

Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

2016 © Microsoft Corporation. All rights reserved.

```
#if ABSOLUTE  
    position = new;  
#elif RELATIVE  
    position += new;  
#endif
```

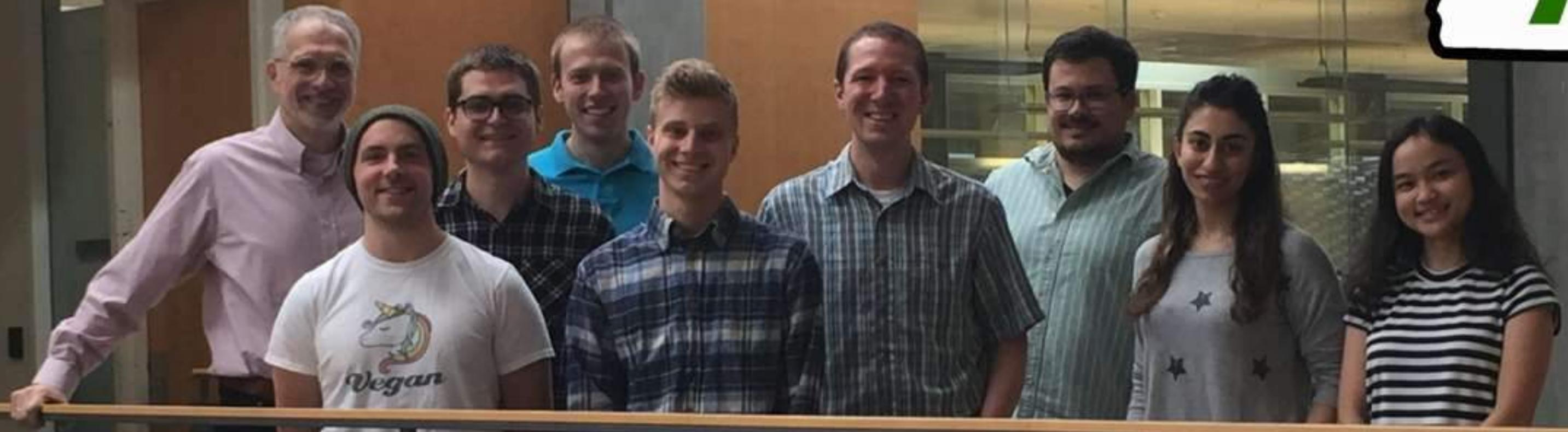


Why don't we have both?

Applications of variational programming

Eric Walkingshaw
Oregon State University

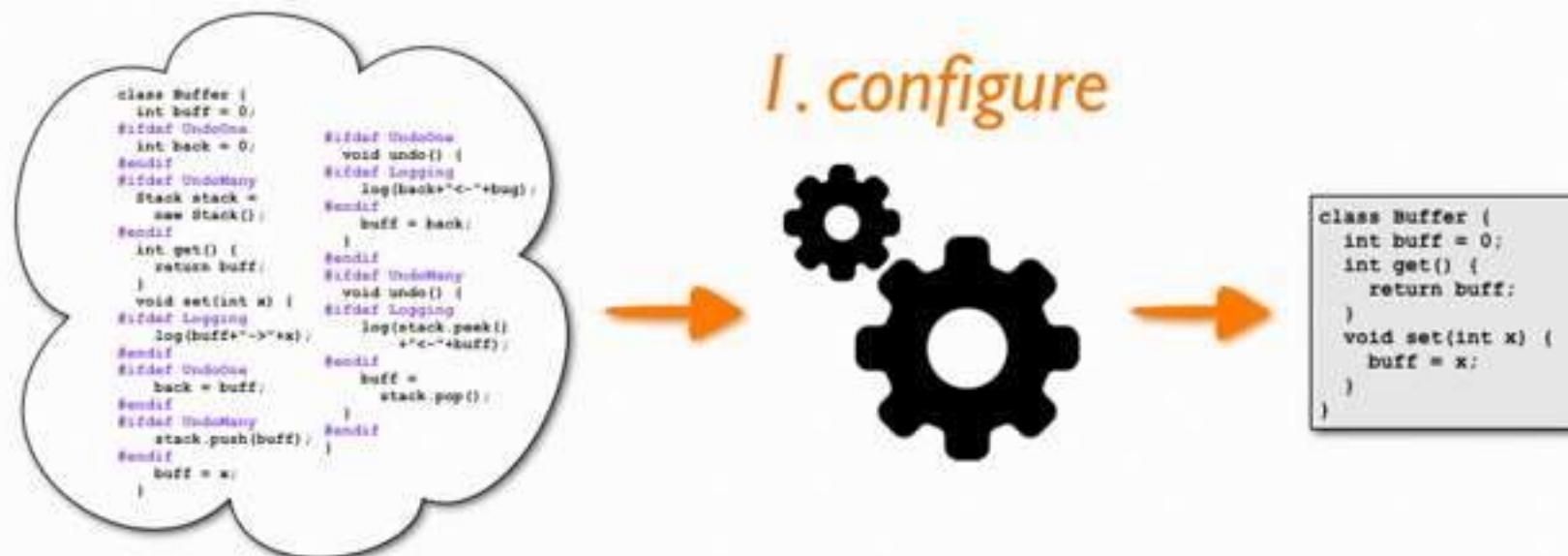
PL Group @ OSU



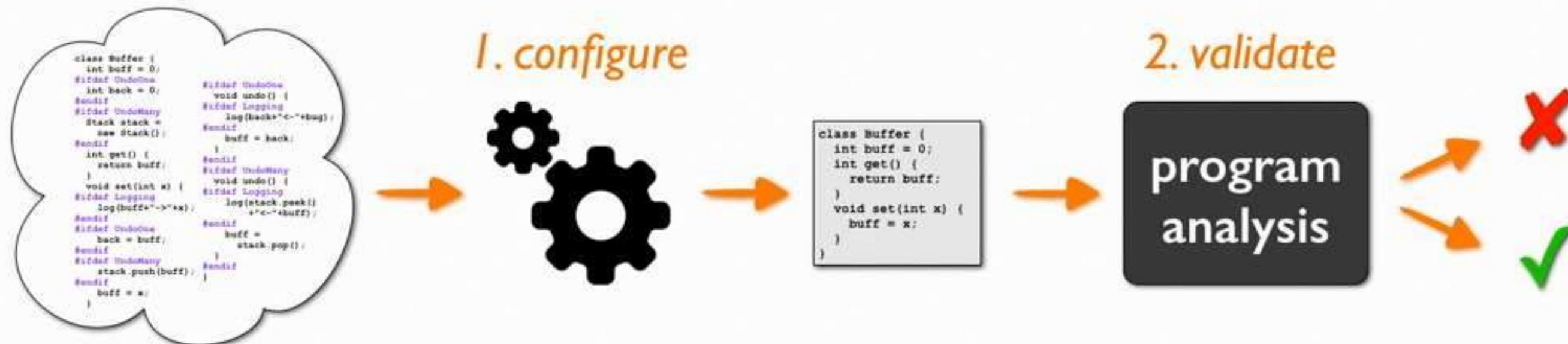
Original motivation: highly configurable software systems



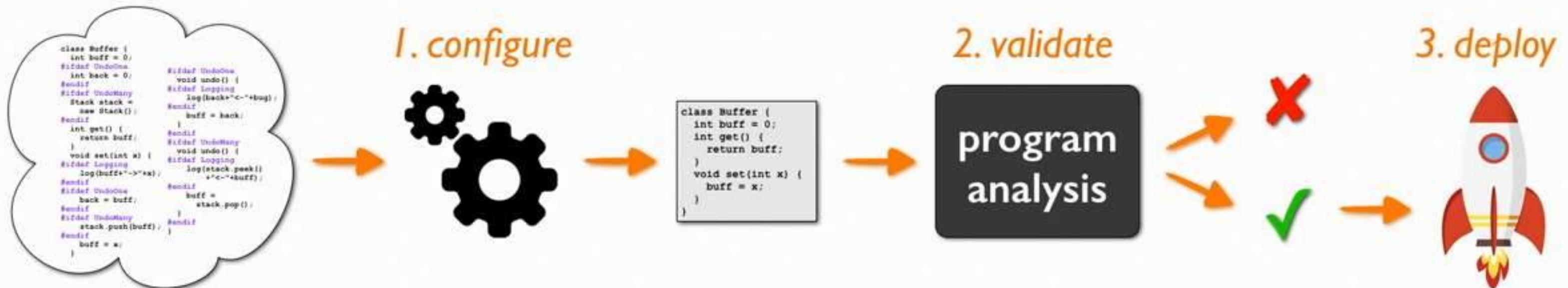
Original motivation: highly configurable software systems



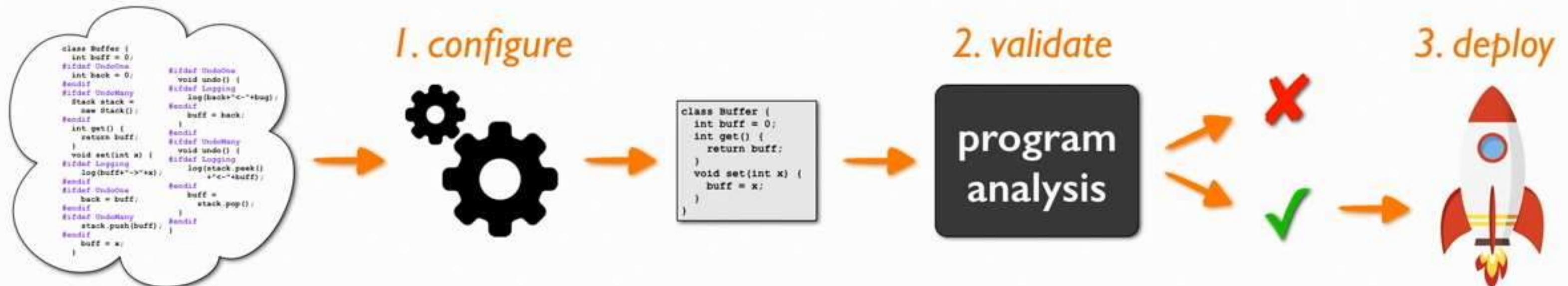
Original motivation: highly configurable software systems



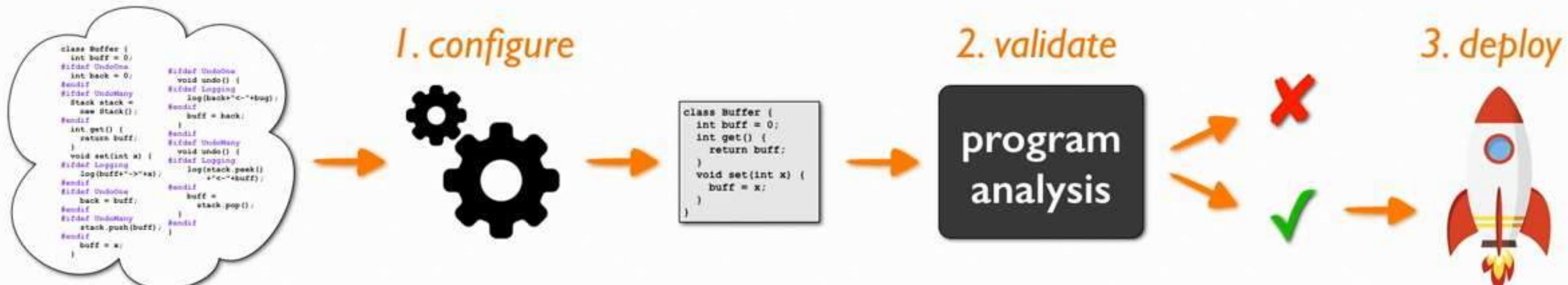
Original motivation: highly configurable software systems



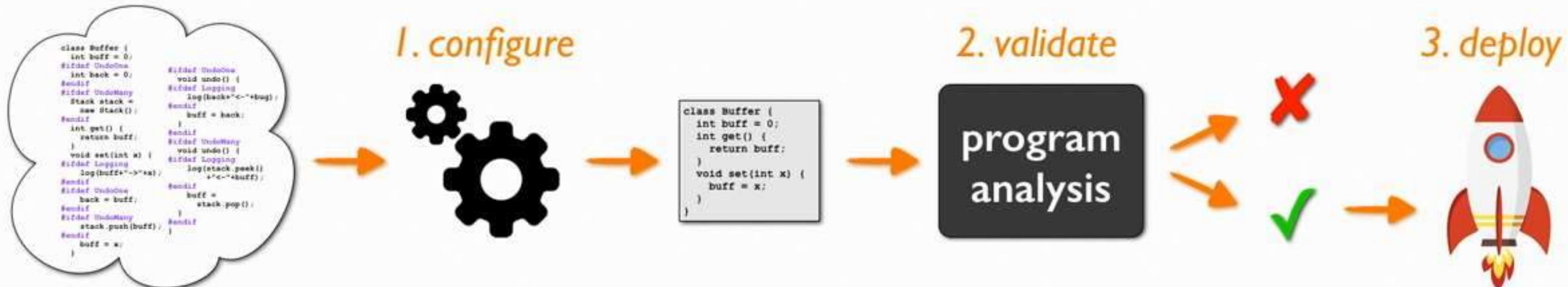
Original motivation: highly configurable software systems



Original motivation: highly configurable software systems



Original motivation: highly configurable software systems

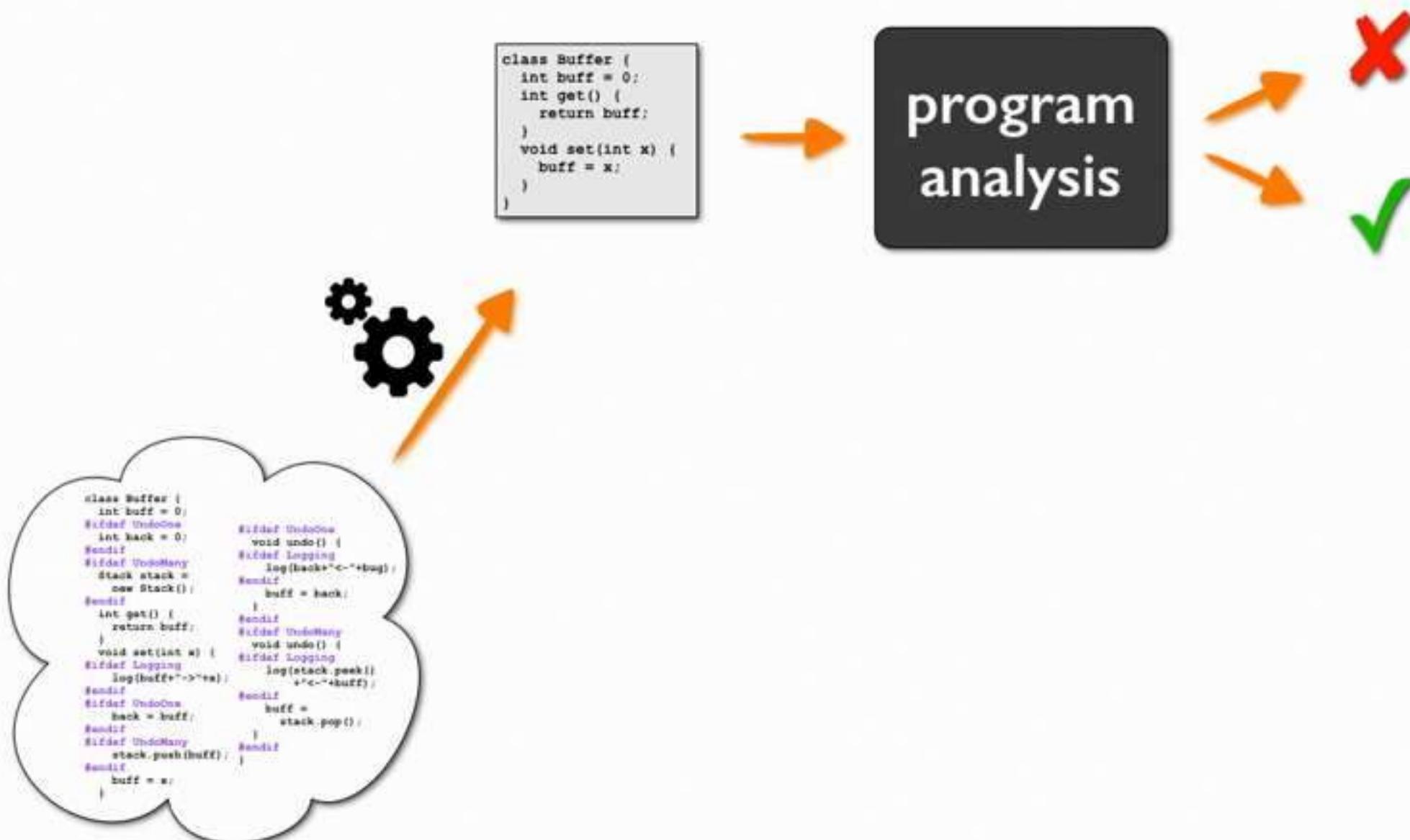


This process finds errors too late!

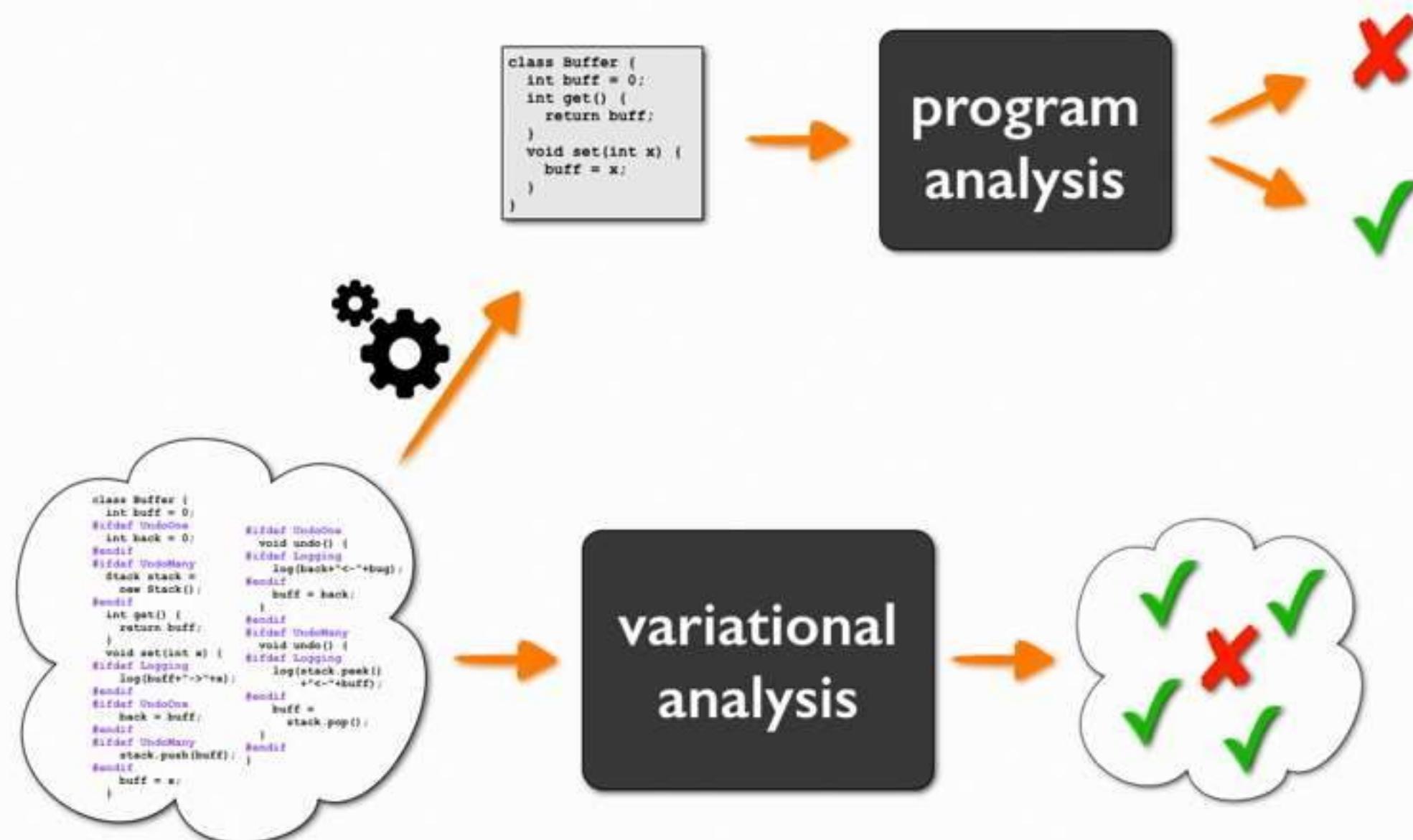
... but way too many configs to check them all



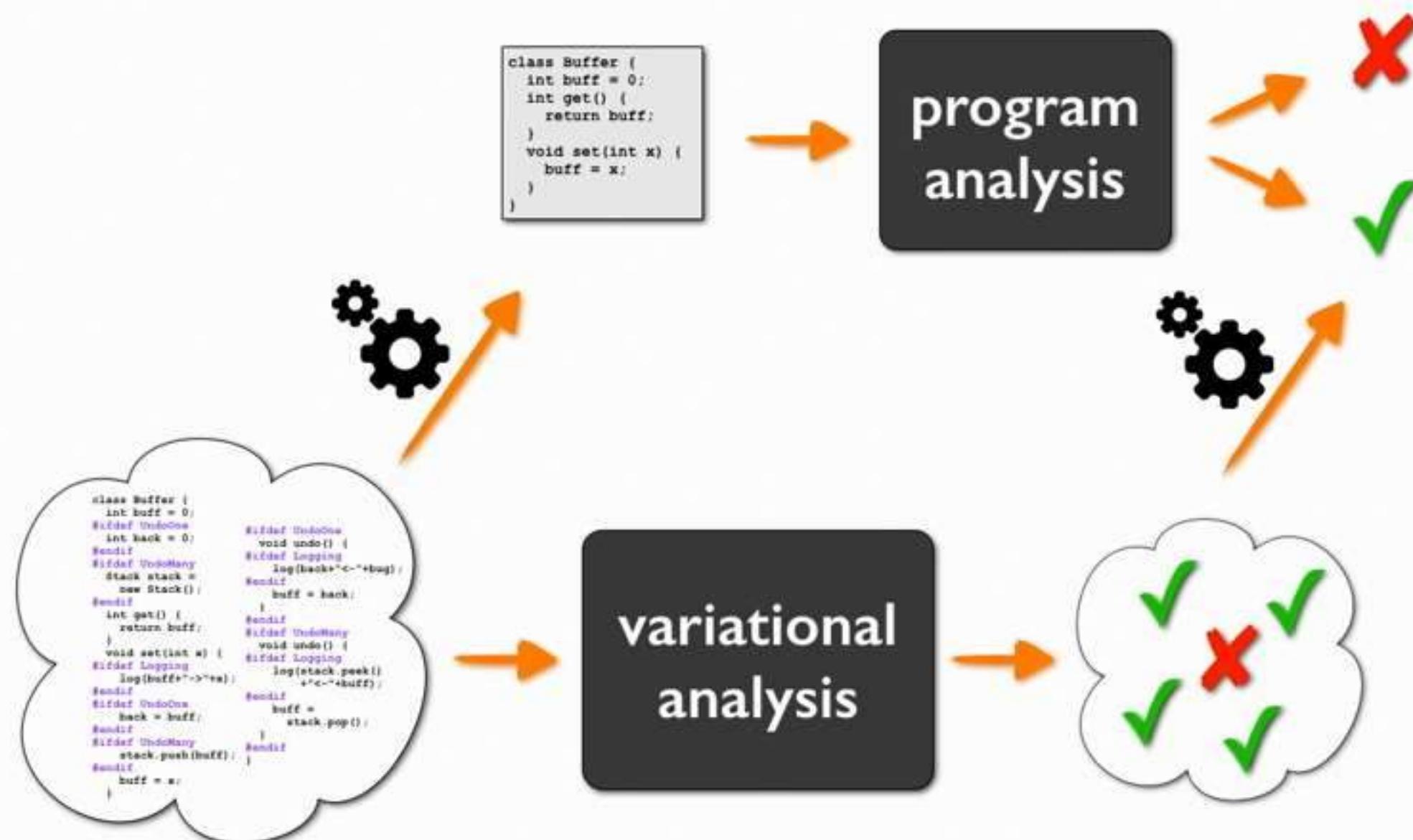
Solution: variational analyses



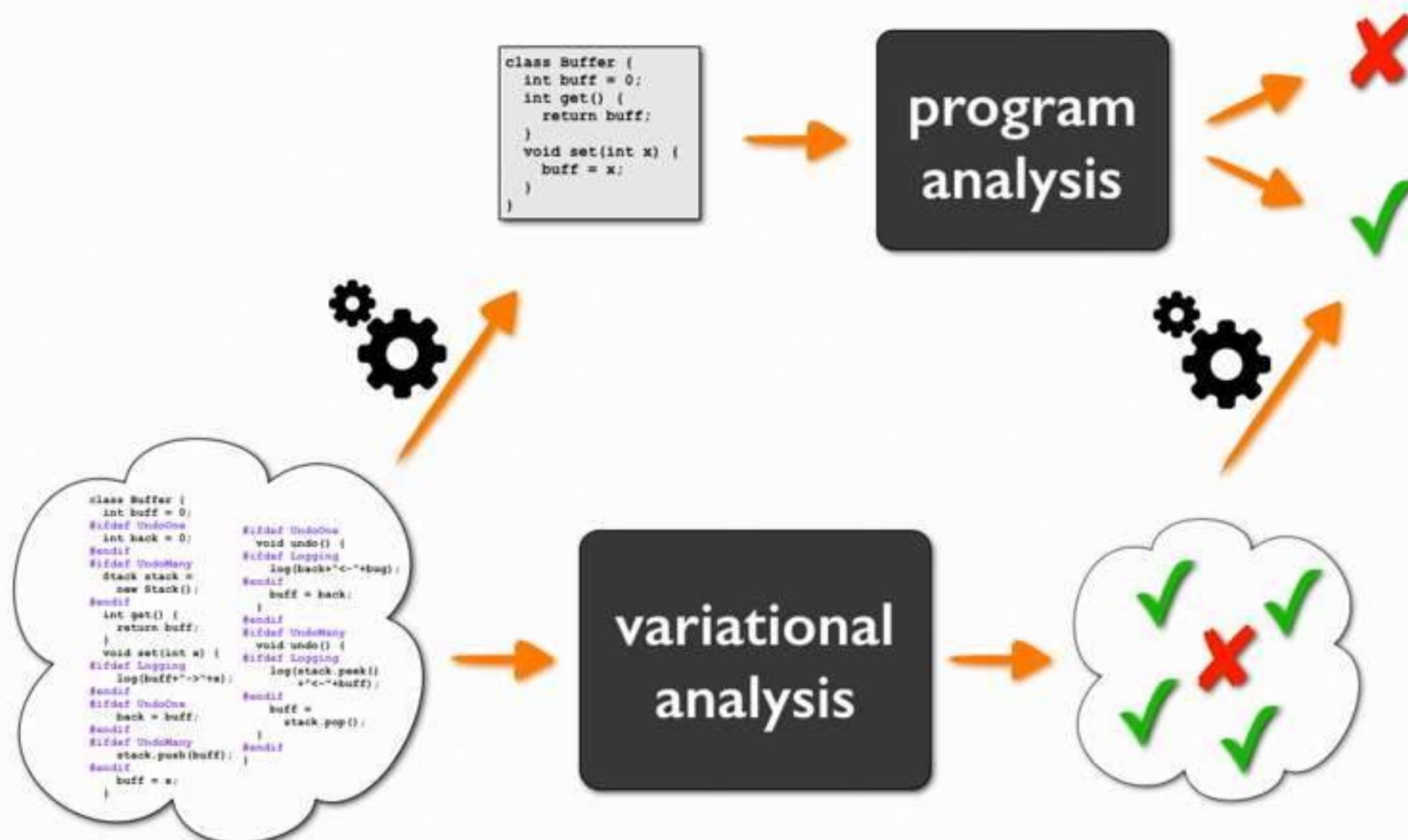
Solution: variational analyses



Solution: variational analyses



Solution: variational analyses

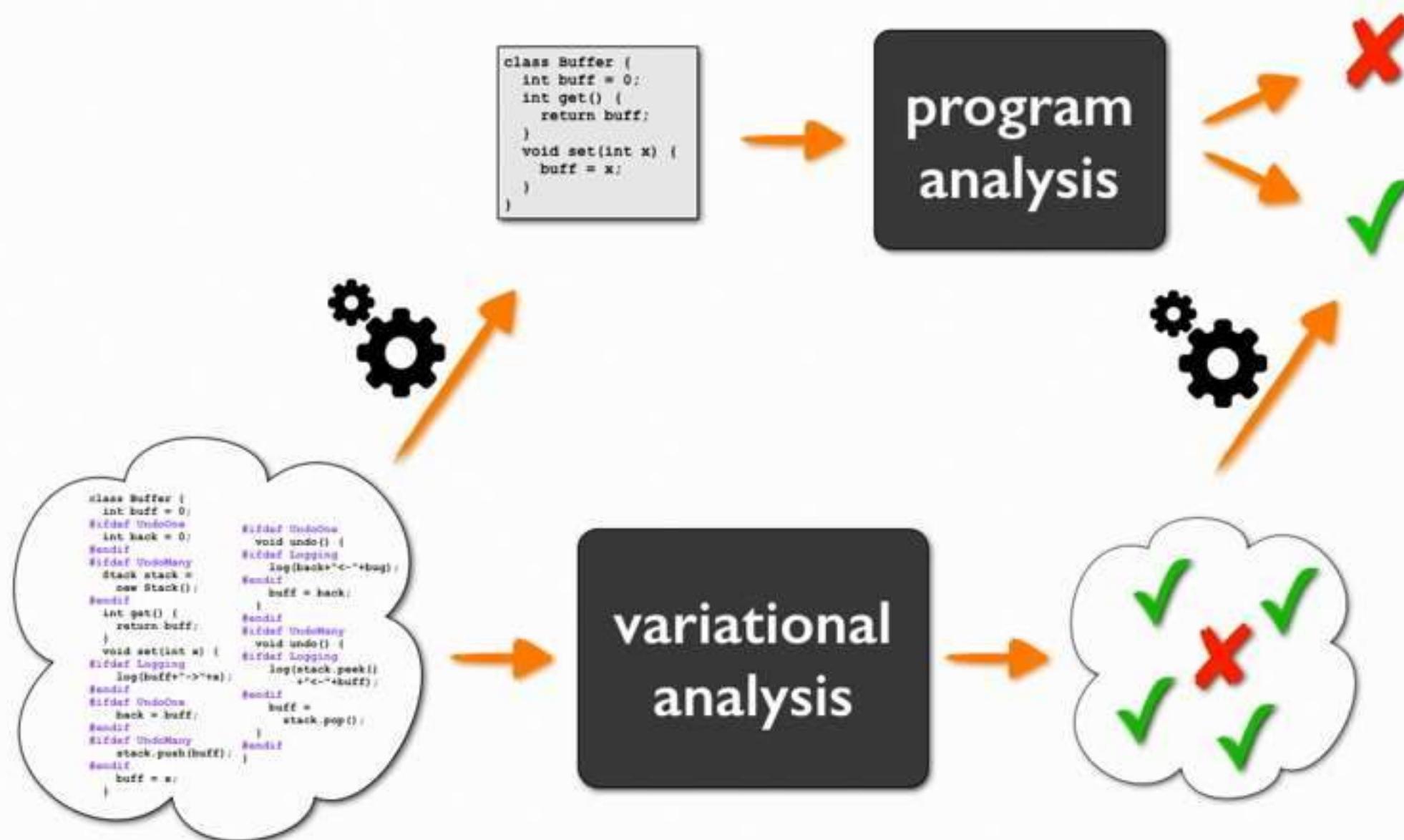


Our work:

- languages
- theory
- data structures
- algorithms

to do this *correctly* and *efficiently*

Solution: variational analyses



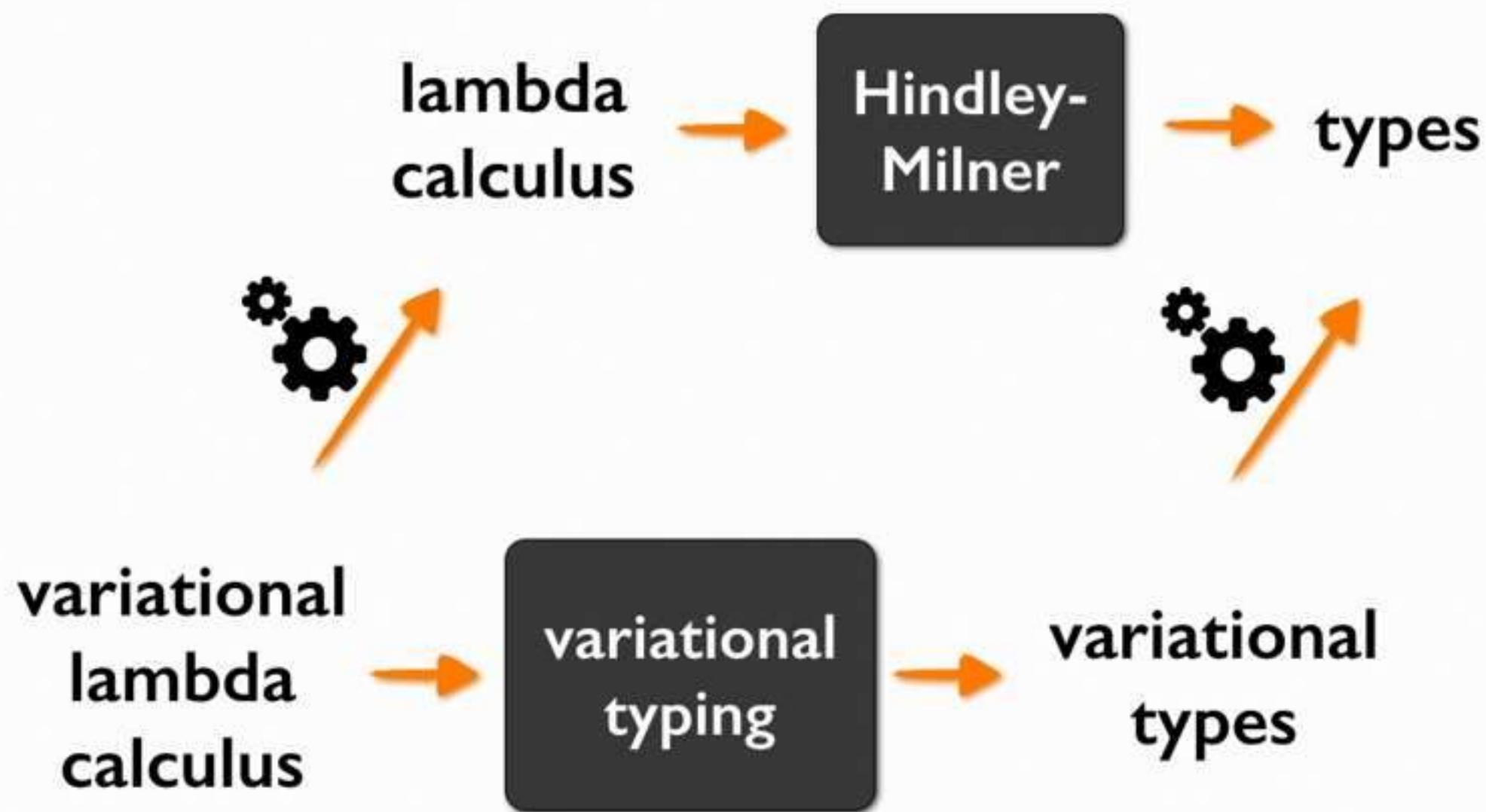
Our work:

- languages
- theory
- data structures
- algorithms

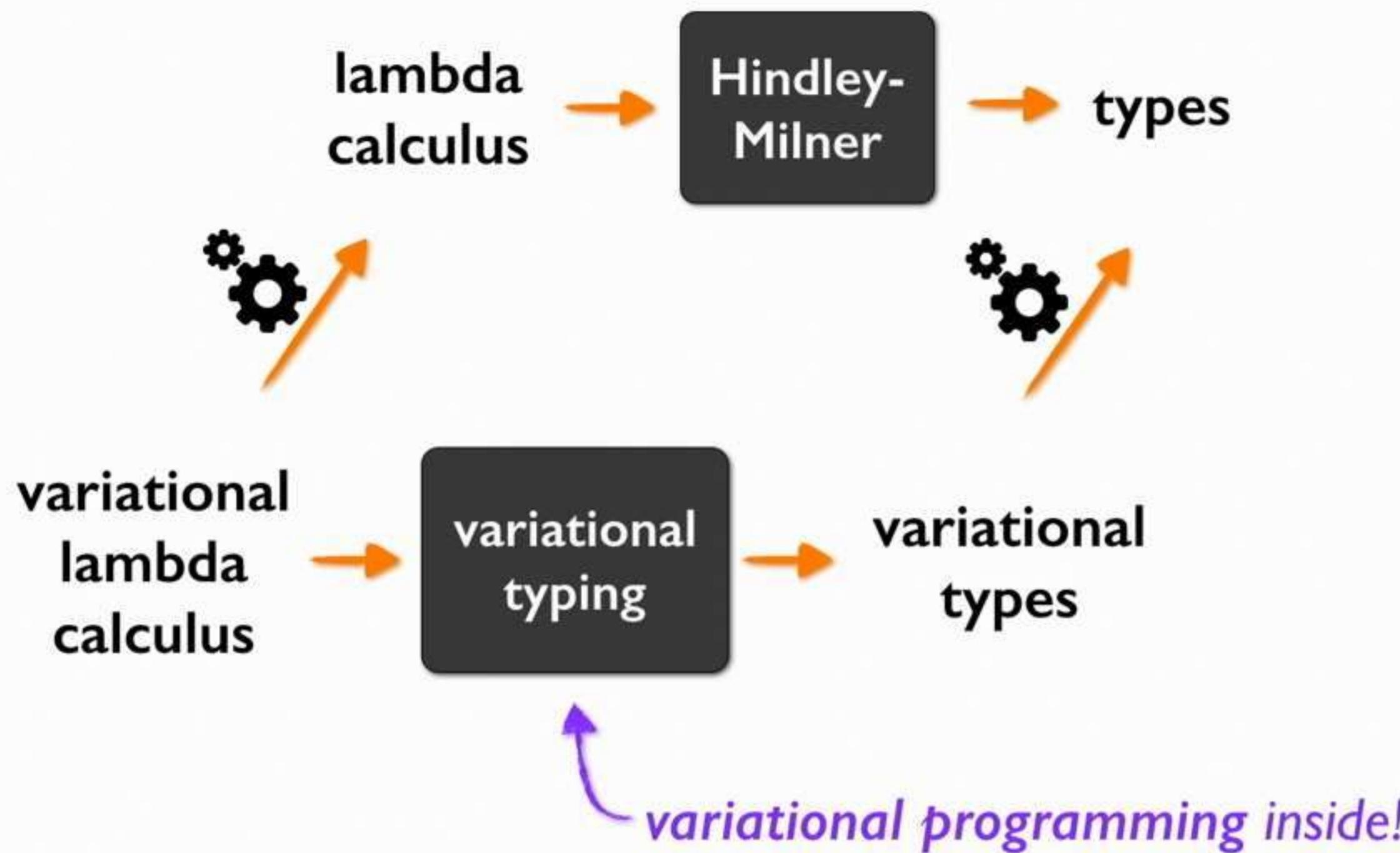
to do this *correctly* and *efficiently*

“variational programming”

Example: variational typing



Example: variational typing



Variational programming by example

Variational programming by example

A{2,3} + 4

Variational programming by example

A{2,3} + 4

↳ A{6,7}

Variational programming by example

$A\langle 2,3 \rangle + 4$

$\mapsto A\langle 6,7 \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$

Variational programming by example

$A\langle 2,3 \rangle + 4$

$\mapsto A\langle 6,7 \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$

$\mapsto A\langle 12,23 \rangle$

Variational programming by example

$A\langle 2,3 \rangle + 4$
 $\mapsto A\langle 6,7 \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$
 $\mapsto A\langle 12,23 \rangle$

$A\langle 2,3 \rangle + B\langle 10,20 \rangle$

Variational programming by example

$A\langle 2,3 \rangle + 4$
 $\mapsto A\langle 6,7 \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$
 $\mapsto A\langle 12,23 \rangle$

$A\langle 2,3 \rangle + B\langle 10,20 \rangle$
 $\mapsto A\langle B\langle 12,22 \rangle, B\langle 13,23 \rangle \rangle$

Variational programming by example

$A(2,3) + 4$
 $\mapsto A(6,7)$

$A(\text{True}, 3)$

$A(2,3) + A(10,20)$
 $\mapsto A(12,23)$

$A(2,3) + B(10,20)$
 $\mapsto A(B(12,22), B(13,23))$

Variational programming by example

$A\langle 2,3 \rangle + 4$
 $\mapsto A\langle 6,7 \rangle$

A(True,3)
: A(Bool,Int)

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$
 $\mapsto A\langle 12,23 \rangle$

$A\langle 2,3 \rangle + B\langle 10,20 \rangle$
 $\mapsto A\langle B\langle 12,22 \rangle, B\langle 13,23 \rangle \rangle$

Variational programming by example

$A\langle 2,3 \rangle + 4$
 $\mapsto A\langle 6,7 \rangle$

$A\langle \text{True}, 3 \rangle$
 $: A\langle \text{Bool}, \text{Int} \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$
 $\mapsto A\langle 12,23 \rangle$

$A\langle \text{succ}, \text{even} \rangle$

$A\langle 2,3 \rangle + B\langle 10,20 \rangle$
 $\mapsto A\langle B\langle 12,22 \rangle, B\langle 13,23 \rangle \rangle$

Variational programming by example

$A(2,3) + 4$
 $\mapsto A(6,7)$

$A(\text{True}, 3)$
 $: A(\text{Bool}, \text{Int})$

$A(2,3) + A(10,20)$
 $\mapsto A(12,23)$

$A(\text{succ}, \text{even})$
 $: \text{Int} \rightarrow A(\text{Int}, \text{Bool})$

$A(2,3) + B(10,20)$
 $\mapsto A(B(12,22), B(13,23))$

Variational programming by example

$A\langle 2,3 \rangle + 4$
 $\mapsto A\langle 6,7 \rangle$

$A\langle \text{True}, 3 \rangle$
 $: A\langle \text{Bool}, \text{Int} \rangle$

$A\langle 2,3 \rangle + A\langle 10,20 \rangle$
 $\mapsto A\langle 12,23 \rangle$

$A\langle \text{succ}, \text{even} \rangle$
 $: \text{Int} \rightarrow A\langle \text{Int}, \text{Bool} \rangle$
 $\equiv A\langle \text{Int} \rightarrow \text{Int}, \text{Int} \rightarrow \text{Bool} \rangle$

$A\langle 2,3 \rangle + B\langle 10,20 \rangle$
 $\mapsto A\langle B\langle 12,22 \rangle, B\langle 13,23 \rangle \rangle$

Variational programming by example

A(not,even) A(True,3)

Variational programming by example

`A(not,even) A(True,3)`

↳ **False**

Variational programming by example

`A(not,even) A(True,3)`

↳ `False`

≡ `A(False,False)`

Variational programming by example

$A(\text{not}, \text{even}) \ A(\text{True}, 3)$

$\mapsto \text{False}$

$\equiv A(\text{False}, \text{False})$

$A(\text{succ}, \text{even}) \ B(2, 4)$

Variational programming by example

$A(\text{not}, \text{even}) \ A(\text{True}, 3)$

$\mapsto \text{False}$

$\equiv A(\text{False}, \text{False})$

$A(\text{succ}, \text{even}) \ B(2, 4)$

$\mapsto A(B(3, 5), \text{True})$

Variational programming by example

```
A(not,even) A(True,3)
```

```
↪ False
```

```
≡ A(False,False)
```

```
A(succ,even) B(2,4)
```

```
↪ A(B(3,5),True)
```

```
vsum (A(2,3) + B(10,20))
```

Variational programming by example

```
A(not,even) A(True,3)
```

```
↪ False
```

```
≡ A(False,False)
```

```
A(succ,even) B(2,4)
```

```
↪ A(B(3,5),True)
```

```
vsum (A(2,3) + B(10,20))
```

```
↪ 70
```

Variational programming by example

`A(not,even) A(True,3)`

$\mapsto \text{False}$

$\equiv \text{A}(\text{False}, \text{False})$

`A(succ,even) B(2,4)`

$\mapsto \text{A}(\text{B}(3,5), \text{True})$

`vsum (A(2,3) + B(10,20))`

$\mapsto 70$

`vmax (A(2,3) + B(10,20))`

Variational programming by example

```
A(not,even) A(True,3)  
↳ False  
≡ A(False,False)
```

```
vsum (A(2,3) + B(10,20))  
↳ 70
```

```
A(succ,even) B(2,4)  
↳ A(B(3,5),True)
```

```
vmax (A(2,3) + B(10,20))  
↳ 23 @ [A.R,B.R]
```

Challenge #1

exponential variation space

Challenge #1

exponential variation space

Share as much as possible

- split late and join early
- clever data structures

Challenge #1

exponential variation space

Share as much as possible

- split late and join early
- clever data structures

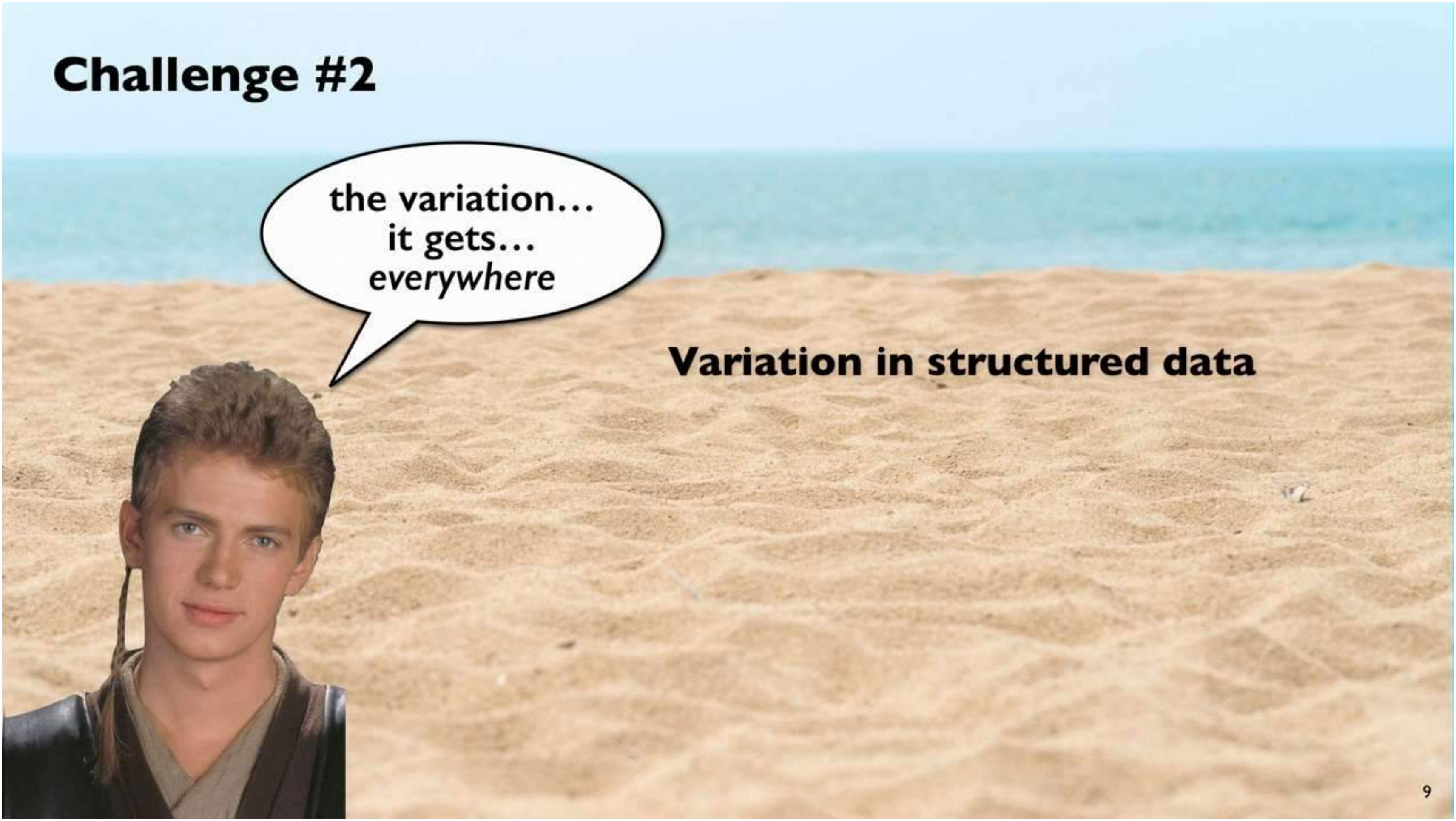
Pick the right domains

Challenge #2



the variation...
it gets...
everywhere

Challenge #2



the variation...
it gets...
everywhere

Variation in structured data

Challenge #2

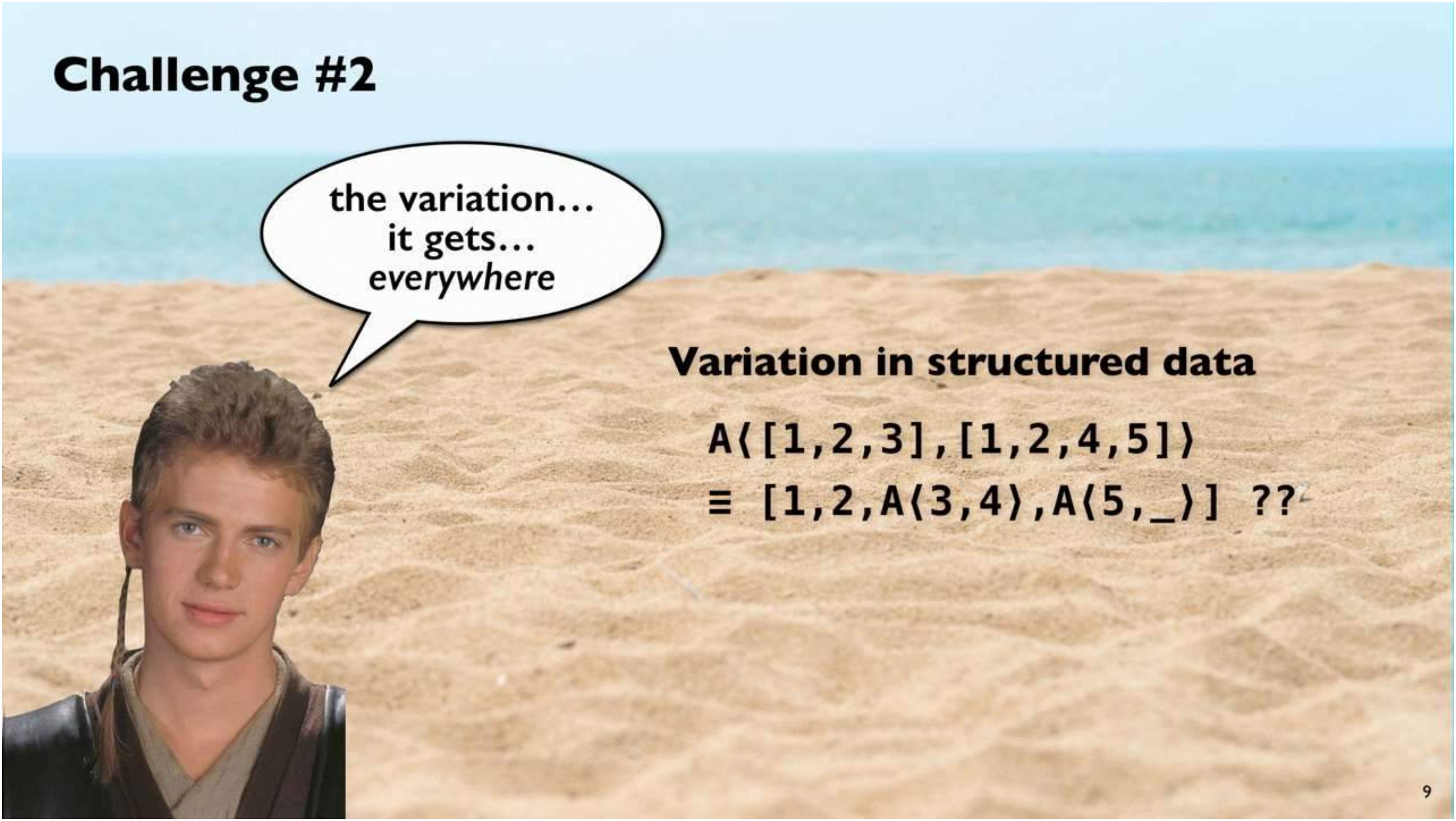


the variation...
it gets...
everywhere

Variation in structured data

`A([1,2,3],[1,2,4,5])`

Challenge #2

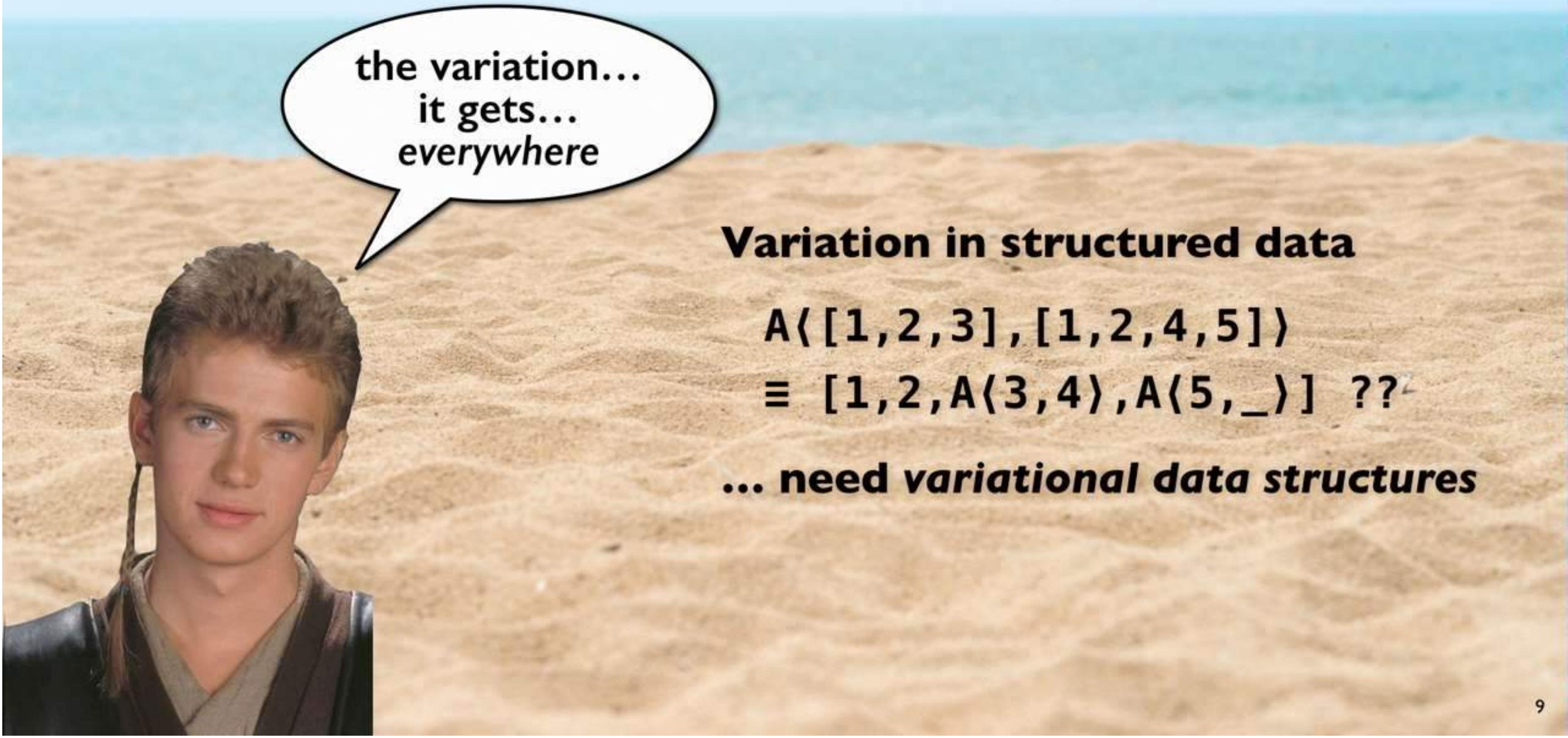


the variation...
it gets...
everywhere

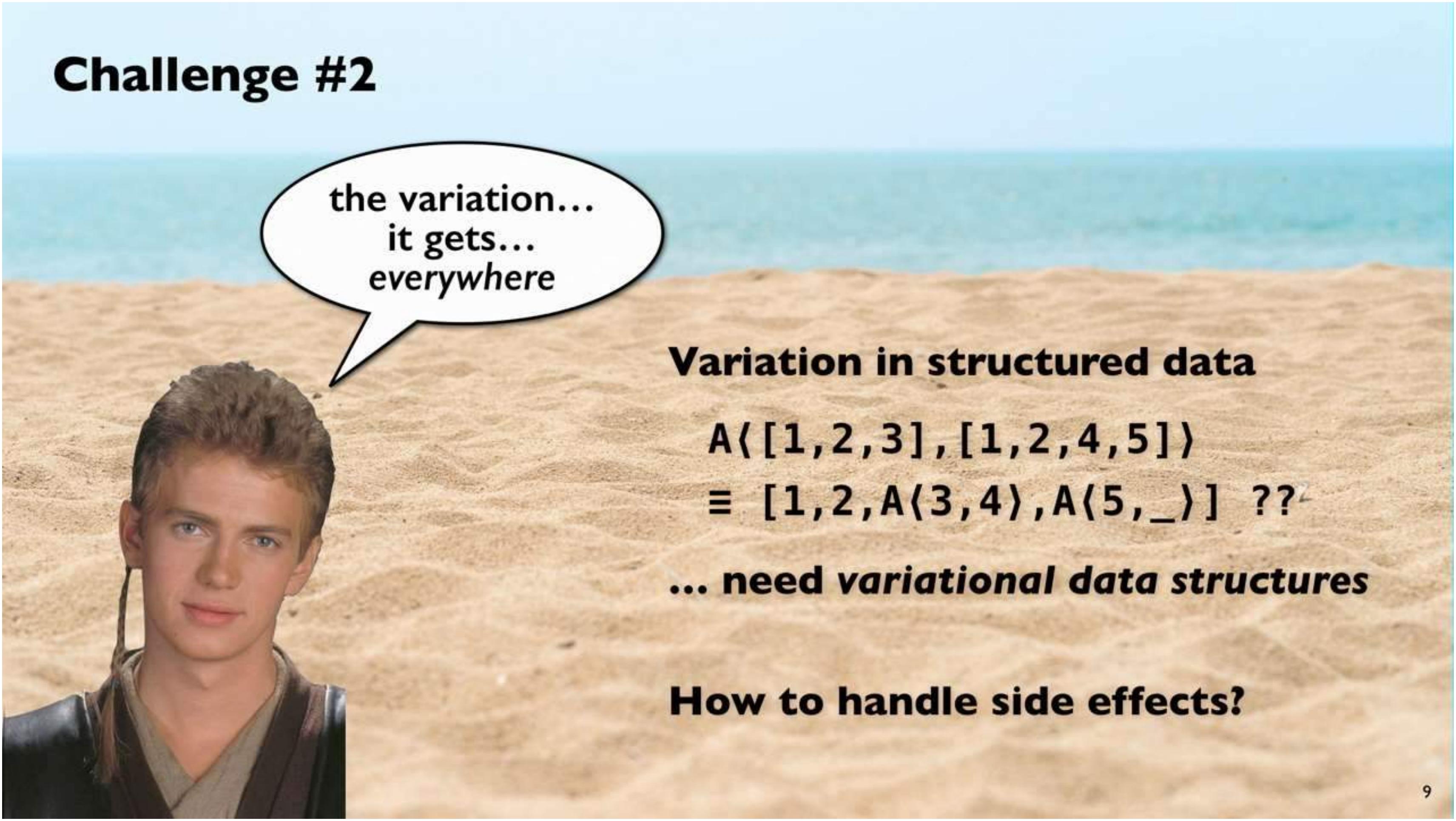
Variation in structured data

$A([1,2,3],[1,2,4,5])$
 $\equiv [1,2,A(3,4),A(5,_)] \text{ ??}$

Challenge #2



Challenge #2



the variation...
it gets...
everywhere

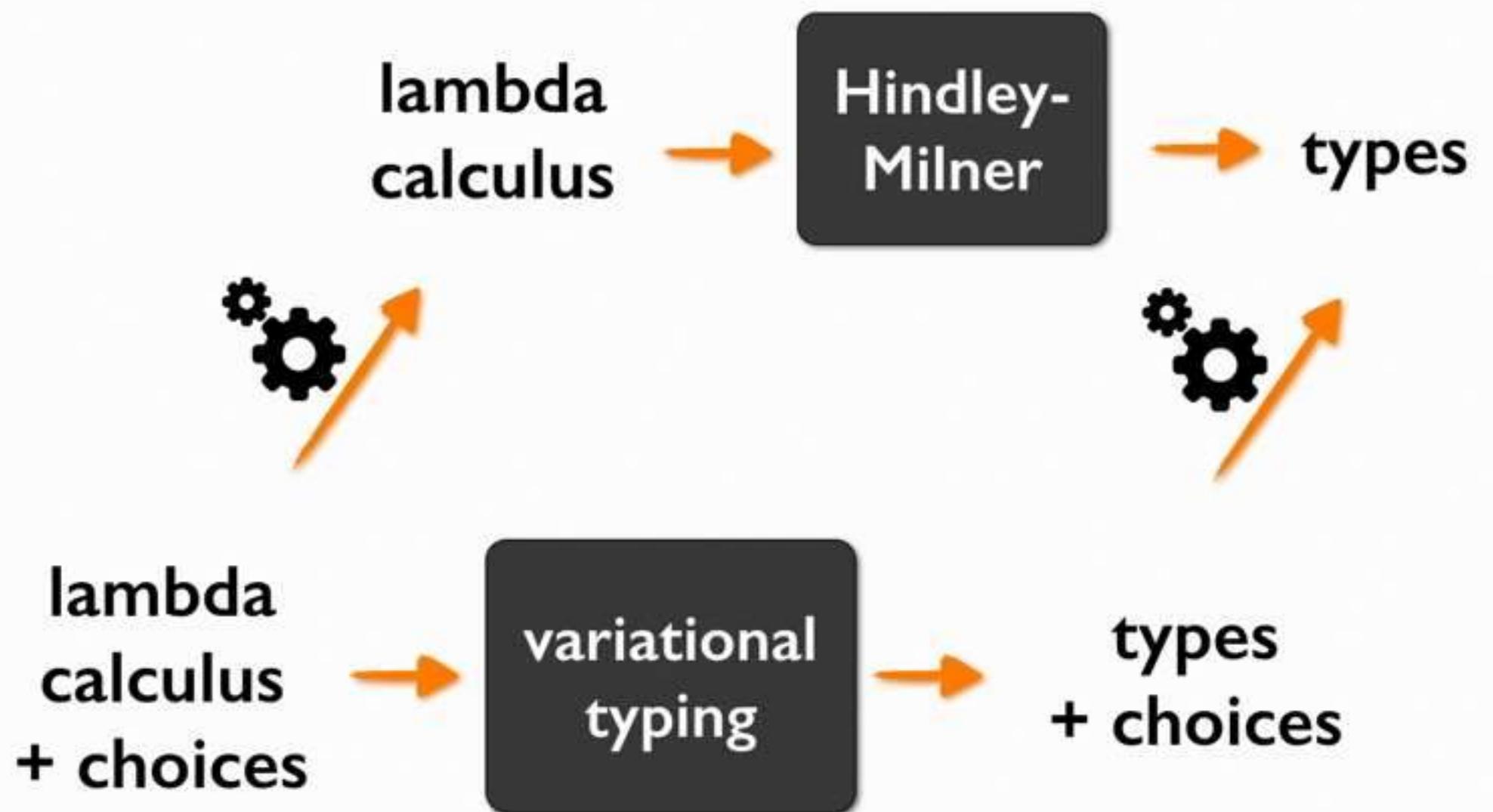
Variation in structured data

$$\begin{aligned} A([1,2,3], [1,2,4,5]) \\ \equiv [1,2,A(3,4),A(5,_)] \quad ?? \end{aligned}$$

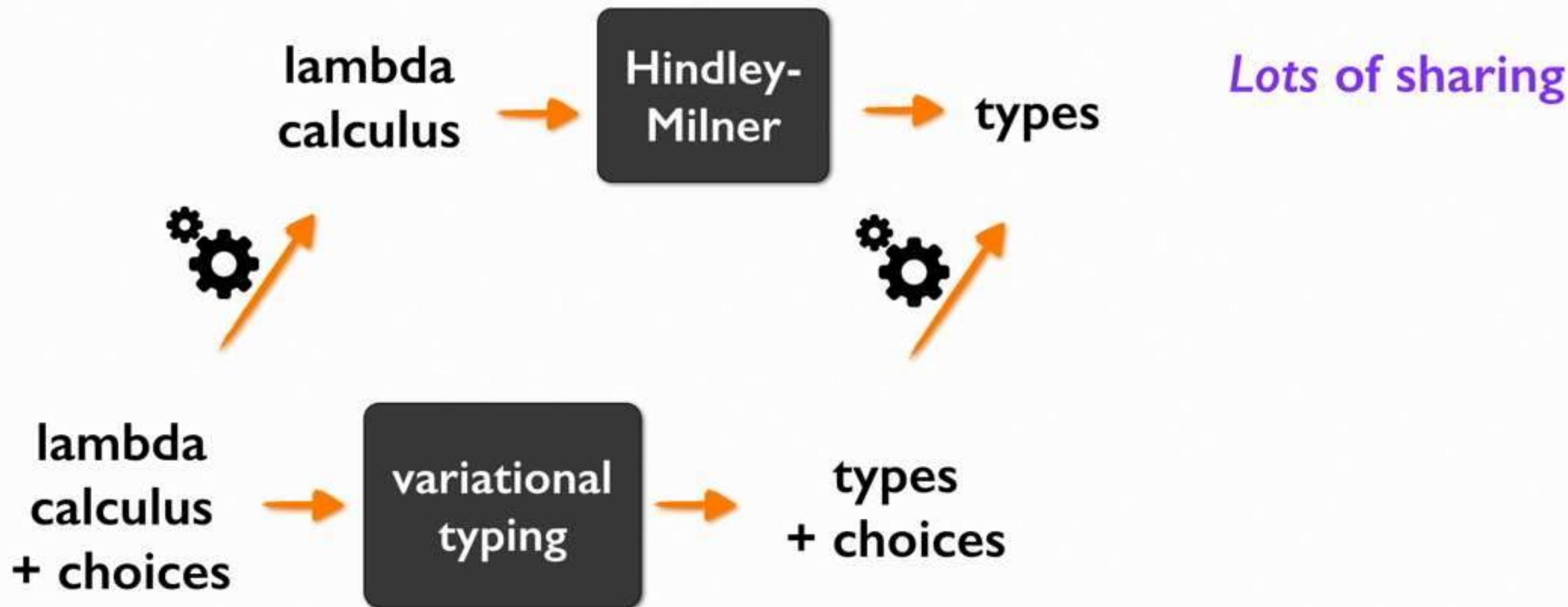
... need *variational data structures*

How to handle side effects?

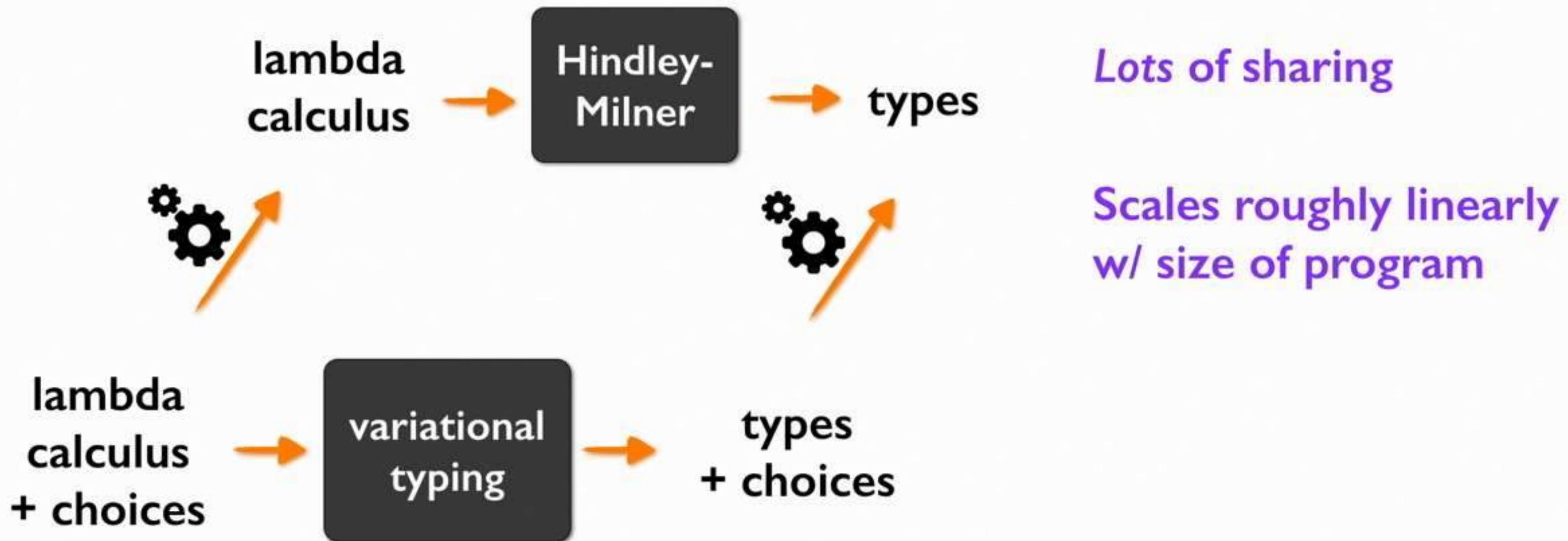
Variational typing



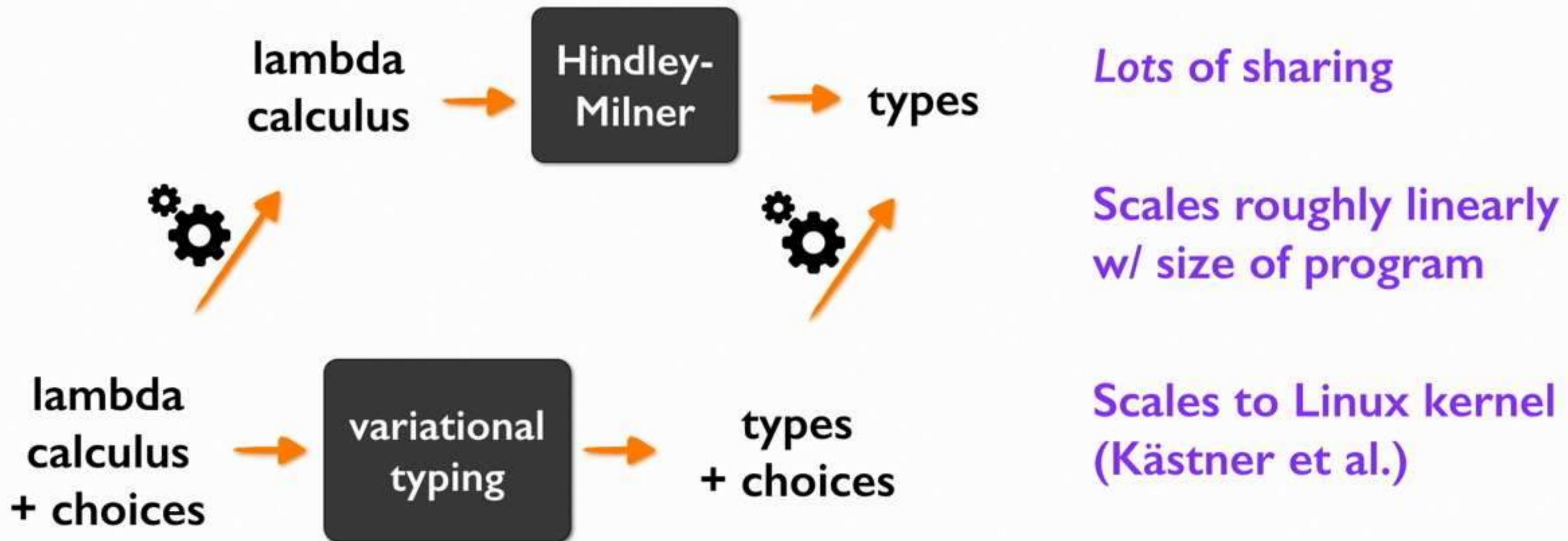
Variational typing



Variational typing



Variational typing



Only good for product line analyses?



Only good for product line analyses?

What makes a good nail?

- lots of variation
- lots of sharing



Only good for product line analyses?

What makes a good nail?

- lots of variation
- lots of sharing

Idea: use variation to explore hypothetical scenarios ...



Why don't we have both?

Error location

```
fold f z []      = [z]
fold f z (h:t) = fold f (f z h) t

flip f x y = f y x

reverse = fold (flip (:)) []

palindrome xs = reverse xs == xs
```

Error location

```
fold f z []      = [z]
fold f z (h:t) = fold f (f z h) t

flip f x y = f y x

reverse = fold (flip (:)) []

palindrome xs = reverse xs == xs
```

- Occurs check: cannot construct the infinite type: $a \sim [a]$
Expected type: $[[a]]$
Actual type: $[a]$
- In the second argument of ' $(==)$ ', namely ' xs '
In the expression: $reverse\ xs\ ==\ xs$

Error location

```
fold f z []      = [z]
fold f z (h:t) = fold f (f z h) t

flip f x y = f y x

reverse = fold (flip (:)) []

palindrome xs = reverse xs == xs
```

*error is in base case of fold
... but fold type checks!*

use of fold also type checks!

error finally detected

- Occurs check: cannot construct the infinite type: $a \sim [a]$
Expected type: $[[a]]$
Actual type: $[a]$
- In the second argument of ' $(==)$ ', namely ' xs '
In the expression: $reverse\ xs\ ==\ xs$

Counterfactual typing

Problem: locating the cause of a type error is hard

- type inference *commits too early*
- a successfully inferred type could be wrong!

Error location

```
fold f z []      = [z]
fold f z (h:t) = fold f (f z h) t

flip f x y = f y x

reverse = fold (flip (:)) []

palindrome xs = reverse xs == xs
```

*error is in base case of fold
... but fold type checks!*

use of fold also type checks!

error finally detected

- Occurs check: cannot construct the infinite type: $a \sim [a]$
Expected type: $[[a]]$
Actual type: $[a]$
- In the second argument of ' $(==)$ ', namely ' xs '
In the expression: $reverse\ xs\ ==\ xs$

Counterfactual typing

Problem: locating the cause of a type error is hard

- type inference *commits too early*
- a successfully inferred type could be wrong!

Counterfactual typing

Problem: locating the cause of a type error is hard

- type inference *commits too early*
- a successfully inferred type could be wrong!

Solution:



Counterfactual typing

Problem: locating the cause of a type error is hard

- type inference *commits too early*
- a successfully inferred type could be wrong!

Solution:



I. Error-tolerant variational type inference
where for every subexpression

$$e : T \rightarrow e : d(T, a) \quad d \& a \text{ are fresh}$$

Counterfactual typing

Problem: locating the cause of a type error is hard

- type inference *commits too early*
- a successfully inferred type could be wrong!

Solution:



I. Error-tolerant variational type inference
where for every subexpression

$$e : T \rightarrow e : d(T, a) \quad d \& a \text{ are fresh}$$

2. Search output variational type for

- non-error type
- as few right selections as possible

Migrating gradual types

```
def f(mode:bool, x):  
    if mode:  
        return even(x)  
    else:  
        return not(x)
```

Gradual typing
mix static and dynamic types
in the same program

Migrating gradual types

```
def f(mode:bool, x):  
    if mode:  
        return even(x)  
    else:  
        return not(x)
```

Gradual typing
mix static and dynamic types
in the same program

Migration challenges: (adding/removing annotations)

- mutually exclusive annotations

Migrating gradual types

```
def f(mode:bool, x):  
    if mode:  
        return even(x)  
    else:  
        return not(x)
```

Gradual typing
mix static and dynamic types
in the same program

Migration challenges: (adding/removing annotations)

- mutually exclusive annotations
- local type-safety maxima
- potential for extreme performance degradation

Migrational typing

Problem: migrating gradual types is perilous
and exploration by trial-and-error is infeasible

Solution:



1. Every parameter is initially $d(a, ?)$
 d & a are fresh $?$ is the dynamic type
2. Variational gradual type inference + cost analysis
output = summary of all possible migrations
3. Filter/search variational output
most static = fewest right selections
cheapest = lowest cost

Variational programming take aways

Key ideas

- make variation *explicit* in programs and data,
then compute with it!
- share as much as possible between variants

Can efficiently explore lots of hypothetical scenarios





Chapel Comes of Age: Productive Parallelism at Scale

Brad Chamberlain, Chapel Team, Cray Inc.

PNW PLSE Workshop

May 14, 2018



COMPUTE

STORE

ANALYZE



Chapel: Niche or Quiche?

Brad Chamberlain, Chapel Team, Cray Inc.

PNW PLSE Workshop

May 14, 2018



COMPUTE

|

STORE

ANALYZE

What is Chapel?



Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative



COMPUTE

STORE

ANALYZE

What is Chapel?

Chapel: A productive parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

- Support general parallel programming
 - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive



Chapel and Productivity



Chapel aims to be as...

...programmable as Python

...fast as Fortran

...scalable as MPI

...portable as C

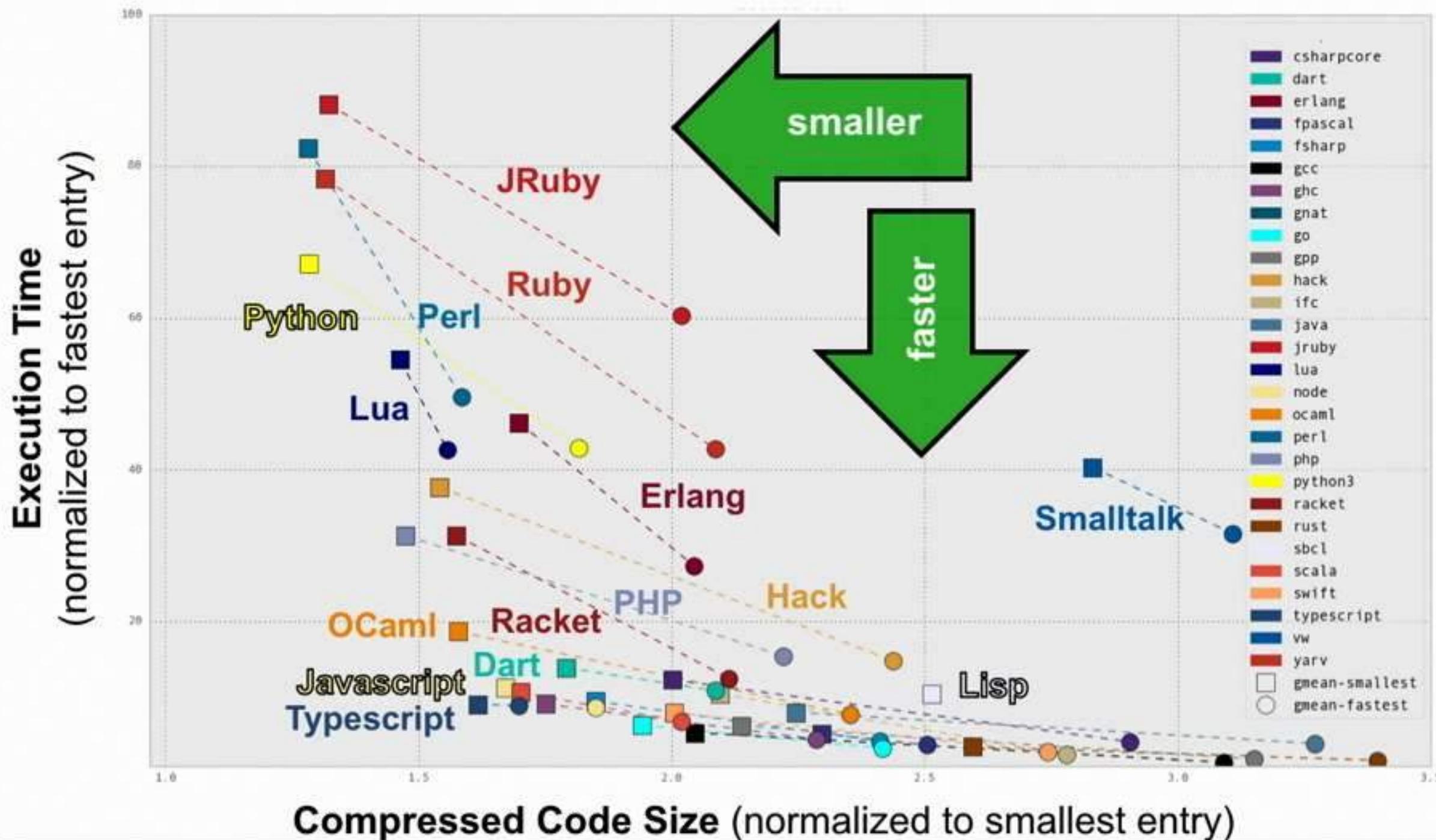
...flexible as C++

...fun as [your favorite programming language]



CLBG Cross-Language Summary

(Oct 2017 standings)



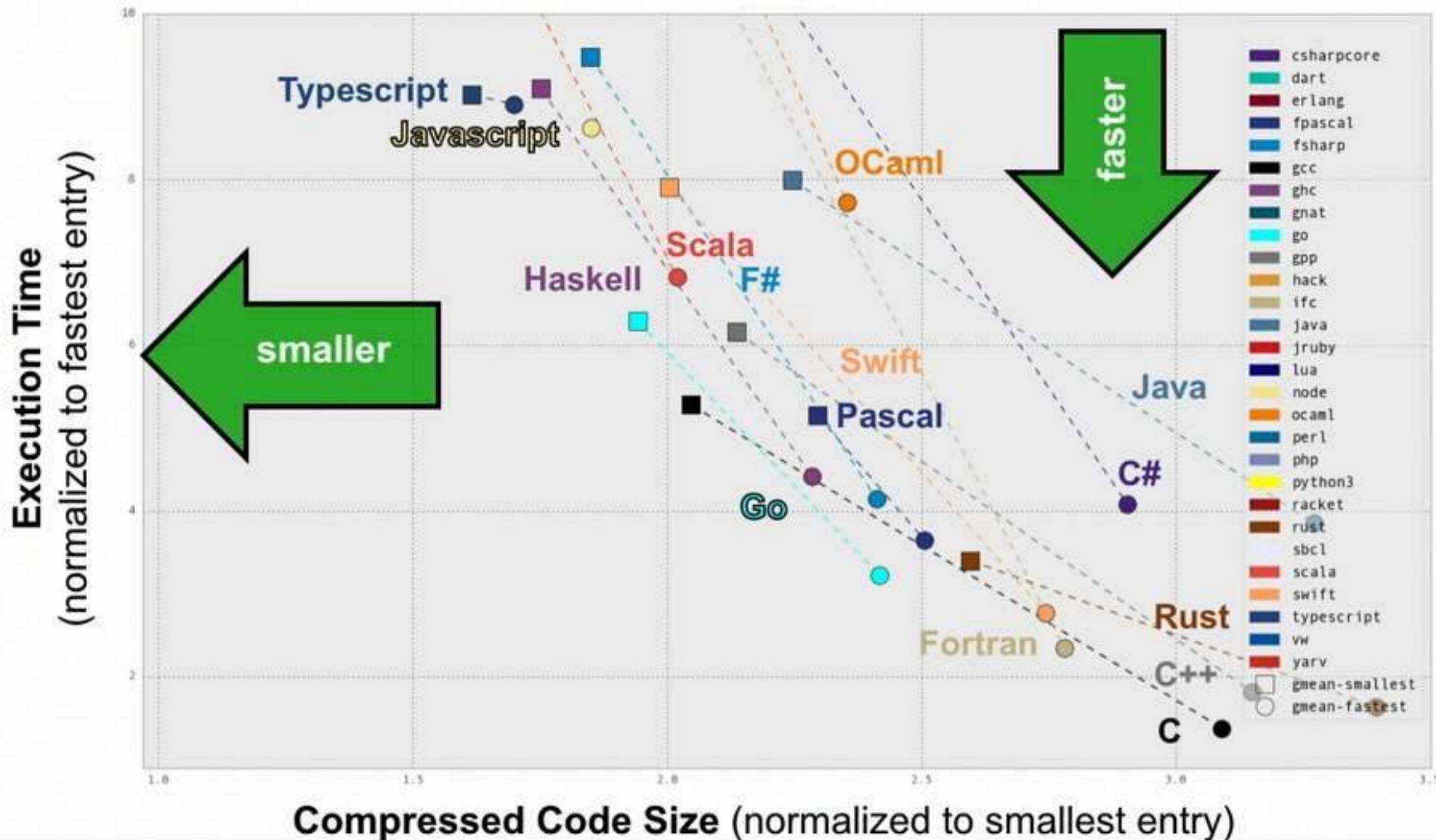
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



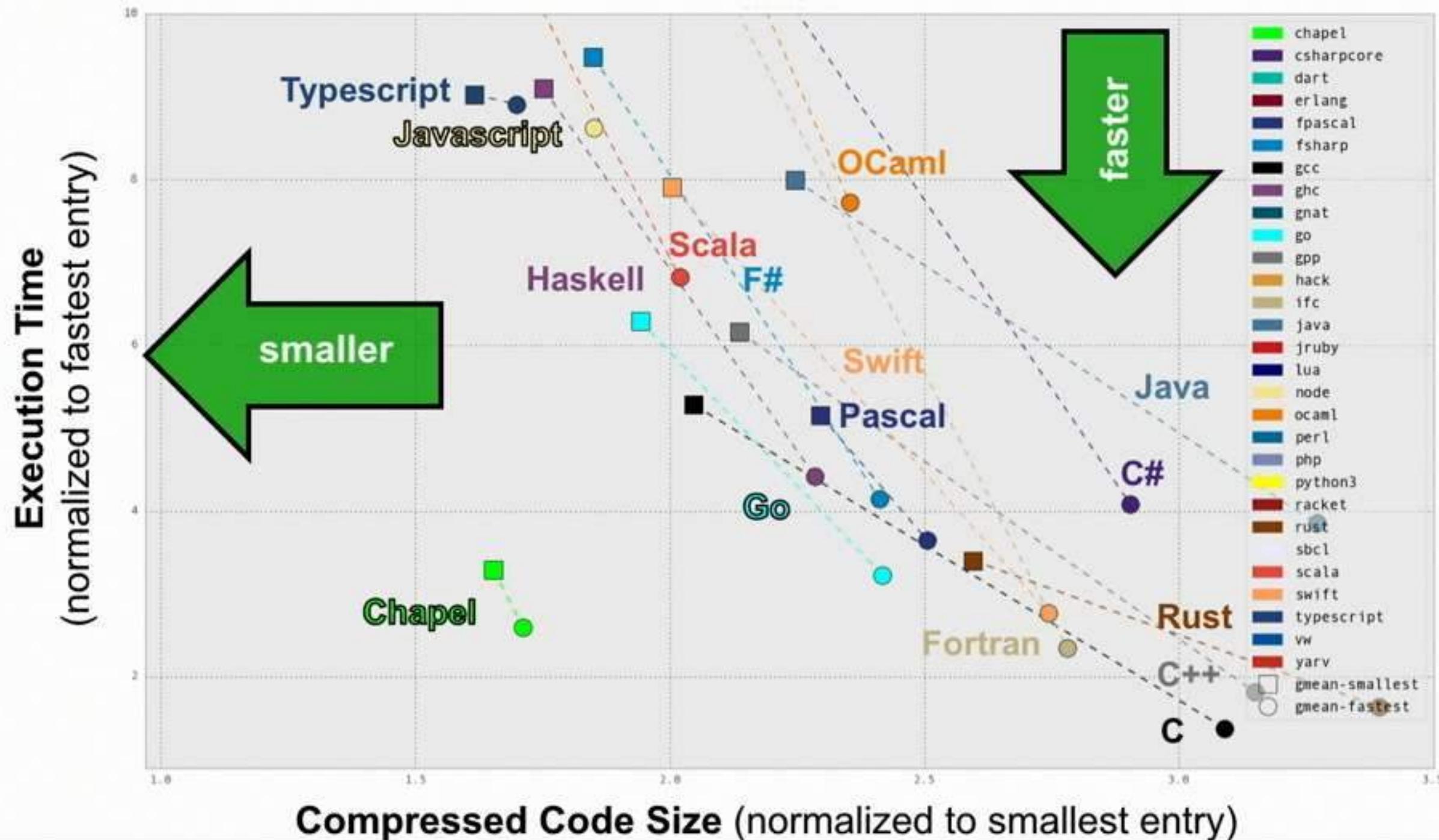
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



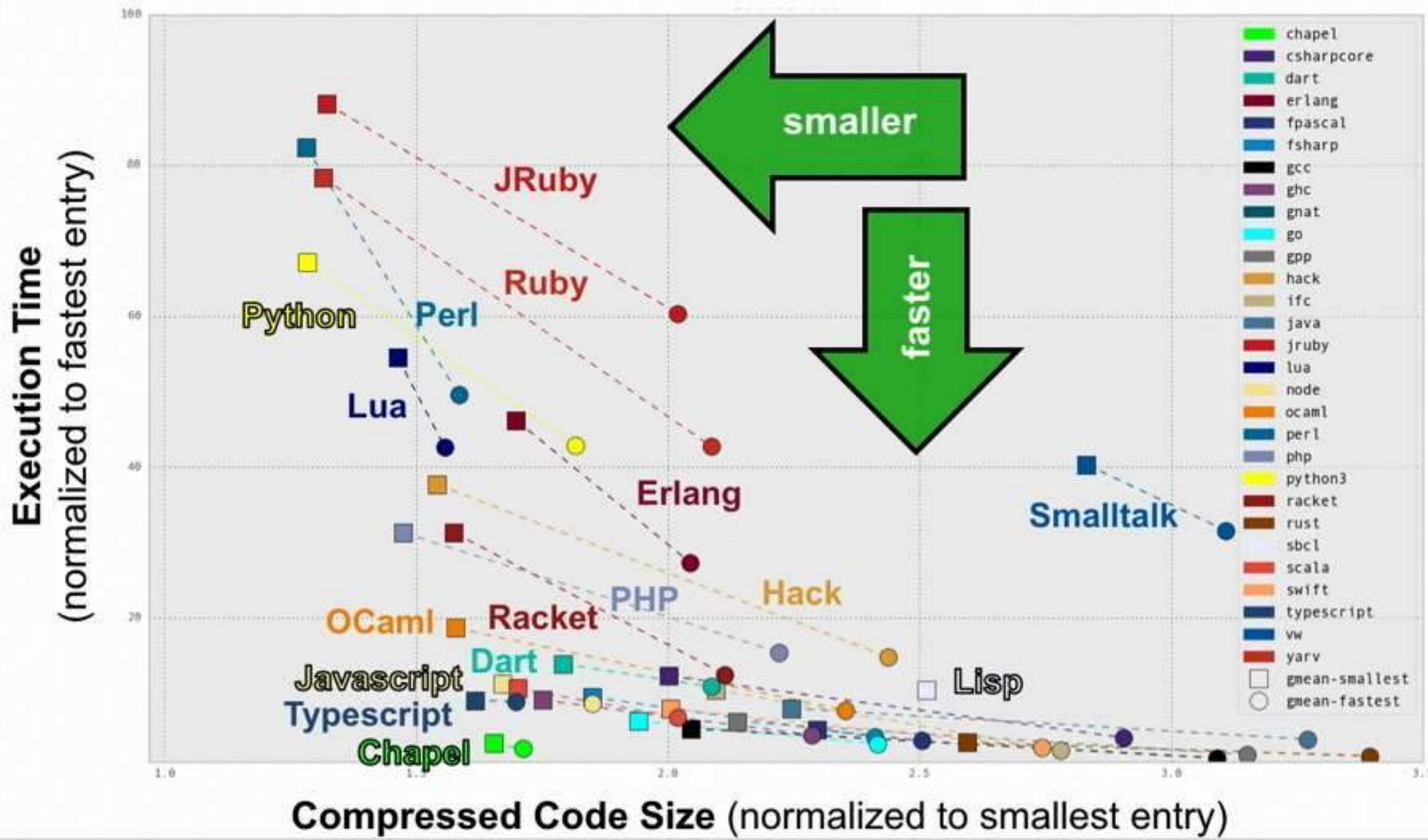
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Code Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {
    printColorEquations();

    const group1 = {i in 1..popSize1} new Chameneos(i, ((i-1)%3):Color);
    const group2 = {i in 1..popSize2} new Chameneos(i, colors10[i]);

    cobegin {
        holdMeetings(group1, n);
        holdMeetings(group2, n);
    }

    print(group1);
    print(group2);

    for c in group1 do delete c;
    for c in group2 do delete c;
}

// Print the results of getNewColor() for all color pairs.
// proc printColorEquations() {
//     for c1 in Color do
//         for c2 in Color do
//             writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
//     writeln();
// }

// Hold meetings among the population by creating a shared meeting
// place, and then creating per-chameneos tasks to have meetings.
// proc holdMeetings(population, numMeetings) {
//     const place = new MeetingPlace(numMeetings);

//     coforall c in population do          // create a task per chameneos
//         c.haveMeetings(place, population);

//     delete place;
// }
```

excerpt from 1210.gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t active_cpus;
    FILE* f;
    char buf[2048];
    pos;
    char const* int;
    int int;
    int int;
    int int;
    size_t int;
    size_t size_t;
    size_t size_t;

    char const* processor_str = "processor";
    size_t processor_str_len = strlen(processor_str);
    char const* physical_id_str = "physical id";
    size_t physical_id_str_len = strlen(physical_id_str);
    char const* core_id_str = "core id";
    size_t core_id_str_len = strlen(core_id_str);
    char const* cpu_cores_str = "cpu cores";
    size_t cpu_cores_str_len = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Code Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {
    printColorEquations();

    const group1 = [i in 1..popSize1] new Chameneos(i, 0);
    const group2 = [i in 1..popSize2] new Chameneos(i, 0);
    .....

    cobegin {
        holdMeetings(group1, n);
        holdMeetings(group2, n);
    }

    print(group1);
    print(group2);

    for c in group1 do delete c;
    for c in group2 do delete c;
}

// Print the results of getNewColor() for all colors
// per Chameneos
proc printColorEquations() {
    for c1 in Color do
        for c2 in Color do
            writeln(c1, " + ", c2, " = ", getNewColor(c1, c2));
    writeln();
}

// Hold meetings among the population by creating a shared
// Place, and then creating per-chameneos tasks to have
// them meet
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

    coforall c in population do          // create a task
        c.haveMeetings(place, population);

    delete place;
}
```

excerpt from 1210.gz Chapel entry

```
cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
}
```

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    active_cpus;
    f;
    buf [2048];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_cores;
    apic_id;
    cpu_count;
    i;

    processor_str      = "processor";
    processor_str_len = strlen(processor_str);
    physical_id_str   = "physical id";
    physical_id_str_len = strlen(physical_id_str);
    core_id_str        = "core id";
    core_id_str_len   = strlen(core_id_str);
    cores              = "cores";
    cores_len          = strlen(cores);
    cpu_cores_str      = "cpu_cores";
    cpu_cores_str_len = strlen(cpu_cores_str);
}
```

```
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

    coforall c in population do          // create a task
        c.haveMeetings(place, population);

    delete place;
}
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Code Comparisons



Can also browse program source code (*but this requires actual thought!*):

```
proc main() {  
  
    char const* core_id_str = "core id"  
    size_t core_id_str_len = strlen(core_id_str);  
    char const* cpu_cores_str = "cpu cores"  
    size_t cpu_cores_str_len = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
}
```

excerpt from 1210.gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)  
{  
    cpu_set_t active_cpus;  
    FILE* f;  
    char buf[2048];  
    pos;  
    cpu_idx;  
    physical_id;  
    core_id;  
    cpu_cores;  
    apic_id;  
    cpu_count;  
    i;  
  
    char const* processor_str = "processor";  
    size_t processor_str_len = strlen(processor_str);  
    physical_id_str = "physical id";  
    physical_id_str_len = strlen(physical_id_str);  
    core_id_str = "core id";  
    core_id_str_len = strlen(core_id_str);  
    cpu_cores_str = "cpu cores";  
    cpu_cores_str_len = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
    is_smp[0] = 1;  
    CPU_ZERO(affinity1);
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

Excerpt from PNW PLSE Review

*“Chapel has been around for quite a while,
and it still seems like a niche language...”*





Chapel: “A Niche Language”?

Chapel is arguably niche in that it...
...was originally designed for HPC



COMPUTE

STORE

ANALYZE

Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

...was originally designed for HPC

Yet, Chapel’s chief concerns aren’t HPC-specific:

- performance



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

...was originally designed for HPC

Yet, Chapel’s chief concerns aren’t HPC-specific:

- performance
- programmability (cf. Python)



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

...was originally designed for HPC

Yet, Chapel's chief concerns aren't HPC-specific:

- performance
- programmability (cf. Python)
- parallelism (cf. multicore)



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

...was originally designed for HPC

Yet, Chapel’s chief concerns aren’t HPC-specific:

- performance
- programmability (cf. Python)
- parallelism (cf. multicore)
- distributed memory (cf. cloud computing)



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

- ...was originally designed for HPC

- ...has only a modest-sized community (so far)



Chapel: “A Niche Language”?

Chapel is arguably niche in that it...

- ...was originally designed for HPC

- ...has only a modest-sized community (so far)

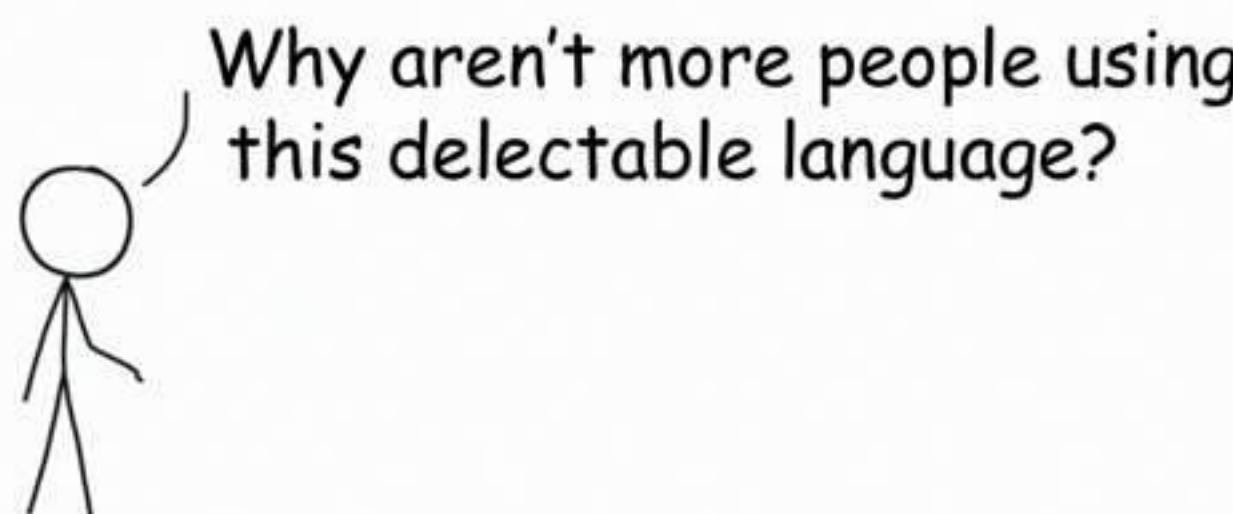
Yet, we've historically discouraged its use in production...



Chapel: A Quiche Language!



The outsider's impression:



COMPUTE

STORE

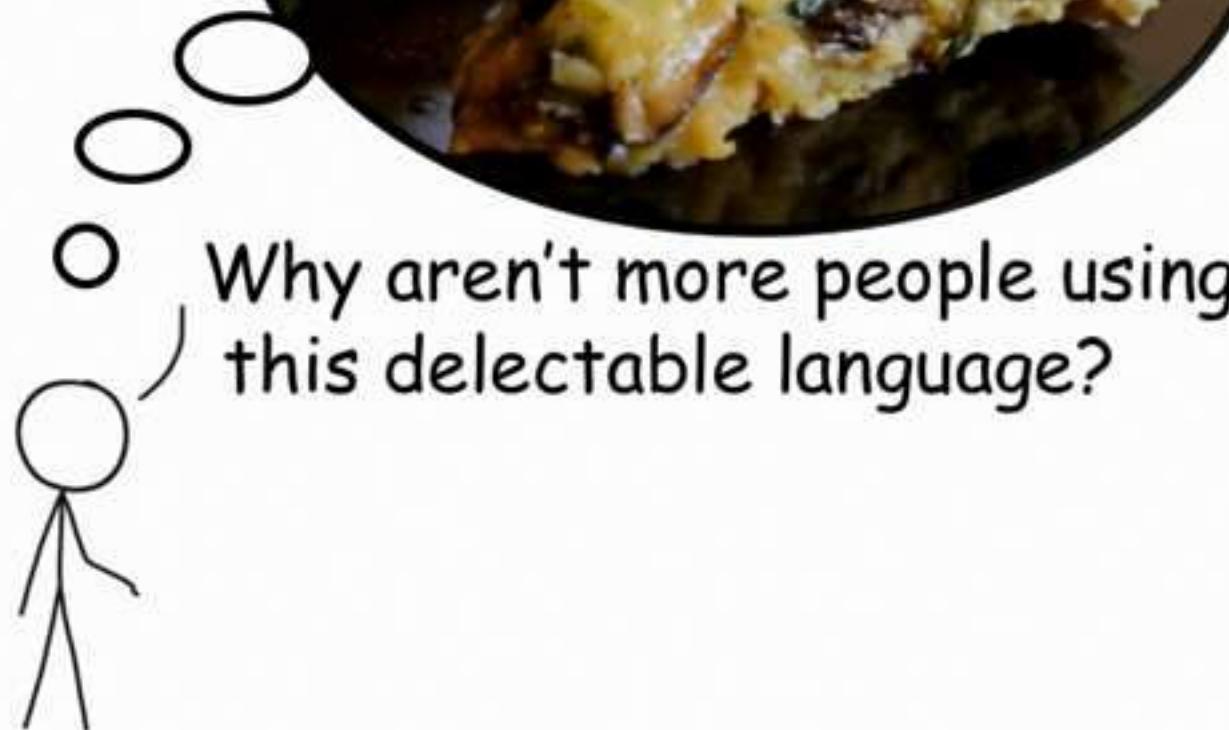
ANALYZE

Copyright 2018 Cray Inc.

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>

Chapel: A Quiche Language!

The outsider's impression:



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>

Chapel: A Quiche Language!



The outsider's impression:



Why aren't more people using
this delectable language?

The reality, for most of Chapel's history:



COMPUTE

STORE

ANALYZE

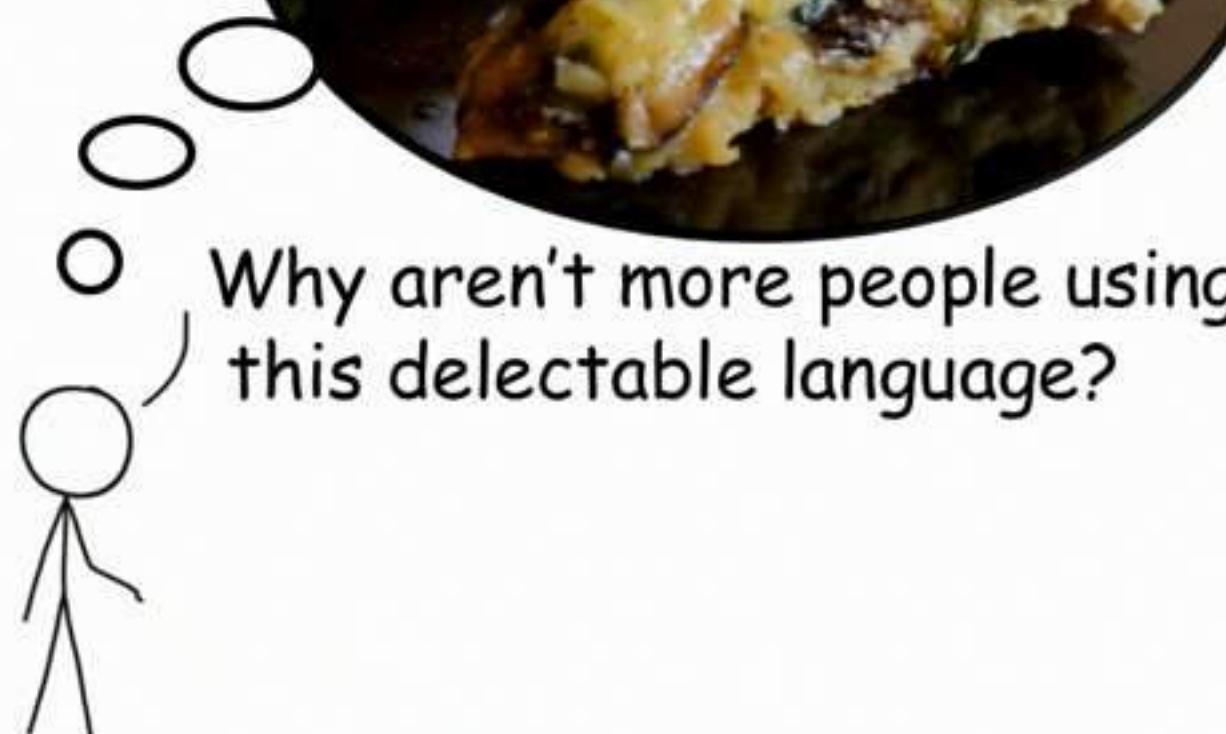
Copyright 2018 Cray Inc.

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>

Chapel: A Quiche Language!



The outsider's impression:



Though recently, it's more like:



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Image sources: <https://www.themountaintinkitchen.com>, <https://xkcd.com/>

Chapel: “Been Around for Quite Awhile”



Chapel’s Infancy: DARPA HPCS (2003–2012)

- Research focus: ~6-7 FTEs
 - distinguish locality from parallelism
 - seamlessly mix data- and task-parallelism
 - support user-defined distributed arrays, parallel iterators



COMPUTE

STORE

ANALYZE

Chapel: “Been Around for Quite Awhile”



Chapel’s Infancy: DARPA HPCS (2003–2012)

- Research focus: ~6-7 FTEs
 - distinguish locality from parallelism
 - seamlessly mix data- and task-parallelism
 - support user-defined distributed arrays, parallel iterators

Chapel’s Adolescence: “the five-year push” (2013–2018)

- Development focus: ~13-14 FTEs
 - **performance and scalability**
 - **ecosystem:** documentation, libraries, tools, ...
 - **base language fixes:** OOP features, error-handling, strings, ...



COMPUTE

STORE

ANALYZE

Chapel: “Been Around for Quite Awhile”



Chapel’s Infancy: DARPA HPCS (2003–2012)

- Research focus: ~6-7 FTEs
 - distinguish locality from parallelism
 - seamlessly mix data- and task-parallelism
 - support user-defined distributed arrays, parallel iterators

Chapel’s Adolescence: “the five-year push” (2013–2018)

- Development focus: ~13-14 FTEs
 - **performance and scalability**
 - **ecosystem:** documentation, libraries, tools, ...
 - **base language fixes:** OOP features, error-handling, strings, ...

Then Now



COMPUTE

STORE

ANALYZE

Chapel Ecosystem: Then vs. Now



COMPUTE

STORE

ANALYZE

Documentation: Then

After HPCS:

- a PDF language specification

Chapel Language Specification
Version 0.93

Cray Inc.
901 Fifth Avenue, Suite 1000
Seattle, WA 98164

April 18, 2013



COMPUTE

STORE

ANALYZE



Documentation: Then



After HPCS:

- a PDF language specification
- a Quick Reference sheet

Chapel Language Specification
Version 0.93

Cray Inc.
901 Fifth Avenue, Suite 1000
Seattle, WA 98164

April 18, 2013

Chapel Quick Reference

Quick Start

Use `git clone https://github.com/cray/cray.git` to get a copy of the "full" Chapel source code.

1. Create the file `hello.chpl` with the following code:

```
#!/usr/bin/env chpl
x = 1;
y = 2;
```

2. Compile and run it:

```
chpl hello.chpl
./hello
```

Comments

/* single-line comments */
// multi-line comments
*/

Primitive Types

Type	Default init	Final value	Default init
bool	bool(true)	0.0	0.0
int	0	0	0
float	0.0	0.0	0.0
double	0.0	0.0	0.0
complex	0.0 + 0.0i	0.0	0.0 + 0.0i
string	""	"	"

Variables, Constants and Configuration

var x: int = 1; // constant of type int at 0.000
var y: float = 2.0; // constant of type float at 2.000
const pi: float = 3.141592653589793; // constant of type float at 3.141592653589793
const e: float = 2.718281828459045; // constant of type float at 2.718281828459045
const inf: float = inf; // constant of type float at infinity

Modules

module M { var x = 10; } // module definition
module N { var y = 20; } // module definition
print(M.x); // prints 10
print(N.y); // prints 20

Expression Precedence and Associativity

Operations

1.12 member access, call and index
1.13 conversion rule
1.14 cast
1.15 type
1.16 expression
1.17 assignment
1.18 if-then-else
1.19 for
1.20 while
1.21 do-while
1.22 for-each
1.23 break
1.24 continue
1.25 switch
1.26 case
1.27 default
1.28 return
1.29 throw
1.30 try-catch
1.31 catch
1.32 finally
1.33 catch-all
1.34 catch-all
1.35 catch-all
1.36 catch-all
1.37 catch-all
1.38 catch-all
1.39 catch-all
1.40 catch-all
1.41 catch-all
1.42 catch-all
1.43 catch-all
1.44 catch-all
1.45 catch-all
1.46 catch-all
1.47 catch-all
1.48 catch-all
1.49 catch-all
1.50 catch-all
1.51 catch-all
1.52 catch-all
1.53 catch-all
1.54 catch-all
1.55 catch-all
1.56 catch-all
1.57 catch-all
1.58 catch-all
1.59 catch-all
1.60 catch-all
1.61 catch-all
1.62 catch-all
1.63 catch-all
1.64 catch-all
1.65 catch-all
1.66 catch-all
1.67 catch-all
1.68 catch-all
1.69 catch-all
1.70 catch-all
1.71 catch-all
1.72 catch-all
1.73 catch-all
1.74 catch-all
1.75 catch-all
1.76 catch-all
1.77 catch-all
1.78 catch-all
1.79 catch-all
1.80 catch-all
1.81 catch-all
1.82 catch-all
1.83 catch-all
1.84 catch-all
1.85 catch-all
1.86 catch-all
1.87 catch-all
1.88 catch-all
1.89 catch-all
1.90 catch-all
1.91 catch-all
1.92 catch-all
1.93 catch-all
1.94 catch-all
1.95 catch-all
1.96 catch-all
1.97 catch-all
1.98 catch-all
1.99 catch-all
1.100 catch-all
1.101 catch-all
1.102 catch-all
1.103 catch-all
1.104 catch-all
1.105 catch-all
1.106 catch-all
1.107 catch-all
1.108 catch-all
1.109 catch-all
1.110 catch-all
1.111 catch-all
1.112 catch-all
1.113 catch-all
1.114 catch-all
1.115 catch-all
1.116 catch-all
1.117 catch-all
1.118 catch-all
1.119 catch-all
1.120 catch-all
1.121 catch-all
1.122 catch-all
1.123 catch-all
1.124 catch-all
1.125 catch-all
1.126 catch-all
1.127 catch-all
1.128 catch-all
1.129 catch-all
1.130 catch-all
1.131 catch-all
1.132 catch-all
1.133 catch-all
1.134 catch-all
1.135 catch-all
1.136 catch-all
1.137 catch-all
1.138 catch-all
1.139 catch-all
1.140 catch-all
1.141 catch-all
1.142 catch-all
1.143 catch-all
1.144 catch-all
1.145 catch-all
1.146 catch-all
1.147 catch-all
1.148 catch-all
1.149 catch-all
1.150 catch-all
1.151 catch-all
1.152 catch-all
1.153 catch-all
1.154 catch-all
1.155 catch-all
1.156 catch-all
1.157 catch-all
1.158 catch-all
1.159 catch-all
1.160 catch-all
1.161 catch-all
1.162 catch-all
1.163 catch-all
1.164 catch-all
1.165 catch-all
1.166 catch-all
1.167 catch-all
1.168 catch-all
1.169 catch-all
1.170 catch-all
1.171 catch-all
1.172 catch-all
1.173 catch-all
1.174 catch-all
1.175 catch-all
1.176 catch-all
1.177 catch-all
1.178 catch-all
1.179 catch-all
1.180 catch-all
1.181 catch-all
1.182 catch-all
1.183 catch-all
1.184 catch-all
1.185 catch-all
1.186 catch-all
1.187 catch-all
1.188 catch-all
1.189 catch-all
1.190 catch-all
1.191 catch-all
1.192 catch-all
1.193 catch-all
1.194 catch-all
1.195 catch-all
1.196 catch-all
1.197 catch-all
1.198 catch-all
1.199 catch-all
1.200 catch-all
1.201 catch-all
1.202 catch-all
1.203 catch-all
1.204 catch-all
1.205 catch-all
1.206 catch-all
1.207 catch-all
1.208 catch-all
1.209 catch-all
1.210 catch-all
1.211 catch-all
1.212 catch-all
1.213 catch-all
1.214 catch-all
1.215 catch-all
1.216 catch-all
1.217 catch-all
1.218 catch-all
1.219 catch-all
1.220 catch-all
1.221 catch-all
1.222 catch-all
1.223 catch-all
1.224 catch-all
1.225 catch-all
1.226 catch-all
1.227 catch-all
1.228 catch-all
1.229 catch-all
1.230 catch-all
1.231 catch-all
1.232 catch-all
1.233 catch-all
1.234 catch-all
1.235 catch-all
1.236 catch-all
1.237 catch-all
1.238 catch-all
1.239 catch-all
1.240 catch-all
1.241 catch-all
1.242 catch-all
1.243 catch-all
1.244 catch-all
1.245 catch-all
1.246 catch-all
1.247 catch-all
1.248 catch-all
1.249 catch-all
1.250 catch-all
1.251 catch-all
1.252 catch-all
1.253 catch-all
1.254 catch-all
1.255 catch-all
1.256 catch-all
1.257 catch-all
1.258 catch-all
1.259 catch-all
1.260 catch-all
1.261 catch-all
1.262 catch-all
1.263 catch-all
1.264 catch-all
1.265 catch-all
1.266 catch-all
1.267 catch-all
1.268 catch-all
1.269 catch-all
1.270 catch-all
1.271 catch-all
1.272 catch-all
1.273 catch-all
1.274 catch-all
1.275 catch-all
1.276 catch-all
1.277 catch-all
1.278 catch-all
1.279 catch-all
1.280 catch-all
1.281 catch-all
1.282 catch-all
1.283 catch-all
1.284 catch-all
1.285 catch-all
1.286 catch-all
1.287 catch-all
1.288 catch-all
1.289 catch-all
1.290 catch-all
1.291 catch-all
1.292 catch-all
1.293 catch-all
1.294 catch-all
1.295 catch-all
1.296 catch-all
1.297 catch-all
1.298 catch-all
1.299 catch-all
1.2000 catch-all
1.2001 catch-all
1.2002 catch-all
1.2003 catch-all
1.2004 catch-all
1.2005 catch-all
1.2006 catch-all
1.2007 catch-all
1.2008 catch-all
1.2009 catch-all
1.2010 catch-all
1.2011 catch-all
1.2012 catch-all
1.2013 catch-all
1.2014 catch-all
1.2015 catch-all
1.2016 catch-all
1.2017 catch-all
1.2018 catch-all
1.2019 catch-all
1.2020 catch-all
1.2021 catch-all
1.2022 catch-all
1.2023 catch-all
1.2024 catch-all
1.2025 catch-all
1.2026 catch-all
1.2027 catch-all
1.2028 catch-all
1.2029 catch-all
1.2030 catch-all
1.2031 catch-all
1.2032 catch-all
1.2033 catch-all
1.2034 catch-all
1.2035 catch-all
1.2036 catch-all
1.2037 catch-all
1.2038 catch-all
1.2039 catch-all
1.2040 catch-all
1.2041 catch-all
1.2042 catch-all
1.2043 catch-all
1.2044 catch-all
1.2045 catch-all
1.2046 catch-all
1.2047 catch-all
1.2048 catch-all
1.2049 catch-all
1.2050 catch-all
1.2051 catch-all
1.2052 catch-all
1.2053 catch-all
1.2054 catch-all
1.2055 catch-all
1.2056 catch-all
1.2057 catch-all
1.2058 catch-all
1.2059 catch-all
1.2060 catch-all
1.2061 catch-all
1.2062 catch-all
1.2063 catch-all
1.2064 catch-all
1.2065 catch-all
1.2066 catch-all
1.2067 catch-all
1.2068 catch-all
1.2069 catch-all
1.2070 catch-all
1.2071 catch-all
1.2072 catch-all
1.2073 catch-all
1.2074 catch-all
1.2075 catch-all
1.2076 catch-all
1.2077 catch-all
1.2078 catch-all
1.2079 catch-all
1.2080 catch-all
1.2081 catch-all
1.2082 catch-all
1.2083 catch-all
1.2084 catch-all
1.2085 catch-all
1.2086 catch-all
1.2087 catch-all
1.2088 catch-all
1.2089 catch-all
1.2090 catch-all
1.2091 catch-all
1.2092 catch-all
1.2093 catch-all
1.2094 catch-all
1.2095 catch-all
1.2096 catch-all
1.2097 catch-all
1.2098 catch-all
1.2099 catch-all
1.2100 catch-all
1.2101 catch-all
1.2102 catch-all
1.2103 catch-all
1.2104 catch-all
1.2105 catch-all
1.2106 catch-all
1.2107 catch-all
1.2108 catch-all
1.2109 catch-all
1.2110 catch-all
1.2111 catch-all
1.2112 catch-all
1.2113 catch-all
1.2114 catch-all
1.2115 catch-all
1.2116 catch-all
1.2117 catch-all
1.2118 catch-all
1.2119 catch-all
1.2120 catch-all
1.2121 catch-all
1.2122 catch-all
1.2123 catch-all
1.2124 catch-all
1.2125 catch-all
1.2126 catch-all
1.2127 catch-all
1.2128 catch-all
1.2129 catch-all
1.2130 catch-all
1.2131 catch-all
1.2132 catch-all
1.2133 catch-all
1.2134 catch-all
1.2135 catch-all
1.2136 catch-all
1.2137 catch-all
1.2138 catch-all
1.2139 catch-all
1.2140 catch-all
1.2141 catch-all
1.2142 catch-all
1.2143 catch-all
1.2144 catch-all
1.2145 catch-all
1.2146 catch-all
1.2147 catch-all
1.2148 catch-all
1.2149 catch-all
1.2150 catch-all
1.2151 catch-all
1.2152 catch-all
1.2153 catch-all
1.2154 catch-all
1.2155 catch-all
1.2156 catch-all
1.2157 catch-all
1.2158 catch-all
1.2159 catch-all
1.2160 catch-all
1.2161 catch-all
1.2162 catch-all
1.2163 catch-all
1.2164 catch-all
1.2165 catch-all
1.2166 catch-all
1.2167 catch-all
1.2168 catch-all
1.2169 catch-all
1.2170 catch-all
1.2171 catch-all
1.2172 catch-all
1.2173 catch-all
1.2174 catch-all
1.2175 catch-all
1.2176 catch-all
1.2177 catch-all
1.2178 catch-all
1.2179 catch-all
1.2180 catch-all
1.2181 catch-all
1.2182 catch-all
1.2183 catch-all
1.2184 catch-all
1.2185 catch-all
1.2186 catch-all
1.2187 catch-all
1.2188 catch-all
1.2189 catch-all
1.2190 catch-all
1.2191 catch-all
1.2192 catch-all
1.2193 catch-all
1.2194 catch-all
1.2195 catch-all
1.2196 catch-all
1.2197 catch-all
1.2198 catch-all
1.2199 catch-all
1.2200 catch-all
1.2201 catch-all
1.2202 catch-all
1.2203 catch-all
1.2204 catch-all
1.2205 catch-all
1.2206 catch-all
1.2207 catch-all
1.2208 catch-all
1.2209 catch-all
1.2210 catch-all
1.2211 catch-all
1.2212 catch-all
1.2213 catch-all
1.2214 catch-all
1.2215 catch-all
1.2216 catch-all
1.2217 catch-all
1.2218 catch-all
1.2219 catch-all
1.2220 catch-all
1.2221 catch-all
1.2222 catch-all
1.2223 catch-all
1.2224 catch-all
1.2225 catch-all
1.2226 catch-all
1.2227 catch-all
1.2228 catch-all
1.2229 catch-all
1.2230 catch-all
1.2231 catch-all
1.2232 catch-all
1.2233 catch-all
1.2234 catch-all
1.2235 catch-all
1.2236 catch-all
1.2237 catch-all
1.2238 catch-all
1.2239 catch-all
1.2240 catch-all
1.2241 catch-all
1.2242 catch-all
1.2243 catch-all
1.2244 catch-all
1.2245 catch-all
1.2246 catch-all
1.2247 catch-all
1.2248 catch-all
1.2249 catch-all
1.2250 catch-all
1.2251 catch-all
1.2252 catch-all
1.2253 catch-all
1.2254 catch-all
1.2255 catch-all
1.2256 catch-all
1.2257 catch-all
1.2258 catch-all
1.2259 catch-all
1.2260 catch-all
1.2261 catch-all
1.2262 catch-all
1.2263 catch-all
1.2264 catch-all
1.2265 catch-all
1.2266 catch-all
1.2267 catch-all
1.2268 catch-all
1.2269 catch-all
1.2270 catch-all
1.2271 catch-all
1.2272 catch-all
1.2273 catch-all
1.2274 catch-all
1.2275 catch-all
1.2276 catch-all
1.2277 catch-all
1.2278 catch-all
1.2279 catch-all
1.2280 catch-all
1.2281 catch-all
1.2282 catch-all
1.2283 catch-all
1.2284 catch-all
1.2285 catch-all
1.2286 catch-all
1.2287 catch-all
1.2288 catch-all
1.2289 catch-all
1.2290 catch-all
1.2291 catch-all
1.2292 catch-all
1.2293 catch-all
1.2294 catch-all
1.2295 catch-all
1.2296 catch-all
1.2297 catch-all
1.2298 catch-all
1.2299 catch-all
1.2300 catch-all
1.2301 catch-all
1.2302 catch-all
1.2303 catch-all
1.2304 catch-all
1.2305 catch-all
1.2306 catch-all
1.2307 catch-all
1.2308 catch-all
1.2309 catch-all
1.2310 catch-all
1.2311 catch-all
1.2312 catch-all
1.2313 catch-all
1.2314 catch-all
1.2315 catch-all
1.2316 catch-all
1.2317 catch-all
1.2318 catch-all
1.2319 catch-all
1.2320 catch-all
1.2321 catch-all
1.2322 catch-all
1.2323 catch-all
1.2324 catch-all
1.2325 catch-all
1.2326 catch-all
1.2327 catch-all
1.2328 catch-all
1.2329 catch-all
1.2330 catch-all
1.2331 catch-all
1.2332 catch-all
1.2333 catch-all
1.2334 catch-all
1.2335 catch-all
1.2336 catch-all
1.2337 catch-all
1.2338 catch-all
1.2339 catch-all
1.2340 catch-all
1.2341 catch-all
1.2342 catch-all
1.2343 catch-all
1.2344 catch-all
1.2345 catch-all
1.2346 catch-all
1.2347 catch-all
1.2348 catch-all
1.2349 catch-all
1.2350 catch-all
1.2351 catch-all
1.2352 catch-all
1.2353 catch-all
1.2354 catch-all
1.2355 catch-all
1.2356 catch-all
1.2357 catch-all
1.2358 catch-all
1.2359 catch-all
1.2360 catch-all
1.2361 catch-all
1.2362 catch-all
1.2363 catch-all
1.2364 catch-all
1.2365 catch-all
1.2366 catch-all
1.2367 catch-all
1.2368 catch-all
1.2369 catch-all
1.2370 catch-all
1.2371 catch-all
1.2372 catch-all
1.2373 catch-all
1.2374 catch-all
1.2375 catch-all
1.2376 catch-all
1.2377 catch-all
1.2378 catch-all
1.2379 catch-all
1.2380 catch-all
1.2381 catch-all
1.2382 catch-all
1.2383 catch-all
1.2384 catch-all
1.2385 catch-all
1.2386 catch-all
1.2387 catch-all
1.2388 catch-all
1.2389 catch-all
1.2390 catch-all
1.2391 catch-all
1.2392 catch-all
1.2393 catch-all
1.2394 catch-all
1.2395 catch-all
1.2396 catch-all
1.2397 catch-all
1.2398 catch-all
1.2399 catch-all
1.2400 catch-all
1.2401 catch-all
1.2402 catch-all
1.2403 catch-all
1.2404 catch-all
1.2405 catch-all
1.2406 catch-all
1.2407 catch-all
1.2408 catch-all
1.2409 catch-all
1.2410 catch-all
1.2411 catch-all
1.2412 catch-all
1.2413 catch-all
1.2414 catch-all
1.2415 catch-all
1.2416 catch-all
1.2417 catch-all
1.2418 catch-all
1.2419 catch-all
1.2420 catch-all
1.2421 catch-all
1.2422 catch-all
1.2423 catch-all
1.2424 catch-all
1.2425 catch-all
1.2426 catch-all
1.2427 catch-all
1.2428 catch-all
1.2429 catch-all
1.2430 catch-all
1.2431 catch-all
1.2432 catch-all
1.2433 catch-all
1.2434 catch-all
1.2435 catch-all

Documentation: Then

After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs

```

bradc -- sam.bradc@troll.cray.com -- bash
File Edit Options Buffers Tools Help
Chapel doc README
This directory contains the following documentation:
README : this file
README.bugs : how to report bugs or suggestions to the Chapel team
README.building : information about building the Chapel compiler
README.chplenv : setting up your environment to use Chapel
README.compiling : how to use the Chapel compiler to compile code
README.executing : execution options for Chapel programs
README.multilocale : how to execute Chapel on multiple locales
README.threads : explains how Chapel tasks are implemented using threads
README.xt-cnl : notes for Cray XT (MPCOS/cn) users
README.cygwin : notes for Cygwin users
README.extern : technical note on interfacing with external C routines
README.Format : technical note on controlling value-to-string formatting
README.prereqs : prerequisites for using Chapel

chapelLanguageSpec.pdf : the current draft of the Chapel language
specification

hpccOverview.pdf : a high-level overview of our implementations of
the MPC Challenge benchmarks for STREAM Triad,
Random Access, and FFT in Chapel

hpccTutorial.pdf : a companion paper to the previous that provides a
detailed walkthrough of our implementations of
the MPC Benchmarks to serve as a tutorial to
Chapel and the codes themselves

quickReference.pdf : a one-sheet, tri-fold overview of Chapel syntax
for quick reference

For more Information
For additional information about Chapel, please refer to:
* "Parallel Programmability and the Chapel Language" by Bradford
L. Chamberlain, David Callahan, and Hans P. Zima, published in the
International Journal of High Performance Computing Applications,
August 2007, 21(3): 291-312.

```



Chapel Quick Reference

Quick Start:

User can run a one-line "Hello world" program

1. Create the file hello.chpl:
echo "print(\"Hello, world!\")" > hello.chpl
2. Compile and run it:
\$ chpl hello.chpl
\$./hello

Comments:

- // single-line comment
- /* multi-line comment */

Primitive Types:

Type	Default size	Default name	Default unit
bool	small bool	b	bool
int	int	i	int
float	float	f	float
double	double	d	double
complex	complex	c	complex
string	string	s	string
char	char	ch	char
void	void	v	void

Variables, Constants and Configuration:

var my_int = 1234; const my_double = 3.14159;

var my_bool = false; const my_float = 0.0;

const my_double_pi = 3.141592653589793;

const my_int_pi = 3.14;

const my_double_pi_over_2 = 1.5708;

const my_double_pi_over_4 = 0.7854;

const my_int_pi_over_2 = 1;

const my_double_pi_over_4 = 0.3927;

const my_int_pi_over_8 = 0.1963;

const my_double_pi_over_8 = 0.0985;

const my_int_pi_over_16 = 0.0493;

const my_double_pi_over_16 = 0.0247;

const my_int_pi_over_32 = 0.0247;

const my_double_pi_over_32 = 0.0123;

const my_int_pi_over_64 = 0.0123;

const my_double_pi_over_64 = 0.0061;

const my_int_pi_over_128 = 0.0061;

const my_double_pi_over_128 = 0.0031;

const my_int_pi_over_256 = 0.0031;

const my_double_pi_over_256 = 0.0015;

const my_int_pi_over_512 = 0.0015;

const my_double_pi_over_512 = 0.00075;

const my_int_pi_over_1024 = 0.00075;

const my_double_pi_over_1024 = 0.000375;

const my_int_pi_over_2048 = 0.000375;

const my_double_pi_over_2048 = 0.0001875;

const my_int_pi_over_4096 = 0.0001875;

const my_double_pi_over_4096 = 0.00009375;

const my_int_pi_over_8192 = 0.00009375;

const my_double_pi_over_8192 = 0.000046875;

const my_int_pi_over_16384 = 0.000046875;

const my_double_pi_over_16384 = 0.0000234375;

const my_int_pi_over_32768 = 0.0000234375;

const my_double_pi_over_32768 = 0.00001171875;

const my_int_pi_over_65536 = 0.00001171875;

const my_double_pi_over_65536 = 0.000005859375;

const my_int_pi_over_131072 = 0.000005859375;

const my_double_pi_over_131072 = 0.0000029296875;

const my_int_pi_over_262144 = 0.0000029296875;

const my_double_pi_over_262144 = 0.00000146484375;

const my_int_pi_over_524288 = 0.00000146484375;

const my_double_pi_over_524288 = 0.000000732421875;

const my_int_pi_over_1048576 = 0.000000732421875;

const my_double_pi_over_1048576 = 0.0000003662109375;

const my_int_pi_over_2097152 = 0.0000003662109375;

const my_double_pi_over_2097152 = 0.00000018310546875;

const my_int_pi_over_4194304 = 0.00000018310546875;

const my_double_pi_over_4194304 = 0.000000091552734375;

const my_int_pi_over_8388608 = 0.000000091552734375;

const my_double_pi_over_8388608 = 0.0000000457763671875;

const my_int_pi_over_16777216 = 0.0000000457763671875;

const my_double_pi_over_16777216 = 0.00000002288818359375;

const my_int_pi_over_33554432 = 0.00000002288818359375;

const my_double_pi_over_33554432 = 0.000000011444091796875;

const my_int_pi_over_67108864 = 0.000000011444091796875;

const my_double_pi_over_67108864 = 0.0000000057220458984375;

const my_int_pi_over_134217728 = 0.0000000057220458984375;

const my_double_pi_over_134217728 = 0.00000000286102294921875;

const my_int_pi_over_268435456 = 0.00000000286102294921875;

const my_double_pi_over_268435456 = 0.000000001430511474609375;

const my_int_pi_over_536870912 = 0.000000001430511474609375;

const my_double_pi_over_536870912 = 0.0000000007152557373046875;

const my_int_pi_over_1073741824 = 0.0000000007152557373046875;

const my_double_pi_over_1073741824 = 0.00000000035762786865234375;

const my_int_pi_over_2147483648 = 0.00000000035762786865234375;

const my_double_pi_over_2147483648 = 0.000000000178813934326171875;

const my_int_pi_over_4294967296 = 0.000000000178813934326171875;

const my_double_pi_over_4294967296 = 0.0000000000894069671630859375;

const my_int_pi_over_8589934592 = 0.0000000000894069671630859375;

const my_double_pi_over_8589934592 = 0.00000000004470348358154296875;

const my_int_pi_over_17179869184 = 0.00000000004470348358154296875;

const my_double_pi_over_17179869184 = 0.000000000022351741790771484375;

const my_int_pi_over_34359738368 = 0.000000000022351741790771484375;

const my_double_pi_over_34359738368 = 0.0000000000111758708953857421875;

const my_int_pi_over_68719476736 = 0.0000000000111758708953857421875;

const my_double_pi_over_68719476736 = 0.00000000000558793544769287109375;

const my_int_pi_over_137438953472 = 0.00000000000558793544769287109375;

const my_double_pi_over_137438953472 = 0.000000000002793967723846435546875;

const my_int_pi_over_274877906944 = 0.000000000002793967723846435546875;

const my_double_pi_over_274877906944 = 0.0000000000013969838619232177234375;

const my_int_pi_over_549755813888 = 0.0000000000013969838619232177234375;

const my_double_pi_over_549755813888 = 0.00000000000069849193096160886171875;

const my_int_pi_over_1099511627776 = 0.00000000000069849193096160886171875;

const my_double_pi_over_1099511627776 = 0.0000000000003492459654808044308546875;

const my_int_pi_over_2199023255552 = 0.0000000000003492459654808044308546875;

const my_double_pi_over_2199023255552 = 0.00000000000017462298274040221542734375;

const my_int_pi_over_4398046511104 = 0.00000000000017462298274040221542734375;

const my_double_pi_over_4398046511104 = 0.0000000000000873114913702011077136875;

const my_int_pi_over_8796093022208 = 0.0000000000000873114913702011077136875;

const my_double_pi_over_8796093022208 = 0.00000000000004365574568510055385684375;

const my_int_pi_over_17592186044416 = 0.00000000000004365574568510055385684375;

const my_double_pi_over_17592186044416 = 0.000000000000021827872842550276928421875;

const my_int_pi_over_35184372088832 = 0.000000000000021827872842550276928421875;

const my_double_pi_over_35184372088832 = 0.0000000000000109139364212751384642109375;

const my_int_pi_over_70368744177664 = 0.0000000000000109139364212751384642109375;

const my_double_pi_over_70368744177664 = 0.0000000000000054569682106375692321054736875;

const my_int_pi_over_14073748835532 = 0.0000000000000054569682106375692321054736875;

const my_double_pi_over_14073748835532 = 0.00000000000000272848410531878461605273684375;

const my_int_pi_over_28147497671064 = 0.00000000000000272848410531878461605273684375;

const my_double_pi_over_28147497671064 = 0.000000000000001364242052659392308026373621875;

const my_int_pi_over_56294995342128 = 0.000000000000001364242052659392308026373621875;

const my_double_pi_over_56294995342128 = 0.0000000000000006821210263296961540131873109375;

const my_int_pi_over_112589990684256 = 0.0000000000000006821210263296961540131873109375;

const my_double_pi_over_112589990684256 = 0.0000000000000003410605131648480770065936521875;

const my_int_pi_over_225179981368512 = 0.0000000000000003410605131648480770065936521875;

const my_double_pi_over_225179981368512 = 0.0000000000000001705302565824240385032968264375;

const my_int_pi_over_450359962737024 = 0.0000000000000001705302565824240385032968264375;

const my_double_pi_over_450359962737024 = 0.00000000000000008526512829121201925164841321875;

const my_int_pi_over_900719925474048 = 0.00000000000000008526512829121201925164841321875;

const my_double_pi_over_900719925474048 = 0.000000000000000042632564145606009625824206409375;

const my_int_pi_over_1801439850948096 = 0.000000000000000042632564145606009625824206409375;

const my_double_pi_over_1801439850948096 = 0.0000000000000000213162820728030048129121032046875;

const my_int_pi_over_3602879701896192 = 0.0000000000000000213162820728030048129121032046875;

const my_double_pi_over_3602879701896192 = 0.00000000000000001065814103640150240615605160234375;

const my_int_pi_over_7205759403792384 = 0.00000000000000001065814103640150240615605160234375;

const my_double_pi_over_7205759403792384 = 0.000000000000000005329070518200751203078025801171875;

const my_int_pi_over_14411518807584768 = 0.000000000000000005329070518200751203078025801171875;

const my_double_pi_over_14411518807584768 = 0.00000000000000000266453525910037560153901290058546875;

const my_int_pi_over_28823037615169536 = 0.00000000000000000266453525910037560153901290058546875;

const my_double_pi_over_28823037615169536 = 0.000000000000000001332267629550187800769506454792890625;

const my_int_pi_over_57646075230339072 = 0.000000000000000001332267629550187800769506454792890625;

const my_double_pi_over_57646075230339072 = 0.000000000000000000666133814775093900384753227494446875;

const my_int_pi_over_115292150460678144 = 0.000000000000000000666133814775093900384753227494446875;

const my_double_pi_over_115292150460678144 = 0.0000000000000000003330669073875469501923766137472234375;

const my_int_pi_over_230584300921356288 = 0.0000000000000000003330669073875469501923766137472234375;

const my_double_pi_over_230584300921356288 = 0.00000000000000000016653345369377347509618830872361140625;

const my_int_pi_over_461168601842712576 = 0.00000000000000000016653345369377347509618830872361140625;

const my_double_pi_over_461168601842712576 = 0.000000000000000000083266726846886737548094154361805625;

const my_int_pi_over_922337203685425152 = 0.0000000000

Documentation: Then

After HPCS:

- a PDF language specification
- a Quick Reference sheet
- a number of READMEs
- ~22 primer examples

Chapel Language Specification
Version 0.93

Cray Inc
901 Fifth Avenue, Suite 1000
Seattle, WA 98164

April 18, 2013

Chapel Quick Reference

Quick Start

Use `chpl` to run the "Hello, world!" program.

1. Create the file `hello.chpl`: `echo "Hello, world!" > hello.chpl`
2. Compile and run it: `chpl hello.chpl` or `./hello.chpl`.

Expression Precedence and Associativity

Precedence	Associativity
11-12	operator precedence, left to right
10	parentheses
9-10	expressions
8	operator associativity
7	function, method, operator, apply, closure
6	loop

Statements

- `if` condition do ...
`while` condition do ...
`do` condition `while` condition
`for` counter `in` range
`break` or `break` counter
`continue` or `continue` counter
- `procedure`
`proc` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}
`function`
`func` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}

Procedures

`proc` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}
`function`
`func` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}

Normal Argument Passes

Normal Argument Passes	
value	constant
expr	variable
param	parameter
ref	pointer to variable
const	constant value or reference, cannot bind to variables modified
out	like <code>out</code> for arrays, <code>outvar</code> , <code>const</code> , <code>ref</code> , indicates the argument

Normal Formal Arguments

`proc` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}
`function`
`func` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}

Default Values for Formal Arguments

`proc` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}
`function`
`func` identifier(`arg1`, ..., `argN`) {
 `return` `value`;
}

File README Top LI (fundamental)

bradc — ssh bradc@trill.cray.com — bash

FILE EDIT OPTIONS BUFFERS TOOLS HELP

Chapel doc README

This directory contains the following documentation:

```

README           : this file
README.bugs      : how to report bugs or suggestions to the Chapel team
README.building : information about building the Chapel compiler
README.chplenv   : setting up your environment to use Chapel
README.compiling : how to use the Chapel compiler to compile code
README.executing : execution options for Chapel programs
README.multilocale : how to execute Chapel on multiple locales
README.threads   : explains how Chapel tasks are implemented using threads
README.xt-cnl    : notes for Cray XT (MIMICOS/tcl) users
README.cygwin    : notes for Cygwin users
README.extern    : technical note on interfacing with external C routines
README.format    : technical note on controlling value-to-string formatting
README.prereqs   : prerequisites for using Chapel

chapelLanguageSpec.pdf : the current draft of the Chapel language specification

hpccOverview.pdf : a high-level overview of our implementations of the HPC Challenge benchmarks for STREAM Triad, Random Access, and FFT in Chapel

hpccTutorial.pdf : a companion paper to the previous that provides a detailed walkthrough of our implementations of the HPC benchmarks to serve as a tutorial to Chapel and the codes themselves

quickReference.pdf : a one-sheet, tri-fold overview of Chapel syntax for quick reference

```

For more Information

For additional information about Chapel, please refer to:

- * "Parallel Programmability and the Chapel Language" by Bradford L. Chamberlain, David Callahan, and Hans P. Zima, published in the International Journal of High Performance Computing Applications, August 2007, 21(3): 291-312.

bradc@trill:~/chapel\$ Top LI (Chapel/5 Address)

Loading /users/bradc/chapel/highlight/emacs/22/chpl-mode.el (source)...done



COMPUTE

STORE

ANALYZE

Documentation: Now



Now: 200+ modern, hyperlinked, web-based documentation pages

The screenshot displays three interconnected web pages from the Chapel Documentation 1.16 site:

- Compiling and Running Chapel**: This page includes a sidebar with links to "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools". The main content area contains a list of links: "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools".
- Using Chapel**: This page includes a sidebar with links to "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools". The main content area contains a section titled "Contents:" followed by a list of links: "Chapel Prerequisites", "Setting up Your Environment for Chapel", "Building Chapel", "Compiling Chapel Programs", "Chapel Man Page", "Executing Chapel Programs", "Multilocale Chapel Execution", "Chapel Launchers", "Chapel Tasks", "Debugging Chapel Programs", and "Reporting Chapel Issues".
- Task Parallelism**: This page includes a sidebar with links to "Quickstart Instructions", "Using Chapel", "Platform-Specific Notes", "Technical Notes", and "Tools". The main content area contains a section titled "Begin Statements" with the text: "The `begin` statement spawns a thread of execution that is independent of the current (main) thread of execution." and code examples:

```
writeln("1. APP: The begin statement APP");  
begin writeln("2. APP: output from spawned task");
```

 It also contains a section titled "Cobegin Statements" with the text: "The main thread of execution continues on to the next statement. There is no guarantee as to which statement will execute first." and code examples:

```
writeln("1. output from main task");  
cobegin writeln("2. output from spawned task")
```



COMPUTE

STORE

ANALYZE

Libraries: Then



After HPCS: ~25 library modules

- documented via source comments, if at all:

```
brdc — ssh brdc@troll.cray.com — bash
file Edit Options Buffers Tools chpl Help
// Copyright (c) 2004-2013, Cray Inc. (See LICENSE file for more details)

// Random Module
//
// This standard module contains a random number generator based on
// the one used in the NPB benchmarks. Tailoring the NPB comments to
// this code, we can say the following:
//
// This generator returns uniform pseudorandom real values in the
// range [0, 1] by using the linear congruential generator
//
// x_{k+1} = a x_k (mod 2^{46})
//
// where 0 < x_k < 2^{46} and 0 < a < 2^{46}. This scheme generates
// 2^{44} numbers before repeating. The seed value must be an odd
// 64-bit integer in the range [1, 2^{46}). The generated values are
// normalized to be between 0 and 1, i.e., 2^{46} - x_k = x_{k+1}.
//
// This generator should produce the same results on any computer
// with at least 48 mantissa bits for real(64) data.
//
// Open Issues
//
// 1. We would like to support general serial and parallel iterators
// on the RandomStream class, but this is not possible with our
// current parallel iterator framework.
//
// 2. The random number generation functionality in this module is
// currently restricted to 64-bit real, 64-bit imag, and 128-bit
// complex values. This should be extended to other primitive types
// for which this would make sense. Coercions are insufficient.
//
// 3. Can the multiplier 'arand' be moved into the RandomStream class
// so that it can be changed by a user of this class.
//
// 4. By default, the random stream seed is initialized based on the
// current time in microseconds, allowing for some degree of
// randomness. The intent of the SeedGenerator enumerated type is to
// provide a menu of options for initializing the random stream seed,
// but only one option is implemented to date.
//
// Note on Private
//
// It is the intent that once Chapel supports the notion of 'private',
// everything prefixed with RandomPrivate_ will be made private to
// the module.
// ---F1 Random.chpl Top LI (Chapel/L Abbrev)
Mark set
```



```
brdc — ssh brdc@troll.cray.com — bash
file Edit Options Buffers Tools chpl Help
// Copyright (c) 2004-2013, Cray Inc. (See LICENSE file for more details)

extern type qio_regexp_t;
extern record qio_regexp_options_t {
    var utf8:bool;
    var posix:bool;
    var literal:bool;
    var uncapture:bool;
    // These ones can be set inside the regexp
    var ignorecase:bool; // (?i)
    var multiline:bool; // (?m)
    var dotnl:bool; // (?s)
    var nongreedy:bool; // (?U)
}

extern proc qio_regexp_null():qio_regexp_t;
extern proc qio_regexp_init_default_options(ref options:qio_regexp_options_t);
extern proc qio_regexp_create_compile(str:string, strlen:int[64], ref options:qio_regexp_options_t, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags(str:string, strlen:int[64], flags:int[64], isutf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_create_compile_flags_2(str:c_ptr, strlen:int[64], flags:int[64], isutf8:bool, ref compiled:qio_regexp_t);
extern proc qio_regexp_retain(ref compiled:qio_regexp_t);
extern proc qio_regexp_release(ref compiled:qio_regexp_t);

extern proc qio_regexp_get_options(ref regexp:qio_regexp_t, ref options: qio_regexp_options_t);
extern proc qio_regexp_get_pattern(ref regexp:qio_regexp_t, ref pattern: string);
extern proc qio_regexp_get_ncaptures(ref regexp:qio_regexp_t):int[64];
extern proc qio_regexp_anchor_start(ref regexp:qio_regexp_t):int;
extern proc qio_regexp_anchor_both(ref regexp:qio_regexp_t):int;

extern const QIO_REGEXP_ANCHOR_UNANCHORED:c_int;
extern const QIO_REGEXP_ANCHOR_START:c_int;
extern const QIO_REGEXP_ANCHOR_BOTH:c_int;

extern record qio_regexp_string_piece_t {
    var offset:int[64]; // counting from 0, -1 means "NULL"
    var len:int[64];
}

extern proc qio_regexp_string_piece_isnull(ref sp:qio_regexp_string_piece_t):bool;
// ---F1 Regexp.chpl Top LI (Chapel/L Abbrev)
```



COMPUTE

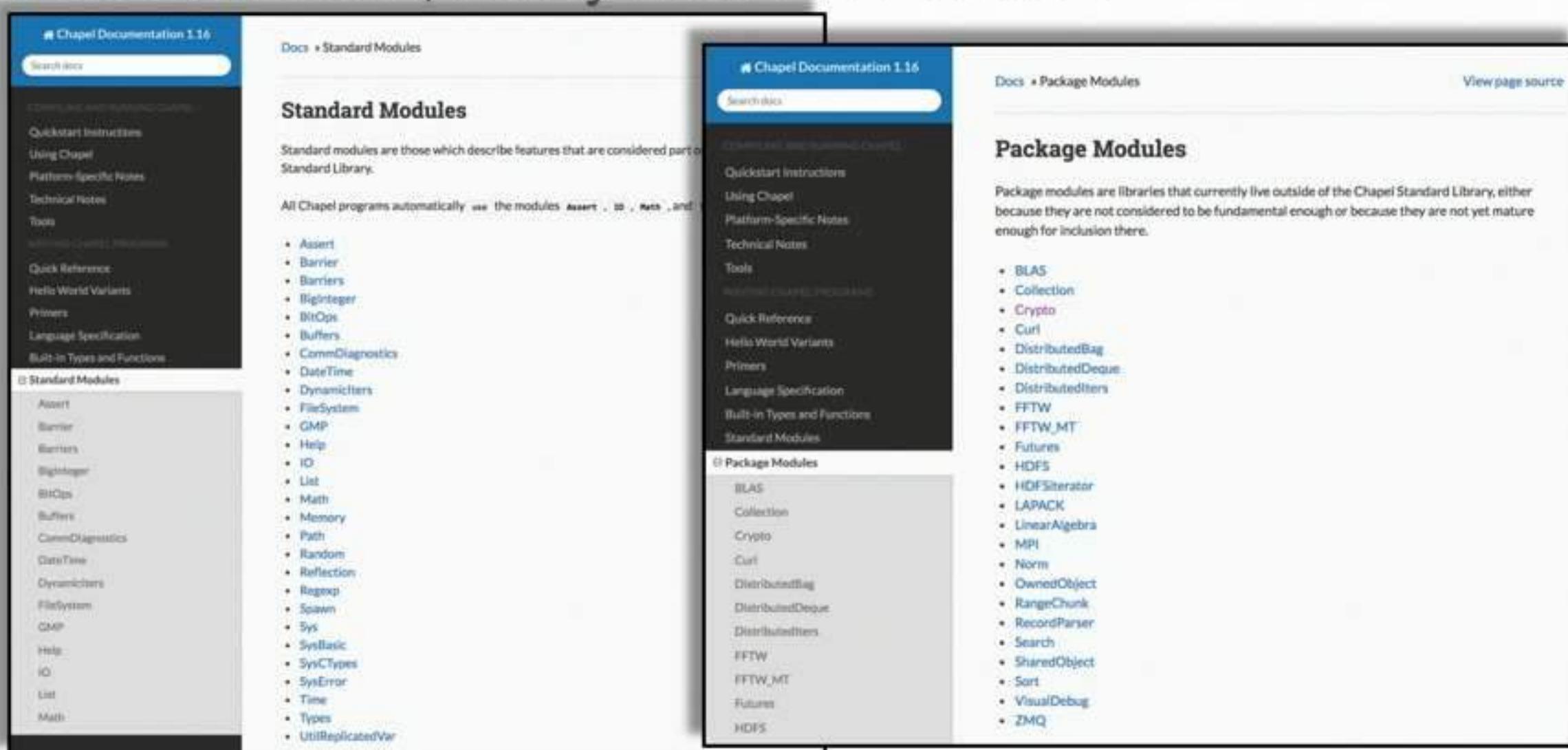
STORE

ANALYZE

Libraries: Now

Now: ~60 library modules

- web-documented, many user-contributed



The screenshot displays two pages from the Chapel Documentation 1.16 website:

- Standard Modules**: A list of 40 standard modules including Assert, Barrier, Barriers, BigInteger, BitOps, Buffers, CommDiagnostics, DateTime, DynamicIterators, FileSystem, GMP, Help, IO, List, Math, Memory, Path, Random, Reflection, Regexp, Spawn, Sys, SysBasic, SysCTypes, SysError, Time, Types, and UtilReplicatedVar.
- Package Modules**: A list of 30 package modules including BLAS, Collection, Crypto, Curl, DistributedBag, DistributedDeque, DistributedIterators, FFTW, FFTW_MT, Futures, HDFS, HDFSIterator, LAPACK, LinearAlgebra, MPI, Norm, OwnedObject, RangeChunk, RecordParser, Search, SharedObject, Sort, VisualDebug, and ZMQ.



COMPUTE

STORE

ANALYZE

After HPCS:

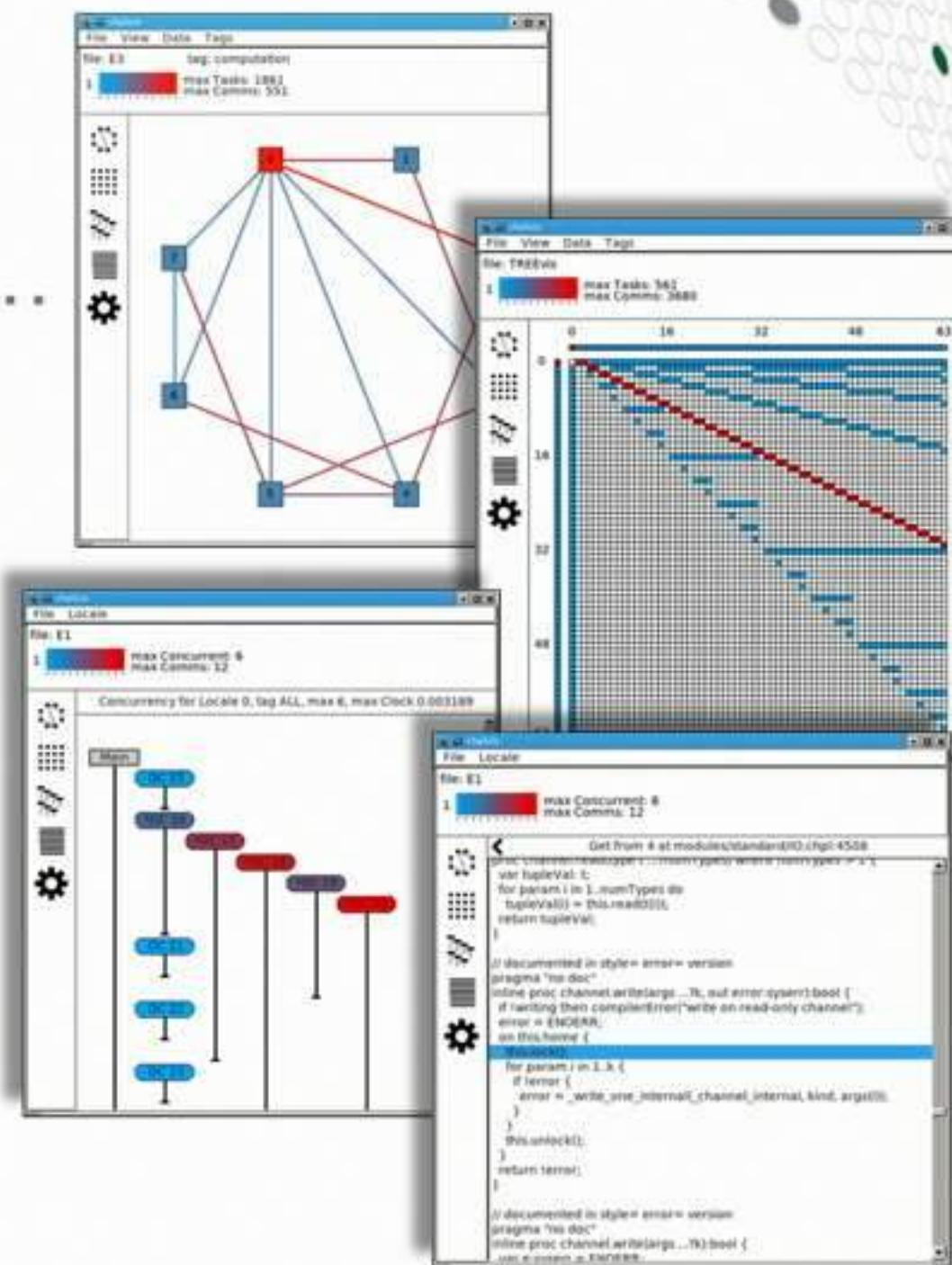
- **highlighting modes** for emacs and vim
- **chpldoc**: documentation tool (rough draft)



Tools: Now

Now:

- **highlighting modes** for emacs, vim, atom, ...
- **chpldoc**: documentation tool
- **mason**: package manager
- **c2chapel**: interoperability aid
- **chpltags**: helps search Chapel code
- **bash tab completion**: command-line help
- **chplvis**: performance visualizer / debugger



Chapel Performance: Then vs. Now



COMPUTE

STORE

ANALYZE

Performance Focus Areas during 5-year push



- Cleaner, simpler generated code
- NUMA sensitivity within multi-socket nodes
- Best-use of RDMA and NIC memory registration
- Reduced overheads in tasks, memory, communication
- Bulk transfer optimizations
- ...and much more...

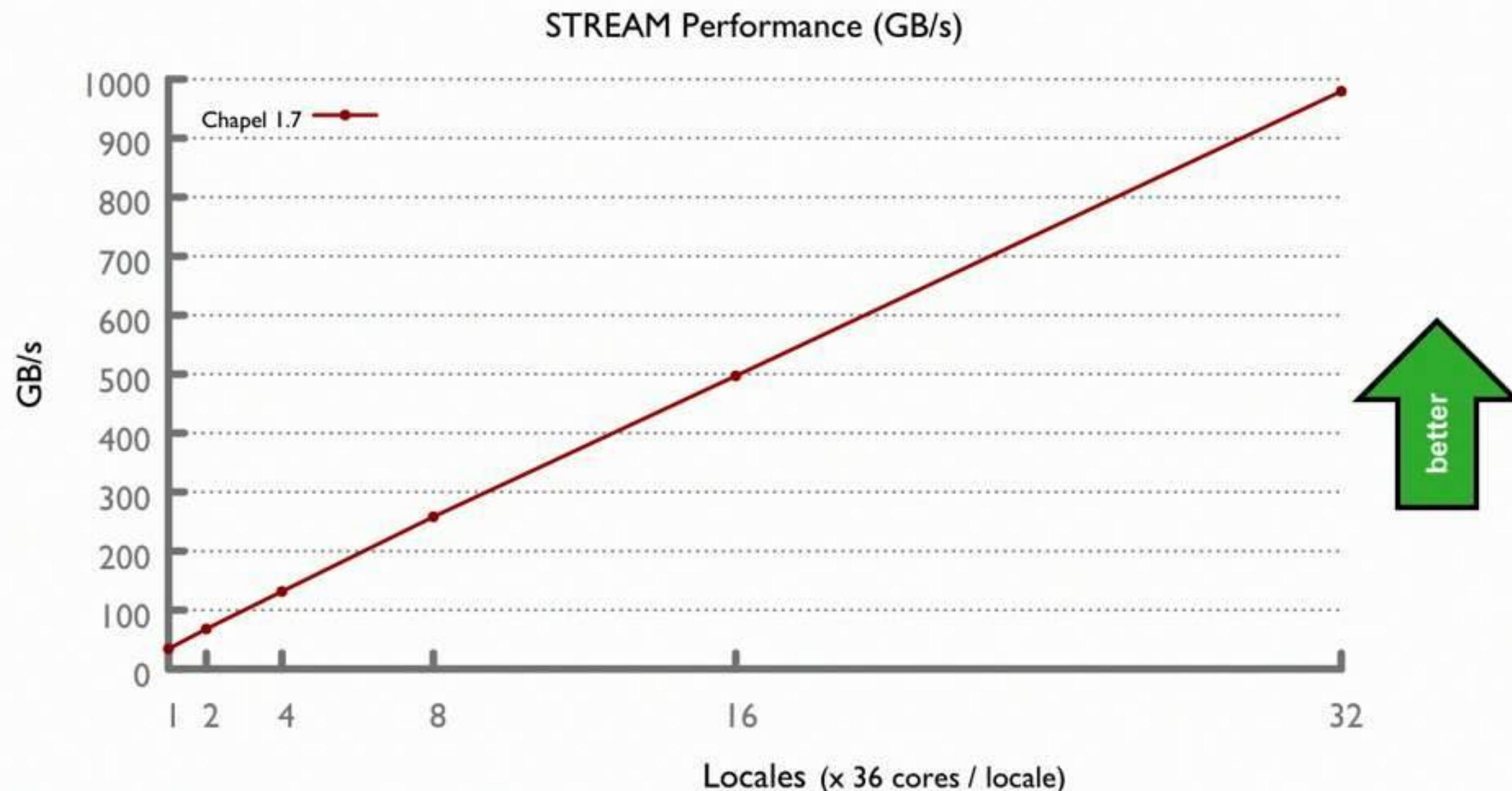


COMPUTE

STORE

ANALYZE

STREAM Triad Performance: Chapel Then

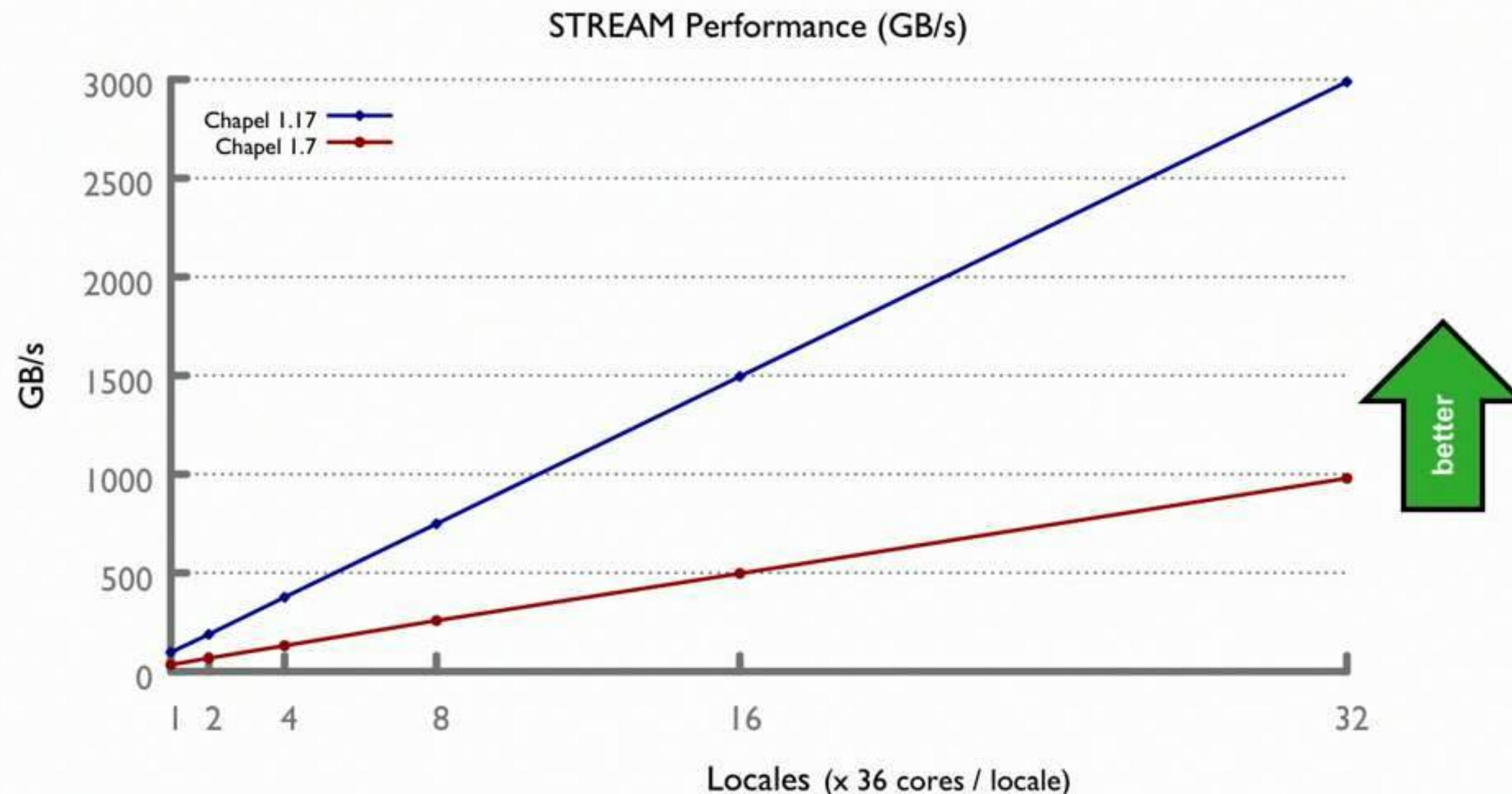


COMPUTE

STORE

ANALYZE

STREAM Triad Performance: Chapel Then vs. Now

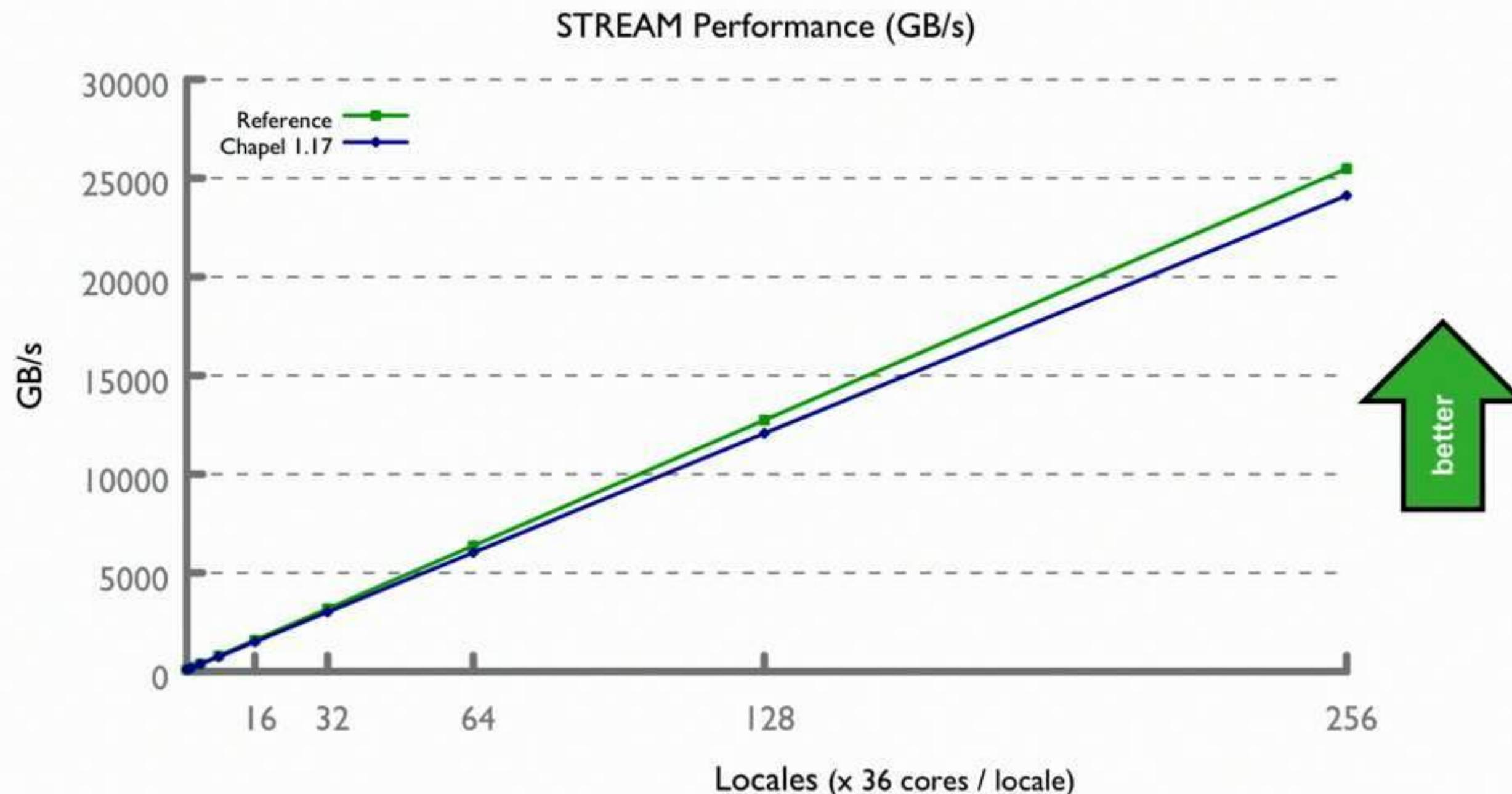


COMPUTE

STORE

ANALYZE

STREAM Triad Performance: Chapel Now vs. ref

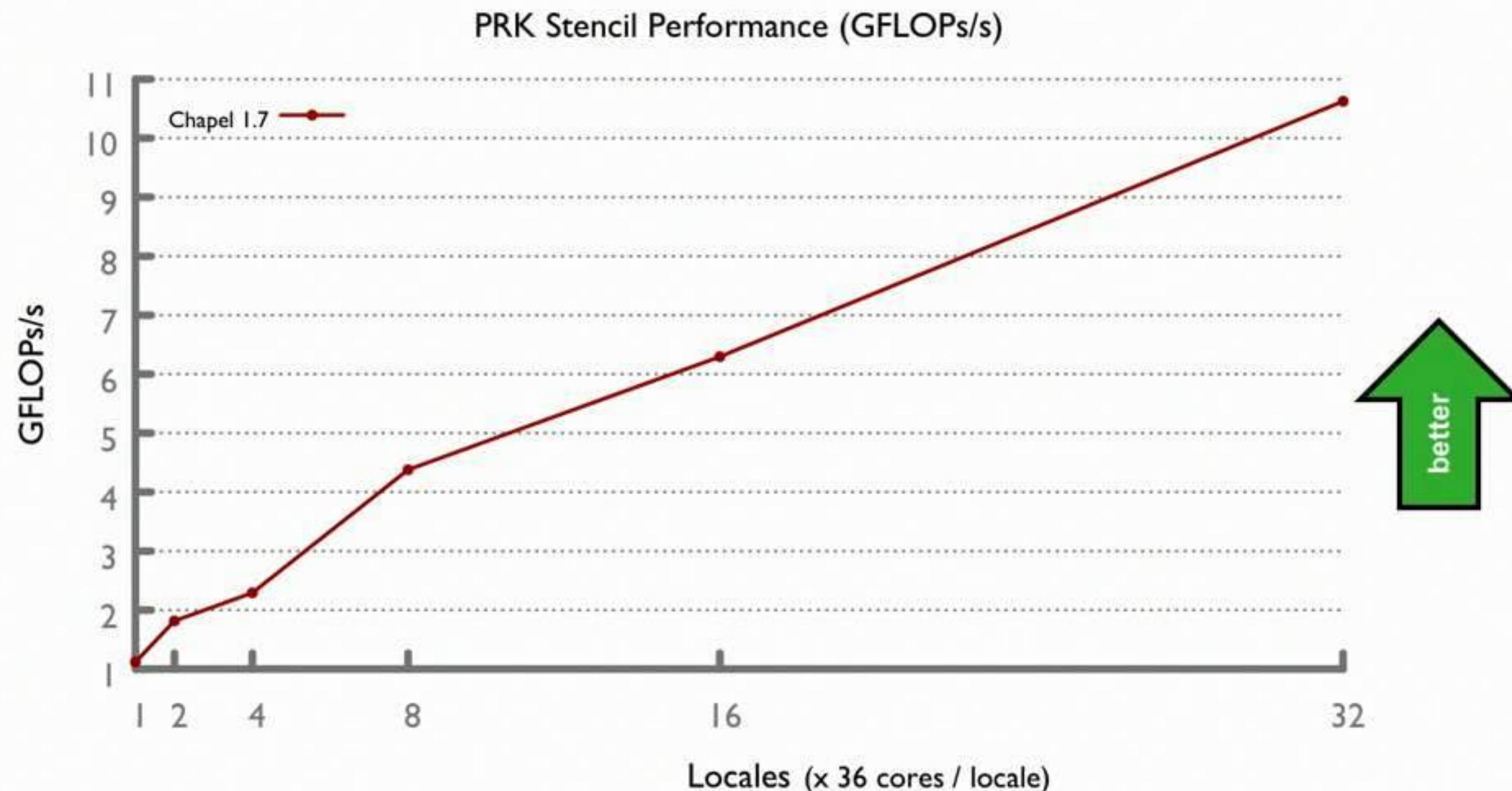


COMPUTE

STORE

ANALYZE

PRK Stencil Performance: Chapel Then

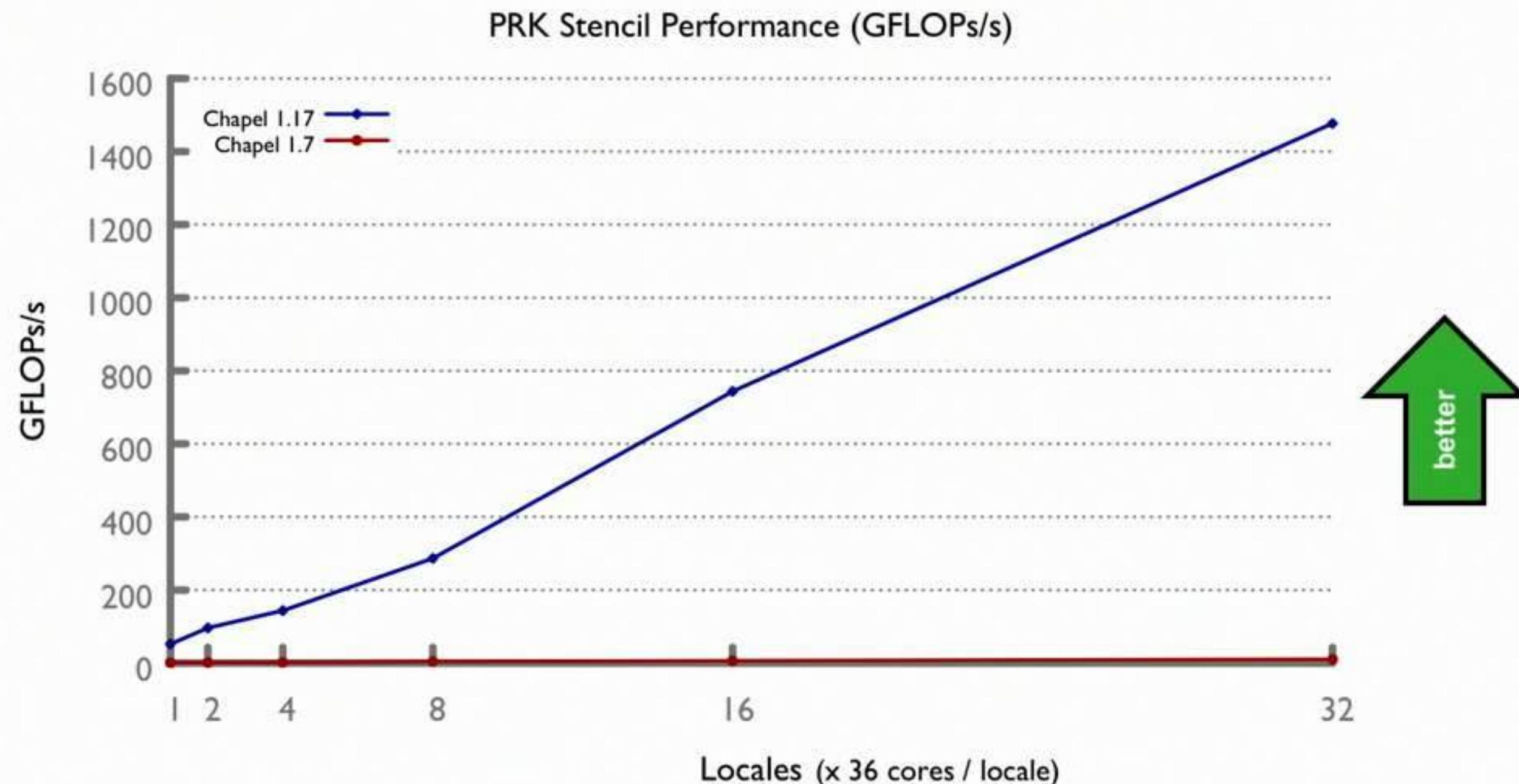


COMPUTE

STORE

ANALYZE

PRK Stencil Performance: Chapel Then vs. Now

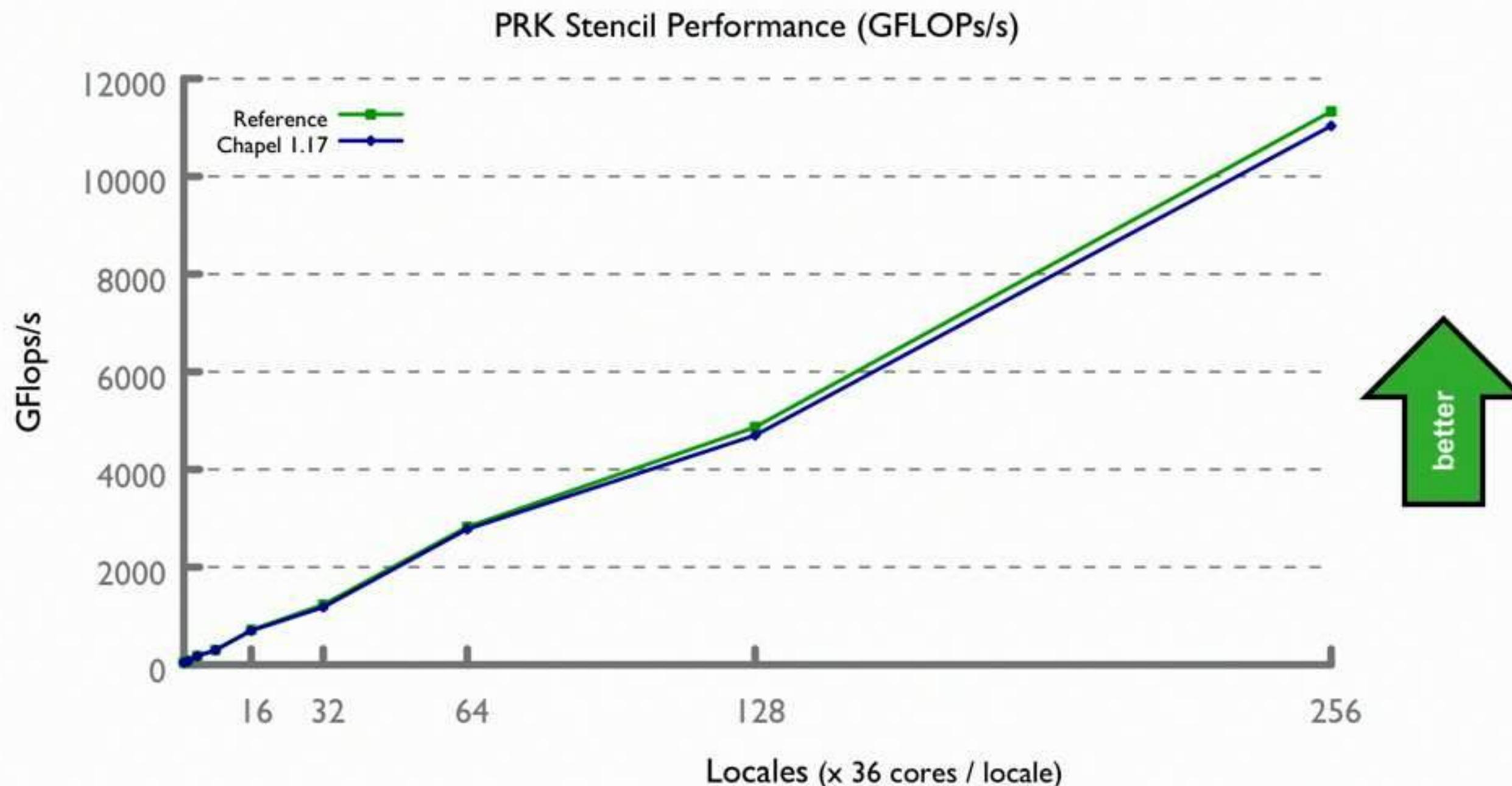


COMPUTE

STORE

ANALYZE

PRK Stencil Performance: Chapel Now vs. ref

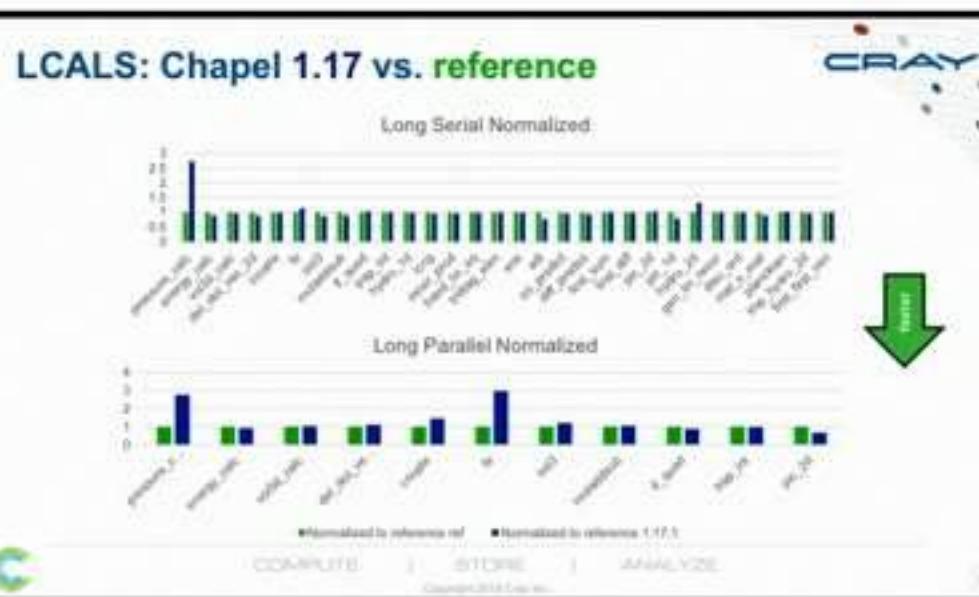


COMPUTE

STORE

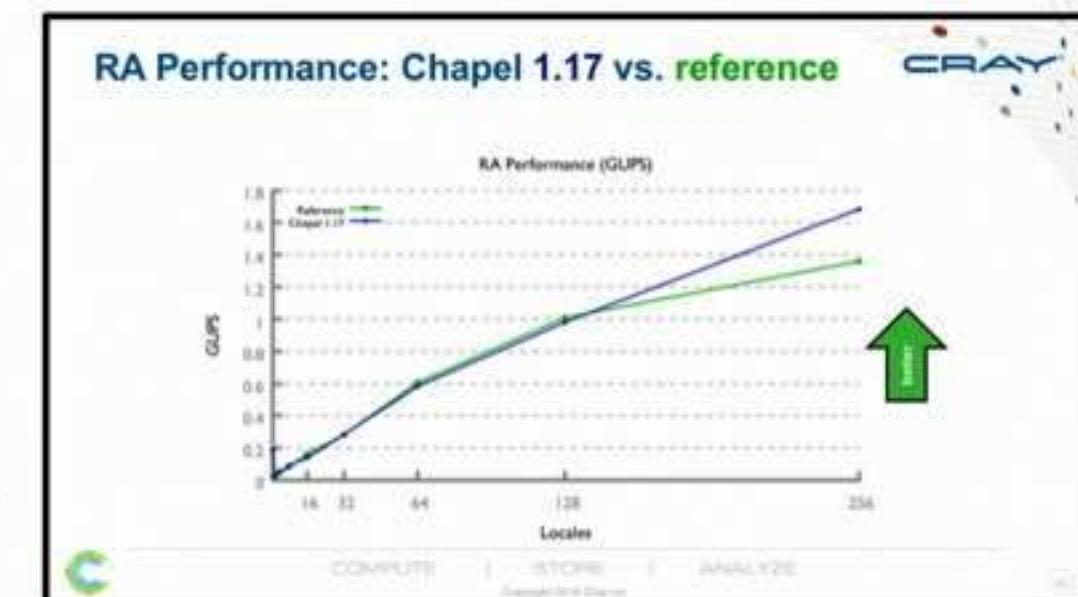
ANALYZE

HPC Patterns: Chapel Now vs. reference



LCALS

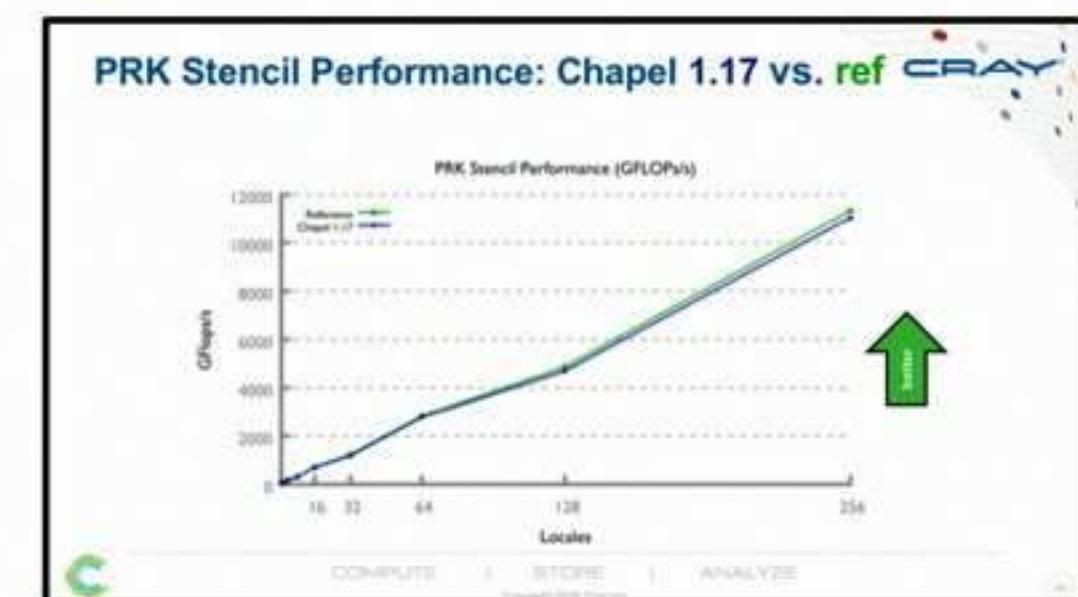
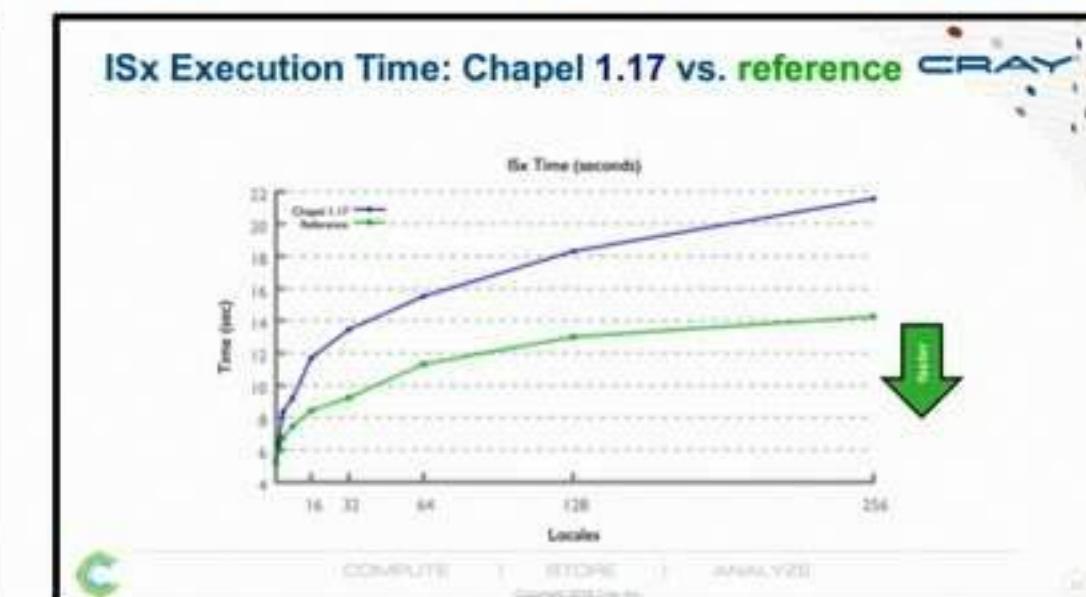
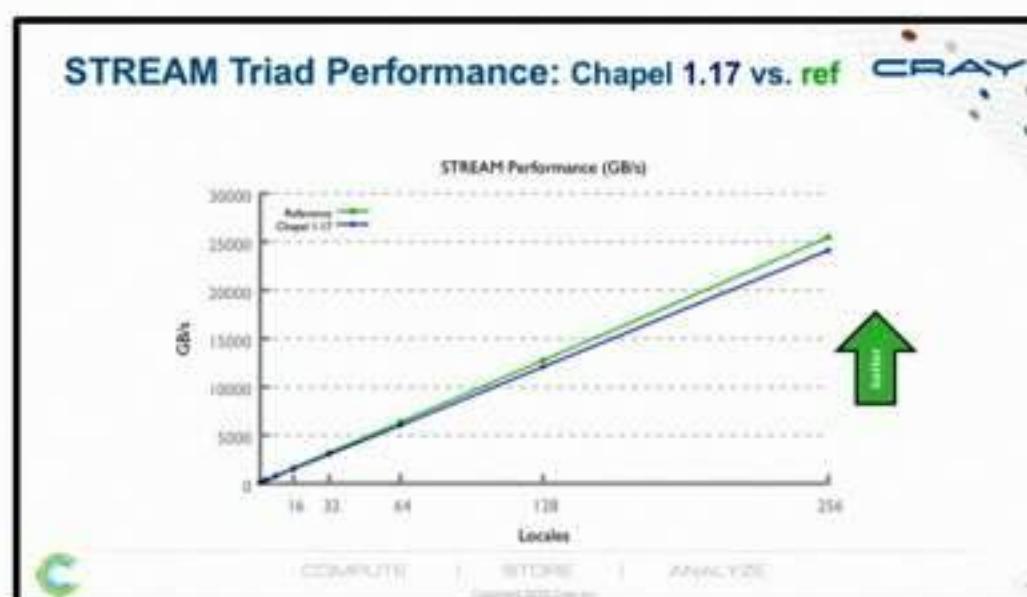
HPCC RA



STREAM
Triad

ISx

PRK
Stencil



COMPUTE

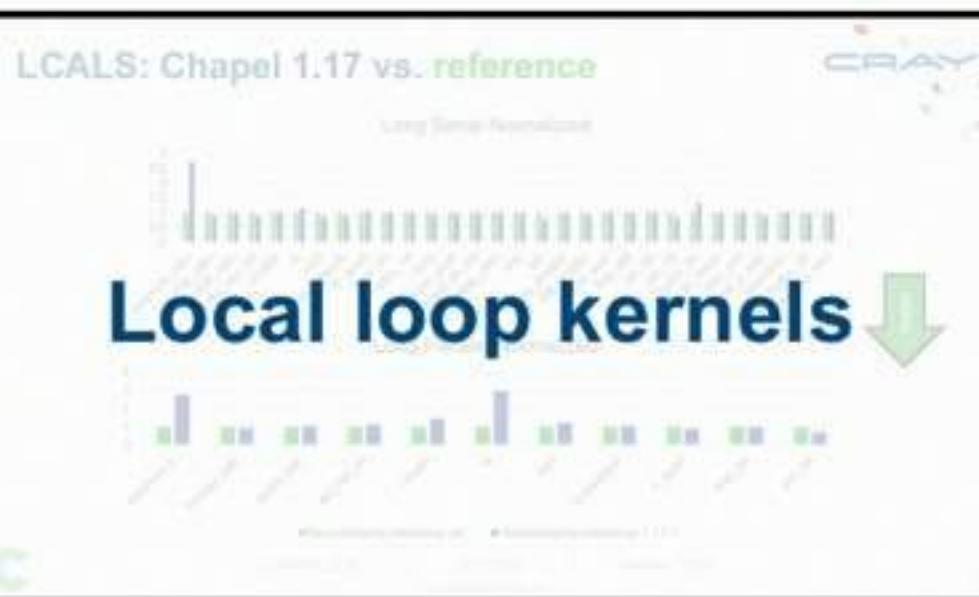
STORE

ANALYZE

Copyright 2018 Cray Inc.

Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPC Patterns: Chapel Now vs. reference



LCALS

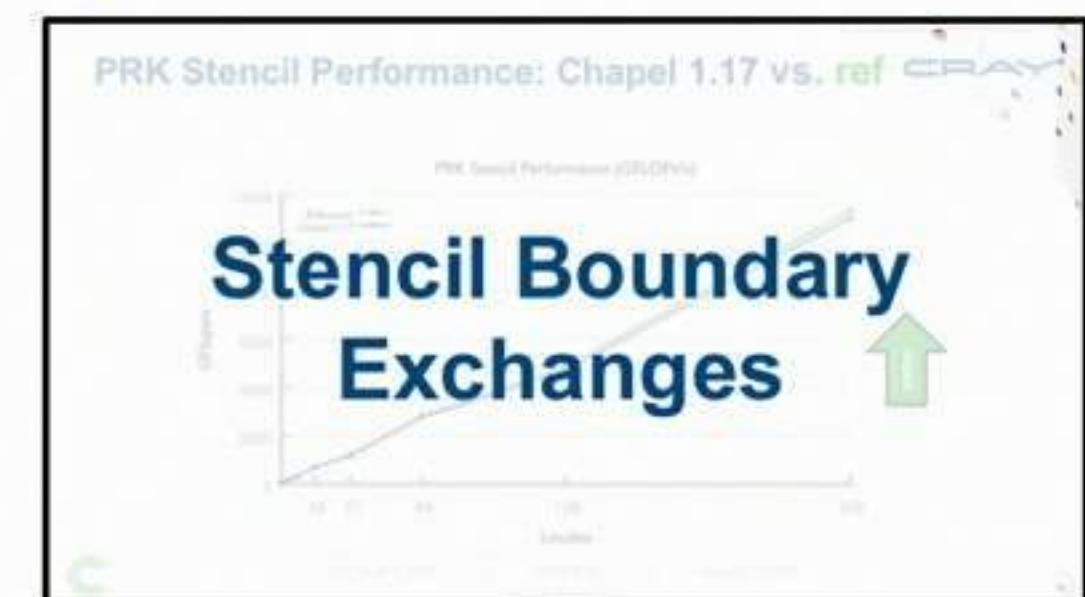
HPCC RA



STREAM
Triad

ISx

PRK
Stencil



COMPUTE

STORE

ANALY

Copyright 2018 Cray Inc.

Nightly performance tickers online at:
<https://chapel-lang.org/perf-nightly.html>

HPCC Random Access Kernel: MPI



```
/* Perform updates to main table. The scalar equivalent is:
 *
 * for(i=0; i<NUPDATE; i++) {
 *   Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *   Table[Ran & (TABSIZEx1)] ^= Ran;
 * }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
while (i < SendCnt) {
    /* receive messages */
    do {
        MPI_Test(&inreq, &have_done, &status);
        if (have_done) {
            if (status.MPI_TAG == UPDATE_TAG) {
                MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                for (j=0; j < recvUpdates; j++) {
                    inmsg = LocalRecvBuffer[bufferBase+j];
                    LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                                  tparams.GlobalStartMyProc;
                    HPCC_Table[LocalOffset] ^= inmsg;
                }
            } else if (status.MPI_TAG == FINISHED_TAG) {
                NumberReceiving--;
            } else {
                MPI_Abort( MPI_COMM_WORLD, -1 );
                MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
            }
        }
    } while (have_done && NumberReceiving > 0);
if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
    GlobalOffset = Ran & (tparams.TableSize-1);
    if (GlobalOffset < tparams.Top)
        WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
    else
        WhichPe = ( (GlobalOffset - tparams.Remainder) /
                     tparams.MinLocalTableSize );
    if (WhichPe == tparams.MyProc) {
        LocalOffset = (Ran & (tparams.TableSize - 1)) -
                      tparams.GlobalStartMyProc;
        HPCC_Table[LocalOffset] ^= Ran;
    }
} else {
    HPCC_InsertUpdate(Ran, WhichPe, Buckets);
    pendingUpdates++;
}
i++;
}

else {
    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    if (have_done) {
        outreq = MPI_REQUEST_NULL;
        pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                             &peUpdates);
        MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                  UPDATE_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= peUpdates;
    }
}

/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
    if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                         MPI_REQUEST_NULL; continue; }

    /* send garbage - who cares, no one will look at it */
    MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
              MPI_COMM_WORLD, tparams.finish_req + proc_count);
}

/* Finish everyone else up... */
while (NumberReceiving > 0) {
    MPI_Wait(&inreq, &status);
    if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
            inmsg = LocalRecvBuffer[bufferBase+j];
            LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                          tparams.GlobalStartMyProc;
            HPCC_Table[LocalOffset] ^= inmsg;
        }
    } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message. Thanks for playing.. */
        NumberReceiving--;
    } else {
        MPI_Abort( MPI_COMM_WORLD, -1 );
    }
}

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);

while (have_done && NumberReceiving > 0);

MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```



COMPUTE

STORE

ANALYZE

HPCC Random Access Kernel: MPI



/ Perform updates to main table. The scalar equivalent is:*

```
* for (i=0; i<NUPDATE; i++) {  
*     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
*     Table[Ran & (TABSIZ-1)] ^= Ran;  
* }
```

For more information, see [http://www.hpmc.org/HPCC/RandomAccessKernel.html](#).
The code is available at [https://github.com/HPCCSystems/HPCC-System/tree/master/HPCC-System/RandomAccessKernel](#).

/* Perform updates to main table. The scalar equivalent is:
*
* **for** (i=0; i<NUPDATE; i++) {
* Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
* Table[Ran & (TABSIZ-1)] ^= Ran;
* }
* /

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:  
*  
*     for (i=0; i<NUPDATE; i++) {  
*         Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
*         Table[Ran & (TABSIZ-1)] ^= Ran;  
*     }  
* /
```

HPCC InsertUpdateTable(HPCCTable, TableID);
 pendingUpdates++;
 if (pendingUpdates >= MAX_PENDING_UPDATES) {
 /* If we have pending updates, then we need to flush them.
 * We do this by sending a message to the master node.
 * This message contains the pending updates and their counts.
 * The master node then processes these updates and sends back
 * an acknowledgement message.
 * Once all pending updates have been processed, we can
 * resume normal operations.
 * Note that this is a blocking call, so we will only return
 * once all pending updates have been processed.



COMPUTE

STORE

ANALYZE

HPCC Random Access Kernel: MPI



/* Perform updates to main table. The scalar equivalent is:

```
* for (i=0; i<NUPDATE; i++) {  
*   Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
*   Table[Ran & (TABSIZ-1)] ^= Ran;  
* }
```

Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do  
    T[r & indexMask] ^= r;
```

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:  
*  
*  for (i=0; i<NUPDATE; i++) {  
*      Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);  
*      Table[Ran & (TABSIZ-1)] ^= Ran;  
*  }  
*/
```

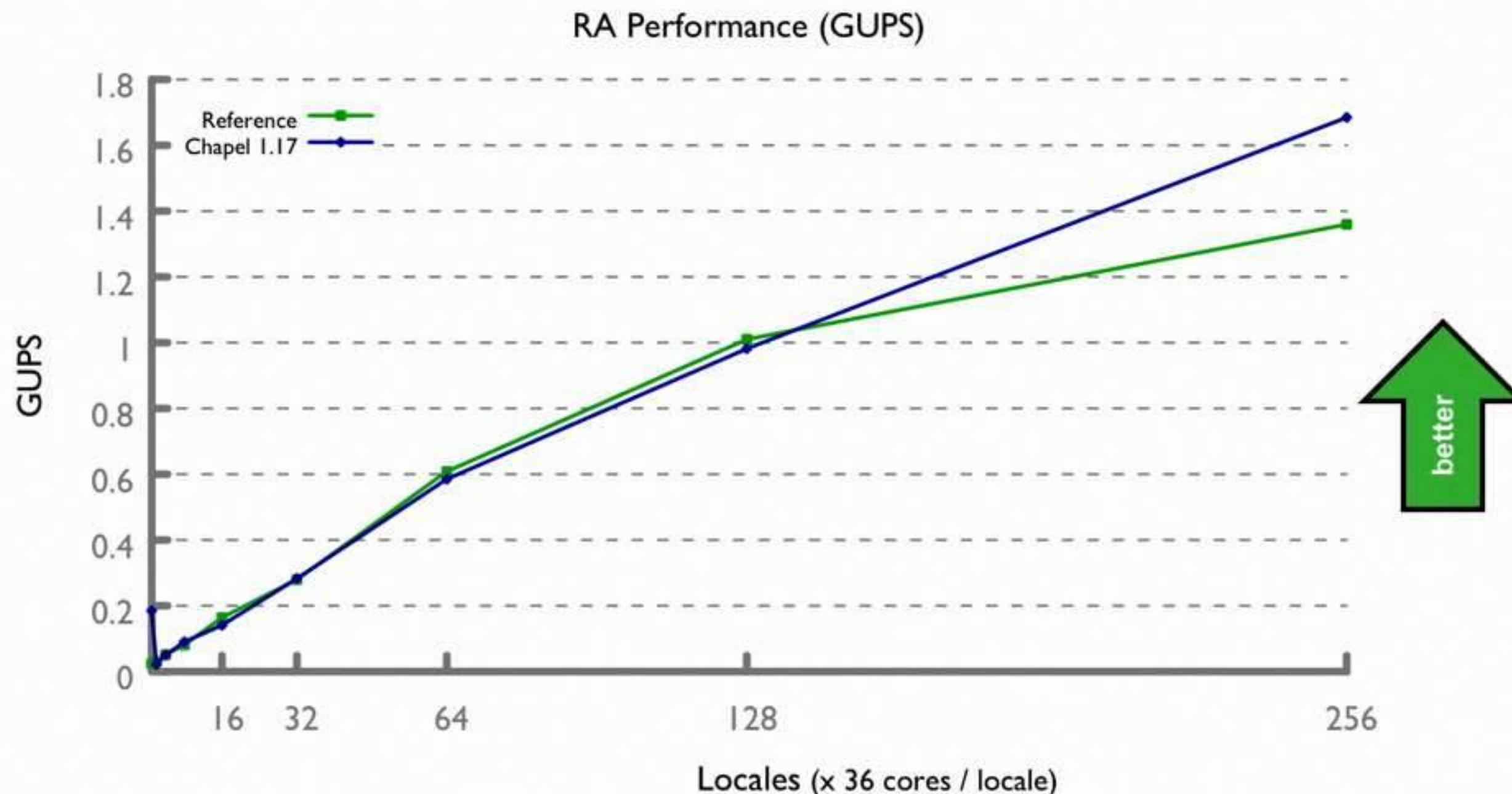


COMPUTE

STORE

ANALYZE

RA Performance: Chapel Now vs. reference



COMPUTE

STORE

ANALYZE

What's Next?



CHIUW 2018: The 5th annual Chapel Implementers and Users Workshop

- Vancouver BC, Friday May 25th
- Details: <https://chapel-lang.org/CHIUW2018.html>



COMPUTE

STORE

ANALYZE

What's Next?



CHIUW 2018: The 5th annual Chapel Implementers and Users Workshop

- Vancouver BC, Friday May 25th
- Details: <https://chapel-lang.org/CHIUW2018.html>



Chapel's college years: plans for 2018-2021

- Further Performance and Scalability Improvements
- Libfabric/OFI Support
- GPU Support
- Cloud Support
- Chapel AI



COMPUTE

STORE

ANALYZE

The Chapel Team at Cray (May 2018)



COMPUTE

STORE

ANALYZE

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

<https://chapel-lang.org/collaborations.html>



COMPUTE

STORE

ANALYZE

Summary

- Chapel's made huge progress over the past five years
- Ready for use in production*
- Open to collaborations
 - Plenty of research questions remain



Musical Ornaments

John Leo

Halfaya Research

May 14, 2018

Outline

- A look toward the future
- Music Tools
- Equivalences and Ornaments

Sources:

- <https://github.com/halfaya/MusicTools>

Robert Harper

Eventually all the arbitrary programming languages are going to be just swept away with the oceans, and we will have the permanence of constructive, intuitionistic type theory as the master theory of computation—without doubt, in my mind, no question. So, from my point of view—this is a personal statement—working in anything else is a waste of time.

CMU Homotopy Type Theory lecture 1, 52:56–53:20.

What will programming look like in 50 years?

- Convergence of math and computer science
- Functional Programming, Algebra of Programming
- Dependent Types or a successor (Cubical?)
- Who does the programming?

How do we get there from here?

- Add dependent types to an industrial-strength language (Haskell)
- Make a dependently typed language (Agda, Idris) practical to use
- Learn how to program using dependent types
- Many theoretical and practical advances are still needed

Euterpea

The Haskell School of Music

— From Signals to Symphonies —



Paul Hudak

Yale University
Department of Computer Science

Music Tools

- Collection of composable tools for synthesis and analysis of music
- Originally written in Haskell
- Converted to Agda, using Haskell for MIDI interface
- Explore programming using dependent types in a circumscribed yet rich domain
- Use math, including transport of equivalences (from HoTT) and Ornaments

Look vs Time (1997)

A musical score consisting of four staves. The top staff is in treble clef, 4/4 time, featuring eighth-note patterns. The second staff is also in treble clef, 4/4 time, with sixteenth-note patterns. The third staff is in bass clef, 4/4 time, with eighth-note patterns. The bottom staff has a square symbol and 4/4 time, containing a rhythmic pattern of vertical dashes and 'x' marks.

Logic Pro X File Edit Track Navigate Record Mix View Window 1 Help

Untitled - Tracks

1 1 84 4/4 Cmaj

Filter: All Instruments

Region: Default

Style: 0

Quantize: Default

Interpretation: ✓

Syncopation:

No Overlap: ✓

Max. Dots: 1

Event: Insert Defaults

MIDI Channel: 1

Velocity: 0

Text: Plain Text

Lyric:

Part Box

Piano Roll Score Step Editor

Layout Edit Functions View

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Accomp Marimba Bass (+12) Drums

Accomp Bass (+12)

Fingerstyle Bass | Cmaj

SoCal | Cmaj

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Score

1 2 3 4 5 6

SoCal Settings EQ

MIDI FX

Drum Kit

Channel EQ Compressor

Solo

Stereo Out

Group

Read

0.0 0.0

Bounce

M S M

SoCal Stereo Out

The screenshot displays the Logic Pro X interface with several key components visible:

- Top Bar:** Standard Mac OS X application menu bar with options like Logic Pro X, File, Edit, Track, Navigate, Record, Mix, View, Window, 1, and Help.
- Title Bar:** "Untitled - Tracks" and transport controls (play/pause, stop, record, etc.) along with tempo (84 BPM), time signature (4/4), and key (Cmaj).
- Left Sidebar:** "Part Box" containing settings for the selected instrument (SoCal). It includes sections for "Settings", "EQ", "MIDI FX", "Drum Kit", "Channel EQ/Compressor", "Solo", "Stereo Out", "Group", and "Read". Below these are two large knobs for "0.0" and "0.0" and a "Bounce" button. At the bottom are "M S" and "M" buttons, and "SoCal" and "Stereo Out" labels.
- Middle Section:** A multi-track piano roll editor showing tracks for "Accomp", "Marimba", "Fingerstyle Bass (+12)", and "Drums". Each track has its own color-coded lane and controls for M (Mute), S (Solo), and R (Record).
- Bottom Section:** A score editor with four staves of musical notation. The staves are labeled 1 through 6. The first staff is treble clef, 4/4 time. The second staff is bass clef, 4/4 time. The third staff is bass clef, 4/4 time. The fourth staff is bass clef, 4/4 time.

Look vs Time (1997)

A musical score for four staves, each in 4/4 time. The top staff is treble clef, the second is bass clef, the third is bass clef, and the bottom staff is a snare drum symbol. The music consists of eighth and sixteenth note patterns.

The first staff (treble) has a pattern of eighth notes and sixteenth-note pairs. The second staff (bass) has a continuous eighth-note bass line. The third staff (bass) has a pattern of eighth notes and sixteenth-note pairs. The bottom staff (snare drum) has a pattern of eighth-note pairs and sixteenth-note pairs.

LookVsTime.agda

```
14 open import MidiEvent
15 open import Util
16
17 tempo : ℕ
18 tempo = 84
19
20 -----
21
22 melodyChannel : Channel-1
23 melodyChannel = # 0
24
25 melodyInstrument : InstrumentNumber-1
26 melodyInstrument = # 8 -- celesta
27
28 melodyNotes : List Note
29 melodyNotes =
30   note (8th 3) (c 3) :: 
31   note (8th 5) (d 3) :: 
32   [REDACTED]
33   note (8th 3) (c 3) :: 
34   note (8th 5) (d 3) :: 
35
36   note (8th 1) (g 3) :: 
37   note (8th 1) (f 3) :: 
38   note (8th 1) (e 3) :: 
39   note (8th 5) (d 3) :: 
40
41   note (8th 1) (g 3) :: 
42   note (8th 1) (f 3) :: 
43   note (8th 1) (e 3) :: 
44   note (8th 5) (d 3) :: 
45
46   note (8th 1) (a 3) ..
```

Music Representation à la Euterpea

```
data Pitch : Set where  
  pitch : ℤ → Pitch
```

```
data Duration : Set where  
  duration : ℕ → Duration
```

```
data Note : Set where  
  note : Duration → Pitch → Note  
  rest : Duration → Note
```

```
data Music : Set where  
  note : Note → Music  
  _∷_ : Music → Music → Music --- sequential  
  _||_ : Music → Music → Music --- parallel
```

Equivalent Representations of Pitch

```
data Pitch : Set where  
  pitch :  $\mathbb{Z} \rightarrow$  Pitch
```

```
chromaticScaleSize :  $\mathbb{N}$   
chromaticScaleSize = 12
```

```
data RelativePitch : Set where  
  relativePitch : Fin chromaticScaleSize  $\rightarrow$  RelativePitch
```

```
data Octave : Set where  
  octave :  $\mathbb{Z} \rightarrow$  Octave
```

```
PitchOctave : Set  
PitchOctave = RelativePitch  $\times$  Octave
```

Equivalences

- Define an equivalence between Pitch and PitchOctave
- Using HoTT techniques, automatically lift this equivalence to functions defined using Pitch
- See *Equivalences for Free!* (Tabareau, Tanter, Sozeau)
- Challenge: Defining base equivalences. Can this be automated?

Ornaments

```
data Music a = ...
```

```
  | Modify Control (Music a)
```

```
data Control = ...
```

```
  | Phrase [PhraseAttribute]
```

```
data PhraseAttribute = ...
```

```
  | Orn Ornament
```

```
data Ornament =
```

```
  Trill | Mordent | InvMordent | DoubleMordent |
```

```
  Turn | TrilledTurn | ShortTrill ...
```

Ornaments

- Functions on a base `Music` structure can be automatically lifted to operate on `Music` ornamented with additional information
- See works on Ornaments by McBride, Dagand and others
- Challenge: Shallow embedding of ornaments in Agda

Conclusion

- Music is a good domain in which to explore practical application of dependent types
- Using math can be more work at first, but should be a big win in the long term
- Figure out how to minimize the work and maximize the reward