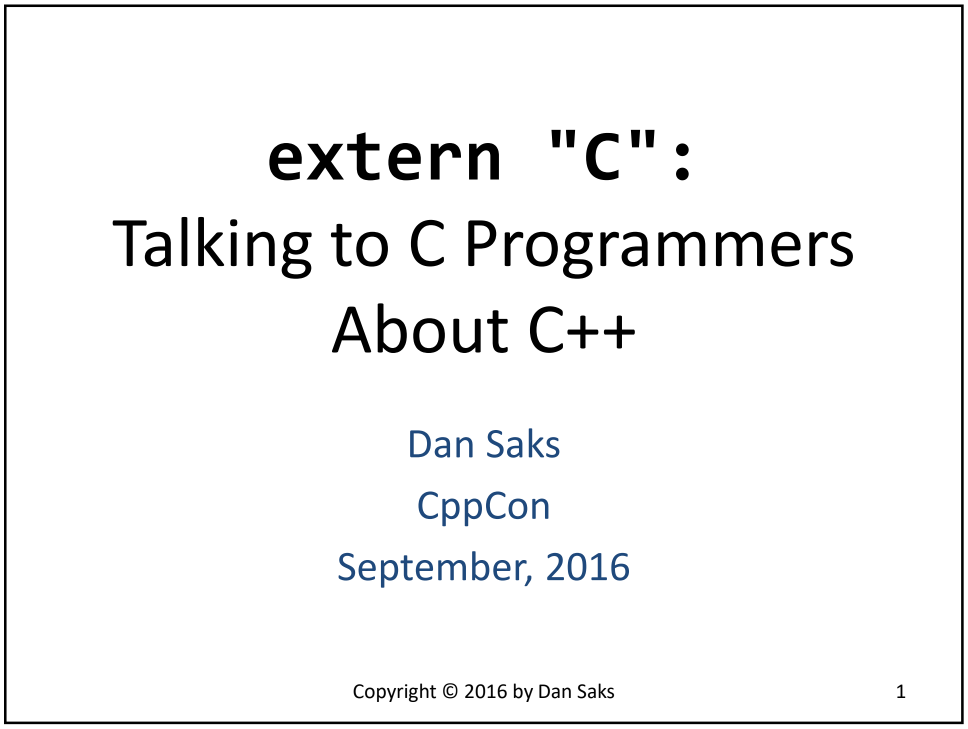


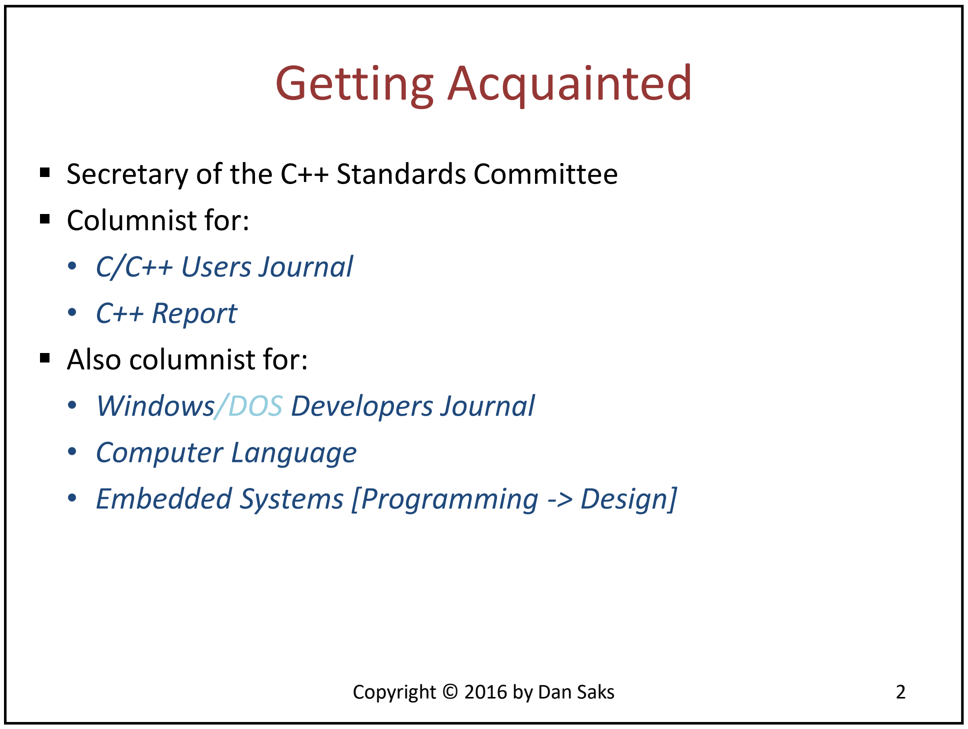
`extern "C":` Talking to C Programmers About C++

Dan Saks

CppCon

September, 2016

1



Getting Acquainted

- Secretary of the C++ Standards Committee
- Columnist for:
  - *C/C++ Users Journal*
  - *C++ Report*
- Also columnist for:
  - *Windows/DOS Developers Journal*
  - *Computer Language*
  - *Embedded Systems [Programming -> Design]*

2

If You Remember Just One Thing…

- When it comes to persuasion…
- “If you’re arguing, you’re losing.”
  —Mike Thomas

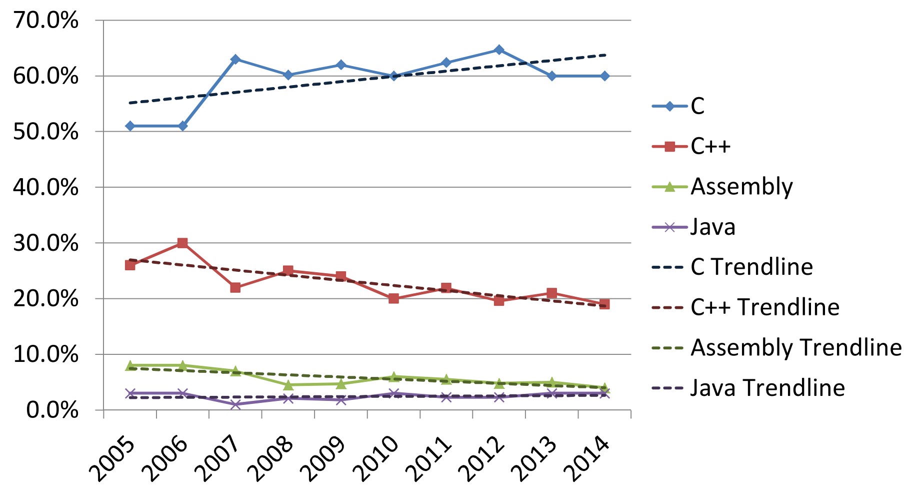
Languages for Embedded Software

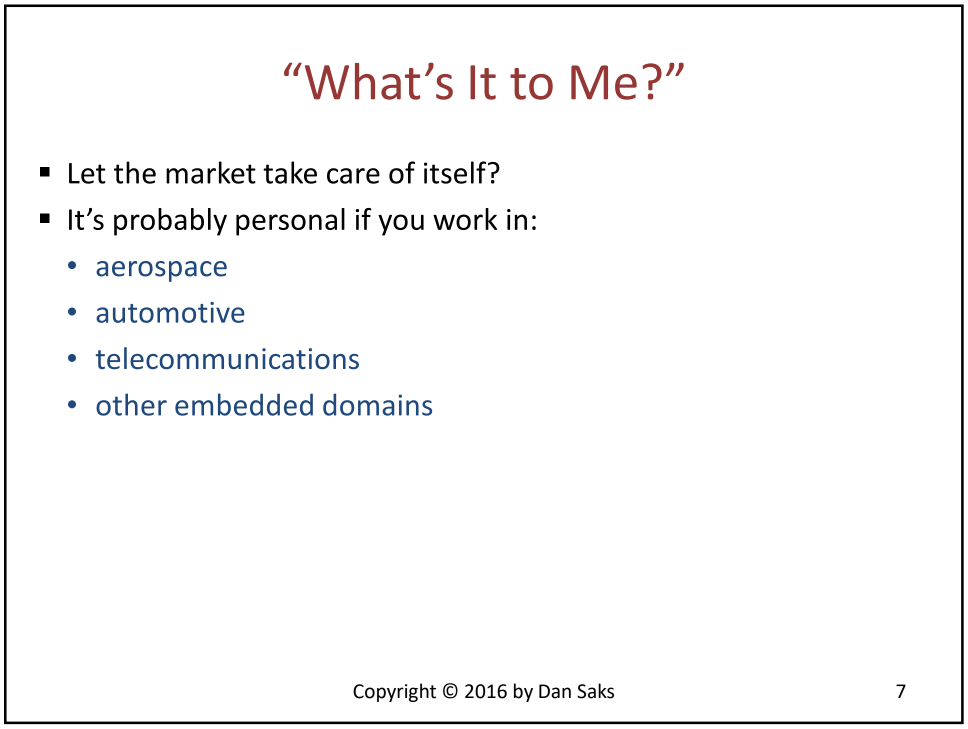
- in the beginning
  - Assembler
- late 1970s
  - C
- mid 1990s
  - C++

Languages for Embedded Software

- *embedded.com* annual reader survey question…
- Complete this sentence: ***"My current embedded project is programmed mostly in…"***

Languages for Embedded Software

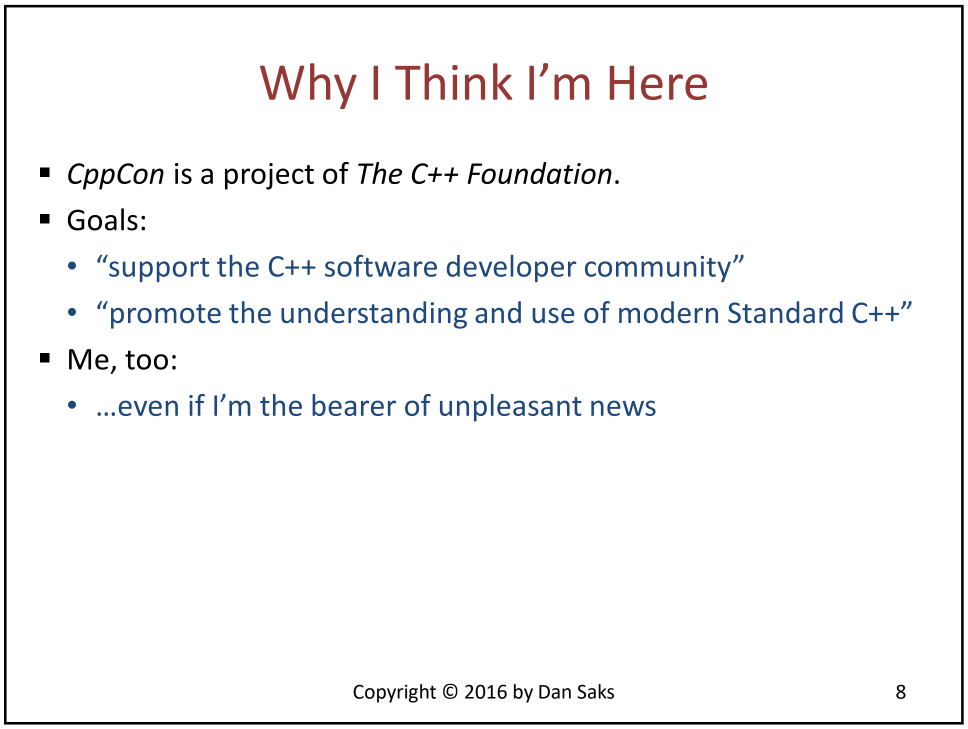




“What’s It to Me?”

- Let the market take care of itself?
- It’s probably personal if you work in:
  - aerospace
  - automotive
  - telecommunications
  - other embedded domains

7



Why I Think I’m Here

- *CppCon* is a project of *The C++ Foundation*.
- Goals:
  - “support the C++ software developer community”
  - “promote the understanding and use of modern Standard C++”
- Me, too:
  - …even if I’m the bearer of unpleasant news

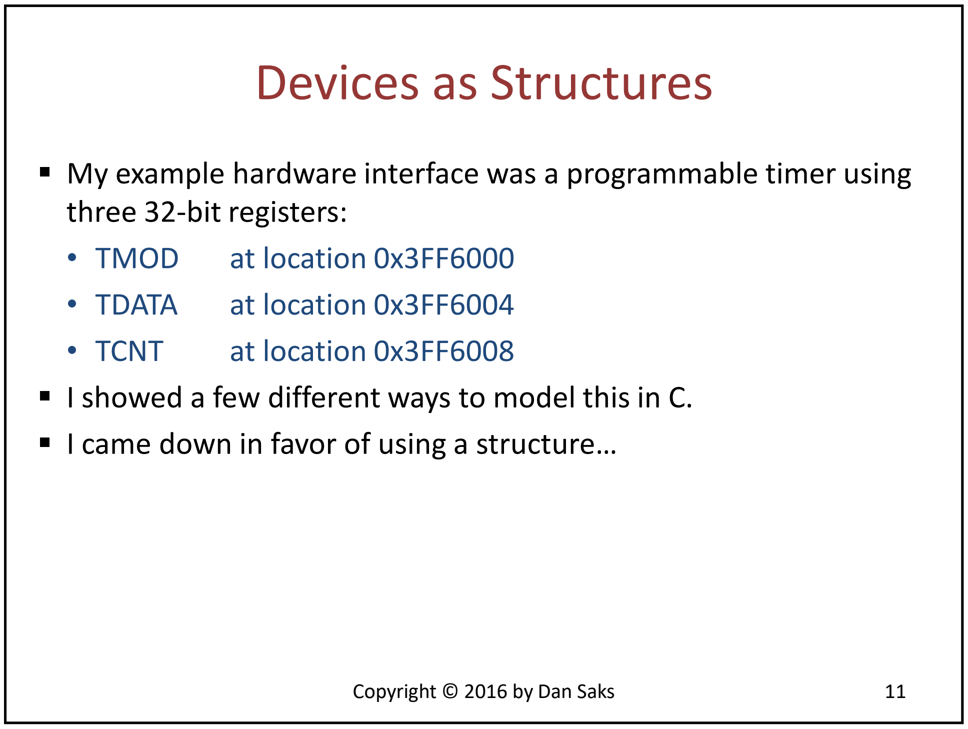
8

What to Do?

- Define the problem:
  - C++ isn’t thriving in application domains where it should.
  - C programmers who should be using C++ are not.
- We all “know” that C++ is better than C.
- The solution?
  - Do a better job of getting the word out?
  - Yes, but probably not in the way you think…

A Cautionary Tale

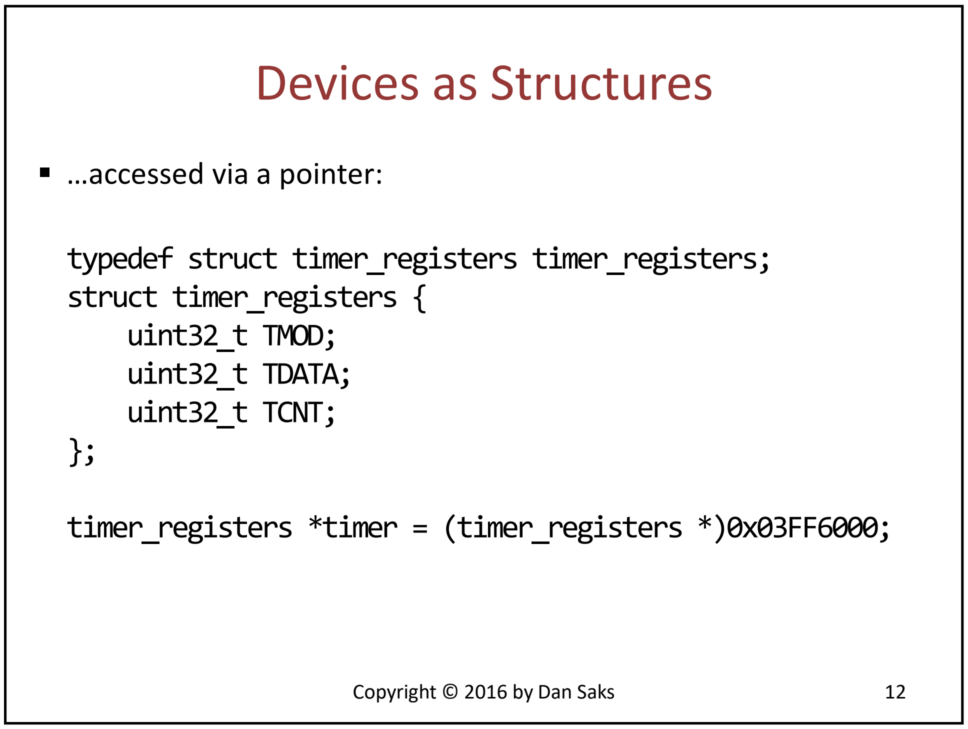
- In May 2010, I wrote a column on representing and manipulating memory-mapped devices in C.
  - I suggested that structures are the best way to represent device registers.
- My June column explained why C++ classes are even better than C structures for this purpose.
- The interaction with my readers was very enlightening.
- The response of the C++ community was, too.



Devices as Structures

- My example hardware interface was a programmable timer using three 32-bit registers:
  - TMOD at location 0x3FF6000
  - TDATA at location 0x3FF6004
  - TCNT at location 0x3FF6008
- I showed a few different ways to model this in C.
- I came down in favor of using a structure…

11



Devices as Structures

- …accessed via a pointer:

```
typedef struct timer_registers timer_registers;
struct timer_registers {
    uint32_t TMOD;
    uint32_t TDATA;
    uint32_t TCNT;
};

timer_registers *timer = (timer_registers *)0x03FF6000;
```

12

The Response

- My readers:
  - Yes, tell us more!
- Me:
  - OK!
  - The next month I showed how to do the same thing...
  - but better, as a C++ class...

Devices as Classes

- The public part:

```
class timer_registers {
public:
    enum { TICKS_PER_SEC = 50000000 };
    typedef uint32_t count_type;
    void disable();
    void enable();
    void set(count_type c);
    count_type get() const;
    ~~~
};
```

Devices as Classes

- The private part:

```
class timer_registers {
    ~~~
private:
    enum { TE = 0x01 };
    device_register TMOD;
    device_register TDATA;
    device_register TCNT;
};
```

- The class has the same storage layout as the structure.

The Responses

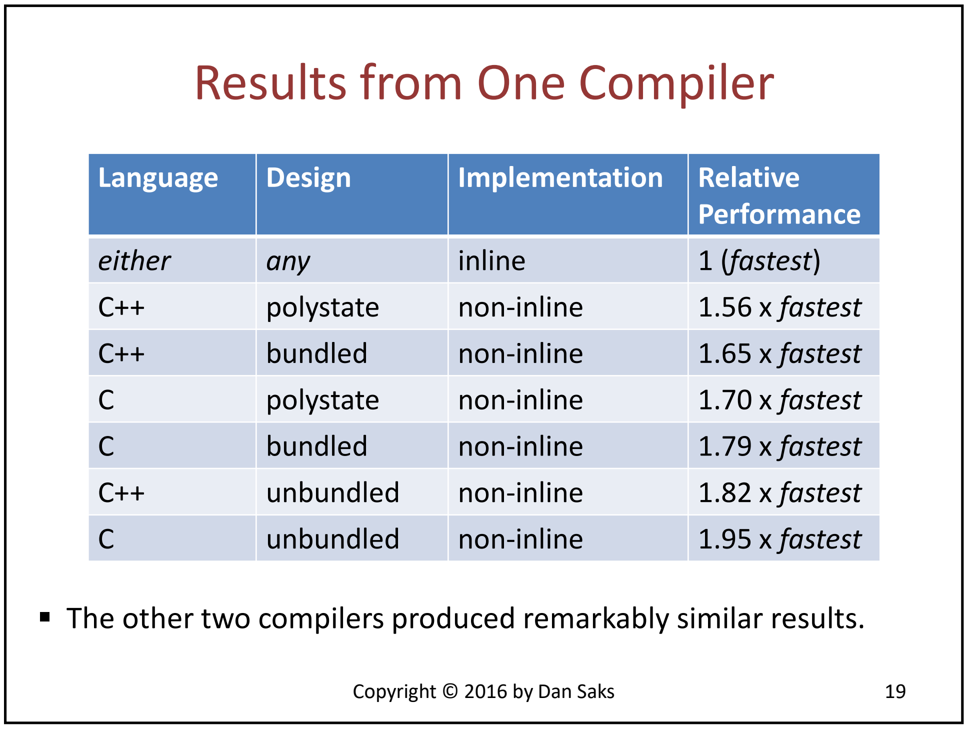
- My reader's were mixed:
  - Some loved it.
  - Others claimed the class incurred a performance penalty.
- Me:
  - I strongly suspected that C++ would perform almost as well, if not just as well, as C.
  - But I had no proof.
  - I decided to get it…

“Measuring Instead of Speculating”

- I followed with a series of columns describing my methodology.
- I tested alternatives for:
  - language: C vs. C++
  - class design: monostate vs. polystate
  - function implementation: inline vs. non-inline
  - access technique: pointer vs. reference vs. linker placement
- I used three different compilers of different vintages.

“Measuring Instead of Speculating”

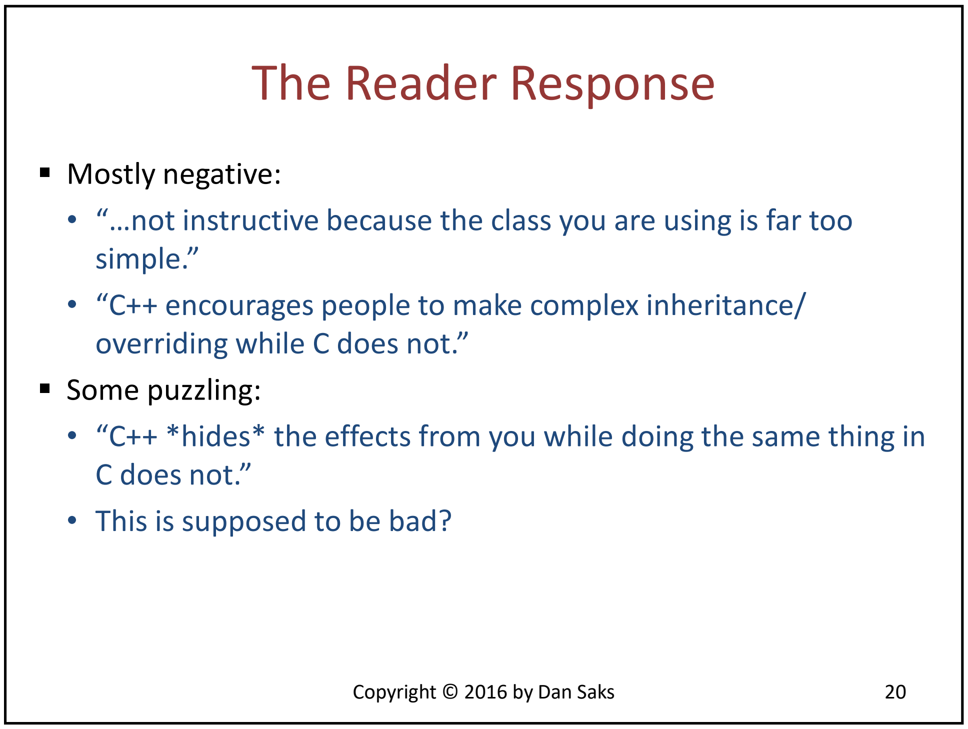
- Several colleagues reviewed my work.
  - All agreed that my methods were basically sound.
- My business partner, Steve Dewhurst, thought the series would make a real splash.
  - There was very little else like it on the Web.
- In December, 2010, I published the results…



Results from One Compiler

Language	Design	Implementation	Relative Performance
either	*any*	inline	1 (*fastest*)
C++	polystate	non-inline	1.56 x *fastest*
C++	bundled	non-inline	1.65 x *fastest*
C	polystate	non-inline	1.70 x *fastest*
C	bundled	non-inline	1.79 x *fastest*
C++	unbundled	non-inline	1.82 x *fastest*
C	unbundled	non-inline	1.95 x *fastest*

- The other two compilers produced remarkably similar results.



The Reader Response

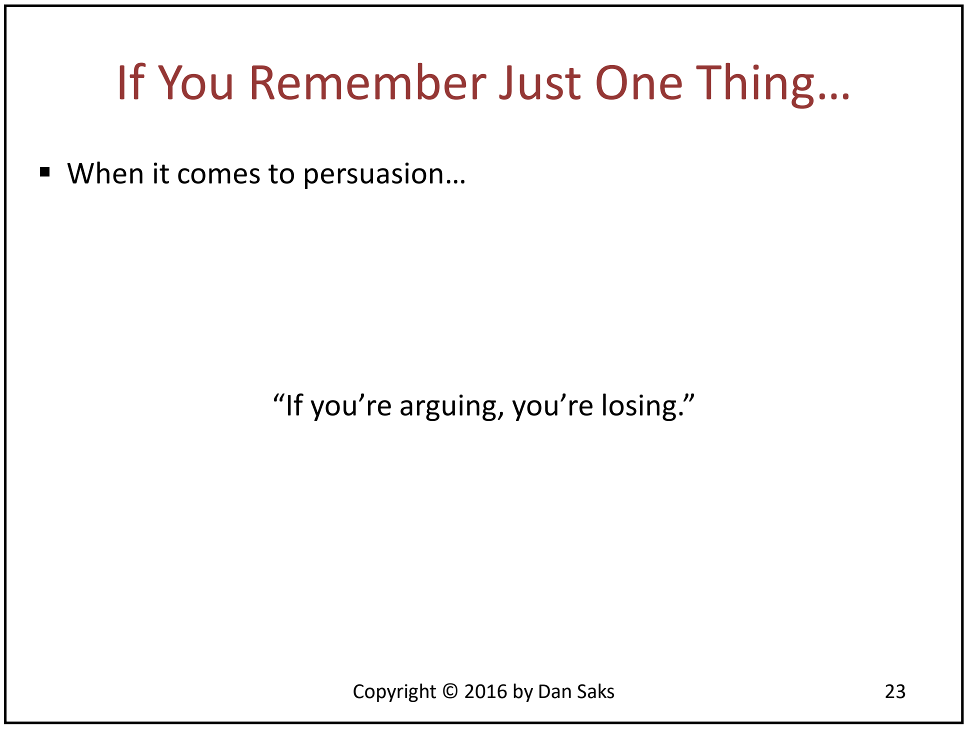
- Mostly negative:
  - "…not instructive because the class you are using is far too simple."
  - "C++ encourages people to make complex inheritance/ overriding while C does not."
- Some puzzling:
  - "C++ *hides* the effects from you while doing the same thing in C does not."
  - This is supposed to be bad?

The C++ Community Response

- Crickets
- I later found out why…

The Rumors of My Death…

- In November, 2012, *Dr. Dobbs Journal* republished one of my *embedded.com* articles:
  - *"Storage layout for polymorphic objects"*
- ***isocpp.org/blog*** published a link to that article, along with this comment:
  - "It's great to see Dan writing about C++ again…"
- Let's consider what to do, after…

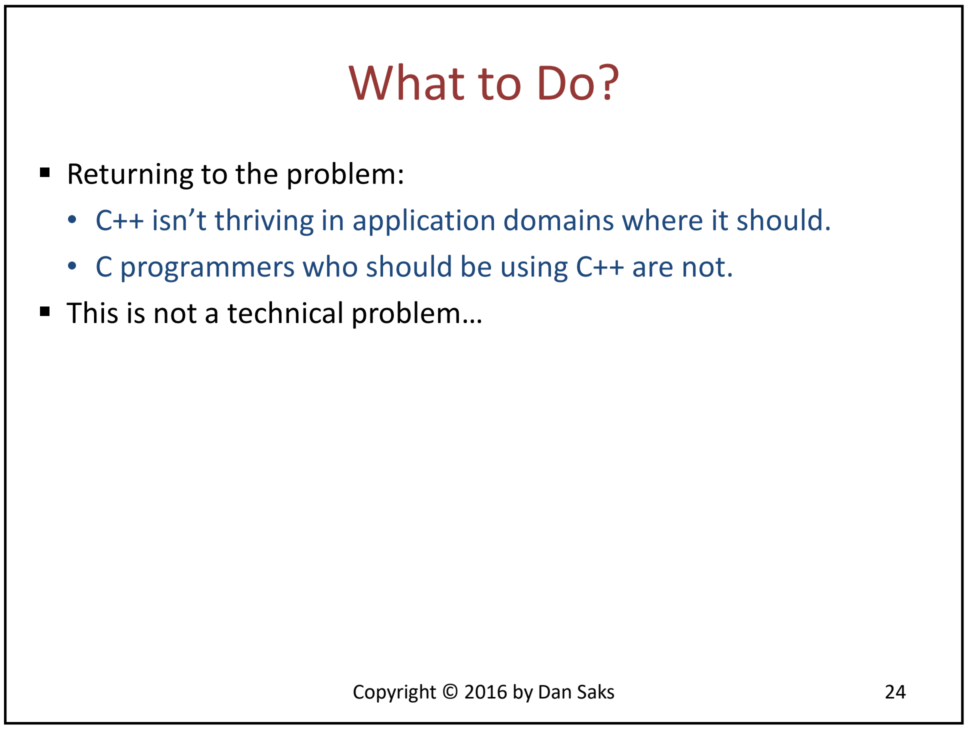


If You Remember Just One Thing…

- When it comes to persuasion…

“If you’re arguing, you’re losing.”

Copyright © 2016 by Dan Saks 23



What to Do?

- Returning to the problem:
  - C++ isn’t thriving in application domains where it should.
  - C programmers who should be using C++ are not.
- This is not a technical problem…

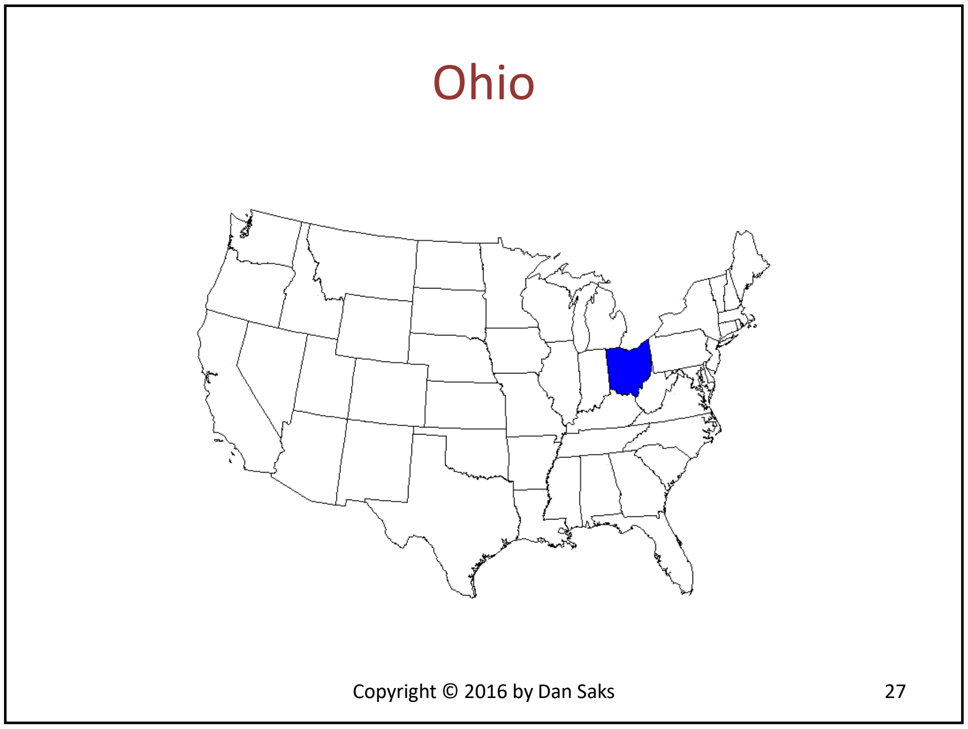
Copyright © 2016 by Dan Saks 24

What to Do?

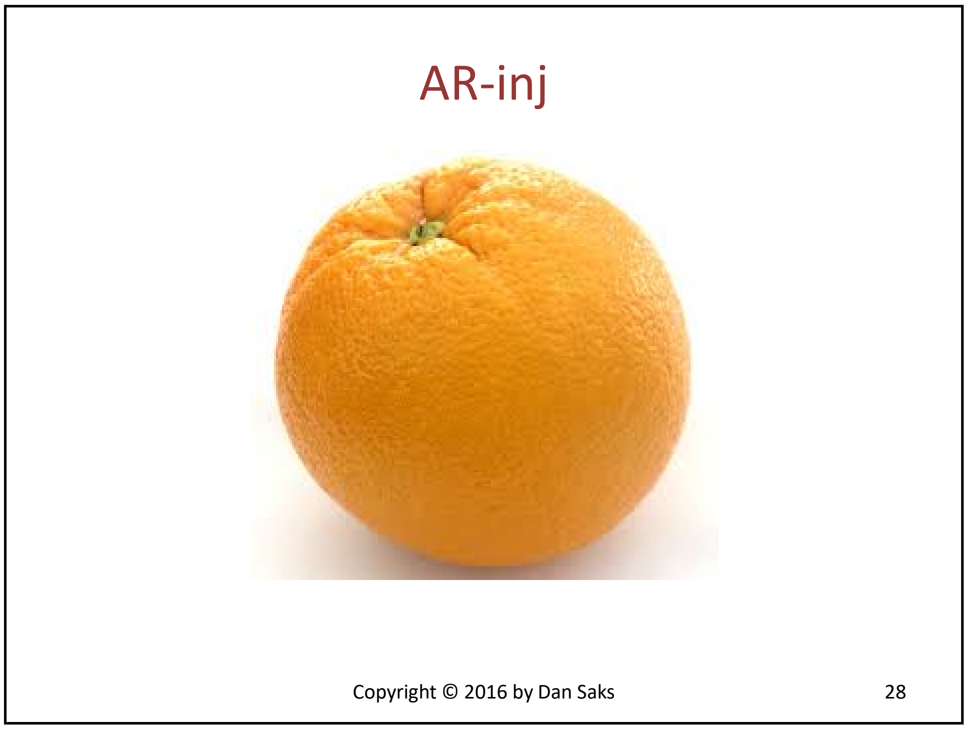
- “The Second Law of Consulting: No matter how it looks at first, it's always a people problem.”
  —Gerald Weinberg: *The Secrets of Consulting*

TMM = Too Much Misinformation?

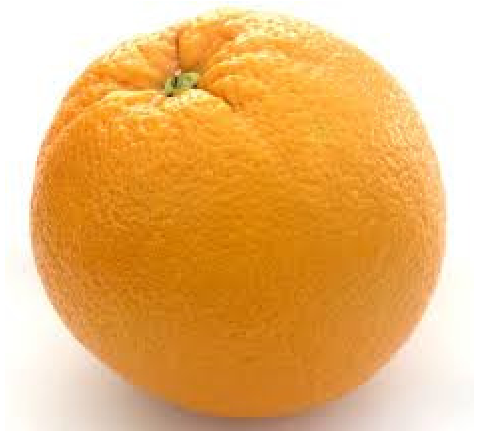
- Yes!
- But now it’s much more than that.
- It has to do with what I learned by living in…



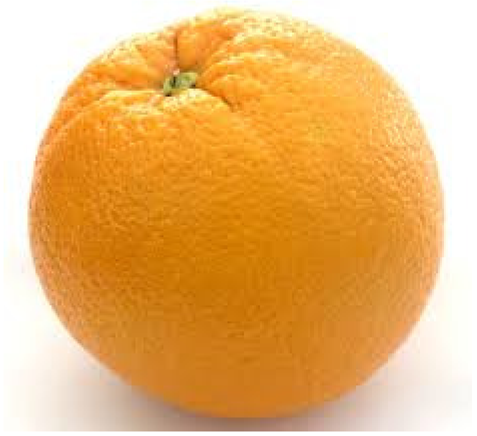
Ohio



AR-inj



OR-inj



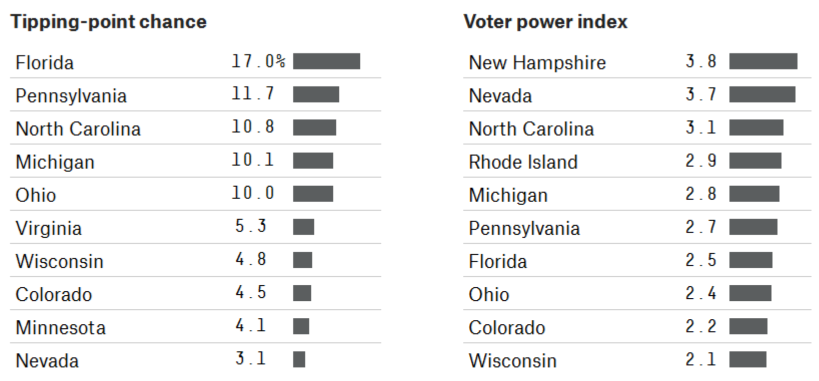
KWAU-tuh

KWAR-ter

I Will Blend In Only So Far

- In the East:
  - merry
  - marry
  - Mary
- In the Midwest:
  - merry
  - merry
  - Merry

We’re Number 5! Or 8?



Tipping-point chance	
Florida	17.0%
Pennsylvania	11.7
North Carolina	10.8
Michigan	10.1
Ohio	10.0
Virginia	5.3
Wisconsin	4.8
Colorado	4.5
Minnesota	4.1
Nevada	3.1

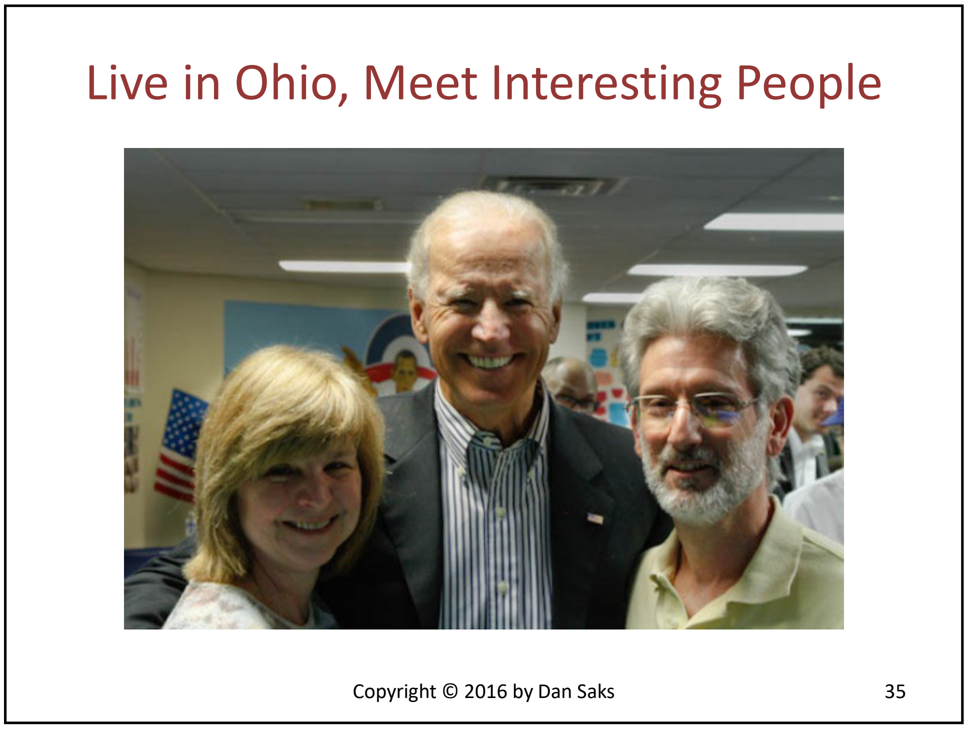
Voter power index	
New Hampshire	3.8
Nevada	3.7
North Carolina	3.1
Rhode Island	2.9
Michigan	2.8
Pennsylvania	2.7
Florida	2.5
Ohio	2.4
Colorado	2.2
Wisconsin	2.1

- http://projects.fivethirtyeight.com/2016-election-forecast

Copyright © 2016 by Dan Saks 33

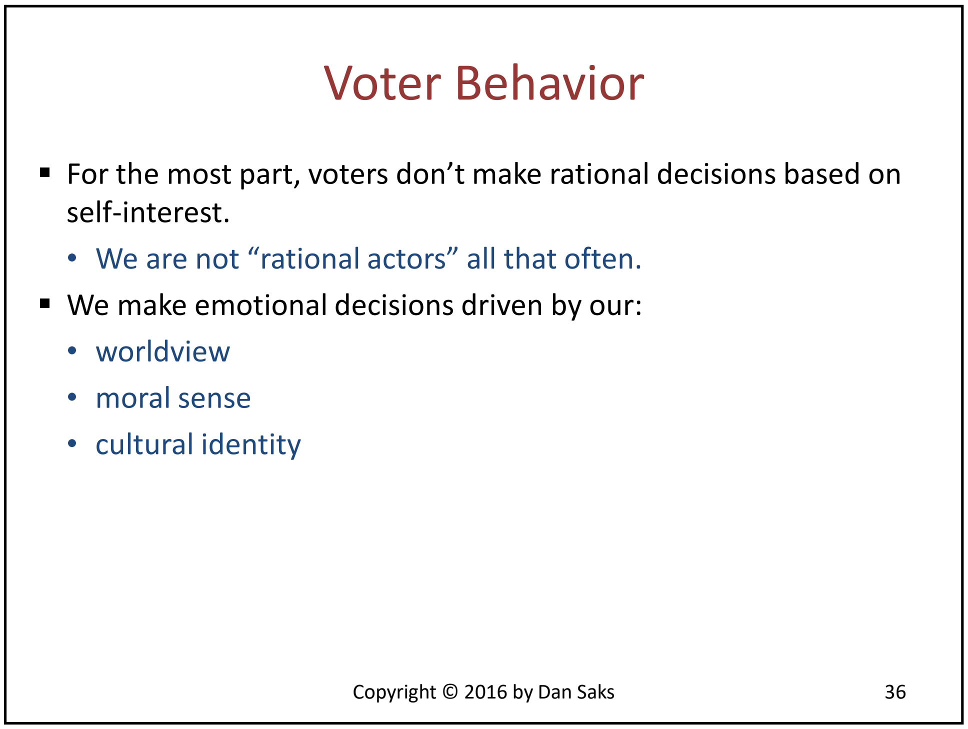
Live in Ohio, Meet Interesting People

Copyright © 2016 by Dan Saks 34



Live in Ohio, Meet Interesting People

35



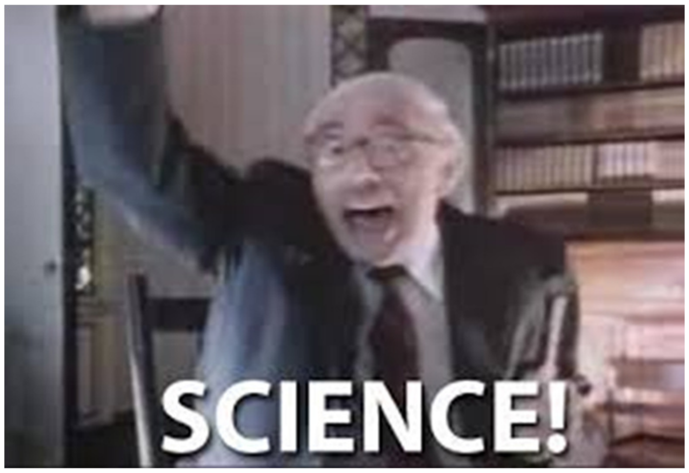
Voter Behavior

- For the most part, voters don’t make rational decisions based on self-interest.
  - We are not “rational actors” all that often.
- We make emotional decisions driven by our:
  - worldview
  - moral sense
  - cultural identity

36

People Behavior

- People make personal, and professional, decisions based largely on emotional rather than rational factors.
- Software engineers are people, my friend.



- Which we’ll get to in just a second, after this…

If You Remember Just One Thing…

“If you’re arguing, you’re losing.”

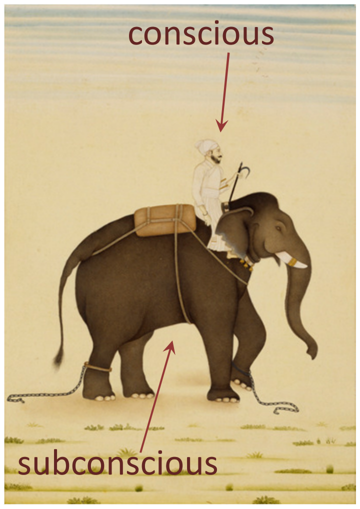
Science!

- It’s a combination of:
  - Psychology and Cognitive Science
    - mind, behavior and information processing
  - Linguistics
    - language
  - Political Science
    - government and political behavior

What Science Tells Us

- "…the mind is divided, like a rider on an elephant, and the rider's job is to serve the elephant."

  —Jonathan Haidt

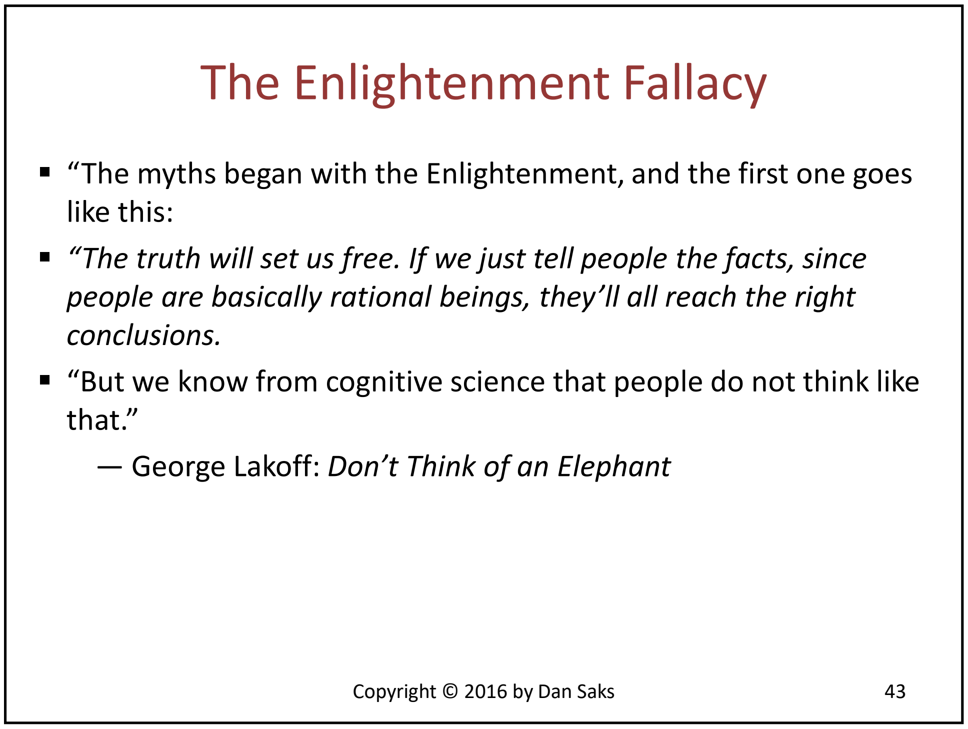


Copyright © 2016 by Dan Saks 41

Motivated Reasoning

- To varying degrees, we all:
  - make gut-level decisions, and
  - use our intellect to justify those decisions.
- *Motivated reasoning* is also known as:
  - motivated cognition
  - post hoc reasoning

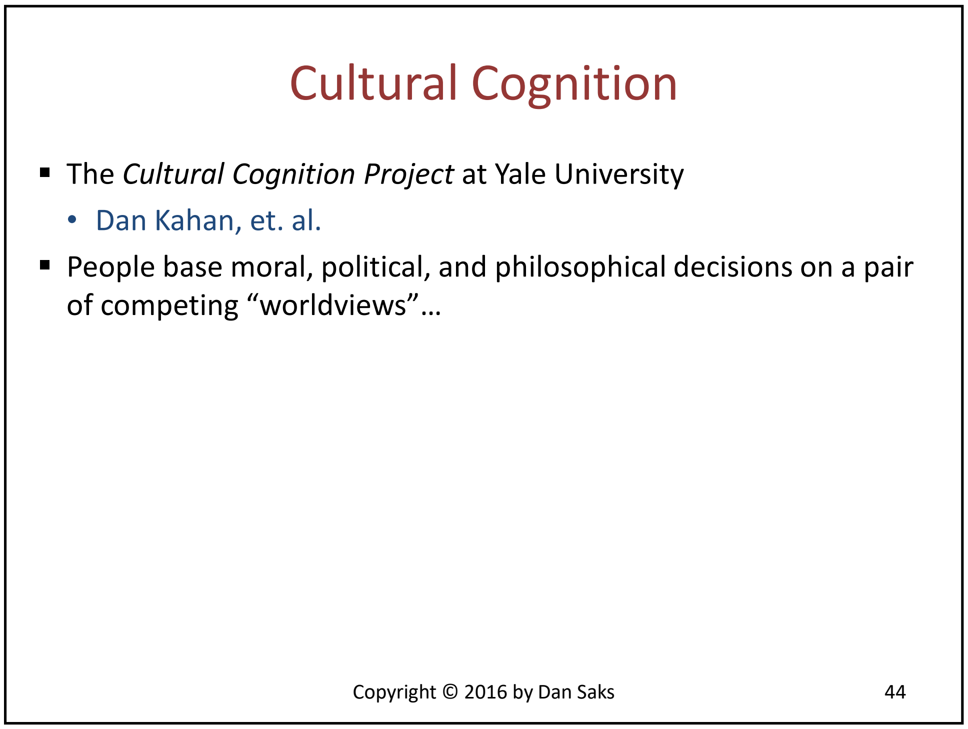
Copyright © 2016 by Dan Saks 42



The Enlightenment Fallacy

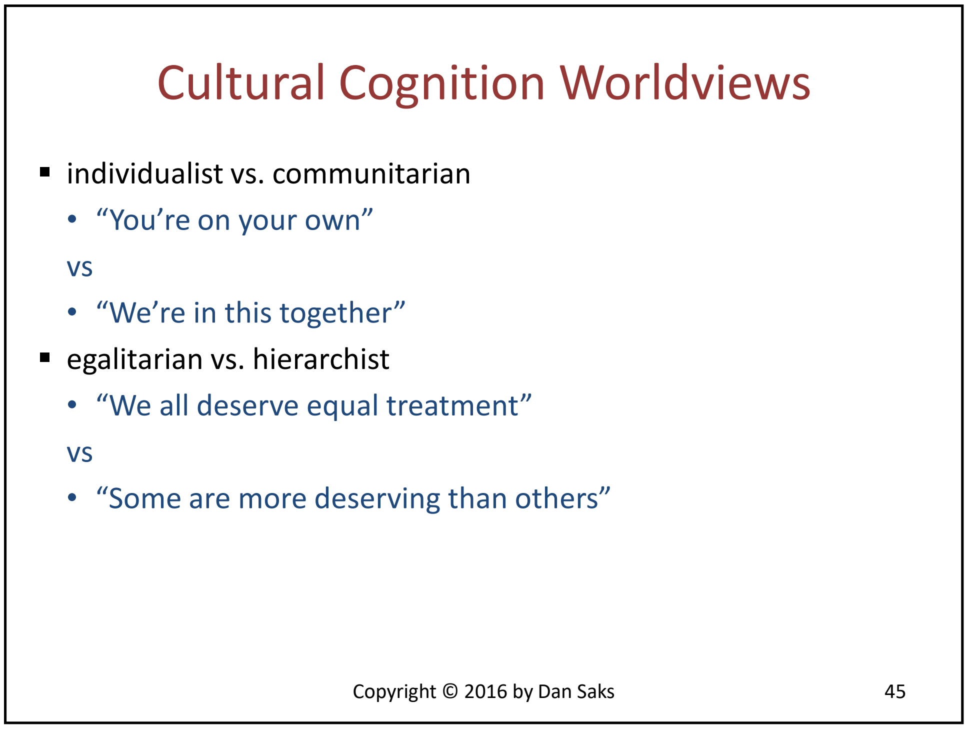
- “The myths began with the Enlightenment, and the first one goes like this:
- *“The truth will set us free. If we just tell people the facts, since people are basically rational beings, they’ll all reach the right conclusions.*
- “But we know from cognitive science that people do not think like that.”

    — George Lakoff: *Don’t Think of an Elephant*



Cultural Cognition

- The *Cultural Cognition Project* at Yale University
    - Dan Kahan, et. al.
- People base moral, political, and philosophical decisions on a pair of competing “worldviews”…

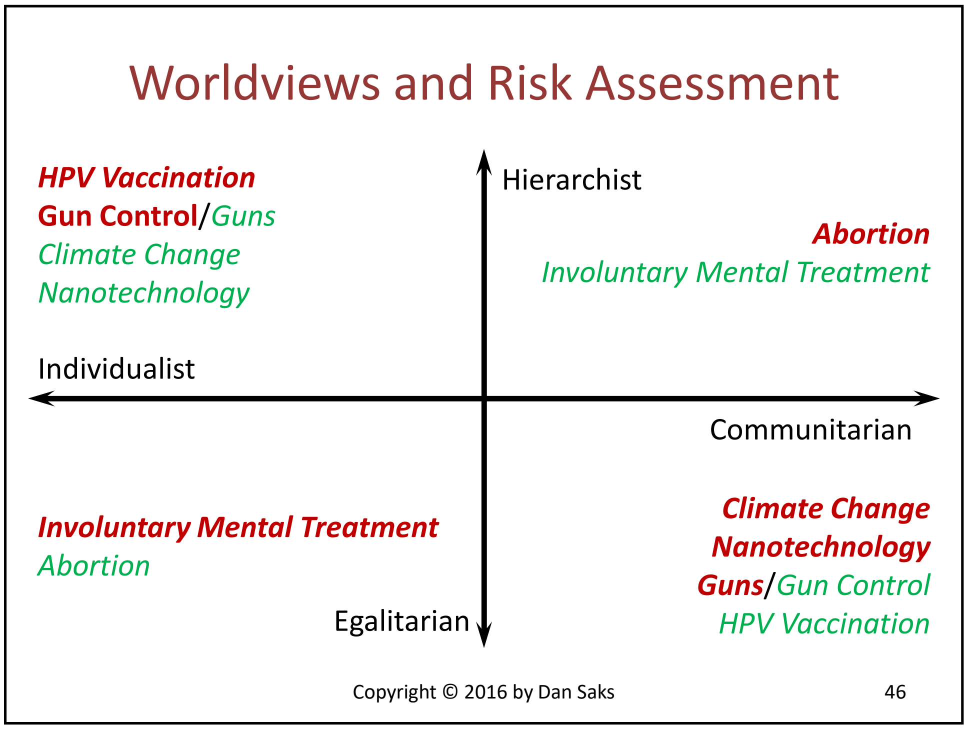


Cultural Cognition Worldviews

- individualist vs. communitarian
  - “You’re on your own”

  vs
  - “We’re in this together”
- egalitarian vs. hierarchist
  - “We all deserve equal treatment”

  vs
  - “Some are more deserving than others”



Worldviews and Risk Assessment

HPV Vaccination
Gun Control/*Guns*
Climate Change
Nanotechnology

Hierarchist

Abortion
Involuntary Mental Treatment

Individualist

Communitarian

Involuntary Mental Treatment
Abortion

Egalitarian

Climate Change
Nanotechnology
Guns/*Gun Control*
HPV Vaccination

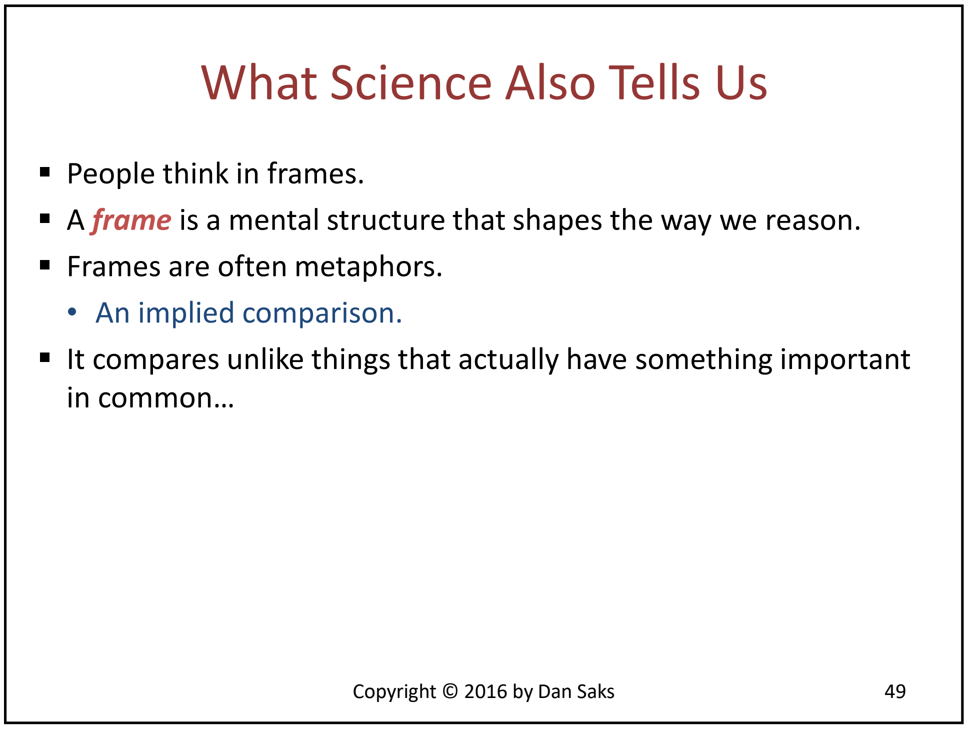
Motivated Numeracy

- Numeracy: the ability the comprehend numbers
- Kahan, et. al. found that:
  - An otherwise numerate person will likely misunderstand data if understanding it challenges his/her preexisting beliefs.

Motivated Reasoning

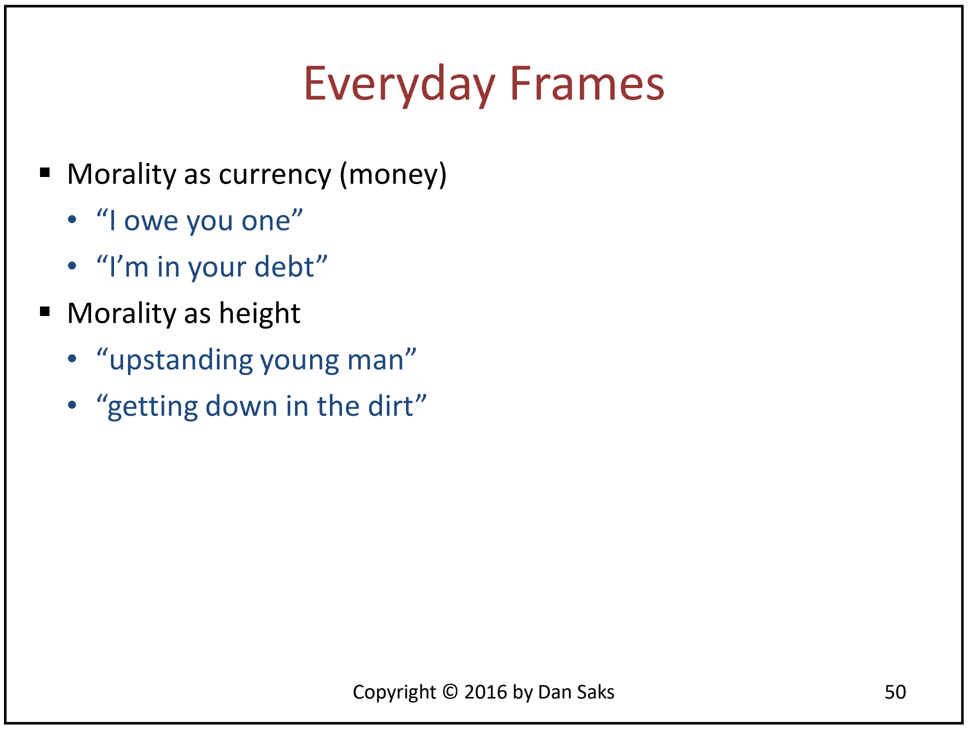
- “Reasoning will never make a Man correct an ill Opinion, which by Reasoning he never acquired.”

  — Jonathan Swift



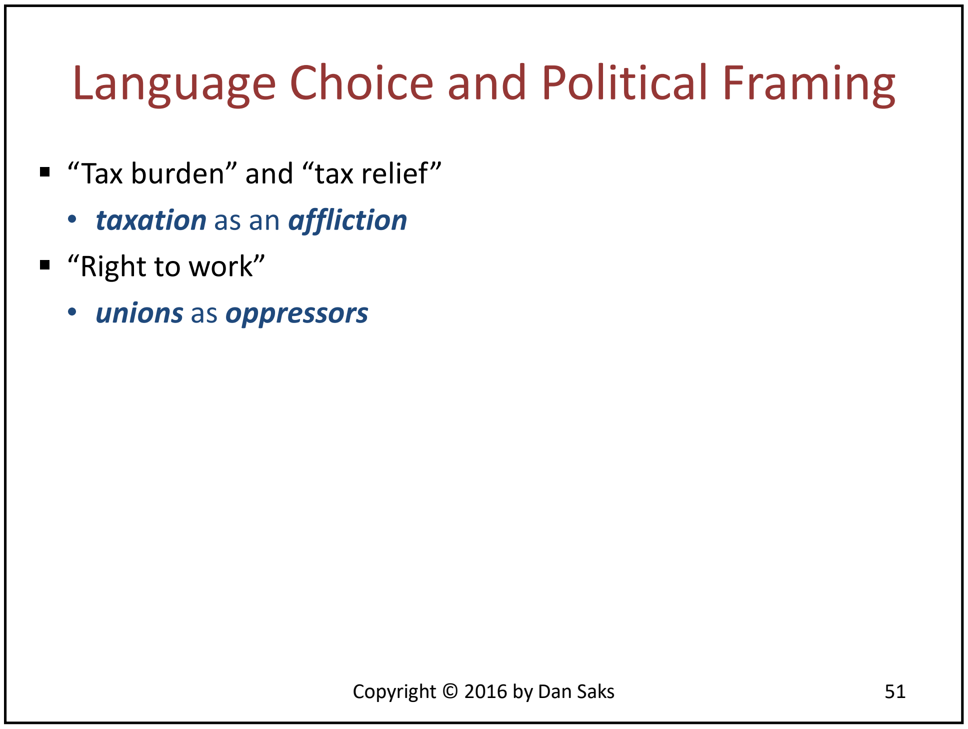
What Science Also Tells Us

- People think in frames.
- A ***frame*** is a mental structure that shapes the way we reason.
- Frames are often metaphors.
  - An implied comparison.
- It compares unlike things that actually have something important in common…



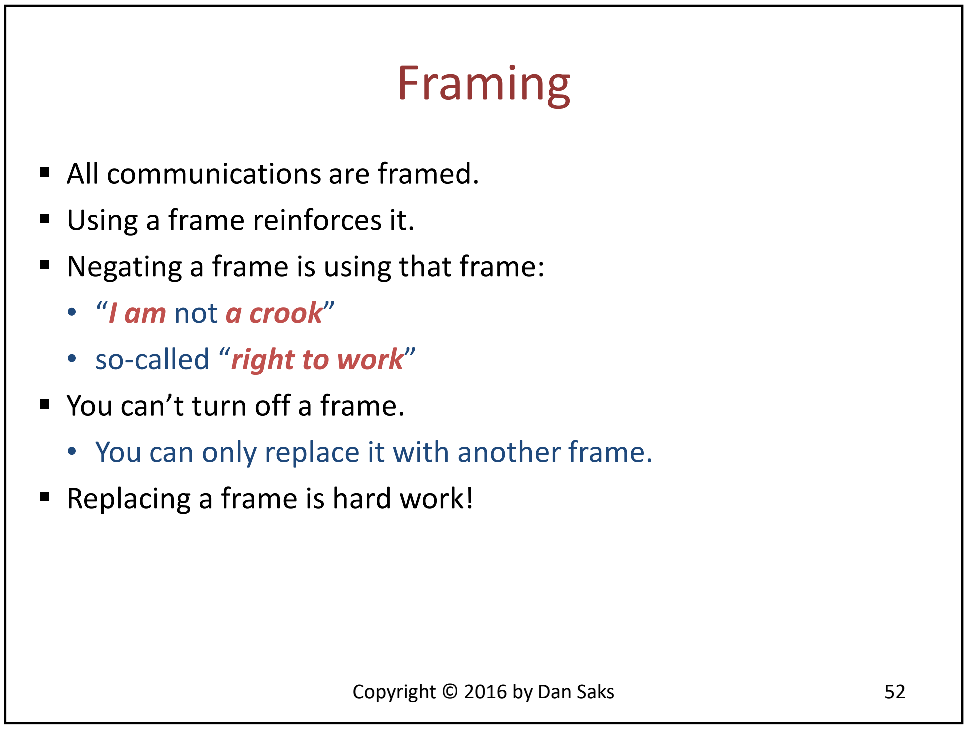
Everyday Frames

- Morality as currency (money)
  - “I owe you one”
  - “I’m in your debt”
- Morality as height
  - “upstanding young man”
  - “getting down in the dirt”



Language Choice and Political Framing

- "Tax burden" and "tax relief"
  - ***taxation*** as an ***affliction***
- "Right to work"
  - ***unions*** as ***oppressors***



Framing

- All communications are framed.
- Using a frame reinforces it.
- Negating a frame is using that frame:
  - "***I am*** not ***a crook***"
  - so-called "***right to work***"
- You can't turn off a frame.
  - You can only replace it with another frame.
- Replacing a frame is hard work!

If You Remember Just One Thing…

"If you're arguing, you're losing."

A Possible C Frame

- If you want to persuade C programmers to use C++, you have to understand how they view programming.
- Here's one frame I've encountered…

memcpy Copies Arrays

- C’s memcpy function can copy one array to another:

```
int ix[10], iy[10];
~~~
memcpy(ix, iy, sizeof(ix));     // copy iy to ix
```

memcpy is Flexible

- It can copy arrays of any type and any length:

```
double dx[20], dy[20];
~~~
memcpy(dx, dy, sizeof(dx));     // copy dy to dx
```

memcpy is Very Flexible

- Really, ***any*** type and ***any*** length:

```
struct widget wx[30], wy[30];
~~~
memcpy(wx, wy, sizeof(wx));     // copy wy to wx
```

- What's not to like?

memcpy is Lax

- memcpy can copy incompatible arrays:

```
int ix[10];
~~~
double dy[20];
~~~
memcpy(ix, dy, sizeof(ix));     // ix == gibberish
```

- This leaves gibberish in ix.

memcpy is Very Lax

- memcpy can copy too much:

```
int ix[10], iy[20];
~~~
double dy[20];
~~~
memcpy(ix, dy, sizeof(dy));     // too far
```

- This overflows ix, probably clobbering iy.

C's Compile-Time Checking is Weak

- C will compile code with all sorts of bugs.
  - More so than most languages.
  - C programmers know this ***and accept it***.
- This breeds an unfortunate mindset…

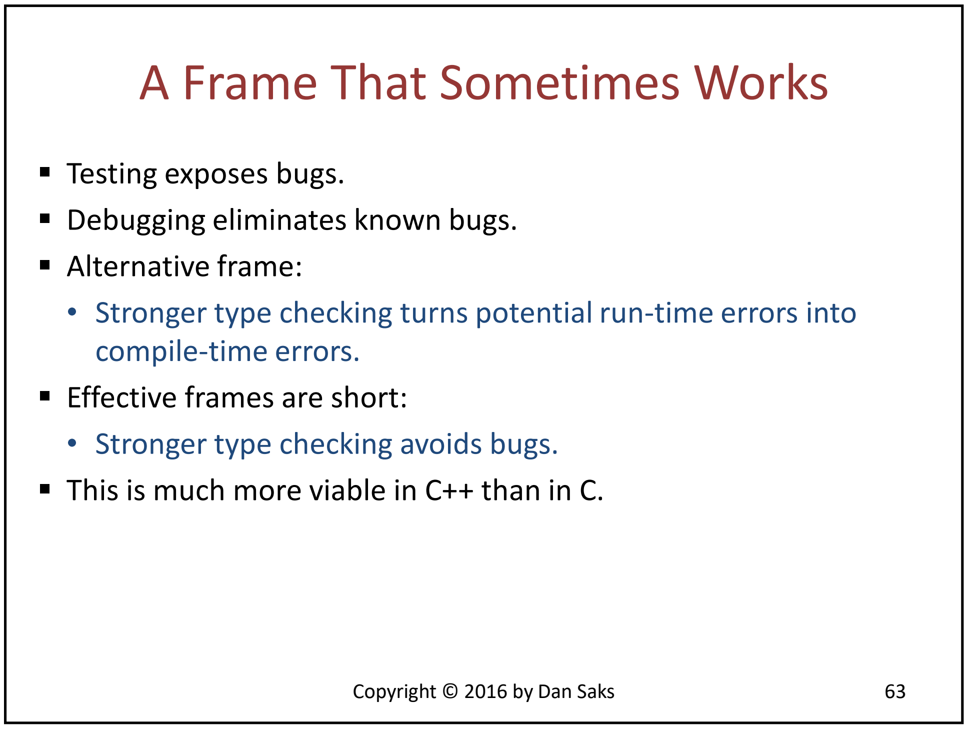
An All-Too-Common C Mindset

- Just get the code to compile, so you can get to real work…
- …debugging.
- "***The real work is debugging***" is a frame.
- What do you replace it with?

Replacing a Frame

- Again, replacing a frame is hard.
- Successful reframing usually requires experimentation:

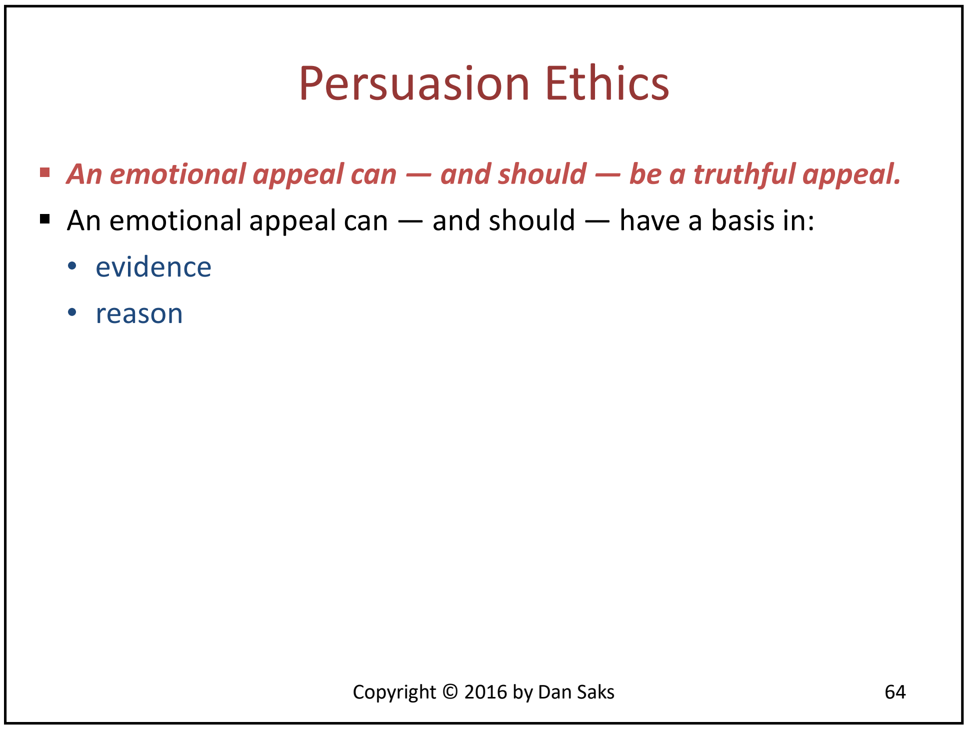
```
do {
    formulate an alternative frame
    try it out
} while (!successful);
```



A Frame That Sometimes Works

- Testing exposes bugs.
- Debugging eliminates known bugs.
- Alternative frame:
  - Stronger type checking turns potential run-time errors into compile-time errors.
- Effective frames are short:
  - Stronger type checking avoids bugs.
- This is much more viable in C++ than in C.

Copyright © 2016 by Dan Saks 63



Persuasion Ethics

- ***An emotional appeal can — and should — be a truthful appeal.***
- An emotional appeal can — and should — have a basis in:
  - evidence
  - reason

Copyright © 2016 by Dan Saks 64

Stronger Type Checking Avoids Bugs?

- There's not much published evidence that this is true.
  - Do we really know to a scientific certainty that it is?
  - Probably not.
- The Foundation should consider linking to whatever's available from their site.
- And even if it is true, saying so can make things worse…

Facts Can Backfire

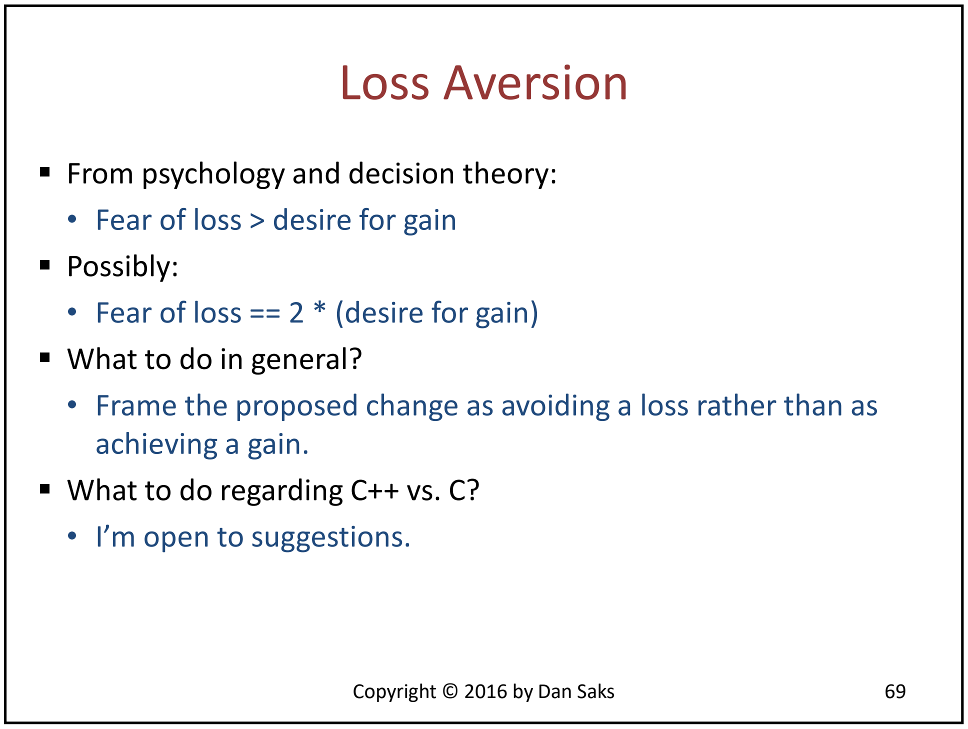
- In case I haven't filled you with enough despair…
- Brendan Nyhan and Jason Reifler have identified the "backfire effect":
  - Telling misinformed people correct information can actually ***increase*** belief in the misinformation.
  - And it doesn't matter where the correcting information comes from.

Frames Filter Facts

- For a “fact” to be believed, it must fit an accepted frame.
- It’s very hard to unmoor existing, erroneous frames.
- But, whatever you do, don’t make it worse…

If You Remember Just One Thing…

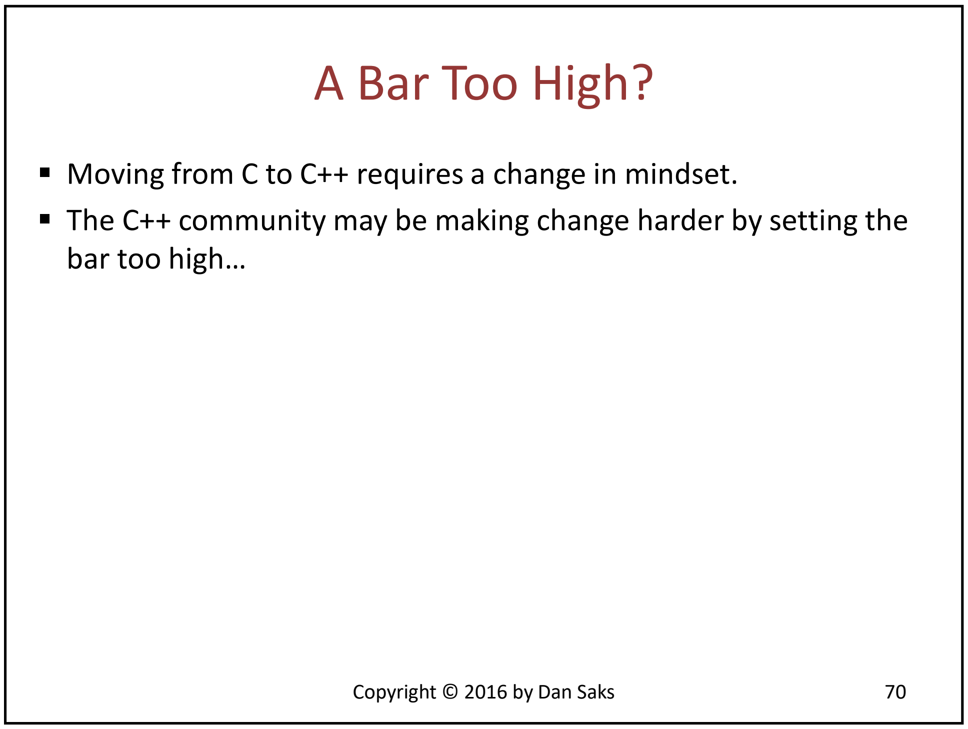
“If you’re arguing, you’re losing.”



Loss Aversion

- From psychology and decision theory:
  - Fear of loss > desire for gain
- Possibly:
  - Fear of loss == 2 * (desire for gain)
- What to do in general?
  - Frame the proposed change as avoiding a loss rather than as achieving a gain.
- What to do regarding C++ vs. C?
  - I'm open to suggestions.

69



A Bar Too High?

- Moving from C to C++ requires a change in mindset.
- The C++ community may be making change harder by setting the bar too high...

70

A Bar Too High?

- The “modern” approach to learning C++:
  - Use streams instead of FILEs.
  - Use vectors instead of arrays.
  - Use strings instead of null-terminated character sequences.
- For non-C programmers, this is almost certainly the best approach.

A Bar Too High?

- C++ was once a “Better C”.
- Now, it’s touted as a “new language”.
- The irony is:
  - A key aspect of C++ that made it popular is now deprecated.
  - For many C users, moving incrementally from C to C++ is probably much more practical.
- Some, possibly many, projects stay with C because they can’t bridge the widening cultural gap to C++.
- The “best” may be the enemy of the “better”.

The Only Way

- "There is only one way under high heaven to get anybody to do anything. Did you ever stop to think of that? Yes, just one way. And that is by making the other person want to do it.
- "Remember, there is no other way."

  — Dale Carnegie: *How to Win Friends and Influence People*

- "So the only way on earth to influence other people is to talk about what they want and show them how to get it."

  — Dale Carnegie: *How to Win Friends and Influence People*

Concrete Suggestions

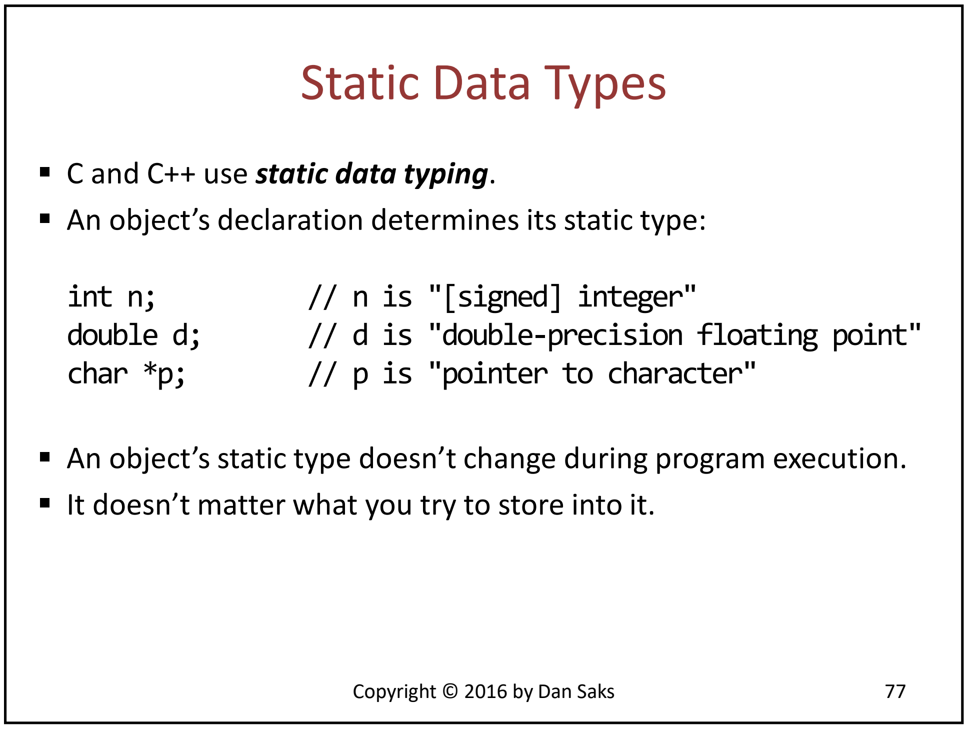
- Don't lead with (your version of) the "facts".
- Find out what your colleagues want out of their software development.
  - Identify specific, actionable goals and concerns.
- Figure out how to align their goals with yours.
- Ask what you can do to:
  - Allay their concerns.
  - Help them get what you want in common.
- You won't get there by goading, or belittling.
- And, of course…

If You Remember Just One Thing…

"If you're arguing, you're losing."

Concrete Suggestions

- Develop real insight into, and appreciation of, the C++ type system.
- Get good at articulating it.
- You have my permission to borrow, and improve upon, the following…

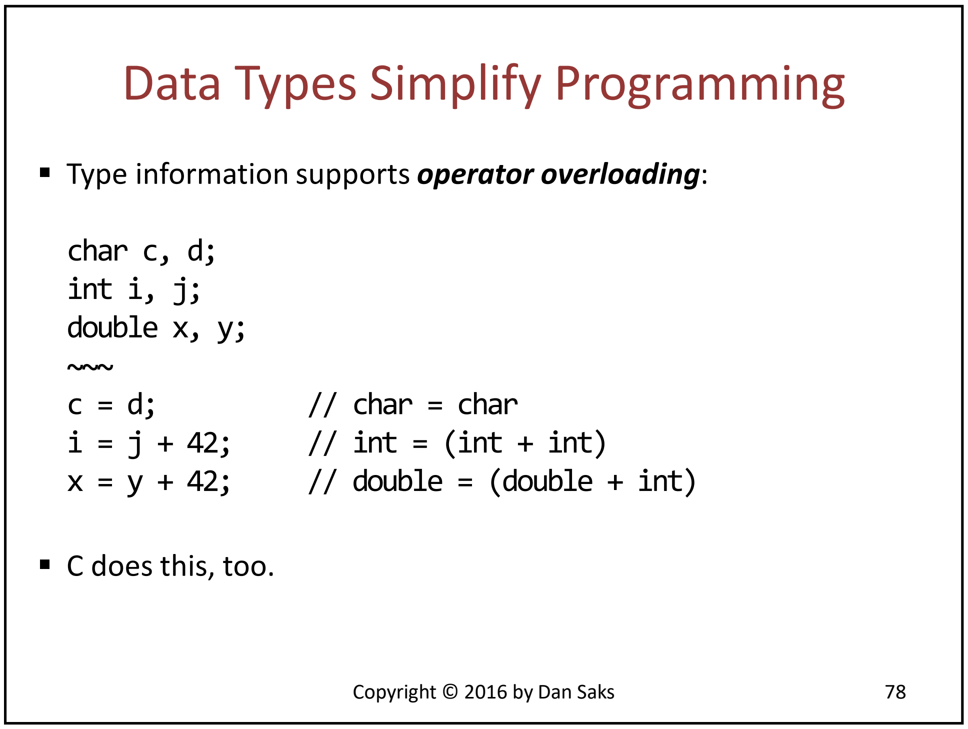


Static Data Types

- C and C++ use ***static data typing***.
- An object’s declaration determines its static type:

```
int n;         // n is "[signed] integer"
double d;      // d is "double-precision floating point"
char *p;       // p is "pointer to character"
```

- An object’s static type doesn’t change during program execution.
- It doesn’t matter what you try to store into it.



Data Types Simplify Programming

- Type information supports ***operator overloading***:

```
char c, d;
int i, j;
double x, y;
~~~
c = d;         // char = char
i = j + 42;    // int = (int + int)
x = y + 42;    // double = (double + int)
```

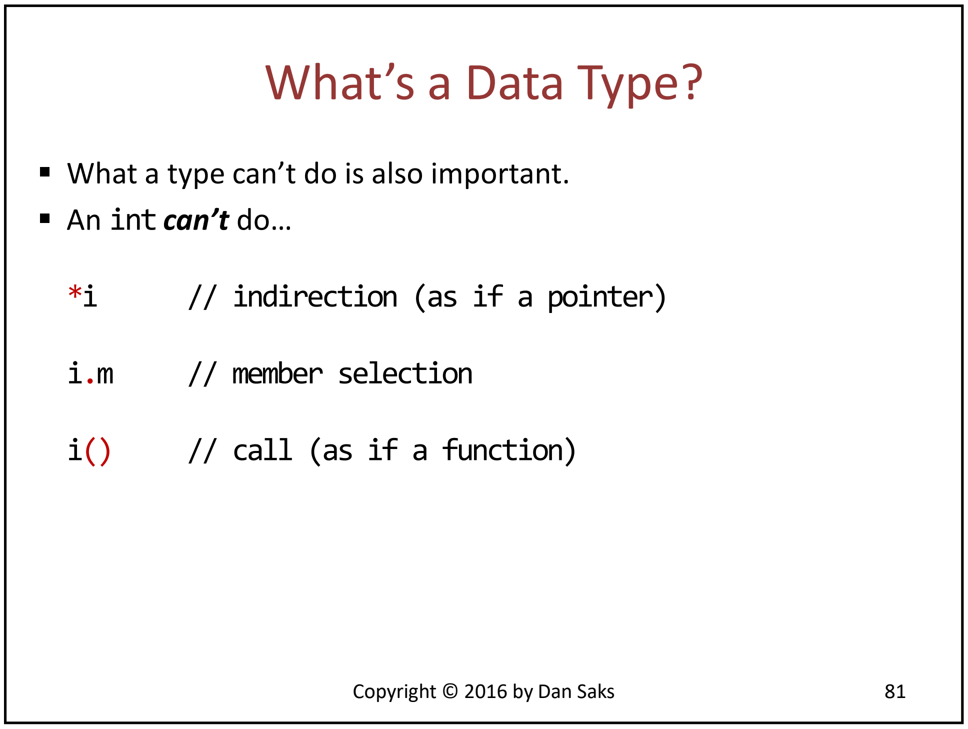
- C does this, too.

What's a Data Type?

- A ***data type*** is a bundle of compile-time properties for an object:
  - ***size*** and ***alignment***
  - ***set of valid values***
  - ***set of permitted operations***

What's a Data Type?

- On a typical 32-bit processor, type `int` has:
  - ***size*** and ***alignment*** of 4 (bytes)
  - ***values*** from -2147483648 to 2147483647, inclusive
    - integers only
  - ***operations*** including:
    - unary +, -, !, ~, &, ++, --
    - binary =, +, -, *, /, %, <, >, ==, !=, &, |, &&, ||



What's a Data Type?

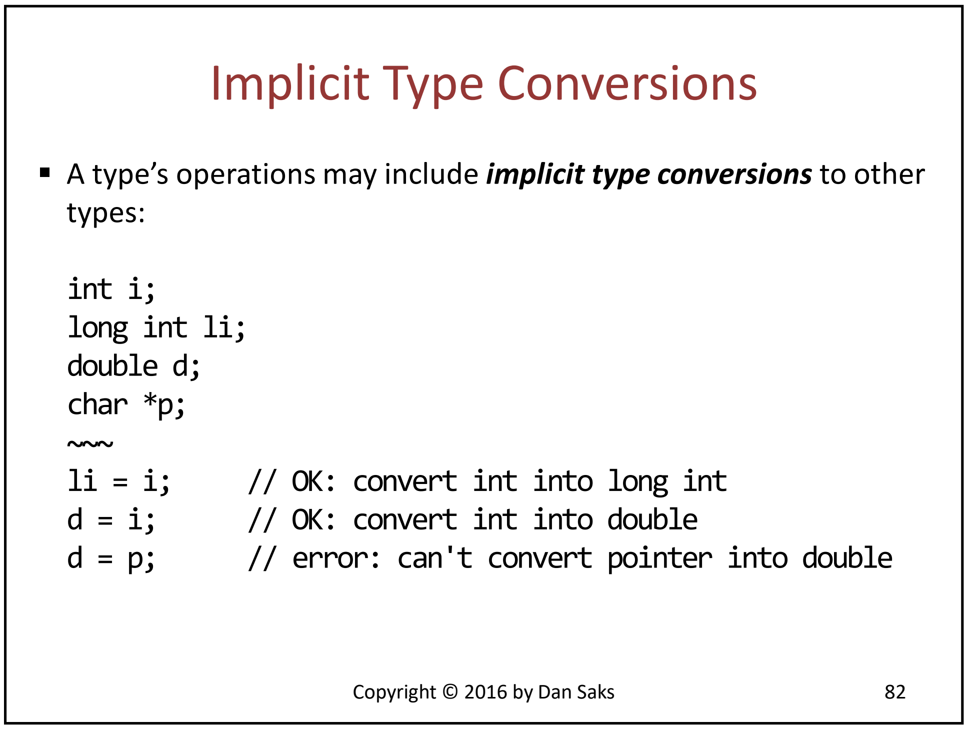
- What a type can't do is also important.
- An `int` ***can't*** do…

```
*i      // indirection (as if a pointer)

i.m     // member selection

i()     // call (as if a function)
```

Copyright © 2016 by Dan Saks 81

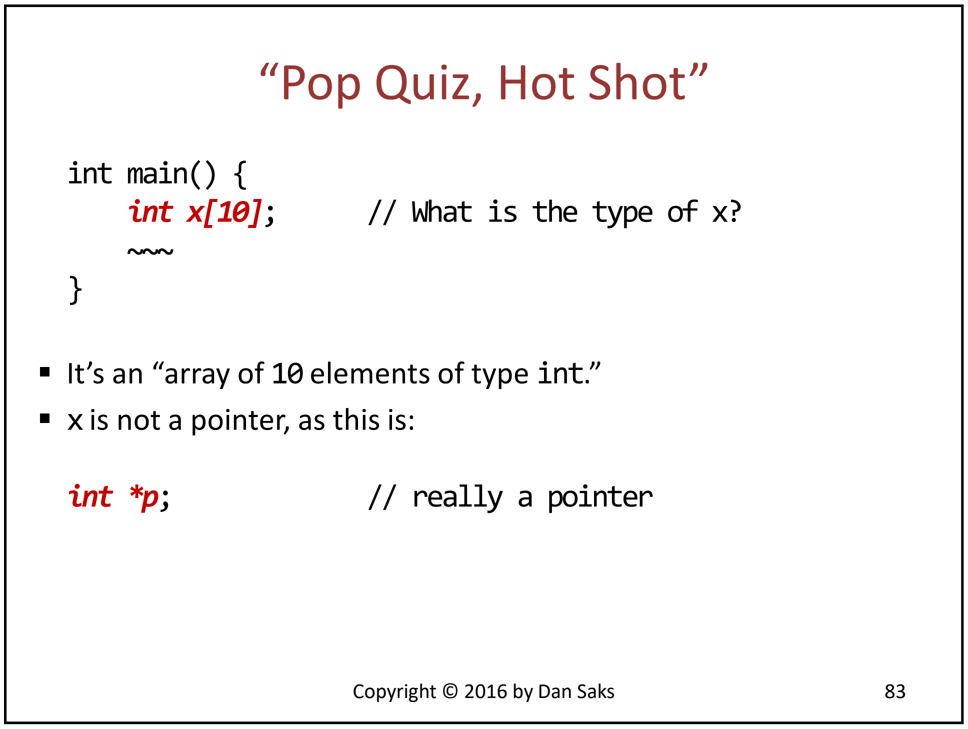


Implicit Type Conversions

- A type's operations may include ***implicit type conversions*** to other types:

```
int i;
long int li;
double d;
char *p;
~~~
li = i;     // OK: convert int into long int
d = i;      // OK: convert int into double
d = p;      // error: can't convert pointer into double
```

Copyright © 2016 by Dan Saks 82

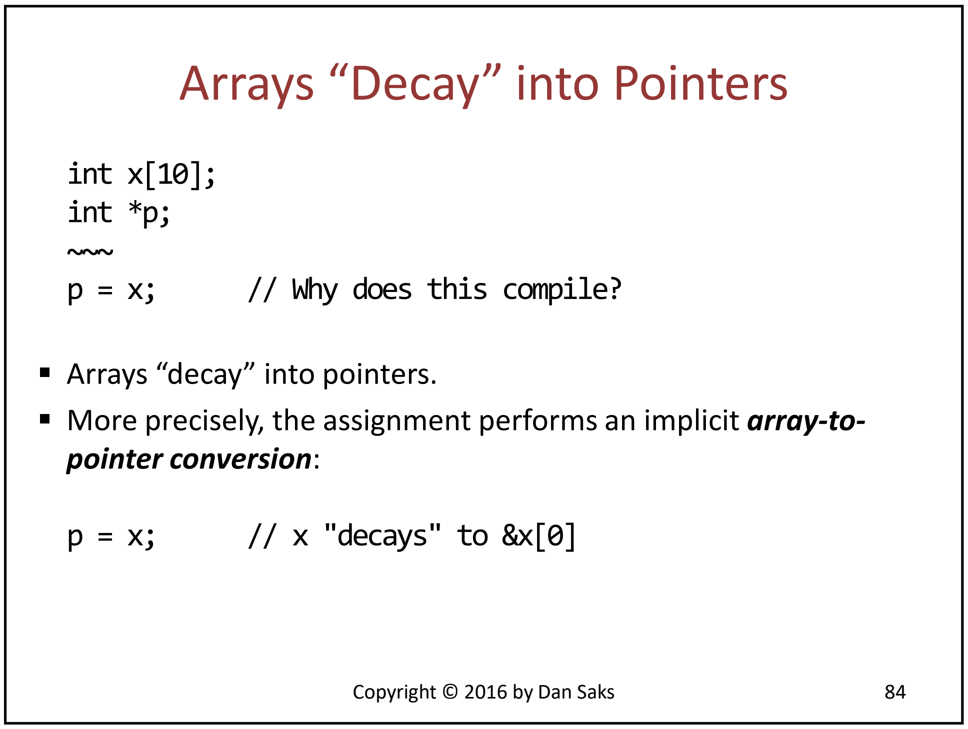


“Pop Quiz, Hot Shot”

```
int main() {
    int x[10];      // What is the type of x?
    ~~~
}
```

- It’s an “array of 10 elements of type int.”
- x is not a pointer, as this is:

```
int *p;             // really a pointer
```

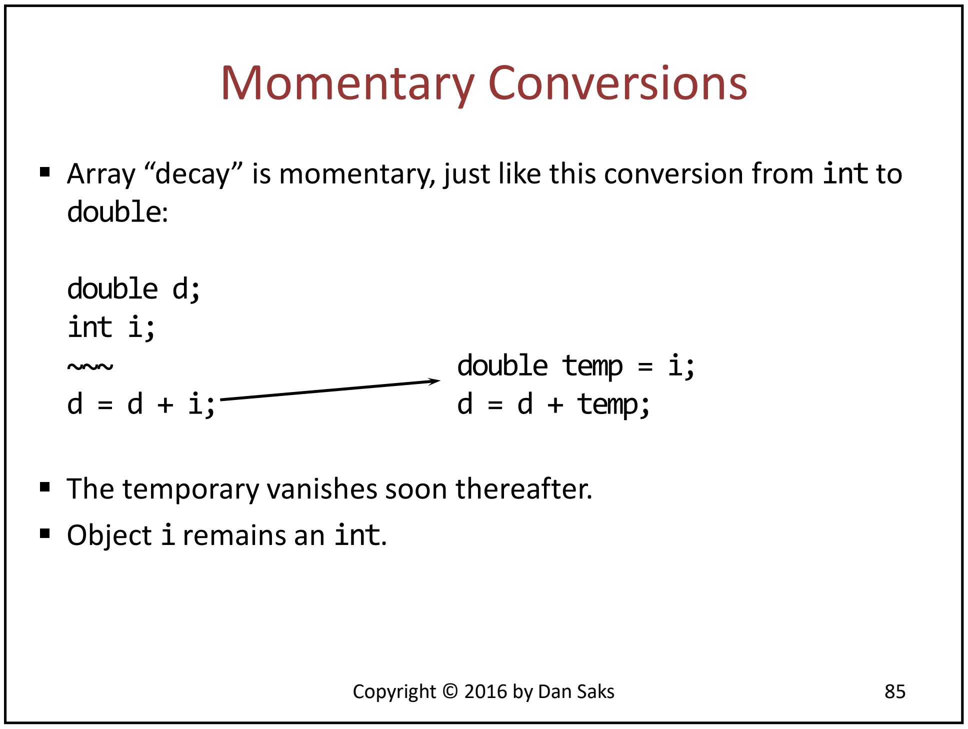


Arrays “Decay” into Pointers

```
int x[10];
int *p;
~~~
p = x;      // Why does this compile?
```

- Arrays “decay” into pointers.
- More precisely, the assignment performs an implicit ***array-to-pointer conversion***:

```
p = x;      // x "decays" to &x[0]
```

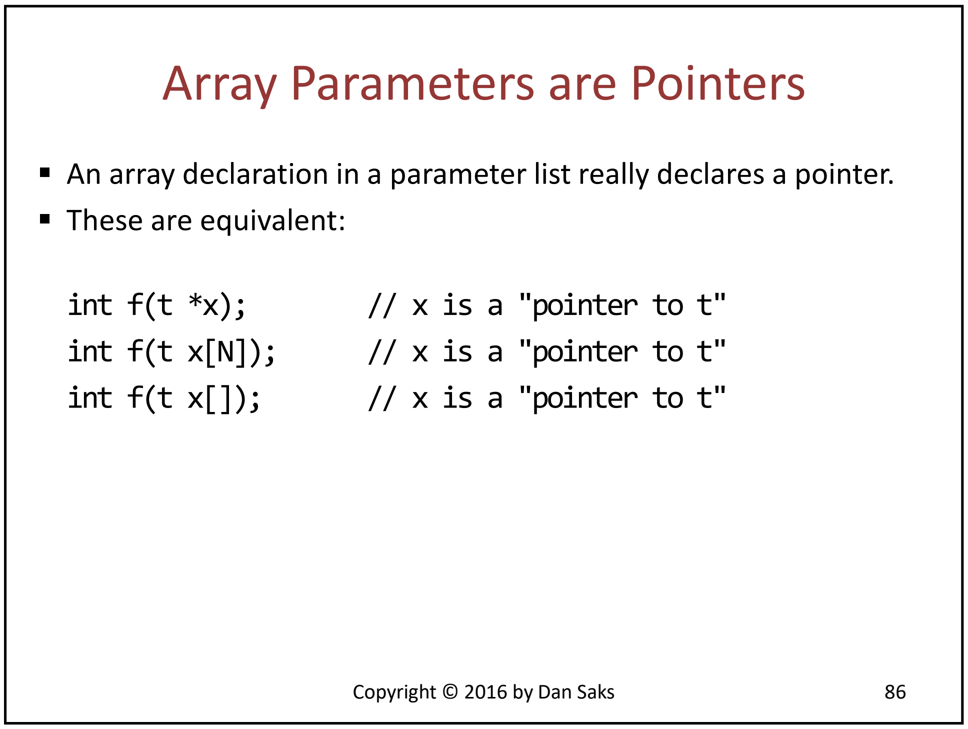


Momentary Conversions

- Array “decay” is momentary, just like this conversion from `int` to `double`:

```
double d;
int i;
~~~                          double temp = i;
d = d + i;                   d = d + temp;
```

- The temporary vanishes soon thereafter.
- Object `i` remains an `int`.



Array Parameters are Pointers

- An array declaration in a parameter list really declares a pointer.
- These are equivalent:

```
int f(t *x);        // x is a "pointer to t"
int f(t x[N]);      // x is a "pointer to t"
int f(t x[]);       // x is a "pointer to t"
```

But Who Uses Arrays Anymore?

- Again, Modern C++ users avoids arrays.
- However, you probably can’t avoid them if you work with:
  - practicing C programmers, or
  - on existing legacy code.

87

Preventing Accidents

- Type information ***helps prevent accidents***:

```
int *p, *q;
double x, y;
~~~
p = q / 4;          // error: can't divide a pointer
x = y & 0xFF;       // error: can't bitwise-and a double
```

- Compilers use type information to turn potential run-time errors into compile-time errors.
- C++ helps you leverage this better than C.

88

The Most Important Design Guideline?

- "Make interfaces easy to use correctly and hard to use incorrectly."

  —Scott Meyers

- `memcpy` is neither ETUC[1] nor HTUI[2]...

1. Easy To Use Correctly
2. Hard To Use Incorrectly

memcpy Isn't All That ETUC

- Again, a typical call looks like:

```
memcpy(ix, iy, n);  // n is the size to copy
```

- This would be easier:

```
memcpy(ix, iy);     // easier: compiler supplies size
```

- This would be even easier:

```
ix = iy;            // even easier: familiar assignment
```

memcpy is Definitely Not HTUI

- Again, the real problem with memcpy is that it invites errors:

```
int ix[10], iy[10];
~~~
double dy[20];
~~~
memcpy(ix, dy, sizeof(ix));     // fills ix with garbage
memcpy(ix, dy, sizeof(dy));     // copies past ix
```

Little to Work With in C

- The C alternative is completely impractical…
- Write a copy function for each type and size of interest:

```
void copy_char_10(char (*dst)[10], char (*src)[10]);
void copy_char_20(char (*dst)[20], char (*src)[20]);

void copy_int_10(int (*dst)[10], int (*src)[10]);
void copy_int_20(int (*dst)[20], int (*src)[20]);
```

- Yikes!

A Simple Alternative in C++

- In C++, you can write a ***function template*** that safely copies arrays of only the same element type and size:

```
int ix[10], iy[10], iz[20];
double dx[10];
~~~
array_copy(ix, iy);     // OK: same type and size
array_copy(ix, iz);     // error: size mismatch
array_copy(ix, dx);     // error: type mismatch
```

93

A Simple Alternative in C++

- The implementation is remarkably simple:

```
template <typename t, size_t n>
inline void array_copy(t (&dst)[n], t (&src)[n]) {
    for (size_t i = 0; i < n; ++i) {
        dst[i] = src[i];
    }
}
```

- array_copy is ETUC and HTUI.
- array_copy's parameters have the form ***t (&r)[n]***…

94

Reference-to-Array Parameters

- An array declaration in a parameter list declares a pointer:

```
int f(T *x);        // x is a "pointer to T"
int f(T x[N]);      // x is a "pointer to T"
int f(T x[]);       // x is a "pointer to T“
```

- This is different:

```
int f(T (&r)[N]);   // x is a "reference to array of T"
```

- The array dimension is part of the parameter type.
- It doesn’t “decay” away.

Keeping Up with memcpy

- If, for some reason, memcpy is faster than array_copy, you can do this:

```
template <typename t, size_t n>
inline void array_copy(t (&dst)[n], t (&src)[n]) {
    memcpy(dst, src, n * sizeof(t));
}
```

- array_copy can copy arrays of any type, but…
- memcpy works only with arrays whose elements are trivially assignable…

Restricting What You Copy

- You can use a type trait to restrict the copied type:

```
template <typename t, size_t n>
inline void array_copy(t (&dst)[n], t (&src)[n]) {
    static_assert(
        is_trivially_copy_assignable<t>::value,
        "element type has non-trivial copy assignment"
    );
    memcpy(dst, src, n * sizeof(t));
}
```

Who Cares?

- Making built-in array copies safer is probably not your concern.
  - After all, you've got vectors.
- But for many C programmers, it is a concern.
- If you feed them vector<T> or even array<T> before they're ready, you'll lose them.

Future Directions?

- Decide if we (the C++ Community) are really concerned about this.
- Assess the potential gain in market share.
- Identify factors that are indeed inhibiting C programmers from moving to C++.
- Decide what actions, if any, are appropriate and feasible.

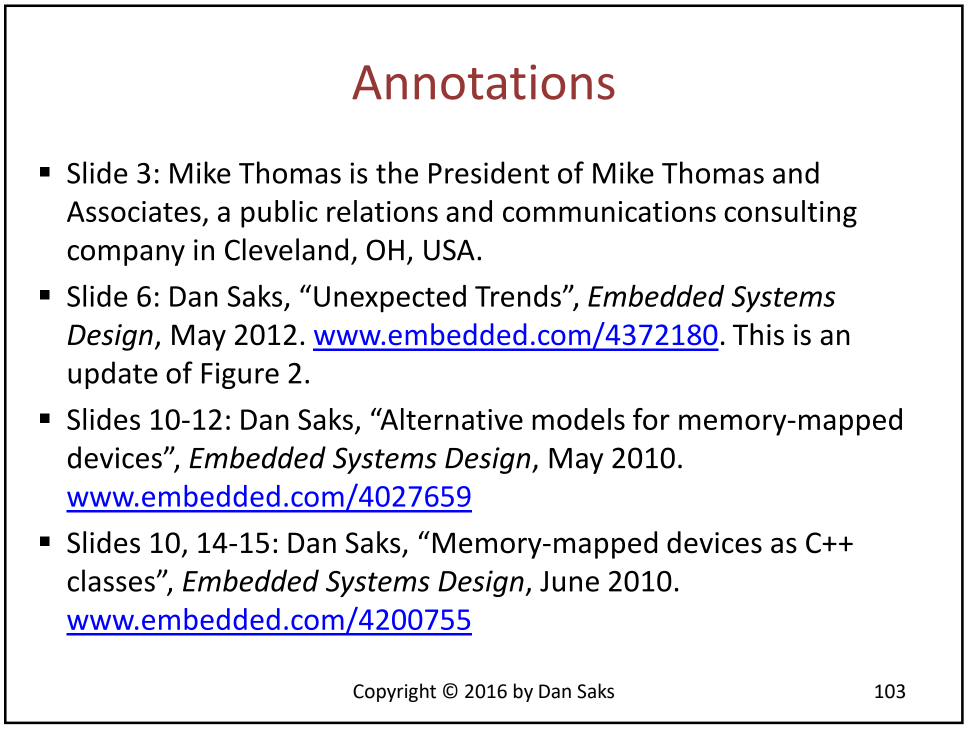
Living by Example

- Where to begin?
  - Look in the mirror.
- Try this line of inquiry the next time someone presents you with something you doubt:
  - How can ***we*** verify this?
  - Is there already credible evidence to back this up?
  - If not, is there a testable hypothesis?
  - If testing isn't feasible, how do we proceed rationally?
- If you want others to act more rationally, set a good example.

Oh, Did I Forget Something?

“If you’re arguing, you’re losing.”

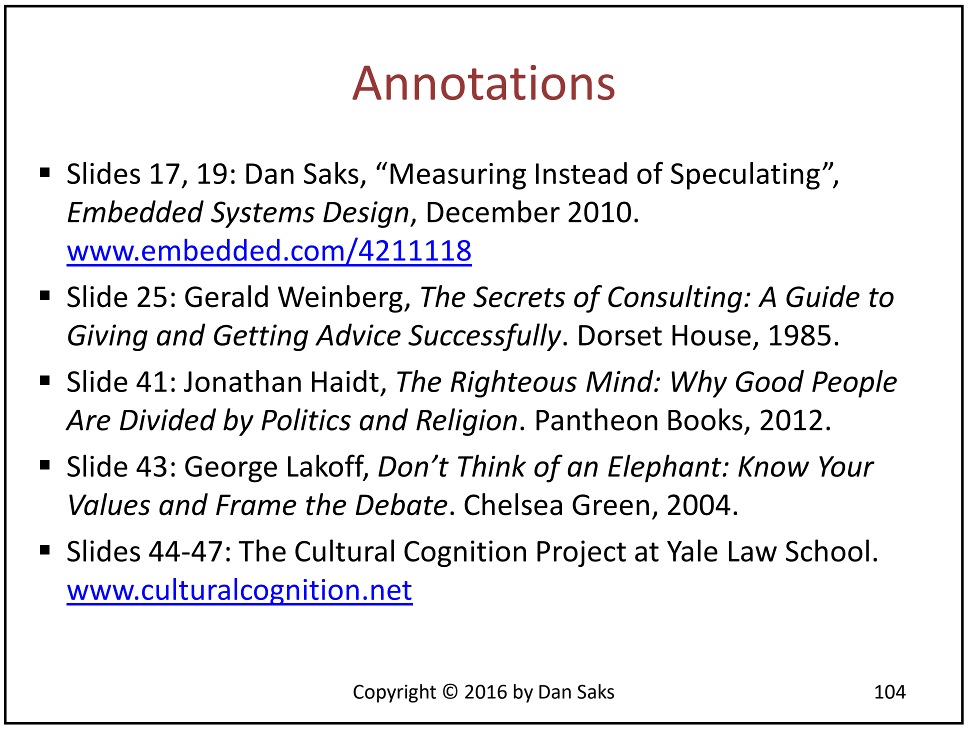
Thanks for sticking around



Annotations

- Slide 3: Mike Thomas is the President of Mike Thomas and Associates, a public relations and communications consulting company in Cleveland, OH, USA.
- Slide 6: Dan Saks, “Unexpected Trends”, *Embedded Systems Design*, May 2012. www.embedded.com/4372180. This is an update of Figure 2.
- Slides 10-12: Dan Saks, “Alternative models for memory-mapped devices”, *Embedded Systems Design*, May 2010. www.embedded.com/4027659
- Slides 10, 14-15: Dan Saks, “Memory-mapped devices as C++ classes”, *Embedded Systems Design*, June 2010. www.embedded.com/4200755

103



Annotations

- Slides 17, 19: Dan Saks, “Measuring Instead of Speculating”, *Embedded Systems Design*, December 2010. www.embedded.com/4211118
- Slide 25: Gerald Weinberg, *The Secrets of Consulting: A Guide to Giving and Getting Advice Successfully*. Dorset House, 1985.
- Slide 41: Jonathan Haidt, *The Righteous Mind: Why Good People Are Divided by Politics and Religion*. Pantheon Books, 2012.
- Slide 43: George Lakoff, *Don’t Think of an Elephant: Know Your Values and Frame the Debate*. Chelsea Green, 2004.
- Slides 44-47: The Cultural Cognition Project at Yale Law School. www.culturalcognition.net

104

Annotations

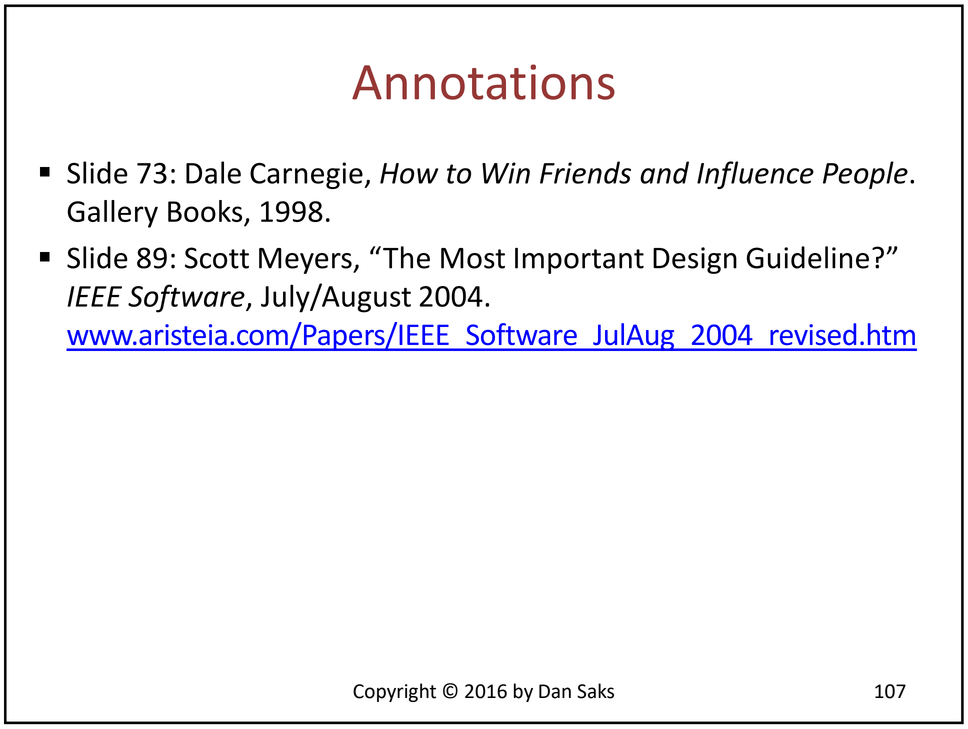
- Slide 45: Dan Kahan, “Cultural vs. ideological cognition, part 1”, *The Cultural Cognition Project* (blog). December 20, 2011. www.culturalcognition.net/blog/2011/12/20/cultural-vs-ideological-cognition-part-1.html
- Slide 46: Nancy Huynh, “Cultural Cognition and Scientific Consensus”, *Yale Scientific Magazine*, May 2011. www.yalescientific.org/2011/05/cultural-cognition-and-scientific-consensus
- Slide 47: Dan Kahan, et al., “Motivated Numeracy and Enlightened Self-Government”. *The Cultural Cognition Project*, September 2013. www.culturalcognition.net/blog/2013/9/4/motivated-numeracy-new-paper.html

105

Annotations

- Slides 49-50: George Lakoff, “Metaphor, Morality, and Politics: Or, Why Conservatives Have Left Liberals In the Dust”. *Social Research*, Summer 1995. web.archive.org/web/20080919182215/http:/www.rockridgeinstitute.org/research/lakoff/New_School.pdf
- Slide 66: Brendan Nyhan and Jason Reifler, “When Corrections Fail: The persistence of political misperceptions”, *Political Behavior*, June 2010. www.dartmouth.edu/~nyhan/nyhan-reifler.pdf
- Slide 69: Daniel Kahneman, *Thinking Fast and Slow*. Farrar, Straus and Giroux, 2011.

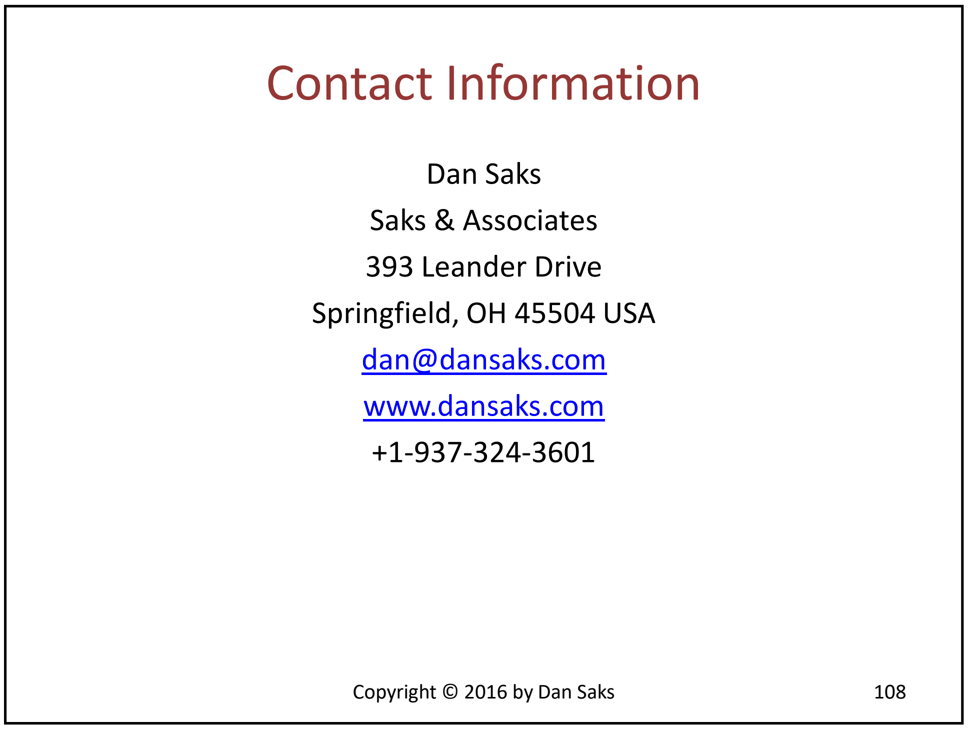
106



Annotations

- Slide 73: Dale Carnegie, *How to Win Friends and Influence People*. Gallery Books, 1998.
- Slide 89: Scott Meyers, "The Most Important Design Guideline?" *IEEE Software*, July/August 2004. www.aristeia.com/Papers/IEEE_Software_JulAug_2004_revised.htm

107



Contact Information

Dan Saks
Saks & Associates
393 Leander Drive
Springfield, OH 45504 USA
dan@dansaks.com
www.dansaks.com
+1-937-324-3601

108