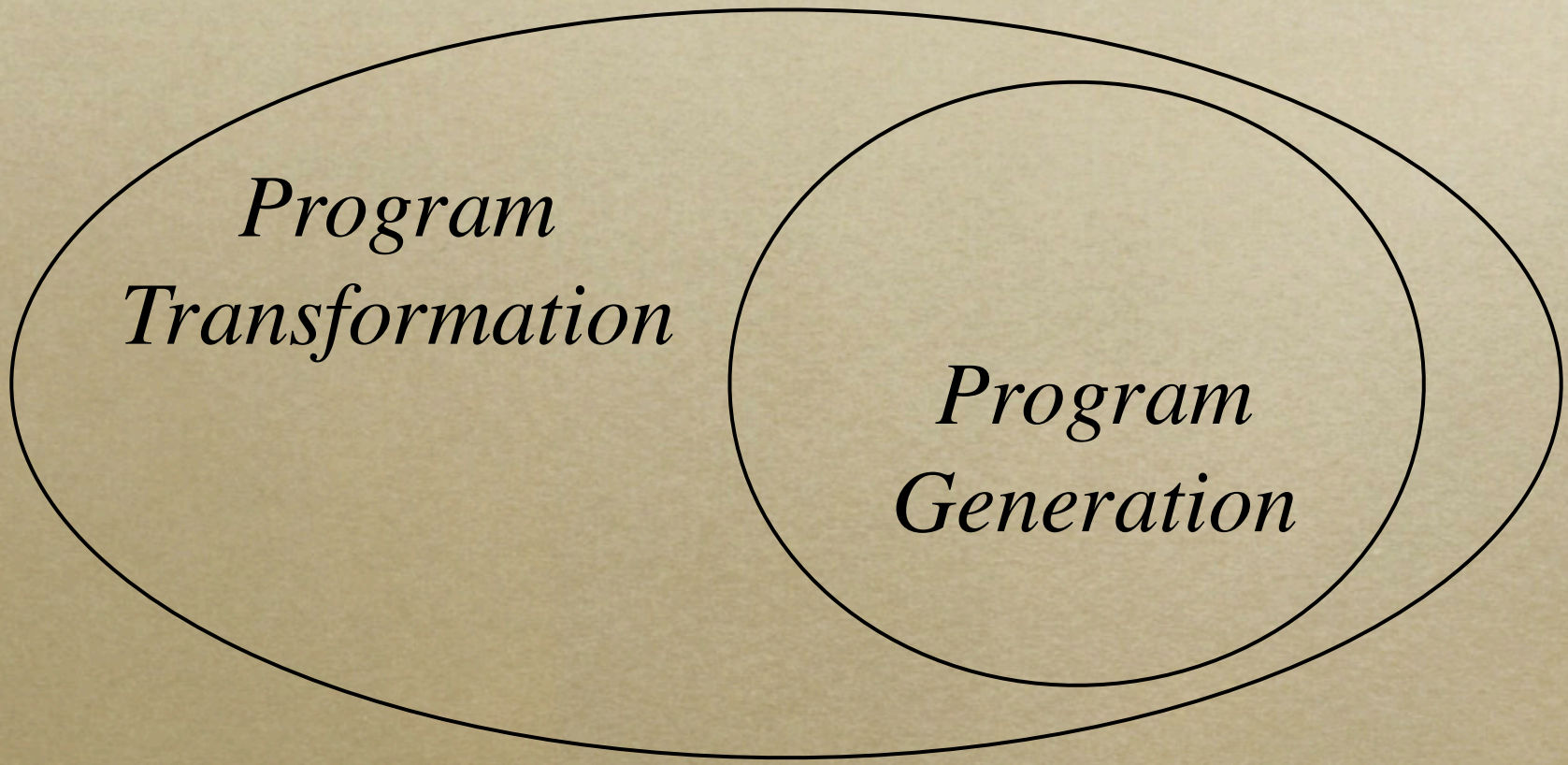
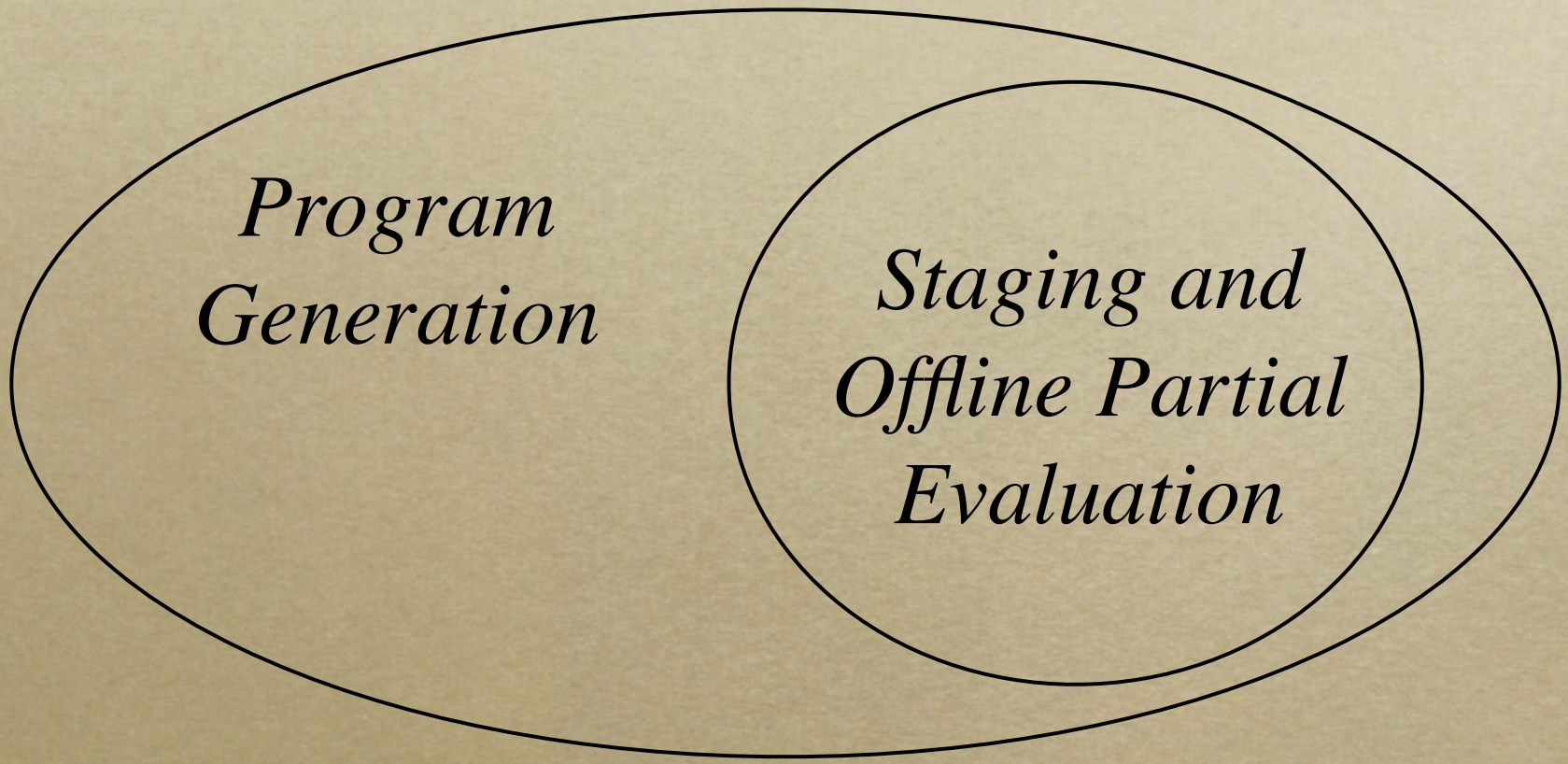
Staging
15 Years Later

Walid Taha
Halmstad University, Halmstad, Sweden
Rice University, Houston, TX, USA

About “Walid”

- *94-99: PhD at OGI w/ Prof. Sheard*
- *99-00: Post-doc at Chalmers, w/ Prof. Hughes*
- *00-02: Researcher, Yale, w/ Prof. Hudak*
- *02-10: Assistant Professor at Rice*
- *10-: Professor at Halmstad*


Program
Transformation
Program
Generation


Program Generation
Staging and Offline Partial Evaluation

Staging *a la* 1997

Program 	***Staged or Annotated Program***
a = 1	*a = 1*
b = 2	*b = 2*
c = 3	*c = 3*
*d = (x+a)-(b*c)*	*d = run "(x+a) - `(lift (b*c))"*
... at runtime	***... at runtime (2nd stage)***
*d = (x+a)-(b*c)*	*d = (x+1)-6*

Since then

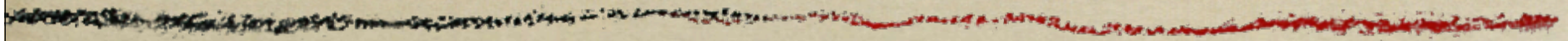
- *Implementations*: *MetaML, MetaOCaml, Template Haskell, Java Mint, BER MetaOCaml, ConCoqtion, lightweight staging libraries, ...*
- *Type systems and semantics*: *Lots!*
- *Programming*: *Tag-less staged interp's, monadic staging, abstract interpretation*

Today

- *Lots technical results amassed!*
  - *Semantics, type systems, formal reasoning principles, implementation techniques, programming case studies*
- *Stepping back, what's emerging picture?*

This Talk

- *Staging as an optimization*
- *Staging and partial evaluation*
- *Things that stage well*
- *The perfect language for staging*
- *What staging types actually do*
- *Conclusion and Challenges*



Staging as an Optimization

Partial Evaluation

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia

Article Talk Read Edit View hist

Our updated Terms of Use will become effecti

Partial evaluation

From Wikipedia, the free encyclopedia

Not to be confused with partial application.

In computing, **partial evaluation** is a technique for several different types of program optimization by specialization. The most straightforward application is to produce new programs which run faster than the originals while being guaranteed to behave in the same way.

Traditional view

- *Given the program*
  - `power(x,n) = if n=1 then x else x*power(x,n-1)`
- *Partial evaluate for n=2*
  - `power2(x) = x*x`
- *Reusability and performance!*

What happens in practice

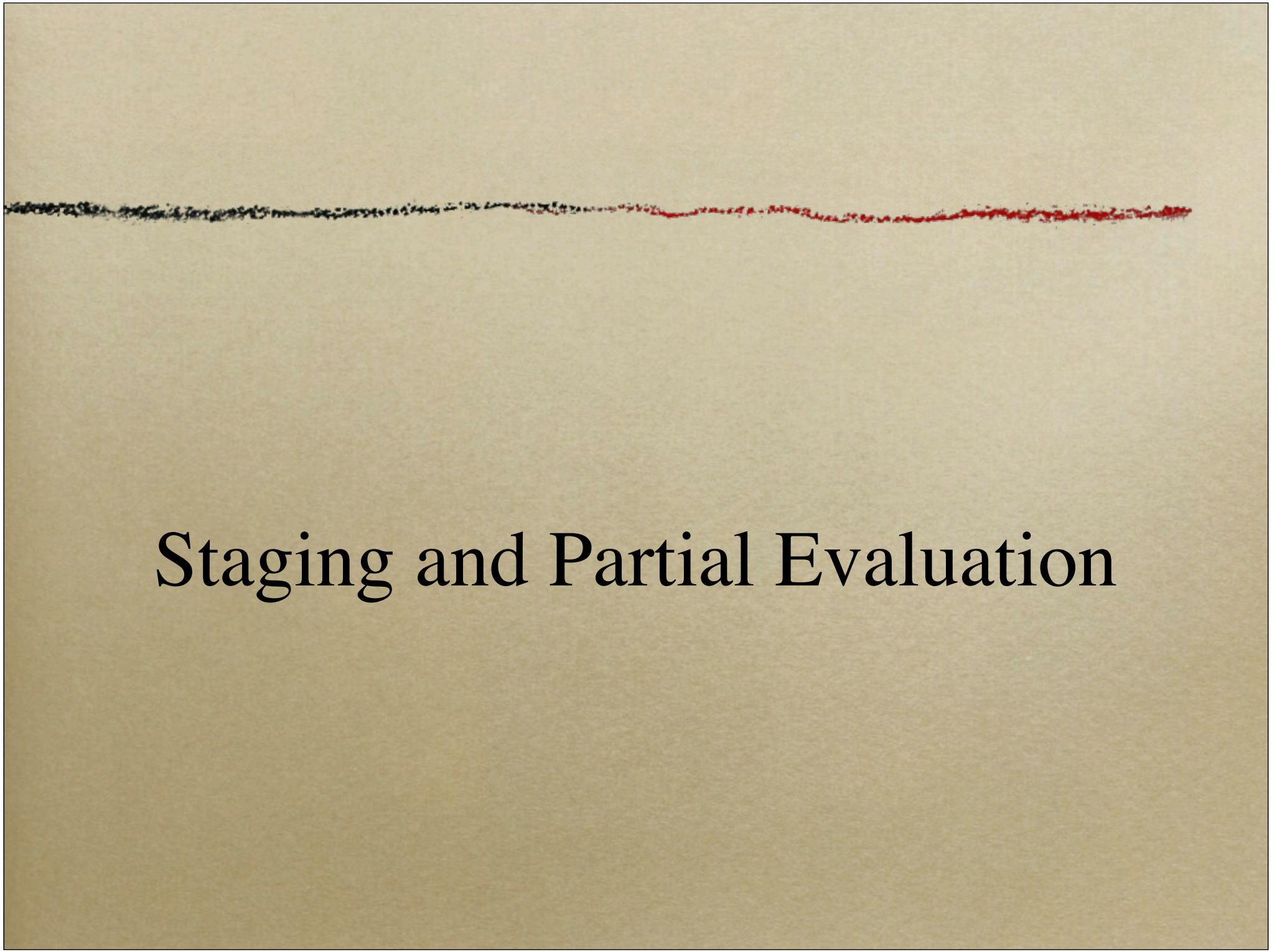
- *Programmer writes*
  - `square(x) = x*x`
  - `cube(x) = x*x*x`
  - `fourthPower(x) = x*x*x*x`
- *Eventually, programmer scratches head*
- *Programmer says "Naaah". Moves on*

Staging as an optimization

- *Partial evaluation and staging can be great optimizations, but they often work best on programs that just don't exist yet*
- *Creating stageable programs is tricky, and is still, in most cases, a big investment. Usually, too big…*

Staging and evaluation order

- *Staging is about very fine control over evaluation order in programs*
- *Traditional strategies*
  - *CBV, CBN only evaluate closed code*
- *What if you want to be MORE strict?*
- *Go under binders. Introduces open code*


Staging and Partial Evaluation

Staging *a la* 1997

Program 	***Staged or Annotated Program***
a = 1	*a = 1*
b = 2	*b = 2*
c = 3	*c = 3*
*d = (x+a)-(b*c)*	*d = run "(x+a) - `(lift (b*c))"*
... at runtime	***... at runtime (2nd stage)***
*d = (x+a)-(b*c)*	*d = (x+1)-6*

Partial Evaluation *a la* 1985

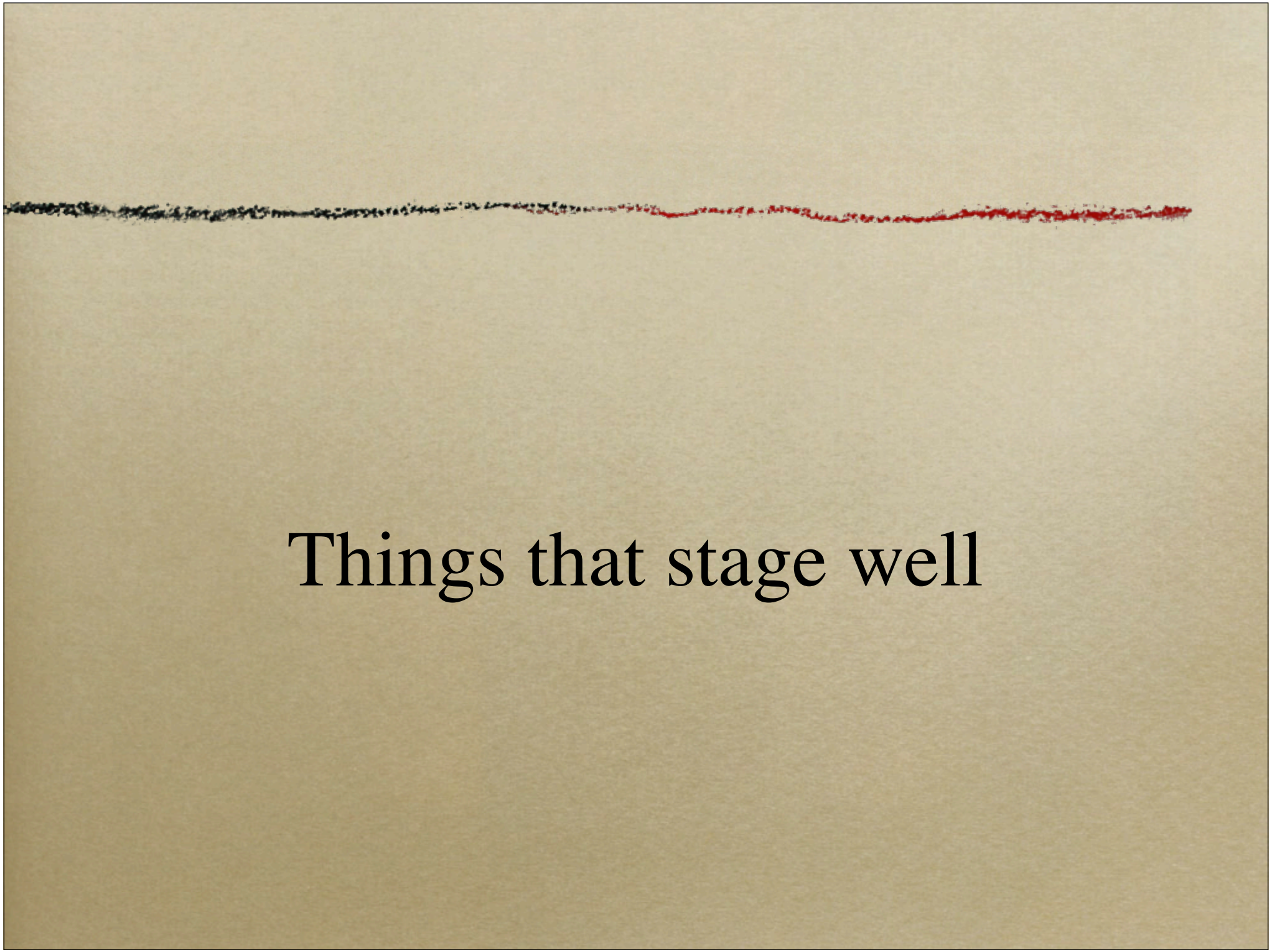
Program	***Binding-Time Annotations***
a = 1	*a = 1*
b = 2	*b = 2*
c = 3	*c = 3*
*d = (x+a)-(b*c)*	*d = run "(x+a) - `(lift (b*c))"*
... at runtime	***Specialized Program***
*d = (x+a)-(b*c)*	*d = (x+1)-6*

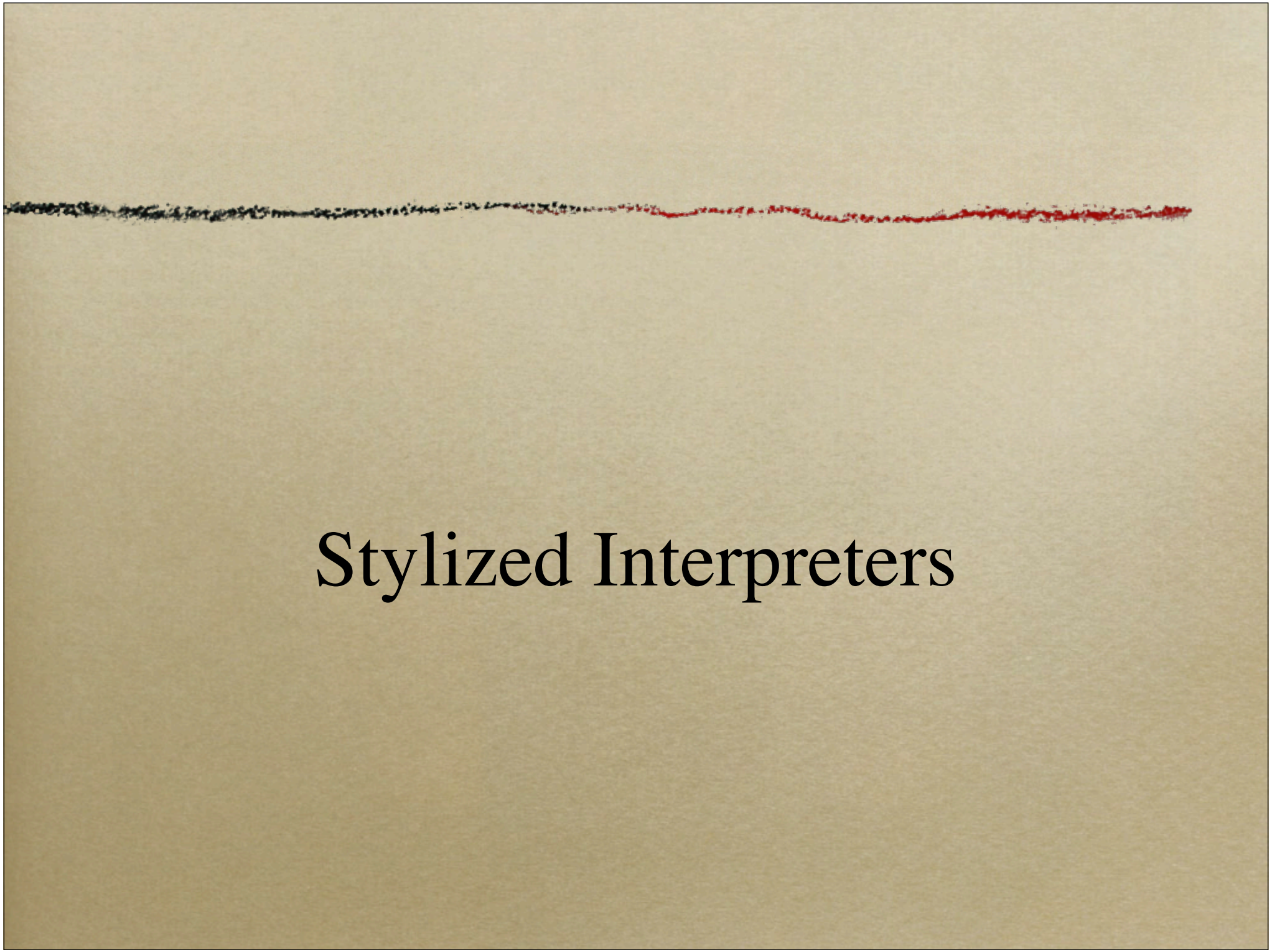
The Paradox

- *Staging cannot do more than partial evaluation (PE)*
- *Staging is less automatic than offline partial evaluation. It's manual binding time analysis*

Staging vs. partial evaluation

- *Traditionally, when "a program did not partial evaluate right", it was hard to figure out why. Manual staging seems to*
  - *help explain to users how partial evaluation works*
  - *help users study the stageability of algorithms*


Things that stage well


Stylized Interpreters

What is an interpreter?

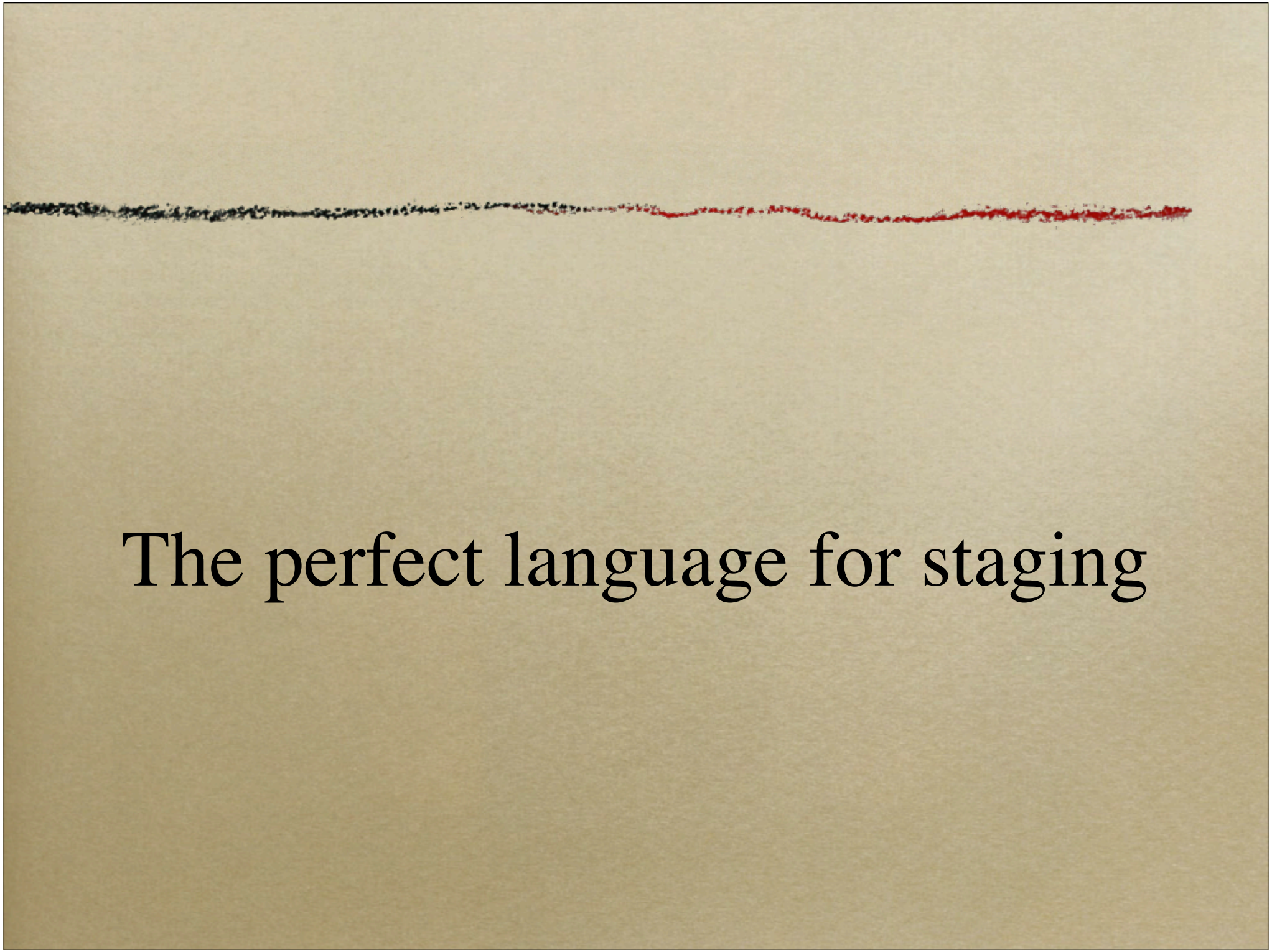
- *It's a pattern!*
  - *Early input (program)*
  - *Late, varying input (the data)*
- *PLs, DSLs, runtime reflection, FFTW*
- *Hygienic macros, HDLs*
- *Software libraries*

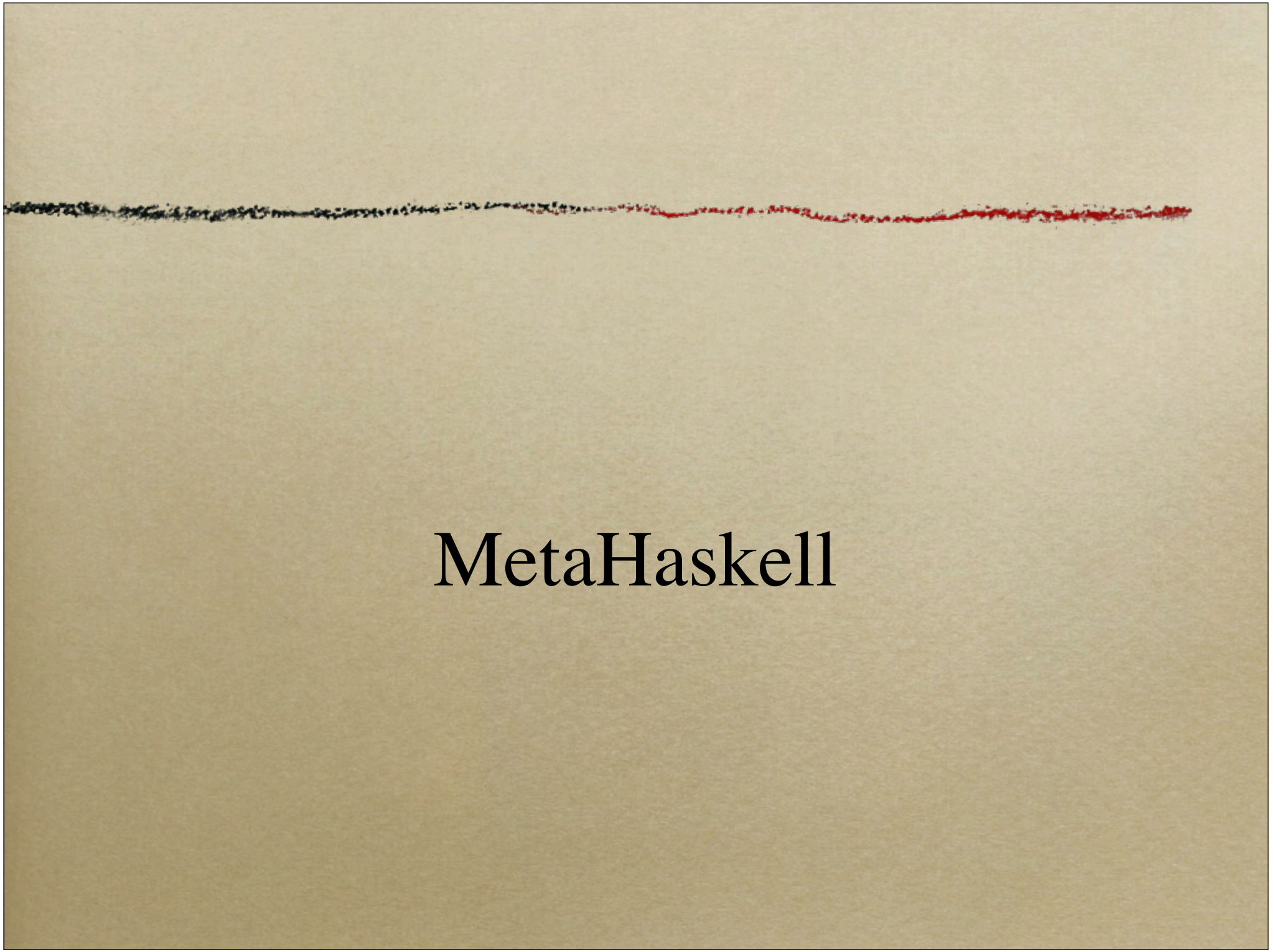
Stylized how?

- *We need it to be stageable*
- *Classic: "What not to do when writing an interpreter for staging", 1996*
- *Denotationally compositional*
- *"Looks like a translation if you squint"*
- *Already in monadic or CPS style*

Hands on tutorial exist

- *"Gentle introduction to multi-stage programming (Parts I and II)"*
- *"DSL implementation in MetaOCaml, Template Haskell, and C++"*


The perfect language for staging

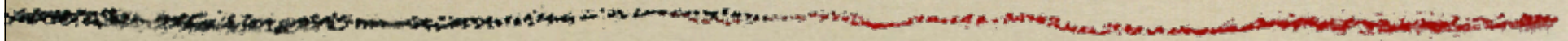

MetaHaskell

Why Haskell?

- *Purely functional, no side effects*
  - *A safe, fully static type system exists*
- *Lazy*
  - *Simplifies reasoning about staging*
- *Monads*
- *Very rich type system*

What needs to be done?

- *Convince Simon Peyton Jones :-)*
- *Need to identify research challenges*
  - *Gradual-typing based approach*
    - *Combine Template Haskell & MSP*
    - *Checking soundness w/ full typ. sys. Runtime code generation*



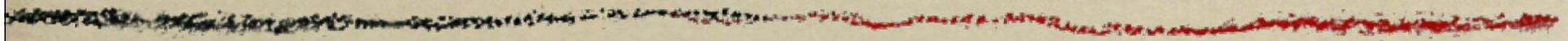
What staging types actually do

How staging types work

- *Types:* $s,t ::= \mathbf{int} \mid t*t \mid t+t \mid t \hat{} t \mid \langle t \rangle$
- *The curious facts*
  - $\mathbf{int} \not\sim \langle \mathbf{int} \rangle$, *but we have* $\langle \mathbf{int} \rangle \hat{} \mathbf{int}$
  - $\langle s * t \rangle \sim \langle s \rangle * \langle t \rangle$
  - $\langle s \hat{} t \rangle \sim \langle s \rangle \hat{} \langle t \rangle$
  - *and sometimes:* $\langle s+t \rangle \sim \langle s \rangle + \langle t \rangle$

What staging types really do

- *The code type flows to the leaves*
  - *This provides a normal form*
- *A lot like basic unit checking in physics*
- *Provide a great basis for programming with abstract interpretation*



Conclusion

Summary

- *Staging is useful because it helps us analyze the stageability of algorithms*
- *Interpreters are the "killer app"*
- *Haskell is the ideal staging language*
- *Staged types flow to leaves of types*
- *Stageability is like basic unit checking*

Challenges

- *Existential questions (re indexed types)*
  - *Is staging (MSP) really necessary*
  - *Is it enough?*
- *SML: Standard Macro Language*
  - *Extensible grammar, type system*
  - *Language independent*

Challenges

- *Type safe staging w/shared mutable state*
  - *SOA: Separability-based type system*
  - *Question: Do we REALLY need it?*
- *Type safe runtime code generation*
- *Computer-aided stageability analysis*
- *Compiler/architecture code-design*

Challenges

- *Mathematical equations, esp. hybrid differential equations. Example:*

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) = \frac{\partial L}{\partial x_i} \qquad (1 \leq i \leq N).$$

- *Preliminary results quite promising*

Concrete milestone challanges

- *MSP-based FFT that beats FFT(E/W)*
- *Matrix operations that beats BLAS*
- *MSP-based BED formal tools*
- *MSP-based NAS-parallel benchmarks*
- *MSP-based TCE-engine benchmarks*