METAPROGRAMMING IN THE REAL WORLD

DON CLUGSTON
My experience with Solar Photovoltaics
My experience with Solar Photovoltaics

- 1995: Spacecraft and Hippies
My experience with Solar Photovoltaics
- 1995: Spacecraft and Hippies
- 2007: NYSE, $Billion
My experience with Solar Photovoltaics

- 1995: Spacecraft and Hippies
- 2007: NYSE, $Billion
- 2011: Commodity market
My experience with Solar Photovoltaics
- 1995: Spacecraft and Hippies
- 2007: NYSE, $Billion
- 2011: Commodity market

Early adopters show where your guesses were wrong!
Founded 2009, Berlin by 3 PhDs
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
SOCIOMANTIC LABS GMBH

- Founded 2009, Berlin by 3 PhDs
- Real-time bidding for online advertising
- Technology based
  - Core technology is 100% D
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
- Core technology is 100% D

Expanding globally
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
- Core technology is 100% D

Expanding globally
- Serving 50+ markets on 6 continents
 Founded 2009, Berlin by 3 PhDs

 Real-time bidding for online advertising

 Technology based
 - Core technology is 100% D

 Expanding globally
 - Serving 50+ markets on 6 continents
 - Offices in 9 countries
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
- Core technology is 100% D

Expanding globally
- Serving 50+ markets on 6 continents
- Offices in 9 countries
- 100+ employees, 25+ nationalities
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
- Core technology is 100% D

Expanding globally
- Serving 50+ markets on 6 continents
- Offices in 9 countries
- 100+ employees, 25+ nationalities

Profitable
Founded 2009, Berlin by 3 PhDs

Real-time bidding for online advertising

Technology based
- Core technology is 100% D

Expanding globally
- Serving 50+ markets on 6 continents
- Offices in 9 countries
- 100+ employees, 25+ nationalities

Profitable
- Growth based entirely on revenue
User visits web page
User visits web page

While it loads, website auctions an ad space
User visits web page

While it loads, website auctions an ad space

We bid on behalf of our advertisers
User visits web page

While it loads, website auctions an ad space

We bid on behalf of our advertisers

Highest bidder gets to show their ad in the space
User visits web page

While it loads, website auctions an ad space

We bid on behalf of our advertisers

Highest bidder gets to show their ad in the space

Bids must be placed within 50 milliseconds
User visits web page

While it loads, website auctions an ad space

We bid on behalf of our advertisers

Highest bidder gets to show their ad in the space

Bids must be placed within 50 milliseconds
- Including internet latency
User visits web page

While it loads, website auctions an ad space

We bid on behalf of our advertisers

Highest bidder gets to show their ad in the space

Bids must be placed within 50 milliseconds
- Including internet latency

Billions of auctions per day
Must calculate how much this ad space is worth
Must calculate how much this ad space is worth

Bid accuracy improves with more data
- Must calculate how much this ad space is worth
- Bid accuracy improves with more data
  - Terabytes/day

Some form of NoSQL database mandatory
- Relational databases too slow + don't scale

Latency (rather than speed) is king

Two approaches
- 1 Off-the shelf product + workarounds
- 2 Custom, intrinsically fast solution
- Must calculate how much this ad space is worth

- Bid accuracy improves with more data
  - Terabytes/day

- Relational databases too slow + don’t scale
Must calculate how much this ad space is worth

Bid accuracy improves with more data
- Terabytes/day

Relational databases too slow + don’t scale

Everyone else uses an off-the-shelf NoSQL product
Must calculate how much this ad space is worth

Bid accuracy improves with more data
  - Terabytes/day

Relational databases too slow + don’t scale

Everyone else uses an off-the-shelf NoSQL product
  - and works around the speed bottlenecks
Must calculate how much this ad space is worth

Bid accuracy improves with more data
  - Terabytes/day

Relational databases too slow + don’t scale

Everyone else uses an off-the-shelf NoSQL product
  - and works around the speed bottlenecks

But we created an intrinsically fast solution, using D
- Must calculate how much this ad space is worth

- Bid accuracy improves with more data
  - Terabytes/day

- Relational databases too slow + don’t scale

- Everyone else uses an off-the-shelf NoSQL product
  - and works around the speed bottlenecks

- But we created an intrinsically fast solution, using D

- 50 milliseconds (minus net latency) to place a bid
- Must calculate how much this ad space is worth
- Bid accuracy improves with more data
  - Terabytes/day
- Relational databases too slow + don’t scale
- Everyone else uses an off-the-shelf NoSQL product
  - and works around the speed bottlenecks
- But we created an intrinsically fast solution, using D
- 50 milliseconds (minus net latency) to place a bid
  - Typical hard disk seek time is 9 ms
Must calculate how much this ad space is worth

Bid accuracy improves with more data
- Terabytes/day

Relational databases too slow + don’t scale

Everyone else uses an off-the-shelf NoSQL product
- and works around the speed bottlenecks

But we created an intrinsically fast solution, using D

50 milliseconds (minus net latency) to place a bid
- Typical hard disk seek time is 9 ms
- For most bids we achieve $\leq 2$ ms
OUR TECHNOLOGY STACK

- Tango-based runtime (modified), own libraries
OUR TECHNOLOGY STACK

- Tango-based runtime (modified), own libraries
  - Avoid ALL heap activity
OUR TECHNOLOGY STACK

- Tango-based runtime (modified), own libraries
  - Avoid ALL heap activity

- Fiber-based concurrency (not threads)
Tango-based runtime (modified), own libraries
- Avoid ALL heap activity

Fiber-based concurrency (not threads)

‘Swarm’ In-memory Distributed Hash Table
OUR TECHNOLOGY STACK

- Tango-based runtime (modified), own libraries
  - Avoid ALL heap activity
- Fiber-based concurrency (not threads)
- ‘Swarm’ In-memory Distributed Hash Table
- Data stored in D format, no conversion
Tango-based runtime (modified), own libraries
- Avoid ALL heap activity

Fiber-based concurrency (not threads)

‘Swarm’ In-memory Distributed Hash Table

Data stored in D format, no conversion

All processes stream-based and completely scalable
Direct binding to C libraries
- Direct binding to C libraries
- Array slices
- Direct binding to C libraries

- Array slices
  - Avoid heap activity, but stay correct
Direct binding to C libraries

Array slices
- Avoid heap activity, but stay correct

Painless compile-time programming
Direct binding to C libraries

Array slices
- Avoid heap activity, but stay correct

Painless compile-time programming
- eg, for serialization
Features to drop from C++

- C source code compatibility
- Link compatibility with C++
- Multiple inheritance
- Preprocessor
- Templates

Templates!

"If a language can capture 90% of the power of C++ with 10% of its complexity, I argue that is a worthwhile tradeoff." - DMD FAQ
Templates!

static if, static assert

"If a language can capture 90% of the power of C++ with 10% of its complexity, I argue that is a worthwhile tradeoff." - DMD FAQ
Templates!

static if, static assert

Some reflection -- is() expressions
Templates!

static if, static assert

Some reflection -- is() expressions

Still defensive w.r.t C++
Templates!

static if, static assert

Some reflection -- is() expressions

Still defensive w.r.t C++
- “If a language can capture 90% of the power of C++ with 10% of its complexity, I argue that is a worthwhile tradeoff.” – DMD FAQ
Improved constant folding
D METAPROGRAMMING IN 2007

- Improved constant folding
- Compile Time Function Execution (CTFE)
- Improved constant folding
- Compile Time Function Execution (CTFE)
- string mixins
D METAPROGRAMMING IN 2007

- Improved constant folding
- Compile Time Function Execution (CTFE)
- string mixins
- stringof
Template constraints
D METAPROGRAMMING IN 2013

- Template constraints
- __traits (just as ugly as is() expressions!)
Template constraints

__traits (just as ugly as is() expressions!)

alias this
Template constraints

__traits (just as ugly as is() expressions!)

alias this

opDispatch
Template constraints

__traits (just as ugly as is() expressions!)

alias this

opDispatch

Dramatic implementation improvements
We got here by incremental improvements
We got here by incremental improvements

Programmers follow the same learning curve
We got here by incremental improvements

Programmers follow the same learning curve

Metaprogramming is an unexpected strength of D
We got here by incremental improvements

Programmers follow the same learning curve

Metaprogramming is an unexpected strength of D

We still have some detritus
Return on Investment (ROI) formula:

\[(\text{Benefit} - \text{Cost}) / \text{Cost}\]
At what time does this become positive?

- \( \frac{(\text{Benefit} - \text{Cost})}{\text{Cost}} \)

- At what time does this become positive?
(Benefit - Cost) / Cost

At what time does this become positive?

The time until you obtain benefit can be as important as the cost!
RETURN ON INVESTMENT (ROI)

- \( \frac{\text{Benefit} - \text{Cost}}{\text{Cost}} \)
- At what time does this become positive?
- The time until you obtain benefit can be as important as the cost!
- Benefit > Cost at t = infinity is not enough!
At what time does this become positive?

The time until you obtain benefit can be as important as the cost!

Benefit > Cost at t = infinity is not enough!

Who gets the benefit?
Language changes must NEVER break code
Language changes must NEVER break code

Except in extreme cases
Breaking code is an up-front cost
Breaking code is an up-front cost

But keeping mis-features is worse!
Breaking code is an up-front cost

But keeping mis-features is worse!
- an on-going cost
Breaking code is an up-front cost

But keeping mis-features is worse!
- an on-going cost

Gratuitous name changes have very poor ROI
Breaking code is an up-front cost

But keeping mis-features is worse!
  - an on-going cost

Gratuitous name changes have very poor ROI

If the benefit is instant, any cost is OK
Breaking code is an up-front cost

But keeping mis-features is worse!
- an on-going cost

Gratuitous name changes have very poor ROI

If the benefit is instant, any cost is OK
- eg if it catches a bug
Breaking code is an up-front cost

But keeping mis-features is worse!
  - an on-going cost

Gratuitous name changes have very poor ROI

If the benefit is instant, any cost is OK
  - eg if it catches a bug

Breaking changes can be met with enthusiasm!
Expectation
Expectation
- Easier than in C++
Expectation

- Easier than in C++
- But still only used by wizards, in libraries
Expectation

- Easier than in C++
- But still only used by wizards, in libraries

Experience

- Used even in application code!
- Used even by new D programmers!
- Entry level is very low
- ‘static if’ is instantly understood
- ROI is excellent
- Improves programmer morale
Expectation
- Easier than in C++
- But still only used by wizards, in libraries

Experience
- Used even in application code!
METAPROGRAMMING

**Expectation**
- Easier than in C++
- But still only used by wizards, in libraries

**Experience**
- Used even in application code!
- Used even by new D programmers!
**Expectation**

- Easier than in C++
- But still only used by wizards, in libraries

**Experience**

- Used even in application code!
- Used even by new D programmers!
- Entry level is very low
**Expectation**
- Easier than in C++
- But still only used by wizards, in libraries

**Experience**
- Used even in in application code!
- Used even by new D programmers!
- Entry level is very low
  - ‘static if’ is instantly understood
**Expectation**
- Easier than in C++
- But still only used by wizards, in libraries

**Experience**
- Used even in application code!
- Used even by new D programmers!
- Entry level is very low
  - `static if` is instantly understood
  - ROI is excellent
**Expectation**
- Easier than in C++
- But still only used by wizards, in libraries

**Experience**
- Used even in application code!
- Used even by new D programmers!
- Entry level is very low
  - ‘static if’ is instantly understood
  - ROI is excellent

**Improves programmer morale**
**ERROR MESSAGES**

- **Expectation**
Expectation

- Lowest importance of any type of compiler bug
Expectation
- Lowest importance of any type of compiler bug

Experience
**Expectation**
- Lowest importance of any type of compiler bug

**Experience**
- Make advanced features seem simpler
Expectation
- Lowest importance of any type of compiler bug

Experience
- Make advanced features seem simpler
- Have a pedagogic role
**Expectation**

- Lowest importance of any type of compiler bug

**Experience**

- Make advanced features seem simpler
- Have a pedagogic role
- Good error messages save time.. and time is money
ERROR MESSAGES

**Expectation**
- Lowest importance of any type of compiler bug

**Experience**
- Make advanced features seem simpler
- Have a pedagogic role
- Good error messages save time.. and time is money
- Error messages are the reason we use statically-typed languages!
Expectation
Expectation

Huge win! Used everywhere
Expectation

Huge win! Used everywhere

Subliminal metaprogramming!
Expectation

- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
Expectation

Huge win! Used everywhere

Subliminal metaprogramming!

Increased power will increase adoption
  - Pointers, throw exceptions, ...
COMPILE TIME FUNCTION EXECUTION

- Expectation
- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
  - Pointers, throw exceptions, ...
- Experience
Expectation

- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
  - Pointers, throw exceptions, ...

Experience

- CTFE hardly gets used, because it’s too slow
Expectation

- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
  - Pointers, throw exceptions, ...

Experience

- CTFE hardly gets used, because it’s too slow
  - Fast compilation is addictive!
Expectation
- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
  - Pointers, throw exceptions, ...

Experience
- CTFE hardly gets used, because it’s too slow
  - Fast compilation is addictive!

- Why isn’t it fast yet?
**Expectation**

- Huge win! Used everywhere
- Subliminal metaprogramming!
- Increased power will increase adoption
  - Pointers, throw exceptions, ...

**Experience**

- CTFE hardly gets used, because it’s too slow
  - Fast compilation is addictive!

**Why isn’t it fast yet?**
- Because of the history
Expectation

• Huge win! Used everywhere
• Subliminal metaprogramming!

• Increased power will increase adoption
  - Pointers, throw exceptions, ...

Experience

• CTFE hardly gets used, because it’s too slow
  - Fast compilation is addictive!

Why isn’t it fast yet?
  - Because of the history
  - Many unintended dependencies
Expectation

Huge win! Used everywhere

Subliminal metaprogramming!

Increased power will increase adoption
- Pointers, throw exceptions, ...

Experience

CTFE hardly gets used, because it’s too slow
- Fast compilation is addictive!

Why isn’t it fast yet?
- Because of the history
- Many unintended dependencies
- Front-end must be in a valid state!
Expectation
Expectation
- Tutorials are almost irrelevant
Expectation
- Tutorials are almost irrelevant

Experience
Expectation
- Tutorials are almost irrelevant

Experience
- Absence of tutorials is an embarrassment
Much smaller problem than expected
Much smaller problem than expected

Template bugs rarely encountered in D1
Much smaller problem than expected

Template bugs rarely encountered in D1

64 bit code generation a nightmare
Much smaller problem than expected

Template bugs rarely encountered in D1

64 bit code generation a nightmare
  - But mostly a one-off cost borne by us
- Much smaller problem than expected
- Template bugs rarely encountered in D1
- 64 bit code generation a nightmare
  - But mostly a one-off cost borne by us
- Otherwise, IDE bugs much worse
D is moving out of research mode
D is moving out of research mode
- We can no longer ignore implementation issues
D is moving out of research mode
  - We can no longer ignore implementation issues

A Return-On-Investment model is useful
D is moving out of research mode
- We can no longer ignore implementation issues

A Return-On-Investment model is useful
- D must deliver value in the near-term
D is moving out of research mode
- We can no longer ignore implementation issues

A Return-On-Investment model is useful
- D must deliver value in the near-term

Metaprogramming is a strength of D in the real world
D is moving out of research mode
- We can no longer ignore implementation issues

A Return-On-Investment model is useful
- D must deliver value in the near-term

Metaprogramming is a strength of D in the real world
- D does deliver ROI for Sociomantic Labs
D is moving out of research mode
  - We can no longer ignore implementation issues

A Return-On-Investment model is useful
  - D must deliver value in the near-term

Metaprogramming is a strength of D in the real world
  - D does deliver ROI for Sociomantic Labs
  - But not yet in all areas
WE’RE HIRING!

www.sociomantic.com