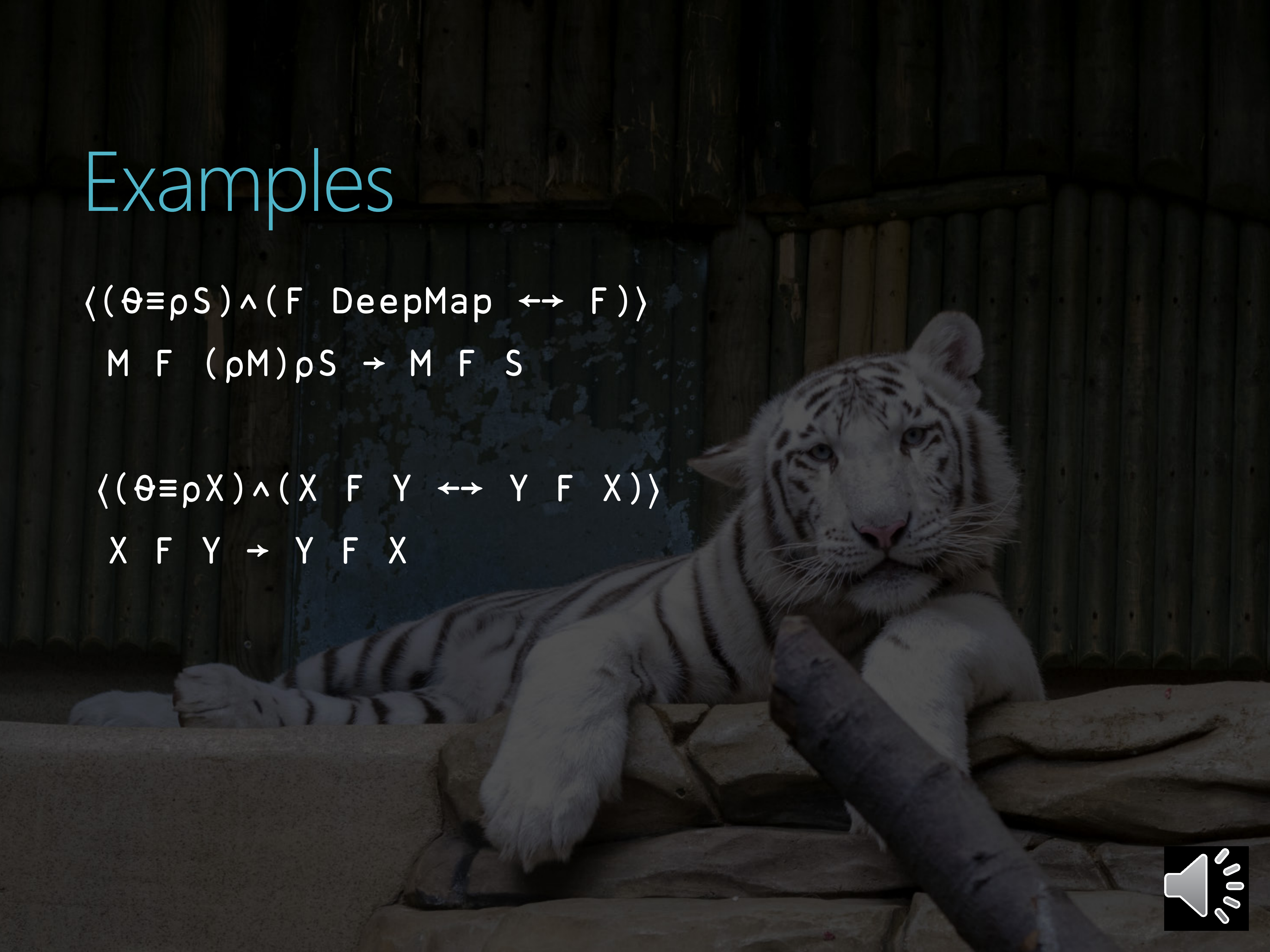
APL is math you can use. Let’s use it for performance.

Examples



```
 Life←{⊃1 ⍵∨.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1⌽¨⊂⍵}
⟨(⍴Life A) ←→ (⍴A)⟩


⟨Z F (Y F X) ←→ (Z F Y) F X⟩
 F/⍵


⟨G DeepMap ←→ G⟩
 X+Y G Z ←→ {x y z←⍵ ⋄ x+y G z} DeepMap X Y Z
 ←→ {⟨(,3)≡⍴⍵⟩ x y z←⍵ ⋄ x+y G z} DeepMap X Y Z
```

Examples



```
⟨(⍬≡⍴S)∧(F DeepMap ←→ F)⟩
 M F (⍴M)⍴S → M F S

 ⟨(⍬≡⍴X)∧(X F Y ←→ Y F X)⟩
 X F Y → Y F X
```

Plans and Schemes

Usable version this year

Target multi-core, GPU, and distributed clusters

Fully integrate with Dyalog

Focus on scalable parallel performance

Leverage APL-style formalism

Create a dialog between tuning expert and compiler

Version 1 will have low-hanging fruit optimizations

Interpreter vs. Compiler

INTERPRETER	COMPILER
Garbage collected	Stack-based allocation
Idiom-based special casing	Whole program optimization
No static verification	Explicit verification and proof
No user-guided optimization	Safe user-defined optimizations
Supports much more of APL	Restricted to Co-Dfns
Numerous extensions	Limited extensions and interop

Using Co-Dfns should require less than trivial effort.

Demo time

Performance not included, some assembly required.

We're not preaching super-compilation here.

Coding Goodies

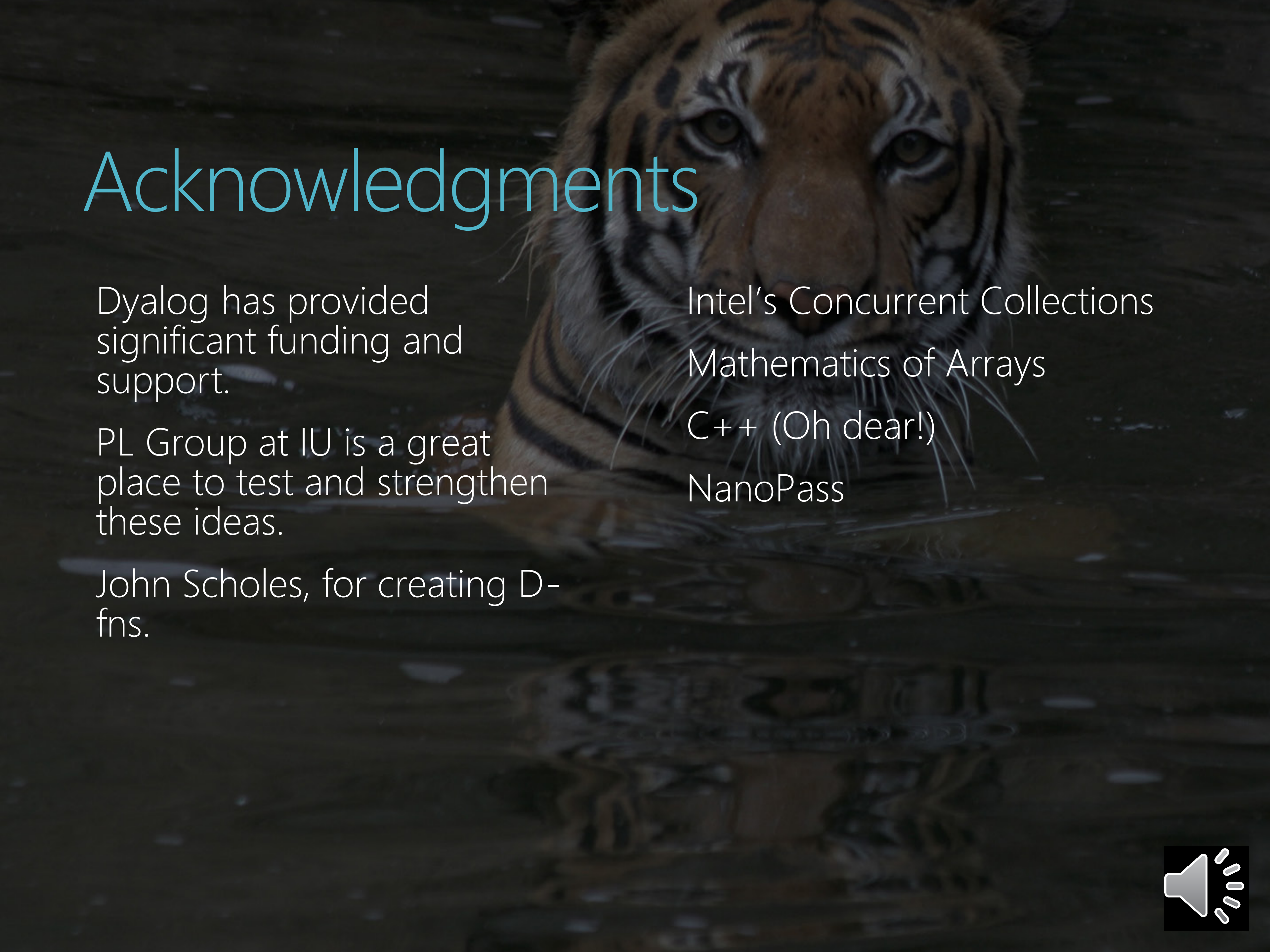
Easily integrated with other code

A complete, useful, general parser for APL

A complete, rigorous specification of the language

Supports multiple runtimes

Language integration



Acknowledgments

Dyalog has provided significant funding and support.

PL Group at IU is a great place to test and strengthen these ideas.

John Scholes, for creating D-fns.

Intel’s Concurrent Collections

Mathematics of Arrays

C++ (Oh dear!)

NanoPass

Thank you

Email: awhsu@indiana.edu

Photo Credits

Hotnstock

MoonsongStock

Ladyaway

Fall_Stock

Estruda

Animalphotos

Dark_dragon_stock

Drezdany_stocks

Stupiddeeppeople

Thiselectricheart

Vampsstock

NefaroStock

NikiljuiceStock