B|CUBED
Engineering

Improved Messaging using Modern PMTs

Innovate

Create and innovate without all the red tape – ideas matter

Collaborate

Work with our team and customers to drive impactful solutions

Hybrid

Work remotely and from the office

PSP

+25% of base salary automatically deposited to retirement account

IBA

+25% of base salary towards benefits and medical expenses

B|CUBED

Learn more at bcubed-corp.com

SWAMPDONKEY

ROGUE

MCMT

B|CUBED
Engineering

Outline

- What is a PMT?
- Why are we doing something new?
- What are we doing?
- What have we accomplished over the past year?
- Where are we going?

B|CUBED Engineering

What is a PMT?

- Polymorphic Type
- Groups arbitrary data types together.
  - Think of a python dictionary or json.
- Includes serialization/deserialization functions.
  - Allows us to send data over the network (distributed processing).
- Used in GNURadio for async messages and data tags.

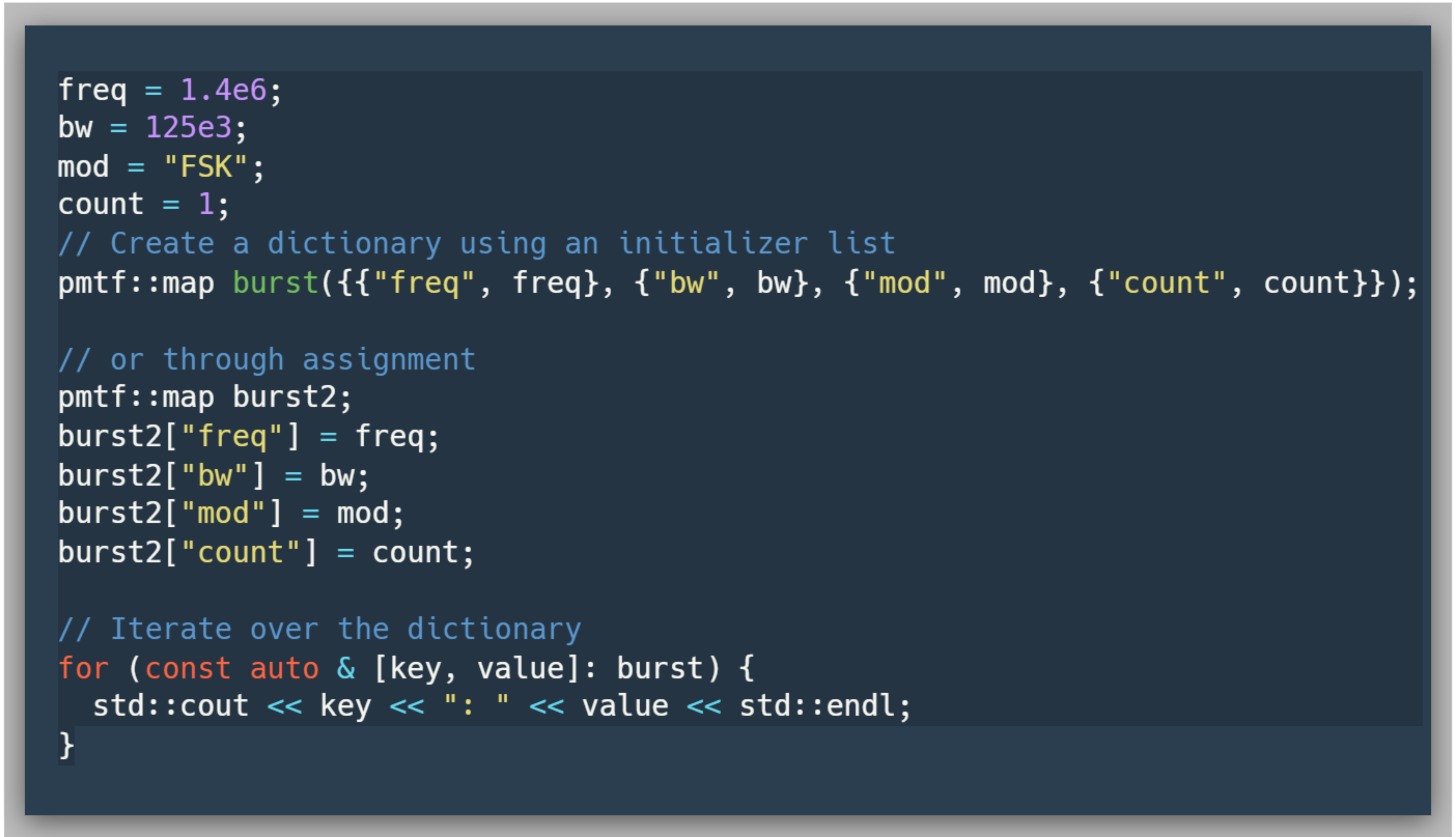
B|CUBED Engineering

Why do we want new PMTs?

- Inconsistent and hard to remember function names
  - E.g. pmt::from_long() and pmt::is_integer()
- Message Validation is really hard
  - E.g. Is this message a dictionary with certain keys and value types.
- Slow performance
- Difficult to do memory safe operations
  - Leads to frequent segfaults.

B|CUBED Engineering

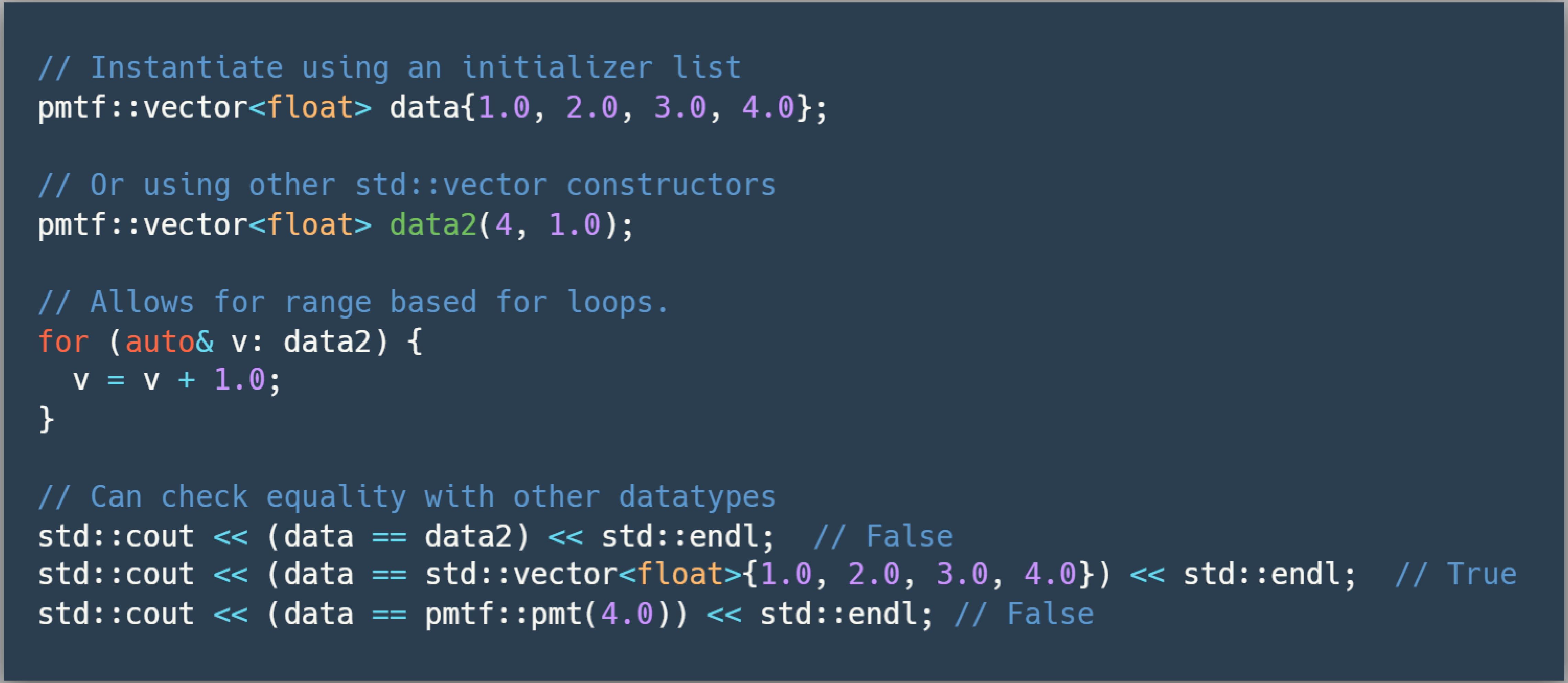
PMTs Maps with Modern C++



```
freq = 1.4e6;
bw = 125e3;
mod = "FSK";
count = 1;
// Create a dictionary using an initializer list
pmtf::map burst({{"freq", freq}, {"bw", bw}, {"mod", mod}, {"count", count}});

// or through assignment
pmtf::map burst2;
burst2["freq"] = freq;
burst2["bw"] = bw;
burst2["mod"] = mod;
burst2["count"] = count;

// Iterate over the dictionary
for (const auto & [key, value]: burst) {
  std::cout << key << ": " << value << std::endl;
}
```

B|CUBED Engineering

PMT Vectors with Modern C++

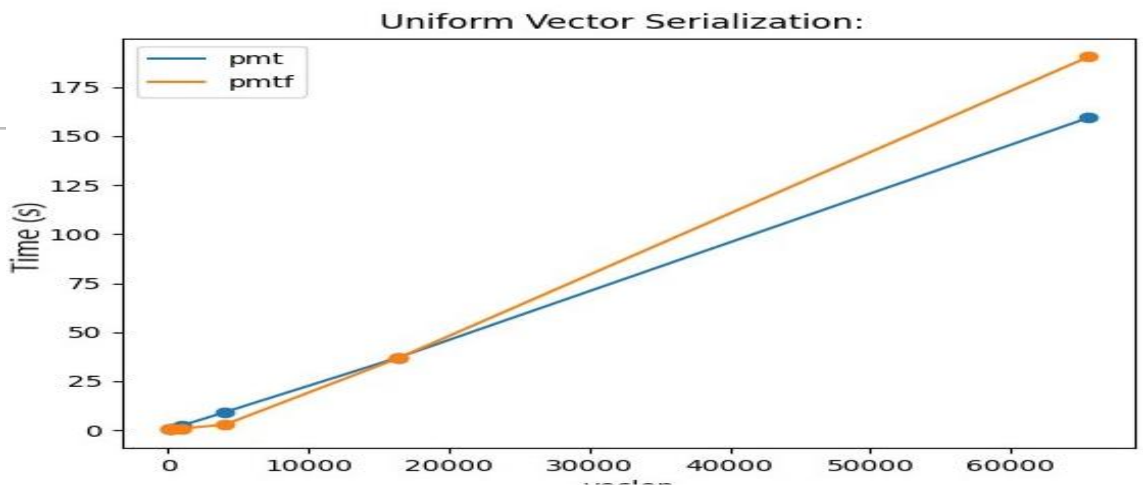


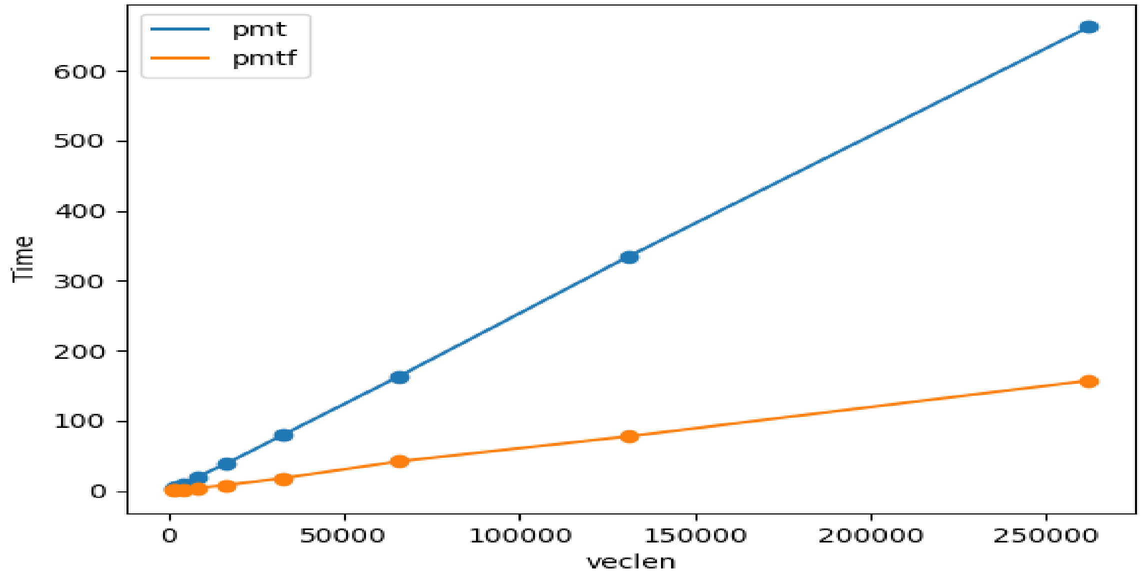
```cpp
// Instantiate using an initializer list
pmtf::vector<float> data{1.0, 2.0, 3.0, 4.0};

// Or using other std::vector constructors
pmtf::vector<float> data2(4, 1.0);

// Allows for range based for loops.
for (auto& v: data2) {
  v = v + 1.0;
}

// Can check equality with other datatypes
std::cout << (data == data2) << std::endl;  // False
std::cout << (data == std::vector<float>{1.0, 2.0, 3.0, 4.0}) << std::endl;  // True
std::cout << (data == pmtf::pmt(4.0)) << std::endl; // False
```

B|CUBED
Engineering

Recent Progress

- Fixed a major performance Issue.
  - Vector serialization is one of the most common pmt operations.
  - In many cases, it was much slower than the original pmt implementation.
  - Required a major rewrite of the whole library.
- Integrated into GNURadio 4.0





B|CUBED Engineering

What's Next?

- Add data validation functionality.
  - Similar to JSON Schemas
- Benchmark all major functions.
  - Compare to present pmt equivalent.
  - This is how we found the vector serialization issue.
  - Already found a few areas for improvement.
- Ensure 100% code coverage for unit tests.
- Build on different OS's, architectures, and with different compilers.
- Support Gnuradio 4.0 development.

Questions??

- Repo Available at https://github.com/gnuradio/pmt
- Contact me: jsallay@bcubed-corp.com

My coworkers Joe Zeigler and Josh Williams are also attending;
Give any of us a shout to learn more about fun work at BCubed!