designeast
center of the engineering universe

Top 10 Reasons to Use C++ for Embedded DSP

Matt Liberty
Jetperch LLC

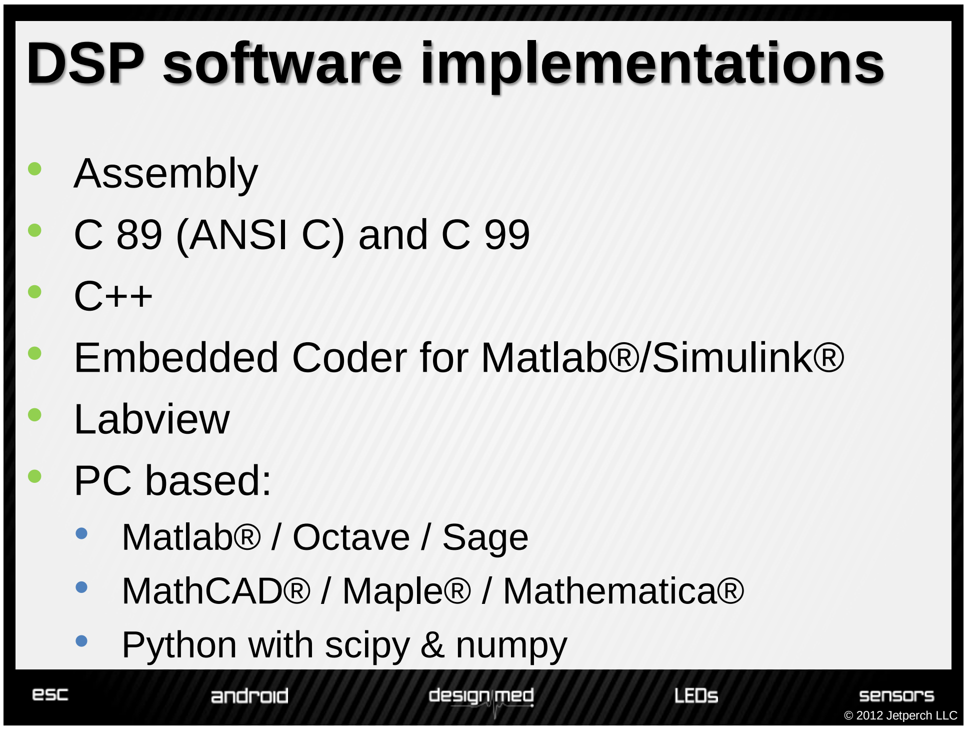
ESC-3015

UBM Electronics

Agenda

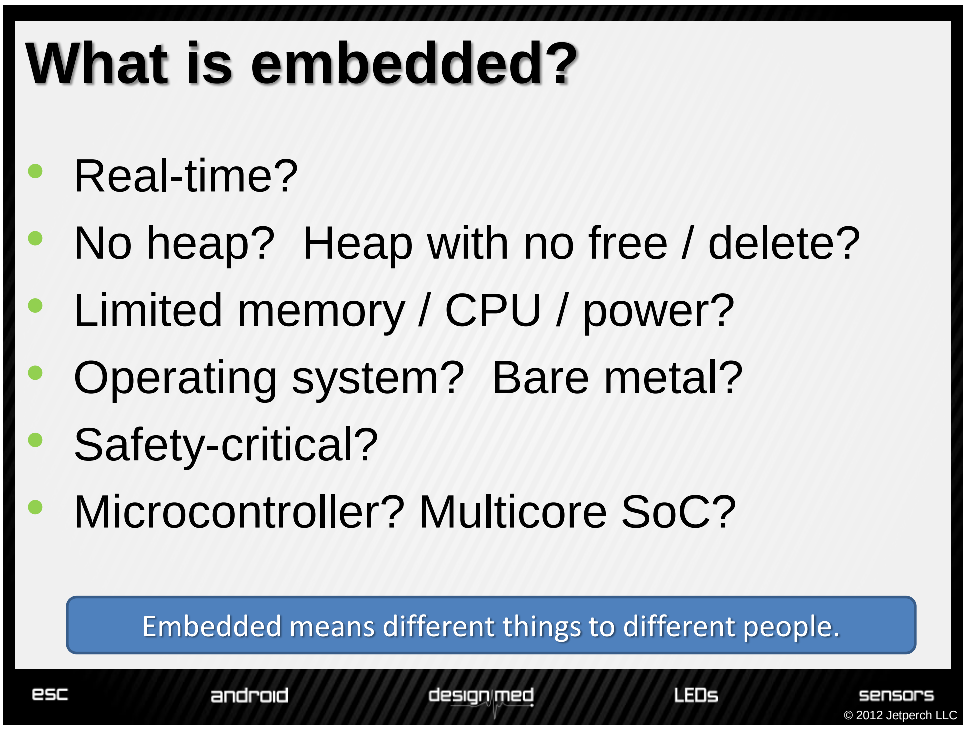
- C++ compare to what?
- Top 10
- C++ concerns
- Conclusion

esc android design med LEDs sensors



DSP software implementations

- Assembly
- C 89 (ANSI C) and C 99
- C++
- Embedded Coder for Matlab®/Simulink®
- Labview
- PC based:
  - Matlab® / Octave / Sage
  - MathCAD® / Maple® / Mathematica®
  - Python with scipy & numpy



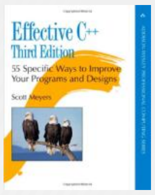
What is embedded?

- Real-time?
- No heap? Heap with no free / delete?
- Limited memory / CPU / power?
- Operating system? Bare metal?
- Safety-critical?
- Microcontroller? Multicore SoC?

Embedded means different things to different people.

Areas of C++

"The easiest way is to view C++ not as a single language but as a federation of related languages."
– Scott Meyers, Effective C++ 3rd Edition.

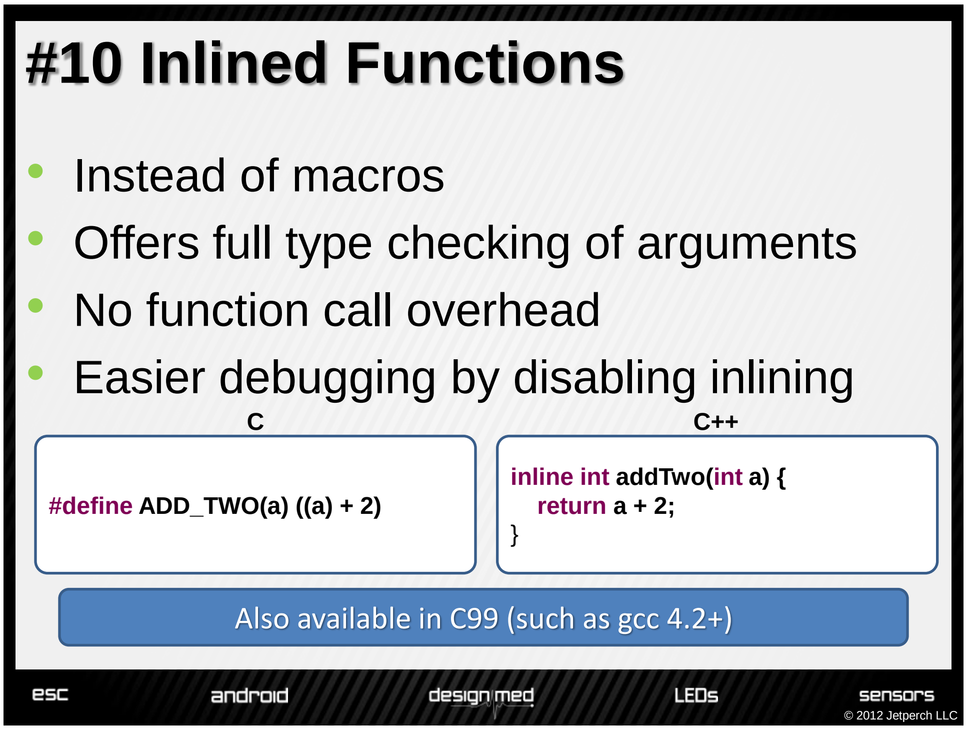


- C
- Object Oriented
- Templates
- Standard Template Library (STL)

C++ Philosophy: You only pay for what you use!

Agenda

- C++ compare to what?
- Top 10
- C++ concerns
- Conclusion



#10 Inlined Functions

- Instead of macros
- Offers full type checking of arguments
- No function call overhead
- Easier debugging by disabling inlining

C

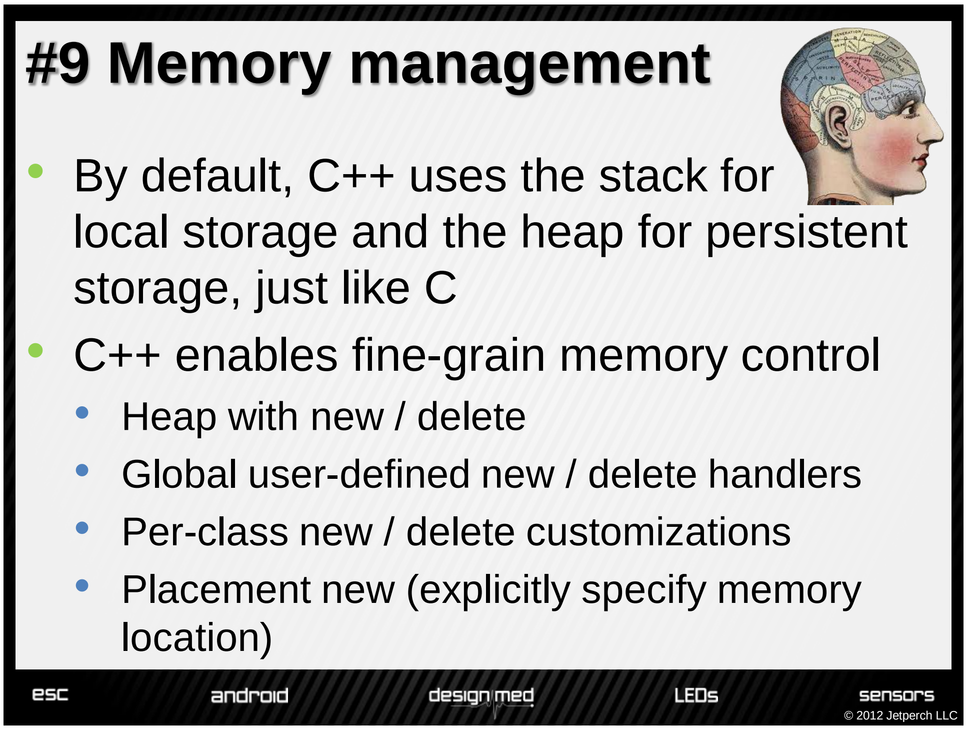
```c
#define ADD_TWO(a) ((a) + 2)
```

C++

```cpp
inline int addTwo(int a) {
    return a + 2;
}
```

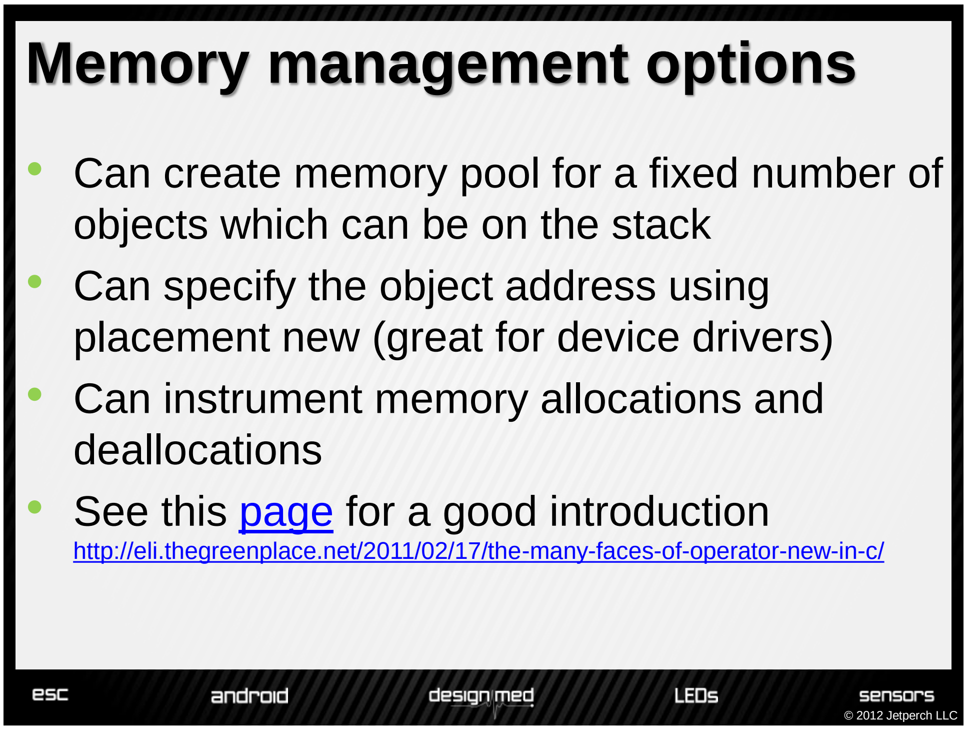
Also available in C99 (such as gcc 4.2+)

9



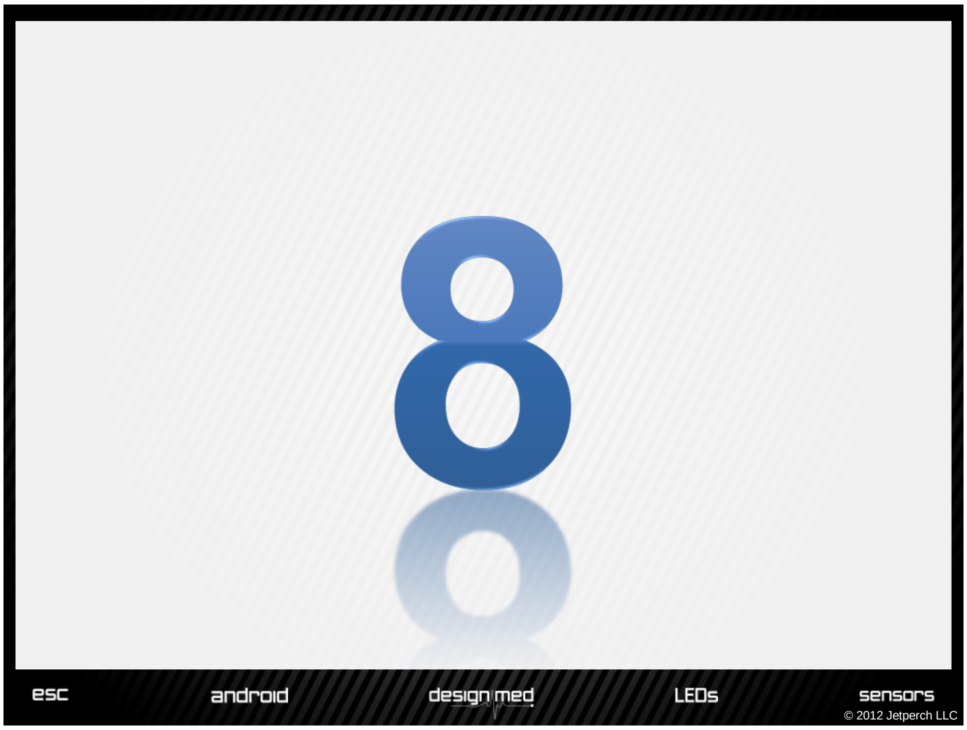
#9 Memory management

- By default, C++ uses the stack for local storage and the heap for persistent storage, just like C
- C++ enables fine-grain memory control
  - Heap with new / delete
  - Global user-defined new / delete handlers
  - Per-class new / delete customizations
  - Placement new (explicitly specify memory location)

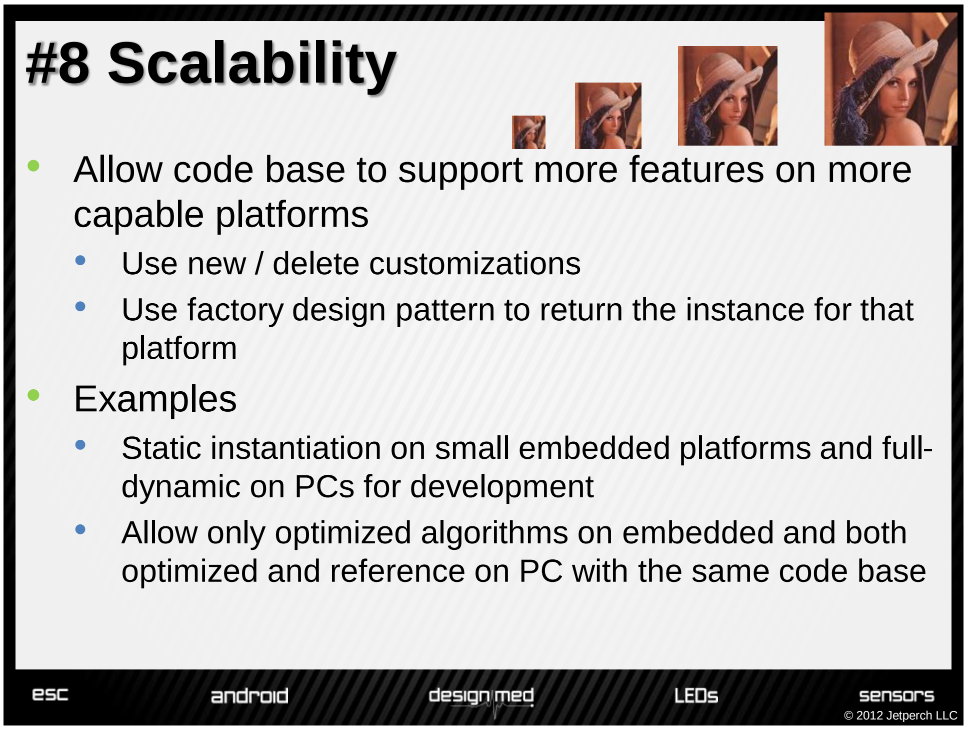


Memory management options

- Can create memory pool for a fixed number of objects which can be on the stack
- Can specify the object address using placement new (great for device drivers)
- Can instrument memory allocations and deallocations
- See this page for a good introduction
  http://eli.thegreenplace.net/2011/02/17/the-many-faces-of-operator-new-in-c/



8



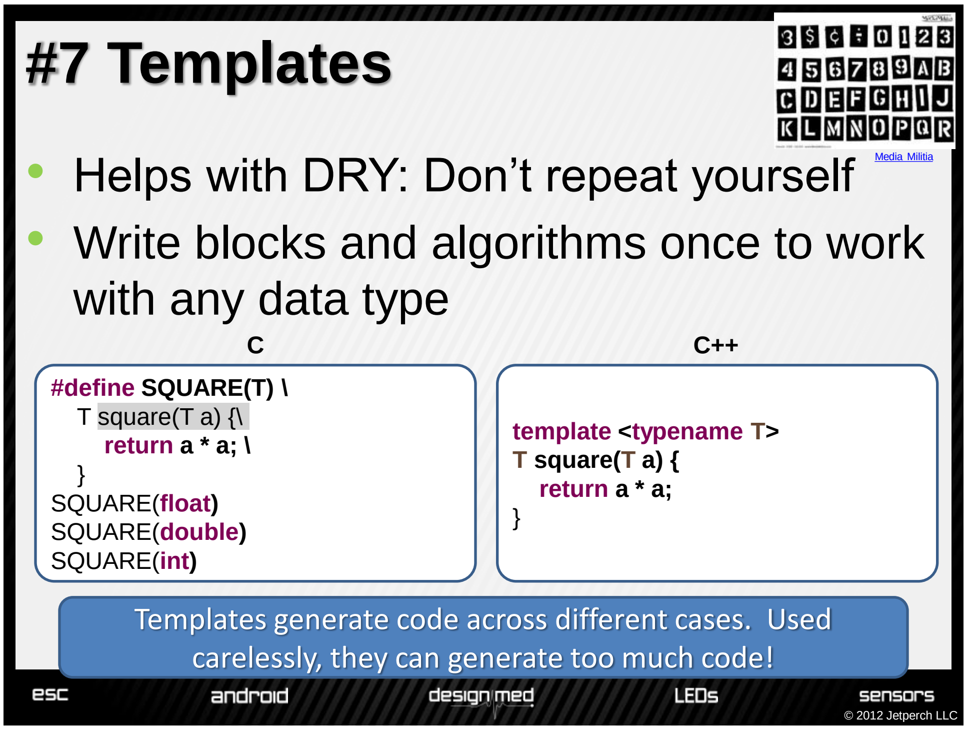
#8 Scalability

- Allow code base to support more features on more capable platforms
  - Use new / delete customizations
  - Use factory design pattern to return the instance for that platform
- Examples
  - Static instantiation on small embedded platforms and full-dynamic on PCs for development
  - Allow only optimized algorithms on embedded and both optimized and reference on PC with the same code base

esc android design med LEDs sensors

7

esc android design med LEDs sensors



#7 Templates

- Helps with DRY: Don't repeat yourself
- Write blocks and algorithms once to work with any data type

C

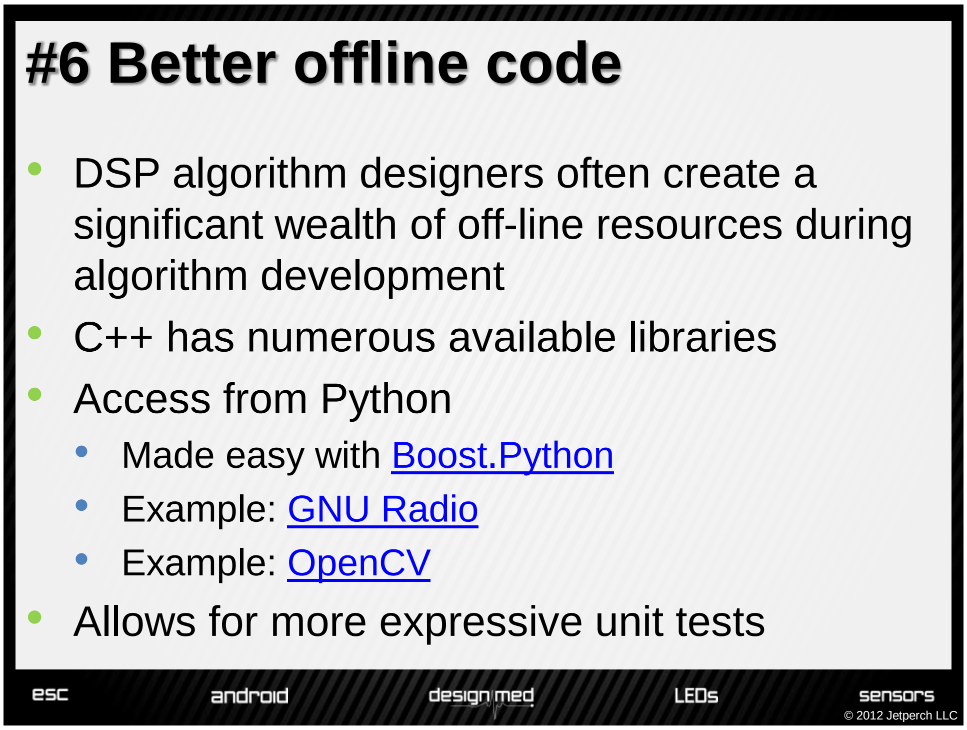
```
#define SQUARE(T) \
    T square(T a) {\
        return a * a; \
    }
SQUARE(float)
SQUARE(double)
SQUARE(int)
```

C++

```
template <typename T>
T square(T a) {
    return a * a;
}
```

Templates generate code across different cases. Used carelessly, they can generate too much code!

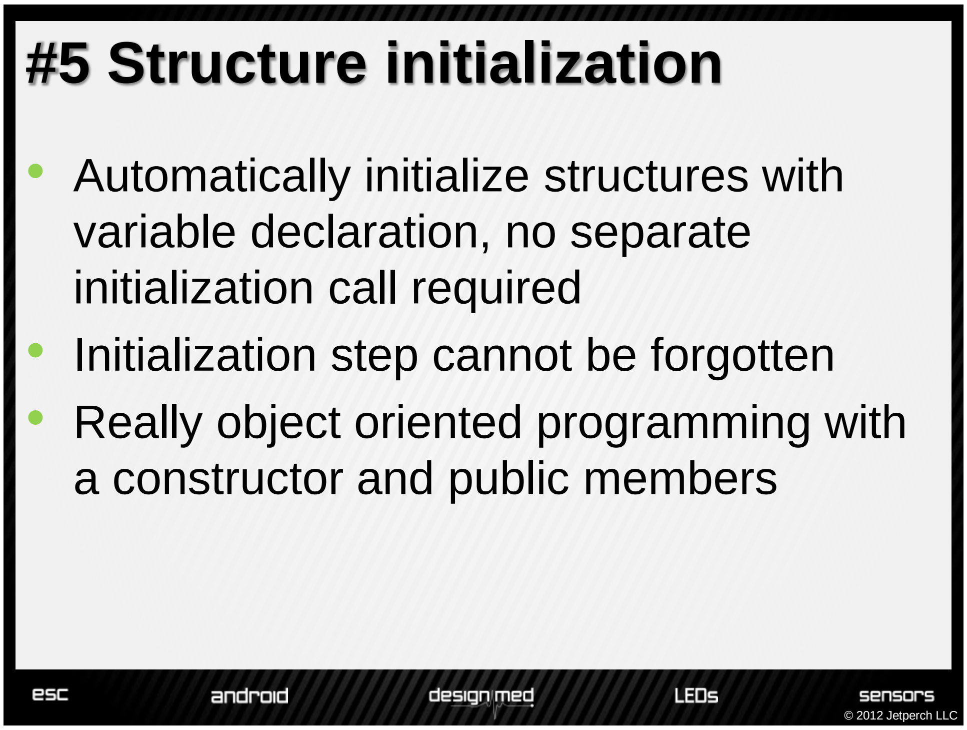
6



#6 Better offline code

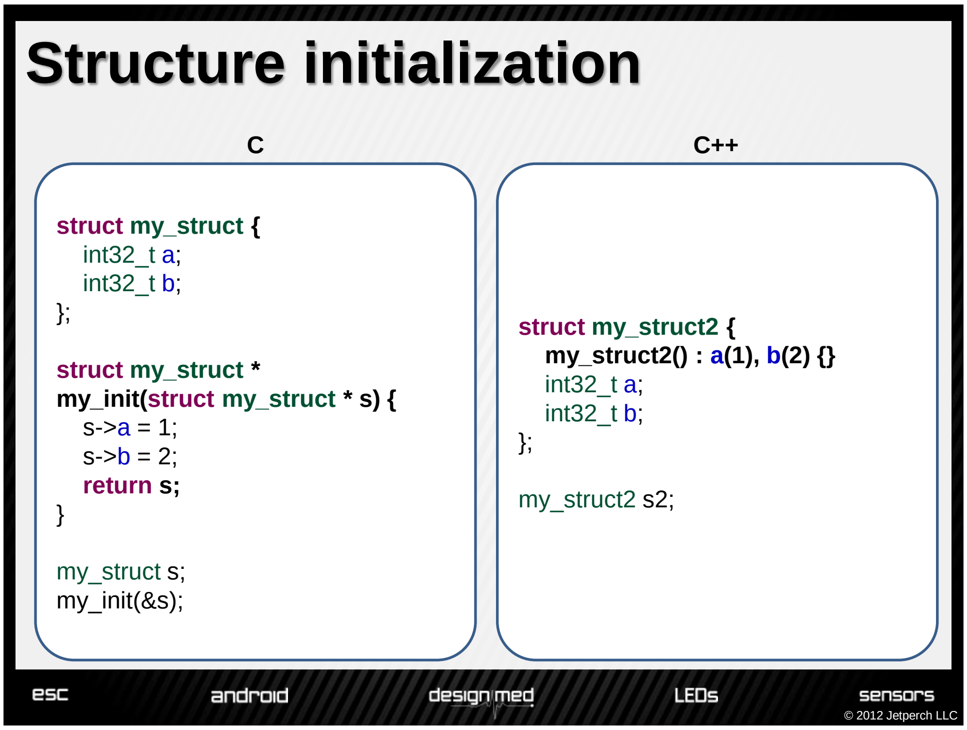
- DSP algorithm designers often create a significant wealth of off-line resources during algorithm development
- C++ has numerous available libraries
- Access from Python
  - Made easy with Boost.Python
  - Example: GNU Radio
  - Example: OpenCV
- Allows for more expressive unit tests

5



#5 Structure initialization

- Automatically initialize structures with variable declaration, no separate initialization call required
- Initialization step cannot be forgotten
- Really object oriented programming with a constructor and public members



Structure initialization

C

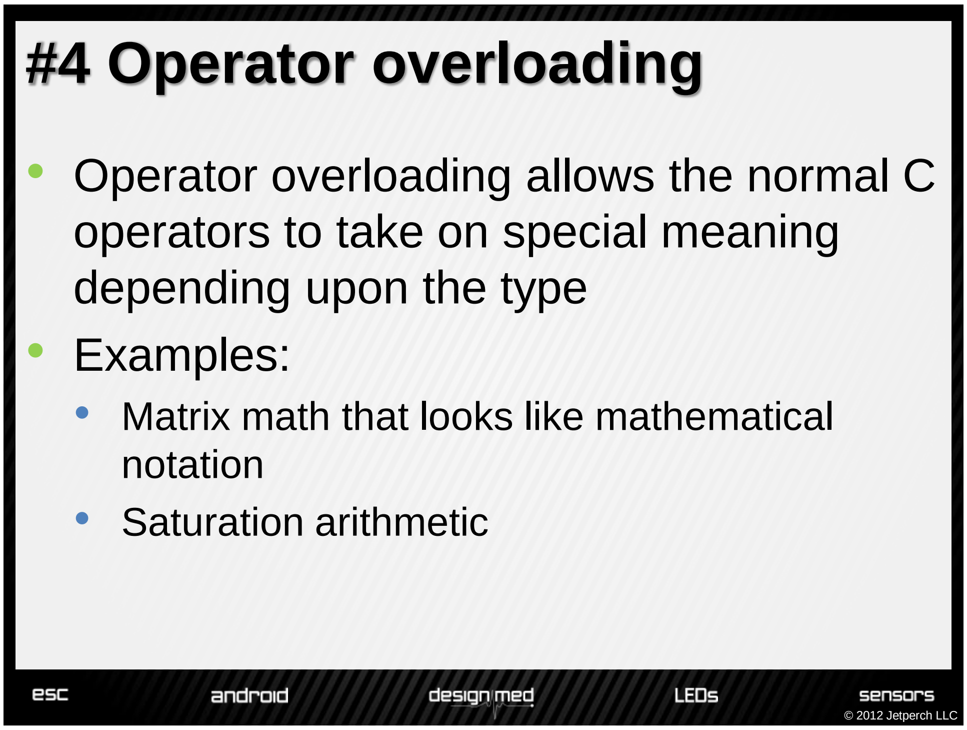
```
struct my_struct {
    int32_t a;
    int32_t b;
};

struct my_struct *
my_init(struct my_struct * s) {
    s->a = 1;
    s->b = 2;
    return s;
}

my_struct s;
my_init(&s);
```

C++

```
struct my_struct2 {
    my_struct2() : a(1), b(2) {}
    int32_t a;
    int32_t b;
};

my_struct2 s2;
```

4

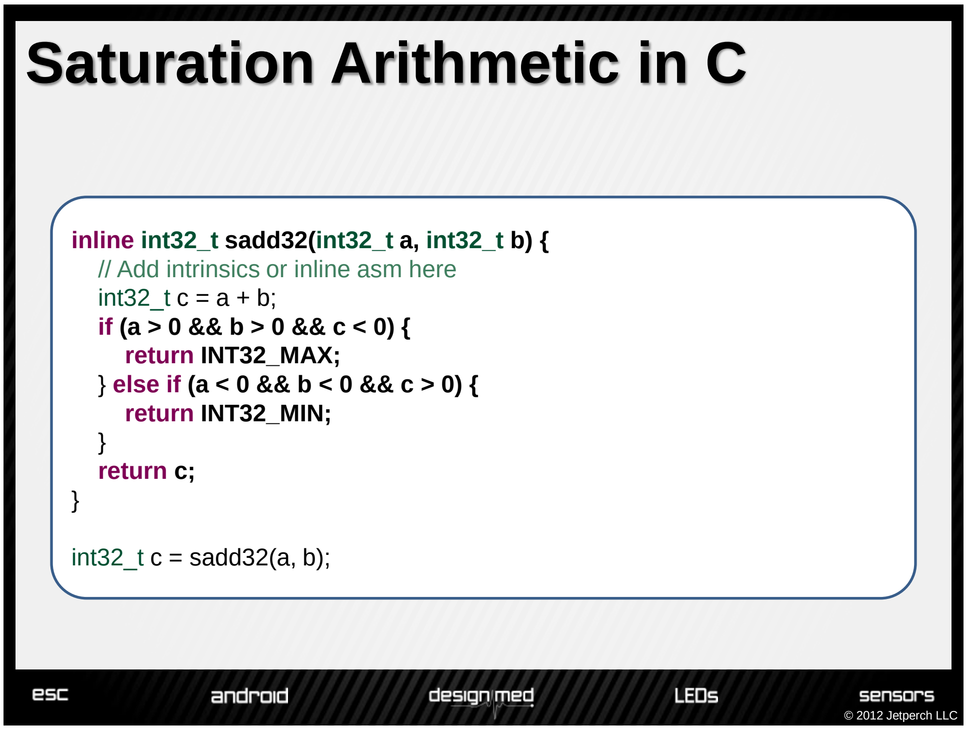
esc
android
design med
LEDs
sensors



#4 Operator overloading

- Operator overloading allows the normal C operators to take on special meaning depending upon the type
- Examples:
  - Matrix math that looks like mathematical notation
  - Saturation arithmetic

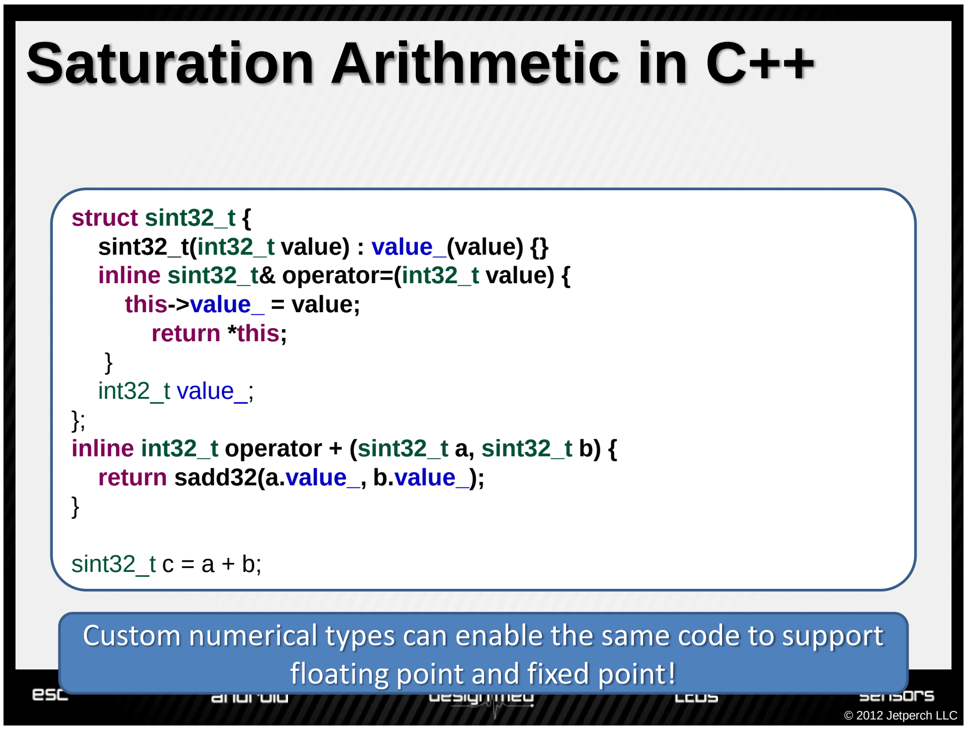
esc
android
design med
LEDs
sensors



Saturation Arithmetic in C

```
inline int32_t sadd32(int32_t a, int32_t b) {
   // Add intrinsics or inline asm here
   int32_t c = a + b;
   if (a > 0 && b > 0 && c < 0) {
      return INT32_MAX;
   } else if (a < 0 && b < 0 && c > 0) {
      return INT32_MIN;
   }
   return c;
}

int32_t c = sadd32(a, b);
```

esc android design med LEDs sensors



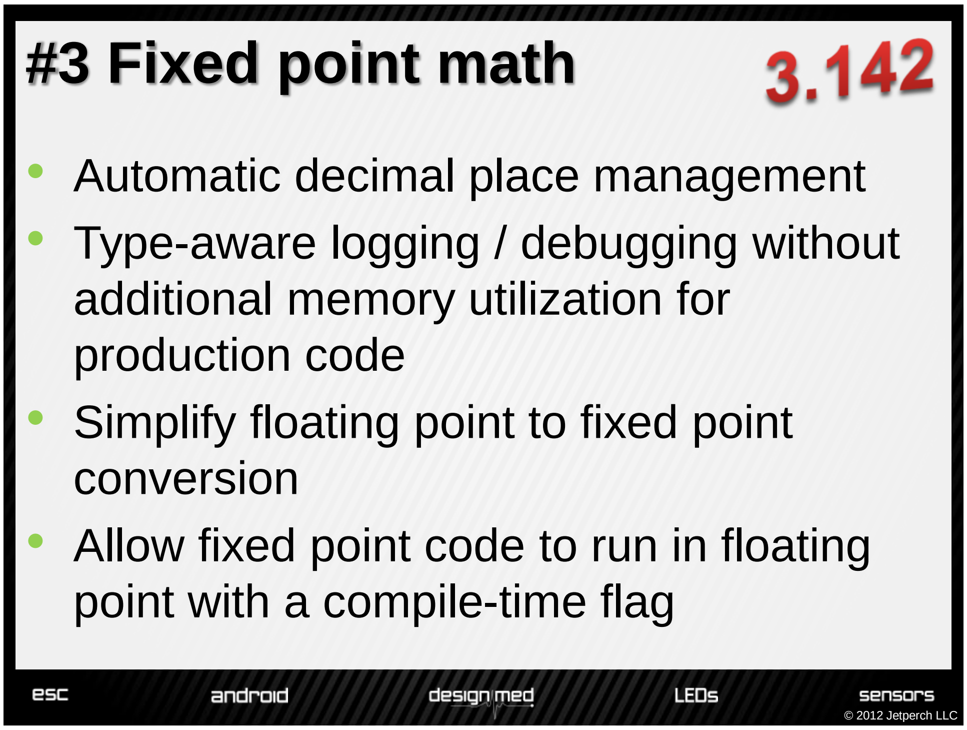
Saturation Arithmetic in C++

```
struct sint32_t {
   sint32_t(int32_t value) : value_(value) {}
   inline sint32_t& operator=(int32_t value) {
      this->value_ = value;
         return *this;
    }
   int32_t value_;
};
inline int32_t operator + (sint32_t a, sint32_t b) {
   return sadd32(a.value_, b.value_);
}

sint32_t c = a + b;
```

Custom numerical types can enable the same code to support floating point and fixed point!

esc android design med LEDs sensors

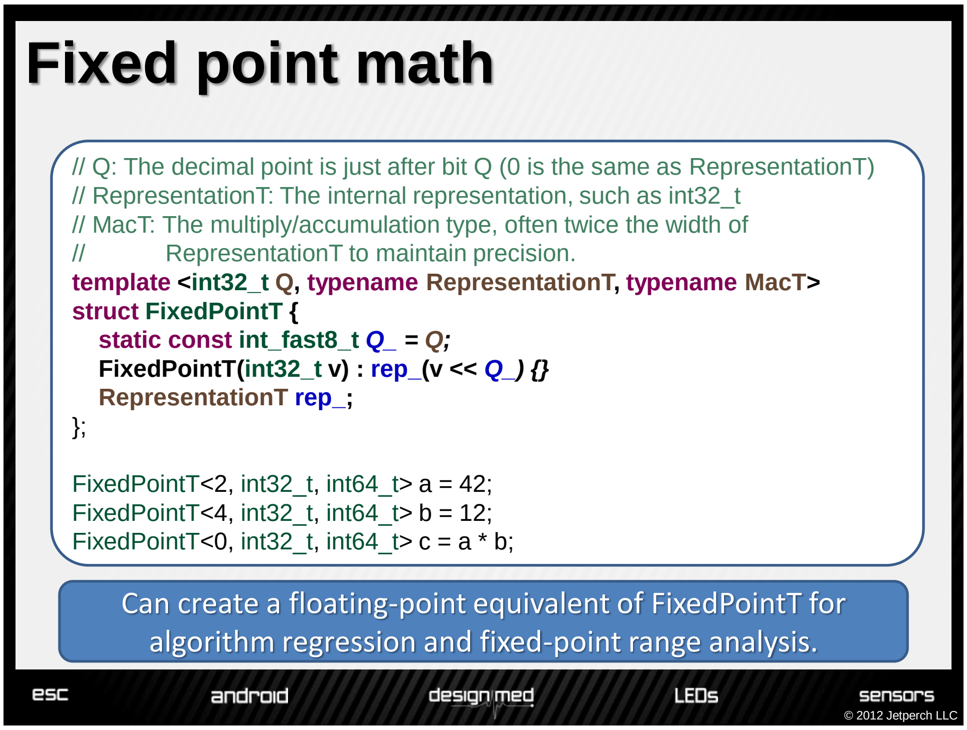
3



#3 Fixed point math

3.142

- Automatic decimal place management
- Type-aware logging / debugging without additional memory utilization for production code
- Simplify floating point to fixed point conversion
- Allow fixed point code to run in floating point with a compile-time flag

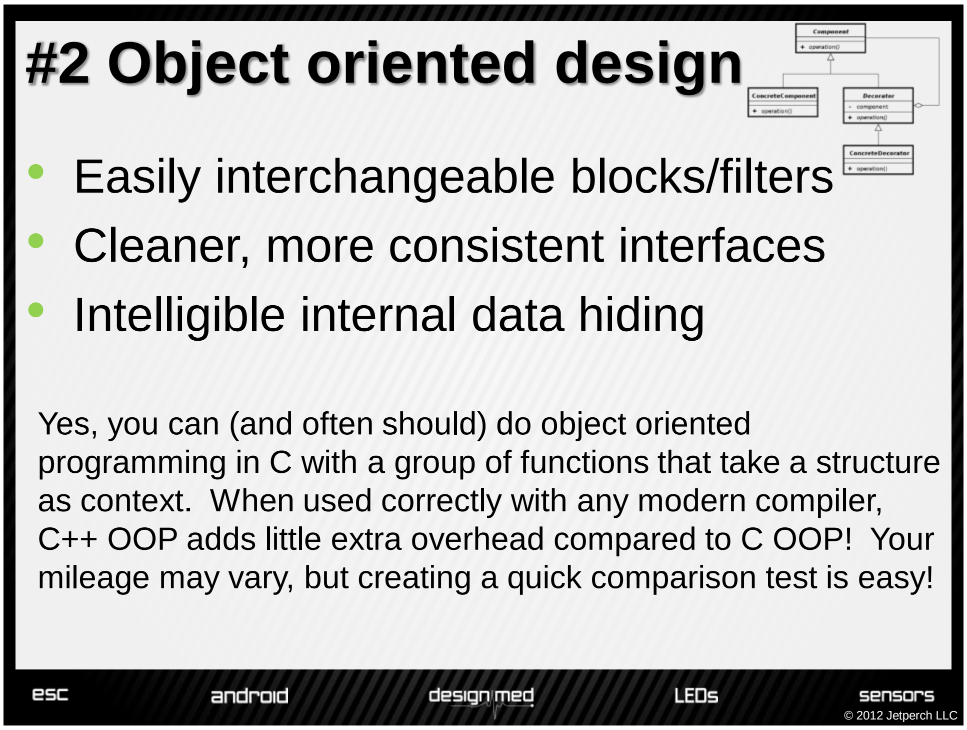


Fixed point math

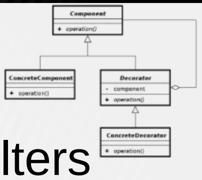
```
// Q: The decimal point is just after bit Q (0 is the same as RepresentationT)
// RepresentationT: The internal representation, such as int32_t
// MacT: The multiply/accumulation type, often twice the width of
//           RepresentationT to maintain precision.
template <int32_t Q, typename RepresentationT, typename MacT>
struct FixedPointT {
   static const int_fast8_t Q_ = Q;
   FixedPointT(int32_t v) : rep_(v << Q_) {}
   RepresentationT rep_;
};

FixedPointT<2, int32_t, int64_t> a = 42;
FixedPointT<4, int32_t, int64_t> b = 12;
FixedPointT<0, int32_t, int64_t> c = a * b;
```

Can create a floating-point equivalent of FixedPointT for algorithm regression and fixed-point range analysis.

2

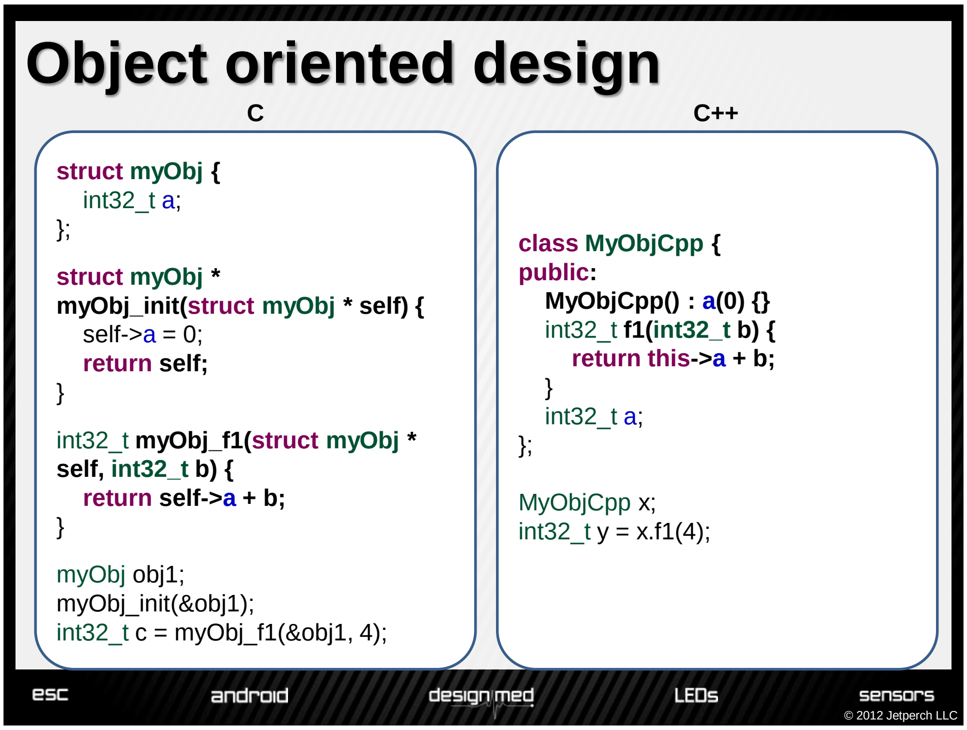


#2 Object oriented design



- Easily interchangeable blocks/filters
- Cleaner, more consistent interfaces
- Intelligible internal data hiding

Yes, you can (and often should) do object oriented programming in C with a group of functions that take a structure as context. When used correctly with any modern compiler, C++ OOP adds little extra overhead compared to C OOP! Your mileage may vary, but creating a quick comparison test is easy!



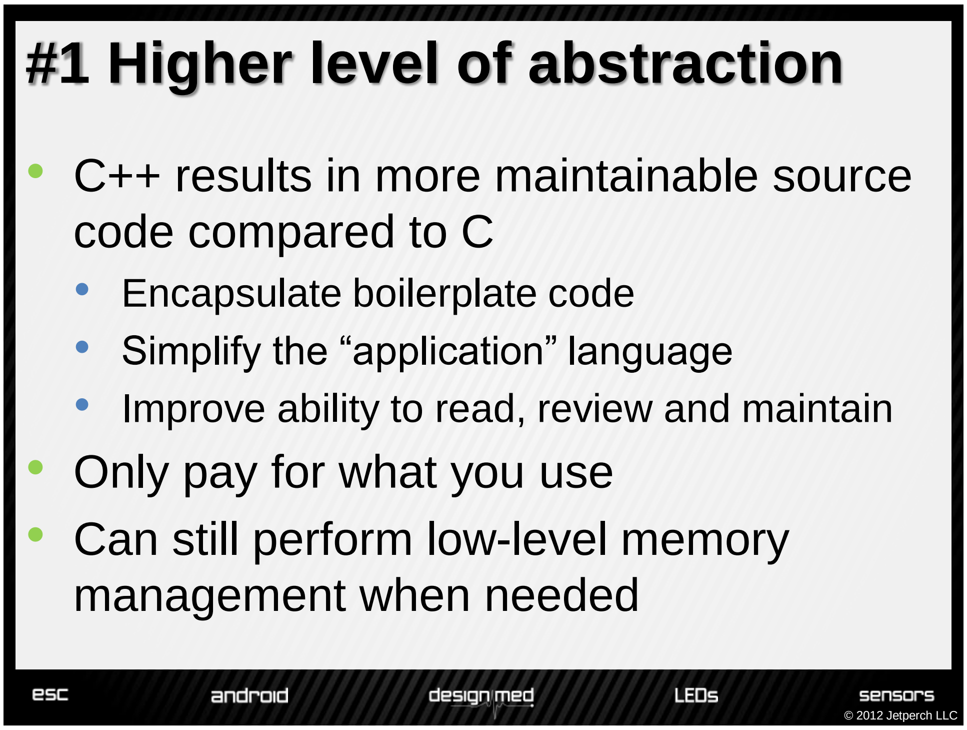
Object oriented design

C

```
struct myObj {
    int32_t a;
};

struct myObj *
myObj_init(struct myObj * self) {
    self->a = 0;
    return self;
}

int32_t myObj_f1(struct myObj *
self, int32_t b) {
    return self->a + b;
}

myObj obj1;
myObj_init(&obj1);
int32_t c = myObj_f1(&obj1, 4);
```

C++

```
class MyObjCpp {
public:
    MyObjCpp() : a(0) {}
    int32_t f1(int32_t b) {
        return this->a + b;
    }
    int32_t a;
};

MyObjCpp x;
int32_t y = x.f1(4);
```



#1 Higher level of abstraction

- C++ results in more maintainable source code compared to C
  - Encapsulate boilerplate code
  - Simplify the "application" language
  - Improve ability to read, review and maintain
- Only pay for what you use
- Can still perform low-level memory management when needed

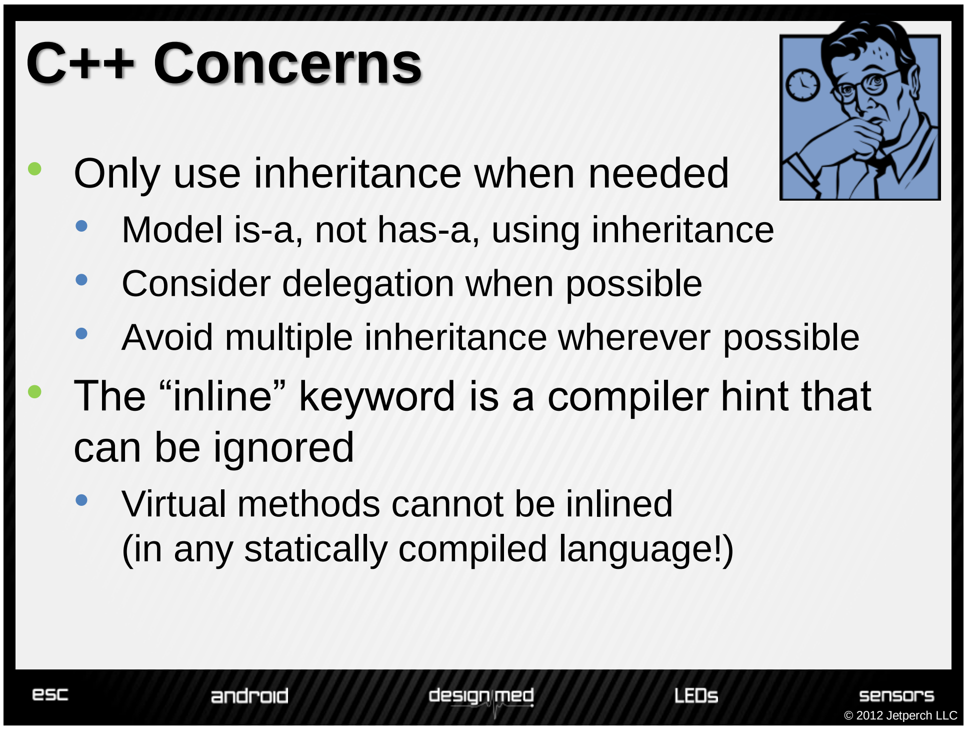
esc
android
design med
LEDs
sensors
© 2012 Jetperch LLC

Agenda

- C++ compare to what?
- Top 10
- C++ concerns
- Conclusion

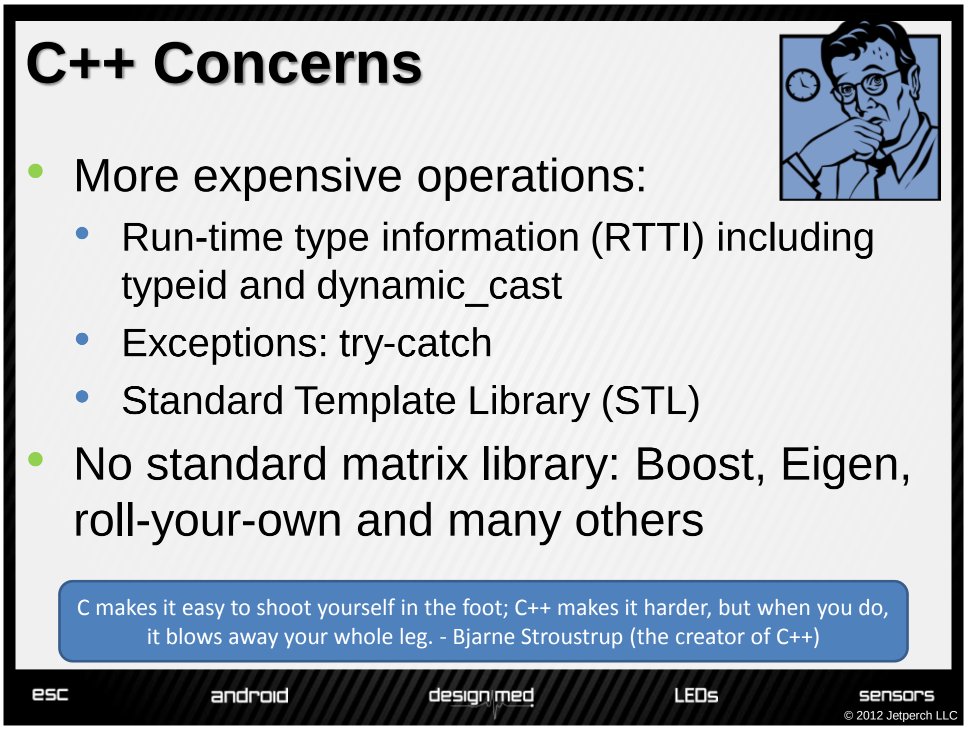
C++ Concerns

- Compiler support & optimizations vary
  - Your compiler may not support all of C++
  - EC++ (mostly dead?) drops templates, exceptions and STL
- Uninformed use of C++ features can cause code bloat
  - Functionality adds resource costs regardless of the implementation language!
  - Only use the features that you need
  - When resources are critical, profile the language features of interest!



C++ Concerns

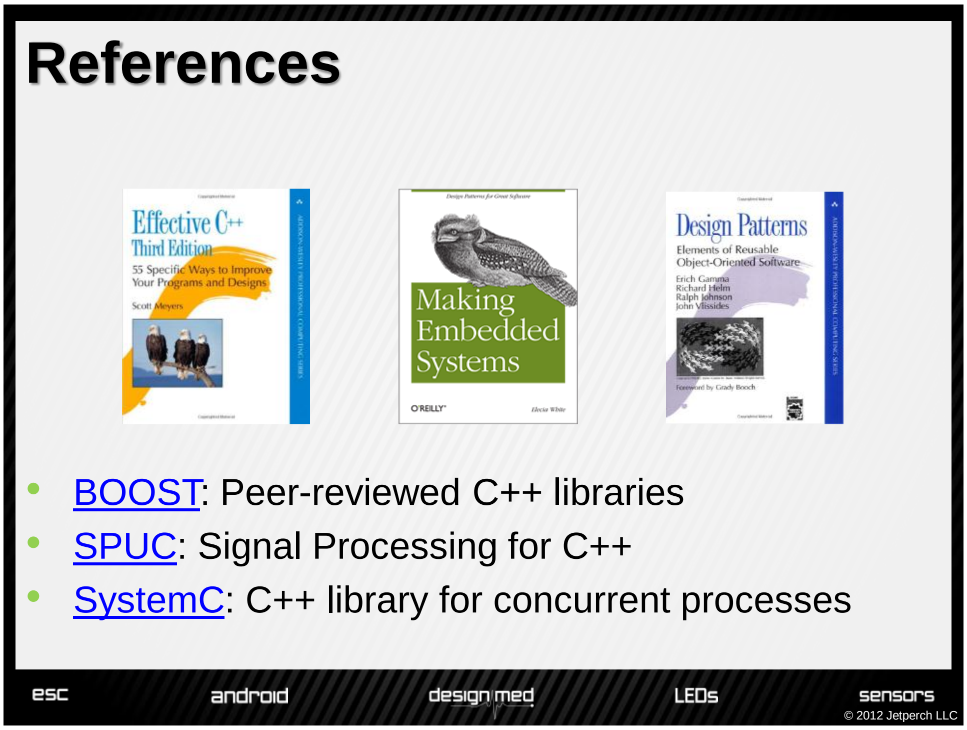
- Only use inheritance when needed
  - Model is-a, not has-a, using inheritance
  - Consider delegation when possible
  - Avoid multiple inheritance wherever possible
- The “inline” keyword is a compiler hint that can be ignored
  - Virtual methods cannot be inlined
    (in any statically compiled language!)



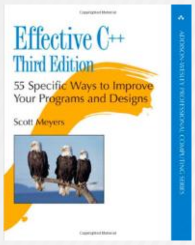
C++ Concerns

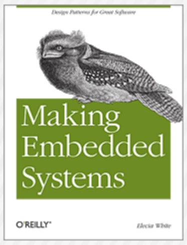
- More expensive operations:
  - Run-time type information (RTTI) including typeid and dynamic_cast
  - Exceptions: try-catch
  - Standard Template Library (STL)
- No standard matrix library: Boost, Eigen, roll-your-own and many others

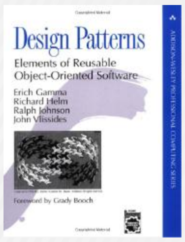
C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg. - Bjarne Stroustrup (the creator of C++)



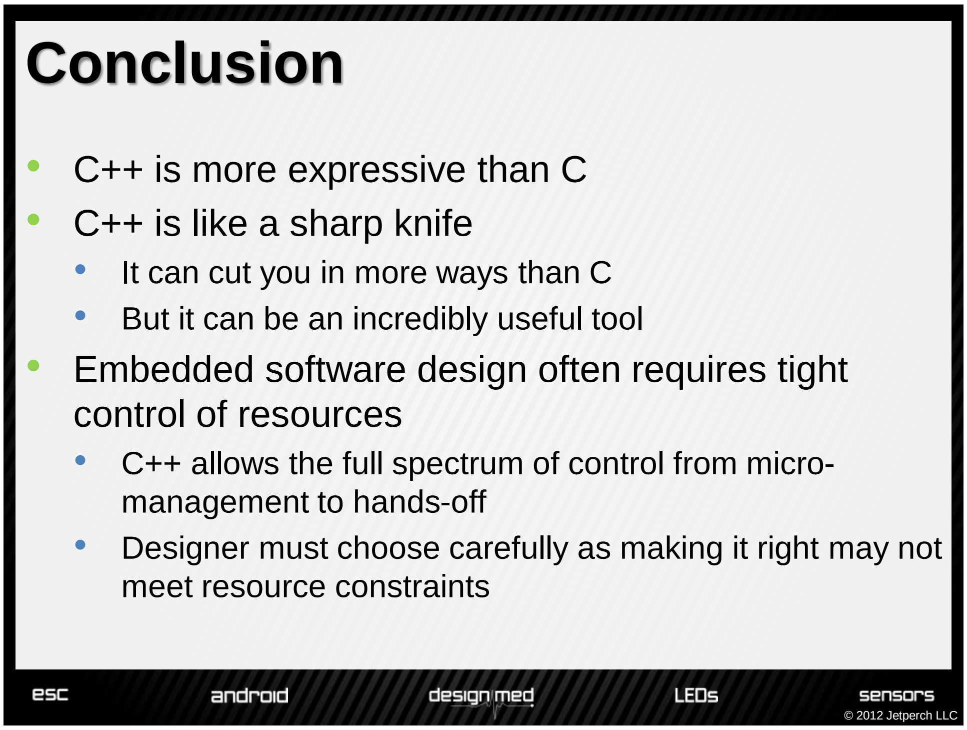
References


Effective C++ Third Edition
55 Specific Ways to Improve Your Programs and Designs
Scott Meyers





- BOOST: Peer-reviewed C++ libraries
- SPUC: Signal Processing for C++
- SystemC: C++ library for concurrent processes



Conclusion

- C++ is more expressive than C
- C++ is like a sharp knife
  - It can cut you in more ways than C
  - But it can be an incredibly useful tool
- Embedded software design often requires tight control of resources
  - C++ allows the full spectrum of control from micro-management to hands-off
  - Designer must choose carefully as making it right may not meet resource constraints

designeast
center of the engineering universe
UBM
Electronics

Top 10 Reasons to Use C++ for Embedded DSP

Matt Liberty
matt@jetperch.com
Jetperch LLC