Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors

Samuel Thomas, **James Bornholt**

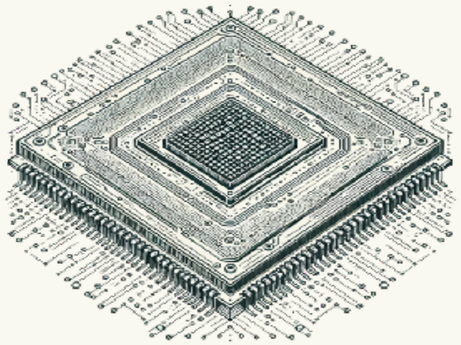
UT Austin

ASPLOS ‘24

Programmer

```
for i in 0..len(A):
  C[i] = A[i] + B[i]
```

Vectorizing Compiler

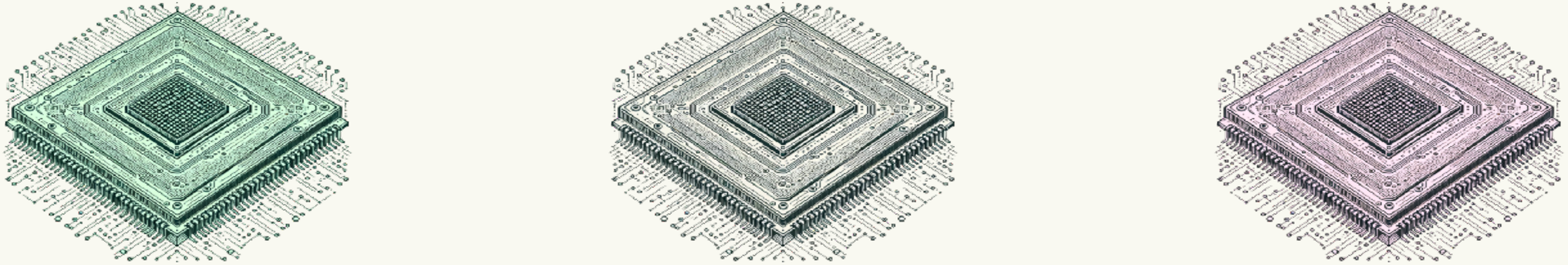


Digital Signal Processor (DSP)

Programmer

```
for i in 0..len(A):
  C[i] = A[i] + B[i]
```

Vectorizing Compiler



Digital Signal Processor (DSP)

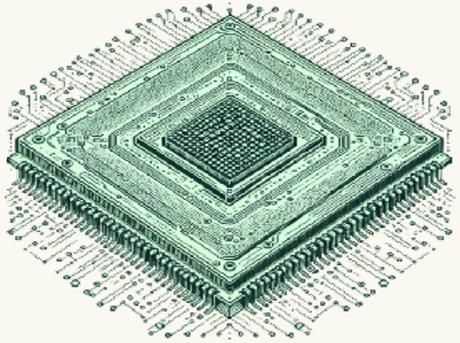
Programmer

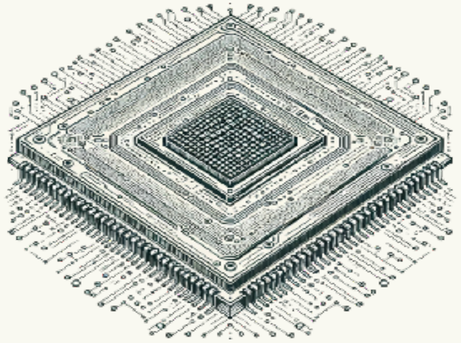
```
for i in 0..len(A):
  C[i] = A[i] + B[i]
```

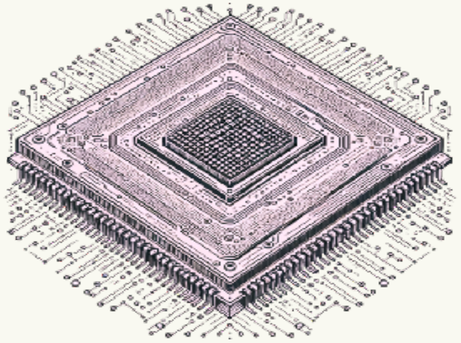
Vectorizing Compiler

Vectorizing Compiler

Vectorizing Compiler







Digital Signal Processor (DSP)

Programmer

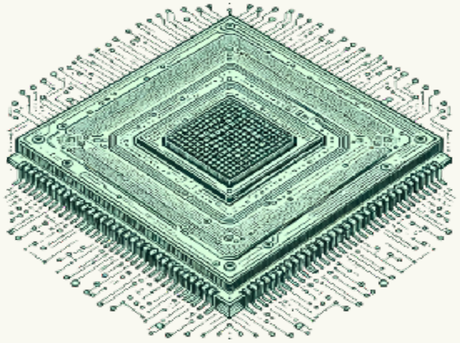
```
for i in 0..len(A):
  C[i] = A[i] + B[i]
```

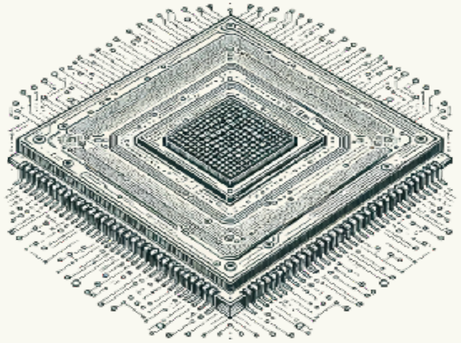
Automate

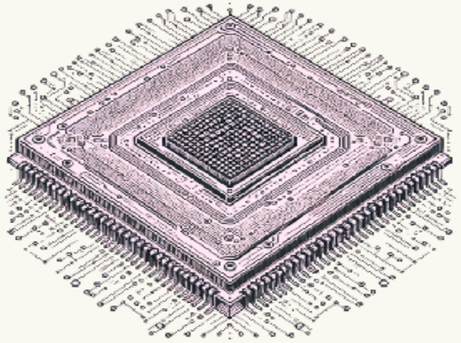
Vectorizing Compiler

Vectorizing Compiler

Vectorizing Compiler







Digital Signal Processor
(DSP)

Programmer

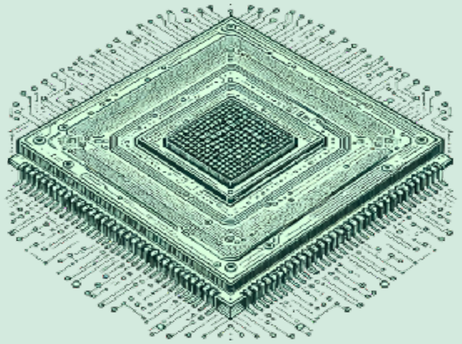
```
for i in 0..len(A):
  C[i] = A[i] + B[i]
```

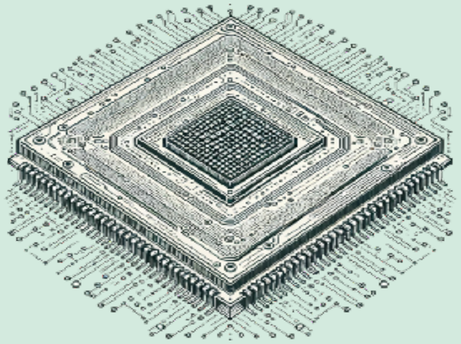
Automate

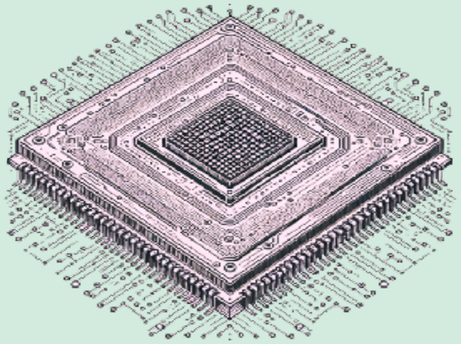
Vectorizing Compiler

Vectorizing Compiler

Vectorizing Compiler



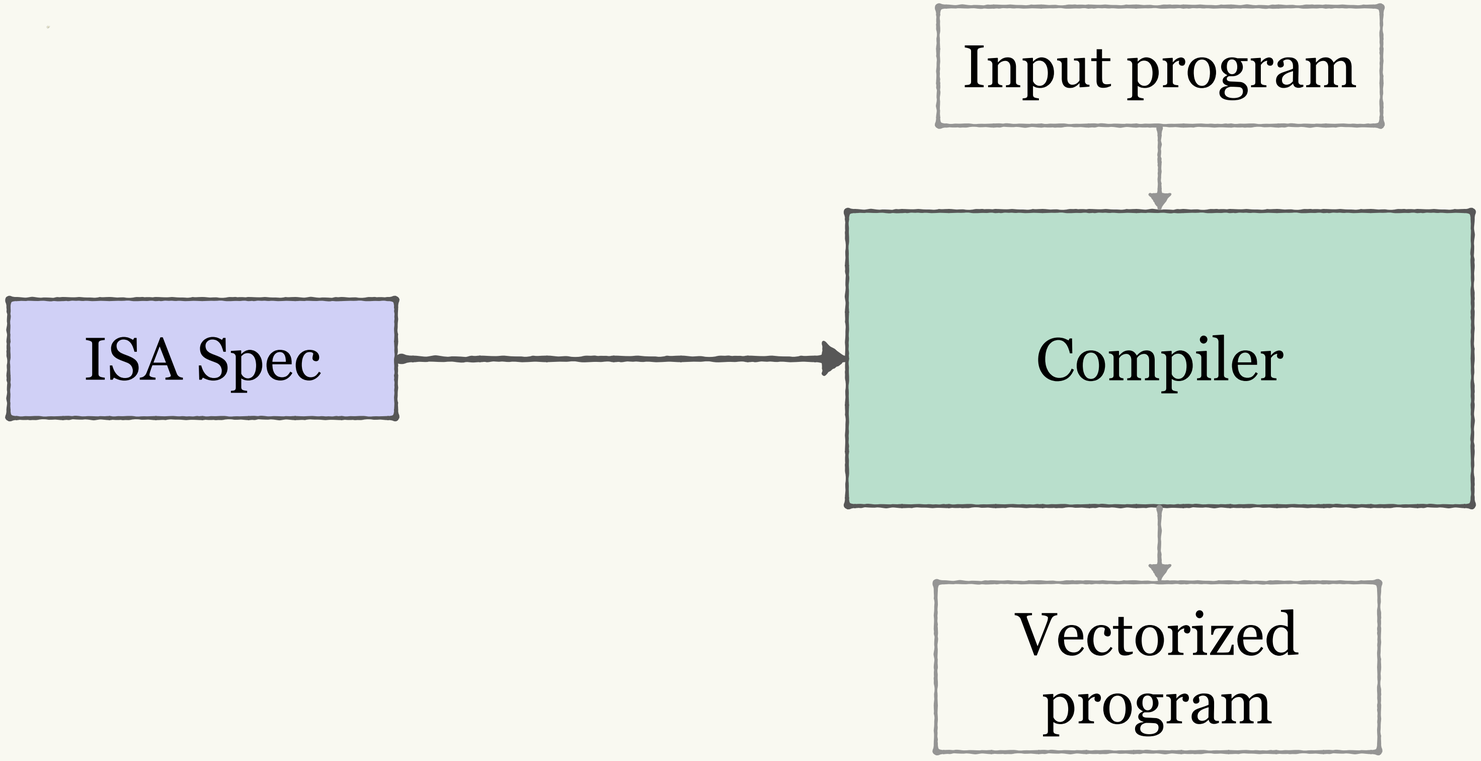




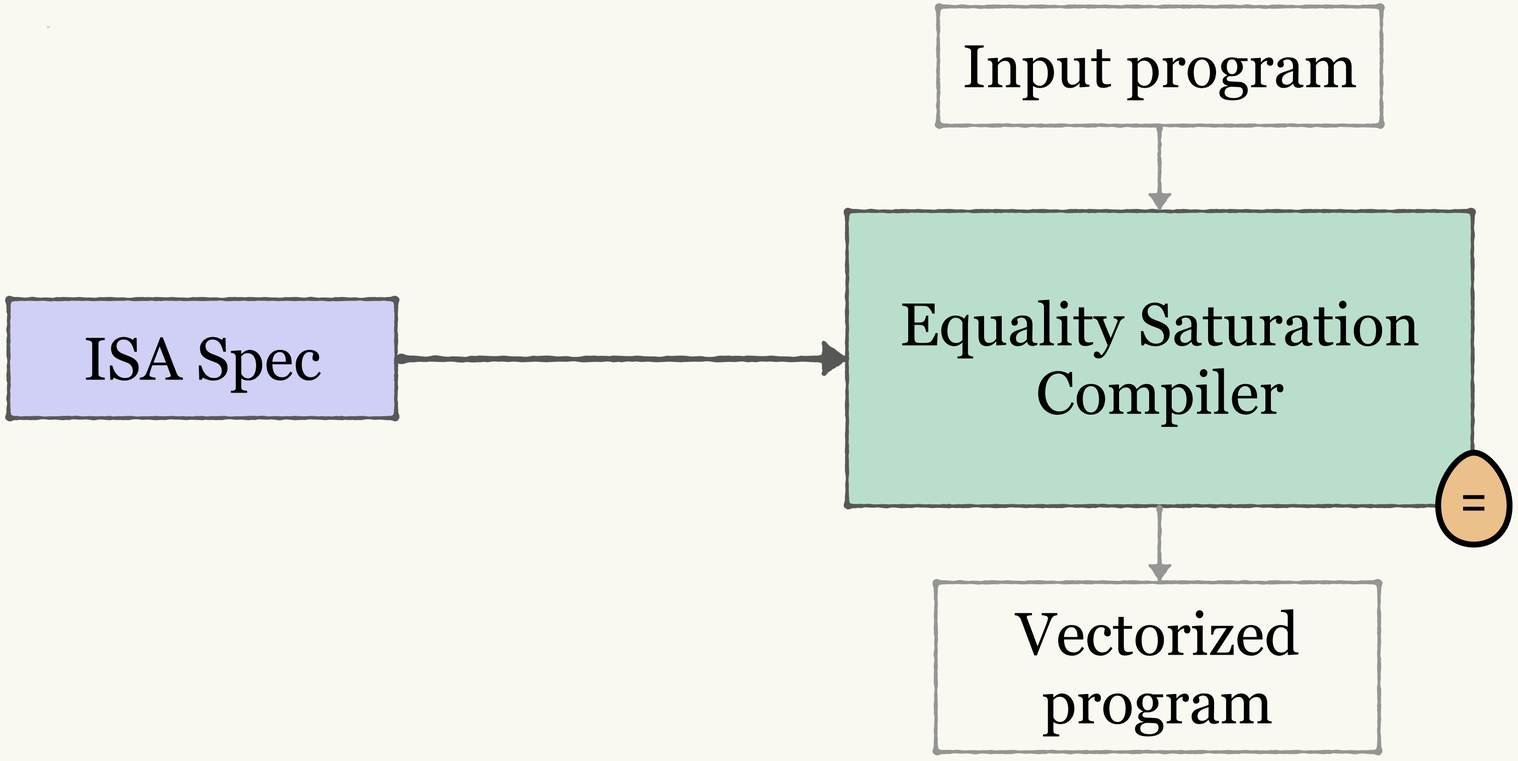
Explore

Digital Signal Processor (DSP)

Isaria

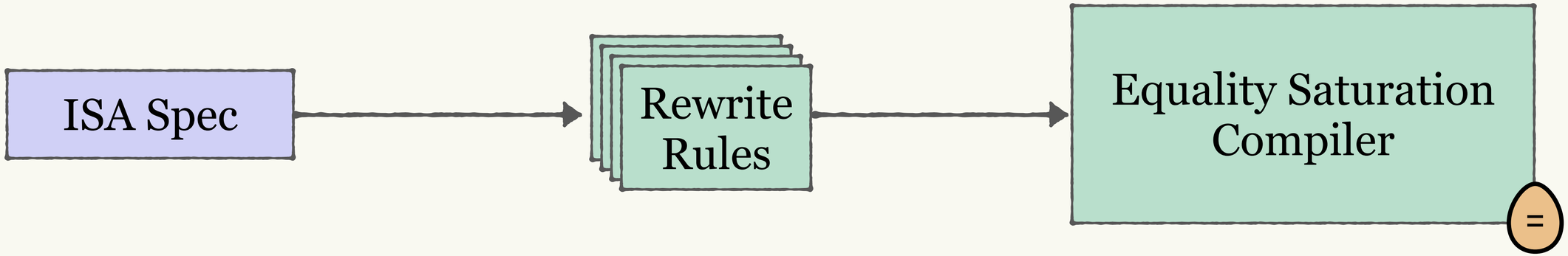


Isaria

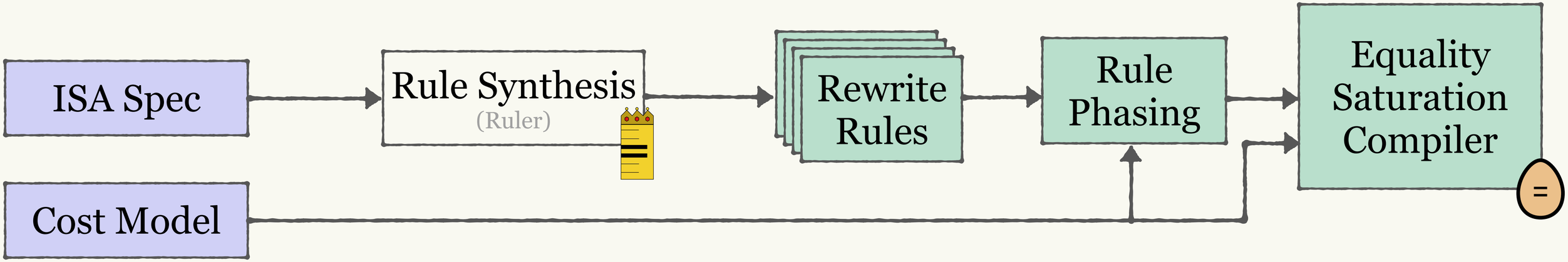


Diospyros: Alexa VanHattum, et al (ASPLOS '21)

Isaria



Isaria

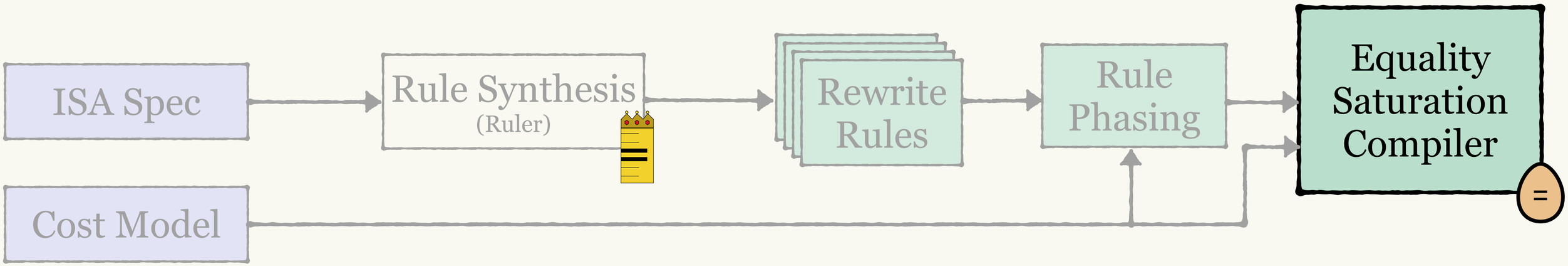


egg: Willsey, et al. (POPL '21)

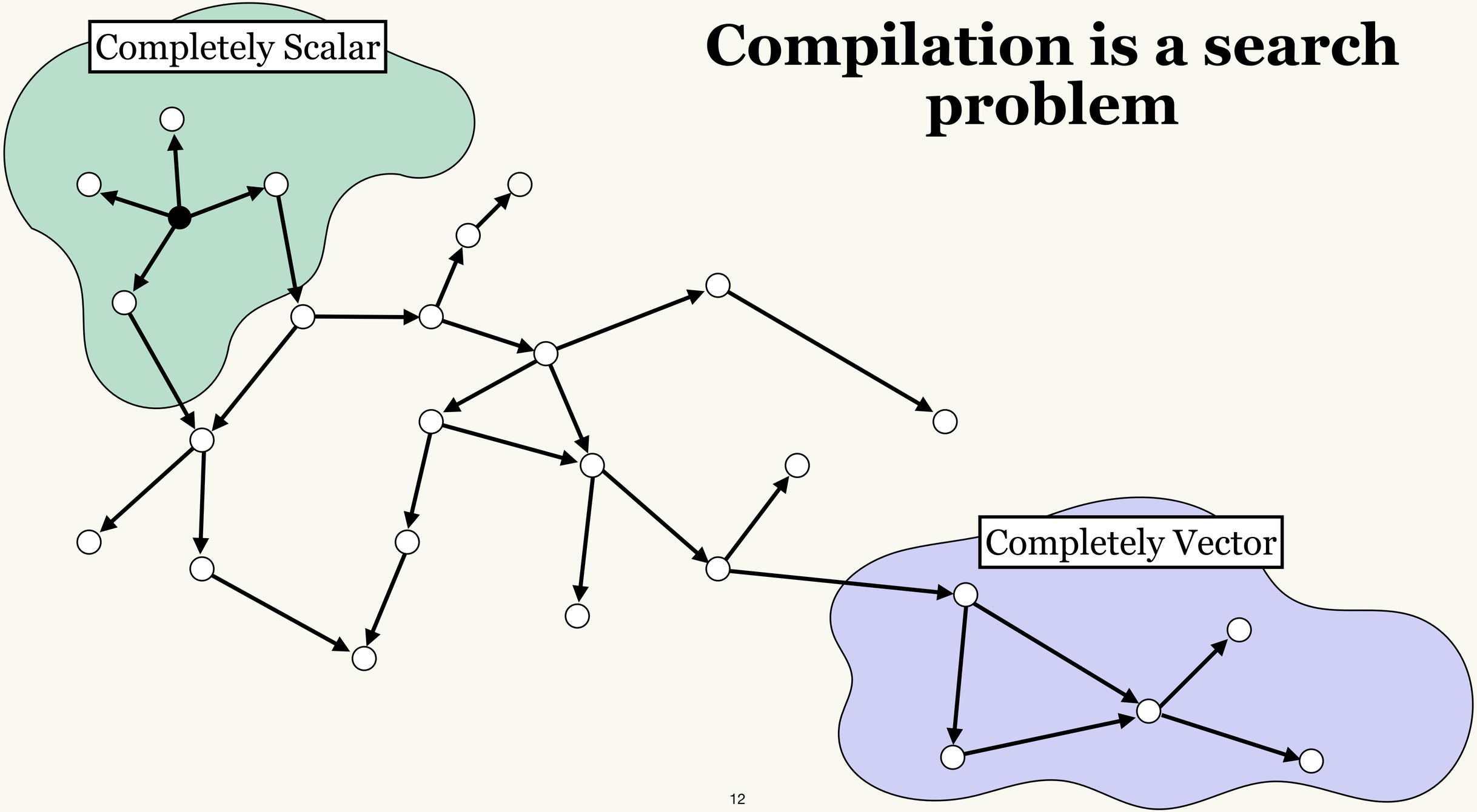
Ruler: Nandi, et al. (OOPSLA '21)

Diospyros: Alexa VanHattum, et al (ASPLOS '21)

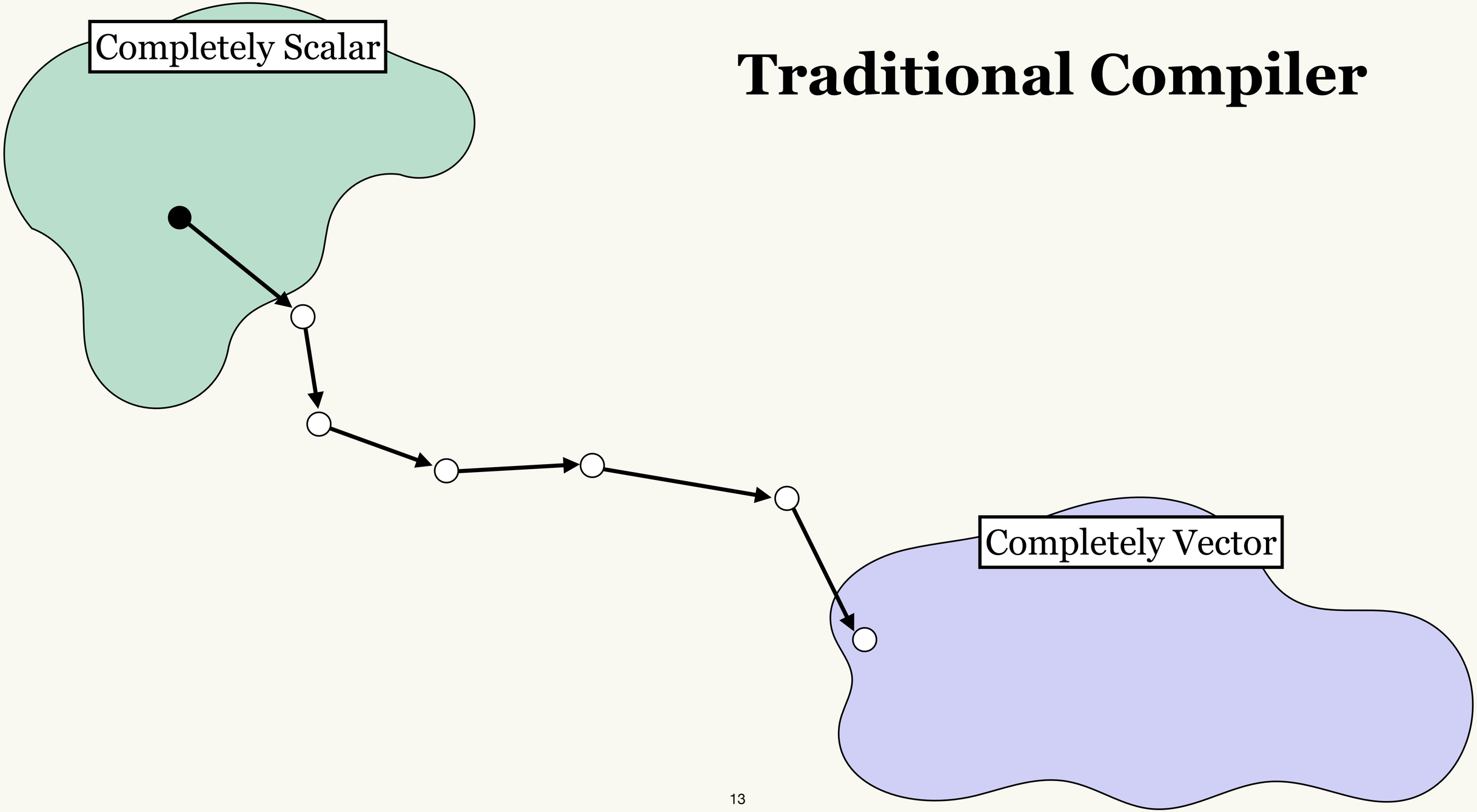
Isaria

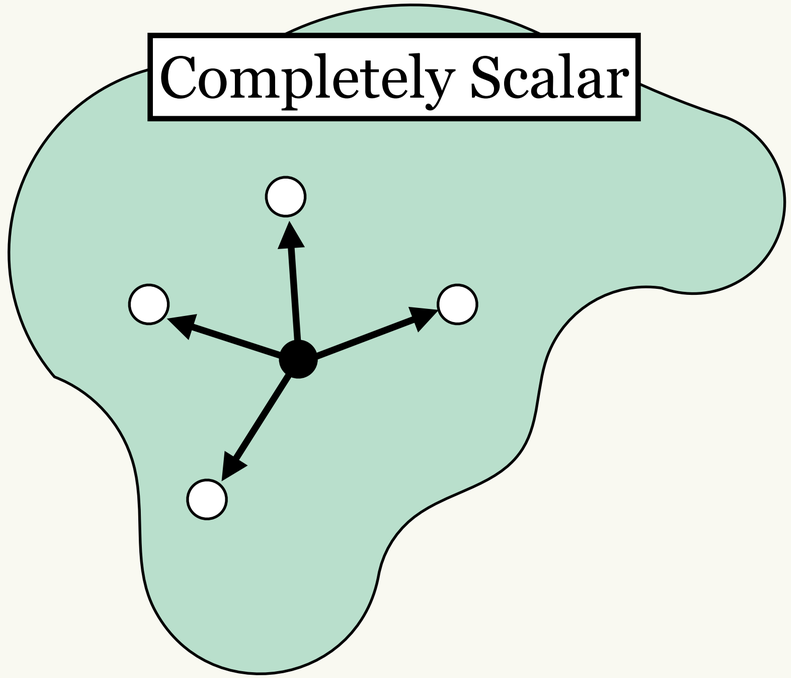

ISA Spec
Cost Model
Rule Synthesis
(Ruler)
Rewrite Rules
Rule Phasing
Equality Saturation Compiler
=

Compilation is a search problem



Traditional Compiler



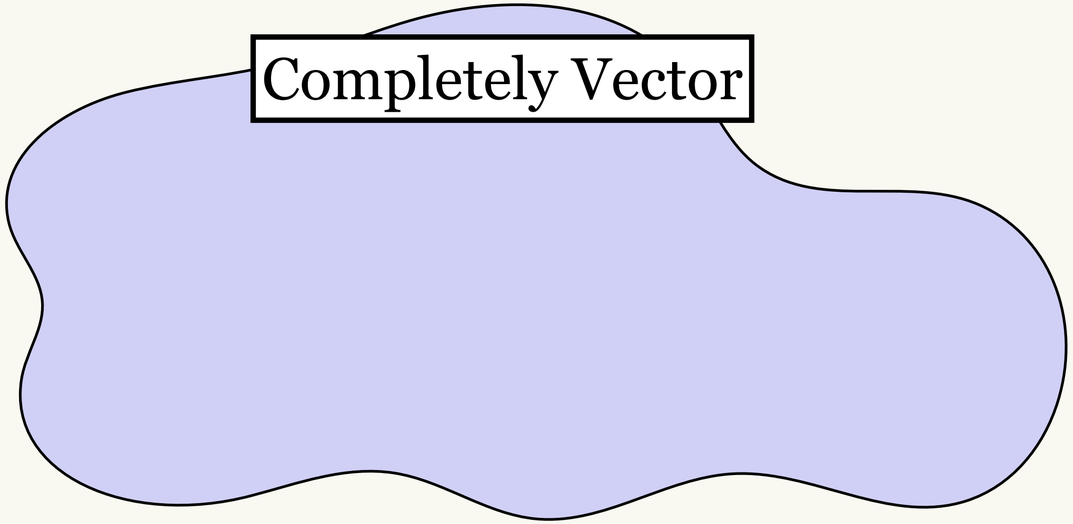


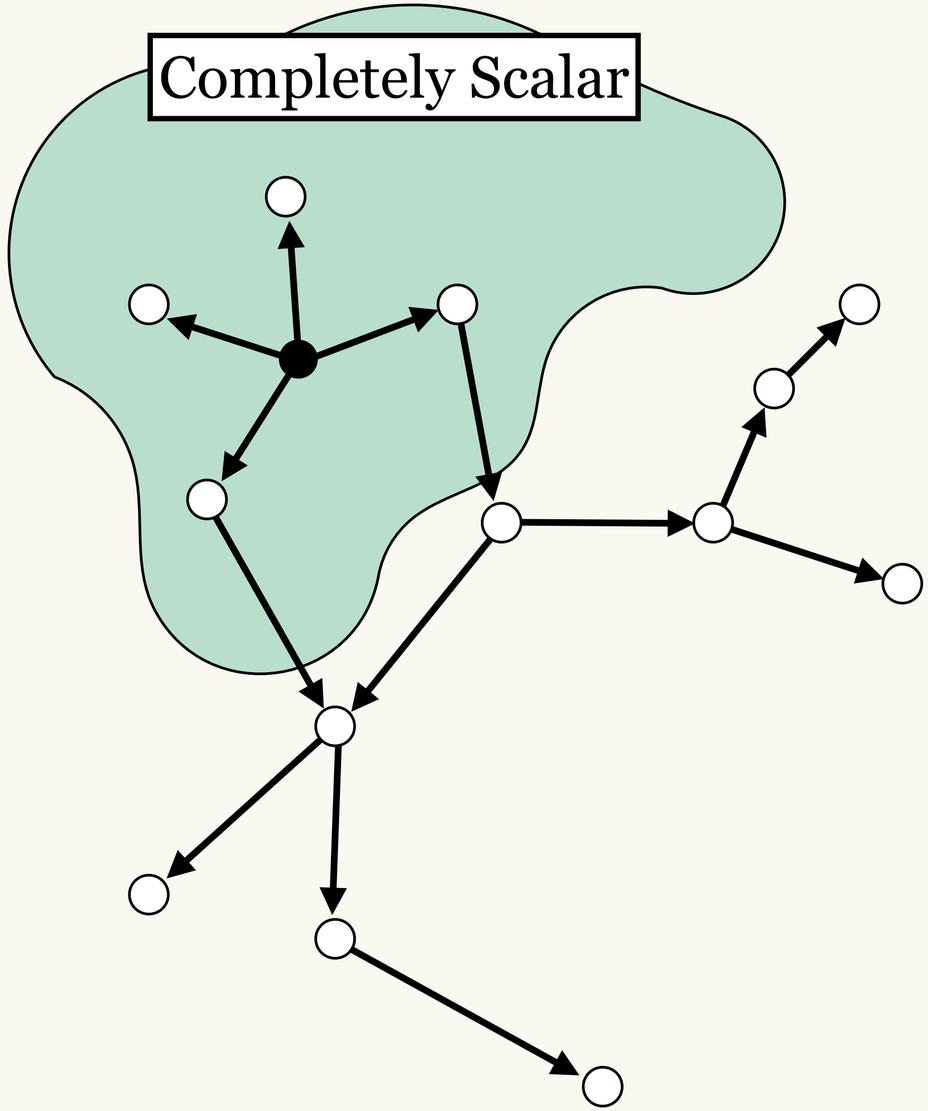
Equality Saturation Compiler

```
(let x (+ a b))          (VecAdd
  (let y (+ c d)) ⇒        (Vec a c)
    (Vec x y)))            (Vec b d))
```

```
(+ a 0) ⇒ a
```

```
(VecAdd
  (VecMul a b) ⇒ (VecMAC a b c)
  c)
```



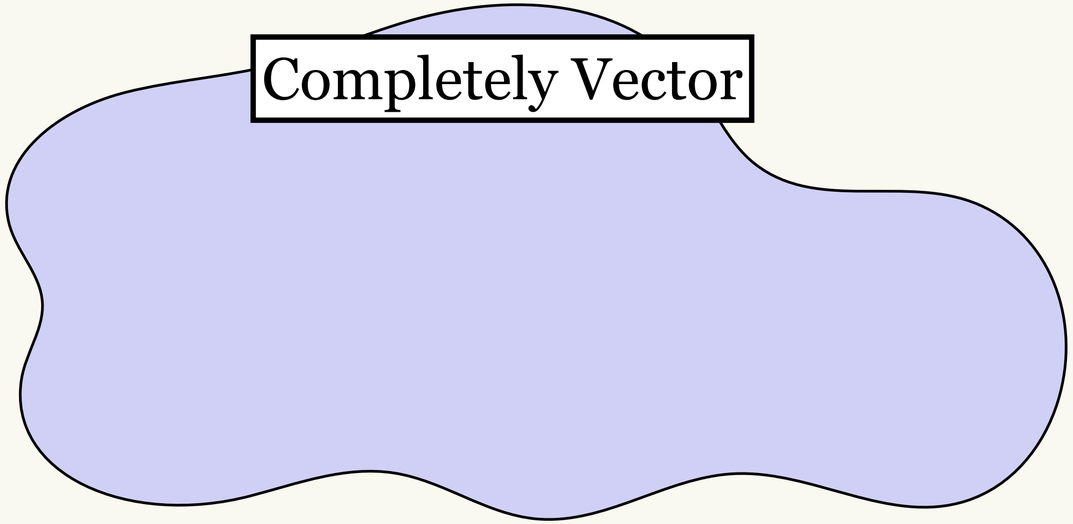


Equality Saturation Compiler

```
(let x (+ a b)          (VecAdd
  (let y (+ c d)  ⇒       (Vec a c)
    (Vec x y)))           (Vec b d))
```

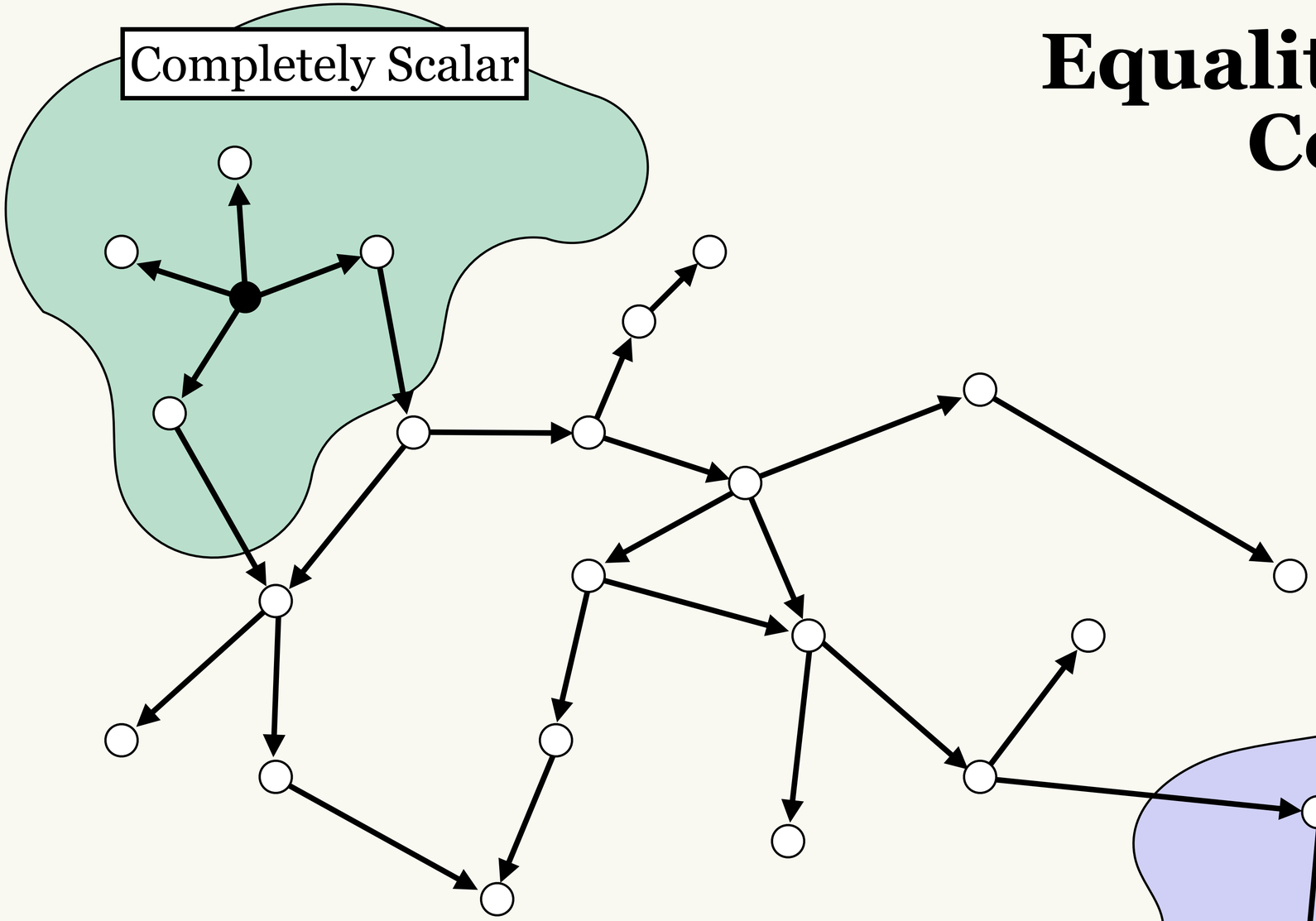
```
(+ a 0) ⇒ a
```

```
(VecAdd
  (VecMul a b) ⇒ (VecMAC a b c)
  c)
```

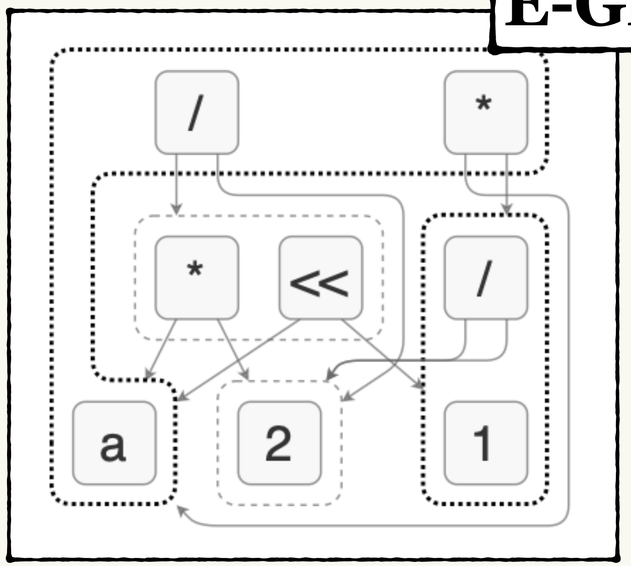


Equality Saturation Compiler

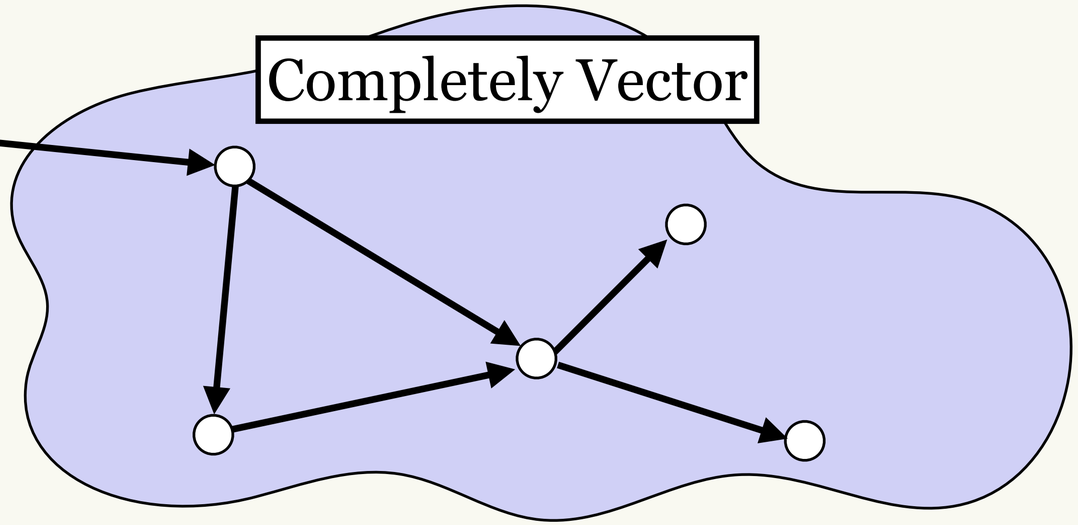
Completely Scalar



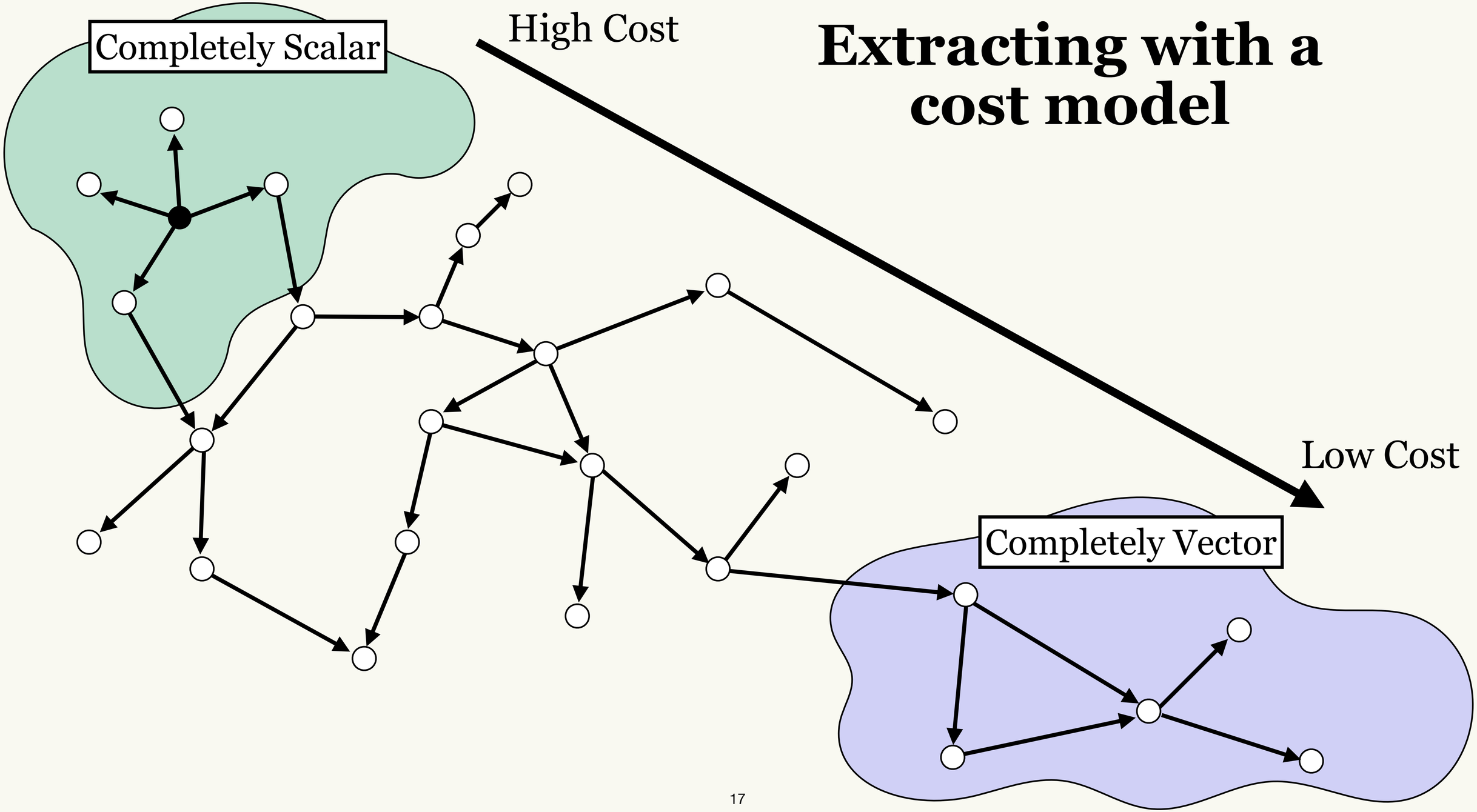
E-Graphs!



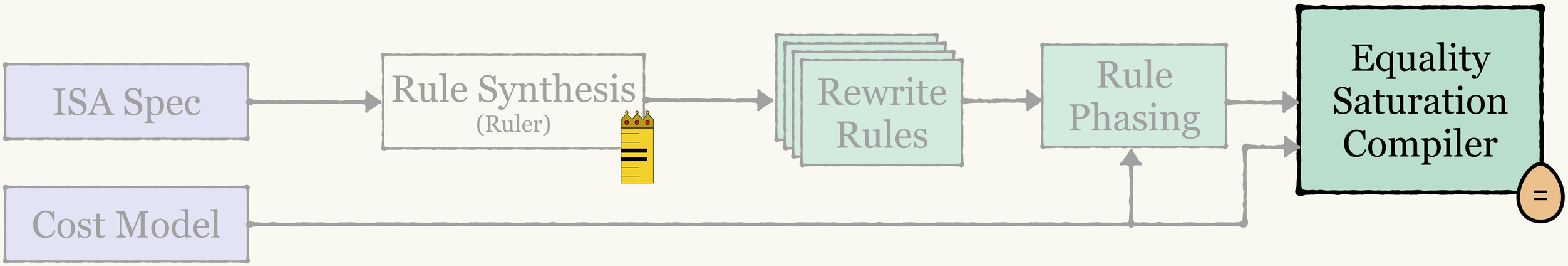
Completely Vector



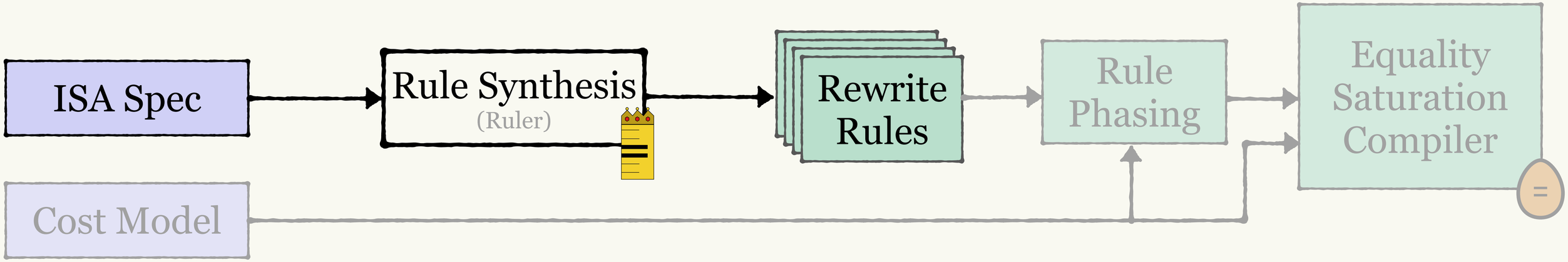
Extracting with a cost model



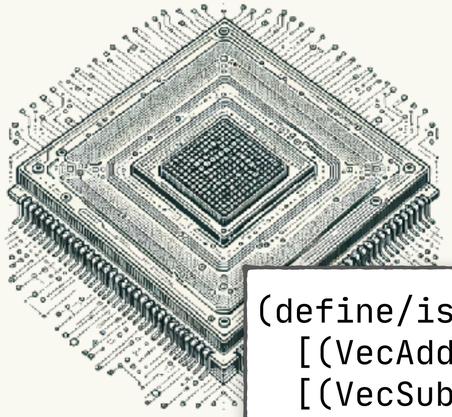
Isaria


ISA Spec
Cost Model
Rule Synthesis
(Ruler)
Rewrite Rules
Rule Phasing
Equality Saturation Compiler
=

Isaria


ISA Spec
Rule Synthesis
(Ruler)
Rewrite Rules
Rule Phasing
Equality Saturation Compiler
=
Cost Model

Generating Rewrite Rules



```
(define/isa
  [(VecAdd x y) ..]
  [(VecSub x y) ..]
  [(+ x y) ..]
  ..)
```

Term Enumeration modulo equivalence

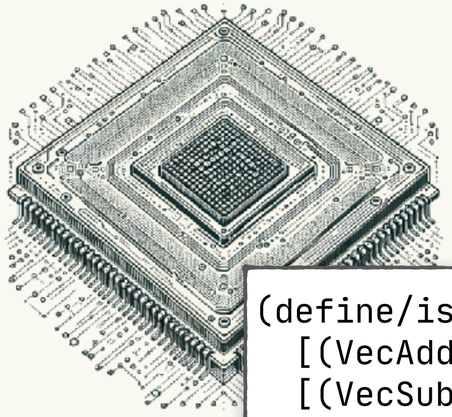
Selection and Validation

Rewrite Rules

```
(Vec (+ ?a ?b)) ⟺ (VecAdd (Vec ?a) (Vec ?b))
(Vec (* ?a ?b)) ⟺ (VecMul (Vec ?a) (Vec ?b))
(+ ?a ?b) ⟺ (+ ?b ?a)
(* ?a ?b) ⟺ (* ?b ?a)
```

Ruler: Nandi, et al. (OOPSLA '21)

Too many rules…



```
(define/isa
  [(VecAdd x y) ..]
  [(VecSub x y) ..]
  [(+ x y) ..]
  ..)
```

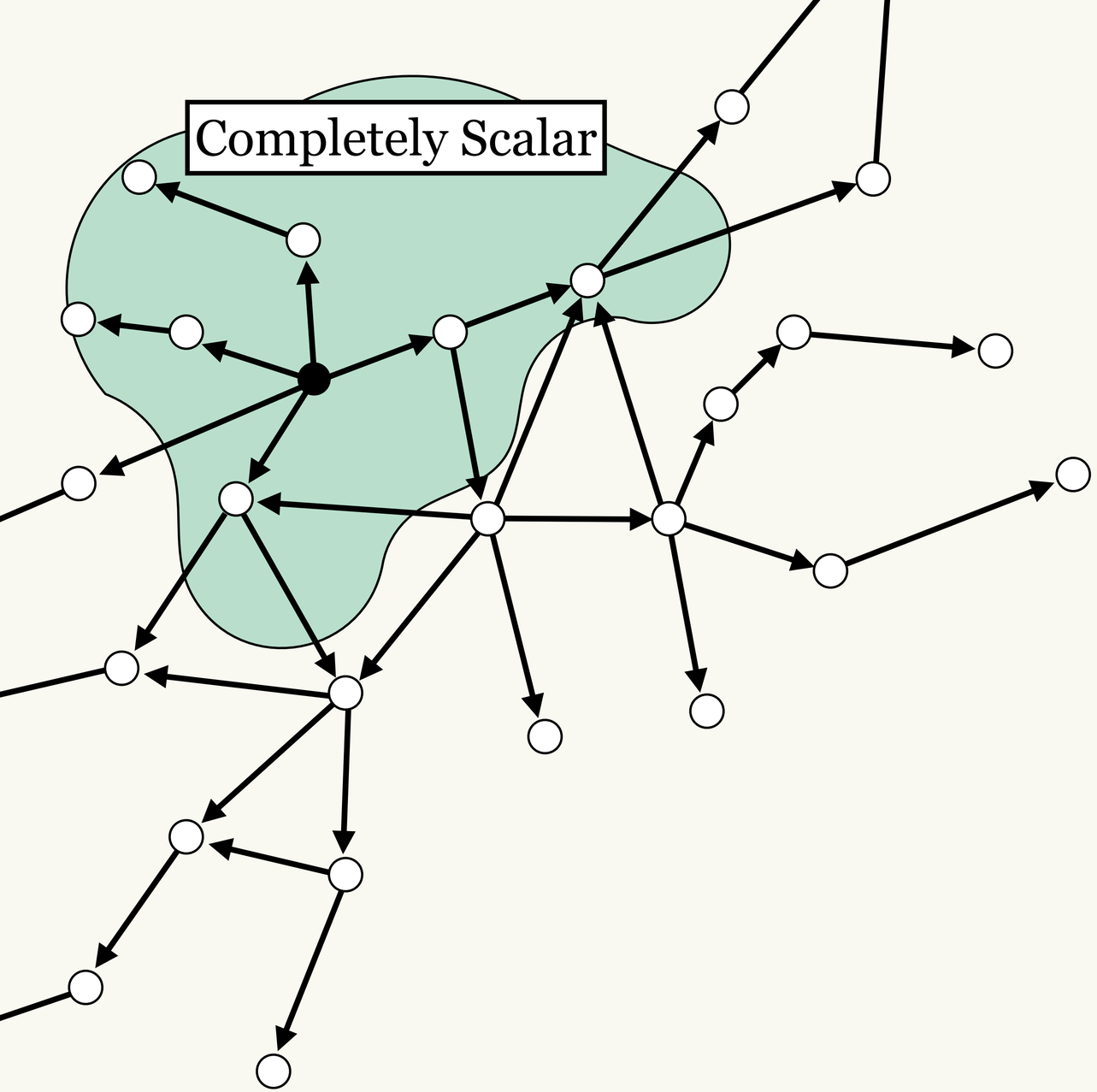
Term Enumeration modulo equivalence

Selection and Validation

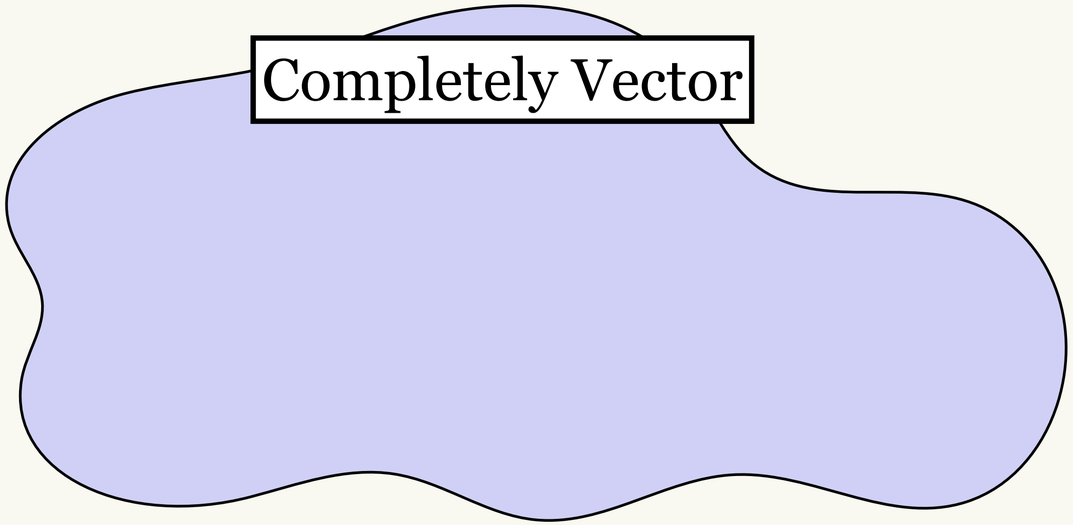
Rewrite Rules

```
(Vec (+ ?a ?b)) ⟺ (VecAdd (Vec ?a) (Vec ?b))
(Vec (* ?a ?b)) ⟺ (VecMul (Vec ?a) (Vec ?b))
(+ ?a ?b) ⟺ (+ ?b ?a)
(* ?a ?b) ⟺ (* ?b ?a)
(VecAdd ?b ?a) ⟺ (VecAdd ?a ?b)
(VecMinus ?b (VecNeg ?a)) ⟺ (VecAdd ?a ?b)
(sgn ?a) ⟺ (sgn (sgn ?a))
(neg (* ?b ?a)) ⟺ (* ?b (neg ?a))
(VecMAC ?c ?b ?a) ⟺ (VecAdd ?c (VecMul ?b ?a))
(VecMul ?b ?a) ⟺ (VecMul ?a ?b)
(/ 0 ?a) ⟺ (/ 0 (sgn ?a))
(VecNeg (VecMul ?b ?a)) ⟺ (VecMul ?a (VecNeg ?b))
```

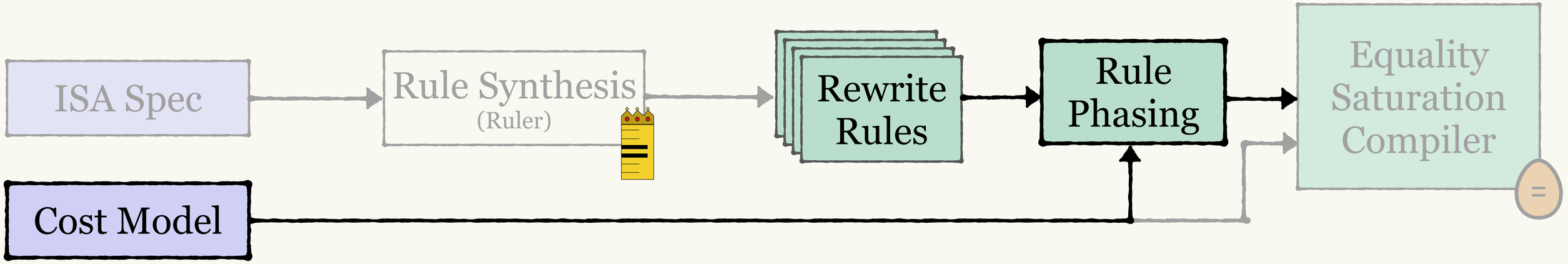
Ruler: Nandi, et al. (OOPSLA '21)



Too many rules…



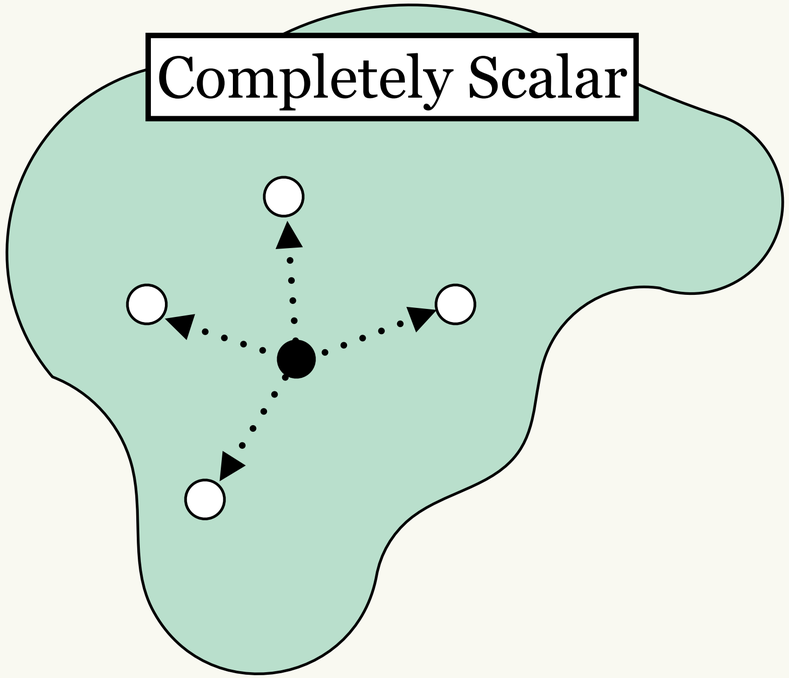
Isaria

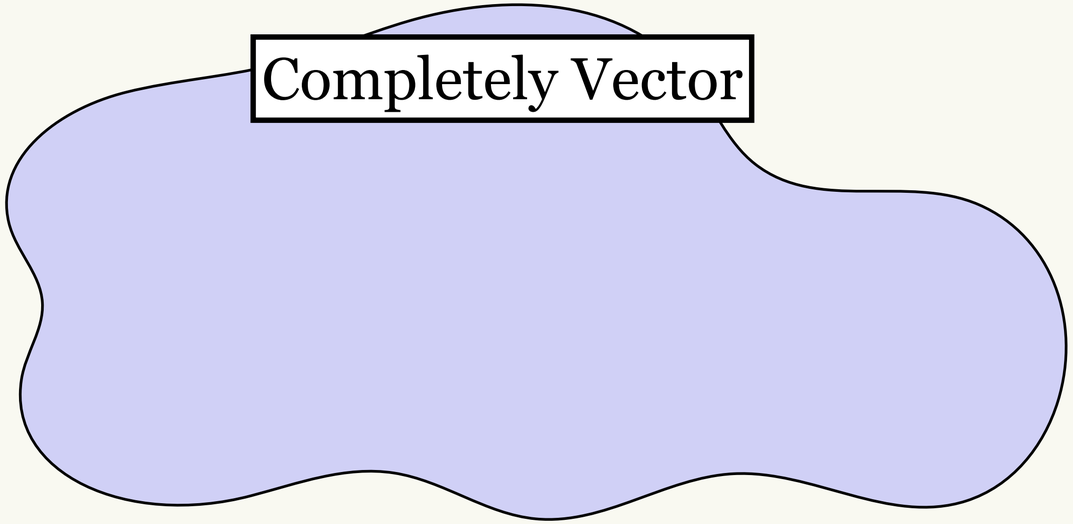

ISA Spec
Rule Synthesis
(Ruler)
Rewrite Rules
Rule Phasing
Equality Saturation Compiler
=
Cost Model

1. Rule Phasing

2. Pruning

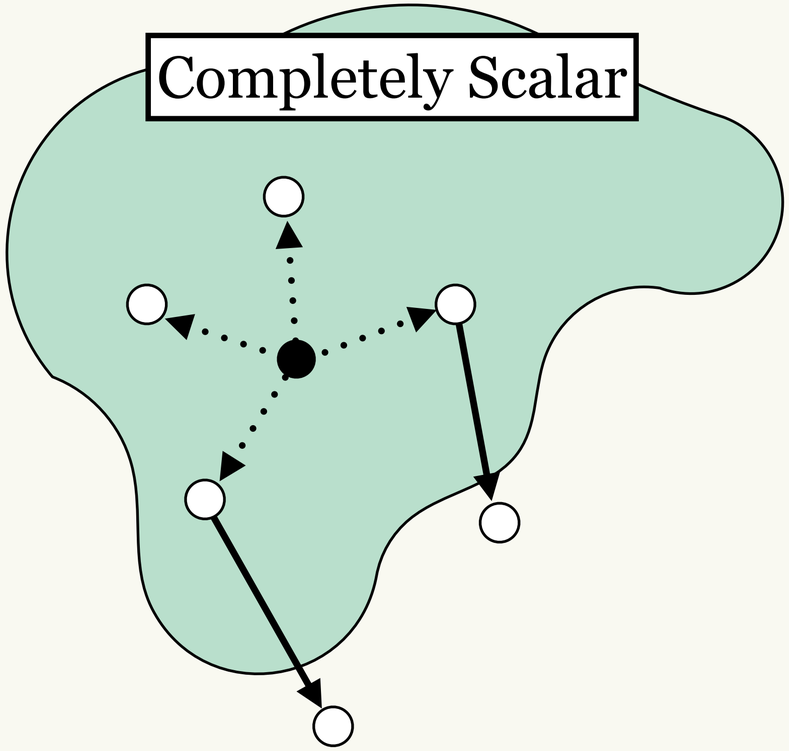
Expansion





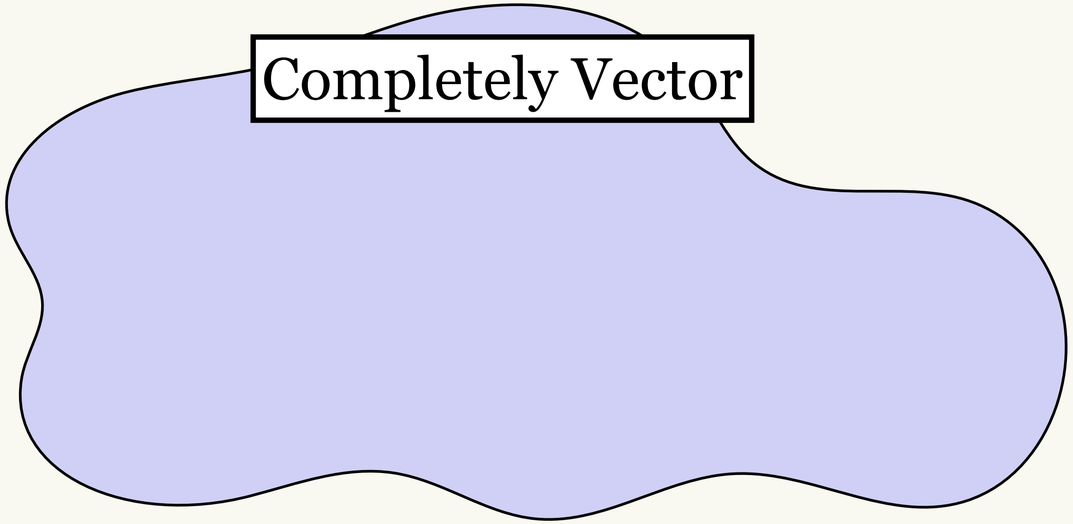
1. Rule Phasing

2. Pruning



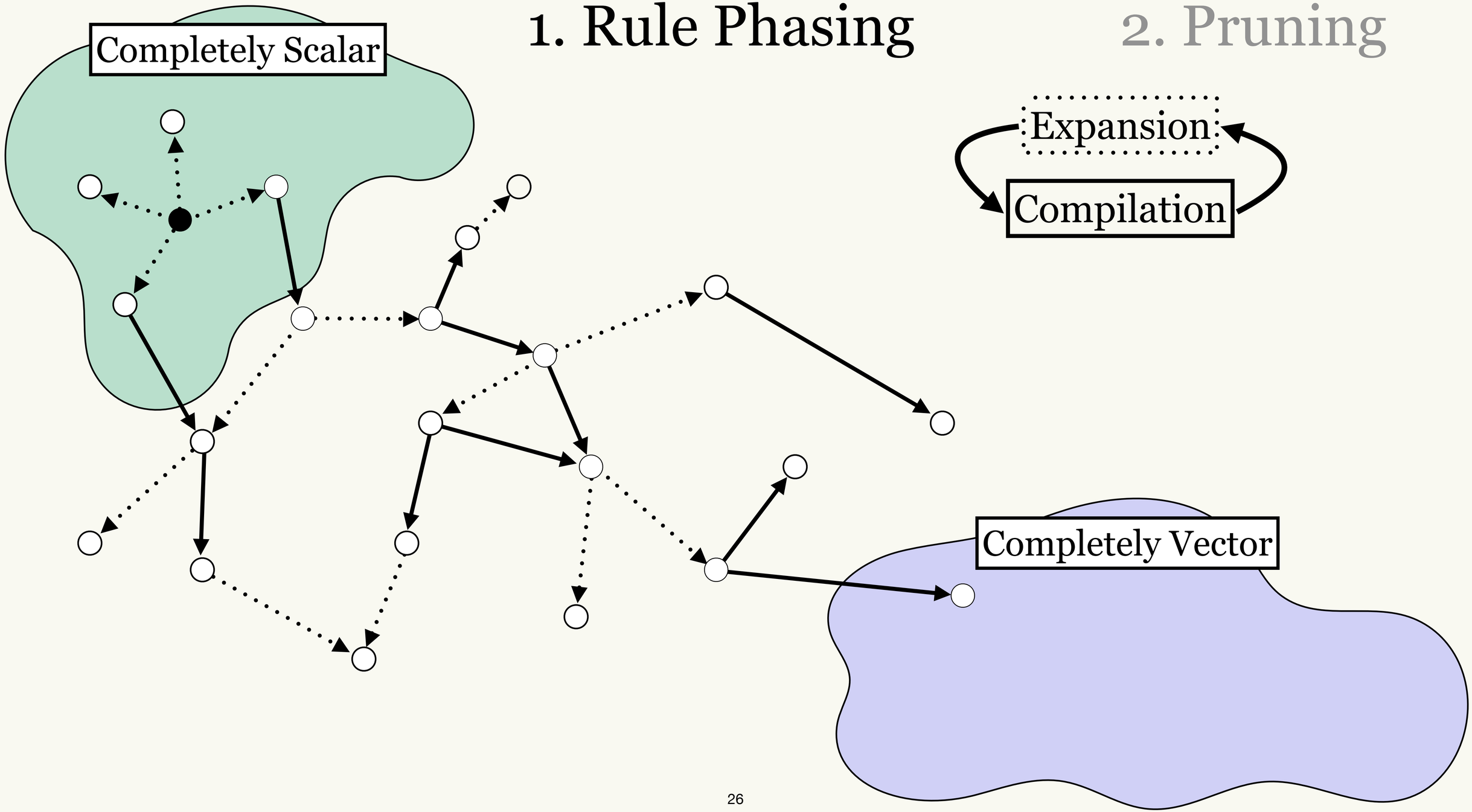
Expansion

Compilation



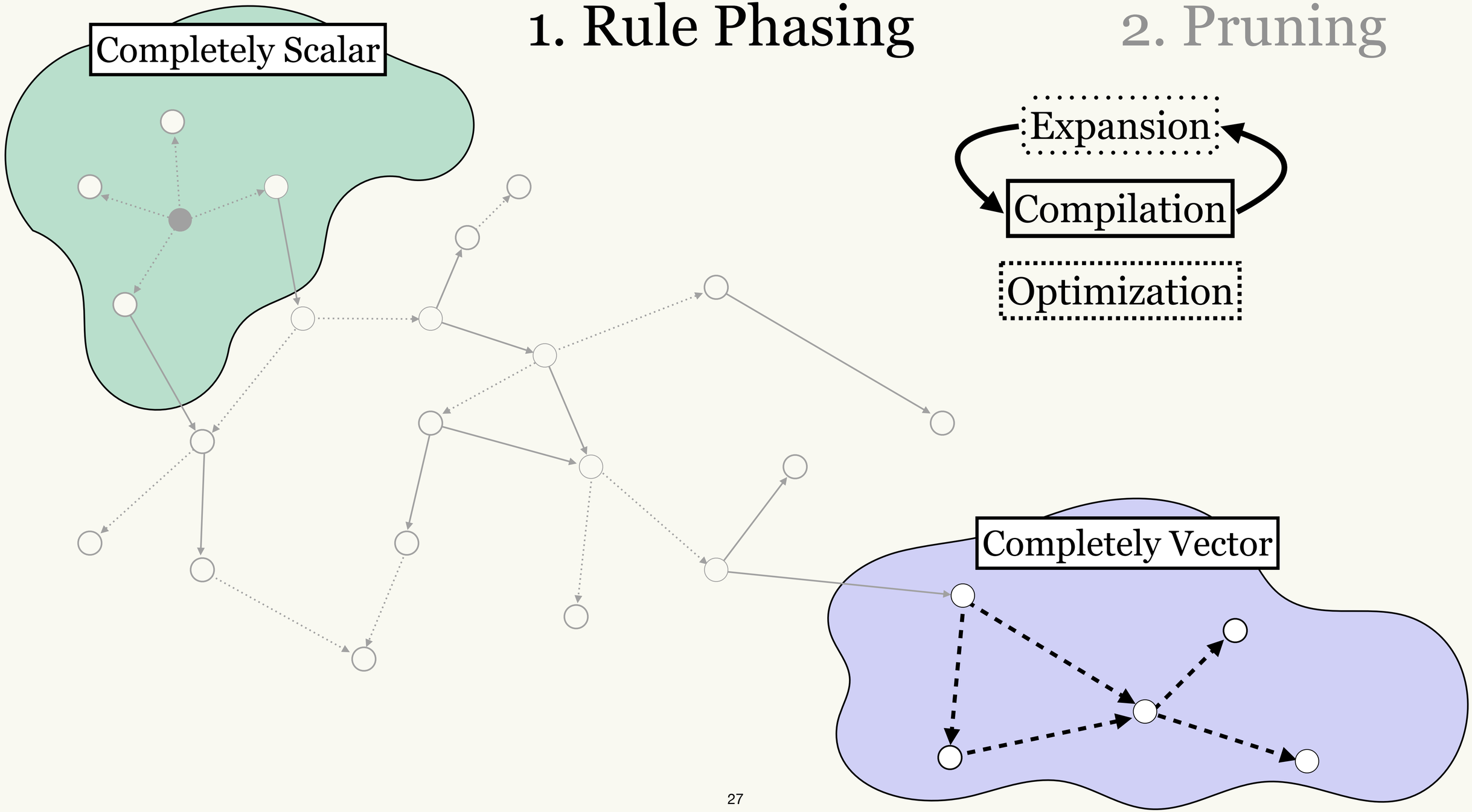
1. Rule Phasing

2. Pruning

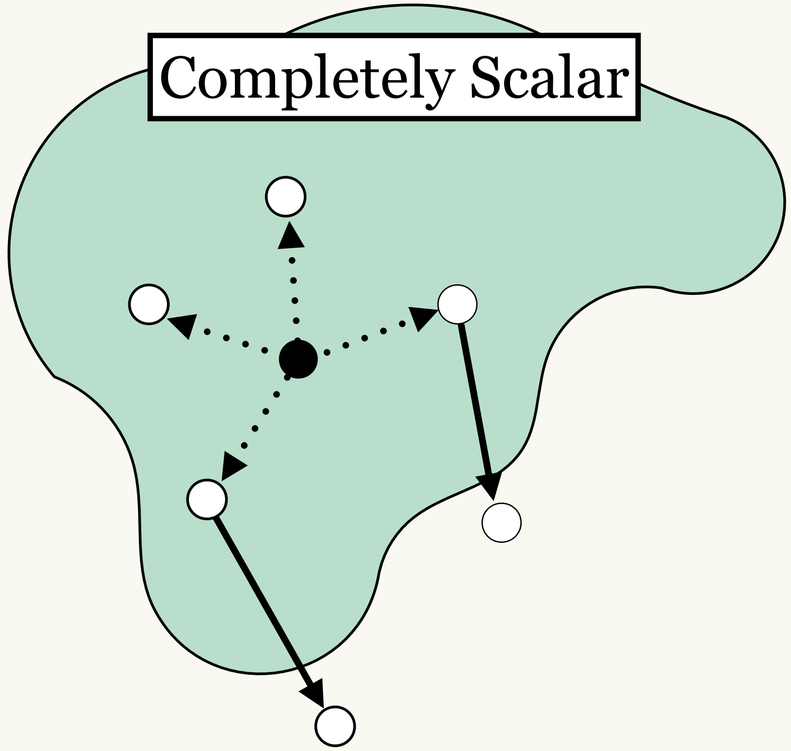


1. Rule Phasing

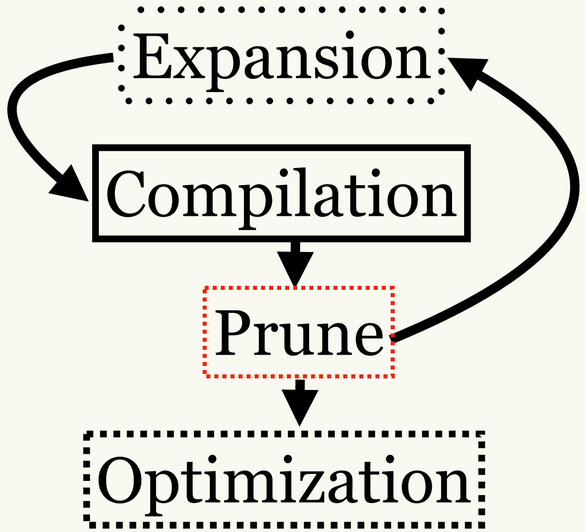
2. Pruning

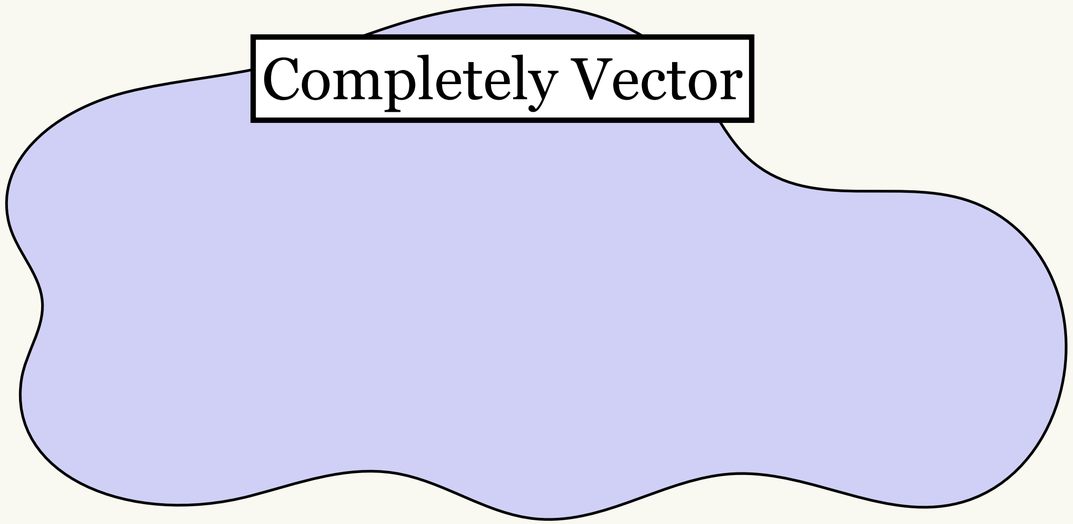


1. Rule Phasing

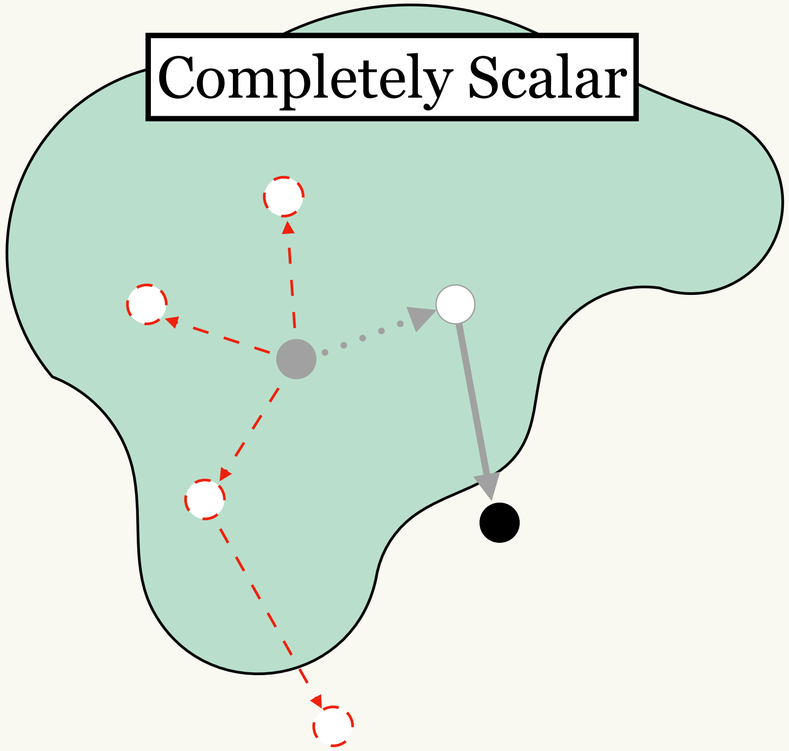


2. Pruning

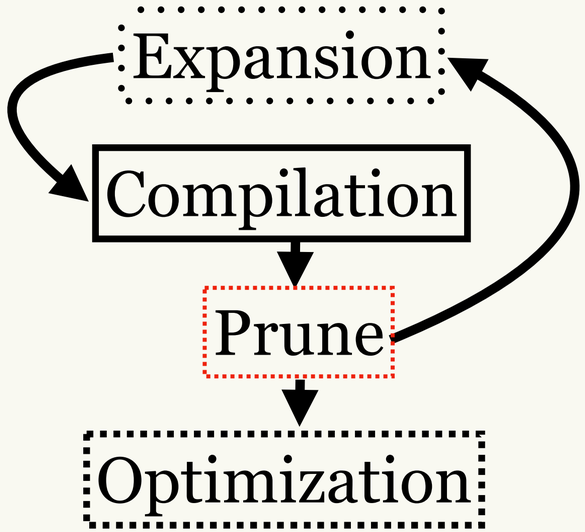


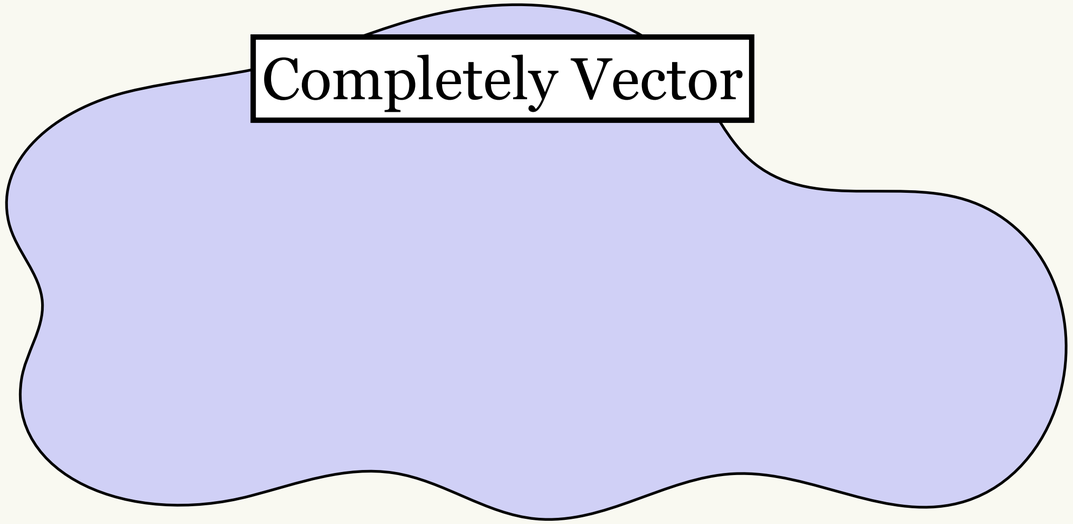


1. Rule Phasing

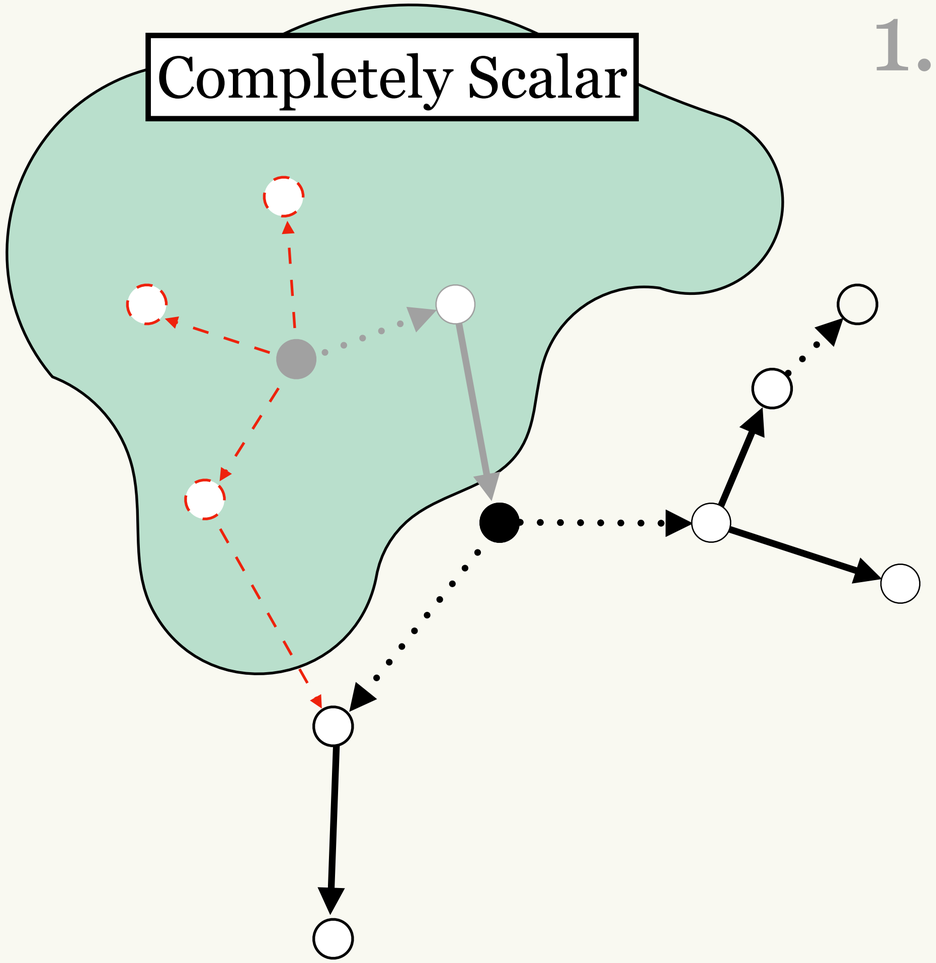


2. Pruning

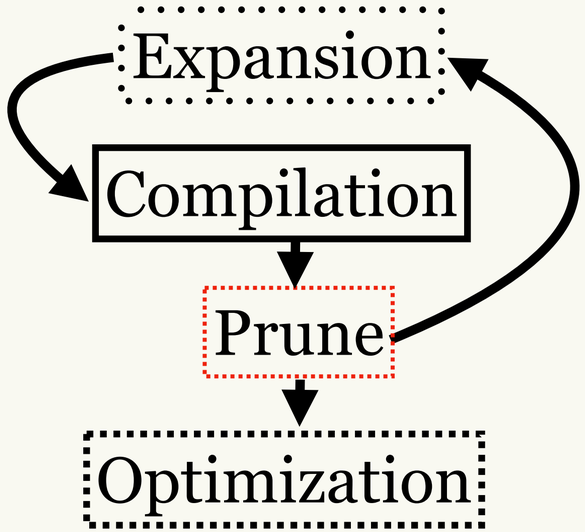


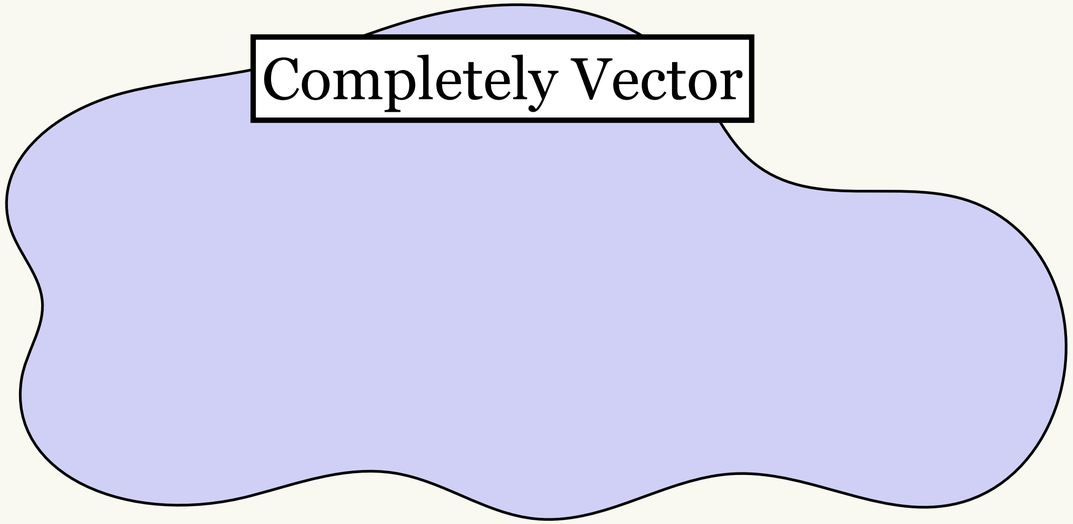


1. Rule Phasing

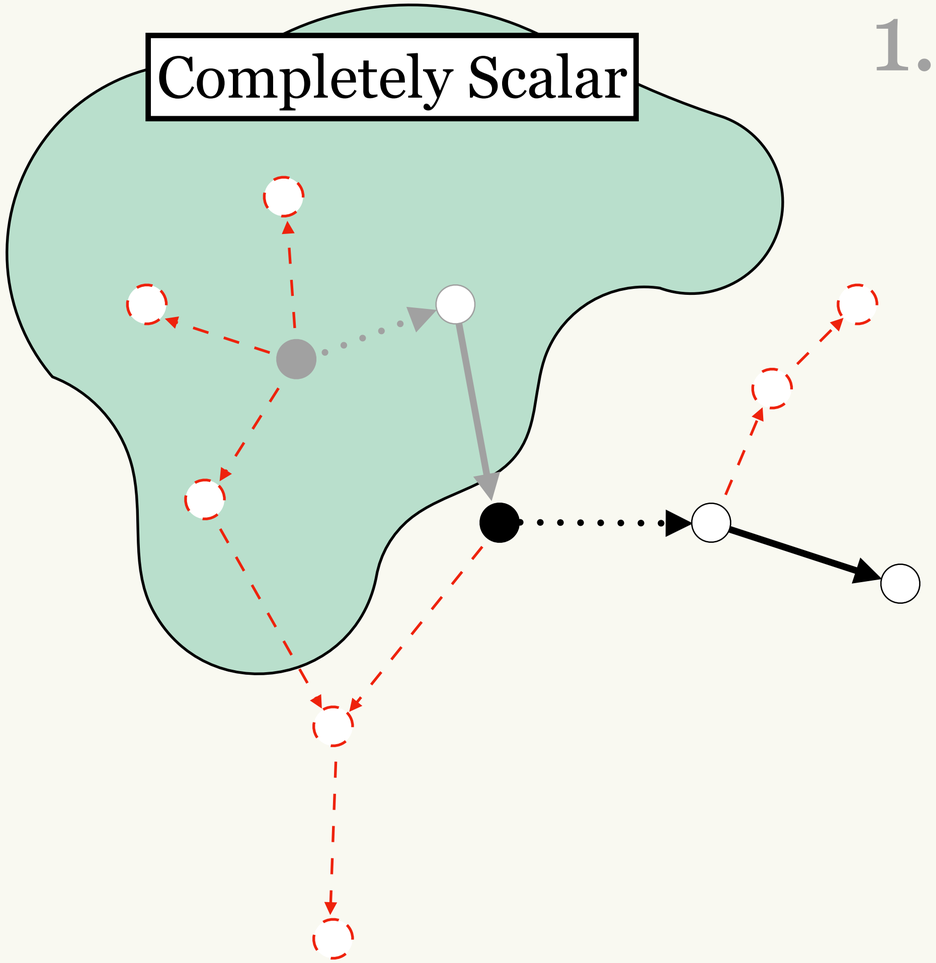


2. Pruning

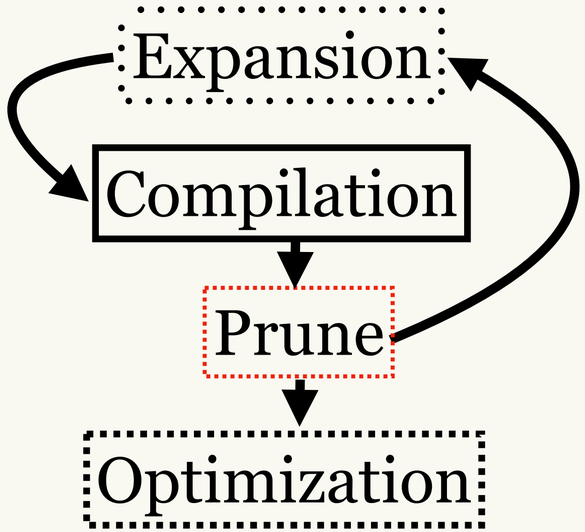


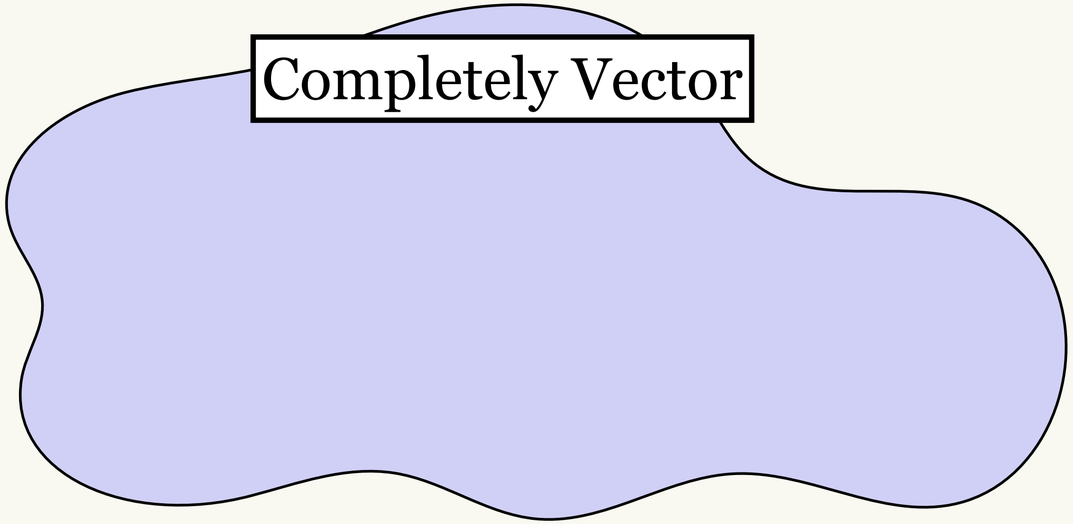


1. Rule Phasing



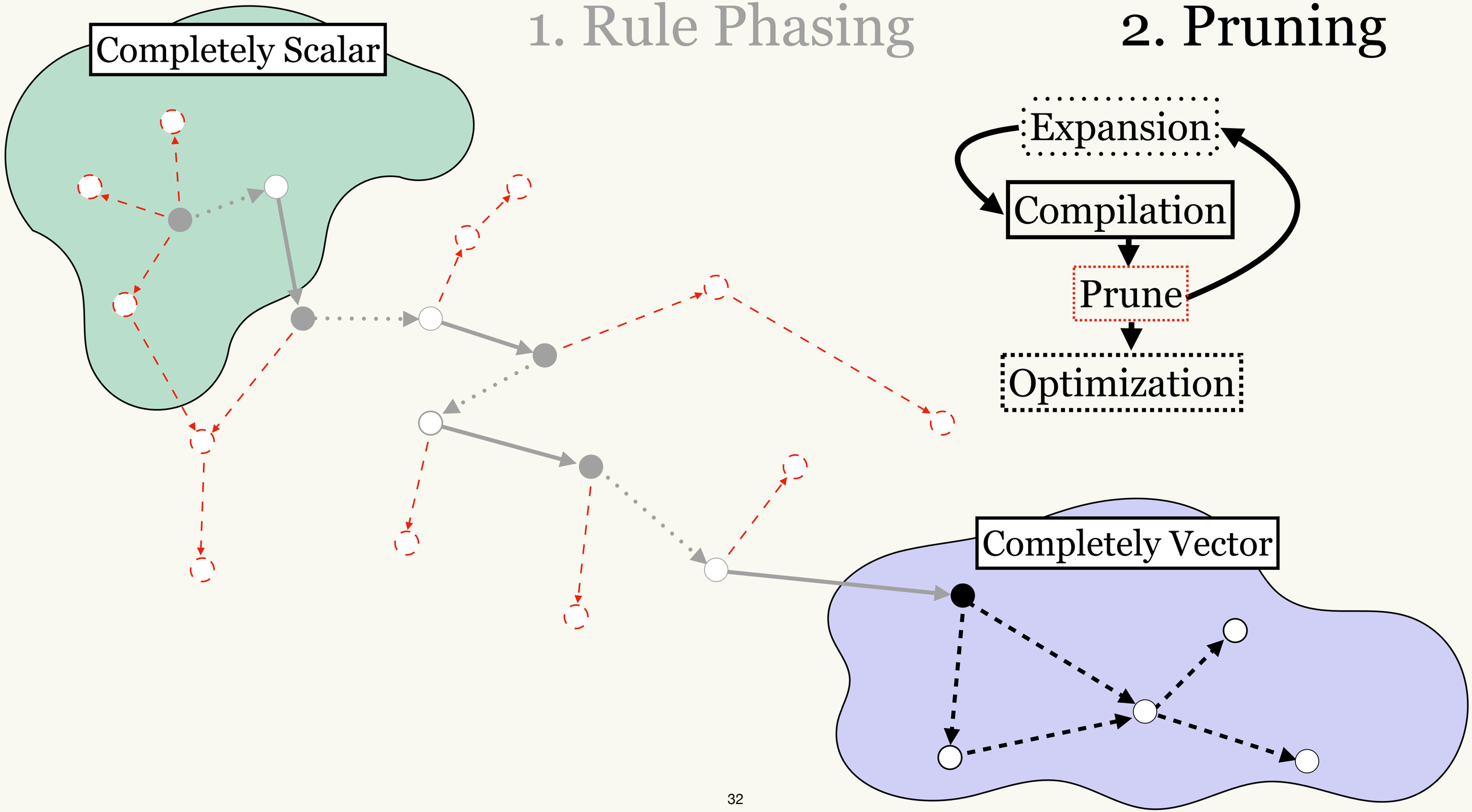
2. Pruning





1. Rule Phasing

2. Pruning



Isaria: Choosing Phases

Expansion

Rewrite Rules

```
(Vec (+ ?a ?b)) ⟺ (VecAdd (Vec ?a) (Vec ?b))
(Vec (* ?a ?b)) ⟺ (VecMul (Vec ?a) (Vec ?b))
(+ ?a ?b) ⟺ (+ ?b ?a)
(* ?a ?b) ⟺ (* ?b ?a)
(VecAdd ?b ?a) ⟺ (VecAdd ?a ?b)
(VecMinus ?b (VecNeg ?a)) ⟺ (VecAdd ?a ?b)
(sgn ?a) ⟺ (sgn (sgn ?a))
(neg (* ?b ?a)) ⟺ (* ?b (neg ?a))
(VecMAC ?c ?b ?a) ⟺ (VecAdd ?c (VecMul ?b ?a))
(VecMul ?b ?a) ⟺ (VecMul ?a ?b)
(/ 0 ?a) ⟺ (/ 0 (sgn ?a))
(VecNeg (VecMul ?b ?a)) ⟺ (VecMul ?a (VecNeg ?b))
⋮
```

ISA Cost Model

Compilation

Optimization

Isaria: Choosing Phases

Expansion

```
(+ ?a ?b) ⟺ (+ ?b ?a)
(* ?a ?b) ⟺ (* ?b ?a)
(sgn ?a) ⟺ (sgn (sgn ?a))
(neg (* ?b ?a)) ⟺ (* ?b (neg ?a))
(/ 0 ?a) ⟺ (/ 0 (sgn ?a))
```

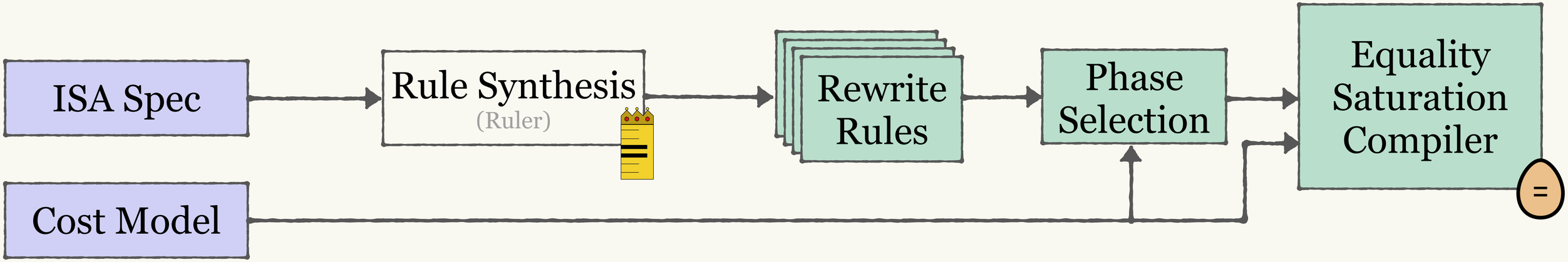
Compilation

```
(Vec (+ ?a ?b)) ⇒ (VecAdd (Vec ?a) (Vec ?b))
(Vec (* ?a ?b)) ⇒ (VecMul (Vec ?a) (Vec ?b))
```

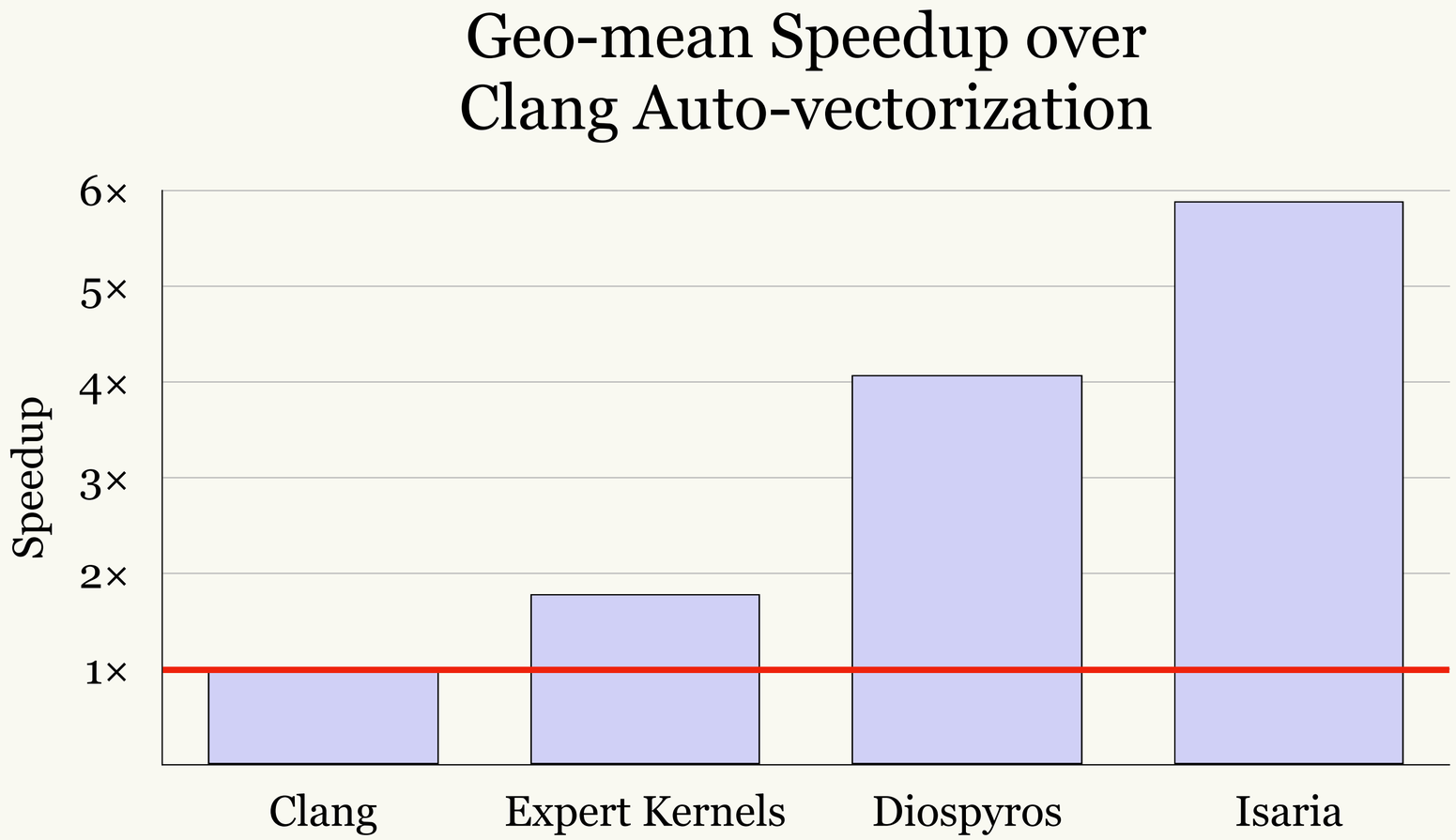
Optimization

```
(VecAdd ?b ?a) ⟺ (VecAdd ?a ?b)
(VecMinus ?b (VecNeg ?a)) ⟺ (VecAdd ?a ?b)
(VecMAC ?c ?b ?a) ⟺ (VecAdd ?c (VecMul ?b ?a))
(VecMul ?b ?a) ⟺ (VecMul ?a ?b)
(VecNeg (VecMul ?b ?a)) ⟺ (VecMul ?a (VecNeg ?b))
```

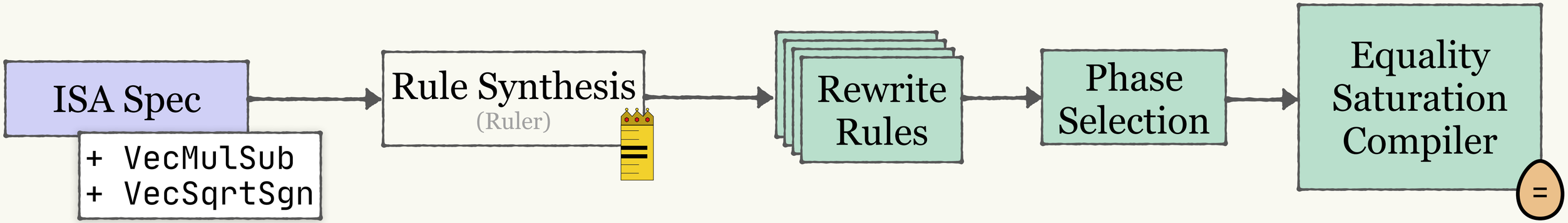
Isaria


ISA Spec
Cost Model
Rule Synthesis
(Ruler)
Rewrite Rules
Phase Selection
Equality Saturation Compiler
=

How do we compare against handwritten compilers?



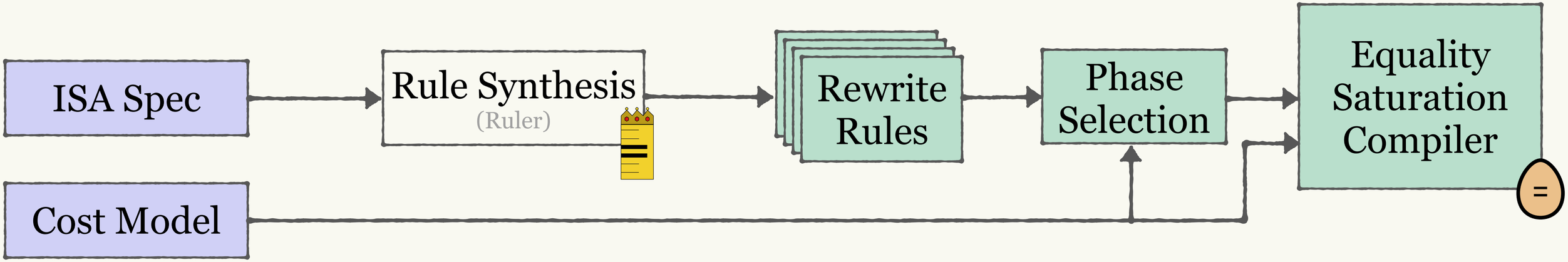
Case Study



	VecMulSub	**No VecMulSub**
VecSqrtSgn	2.0%	1.7%
No VecSqrtSgn	0.5%	-

Performance improvement for QR-Decomposition

Thanks!


ISA Spec
Cost Model
Rule Synthesis
(Ruler)
Rewrite Rules
Phase Selection
Equality Saturation Compiler
=